UNIVERSIDAD
DE MÁLAGA

Nuevas tendencias en fundamentos teóricos y aplicaciones de la minería de datos aplicada a la clasificación de textos en lenguaje natural

José María Carmona Cejudo

Directores:
Rafael Morales Bueno
Manuel Baena García

Para optar al grado de:
Doctor Ingeniero en Informática

Departamento de Lenguajes y Ciencias de La Computación
Escuela Técnica Superior de Ingenieria Informática
Universidad de Málaga

Junio 2013

El Dr. D. Rafael Morales Bueno, Catedrático del Área de Lenguajes y Sistemas Informáticos de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Málaga, y el Dr. D. Manuel Baena García, Doctor en Informática, certifican que,

D. José María Carmona Cejudo, Ingeniero en Informática, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, bajo su dirección, el trabajo de investigación correspondiente a su tesis doctoral titulada:

Nuevas tendencias en fundamentos teóricos y aplicaciones de la minería de datos aplicada a la clasificación de textos en lenguaje natural

Revisado el presente trabajo, estiman que puede ser presentado al tribunal que ha de juzgarlo, y autoriza la presentación de esta Tesis Doctoral en la Universidad de Málaga.

Fdo.: Rafael Morales Bueno Manuel Baena García

Málaga

"We have an opportunity for everyone in the world to have access to all the world's information. This has never before been possible. Why is ubiquitous information so profound? It's a tremendous equalizer. Information is power."

Eric Schmidt, Univ. of Pennsylvania Commencement Address, 2009.

"There is, perhaps, one universal truth about all forms of human cognition: the ability to deal with knowledge is hugely exceeded by the potential knowledge contained in man's environment. To cope with this diversity, man's perception, his memory, and his thought processes early become governed by strategies for protecting his limited capacities from the confusion of overloading. We tend to perceive things schematically, for example, rather than in detail, or we represent a class of diverse things by some sort of averaged typical instance"

Jerome S. Bruner

Agradecimientos

Quisiera agradecer en esta página a todas las personas que han hecho posible que este trabajo de tesis doctoral se haga realidad.

En primer lugar, a mis directores de tesis, con los que he compartido mi trayectoria en la Universidad desde el Proyecto Fin de Carrera, pasando por el Máster ISIA, la colaboración con el Hospital Costa del Sol y, finalmente, el doctorado. Gracias a Rafael, que me ha guiado siempre al objetivo, ayudándome en todo lo posible, y mostrándome toda su comprensión y apoyo, también en situaciones personales. Y por supuesto a Manuel, del que he aprendido tantas cosas que no sabría por dónde empezar. Gran profesional y gran persona, sin él esta tesis no hubiera sido posible.

Gracias también a todos los demás investigadores con los que he colaborado durante la realización de esta tesis. Por supuesto a Gladys Castillo, que siempre me ha transmitido su pasión por la investigación y por el trabajo bien hecho; considero que una de los mejores frutos de este trabajo ha sido poder colaborar con ella. Y también me gustaría agradecer a José del Campo, trabajar y compartir todo este tiempo con él ha sido un placer. Y a João Gama, Albert Bifet, Jessie Read, y tantos otros, de los que tanto he aprendido sobre minería de datos. Gracias también a Rudi Studer y Achim Rettinger por acogerme en el KIT, y a todas las personas que tuve el placer de conocer allí. Entre ellos, no puedo dejar de mencionar a mi compañera de despacho, Roberta, por su apoyo y amistad.

No me puedo olvidar de los casi tres años que he estado trabajando en el Hospital Costa del Sol con Marisa Hortas. Su capacidad de trabajo y su habilidad para unir dos campos tan diferentes como la informática y el análisis clínico, además de su calidez como persona, han sido una gran inspiración para mí.

No puedo dejar de mencionar tampoco a mis compañeros de viaje en el laboratorio 3.3.10. Especialmente quiero agradecer a Jaime y Enrique, los dos veteranos del laboratorio, que me han ayudado en tantas tareas, y con los que he compartido tan buenos momentos.

Pero el agradecimiento es extensivo a cada uno de las personas que han pasado por este laboratorio en todos estos años. Por supuesto, un agradecimiento muy especial va a Martyna, por su paciencia revisando y corrigiendo artículos; una parte importante de esta tesis es trabajo suyo también.

Por supuesto, gracias al Ministerio de Educación y Ciencia, por haberme concedido la beca para Formación de Profesorado Universitario (AP2009-1457), incluyendo la ayuda para la estancia en Alemania, sin la cual habría sido imposible llegar hasta aquí.

Por último, a nivel más personal, a Brise y a mi familia, que siempre me ha apoyado en este camino tan complicado.

Gracias a todos.

UNIVERSITY OF MÁLAGA

Abstract

Departamento de Lenguajes y Ciencias de La Computación
Escuela Técnica Superior de Ingenieria Informática

Doctor of Philosophy

by José María Carmona Cejudo

In the last few years there has been a rapid increase in the amount of electronically available data. This has fostered the emergence of novel data mining and machine learning applications able to extract information and knowledge. A significant proportion of these data sources are in the form of natural text, something which involves difficulties not present in other domains, such as their unstructured nature and the high dimensionality of the datasets. Natural text has to be preprocessed so that it can be analyzed by computers, and learning algorithms have to be able to cope with such high-dimensional feature spaces. Text mining techniques are invaluable to extract knowledge from natural text, as well as from other types of unstructured, alphabet-based data such as DNA strings.

Many of these data sources are not available as closed-ended datasets, but rather as data streams of examples that arrive in a sequence over time. This includes many text data sources, such as web pages, emails or blog posts. Given the unbounded nature of these datasets, it is important to work with scalable algorithms that use reduced time and memory. Additionally, it is important for the algorithms to be able to adapt to changes in the underlying statistical distributions governing the data. This is especially difficult in the case of data streams, because of their high dimensionality. In order for text streams to be computationally tractable, it is necessary to previously reduce the dimensionality of the datasets, employing only the most relevant terms in the learning algorithms. However, the

importance of the terms change over time, which in practice means that it is necessary to work under the assumption of a dynamic feature space. Keeping track of this evolving high-dimensional feature space is an intrinsically complex problem, since the importance of each feature depends on the others.

Such challenges are tackled in this thesis. We present GNUsmail, a framework for text stream classification in the domain of electronic email, and use it to study the nature of concept drift in text streams. We introduce a framework for adaptive classification, ABC-DynF, which is able to adapt to dynamic feature spaces, incorporating new features and labels to previously existing models. We also study the problem of summarization in text streams, and propose TF-SIDF / BM25, an approach for approximate weighting function approximation which makes it possible to extract keywords and construct word clouds from text streams in an efficient way. Finally, we present STFSIDF, an incremental approach for online feature selection which minimizes the number of weight recalculations while keeping continuously updated lists of the most relevant features. STFSIDF uses approximate algorithms to reduce the space complexity derived from the high dimensionality of the data sources.

Contents

List of Figures

List of Tables

Abbreviations

ABC-DynF	Adaptive Bayesian Classifier with Dynamic Features
ADWIN	Adaptive Windowing
AUC	Area Under Curve
AV	Added Value
AWSOM	Arbitrary Window Stream Modeling Method
BR	Binary Relevance
CVFDT	Continuous Very Fast Decision Trees
DDM	Drift Detection Method
DNA	Deoxyribonucleic acid
EDDM	Early Drift Detection Method
EPPT	Ensemble of Pruned Problem Transformation
FAE	Feature Adaptive Ensemble
FCWM	Fixed Cumulative Windows Model
KDD	Knowledge Discovery in Databases
LLSF	Linear Least Squares Fit
LSI	Latent Semantic Indexing
LP	Label Powerset
LR	Label Ranking
MLC	Multi-label Classification

MLE	Maximum Likelihood Estimator
MLR	Multi-label Ranking
MOA	Massive Online Analysis
NB	Naïve Bayes
NLP	Natural Language Processing
NN-ge	Nearest Neighbours with Generalized Exemplars
OLIN	Online Information Network
PHT	Page-Hinkley Test
PPT	Pruned Problem Transformation
RAkEL	Random k-Labelsets
ROC	Receiver Operating Characteristic
SANSPOS	String ANalysis by Sliding POsitioning Strategy
SPC	Statistical Process Control
STR	Short Tandem Repeat
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TF-IDF	Term Frequency - Inverse Document Frequency
TF-SIDF	Term Frequency - Sketched Inverse Document Frequency
VFDT	Very Fast Decision Trees
WEKA	Waikato Environment for Knowledge Analysis
UFFT	Ultra Fast Forest of Tress

Chapter 1

Introduction

Everything must have a beginning ... and that beginning must be linked to something that went before

Mary Shelley, *Frankenstein*

1.1 Introduction and motivation

In the last few years there has been a rapid increase in the amount of available data, largely due to Web 2.0 and mobile applications. This data explosion has made possible novel applications such as product recommendation [1], sentiment analysis [2], automatic spam detection [3], automatized analysis of medical literature [4], or personalized business advertising [5], among others. These applications are important not only because of their scientific challenges, but also for their potential economic and social impact. Extracting previously unavailable knowledge and patterns is of great importance for enterprises, since it allows them to be in a better position to undertake the right measures to successfully continue their growth.

For these reasons, it is becoming more and more important to develop computational techniques able to exploit this vast amount of data, helping to extract previously unknown information in order to

make better informed decisions. Making sense of large amounts of data is a non-trivial task that involves fields such as statistics, artificial intelligence, database technologies and, needless to say, domain knowledge about the specific problems. Making data useful involves a deep understanding of the problem and data. It is necessary to explore and improve the quality of data (which can be incomplete and even inconsistent in its original form), to consolidate heterogeneous data sources, and to apply sound modelling techniques. It is therefore a multidisciplinary task that requires participants with different backgrounds. The fields of *data mining* and *machine learning* have proven themselves to be essential for making information useful. They provide a large toolkit of effective techniques for data preprocessing, information extraction, predictive analysis, and clustering of data items. They are, at the end, the technologies that have made possible the aforementioned applications.

Non-surprisingly, most information is nowadays stored as documents in natural language [6]: news, articles from journals, forums, social networks, blogs or speech records represent a vast source of information that offers a unique opportunity to extract useful knowledge. The application of data mining to text datasets is known as *text mining*. Thanks to this technology, enterprises can mine blogs and tweets to find out what the users think about their products, mail clients can learn to differentiate spam and legitimate mail, search engines can cluster pages or news articles according to its contents, and so on. Other, somehow more controversial applications include mining the contents of chats and email to identify terrorism or violence threats [7].

In order to provide a concrete example of the potential of text mining, we refer to an influential early work by Don R. Swanson and Neil R. Smalheiser [4]. This work showed that automated analysis of the medical literature can lead to interesting hypotheses about causes of rare diseases. For example, when investigating causes of migraine headaches, they extracted various pieces of evidence that suggested that magnesium deficiency plays a role in some kinds of migraine headache, a hypothesis which did not previously exist in

the medical literature. The point is that a new potentially plausible medical hypothesis was derived from a combination of text fragments by means of computational data analysis. The extracted hypothesis was subsequently validated by medical experts [4]. This shows to what extent text mining is a promising field of research. In fact, text mining techniques are versatile enough to be applied to non-linguistic data sources. For example, they have been used with success to classify DNA strings [8], by extracting substrings from nucleotide sequences.

Unfortunately, the computational analysis of unstructured information such as free text is a extremely challenging task, since it involves some kind of understanding of natural language by computers. We have to remember that natural language understanding is in fact the criterion proposed by the classical Turing Test to judge if a machine exhibits intelligent behaviour [9]. This criterion depends on the ability of a 'computing machine' to impersonate a human sufficiently well in a conversation. There are several reasons for the difficulty of this task. Human language is ambiguous in terms of grammar and vocabulary, and background knowledge is not always present, or at least it is not easily accessible. In this thesis we assume that no *exogenous* knowledge is available, in the form of metadata, semantic annotations or others. That is, we only consider *endogenous* knowledge extracted exclusively from text documents.

A particularly challenging aspect of text mining is the high dimensionality associated to data sources. The large size of the vocabularies makes it necessary to develop scalable techniques that can deal with such a high dimensionality. In fact, text mining is a good benchmark for checking whether learning algorithms can scale to substantial sizes, as pointed out by Sebastiani [10]. This happens because even moderately large text datasets contain a large collection of different words that can potentially be used as classification features. This problem is intensified when more complex features are considered, such as n-grams or phrases. Such a high dimensionality leads to very sparse data, since most of the words of the vocabulary do not appear in a given document. An undesirable consequence is the so-called *Hughes-effect* [11]: given a fixed number of

training examples, the predictive power reduces as the dimensionality increases. The models will be prone to have a large variance, since they are too sensitive to the learning examples. In addition to this, high dimensionalities are also prejudicial from the computational point of view, since there are more features to be examined by the algorithms.

Multilabel classification is another kind of challenge which affects text classification. The classical single-label classification task consists in assigning a label from a predefined label-set to each document. By contrast, in many text classification tasks, it seems reasonable to assign more than one label to each document, which gives birth to the *multi-label* classification paradigm. This is the case of news classification. Let us think for example about a journal article about the consequences of the Fukushima nuclear disaster in 2012. It could be considered as an article about nuclear energy, about the consequences of such disaster in the economy of Japan, or even about its long-term health effects. Another example could be an article about the celebrations of a football team. Such article would be related to sport, but maybe also to street vandalism, if this was the case. These examples illustrate the fact that, in many occasions, it is desirable to produce learning models able to assign more than one single category to each data example.

A different type of challenge in data mining in general, and text mining in particular, is that of (theoretically) unbounded data sources, giving birth to the paradigm of *stream mining*. Traditional data mining problems involve a closed-ended dataset that contains a stationary collection of data examples. No new examples can be added to the dataset, and each one of them can be analysed as many times as necessary in order to preprocess the data and learn models for classification or clustering. The obtained model is not modified anymore. By contrast, new data sources have emerged where there is no previously fixed dataset, but a stream of data examples that arrive one after another. Such streams are highly dynamic and subject to evolution of their properties over time. Data streams have strict space and time requirements. Typically, data examples cannot be stored for future reference, and they have to be processed on-the-fly.

Each example can be analysed only once, or a very limited number of times at most. Moreover, the learning model has to be ready to make predictions at any given moment.

A particular difficulty associated with data streams is the emergence of *concept drift*. In contrast with classical, stationary datasets, the statistical distributions associated with data streams are subject to evolution. The distribution of labels, as well as the relationship between data and labels, changes over time. This means that the learning models have to be able to detect and adapt to these changes. A difficulty associated to text data streams corresponds to *contextual* concept drifts [12], which basically means that the relevance of the attributes changes over time. This is not an important issue in datasets with few possible attributes. Nevertheless, in text data streams, the features to be used for predictions are also subject to change over time, as new words appear or fall out of use, or as some words suddenly become more important for classification. This requires, on the one hand, incremental feature selection methods able to deal with text streams in an efficient way, and, on the other hand, classification models that work under the assumption of a dynamic feature space, that is, models that can vary the feature set over time, without having to train the model from scratch.

Finally, as we mentioned before, data streams have strict space requirements. Therefore, strategies to process high-dimensional data streams in efficient space are a necessity. Scalability is an ineluctable requirement for algorithms that work in a data stream scenario, and, in particular, linear complexities are something to be avoided [13]. Algorithms able to compress the dimensionality of the problem by projecting the data into a lower-dimensional space, such as sketch-based algorithms, represent a promising solution for tasks such as feature selection in sublinear size.

1.2 Objectives

The objectives that we pursue in this thesis are focused on text stream mining with dynamic feature spaces and concept drift. We

are interested in exploring techniques for tackling the challenges we have mentioned in the previous subsection, and in comparing them with the current state-of-art in text classification and data streams. We are also interested in new applications of text mining to non-linguistic sources. More specifically, in this thesis we pursue the following objectives:

- Review of the state of art of different fields related to our main topic: text classification, multilabel classification, data stream mining, concept drift detection, evaluation of online classifiers, and approximate algorithms. This will serve as the basis for the development of our approaches and algorithms.
- Application of text classification techniques to DNA strings, in order to select meaningful features to provide a word-based representation of DNA that can be used for visualization and classification into groups.
- Evaluation of different multi-label algorithms with respect to different multi-label performance measures in the domain of email classification, and study of relationships between measures. Exploration of the effect of feature selection regarding the performance of multi-label algorithms, and study of the best algorithms for this task. We plan to explore if our conclusions can be generalized to different multilabel performance measures.
- Implementation and publication of an open-source, Java-based extensible framework aimed at classification of streams of electronic mail with concept drift, including email extraction tools, feature selection techniques and state-of-art classification algorithms for stream classification and concept-drift detection.
- Use of statistical inference techniques to study the emergence of different types of concept drift in email datasets.
- Implementation and publication of an adaptive learning framework with dynamic feature space, to be used to compare several incremental and adaptive strategies to cope with concept drift and high-dimensional dynamic feature spaces.

- Study of how feature ranking methods, concept drift monitoring, adaptive strategies and the implementation of a dynamic feature space can affect the performance of email classification systems.
- Study of the effect of contextual concept drift in text streams by tracking the relevance of previous and new features and updating the feature space, using learning models able to include new attributes and drop old ones.
- Comparison of different strategies for updating the feature space.
- Study of sublinear approaches to incremental document vectorization, in order to comply with the space requirements found in the context of data streams. Studying to what extent these techniques affect the performance of the algorithms, and exploration of the best configurations for the data structures.
- Using the aforementioned mechanism to efficient keyword extraction and word cloud computation in text streams, in order to offer a mechanism for online document summarization with reduced space requirements.
- Introduction of a mechanism that allows time-efficient feature selection for text streams using weighting functions, avoiding continuous feature value update.
- Use of sketch-based algorithms for reducing the space requirements of the aforementioned mechanism, in order for the algorithm to be suitable for data streams. Optimization of the parametrization in order to obtain better approximations using less space.

In this thesis we use techniques from the fields of machine learning and data mining, such as feature space reduction, classification techniques and evaluation methodologies, as well as statistical inference for hypothesis proving. More concretely, we study multilabel learning and data stream mining techniques applied to documents in natural text. We use only endogenous knowledge, that is, information

included in the data, explicitly avoiding external knowledge such as ontologies, since this kind of knowledge is not always available or easy to collect. We use techniques from Information Retrieval and Natural Language Processing for preprocessing the datasets when necessary, although they are not the main topic of interest in this dissertation.

1.3 Outline of this thesis

The rest of this thesis is structured as follows:

- **Chapter 2: Fundamentals of Text Classification.** This chapter reviews the topic of text classification. The related fields of machine learning, data mining, natural language processing and information retrieval are put into relation with text classification. The main phases involved in text classification projects are discussed. This chapter justifies the importance of the topic, its main challenges and the usefulness of machine learning and data mining for dealing with this task.

- **Chapter 3: Applying text mining techniques to DNA strings.** This chapter shows how text mining techniques can be applied to non-linguistic domains (DNA strings), by using appropriate feature selection procedures. We propose a parallel version of the SANSPOS algorithm to extract frequent substrings, and an interestingness function to select the most meaningful ones. Relevant DNA substrings of variable length are used as features. This strategy is used for visualizing and comparing DNA sequences, as well as for mitochondrial DNA haplogroup classification.

- **Chapter 4: Multilabel Text Mining.** This chapter presents the topic of multilabel text mining. The main theoretical concepts are discussed, and an experimental evaluation of the impact of feature selection for multilabel classification of an email dataset is offered.

- **Chapter 5: Data Mining for Text Streams.** This chapter introduces the topic of data stream mining applied to text streams. Data stream mining is basically the application of data mining method to streams of data, where data items arrive continuously and cannot be stored for future reference. We present GNUsmail, a flexible framework for mining streams of emails, and use it to study strategies for adaptation to changes in the distribution of the data and labels (concept drifts). We compare several classification algorithms that can be applied to this field using state-of-art data stream evaluation techniques.

- **Chapter 6: Comparative Study on Feature Selection and Adaptive Strategies for Email Foldering Using the ABC-DynF Framework.** This chapter presents ABC-DynF, an adaptive learning framework with dynamic feature space that we use to compare several incremental and adaptive strategies to cope with these two difficulties. Several studies are carried out using datasets from the ENRON email corpus and different configuration settings of the framework. The main aim is to study how feature ranking methods, concept drift monitoring, adaptive strategies and the implementation of a dynamic feature space can affect the performance of Bayesian email classification systems.

- **Chapter 7: Online Attribute Weighting for Document Vectorization and Keyword Retrieval.** This chapter is devoted to the study of document vectorization in data streams. The exceedingly high dimensionality of text data streams make this problem too expensive in terms of the necessary space, since statistics have to be collected for every word in the vocabulary. Henceforth, a new approach for weighting function approximation using sketches, hash-based models able to project the data into a space with a lower dimensionality, saving space without losing effectiveness.

- **Chapter 8: STFSIDF: Efficient Online Feature Selection for Categorization in Text Streams.** This chapter deals with incremental feature selection in combination with

online classifiers accepting dynamic feature spaces. An approach is proposed for updating the set of used features in efficient time and space.

- **Chapter 9: Conclusions and Future Work**: This chapter provides our conclusions and proposes some possibilities for future work.
- **Appendix A: Resumen en español.** This appendix contains a summary of the thesis in Spanish.
- **Appendix B: Conclusiones y trabajo futuro.** This appendix is the Spanish translation of Chapter 9.

Chapter 2

Fundamentals of Text Classification

2.1 Introduction

As we mentioned in Chapter 1, text classification is the application of supervised learning techniques from the field of data mining and machine learning to data sets that contain documents written in natural language. Given the unstructured nature of natural language text, it is necessary to transform the documents into a machine-understandable format, more suitable for computer processing. In order to obtain suitable document representations that capture the underlying *semantics* of the text, techniques from Information Retrieval and Natural Language Processing have to be employed. This alone is generally not enough, since text documents suffer from problems such as their high dimensionality, which means that the documents have to be transformed into a more reduced feature space. After that, this processed versions of the documents can be used for learning and mining task, such as classification, using suitable algorithms. Furthermore, it is important to evaluate the performance of the algorithms, in order to select the best methods and parameters.

In this chapter we introduce each of these areas of study, focusing on the aspects more closely related to the problem of text categorization. The rest of this chapter is organized as follows: Section 2.2 gives an introduction to machine learning and data mining in general, which form the basis of text classification. Then, Section 2.3 introduces the key concepts of natural language processing, whereas Section 2.4 explains information retrieval concepts, focusing on document representation models. Section 2.5 is devoted to text mining, showing the relationship with the concepts from the previous Subsections are applied. Then, Section 2.6 explains the transformation of unstructured text into more suitable representations, and Section 2.7 discusses dimensionality reduction. Section 2.8 reviews the main algorithms for text classification, and, finally, Section 2.9 is devoted to techniques for evaluating text classifiers.

2.2 Data mining and machine learning

Progress in digital data acquisition, storage and processing has fostered the emergence of huge databases in all kinds of domains, which offer a vest amount of resources that can be analyzed in order to optimize systems, uncover valuable patterns or make scientific discoveries. The field of study within computer science which is devoted to extracting useful information from such databases is known as *Data mining*.

Data mining can be defined as the *process of analyzing data from different perspectives and summarizing it into useful information*[1]. Data mining systems analyze large quantities of data in order to extract previously unknown patterns. Data sets can be very large, in contrast with smaller ones that are used in the classical exploratory data analysis. Often the problem in data mining is to find generalizations of the data, that is, models that explain why the data is the way it is, and what are the hidden relationships in the data. Such

[1] http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm

Outlook	**Temperature**	**Humidity**	**Wind**	**Play**
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	78	false	yes
rain	70	96	false	yes
rain	68	80	false	yes
rain	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rain	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rain	71	80	true	no

TABLE 2.1: An example of dataset: The Golf dataset

models are used to discover similar groups of data, to find underlying association rules or to make predictions about previously unseen data examples.

Data mining works with *datasets*, sets of measurements taken from some environment or process [14]. Datasets contain a collection of n objects, known as *instances* or *examples*, and for each one of them there are p measurements available, which are known as *features* or *attributes*. A well-known example of artificial dataset which is often used in introductory data mining courses is the *golf dataset*. In this dataset, each example represents a day in a golf club, and each attribute represents an atmospheric characteristic (*outlook*, *temperature*, *humidity* and *wind*). There is another, special attribute, *play*, which indicates whether it was possible to play given the atmospheric conditions. This kind of special attributes are known as *supervising variable*, *class* or *labels*, and are the attributes of interest because we want to use the atmospheric conditions to predict whether, giving those conditions, is it possible to play. Datasets can be represented in tabular form. For example, the golf dataset is represented in Table 2.1:

In data mining terminology, there are different tasks that can extract different kinds of patterns:

- *Classification*: automatic assignment of items into one from within a finite set of categories. The category is the *objective* or *supervising* variable. Classification is an example of *supervised learning*, which means that there is a specific variable of interest, such as in the case of the golf dataset.
- *Regression*: automatic assignment of numeric values to items. It is conceptually similar to classification, but the *objective variable* is continuous instead of discrete. Regression is another example of supervised learning. A possible example of regression could be to forecast the temperature given other weather indicators.
- *Clustering*: search of groups of similar data records, according to some similarity measure. In contrast with classification and regression, there is no objective variable to be predicted. Thus, clustering is an example of *unsupervised learning*. An example of clustering could be finding homogeneous groups of clients of a supermarket.
- *Association rule discovery* between features of the data. Association rule learning is another example of unsupervised learning (although is can also be used for supervised learning). An often cited example of a situation where association rules can be useful is shopping cart analysis, to discover rules such as 'clients that buy beer usually buy chips too', which can be used by the supermarket manager to place the products accordingly in order to maximize sales.

Data mining is a non-trivial task, as we have previously mentioned. Nevertheless, for toy datasets such as the golf example we have mentioned, it can be carried out by hand. For example, in Table 2.1 it can be seen that when `overcast=true`, that is, when the day is overcast, it is always true that `play=true`. Else, if its raining, it is possible to play as long as it is not windy. Finally, if it is sunny, it is

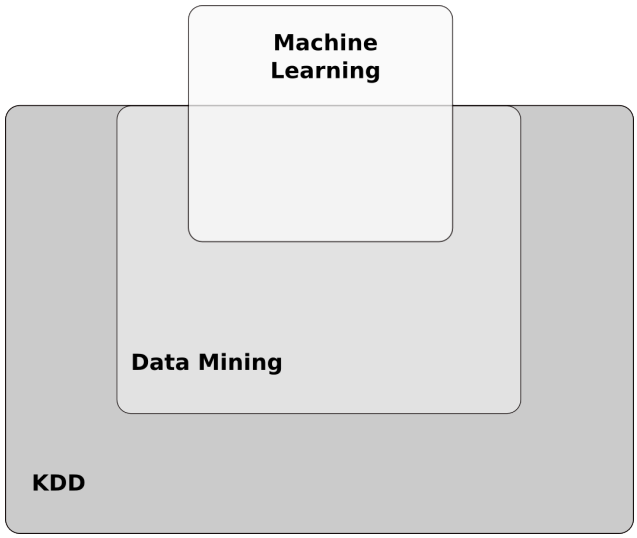


FIGURE 2.1: Relationship between KDD, data mining and machine learning

possible to play if the humidity is low. This kind of rules, that can be extracted by manual inspection, allow us to predict if it will be possible to play, given the weather forecast, and can be potentially useful for golf players and golf resort managers. These rules can be summarized in a *decision tree*, which is a particular kind of model used in data mining, as it is shown in Figure 2.2.

Of course, in non-toy datasets it is not possible to discover patterns by manual examination: learning algorithms have to be used to extract patters from data automatically.

Some authors define data mining as the analytic step of a *Knowledge Discovery in Databases* (KDD) process, while for others it is synonym for such process [15]. Broadly speaking, a generic KDD process can be divided in the following stages:

- *Data selection and preprocessing*, that is, assembling a suitable target data set, large enough to contain the sought patterns, while being small enough to allow the data to be analyzed in an acceptable time. The participation of domain experts is key

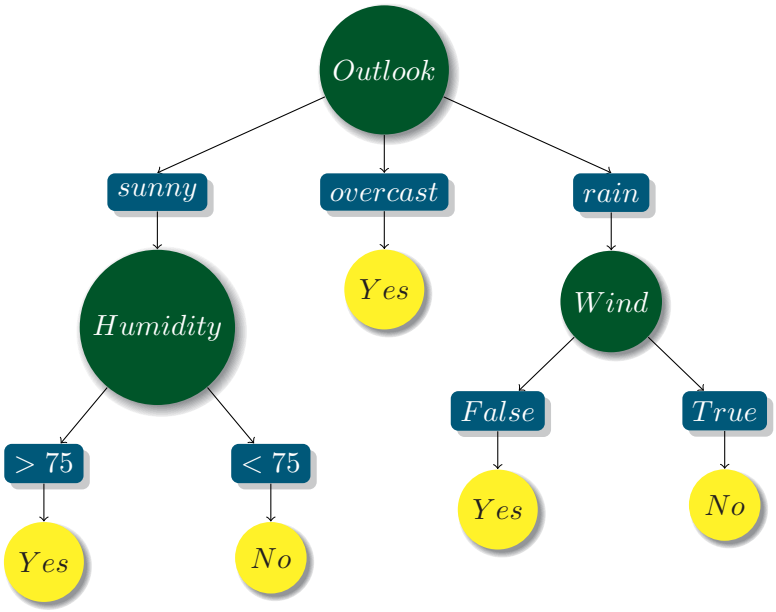


FIGURE 2.2: Example of a *decision tree* to decide if it is possible to play golf given the weather conditions

in this process, since they are familiar with the data sources and the information contained in them.

- *Data transformation*, in order to clean the data and deal with noise and missing data. Real-world data is frequently incomplete, contains noise and even inconsistencies. Think of a dataset containing information about the patients of a hospital. It would not be surprising to find typos (for example, patients with '900' years instead of 90 years due to a typing mistake), patients with missing birth date, incomplete histories or similar issues. It is important for data mining methods to be able to deal with such problems in order to provide well-founded results.

- *Modeling*, which involves the application of algorithms for classification, regression, clustering, association rule mining or others. This is the step where machine learning algorithms is applied to datasets in order to extract knowledge from them,

which will be used for classification, prediction, clustering and understanding of trends and patters present in the data.

- *Evaluation* of the models, in order to be able to state to which extent the learning algorithms have been able to generalize from the training data set. There are a big number of algorithms that can be used for the different data mining tasks. Furthermore, algorithms have several parameters that have to be tuned. It is important to try different algorithms and tune the parameters accordingly in order to obtain models that generalize the data as well as possible.

- *Deployment* of the models in a production environment, allowing the user to apply the models to new data and extract conclusions from them. For example, in the case of the golf dataset, this could consist in a web page which can access weather information and feed this data to a learning model (for example, the decision tree in Figure 2.2) to predict if it is possible to play today.

The KDD process has been formalized and standardized. A popular standard is CRISP-DM [16]. CRISP-DM defines an iterative work flow with the phases represented in Figure 2.3.

1. *Business understanding.* During this phase, the objectives and requirements are decided, and the main tasks are defined.

2. *Data understanding.* The data is collected, and the quality problems it may have are studied and solved.

3. *Data preparation.* Instances and attributes are selected or created, according to necessity.

4. *Modeling.* Different data mining algorithms are executed and tuned on the data, and the best combination is selected.

5. *Evaluation.* The evaluation of the obtained models helps to decide to what extent they are usable.

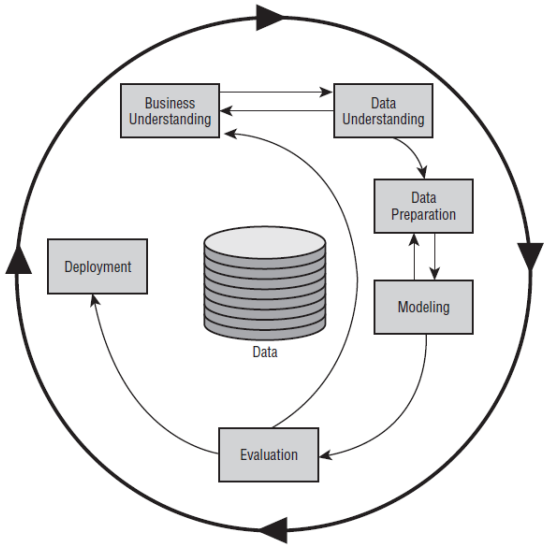


FIGURE 2.3: CRISP-DM standard. Source: http://bit.ly/h1gQPy

6. *Deployment.* The obtained and evaluated models are integrated in other information systems.

A field of study which is closely related to data mining is *machine learning*. Machine learning is a branch of artificial intelligence that aims to develop algorithms that generalize statistical phenomena, in order to make predictions on future data. The objective of a machine learning system is to *generalize from experience*. Mitchell provided the following commonly used definition of Machine Learning in [17]:

Definition 2.1. *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*

The objective of machine learning is to recognize complex patterns and make intelligent decisions based on data. It allows computer programs to 'learn for themselves' to carry out tasks that would be too difficult to be programmed as a fixed set of rules, such as face

recognition, automatic car driving, and other complex tasks where the experience helps to improve the performance of the algorithms.

Data mining and machine learning are overlapping terms, since data mining uses machine learning methods, while machine learning employs data mining methods, such as the preprocessing steps. This overlapping is graphically represented in Figure 2.1. KDD and data mining focus on discovering knowledge from data, including data storage and access, scalability issues, results interpretation and visualization [15], whereas machine learning includes fields not necessarily related to data mining, such as the theory of learning, computational learning theory and reinforcement learning.

Data mining and machine learning have been applied to a vast number of different fields, amongst them documents in natural language. Nevertheless, natural language is not directly processable by machine learning algorithms, given its unstructured nature. It is thus necessary to transform the documents into a format that can be used by learning algorithms to extract information. This transformation is not a trivial task, since the obtained representation has to retain the semantics of the original text. In consequence, specialized techniques from *Natural Language Processing* and *Information Retrieval* have to be applied in order to deal with textual documents. Both fields of study are introduced in the next sections.

2.3 Natural language processing

Natural language processing (NLP) is a multidisciplinary discipline concerned with the interaction between computers and human languages.

Texts usually contain information at different granularity, from words or tokens to rich hierarchical syntactic representations, which means that many different algorithms have to be employed for different tasks. NLP aims to building software able to analyze, understand and generate natural language contents.

During the 60s and 70s, most successful NLP systems, such as SHRDLU and ELIZE, worked with restricted domains and vocabularies, including ontological information for structuring real-world knowledge. At this stage of development, NLP systems were based on more or less complex sets of hand-written rules. The introduction of machine learning during the 80s meant a break-through in the development of NLP, taking advantage of statistical modeling of existing large-scale textual corpora.

Some of the main tasks NLP deals with are the following:

- *Speech recognition*: phoneme recognition and transcription of spoken audio clips into text. This is commonly applied in the field of telephony, for example, where machines can understand the words of the user in order to automatically redirect the user to the most adequate operator.

- *Natural language generation*: generation of information in human language. This task is frequently used to automatically provide textual descriptions of computer databases, and has been applied to provide written weather forecasts from relational databases. A successful application in this line is ARNS (`http://www.cogentex.com/solutions/arns/index.shtml`), designed to provide human-understandable summaries of observations and predictions of water levels, coastal currents and other meteorological and oceanographic data.

- *Machine translation*: transformation of texts from one human language to a different one. Machine translation has played an important role in the history of NLP. The Georgetown experiment in 1954 consisted in automatically translating a dataset of Russian sentences into English, using a simplified grammar and a rather modest vocabulary [18]. Although the results were promising, the ALPAC report in 1966 found that the research on automatic translation had not fulfilled the initial expectations, resulting in a reduction of funding for research in this field [19]. Although machine translation is still an open problem, some successful applications exist, such as Google

Translate, a free multilingual translation platform based on statistical techniques.

- *Automatic summarization*: produce a summary of a longer text, in order to avoid information overload. This can be achieved simply by extracting the most important keywords for the meaning of the text, or by producing coherent sentences that describe the essential contents of the objective text.

- *Named entity recognition*: recognition of proper names inside texts. State-of-art named entity recognizers use linguistic grammar-based techniques and statistical models, and have a near-human performance [20], although they are designed for specific domains [21]. This task is related to *relationship extraction*, the extraction of relationships between named entities appearing in texts.

- *Part-of-Speech Tagging*: given a sentence, determination of the part of speech (verb, noun, etc.) for each word. Different approaches have been used historically for this task, such as hidden Markov models or dynamic programming methods [22]. Triviño-Rodríguez et al. [23] use Multiattribute Prediction Suffix Graphs, a model similar to multiattribute Markov chains, for part-of-speech tagging in Spanish texts. Named entity recognition is a difficult task, since the ambiguities in natural language make it difficult to decide the actual role of a word in a sentence. A paradigmatic example that demonstrates this difficulty is the valid English sentence *Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo*, meaning 'Buffalo-origin bison that other Buffalo bison intimidate, themselves bully Buffalo bison' [24]. Although this is an artificial example, it illustrates the challenges faced by Part-of-Speech tagging applications.

- *Sentiment analysis*: extract subjective information from text (that is, discovering the attitude of the writer about what he or she is writing about). The rise of social media and networks has boosted interest in sentiment analysis, given its potential

economical impact. For example, enterprises can search Twitter examples that talk about them, and discover if the attitude towards their products or services is positive or negative [25].

- *Information Retrieval*: storing, searching and retrieving information. It is actually a separate field within computer science (see section 2.4): storing, searching and retrieving information.

The last one, information retrieval, is key to text mining, since it provides tools for document representation. It is expanded in the next Section.

2.4 Information retrieval

Information retrieval (IR) can be broadly defined as an area of study within computer science concerned with searching for documents with certain characteristics, as well as for information and metadata within them. Typically, an IR process involves entering a query into the system (for example a search string), in order for the system to return the most closely related documents to the user.

In origin, IR could be viewed as a subfield of NLP, although nowadays it is considered as an independent field of study. In fact, IR does not only involve textual documents, but also multimedia content such as images and videos. Nevertheless, we restrict our exposition to the textual domain.

Some of the applications or IR are the following:

- *Document ranking* with respect to query. That is, given a query, retrieving related documents ordered by relevance. This is what web search engines typically do: they are given a search query, and a set of documents related to it is shown in order of relevance.

- *Information filtering*: removing redundant or unwanted information from an information stream. Only the most important

or relevant pieces of information are returned. This is related to the aforementioned field of automatic summarization, as its goal is management of information overload. A particular kind of information filtering systems that has gained popularity in the context of Web 2.0 are recommender systems, which aim at presenting to the user the items (products, movies and so on) that he or she is likely to be more interested in. Machine learning plays an important role in learning to distinguish important from unimportant information.

- *Topic detection and tracking.* This task consists in detecting and tracking emerging events or activities in text streams, such as breaking news topics.

- *Question answering*: providing an answer to a question provided by the user. This task requires a considerable volume of background information in order to be successful.

2.4.1 Document Representation Models

In IR, documents are not typically stored in their raw form. As anticipated in Section 2.1, they are transformed into a suitable *representation model* that enables efficient processing. There are different types of models for this purpose, that can be divided into three main types: *set-theoretic models*, *vector space models* and *probabilistic models.*

The first kind of models, *Set-theoretic models*, represents documents as sets of words or sentences. In order to measure the similarity between a document and a query (or another document), set operations such as *intersection* are used.

The second type corresponds to the *Vector Space Model.* In this approach, a document is represented by a vector of terms (typically words and phrases). Each term corresponds to an independent dimension in a high dimensional vector space, and each document is mapped into a point in this space. The similarity between a document and a query can be measured using a distance measure in the

multidimensional space, such as the *cosine* or *Euclidean* distance. The cosine distance is the most commonly used, given its simple mathematical form, equivalent to the dot product. If D is the vector corresponding to a document and Q the vector corresponding to a query, the similarity can be computed as follows:

$$Sim(D, Q) = \sum_{t_i \in Q, D} w_{t_{i_Q}} w_{t_{i_D}} \tag{2.1}$$

A key component in the Vector Space Model is *term weighting*, that is, assigning *weights* (real values) to terms. This weights can be calculated using different formulations, but in general they depend on *term frequency* (words that appear frequently are considered important), *document frequency* (words that appear in many documents are considered common and not very indicative) and *document length*, mainly used to normalize weights. Two common weighting functions are *TF-IDF* and *Okapi BM25*, which will be explained in later Sections.

In the Vector Space Model, the dimensions need not be words or phrases. There are approaches, such as the *Latent Semantic Indexing* (LSI) in which linear combinations of terms originating from a *Singular Value Decomposition* analysis are used as dimensions. Such techniques uncover the underlying latent semantic structure in the corpora vocabularies.

The third type of document representation models are the *probabilistic models*, in which document retrieval is modeled as a probabilistic inference, and similarities are computed as the probabilities that a given document is relevant for the corresponding query. A prominent example of the type of models is the *Latent Dirichlet allocation* [26].

So far, data mining and machine learning have been introduced, as well as information retrieval and natural language processing, which provide tools for learning algorithms to deal with natural language. In the next Section the interaction of these elements in the context of text mining is explained.

2.5 Text mining

Text mining consists in the application of data mining to textual domains. Its objective is thus to derive relevant, novel and interesting information from text. An important application of data mining is *text classification*, that is, the automatic assignment of documents to predefined categories, based on their contents. The classification can be supervised, where information on the correct classification of the documents is provided externally, or unsupervised document classification (document clustering). In the research community the dominant approach for text categorization is based on machine learning [10].

Until the late '80s the most popular approach was based in manually defining a set of rules that encoded expert knowledge (*knowledge engineering*) [10]. In the '90s, this approach lost popularity in favour of the machine learning paradigm, based on an inductive process that automatically learns a classifier from a set of already classified examples.

2.5.1 Text categorization

Let D be a *domain of documents* containing $|D|$ documents. Let $C = \{c_0, c_1, \ldots, c_{|C-1|}\}$ be a set of symbolic labels. Let $f : D \to C$ be a classification function, that assigns a category to each document. The objective of text categorization is to approximate a target function $\hat{f} : D \to C$ which approximates f as well as possible, according to some similarity measure. f (and thus $\hat{f}$) need not be functions, but can also be general relations. In this case we talk of *multilabel classification* in contrast with *single-label classification*. Chapter 4 is devoted to multilabel learning. In this Chapter we restrain ourselves to single-label classification.

Some applications of text categorization include automatic indexing, document organization, text filtering or hierarchical categorization of web pages.

2.5.2 Machine learning for text categorization

The origins of text categorization can be traced back to the work of Maron [27], where a statistical technique is developed to determine certain probability relationships between document words and categories. During the '80s, the most popular approach to text categorization was based on *knowledge engineering* techniques: expert systems consisting of a set of manually defined rules were developed. An early successful system that used this strategy was the CONSTRUE system [28]. The results obtained were promising (both precision and recall were about 90% for the Reuters collection). Nevertheless, the rules in CONSTRUE were tailored for specific domains, so adapting the system to different ones would be costly. This is known in the experts systems literature as the *knowledge acquisition problem.*

Machine learning has been the dominant approach since the early '90s. In this approach there is an inductive process (the *learner*) that automatically builds a *classifier* using a set of previously classified documents (the *training set*), such that it is able to predict the category of a previously unseen document. This approach offers far more flexibility than the Knowledge Engineering-based approach, since the inductive process can cope with an updated category set, and can be used in different domains [10].

Text Mining involves various phases: indexing (Section 2.6), dimensionality reduction (Section 2.7), learning (Section 2.8) and evaluation (Section 2.9).

2.6 Indexing

The first step when dealing with natural text for machine learning is to transform the documents into a format that can be interpreted by the learning algorithms. This is called *document indexing*, and IR techniques are used for this task (see Section 2.4). A typical approach is based in representing each document as vector $d_j = (w_1, \ldots, w_{|T|})$ of *weighted terms* or features, where T is the

vocabulary (the set of terms that occur in the documents), and w_k is a value that represents the *weight* of that term to document d_j. Broadly speaking, the most important a term is for the semantics of a document, the larger weight it is assigned. Each *term* or feature usually corresponds to one word. In this case, we say we are employing the *bag-of-words* approach. Nevertheless, features can also correspond to n-grams, phrases or more sophisticated representations.

Differences among the different indexing approaches are based on the following:

- Defining what a term is. If each term corresponds to one word, we say we are employing the *bag-of-words* approach. Nevertheless, features can also correspond to n-grams, phrases or more sophisticated representations based on latent factors.

- Deciding how to compute weights. A trivial approach to this is to assign $w_{jk} = 1$ if t_j appears in document d_k, and 0 in other case. Another frequent weighting function is the *term frequency*. In this function, the most frequent a word is in a document, the highest weight it is assigned. The problem of term frequency is that terms that appear in most documents are not very representative of the semantic contents. Let us think about the case of prepositions (to, from, in...). Such words are very frequent inside documents, but at the same time appear in most documents, so their discriminative power is small. That justifies the introduction of functions like *TF-IDF*. This function is defined as follows:

$$\text{TF-IDF}(i,j) = \frac{n_{i,j}}{\sum_k n_{k,j}} log \frac{|D|}{|\{d : t_i \in d\}|} \tag{2.2}$$

  where $n_{i,j}$ be the number of occurrences of the word t_i in document d_j, $|D|$ the cardinality of D, and $|\{d : t_i \in d\}|$ is the number of documents where the term t_i appears.

  Note that all of these weighting functions disregard the order in which documents appear.

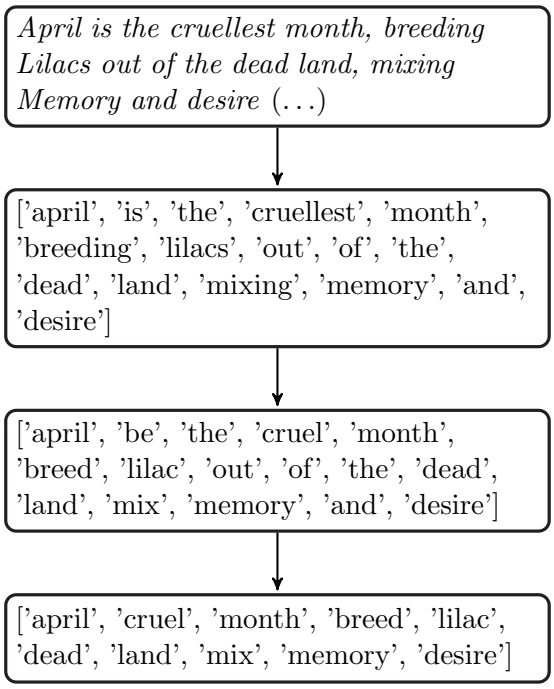


FIGURE 2.4: Example of indexing, using the first verses of *The Waste Land*, by T.S. Eliot. The first node represents the raw text. The second node represents the tokenized text. The text is stemmed (only the word root is kept). In the final step, stop-words are removed. The result can be used in a bag-of-word representation.

A step frequently performed during the indexing is the deletion of *stop words*, that is, structural words with no semantic information, such as prepositions or articles. Additionally, *stemming* is frequently carried out. Stemming consists in removing word suffixes in order to use only the root, which is the part carrying semantic information.

A prominent indexing approach in the literature is the *Darmstadt Indexing Approach*, used in the AIR/X system [29]. This approach is based on the idea of using a much wider set of features, where *properties* build the dimensions (of terms, documents, categories or relationships) of the learning space. This approach allows to take into account features such as the location of a term within a document, which is not possible with the classic bag-of-words approach.

A simple example of document indexing is shown in Figure 2.4. In this example, the original text is converted into lower case and tokenized (that is, the terms are extracted), then the words are stemmed, and finally stopwords are removed, obtaining a vector of words that represent the document.

2.7 Dimensionality reduction

Many algorithms used for text classification typically suffer from high dimensionality, hence it is often necessary to reduce the size of the vector space T and work with a reduced space T'. Such reduction is also useful to reduce overfitting, since *noisy attributes* are removed.

Two main strategies for dimensionality reduction can be defined:

- Dimensionality reduction by *term selection*: T' is a subset of T
- Dimensionality reduction by *term extraction*: the terms in T' are not of the same type of the ones in T (for example, they are obtained by transforming the original terms).

Regarding term selection, two main approaches can be found in the literature: the *wrapper* approach and the *filtering* approach, although hybrid models have been proposed too [30]. Wrapper methods involve a search procedure where a classification performance measure is optimized in the space of possible feature subsets [31]. Their applicability in text classification is limited because they are very computationally intensive: they require building a classification model for each of the evaluated feature sets. Moreover, wrappers have a higher risk of overfitting.

Filtering methods, on the other hand, allow a fast and scalable computation. The resulting feature set is not adjusted to a specific classifier model. They use the intrinsic properties of the data to assess the relevance of features for the classification task. The easiest

way is to rank the features by the value of some score according to a weighting function that measures the relevance of each feature [10] and then choose the features with the highest scores. Among the most widely used weighting functions are document frequency, chi-squared, information gain, mutual information, etc. Previous works [32, 33] present a good overview and some results from empirical studies aimed at comparing several feature selection metrics for text classification

A simple scoring function is chi-squared, based on the common statistical test for independence that measures the divergence from the distribution expected if we assume that feature occurrence is independent of the category [33]. Given a dataset $\mathcal{D}$, the local chi-squared function for a feature t_k in a category c_i can be calculated as follows:

$$\chi^2(t_k, c_i) = \frac{|\mathcal{D}|[P(t_k, c_i)P(\bar{t}_k, \bar{c}_i) - P(t_k, \bar{c}_i)P(\bar{t}_k, c_i)]^2}{P(t_k)P(\bar{t}_k)P(c_i)P(\bar{c}_i)} \tag{2.3}$$

The chi-squared function assumes binary features, so it is appropriate for the bag-of-words representation of text datasets: t_k is `true` in a document $\mathbf{x}$ if term t_k appears in $\mathbf{x}$, and `false` otherwise. Then, the chi-square score for a feature t_k is calculated as the sum of the chi-square scores for each category [10]: $\chi^2(t_k) = \sum_{i=1}^{|C|} \chi^2(t_k, c_i)$, where $|C|$ is the total number of categories.

Dimensionality reduction by term extraction is based on replacing the set of original terms with a different set of *synthetic* terms that maximize effectiveness [10]. The rationale behind this is that the original terms may not be optimal dimensions for document representation, because of the problems of homonymy and polysemy. The idea is to create artificial terms that do not suffer from these problems. Two important techniques in this sense are *term clustering* and *Latent semantic indexing*.

- *Term clustering* is based on the idea of grouping semantically similar terms into *term clusters*, and employing the centroid of the clusters instead of the original terms.

- *Latent semantic indexing* is a technique with roots in IR document classification. It is based on compressing documents into a lower-dimensionality space via a linear transformation, based on co-occurrent patterns. This is achieved using matrix factorization techniques such as Singular Value Decomposition (SVD). LSI is able to correlate semantically related terms that are latent in a collection of text, and it is a method able to deal with polysemy and synonymy. The main disadvantage of LSI is its high computational cost.

Other techniques for term extraction have their origin in the field of soft computing [34], such as Ant Colony Optimization, used in the work of Aghdam et al. [35]. Evolutionary algorithms have also be used, for example in the work of Freitas [36]. In the rest of this thesis we focus on dimensionality reduction by term selection.

2.8 Algorithms for text classification

A large number of different machine algorithms have been used for classifying text; in this Section we review the most relevant ones.

2.8.1 Probabilistic classifiers

The first group of classifiers that we study are *probabilistic classifiers* [37]. They are based in estimating the probability that a document d_j projected into a vector space model with D different words $\langle w_1, w_2, \ldots, w_{|D|} \rangle$ belongs to category c_i, that is $p(c_i|d)$. For categorization purposes, the category with the highest probability is considered to be the category corresponding to the document. This probability can be derived from the distribution of words in classes by using Bayes' theorem:

$$p(c_i|d_j) = \frac{p(c_i)p(d_j|c_i)}{p(d_j)} \tag{2.4}$$

The probability $p(d_j|c_i)$ is difficult to calculate, because of the high number of possible word vectors. Therefore, it is frequent to make the assumption that the features of the vector space are independent, that is, $p(d_j|c_i) = \prod_k p(w_k|c_i)$. That is:

$$p(c_i|d_j) = \frac{p(c_i)}{p(d_j)} \prod_k p(w_k|c_i) \tag{2.5}$$

The resulting classifier is called Naïve Bayes [37], due to the 'naïve' assumption of feature independence. Log-likelihood ratios are used in order to avoid underflow due to multiplication of a large number of small probabilities:

$$\log p(c_i|d_j) = \log \frac{p(c_i)}{p(d_j)} \sum_k \log p(w_k|c_i) \tag{2.6}$$

In order to estimate probabilities, maximum-likelihood estimators approaches (MLE) are usually avoided due to the impact of non-appearing words, which would transform the product of probabilities into 0. Laplacian estimators are a frequently used alternative to MLEs. Despite the independence assumptions of Naïve Bayes, this algorithm has been experimentally found to perform reasonably well in text categorization [38].

2.8.2 Linear classifiers

Another important family of algorithms for text classification are the ones based on regression. A prominent example is logistic regression [39], which, despite its name, is actually a classification method. Logistic regression is based on linear regression, and it is used for binary classification, that is, deciding if an instance corresponds or not to a given category. Logistic regression returns the probability $p(c_i = True|d_j)$ that a document d_j belongs to a category c_i. In the case of having more than one scenario, a logistic regression model can be used for each category, and the category with the highest probability for a given instance is chosen as the assigned category

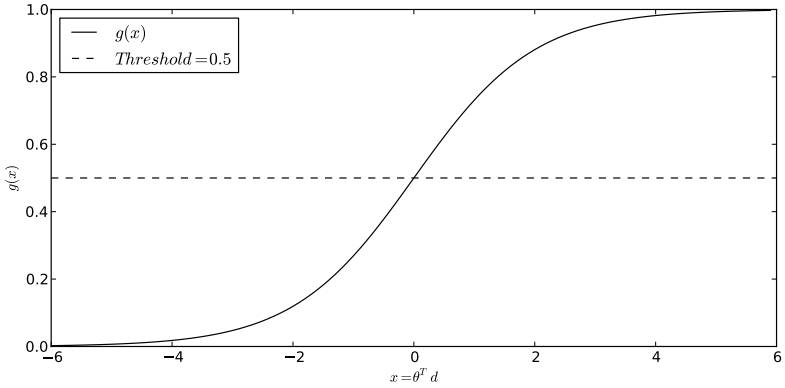


FIGURE 2.5: Example of Logistic Regression

for it (that is, the *One-versus-All* (OVA) binarization strategy is used). The objective of logistic regression is to approximate a function f that maps each instance into 1 (if the instance belongs to the class) or 0 (if it does not belong to the class). To this end, a logistic function $g_c(x)$ is fitted for each class c to the data, where:

$$g_c(d) = \frac{1}{1 + e^{-(\theta^T d)}} \tag{2.7}$$

where θ^T is the vector of parameters to learn in order to adjust the function, at the reason of one parameter for each word in the vocabulary. The parameters can be learnt using an optimization procedure, such as gradient descent. The value of $g_c(d)$ is between 0 and 1, and thus can be interpreted as the probability that d belongs to category c.

From a geometrical point of view, what logistic regression does is to find a hyperplane that separates positive and negative examples: for example, a threshold of $g(d) = 0.5$ can be established, that is, the classifier decides that the category of document d_j is c_i if and only if $g_{c_i}(d_j) > 0.5$. This is shown in Figure 2.5.

The idea of finding a hyperplane that separates positive and negative examples is also exploited by Support Vector Machines (SVM)

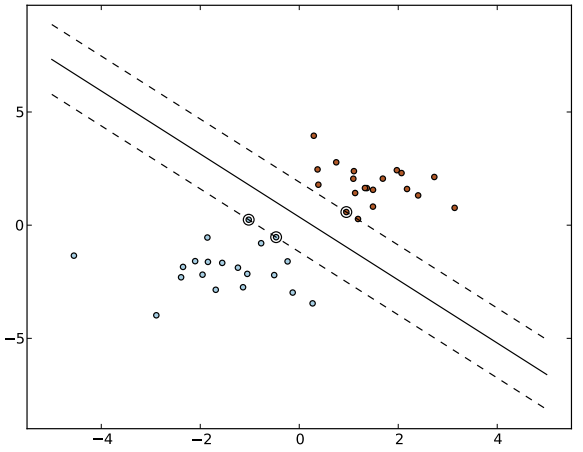


FIGURE 2.6: Maximum-margin hyperplane and margins for a SVM to distinguish between two classes. The instances on the margins are called the support vectors.

algorithms. If the training data are linearly separable , the objective is to select two parallel hyperplanes that separate the negative and positive examples, trying to maximize the 'margin' (that is, the distance between the hyperplanes). See Figure 2.6. If the data are not linearly separable, hyperplanes are selected such that they split the data as cleanly as possible (that is, minimizing the number of misclassified examples). This basic version of SVMs are a type of linear classifier. It is possible to use SVMs for non-linear classification using the so-called *kernel trick*, which consists in replacing the dot products associated to the geometrical equations of the hyperplanes with non-linear kernel functions, such as polynomial functions or the Gaussian radial basis function. Although the kernel trick is a powerful concept, the basic linear version of SVMs usually works well for text classification [10]. The reason is that the kernel trick has the effect of augmenting the dimensionality of the problem. Text is already highly dimensional, which means that applying the kernel trick has the effect of augmenting the error due to variance in the bias-variance trade-off. In other words, the models become prone to overfitting to the training set. For this reason, linear kernels (that

is, the basic linear version of SVM) are commonly used for text classification.

2.8.3 Neural networks

Another common algorithm for text classification are *neural networks*. In a neural network text classifier, there are some input units that represent terms, some intermediate units (the *hidden layers*) and some output units that represent categories. The weights on the edges connecting units represent dependence relation. For classifying a document d_j , its term weights w_k are loaded into the input units, the activation of these units (according to a given activation function) is propagated, and the activation value of the output units determines the categorization decision. Neural networks are frequently trained using the backpropagation algorithm. The simplest kind of neural network is the perceptron algorithm [40], in which the architecture consists only of an input layer, where each input node represents a word, and an output layer where each node represents the output for a given category. The perceptron is a linear algorithm, since the output depends on a lineal combination of the input values. The weight of each term can be interpreted as the relevance of that term with respect to the category. In the perceptron algorithm, each time a new document arrives, the weights are modified if and only if the classification was not correct. In that case, the weights of the terms that occur in the document are increased, while the weights for the other terms are decreased. This means that the perceptron algorithm can be seen as a sort of 'on-the-fly' feature selection mechanism [41].

The inclusion of internal or 'hidden' layers allows neural networks to learn more complicated functions. See a representation of a typical neural network with one hidden layer in Figure 2.7.

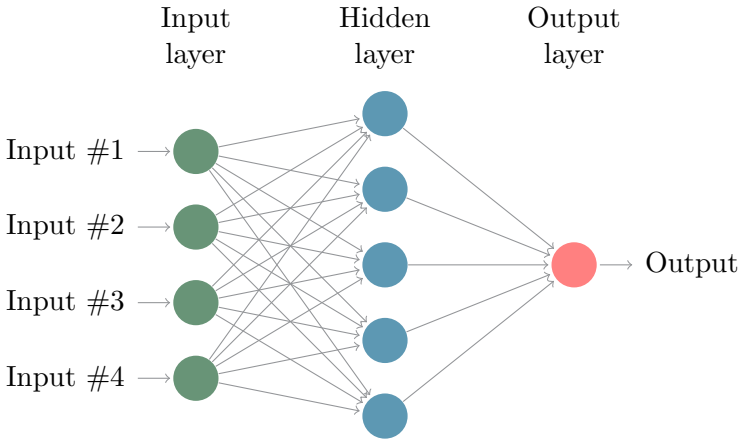


FIGURE 2.7: Example of a neural network architecture with one hidden layer

2.8.4 Example-based learners

A completely different family of text classifiers are the *example-based classifiers* [42]. Example-based classifiers do not build an explicit representation of categories, but rely on the categories of training instances with are similar to the test instance to be classified. These methods are also known as lazy learners, since they '*defer the decision on how to generalize beyond the training data until each new query instance is encountered*' [17]. An example of such algorithms is the *Nearest Neighbor algorithm*, which stores all the training data in memory. When a unlabelled item has to be classified, the algorithm computes the distance to all the stored items, using a similarity function, and determines the nearest neighbor (or the k nearest neighbors) [43]. Examples of similarity functions are the Euclidean distance or the cosine similarity, which is often used in the vector space representation. Lazy learners are fast to train, but they require a large space to store the entire dataset and are slow to evaluate. The use of example generalization helps to reduce these problems; an example of example-based algorithm which generalizes similar examples is NN-ge [44].

2.8.5 Decision trees and association rules

A further commonly-used family of algorithms are those based on *decision trees*. A decision tree classifier consists in a tree data structure, where each internal node n_i represents a feature f_i, and branches departing from n_i represent tests on f_i. Each leaf node represents a category. An example of decision tree is shown in Figure 2.2. In order to classify a given document, its features are recursively checked until a leaf node is reached, which indicates the category. Decision tree learners are based on recursively partitioning the training data into subgroups, following a top-down strategy, by checking the value of a given attribute, which is selected in order to maximize some criterion, such as expected information gain. The training data is recursively partitioned until all the examples in a node have the same value, or until splitting no longer improves the performance. Examples of decision trees learners are ID3 and its successor C4.5 [45]. The main advantage of decision trees is their easy interpretability, since their model is easily interpretable by humans, regardless of their predictive accuracy.

This easy interpretability is also a property of inductive rule classifiers. An *inductive rule classifier* consists in a set of rules, each one of them consisting in a *premise* (a Boolean test over a set of features) and a *clause head*, which represents he category a document should be classified into if the premise is true. Inductive rule learners are usually built in a bottom-up fashion, beginning with very specific rules and generalizing them by simplifying or merging clauses. A prominent example of inductive rule learners for text classification is RIPPER [46], based on recursive data partitioning that supports multi-valued attributes. RIPPER has been successfully applied to email classification [46].

2.8.6 Ensemble learning

Finally, *Ensemble learning* is a technique for combining a set of weaker learners in order to obtain a stronger learner. It has been empirically shown that the performance of ensembles is better when

the models are diverse enough [47]. Ensemble learners try to promote diversity. In the specific case of text classification, this means either using different indexing strategies, and thus different feature space, or using different learning algorithms (or both). In general, ensemble methods involve the simultaneous use of different learning models, which partial results have to be aggregated into the final result.

A simple example of ensemble learning is the *majority vote* strategy [48], where k independent classifiers propose a class for a document, and the class with the most votes is selected. Another important ensemble method is *boosting* [49]. In this method, we have k learners $\Phi_0, \Phi_1 \ldots, \Phi_{k-1}$, all of them learnt sequentially using the same base algorithm. Each classifier Φ_i is tuned in order to solve the most difficult documents to classify found so far. After all the k classifiers have been built, their output is combined using a weighted linear combination rule. The sub-classifiers are not trained in parallel, but sequentially. Boosting algorithms have been proven successful for the task of text categorization.

A different approach was proposed in the work of Ramos-Jiménez al. [50], where the FE-CIDIM algorithm was presented. This algorithm is based on a multiple classifier system that uses decision trees (CIDIM) as the base algorithm. This approach is characterized by the iterative induction of basic classifiers until none of the new basic classifiers improves the performance of the previous ones. It is based on the E-CIDIM algorithm, which keeps a maximum number of trees and induces new trees that may substitute the old trees in the ensemble. The substitution process finishes when none of the new trees improves the accuracy of any of the trees in the ensemble after a pre-configured number of attempts. FE-CIDIM was designed to speed up the convergence of the learning process. A two-layered ensemble system was presented in the work of del Campo-Ávila et al. [51]. This system has a layer with multiple classifiers, and a second layer consisting of a single classifier that solves disagreements among the classifiers in the first layer. A further type of ensemble methods are *correction filters* [52], structures that learn which subspaces of the global space is correctly learnt by each base classifier.

	YES is correct	**NO is correct**
Assigned YES	a_i	b_i
Assigned NO	c_i	d_i

TABLE 2.2: Contingency table for a category i

The results of the base classifiers are then combined according to a voting scheme.

2.9 Evaluation

Evaluation of text classification systems is generally carried out in an experimental fashion, since analyzing such systems analytically would require a formal definition of the problem, while the task of text classification is very difficult to formalise [10]. The main objective of evaluation is to compare the performance of different machine learning algorithms.

2.9.1 Evaluation measures

In the field of text classifiers, evaluation is more focused on *effectiveness* rather than on *efficiency*. That is, the main point of interest is if the system is able to take the correct classification decisions. Effectiveness measures are based on the observation that the category assignments of a binary classifier can be evaluated using a two-way contingency table for each category, as in Table 2.2.

Some important evaluation measures are the following:

- *Precision and recall*. Precision is the probability that, if a decision has been taken to classify document d under category c, this decision is correct, while recall is the probability that if the correct category for document d is c, a decision has been taken in favour of this classification. We have a contingency

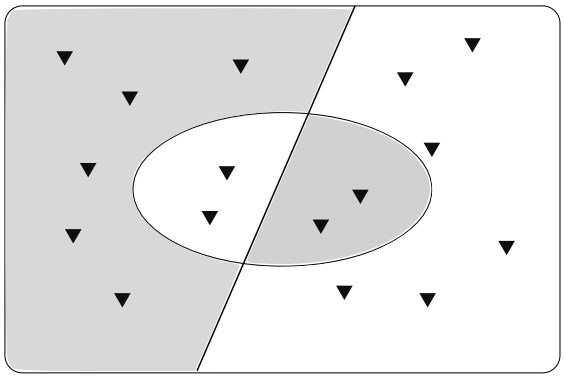


FIGURE 2.8: Precision and recall: graphical representation

table for each category, such as in Table 2.2. Estimates of precision and recall for a category c_i can be obtained as

$$precision_i = \frac{a_i}{a_i + b_i}, \; recall_i = \frac{a_i}{a_i + c_i} \tag{2.8}$$

The relationship between precision and recall is graphically shown in Figure 2.8. In this figure, the documents are represented by points. The set of all examples is delimited by the square area, where the set of documents retrieved by a query is delimited by the oval area. The actual relevant items are at the left side of the straight line. The errors are represented in gray. More specifically, the items at the left of the straight line that are not within the oval are *false negatives* (because they should have been retrieved, but were not), whereas the items at the right side of the straight line that are within the oval are *false positives*, since they were retrieved, but should have not been. *Precision* is the recall of the white region within the oval by the whole oval, and *recall* is the quotient of the white region within the oval and the left region.

Precision and recall are binary measures in nature, that is, they refer to a specific class. Nevertheless, in the case of multiclass problems, the binary measures can be averaged in order to obtain a global evaluation for the set of all categories. There are two ways for averaging these measures. The first one is called *microaveraging*, and is based on summing over

the individual decisions:

$$precision^{\mu} = \frac{\sum_{i=1}^{|C|} a_i}{\sum_{i=1}^{|C|} a_i + b_i},\ recall^{\mu} = \frac{\sum_{i=1}^{|C|} a_i}{\sum_{i=1}^{|C|} a_i + c_i} \tag{2.9}$$

The second one is *macroaveraging*, and is based on first evaluating the result for each category, and then averaging these values, that is:

$$precision^{M} = \frac{\sum_{i=1}^{|C|} precision_i}{|C|},\ recall^{M} = \frac{\sum_{i=1}^{|C|} recall_i}{|C|} \tag{2.10}$$

Microaveraging gives the same weight to all instances, whereas macroaveraging gives the same weight to all categories.

- The F_β measure summarizes precision and recall. More specifically, it is a weighted average of precision and recall. Its most traditional version is the F_1 score (the harmonic mean of precision and recall):

$$F_1 = 2\frac{precision\ recall}{precision + recall}. \tag{2.11}$$

The general form of this measure is

$$F_\beta = (1 + \beta^2)\frac{precision\ recall}{\beta^2 precision + recall} \tag{2.12}$$

When $\beta = 1$, this measure is maximized if $precision = recall$.

- *Error and accuracy*, as well as precision and recall, can be calculated from the contingency table 2.2:

$$accuracy_i = \frac{a + d}{a + b + c + d}, error_i = \frac{b + c}{a + b + c + d} \tag{2.13}$$

These measures are not widely used in text categorization, since the large value of the denominator makes them insensitive to variations in the number of correct decisions [53].

- *11-points average precision.* A text classification system can be tuned to balance between precision and recall. The *11-points*

average precision is computed by allowing the recall to take values 0.0, 0.1, ..., 1.0, and averaging the 11 corresponding precisions.

- AUC (*Area under curve*) is the area under the ROC curve, and is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

2.9.2 Three scenarios for comparing text classification algorithms

There are three main different scenarios where text classification algorithms (and data mining methods, in general) may be compared [15]:

- Two algorithms on a single problem
- Two algorithms on multiple problems
- Several algorithms on multiple problems

A frequently used technique for comparing two algorithms in a single domain, with solid foundations in statistics, is cross-validation . One round of cross validation consists on partitioning the dataset into k disjoint subsets, using $k-1$ subsets to induce a learner, and the remaining subset to measure the performance, using some of the evaluation measures presented in Subsection 2.9.1. This is repeated for each one of the k subsets, using the average of the obtained results. The main advantage of cross-validation is that all the instances are used for both training and evaluation. In order for cross-validation to provide meaningful results, the validation and training sets have to be extracted from the same population.

When we are interested in comparing the performance of two algorithms on multiple problems or datasets, a common statistical technique is the Wilcoxon test [54]. This statistical test can be

seen as a non-parametric alternative to the paired Student's t-test. Non-parametric tests are usually preferred in machine learning and data mining, as discussed in the work of J. Demšar [55], because of the lack of normality and homogeneity of the distributions of datasets. The Wilcoxon test works by ranking the performance of two algorithms on the datasets, and can be used to state whether an algorithm is significantly better that other over the used datasets.

Finally, when we are interested in comparing the performance of two algorithms on multiple problems or datasets, a common statistical technique is the Friedman test, which is a non-parametric equivalent of the two-way ANOVA [15]. The Friedman test is used to check if the rankings of the algorithms are significantly different. If the result of the test is positive (that is, if the null hypothesis is rejected), a post-hoc test such as the Nemenyi of Bonferroni-Dunn test [55] has to be used.

2.9.3 Benchmarks for text categorization

Finally, we cite some of the several publicly available datasets (or *benchmarks*) for experimental evaluation of text classification systems. Some of the most commonly used ones are the following:

- *Reuters* consists of a set of economy-related stories. It contains 21578 manually labelled Reuters news documents from 1987. There are 672 categories. The data was originally collected and labelled by Carnegie Group, Inc. and Reuters, Ltd. in the course of developing the CONSTRUE text categorization system [28]. In 1990, the documents were made available by Reuters and CGI for research purposes to the Information Retrieval Laboratory of the Computer and Information Science Department at the University of Massachusetts at Amherst. It has been a very often used dataset for text categorization. Nevertheless, there are up several different versions of this collection, which makes it difficult to use this dataset for comparison. In this thesis we have used the Reuters-21578 collection.

- *OHSUMED* contains titles and abstracts from medical journals [56]. The categories are terms of the MESH thesaurus.
- *20 Newsgroups*. Consists of messages posted to Usenet newsgroups. The categories are the newsgroup where each message has been posted [57].
- *RCV1* is an archive of over 800,000 manually categorized newswire stories recently made available in 2003 by Reuters, Ltd. for research purposes. The dataset is thoroughly described in the work of Lewis et al. [58].
- The *Enron* dataset contains emails from about 150 users of Enron, organized into folders. The corpus contains about 0.5M messages. This data was originally made public by the Federal Energy Regulatory Commission during a investigation.

Chapter 3

Applying Text Mining Techniques to DNA Strings

3.1 Introduction

Our objective in this Chapter is to show how text mining techniques can be applied to non-linguistic domains, specifically to DNA strings, by using appropriate feature selection procedures. Our base hypothesis is that DNA strings contain meaningful substrings or words, that is, there are especially informative subsequences within whole DNA strings. We aim to extract these words from DNA strings and use them to summarize DNA strings as collections of words, that is, as documents. This representation allows, for example, to provide a visual representation for DNA strings in the form of a word cloud, as well as to apply text mining techniques to DNA strings for tasks such as mitochondrial DNA haplogroup classification.

Representing DNA strings as documents (word collections) is an intuitive possibility, since DNA strings can be viewed as very long sequences of symbols. Extracting words from most natural languages

is generally a straightforward task, since words are separated by spaces and other punctuation signs that delimit them. This task is noticeably more difficult in languages such as Chinese or Japanese, where there is no sign to indicate word separation. In such cases, different algorithmic techniques have been used to identify meaningful units, of heuristic [59] or statistical [60] nature.

Extracting statistically meaningful substrings from DNA strings is a challenging task, given the length of the string and the low number of symbols that constitute the alphabet. It has been shown that the problem of finding frequent non-overlapping factors in a string has a general complexity of $O(n^2/k^2)$ [61], where n is the length of the string and k is the minimum frequency that a substring must have to be considered frequent. Efficient algorithms are of paramount importance to solve this task. In this Chapter we propose a parallel version of the SANSPOS algorithm, and use it as a basic tool to extract words from DNA strings.

The SANSPOS algorithm retrieves all the frequent factors present in a string. However, it has to be taken into account that not every factor is necessarily relevant enough to be considered a word. The reason is that, if a given factor is frequent, all its subfactors are going to be frequent too. To illustrate this, we can think in terms of natural language: if the string *process* is frequent in a document corpus, strings such as *proce*, *roc* or *ocess* are frequent too. But the only actual meaningful word is *process*. The other substrings are frequent only because they are contained in *process*. The same is true for DNA sequences: we are only interested in actual words, that is, in interesting substrings, not in substrings that are frequent only because they are included in a larger word. In order to be able to identify actual, meaningful words, we have to use interestingness functions, similar to the interestingness functions used in association rule learning [62].

In this Chapter we show how to represent DNA strings as documents, using relevant substrings of variable length as features, and two uses of this representation. The first one is the possibility of offering a concise visual representation of the DNA string, showing

the most interesting substrings. The second advantage is the possibility of applying well-tested text classification techniques to DNA strings, using the succinct word-based representation we propose.

This chapter is partially based on the following publication in which the author has participated:

- Baena-García, M.; Carmona-Cejudo, J.M.; Morales-Bueno, R.: "String analysis by sliding positioning strategy", *Journal of Computer and System Sciences* Available online 19 March 2013, ISSN 0022-0000, 10.1016/j.jcss.2013.03.004

The rest of the Chapter is organized as follows. In Section 3.2 we put this Chapter in the context of bioinformatics and data mining. In Section 3.3 we review the related work. Then, in Section 3.4 we explain our parallel SANSPOS algorithm for frequent substring extraction. In Section 3.5 we explain how to use this algorithm to extract features and represent documents as words. Finally, in Subsection 3.6 we apply our proposal to mitochondrial DNA haplogroup classification.

3.2 Background

Bioinformatics is an interdisciplinary field that provides automated methods for analyzing biological data. A major research area within bioinformatics is biological sequence analysis, including sequence storage and compression [63], sequence search [64], sequence visualization [65], sequence alignment [66] or sequence classification [67]. The key challenge in this area is the huge size of DNA strings. Efficient algorithms able to deal with this are of paramount importance.

Bioinformatics involve technologies such as artificial intelligence, soft computing and data mining.

In particular, an important topic is *DNA analysis*. DNA stands for *deoxyribonucleic acid*. It is the molecule that encodes the genetic instructions that describe the development and functioning of living organisms [68]. DNA contains four different types of nucleotides: guanine, adenine, thymine and cytosine. These nucleotides are generally abbreviated as G, A, T, C, and can be thought of as the four symbols that compose the alphabet of DNA. DNA is organized as a double helix with two complementary strands. Each nucleotide in one strand is paired with another nucleotide from the other strand. This configuration is known as *base pairs* [68]. Guanine is always paired to cytosine, and adenine to thymine.

DNA is organized in chromosomes, and the set of chromosomes present in a cell compose its genome. The size of the genome is different in different species. For example, the human gnome has about 3 billion base pairs of DNA arranged into 46 chromosomes. DNA is converted to proteins in two steps. The first one (or *transcription*) converts DNA to messenger RNA. In the second step (*translation*), proteins are obtained. There are specific stretches of DNA, the *genes*, corresponding to a unit of inheritance and associated with some specific function. Genes influence the *phenotype* of an organism, that is, the organism's observable characteristics. The complete set of genes is called the *genotype* of the organism.

Mitochondrial DNA (or *mtDNA*) is the name given to the portion of DNA located in mitochondria within eukaryotic cells. In human beings, mtDNA is the smallest chromosome. It encodes 37 genes and contains around 16600 base pairs. It has the peculiarity that, in most species, it is inherited from the mother only. The comparison of different mtDNA sequences allows to study the evolutionary relationships between species and populations. Differences in human mtDNA sequences conform *haplogroups*, which can be organized according to their genealogical relationship, conforming a *phylogenetic tree*.

Haplogroups are basically genetic population groups, and they are composed of people who share a common ancestor on their maternal lineage.

The field from data mining that aims to find relevant information in symbolic sequences is known as *sequence mining*, and has a direct application to DNA strings, given that they can be represented as very long sequences of symbols. Pattern discovery in strings can be classified as a special case of pattern discovery in databases [61]. The main peculiarity of pattern discovery in strings is that we have a big string over a small set of symbols, instead of a database of small transactions over a big set of items. Therefore, there is a high number of patterns to analyze. The problem of finding frequent patterns in strings has been studied extensively in the literature (see Subsection 3.3)

3.3 Related work

As previously stated, in this Chapter we apply data mining to mitochondrial DNA haplogroup classification. Wong et al. [67] present a comparison of different data mining algorithms for classifying mtDNA sequences into their haplogroups. More specifically, they use random forests with principal component analysis for feature selection, nearest neighbours and support vector machines with RBF kernel. The conclude that SVM outperforms the other algorithms for this task. They use the dataset that was published as a result of the Genographic Project [69], which means that they do not work with raw DNA sequences, but with a set of predefined features, provided by experts and related to some specific coding regions. They found that class imbalance was a relevant problem. Less frequent haplogroups were more difficult to classify correctly. MtDNA haplogroup classification is also discussed in the work of Kloss-Brandstätter et al. [70], where they examine polymorphisms and use them as features. HaploGrep goes through all polymorphisms within a given sequence range from the human mtDNA haplogroup tree starting at a reference sequence. For every possible

haplogroup and sample, a rank is calculated according to the likelihood that the sample belongs to the haplogroup.

These approaches have in common that they use predefined features provided by biologists. In contrast, our proposal does not rely on any specific features. We use text mining techniques in order to automatically find interesting features. Text mining is one of the tools that aid bioinformatics researchers to cope with the information overload associated with DNA strings, although the main field of application until now has been analysis of biomedical literature [71, 72]. In contrast, we propose to apply these techniques to DNA strings directly. Some authors have used text mining techniques for DNA classification, using whole nucleotide sequences. A notable example is the work of Gadia and Rosen [8], which compute the frequency of all N-grams and then select the most relevant N-grams using the TF-IDF weighting function in order to classify genomic fragments. A distance-based classifier was used in their work. They experiment with different N-gram sizes, finding that with $N > 9$ there is a dramatic performance increase. They hypothesize that this is due to the increased probability of N-gram uniqueness.

The work of Gadia and Rosen has the disadvantage that a fix N-gram size has to be chosen beforehand, and the counts for all the different N-grams have to be kept. In contrast, string mining algorithms do not require to fix a size beforehand, and look for frequent substrings regardless of their lengths (although length constraints can be posed).

A popular tool that can solve the problem of finding frequent factors are suffix trees [73, 74], data structures that represent the suffixes of a given string. The use of suffix trees for locating frequent factors was first proposed in the work of Sagot [75]. Arguably, the most influential algorithm for constructing suffix trees from strings is Ukkonen's algorithm [74], which uses linear time for constructing suffix trees. Parallel construction of suffix trees has been recently explored in the work of Mansour et al. [76]. Their proposed ERa algorithm optimizes dynamically the use of memory and amortizes the I/O cost. A suffix tree-based genetic programming algorithm

for frequent pattern discovery was proposed by Sætrom [77]. De Rooij [78, 79] also uses Ukkonen algorithm to construct suffix tress, adding some contextual information to the tree nodes.

There is a problem with suffix tree usage for discovering interesting patterns: only the frequent part of the suffix tree will be used. That is, there are unnecessary nodes in the structure. A different data structure for finding frequent patterns are tries [80], ordered tree structures where each edge represents a symbol and each node is associated with a given pattern. In order to obtain the pattern corresponding to the n-th node, the symbols in the path from the root to n are concatenated. Bodon, in his survey of frequent itemset mining [81], defines two ways to implement tries: *compact* and *non-compact* representation. The compact representation uses an array of pointers to node structures. This has high space requirements, which makes the structure inefficient. The non-compact representation uses a linked list or binary search tree to link the sons of a node. Its disadvantage is that it has a higher search cost.

Vilo defines SPEXS [82], a new algorithm that avoid the problems associated with suffix trees. This algorithm generates a pattern trie with information summaries about the occurrences of each pattern. Nodes of frequent patterns are expanded incrementally to incorporate new patterns. This guarantees that only frequent patterns are constructed. SPEXS does not add every word to the tree but instead builds it on all possible words in breadth-first order. This algorithm has $O(n^2)$ time complexity, but can outperform simple solutions empirically [82]. Nevertheless, it has some limitations processing short tandem repeats.

The SANSPOS algorithm [61] is an Apriori-like algorithm that does a multiple-pass candidate generation and test approach. The algorithm uses the SP-Trie data structure and Positioning Matrices as positioning strategy, which allows to apply prune heuristics with low computational cost and calculate interestingness measures. A description of SANSPOS is provided in Section 3.4.

Regarding DNA visualization, several different approaches can be found in the literature. Some authors propose to view DNA sequences as paths in a three-dimensional space [83]. This representation has been used to compare the genome of human beings and chimpanzees. Another approach is to use spectral analysis [84]. Spectral analysis makes it easier to discover regular patterns in DNA strings. In this line, Berger et al. [85] also propose to apply frequency domain transformations to visualize and analyze DNA sequences. They propose to track the evolution of DNA sequence as a language. Artemis [65] is a popular Java free software for DNA visualization and annotation. It allows visualisation of sequence features and the results of analyses within the context of the sequence. There are some other representation that focus on specific features of DNA sequences. An example are GC-skew graphics. GC-Skew is calculated in a sliding windows as (G - C) / (G + C), where G is the number of G's in the window, and C is the number of C's. GC-Skew graphs are useful to analyze DNA replication. There are also visualization techniques, such as PIP plots [86], specifically designed for showing the differences between DNA strings.

3.4 Parallel SANSPOS algorithm

SANSPOS (String ANalysis by Sliding POsitioning Strategy), first introduced in the work of Baena-García et al. [61], is an algorithm to find frequent patterns in strings by constructing a data structure called *SP-Trie*. This algorithm is based on multiple passes over the input string. In every pass, the algorithm generates and tests new and longer patterns, using the fact that any super-pattern of a non-frequent pattern cannot be frequent. In this sense, the SANSPOS approach bears relation to the APriori algorithm for association rule learning. To reduce the disadvantage of iterations, an iterative approach that performs multiple-level expansions of a trie in every iteration is used. In SANSPOS, the number of levels to expand grows exponentially in every iteration over the string. Hence, the number of iterations to perform is a logarithm of the trie's maximum depth.

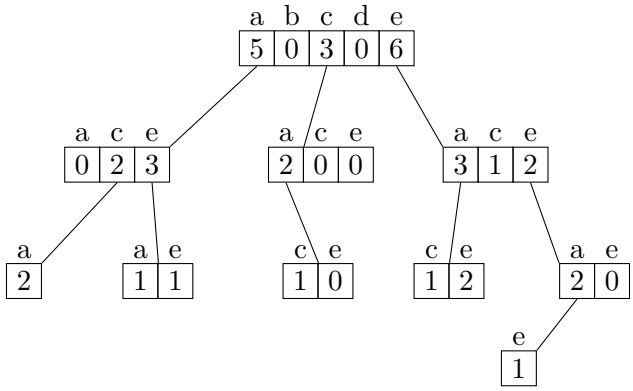


FIGURE 3.1: Trie with compressed representation for the string "eeaeaeeacacaec". The number in a node represents support.

The aforementioned SP-Trie structure is a variation of a trie structure that represents all frequent factors of the input string. This structure has a node organization and routing strategy that uses auxiliary structures called *Positioning Matrices*, which allow to locate patterns in the trie efficiently. The Positioning Matrices allow SANSPOS to efficiently detect short tandem repeats (STRs), that is, patterns that are repeated a high number of times at adjacent positions. This capability turns out to be of paramount importance: as we have previously stated, the space complexity of discovering frequent factors is quadratic in general. Nevertheless, it can be reduced to linear by detecting STRs [87].

In Section 3.3 we have mentioned that two basic ways to implement tries are compact and non-compact representations. There is a third option, which is used by the SP-Trie data structure: the *compressed* representation. This representation uses arrays as children of a node. An example of a trie with compressed representation is depicted in Figure 3.1.

Along with SP-Tries, another important aspect of the SANSPOS algorithm are Positioning Matrices. Let p be a pattern, the *Positioning Matrix* C of a string p is an upper triangular matrix of nodes, where each position of C (c_{ij}) represents the node in the SP-Trie corresponding to the substring $p_i \dots p_j$. This is the value

of $pos(p_i \dots p_j)$. This matrix can be used to locate the position of a pattern within a SP-Trie [61]. The Positioning Matrices can be regarded as a dynamic programming strategy to locate nodes. An important advantage of Positioning Matrices is that they can be reused to locate nodes of two contiguous overlapped patterns by diagonally shifting the matrix. Positioning Matrices allow to apply interestingness rules and to detect STRs.

The initialization procedure of the SANSPOS algorithm creates a SP-Trie structure which contains the frequency of every pattern of length 1 (that is, the support of each symbol σ in the alphabet Σ). For each character in the input sting, its frequency has to be updated in the SP-Trie, using Positioning Matrices to locate their position. Once the trie has been initialized, the input string is transversed several times, analyzing exponentially longer patterns each time. In each iteration, there is a sliding window of a given length which sequentially checks and updates all the patterns of this same length. For each analyzed pattern, we can reuse the indexes calculated for the previous position of the Positioning Matrix. Short tandem repeats (STRs) can be pruned at this stage, as well as factors which frequency is lower than a given threshold (recall that a superfactor of a non-frequent factor cannot be frequent). The objective is to obtain a compressed SP-Trie at the end of each iteration. The output of the SANSPOS algorithm is a SP-Trie structure which contains the frequencies of all the frequent patterns in the input string.

DNA strings are typically very long, something that we can exploit in order to parallelize the process of transversing them. More specifically, if we have an input string of length l, and want to use t different threads to concurrently update the shared SP-Trie structure, we divide the input string into t substrings. The patterns checked in the borders of the substrings are overlapped, so that all the possible patterns are transversed exactly once.

In order to synchronize write access to the shared SP-Trie structure, spinlocks have been used. Spinlocks are basic primitives for locking

access to the mutual exclusion zone, in our case, writes into the SP-Trie, and are available in the Linux Kernel from version 2.2. Spinlocks have been used instead of other synchronization mechanisms because, in our case, context switching would be more expensive than waiting a few CPU cycles, since the threads we use are likely to be blocked for only a short period of time .

3.4.1 Parallel SANSPOS: experimental evaluation

In this section we evaluate the speed-up resulting from using the parallel version of the SANSPOS algorithm. More specifically, we compare the effect of using different numbers of threads. We have tested a parallel version of SANSPOS in a HP 9000 Superdome server with 128 cores and 128 GB of memory. In table 3.1 we show results of speed-up, time average, and efficiency. The speed-up is calculated as the ratio of the time needed by the parallel and the sequential algorithm. It can be calculated as shown in Eq. 3.1, where T_{seq} is the time needed by the sequential algorithm, while T_t is the time needed by the parallel version with t threads.

$$speedup_t = \frac{T_{seq}}{T_t} \tag{3.1}$$

Efficiency is the ratio of the actual speed-up and the maximum possible (ideal) speed-up (that is, the time needed by the sequential algorithm divided by the number of threads), as shown in Eq. 3.2.

$$efficiency_t = \frac{speedup_t}{t} = \frac{T_{seq}}{tT_t} \tag{3.2}$$

We have used the English version of *The Ingenious Gentleman Don Quixote de la Mancha* by Miguel de Cervantes, as downloaded from Project Gutenberg (`http://www.gutenberg.org/ebooks/996`) for evaluation. For this string, the optimum number of threads is found to be 33, as shown in the results of the experiment in Table 3.1. It can be observed that, until this value, the speed-up increases with

the number of threads, but, for larger number of threads, the overhead caused by managing the high number of threads causes the speedup to drop down. This is due to the workload or each thread being too small in comparison with the number of synchronizations. This is graphically depicted in Figure 3.2. From this, we can conclude that the number of threads has to be optimized so that the workload for each one is big enough.

Threads	Time (s)	Speedup	Efficiency
1	58.63	1.00	1.00
2	33.46	1.75	0.88
4	20.48	2.86	0.72
8	13.79	4.25	0.53
17	8.64	6.79	0.40
30	7.17	8.18	0.27
33	**6.94**	**8.45**	**0.26**
39	7.23	8.11	0.21
54	8.39	6.99	0.13
79	10.65	5.51	0.07

TABLE 3.1: Efficiency of parallel version of SANSPOS (time in seconds)

3.5 Using Parallel SANSPOS for DNA feature selection

In this section we explain our procedure to extract words from DNA strings, using the parallel version of the SANSPOS algorithm to retrieve all the frequent factors, and the Added Value (AV) measure to select the most meaningful factors (the words). The result of the procedure is the representation of DNA strings as collection of words associated with a given frequency.

This procedure can be directly applied to DNA string visualization using a word cloud, where the size of each word corresponds to its frequency. This graphical tool can also be used to visually compare

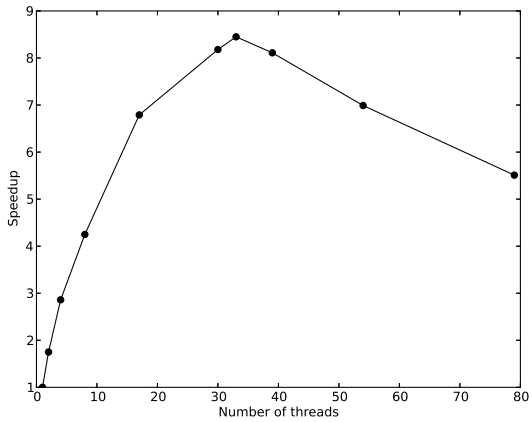


FIGURE 3.2: Speedup of the parallel version of SANSPOS in string #2

different DNA strings. A further use of this document-like representation of DNA is the application of text classification techniques to DNA.

An important parameter that we have to set for the SANSPOS algorithm is the minimum word length. We have found a value between 6 and 10 to be suitable: there are too many frequent substrings with a length lower than 6, and too few with a bigger length.

After its execution, the SANSPOS algorithm returns all the frequent factors present in the DNA string. But recall that, if a given factor is

frequent, all its substrings will be frequent too. For this reason, it is necessary to select only factors that are considered to be interesting, and not their substrings. There are many different interestingness functions available. It has been shown that AV works acceptably well in the domains of DNA analysis and natural language [62]. The AV function is used for calculating association rules interestingness. Given an association rule $X \rightarrow Y$, its AV can be calculated as follows:

$$AV(X \rightarrow Y) = P(Y|X) - P(Y) \tag{3.3}$$

where X and Y are the antecedent and consequence of the association rule, and $P(X)$ is the probability of observing X in a given dataset. This concept can be applied to a given string W if we divide W into two adjacent substrings: $W = W_1W_2$.

More specifically, we use Algorithm 1 to transform a DNA string into a word-based representation:

Algorithm 1 DNA String to document

Require: A DNA nucleotide sequence σ as input
Ensure: A multiset of relevant words found in σ (that is, a document-like representation), in the form of a frequency mp
  $SPTrie \leftarrow Parallel\_SANSPOS(\sigma)$
  $taboos \leftarrow \emptyset$
  $words\_with\_frequency \leftarrow list\_words(SPTrie)$
  for $word \in extract\_unique\_words(words\_with\_frequency)$ **do**
    $taboos\_this\_word \leftarrow \emptyset$
    for $i \in [1, \ldots, |word|]$ **do**
      $X \leftarrow word_{1\ldots i}$
      $Y \leftarrow word_{i+1\ldots|word|}$
      $av\_prefix \leftarrow calculate\_AV(X, Y, word, words\_with\_frequency)$
      $av\_suffix \leftarrow calculate\_AV(Y, X, word, words\_with\_frequency)$
      if $av\_prefix < AV\_THRESHOLD$ **then**
        $taboos\_this\_word \leftarrow taboos\_this\_word \cup \{X\}$
      end if
      if $av\_suffix < AV\_THRESHOLD$ **then**
        $taboos\_this\_word \leftarrow taboos\_this\_word \cup \{Y\}$
      end if
    end for
    $taboos \leftarrow taboos \cup taboos\_this\_word$
  end for
  $final\_frequency\_map \leftarrow \{(w, f) \in word\_frequencies | w \notin taboos\}$
  return $final\_frequency\_map$

In Algorithm 1 we use the following functions:

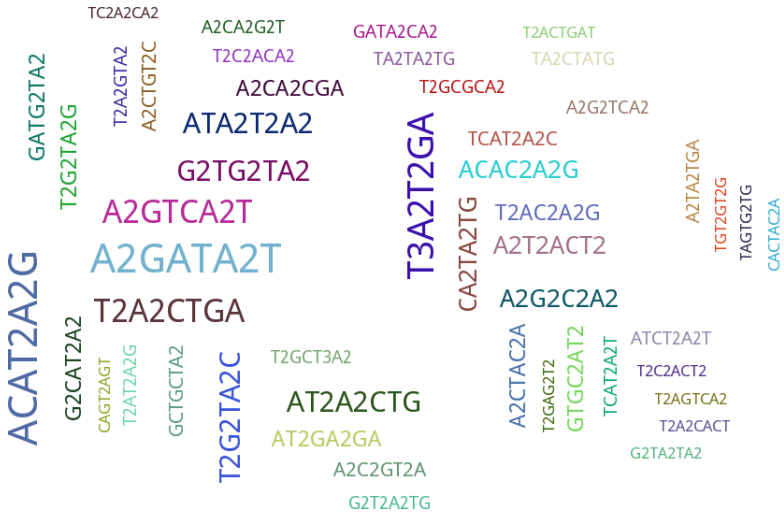
- $Parallel\_SANSPOS$ algorithm, that returns an SP-Trie structure corresponding to an input sequence σ,
- $list\_words$, which takes an SP-Trie as input and returns a list of pairs of the form (w, f), where w is a substring present in σ and f is its associated frequency,
- $extract\_unique\_words$, which takes a list of pairs (w, f) and returns the list of different words, and
- $calculate\_AV(X_1, X_2, X, words\_with\_frequency)$, which calculates the AV value of the rule $X_1 \to X_2$ given X, using the frequencies provided by $words\_with\_frequency$ to calculate the needed probabilities.

After this, we have a list of the interesting frequent factors present in the original DNA string, as well as their frequency. That is, we have a representation of the DNA string consisting of a multiset of words.

The first application is the graphical representation of the words contained in a DNA string. For this, we propose to use a word cloud, where words are represented used different sizes, according to their frequency. See Chapter 7 for a discussion of word clouds to represent documents. We have used the a python library, PyTagCloud (`https://github.com/atizo/PyTagCloud`) to produce the clouds. In Figure 3.3 we show the word clouds corresponding to the genomes of two different kinds of bacteria: M. Genitalium and M. Pnaumoniae, identified by the NBCI accession numbers NC_000908.2 and NC_000912.1, respectively. Mycoplasma Genitalium is a small parasitic bacterium that lives on the ciliated epithelial cells of the primate genital and respiratory tracts, while Mycoplasma Pneumoniae causes a form of atypical bacterial pneumonia.

The aforementioned word clouds show the most frequent words in the genomes of each one of these organisms. We can also use word clouds to underline differences between organisms. For example Figure 3.4 depicts the words that differentiate one genome from the other one, using bigger labels for strings that are very frequent in

(a) M. Genitalium



(b) M. Pneumoniae

FIGURE 3.3: Word clouds corresponding to M. Genitalium and M. Pneumoniae. For the sake of visualization, the nucleotide substrings are represented by a compressed sequence of symbols (A, C, G or T) and numbers, which represent the number of times the last nucleotide is repeated. For example, A2GT3 corresponds to AAGTTT

one gnome and not in the other. This gives us a graphic representation of the differences between two given DNA sequences.

3.6 Application to mtDNA haplogroup classification

In this Section we apply text mining techniques to mtDNA haplogroup classification, using the document-like representation of DNA strings discussed in the previous sections. We use a dataset of 1306 human mitochondrial DNA from the FamilyTreeDNA project (`http://www.familytreedna.com/`). Each sample is labelled with its correct haplogroup, provided by human experts. We transform each string into a word-based document, using the previously proposed methodology. Using this representation, we apply text mining techniques to classify each sample into its corresponding haplogroup.

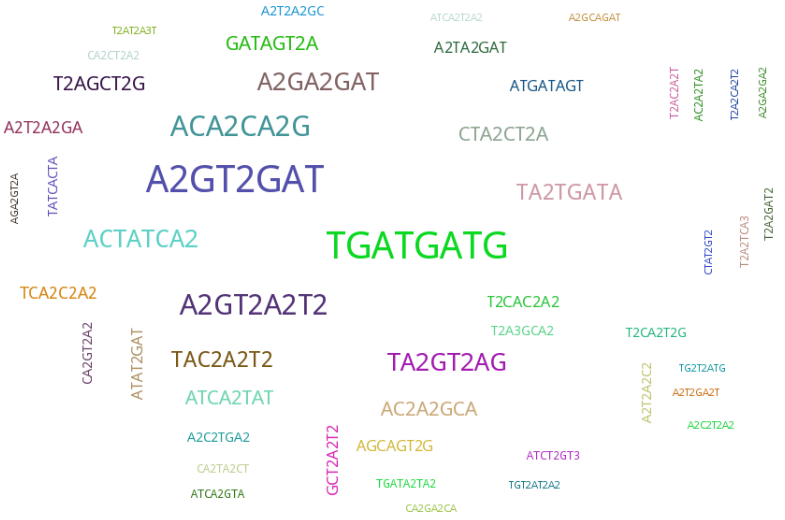
This is a difficult problem for data mining algorithms, due to two main reasons:

- It is a multiclass problem, with a high number of labels (14)
- The classes are unbalanced: there are haplogroups that correspond to many samples, while some other haplogroups are scarcer.

There are some haplogroups that are represented by one or two samples only. We have ignored them, since the number of samples in that haplogroup is too low to allow learning.

We have used SVM with linear kernel as classifier, and a 5-fold cross validation to evaluate its performance. For each of the folds, we use the training set to select the attributes (that is, the words that appear in the test set, but not in the training set, are not used).

The results in terms of precision, recall and F1 by haplogroup are shown in Table 3.2. As expected, the performance varies according



(a) M.Genitalium vs. M.Pnaumoniae

(b) M.Pnaumoniae vs M.Genitalium

FIGURE 3.4: Word clouds corresponding to the difference between the genomes of M.Genitalium and M.Pnaumoniae, and vice versa. Bigger labels correspond to words that are frequent in one genome scarce in the other one. The bigger the label, the bigger the difference. The nucleotide strings are represented using the same compressed representation as in Figure 3.3

Haplogroup	**Number of samples**	**Precision**	**Recall**	**F1**
A	16	0.812	0.765	0.788
C	12	0.917	0.846	0.880
H	412	0.964	0.990	0.977
HV	35	0.829	0.707	0.763
I	22	0.864	0.826	0.845
J	92	1.000	0.989	0.995
K	207	0.981	0.990	0.986
L2	8	1.000	0.727	0.842
N1	6	1.000	0.667	0.800
T	158	1.000	0.994	0.997
U	213	0.953	0.976	0.964
V	20	0.950	0.864	0.905
W	43	0.953	0.932	0.942
X	25	0.880	0.957	0.917

TABLE 3.2: Accuracy: 0.962, Micro F1: 0.963

to the specific haplogroup. For example, it works better for haplogroup J (F1 = 0.995) than for HV (F1 = 0.763). The accuracy of the classifier is 0.962, and the micro-F1 0.963. We depict the confusion matrix of the classifier in Figure 3.5.

Our results are similar to the results found in the literature, such as the work of Wong et al. [67]. Nevertheless, they use a different dataset that does not contain whole sequences, but predefined features, so we cannot directly compare our results to theirs.

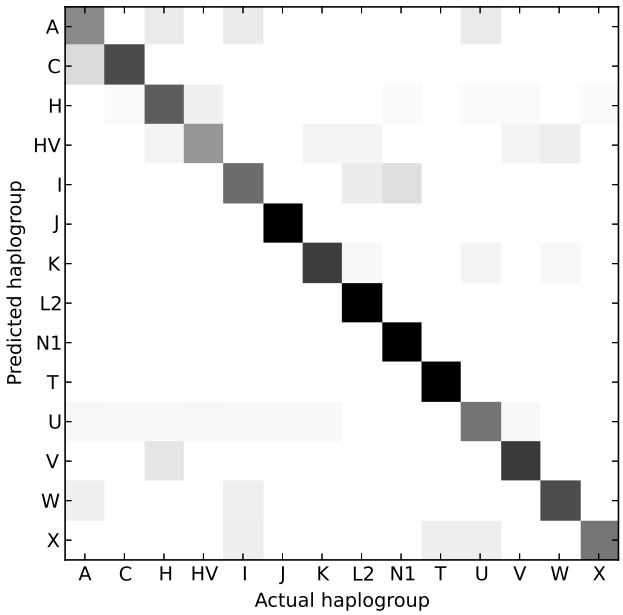


FIGURE 3.5: Confusion matrix for the mitochondrial DNA classification. Darker colours represent higher values

Chapter 4

Multilabel Text Mining

4.1 Introduction

Traditionally, classification tasks deal with *single-label* datasets, where every example is associated with a single label λ from a set of disjoint labels L. Nevertheless, *multilabel* datasets are emerging and gaining interest due to their increasing application to real problems [88]. Multilabel datasets emerge when the examples are associated with a set of labels $Y \subseteq L$, as occurs with image annotation, genomics and our main topic, natural text.

When classifying documents written in natural language, it is often too restrictive to assign only one category to each document. For example, a newspaper article about the reaction of the Catholic church authorities to a new abortion law could be classified under *Politics*, *Religion* or even *Healthcare*, since readers interested in politics, religion or healthcare are possibly going to be interested in this article. For that reason, when dealing with text documents in natural language, multilabel learning has to be considered.

This chapter is based on the following publication in which the author has participated:

- Carmona-Cejudo, J.M., Baena-García, M., del Campo-Ávila, J., Morales-Bueno, R.: Feature extraction for multi-label learning in the domain of email classification. *CIDM* 2011: 30-36

The rest of this chapter is organized as follows. In Section 4.2 we introduce the multilabel learning problem and present related tasks, methods and evaluation measures used in this field. Then, in Sections 4.3 and 4.4 we apply multilabel learning to email classification, comparing the results obtained by different algorithms comparing different metrics.

4.2 Multi-label learning

The continuously growing number of applications of multilabel learning has recently attracted an increasing interest from researchers. Categorisation of textual data is probably the most common application tackled by multilabel learning, and an important volume or multilabel learning work is devoted to text classification.

For instance, McCallum et al. [89] employ a Bayesian model based on mixtures for multilabel text classification. Schapiro and Singer [49] propose two extensions to the the Ada-Boost boosting algorithm in order for this ensemble method to be applicable to multiclass and multilabel contexts, using the Reuters dataset for the evaluation. Ueda and Saito [90] present a multilabel learning algorithm which uses parametric mixture models in opposition to the binary classification approach, applying the resulting algorithm to Web sites categorization. Rousu et al. [91] present an optimized kernel-based algorithm for hierarchical multilabel text classification, again using the Reuters dataset as well as the patents database WIPO-alpha for evaluation.

Apart from text categorization, multilabel learning has been applied to some other domains. For example, in the work of Boutell et al. [92] it is applied to semantic annotation of images, while Qi and al. use multilabel learning for video tagging [93]. Other applications include genomics [94] or music classification [95]. Nevertheless, such domains are out of the scope of this thesis.

4.2.1 Learning: tasks and methods

In the literature on multilabel, learning two different, albeit related tasks can be found:

- *Multilabel classification* (MLC) is devoted to finding a subset of labels to be associated with a given example. That is, for each dataset instance, the label set is divided into two disjoint sets: labels that are relevant, and labels that are not.
- *Label ranking* (LR) consists in finding an ordering of the set of all the labels according to their relation with each example. That is, the most relevant tags will be first in the list, while labels that bear no relation to the instance will be last.

There is a third task, called *multilabel ranking* (MLR), which combines of MLC and LR. That is, it finds an ordering of the labels and additionally finds a bipartition between relevant and irrelevant labels. As a consequence, the methods that solve this new task can be used for both classification and ranking, if they are conveniently adapted. The most straightforward adaptation consists in using a given threshold, so that the labels above that threshold can be included in a *relevant* subset and the other ones in a *irrelevant* subset, obtaining a bipartition.

This chapter is mainly devoted to the MLC task. There are different methods that can be used for this learning task, a categorization of which is depicted in Figure 4.1. Roughly speaking, these methods can be divided into two disjoint categories [88]:

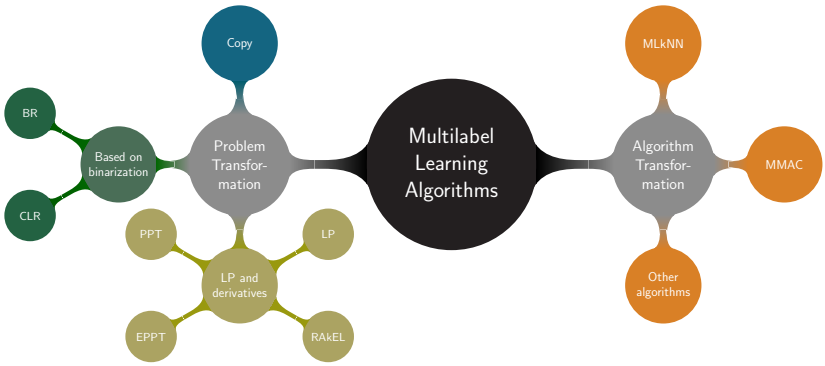


FIGURE 4.1: Classification of multilabel learning algorithms

Instance	**Labels**
i_0	b, c
i_1	a, b
i_2	a, c
i_3	$a,$
i_4	a, b

TABLE 4.1: Example multilabel dataset D_{ex}

- *Problem transformation* methods transform the original multilabel dataset into one or several single-label datasets, and apply standard machine learning algorithms. The results of the single-label problems are combined in order to obtain the multilabel result.
- *Algorithm adaptation* methods extend standard algorithms in order to be able to cope with multilabel datasets directly.

The main advantage of problem transformation methods reside in their flexibility, given that they allow to use any existing single-label machine learning algorithm. In order to illustrate the application of these methods, we use an example multilabel dataset, depicted in Table 4.1. Some of the most common problem transformation methods are the following [88]:

- *Copy* or *copy-weight transformation*: each multilabel example is copied as many times as different labels are associated to

Instance	**Label**
i_0	b
i_0	c
i_1	a
i_1	b
i_2	a
i_2	c
i_3	a
i_4	a
i_4	b

TABLE 4.2: Dataset resulting from applying the *copy* transformation to the example dataset D_{ex}

(a) Label a

Instance	**Label**
i_0	False
i_1	True
i_2	True
i_3	True
i_4	True

(b) Label b

Instance	**Label**
i_0	True
i_1	True
i_2	False
i_3	False
i_4	True

(c) Label c

Instance	**Label**
i_0	True
i_1	False
i_2	True
i_3	False
i_4	False

TABLE 4.3: Datasets resulting after applying the *Binary Relevance* transformation to the example dataset D_{ex}

such example. Weights can optionally be used. After applying this transformation to the multilabel dataset in Table 4.1 we obtain the single-label dataset shown in Table 4.2.

- *Binary Relevance* (BR): as many binary classifiers are learnt as there are labels ($D = |L|$). The original data are transformed into D new datasets that contain all examples of the original dataset labelled with a binary label, which is positive if the corresponding label is present, or false otherwise. This means that this method produces $|D|$ binary classifiers, each one of them responsible for learning a single label. The classifiers are independent, which means that label dependencies are not taken into account. If we use this method for processing the example multilabel dataset, we obtain three single-label datasets, depicted in Figure 4.3.

Instance	**Label**
i_0	bc
i_1	ab
i_2	ac
i_3	a
i_4	ab

TABLE 4.4: Dataset resulting from applying the *Label Powerset* transformation to the example dataset D_{ex}

- *Calibrated Label Ranking* (CLR) [49]: CLR, as BR, is based on constructing binary datasets. But, unlike BR, a classifier is produced for every pair of labels. This algorithm faces complexity problems when the number of different classes grows. CLR uses a virtual label, which is used as a breaking point between relevant and irrelevant labels.

- *Label Powerset* (LP): if this method is used, every set of labels that is assigned to an example is considered as a new class of a single-label classification problem. Thus, there will be as many new single-labels as different sets of labels appear in the dataset. This method models label dependence explicitly. The basic label powerset method faces computational complexity problems, since the number of different labels can be very high. The number of possible combinations is bounded by $min(m, 2^{|L|})$ (where m is the number of examples and L the set of different labels). Moreover, this method is prone to overfitting. See Table 4.4 for the application of this method to the original example dataset.

- *Pruned Problem Transformation* (PPT) [96]: it is an extension of *LP* that attempts to deal with the aforementioned problems. PPT breaks the original sets of labels into smaller and more frequently occurring subsets. Therefore, it reintroduces new examples in the dataset and learns different labels (smaller and more common). Using this approach, an ensemble variation can be built by combining different PPT models: the Ensemble Pruned Problem Transformation (EPPT).

- *Random k-Labelsets* (RAkEL) [97]: this method is another derivative of Label Powerset. In this method, the original dataset is sampled and different small subsets are created. Each new subset is used in the training phase to induce different LP classifiers that will be combined to form an ensemble. This ensemble will rank the labels for a concrete observation (example without labels) and later a bipartition of the labelset could be carried out by selecting a threshold.

LP and its derivatives (RAkEL, PPT and EPPT) have the advantage of overcoming the *label independence* assumption, and are thus able to take advantages from of the relationships between labels. Such transformations can model the fact that some labels are more likely to occur in combination with others. The original LP transformation had the problem of having to take into account an overwhelming number of rarely occurring label sets. RAkEL, PPT and EPPT were designed in order to alleviate this disadvantage using pruning and meta-learning.

We briefly refer to some remarkable algorithm transformation approaches. The C4.5 has been adapted in [6], modifying the formula used to calculate entropy, so that it is able to cope with multilabel scenarios. More specifically, the formula of entropy is modified as follows:

$$Entropy(D) = \sum_{j=1}^{q} (p(\lambda_i) log p(\lambda_j) + q(\lambda_j) log q(\lambda_j)) \tag{4.1}$$

where $p(\lambda_j)$ is the relative frequency of class λ_j, and $q(\lambda_j) = 1 - p(\lambda_j)$. Boosting models have also been adapted for multilabel data, driving the learning process by optimization of different multilabel measures [49]. AdaBoost.MH is designed to minimize Hamming loss, while AdaBoost.MR is designed to place the right labels in the top of the ranking. Probabilistic generative models are proposed in the works of McCallum [98], Ueda and Saito [90], or Daelemas et al. [99]. Neural network variations have been proposed in a paper by

Zhang and Zhou [100], where BP-MLL, an adaptation of the back-propagation algorithm for multilabel learning, was presented, as well as in the work of Cramer and Singer [101], who proposed the MMP model (multiclass multilabel perceptron). There are also methods based on k Nearest Neighbours, such as ML-kNN. An algorithm based on association rules is MMAC [102]. See also the work of Veloso et al. [103], where a lazy multilabel approach is presented.

4.2.2 Evaluation measures for multilabel learning

There is a large number of measures that can be used to evaluate the performance of multilabel methods. In general, they cannot be simultaneously optimized for the same dataset and algorithm, since they measure different aspects of the performance. In fact, minimization of a given metric can result in a high regret for another metric, as pointed out by Dembczyński et al. [104].

Two main classes of evaluation measures exist, which correspond to the two main tasks previously mentioned: multilabel classification (MLC) and label ranking (LR). We have thus measures to assess if the bipartition between relevant and irrelevant labels is correct, and measures to evaluate the appropriateness of the calculated ranking.

In addition to the previous categorisation of evaluation measures, a different distinction can be noted for multilabel classification measures: *example-based* and *label-based* measures. The former calculates the metrics based on all the examples in the evaluation data, while the latter decomposes the calculation into separate evaluations for each label.

In the first group, *example-based*, several measures can be defined. The following notation is used: let N be the size of the dataset, D the number of labels, $\widehat{Y}^{(i)}$ the vector of predicted labels for the i-th example, $Y^{(i)}$ the set of actual labels for the same example, and Δ the symmetric difference operator. Some commonly used metrics are the following [88]:

Instance	**Actual labelset** $(Y^{(i)})$	**Predicted labelset** $(\widehat{Y}^{(i)})$
i_0	$\{a, c\}$	$\{a, d\}$
i_1	$\{b, d\}$	$\{b, d\}$
i_2	$\{a, d\}$	$\{a, d\}$
i_3	$\{b, c\}$	$\{b\}$
i_4	$\{a\}$	$\{a, d\}$

TABLE 4.5: Example dataset with actual and predicted labelsets (adapted from [105])

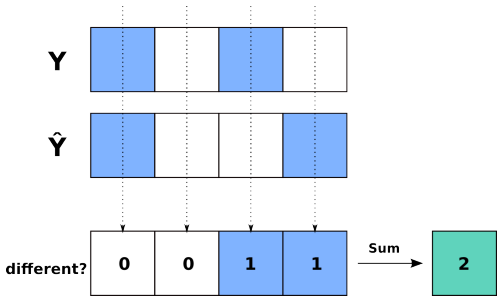


FIGURE 4.2: Graphical representation of Hamming Loss calculation for a given instance

- *Hamming Loss* is the average binary classification error. It can be calculated as:

$$Hamming\_Loss = \frac{1}{D}\frac{1}{N}\sum_{i=1}^{N}|\widehat{Y}^{(i)}\Delta Y^{(i)}|$$

  If the example in Table 4.5 is considered, the Hamming Loss is $\frac{1}{4}\frac{1}{5}(2 + 0 + 0 + 1 + 1)$. See Figure 4.2 for the graphical representation of this calculation for instance i_0.

- *Precision*, the average of predicted labels that are correct. The corresponding formula is:

$$Precision = \frac{1}{N}\sum_{i=1}^{N}\frac{|\widehat{Y}^{(i)} \cap Y^{(i)}|}{|\widehat{Y}^{(i)}|}$$

  For the example in Table 4.5, the precision is $\frac{1}{2}(\frac{1}{2} + \frac{2}{2} + \frac{2}{2} + \frac{1}{1} + \frac{1}{2})$. See Figure 4.3.

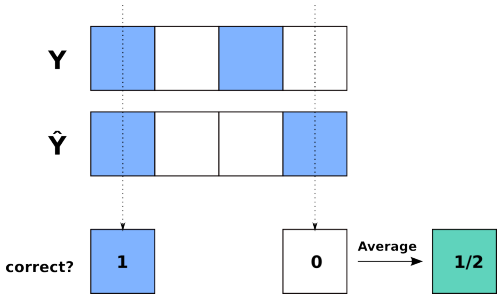


FIGURE 4.3: Graphical representation of precision calculation for a given instance

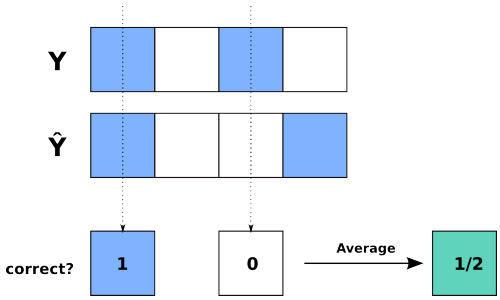


FIGURE 4.4: Graphical representation of Recall calculation for a given instance

- *Recall* is the average proportion of correct labels that were predicted. Its formula is:

$$Recall = \frac{1}{N}\sum_{i=1}^{N}\frac{|\widehat{Y}^{(i)} \cap Y^{(i)}|}{|Y^{(i)}|}$$

  For the example in Table 4.5, the recall is $\frac{1}{2}(\frac{1}{2}+\frac{2}{2}+\frac{2}{2}+\frac{1}{2}+\frac{1}{1})$. See Figure 4.4.

- The F_1 measure is the harmonic mean of precision and recall. Its formula is:

$$F_1 = \frac{1}{N}\sum_{i=1}^{N}\frac{2\cdot|\widehat{Y}^{(i)} \cap Y^{(i)}|}{|\widehat{Y}^{(i)}| + |Y^{(i)}|}$$

  The F_1 measure for Table 4.5 is $\frac{1}{5}2(\frac{1}{4}+\frac{2}{4}+\frac{2}{4}+\frac{1}{3}+\frac{1}{3})$. Precision and recall are somehow contradictory measures, and tuning

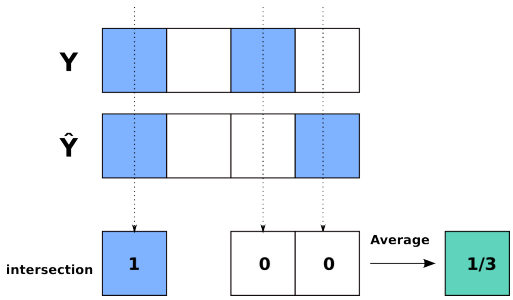


FIGURE 4.5: Graphical representation of accuracy calculation for a given instance

the parameters of a given learning algorithm can favour one against the other. The F_1 value is maximized when precision and recall have similar values.

- *Accuracy*, which can be calculated as follows:

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N}\frac{|\widehat{Y}^{(i)} \cap Y^{(i)}|}{|\widehat{Y}^{(i)} \cup Y^{(i)}|}$$

In the case of Table 4.5, the accuracy would be $\frac{1}{5}(\frac{1}{3}+\frac{2}{2}+\frac{2}{2}+\frac{1}{2}+\frac{1}{2})$. Accuracy is more restrictive than recall, since the large value that their denominator makes the measure more insensitive to variations in the number of correct decisions [53]. The accuracy calculation for instance i_0 is depicted in Figure 4.5.

In the second group, *label-based*, the different measures can be grouped into *macro-averaged* and *micro-averaged* [53]. We use the following notation: let tp_λ, fp_λ, tn_λ, fn_λ be the number of true positives, false positives, true negatives and false negatives; and let B be a binary evaluation measure (accuracy, precision, recall, etc.). The macro-averaged and micro-averaged measures are defined as:

$$B_{macro} = \frac{1}{D}\sum_{i=1}^{D} B(tp_\lambda, fp_\lambda, tn_\lambda, fn_\lambda)$$

$$B_{micro} = B\Big(\sum_{i=1}^{D} tp_\lambda, \sum_{i=1}^{D} fp_\lambda, \sum_{i=1}^{D} tn_\lambda, \sum_{i=1}^{D} fn_\lambda\Big)$$

The *micro* measures average values by giving the same weight to each instance, whereas their *macro* versions give the same weight to each instance, as we saw in Subsection 2.9.1.

4.3 Experimental evaluation

We have conducted an experimental evaluation with the objective of studying the effect of feature space reduction and different problem transformation methods in a specific dataset on different multilabel evaluation measures in the domain of multilabel email foldering. Specifically, our objectives are:

- To study whether reducing the feature space yields statistically significant improvements for different measures.
- To study which problem transformation methods provide better results for different evaluation measures for email classification.

We now describe the datasets and experimental setup of the mentioned evaluation.

4.3.1 Datasets

Two multilabel versions of the well-known ENRON dataset have been used for our experiments. The first one (which we will call *original dataset* from now on), comprises a selection of 1702 emails from the original ENRON dataset, labelled by hand using a set of 53 labels, to classify coarse genre topics (such as company strategy,

personal mail or logistic arrangements), included/forwarded information (newsletters, legal documents, and more), primary topics (regulations, internal projects, company image and more) and even emotional tone. Each of the examples contains 1054 attributes. This dataset can be obtained from the University of Berkeley [1].

The second one (*preprocessed dataset*), contains the same messages and labels as the first one, but with different features. Specifically, we use the ability of the GNUsmail framework (see Subsection 5.3.2) to create a dataset from raw email messages by extracting the 500 most relevant words, following the methodology described in [106]; that is, we have applied stemming to the words, and select 500 most relevant terms according to the *TF-IDF* metric. We have used this dataset to evaluate the impact of linguistic operators such as stemming, not used in the first dataset, which produce a dataset with less attributes, albeit more powerful ones. In order to produce this dataset, the following procedure has been applied to the raw messages:

- Deletion of stop words
- Stemming of the remaining words
- Weighting of the obtained terms, using the *TF-IDF* metric:

$$TF - IDF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} log \frac{|D|}{|\{d \in D : t_i \in d\}|}$$

  where $n_{i,j}$ is the number of occurrences of the term t_i in document d_j, and D the set of emails.

- Selection of 500 terms with the highest weight

Additionally, this preprocessed dataset also takes mail metadata into account. Specifically, the `From` and `To` fields of the email are binarized and included in the dataset. The less frequent senders and receivers are left out in order to further reduce the number of attributes. We have obtained 827 attributes in this version of the

[1] http://bailando.sims.berkeley.edu/enron_email.html

TABLE 4.6: Summary of datasets

Dataset	Number of attributes
Original dataset	1054
Preprocessed dataset	827

dataset. See table 4.6 for a summary of the datasets used in this study.

4.3.2 Experimental setup

In order to compare the performance of the distinct multilabel methods for the domain of email classification. We employ some of the metrics described in Subsection 4.2.2.

We have selected the following multilabel methods for our empirical evaluation:

- Binary Relevance (BR)
- Calibrated Label Ranking (CLR)
- Label Powerset (LP)
- RAkEL
- Pruned Problem Transformation (PPT) and
- Ensembles of Pruned Problem Transformation (EPPT)

We use the approach explained in the work of Katakis et al. [107] for parameter optimization.

Regarding internal classification methods, the following ones have been used:

- Support vector machines with linear kernel, using SMO, a Weka [108] implementation of support vector machines.

- Support vector machines with polynomial kernel. We have used LibSVM, a well-known implementation of support vector machines. Following [88], we have optimized the parameters for this dataset using the F_1 measure, finding that the best option was to use a polynomial kernel of degree $g = 3$, and the cost parameter $C = 1$.
- NNge, a nearest-neighbour-based algorithm, that was shown in a previous work [109] to perform well in this domain.
- J48, the Weka implementation of the well-known C4.5 algorithm for induction of decision trees.
- Naïve Bayes, a simple probabilistic classifier based on the Bayes theorem.
- IBk, a k-Nearest Neighbour classifier.

We are interested in classification performance. Furthermore, we are interested in the effect of the attribute selection described in subsection 4.3.1. We employ the Wilcoxon statistical test in order to study if the obtained differences are significant, using 30 paired 2-fold cross validations over each combination of dataset, internal algorithm and multilabel method. We have established the results obtained for the original dataset as the reference value, and labelled the differences in the following manner: $\oplus$ means that the result using the preprocessed dataset is significantly better, and $\ominus$ means that it is significantly worse.

We provide the scripts and datasets used in the experiments on the Web [2], so that the experiments are reproducible.

4.4 Results and discussion

Having described the experimental evaluation, in this Section we present the results and summarize the most relevant aspects of our study.

[2] http://code.google.com/p/gnusmail

In Table C.1 in Appendix C we show the results (median values) achieved by different combinations of algorithms and problem transformation methods attending to different measures. We examine if the preprocessed dataset favours the learning process. Some observations:

- For the performance measures, the results are usually better when using the preprocessed dataset. In addition, those differences are significant in most cases, as the statistical tests show. Therefore, we could say in a broad sense that the preprocessing methodology applied to the original dataset has provided a new framework that favours the learning task.
- The improvement observed can vary depending on the measure and the algorithm (and problem transformation method), being very small for some measures, like *Hamming Loss*, or bigger others, like *micro-averaged Recall*.

In Tables C.2 and C.3 in Appendix C we show the ranking produced by different combinations of algorithms and problem transformation methods. We search the best combination, and we also test if the preprocessing is appropriate for that combination. The ranking has been calculated independently for every measure and then all the measures have been combined to get an average ranking. It can be observed that the results achieved by SVM with polynomial (cubic) kernel are uniformly better than for any other algorithm. The best multilabel methods are RAkEL and CLR for both datasets.

Overall, the best combination is RAkEL over SVM with cubic kernel. This combination is especially beneficial for the Example Recall measure, where it obtains the best result for both datasets: 0.55 on average for the original and 0.65 for the processed dataset. The difference is even larger for the micro-averaged recall (0.52 for the original dataset and 0.63 for the processed one). This represents a remarkable difference between the datasets, and shows how a careful attribute selection can improve classification performance. On the other hand, if we consider the Hamming Loss and micro-averaged

	H.Loss.	Ex. Accuracy	Ex. Recall	Micro Precision	Micro Recall
H.Loss	1.000 ⊕	0.737 ⊕	-0.640 ⊕	0.895 ⊕	-0.625 ⊕
Ex. Accuracy	0.737 ⊕	1.000 ⊕	-0.036	0.772 ⊕	-0.042
Ex. Recall	-0.640 ⊕	-0.036	1.000 ⊕	-0.546 ⊕	0.993 ⊕
Micro Precision	0.895 ⊕	0.772 ⊕	-0.546 ⊕	1.000 ⊕	-0.526 ⊕
Micro Recall	-0.625 ⊕	-0.042	0.993 ⊕	-0.526 ⊕	1.000 ⊕

TABLE 4.7: Mutual correlations between Hamming loss, example-based accuracy, example-based recall, micro-averaged precision and micro-averaged recall. Statistically significant results are marked with ⊕

precision measures, we observe that RAkEL over SVM with cubic kernel works better for the original dataset. The difference is especially noticeable in the case of micro-averaged precision (0.61 for the original dataset and 0.55 for the preprocessed one).

Nevertheless, it must be underlined that this combination, despite working well for most measures, obtains a relatively poor performance for micro-averaged precision. If we select a combination that optimizes this measure, such as CLR over J48, we see that the results are still better for the preprocessed dataset than for the original one. The same applies for the Hamming Loss. If we consider the runner-up combination, CLR over SVM with cubic kernel, we see that the results are always better for the preprocessed dataset, except again in the case of micro-averaged precision, where no statistically significant difference can be found.

The results intuitively show that there are groups of measures that can be optimized simultaneously, while worsening the performance of another groups of measures. If we take Table C.2 and calculate the correlations table for the different measures, we obtain the results shown in Table 4.7. From this table we observe the following. There is a high correlation between Hamming loss, example accuracy and micro-averaged precision, as well as between example accuracy and micro-averaged precision. Nevertheless, the correlation between these measures and the recall-based ones is negative. On the other hand, the recall-based measures are mutually correlated. This intuitively draws two measure clusters

$$C_1 = \{Hamming\_Loss, Example\_Accuracy, Micro\_Precision\}$$

$$C_2 = \{Example\_Recall, Micro\_Recall\}$$

In Tables 4.8 and 4.9 we show the rankings of the algorithm and method combinations for C_1 and C_2. In Table 4.8 it can be seen that the best combination is CLR with SVM. In general, SVM-based classifiers obtain a good performance. The tree-based classifier J48 obtains also good results when used together with binarization-based problem transformation methods (BR and CLR). We see also that `RAkEL` over `SVM` benefits measures from C_1, whereas EPPT and RAkEL over J48 favours measures from C_2.

We observe that SVM-based algorithms obtain good accuracies, regardless of the multilabel problem transformation method that has been used. We can also see that the CLR transformation method offers good Hamming loss and micro-averaged precision results, regardless of the underlying single-label algorithm. Additionally, CLR does not penalize recall measures when used together with SVM-based algorithms (this is also true for RAkEL), which is an advantage. In any case, it can be observed that methods and algorithms using binarization (CLR and BR, as well as SVM, which internally uses binarization when dealing with multiclass problems) obtain the best results for C_1.

Regarding recall, we see in Table 4.9 that EPPT over J48 is the best method for recall. It can be seen that binarization-based approaches do not obtain good recall results in general. EPPT, on the other hand, is quite successful. This shows that label dependence (which is only modelled in LP-based transformation methods) is more important for recall optimization than for other measures.

(a) Original dataset

Method	Algorithm	Av. Ranking
CLR	LibSVM	2.67
RAkEL	LibSVM	3.67
CLR	J48	5.0
BR	LibSVM	5.33
CLR	SMO	6.33
PPT	LibSVM	6.33
PPT	SMO	6.67
RAkEL	SMO	9.0
BR	J48	9.33
CLR	NNge	10.33
LP	SMO	10.33
BR	NNge	11.67
LP	LibSVM	12.0
BR	SMO	14.67
PPT	NB	15.67
EPPT	LibSVM	17.0
RAkEL	NNge	17.0
LP	NNge	17.33
PPT	NNge	18.33
LP	NB	18.33
EPPT	SMO	19.0
PPT	J48	20.33
EPPT	NB	20.33
RAkEL	J48	22.0
RAkEL	IBk	25.33
PPT	IBk	25.33
LP	IBk	27.0
BR	IBk	27.67
CLR	IBk	27.67
LP	J48	28.33
EPPT	NNge	30.67
EPPT	IBk	31.33
EPPT	J48	31.67
RAkEL	NB	32.67
BR	NB	35.0
CLR	NB	36.0

(b) Preprocessed dataset

Method	Algorithm	Av. Ranking
CLR	LibSVM	3.0
CLR	J48	4.667
BR	LibSVM	6.0
CLR	SMO	6.0
PPT	LibSVM	6.0
BR	J48	6.667
PPT	SMO	8.0
RAkEL	LibSVM	8.0
CLR	NNge	9.0
LP	LibSVM	10.67
LP	SMO	11.0
BR	NNge	11.67
RAkEL	SMO	13.33
BR	SMO	14.33
PPT	NB	15.0
LP	NNge	16.0
PPT	NNge	17.33
EPPT	LibSVM	17.67
PPT	J48	17.67
LP	NB	18.67
RAkEL	NB	19.0
EPPT	NB	19.67
EPPT	SMO	20.33
RAkEL	NNge	24.0
PPT	IBk	25.0
LP	IBk	26.33
BR	IBk	26.67
CLR	IBk	26.67
LP	J48	27.0
RAkEL	IBk	28.33
EPPT	NNge	29.0
EPPT	IBk	30.33
RAkEL	J48	31.67
EPPT	J48	34.33
CLR	NB	34.67
BR	NB	35.0

TABLE 4.8: Rankings for the first cluster of measures (Hamming loss, example accuracy, micro-averaged precision

(a) Original dataset

Method	Algorithm	Av. Ranking
EPPT	J48	1.0
EPPT	NNge	2.0
BR	NB	3.0
CLR	NB	4.0
EPPT	SMO	5.0
EPPT	LibSVM	6.0
RAkEL	J48	7.5
RAkEL	NB	7.5
RAkEL	SMO	8.5
CLR	SMO	10.0
RAkEL	LibSVM	11.0
EPPT	IBk	12.0
CLR	LibSVM	13.0
BR	SMO	14.5
RAkEL	NNge	14.5
PPT	SMO	16.5
BR	LibSVM	17.5
PPT	LibSVM	17.5
LP	SMO	19.0
EPPT	NB	19.5
PPT	NNge	21.5
BR	J48	23.0
LP	LibSVM	23.0
LP	NNge	23.5
CLR	J48	24.0
PPT	J48	26.5
CLR	NNge	27.5
LP	J48	27.5
BR	IBk	30.0
CLR	IBk	30.0
PPT	NB	30.0
BR	NNge	31.0
RAkEL	IBk	32.0
LP	IBk	34.0
LP	NB	35.0
PPT	IBk	36.0

(b) Preprocessed dataset

Method	Algorithm	Av. Ranking
EPPT	J48	1.5
RAkEL	J48	1.5
EPPT	NNge	3.0
EPPT	SMO	4.0
RAkEL	SMO	5.0
EPPT	LibSVM	6.0
RAkEL	LibSVM	7.0
RAkEL	NNge	8.0
CLR	NB	9.0
BR	NB	10.0
RAkEL	NB	11.0
CLR	SMO	12.0
EPPT	IBk	13.0
CLR	LibSVM	14.0
BR	SMO	15.0
BR	LibSVM	16.5
PPT	LibSVM	16.5
PPT	SMO	18.0
EPPT	NB	19.5
LP	SMO	20.0
PPT	NNge	21.0
LP	LibSVM	22.0
BR	J48	23.0
CLR	J48	23.0
LP	NNge	24.5
RAkEL	IBk	26.0
PPT	J48	27.5
CLR	NNge	28.5
LP	J48	28.5
BR	IBk	29.5
CLR	IBk	29.5
BR	NNge	33.0
PPT	NB	33.0
LP	IBk	33.0
LP	NB	35.5
PPT	IBk	35.5

TABLE 4.9: Rankings for the second cluster of measures (Example Recall, Micro Recall)

Chapter 5

Data Mining for Text Streams

5.1 Introduction

This chapter is devoted to *text stream mining* [110], that is, the application of data stream mining to text data sources. Text streams consist of great amounts of documents that arrive continuously and must be processed only once, using limited time and space resources. In this online setting, the nature of examples belonging to a given class may change over time. This chapter presents relevant algorithms for data stream classification, and discusses different proposals to evaluate and compare data stream mining methods, including approaches such as sliding windows and fading factors [111] applied to the classical prequential error. These concepts are applied to the specific case of email foldering.

The main contribution of this chapter is *GNUsmail*, a framework for email foldering which uses stream mining capabilities, including statistical tests to detect significant differences between the performance of online algorithms. We empirically study several dynamical aspects of email datasets, such as the presence of different kinds of concept drift, and the effect of this on classifiers.

This chapter is partially based on the following publications in which the author has participated:

- Carmona-Cejudo, J.M.; Castillo, G.; Baena-García, M.; Morales-Bueno, R.: "A comparative study on feature selection and adaptive strategies for email foldering", *11th International Conference on Intelligent Systems Design and Applications (ISDA)* [112]
- Carmona-Cejudo, J.M., Baena-García, M., del Campo-Ávila, J., Bifet, A., Gama, J., Morales-Bueno, R.: "Online Evaluation of Email Streaming Classifiers Using GNUsmail". *IDA 2011* [113]
- Carmona-Cejudo, J.M., Baena-García, M., del Campo-Ávila, Bifet, A., Morales-Bueno, R.: "GNUsmail: Open Framework for On-line Email Classification". *ECAI 2010*: 90-100 [106]

The rest of this chapter is organized as follows: in Section 5.2, the subject of stream mining is introduced, including algorithms and evaluation approaches. Section 5.3 is devoted to the application of these concepts to the case of *electronic mail mining*. This Section presents GNUsmail, a framework for text categorization which incorporates stream mining capabilities that will be used in the experimental setup. Then, we apply state-of-art evaluation metrics to compare data stream mining algorithms over time, using the GNUsmail framework and several datasets from the ENRON corpus.

5.2 Stream mining

5.2.1 Introduction to data streams

Traditional machine learning algorithms were designed to obtain accurate models from a limited number of training examples. Most machine learning and data mining approaches assume examples generated from a stationary distribution, which fit in main memory [114]. Nevertheless, the quantity of available data is growing continuously, as we have discussed in Chapter 2. The amount of information created or captured is estimated to have exceeded available storage for the first time in 2007 [115]. There are increasingly more data sources that produce unbounded streams of non-stationary data, which, in many cases, are of transient nature, and not necessarily stored in a permanent way. Web 2.0 and the emergence of small devices such as smartphones are some of the reasons for this change of paradigm. As consequence, there is an emerging necessity for algorithms able to deal with this huge amount of evolving information in an computationally efficient way. This bears a relationship with the field of *green computing*, the study and practice of using computer resources in an efficient and environmentally sustainable way [116].

Another emergent aspect of current data sources is their continuous nature, giving birth to the *data stream* paradigm: the data examples arrive in one by one as a high speed stream. The data are not available for random access from disk or memory. Each example is processed only once, using limited space and time resources. The model has to be incrementally updated by the algorithm as each new example is inspected. The learning algorithms should have a additional property, the so-called *anytime* property, which means that the model has to be ready to be applied anytime between training examples. Due to time and space limitations, the computation of exact results will not always be possible [117], which means that it is often necessary to settle for approximate results in order to save system resources, as long as the resulting errors can be controlled and do not affect the effectivity of the learning system.

FIGURE 5.1: Data Stream classification cycle. The model receives data instances and uses them to improve the learning model. Additionally, the system is able to make predictions at any time. Source of the Figure: [115]

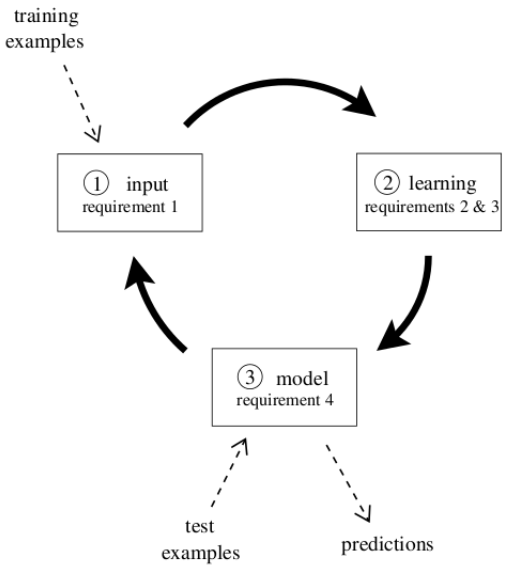
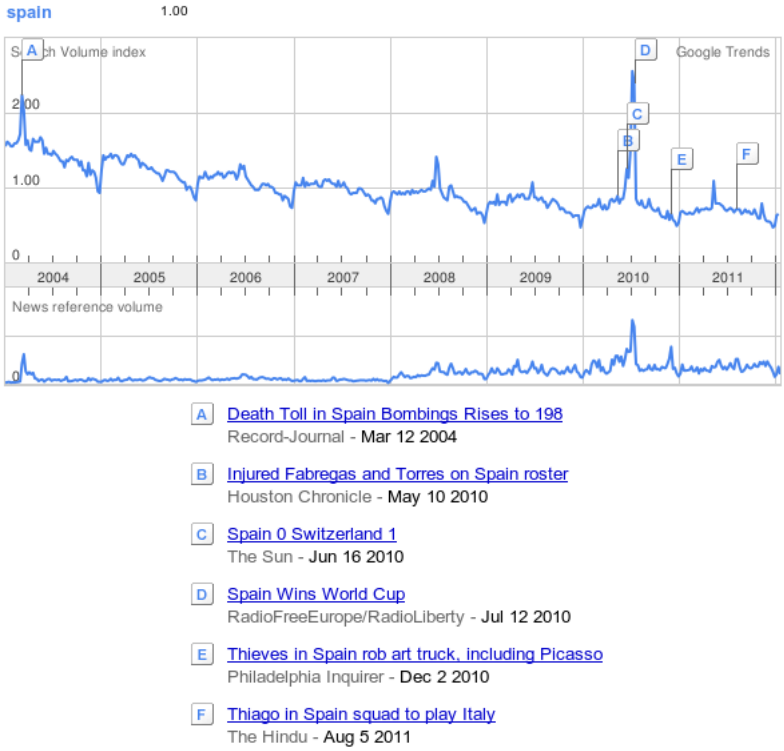


Figure 5.1 illustrates the typical use of a data stream classification algorithm. The following steps are iteratively followed:

1. The next available example from the stream is fed to the algorithm (*Requirement 1*)
2. The example is processed by the algorithm, updating its related data structures. Memory bounds have to be respected (*Requirement 2*), as well as time bounds (*Requirement 3*)
3. The algorithm is ready to accept the next available example. The model has to be usable to predict the class for unseen examples (*Requirement 4*)

Examples of data streams include computer network traffic, phone conversations, ATM transactions, web searches, sensor data and the main topic of this thesis, text streams.

FIGURE 5.2: Evolution of Google look-ups for 'Spain'. The volume of look-ups grows suddenly when Spain gets involved in some relevant event, such as winning a football tournament or a serious terrorist attack (Source: Google Trends)



5.2.2 Concept change

Data streams evolve over time, which means that either the target variable or the data distribution change over time. Machine learning algorithms can learn incorrect models if they suppose that the underlying concept is stationary, when in fact it is changing. The rationale for concept drift tracking is the assumption that the most recent data is more relevant to the model than old examples, which should be eventually forgotten over time. Concept drift is frequent in systems dealing with user preferences. For example, a hypothetical system that recommends scientific articles to a researcher has to cope with changes in the researcher's interests and lines of

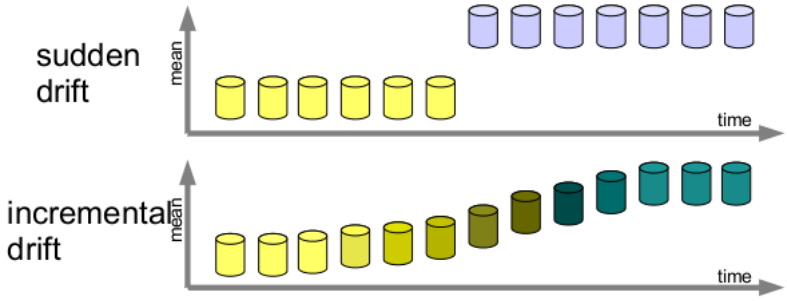
study. Another example of domain which is subject to concept drift are news classification systems, since new informative topics emerge continuously. Let us think about the Fukushima nuclear disaster in 2011. Before the disaster, news containing references to Fukushima were not necessarily related to nuclear energy. Nevertheless, during the crisis, news about Fukushima were very likely to be about the nuclear disaster. Other example of this phenomenon is depicted in Figure 5.2, where it can be observed that from time to time, the term *Spain* is associated with some relevant news (terrorist attacks or winning the Football World cup), which leads to an increase in the number of searches of the term *Spain*. Learning methods should thus be able to track this kind of changes.

Depending on how abrupt the concept change is, we can talk of *concept shift* (sudden change) or *concept drift* (gradual change). For example, a system that predicts air temperature given atmospheric conditions is subject to gradual change, while a system dealing with news can be facing an abrupt change when a hot topic emerges, such as the mentioned Fukushima crisis. In general, it is easier to deal with concept shift, since abrupt changes lead to a quickly noticeable deterioration of the model performance, which is easier to detect than when the change is gradual.

In Figure 5.3, extracted from the work of Žlobaitė [118], an illustration of the main types of concept change is given, assuming one-dimensional data, characterized by its the mean. Data from only one class is depicted.

Transversely to this differentiation between concept drift and shift, two types of concept drift are described in the literature [119]: *real* concept drift and *virtual* concept drift. *Real* concept drift refers to changes in the data distribution that eventually lead to changes in the target concept, whereas *virtual* concept drift is mostly associated with changes in the distribution of the *input space*, for example when the order of training instances is skewed. Nevertheless, in practice, virtual and real concept drift can appear together, for example when both the user's interest and document contents distribution change over time.

FIGURE 5.3: Graphical representation of the types of concept change. (Based on a work by I. Žliobaitė [118])



A different type of concept drift that appears in text data streams consists in the appearance of new highly relevant features (words) that do not belong to the original feature set [120]. This is sometimes known as *contextual* concept drift, and it is typical in text data sources [121]. We will deal with contextual concept shift in Chapter 6.

5.2.3 Learning algorithms for data streams

The data stream mining paradigm implies that learning algorithms must be *incremental*, that is, after a new example arrives, the current model has to be updated using only this example (that is, not referring to old examples). Furthermore, algorithms have to be *adaptive*, that is, they have to be able to adapt to a changing environment.

Broadly speaking, there are two strategies for using machine learning concepts applied to data streams. The first one is the *wrapper approach*, in which already existing batch (offline) algorithms are reused. The incoming training examples have to be stored into batch collections, and the obtained batch models have to be combined to provide the final prediction. In such an approach it is important to select a suitable size for the training batches. Very small sizes will result in poor accuracy, while very large sizes could make it difficult to process the stream at an acceptable speed. The main drawback

of this approach is that it consistently discards old information, regardless of whether it is still relevant or not for classification. The second approach is the *adaptation approach*: methods are designed specifically for the data stream setting. Such methods offer a greater control over processing times and memory management.

Arguably, the most influential algorithm in this field is VFDT, proposed by Domingos et al. [122]. VFDT is an online decision tree learning algorithm based on the Hoeffding bound. Typically, when learning decision trees in a batch scenario, the attribute with the highest information gain is used for split decisions. In the stream setting, Domingos proposes to use the Hoeffding bound, which states that, with probability $1 - \delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{5.1}$$

This bound is true regardless of the data distribution, being for this reason more conservative than distribution-based bounds. In order to decide on which attribute to split when learning a tree, the random variable being estimated is precisely the information gain. Hence, two parameters have to be tuned: $1 - \delta$ (confidence) and ϵ (error). The details of the learning process can be checked in [122]. The main advantage of VFDT are its theoretical properties: Domingos and Hulten [122] prove that its strong guarantees of high asymptotic similarity to the corresponding batch trees. In order to extend VFDT to be able to work with concept drift, Domingos et al. propose CVFDT (Continuous Very Fast Decision Trees) [123], which is based on the idea of growing alternatives subtrees which replace subtrees with a worse quality. UFFT (Ultra Fast Forest of Trees) is an extension of CVFDT which allows handling numerical attributes [124]. Another decision tree learning algorithms has been proposed by Ding et al. [125], based on the Peano count tree data structure.

Aside from decision tree algorithms, other proposals have appeared in the literature. Some of the most relevant ones are the following:

- Wang et al. [126] have proposed a general framework for mining data streams based on weighted classifier ensembles. The weight of each component of the ensemble depends on the expected prediction accuracy.

- Another general framework that supports insertion and deletion of data records has been proposed by Ganti et al. [127]. This model has the advantage of being applicable to any incremental data mining algorithms, that is, to any algorithm that can include new information once it has been generated.

- Papadimitriou et al. have proposed AWSOM (*Arbitrary window Stream modeling Method*), which uses wavelet coefficients for compact information representation [128]. AWSOM was especially designed for handling data streams from sensor networks, and puts emphasis on automatic parameter tuning.

- A further method has been proposed by Aggarwal et al. [129], based on micro-clusters, which can dynamically select the appropriate window of past training data to build the classifier.

- Gaber et al. propose *LWClass* (Lightweight classification), using K-nearest neighbors for updating the frequency of class occurrence[130].

- Last proposes OLIN (Online Information Network), based on building an info-fuzzy network based on a sliding window of the latest examples. The system dynamically adapts the size of the training window and the frequency of model reconstruction to the current rate of concept drift [115], [131]. OLIN generates a new model for every new sliding window. This approach ensures accurate and relevant models over time. However, the cost of generating new models is high.

- Another successful algorithms include SCALLOP, proposed by Ferrer-Troyano et al. [132], a rule-based learner, as well as ANNCAD, proposed by Law and Zaniolo [133], based on nearest neighbors.

An important family of algorithms for data stream mining are *ensemble methods*, which were introduced in Chapter 2. Ensemble methods are based on the idea of combining several weaker models in order to obtain a stronger classifier. An ensemble of classifiers will provide better results than any of the individual members, as long as every member is better than random guessing, and that the models are sufficiently diverse (i.e., that the errors are not correlated) [115]. Such methods exploit instability of base learners to attain diversity. For instance, random trees are good base learners, since they are inherently unstable, because of the greedy local decisions.

Two important ensemble methods that can be applied for online learning are *bagging* and *boosting*. Bagging (abbreviation of *bootstrap aggregating*) involves giving each model in the ensemble a vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly-drawn subset of the training set. Given a training set T of size N, batch bagging creates M base models, each one of them trained by calling the batch learning algorithms L_b on a bootstrap sample of size N created by drawing samples with replacement [134]. Each base model's training set contains K copies of each example from the training set. As $N \to \infty$, the distribution of K tends to a $Poisson(1)$ distribution. Bagging can also be performed online [134]: each time an example arrives, it is replicated K $Poisson(1)$ times, and the model is updated using the base learning algorithm. An unweighted voting setting is used for prediction [134].

Regarding boosting, it involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models have misclassified. Oza also proposed an online adaptation of the popular AdaBoost boosting algorithm, AdaBoost.M1 [134]. AdaBoost works by generating a sequence of base models $h_1, h_2, \ldots, h_m$ using weighted training sets such that the examples misclassified by h_{m-1} are given half the total weight when generating h_m, whereas the correctly classified examples are given the remaining half of the weight. If the base model can not directly employ weights, samples with replacement can be generated.

This strategy can equally be used for data stream scenarios, as it is done in online bagging.

A general adaptive framework that can be used for online classification and that contains the elements reviewed so far is the *Adaptive Prequential Learning Framework* (`AdPreqFr4SL`), proposed by Castillo et al. in [135]. Its main assumption is that observations arrive sequentially, which allows the environment to change over time. Without loss of generality, it can be assumed that at each time point data arrives in batches so that perform model adaptation can be performed in jumps when there is some accumulated experience. These batches are assumed to be of equal size. We provide more details about AdPreqFr4SL in Chapter 6

5.2.4 Change drift detection algorithms

As previously mentioned, data streams are inherently non-stationary, which means that concept drift can occur in them. Strategies for handling concept drift can be classified into two major groups [118]:

- The model learner regularly evolves, independently of alarms or detectors
- The model adaption is initiated by a *trigger* (change detector)

For the first group, no change detection algorithm is needed, because the learner itself deals with concept drift. For the second group, we need change detection algorithms such as the following[118]:

- *Statistical Process Control* (SPC) is based on controlling the probability of error. A *warning* and *drift* level are defined. When the monitored error exceeds the warning threshold, the system is said to be in *warning mode*, and the time stamp t_w is stored. If the error exceeds the drift level at time t_d, the method considers that a change has occurred, and the classifier is retrained using the examples from t_w to t_d [136].

- *Adaptive Windowing* (ADWIN) keeps a sliding window W and compares the distribution of two subwindows of W (W_0 and W_1). If the distributions are distinct enough, a change in the distribution is declared [137]. This method is based on maintaining a sliding window of variable length, which is used for the following tasks:
  - To detect concept change (comparing different subwindows)
  - To obtain updated statistics from recent examples
  - To rebuild or revise the model

  The Hoeffding bound is used for adjusting the size of the sliding window.

- *Fixed Cumulative Windows Model* (FCWM) is based on comparing histograms corresponding to the distribution observed in the past and the current distribution, using the Kullback-Leibler divergence [138].

- The *Page Hinkley Test* (PHT) is based on tracking the accumulated difference between the observed values and their mean, comparing it with its minimum value [139]. The accumulated difference, U_T, is modeled as

$$U_T = \sum_{t=1}^{T} (x_t - \overline{x}_T - \delta) \tag{5.2}$$

  where δ represents the allowed deviations, and $\overline{x}_T = 1/\sum_{t=1}^{T} x_t$. For detecting increases, the minimum value of this variable is computed: $m_T = min(U_t, t = 1, \ldots, T)$ and the difference $U_T - m_T$ is monitored. Concept drift is signaled if this difference is bigger than a threshold λ. λ can be tuned for sensitivity (in order to avoid false alarms at the expense of real changes detection).

- *Drift Detection Method* (DDM): compare statistics of two windows: one with all the data, and another with data from the beginning until the number of errors increase [140]. For each

point i in the stream there is a probability p_i that the prediction is incorrect (the error rate), which standard deviation is given by $s_i = \sqrt{p_i(1-p_1)/i}$. Since for a sufficiently big number of examples the underlying binomial distribution can be approximated by a normal, the $1-\alpha/2$ confidence interval is approximately $p_i \pm \alpha * s_i$. The drift detection method manages two registers during the training of the learning algorithm, p_{min} and s_{min}. For each new example, these variables are updated if $p_i + s_i < p_{min} + s_{min}$. The *warning level* is reached if $p_i + s_i \geq p_{min} + 2 * s_{min}$, whereas the *drift level* is reached if $p_i + s_i \geq p_{min} + 3 * s_{min}$. DDM shows a good performance for detecting abrupt changes, as well as gradual changes provided that the gradual change is not very slow.

- EDDM (Early Drift Detection Method) is similar to DDM, but is based on comparing distances between classification errors [141]. It was designed to improve the detection in case of slow gradual change, measuring distances between errors rather than error rates.

- COC (*Change of Concept*), proposed by Lee and Magoules [142] uses correlation between consecutive batches of examples to discover concept drift in a data stream: if there is the change between two batches of examples, the value distribution of one batch does not correlate with that of the other. This approach uses a user-fixed window size.

5.2.5 Evaluation of stream mining methods

Data stream methods can be evaluated from the point of view of space complexity, learning time and, naturally, generalization power. In this section we deal with the evaluation of generalization power. Time and space complexity are also important factors, although the non-stationary nature of data streams does not have such a direct impact on them.

In data stream scenarios, neither cross-validation nor other sampling procedures are suitable for evaluation, since data are potentially

infinite: there is no 'closed-ended' dataset available. Evaluation in non-static dataset is still an open problem, and, quoting the words of Gama et. al [111], "*there is no golden standards for assessing performance in non-stationary environments*". Sequence analysis methods, like *holdout* or the *prequential* evaluation procedures, are more appropriate. Sequential analysis is a subfield of statistics where the sample size depends randomly on the accumulating data [111].

When using holdout, an independent set (the *holdout set*) is selected. The current decision model is applied to the test set at regular time intervals or number of examples. When using the prequential measure [143], a prediction is made for each new example (i-th example) using the current model. Once the real class is known, we can compute the loss function $L(y_i, \hat{y}_i)$, where L is a loss function (such as the $0-1$ loss function), y_i is the real class and $\hat{y}_i$ is the prediction, and a cumulative loss function S_i is updated: $S_i = \sum_{j=1}^{i} L(y_j, \hat{y}_j)$. After that, the model is updated using that example. Thus, we can evaluate the performance of the system (which is influenced by the increasing number of examples that are constantly arriving) and the possible evolution of the models (because of concept drift). Using the previous cumulative loss function S_i, we can estimate the mean loss at every moment i: $E_i = S_i/i$.

The prequential error is a pessimistic estimator, since the early poor performance of the algorithms is never forgotten and influences the calculated average. Nevertheless, it can be made more realistic by using forgetting mechanisms, such as *sliding windows* or *fading factors*, and additionally overcome other problems that are present in the holdout approach, coming from the non-stationary properties of data streams [111]. The error estimates using forgetting mechanisms converge to the holdout estimator.

The method based on sliding window is based on considering only the last examples. Therefore, not all the examples are used to evaluate the performance: the oldest ones are forgotten and the most recent ones are stored in a window of size w. Those w examples in the window are the examples used to calculate different performance

measures. On the other hand, using fading factors, which is the preferred method according to [111], allows to decrease the influence of the examples in the measure as they get older. For example, if we compute the loss function L for every single example, the prequential error at moment i using fading factors is defined as $E_i = S_i/B_i$, where $S_i = L(y_j, \hat{y_j}) + \alpha \cdot S_{i-1}$ and $B_i = 1 + \alpha \cdot B_{i-1}$ ($\alpha < 1$).

Different ways of comparing the performance of two classifiers exist; the McNemar test [114] is one of the most used tests. This test needs to store and update two quantities: a (the quantity of the examples misclassified by the first classifier and not by the second one) and b (the quantity of the examples misclassified by the second classifier and not by the first one). The McNemar statistic (M) rejects the null hypothesis (the performances are the same) with confidence level of 0.99 if its value is greater than 6.635, since it follows a χ^2 distribution. The statistic is calculated as

$$M = sign(a - b) \times \frac{(a - b)^2}{a + b} \tag{5.3}$$

In order to extend its usage to the prequential approach, we have two options. If we consider a sliding window context, we only need to use the examples in the window. But if we consider the usage of fading factors, we should adapt the definition as follows:

$$M_i = sign(a_i - b_i) \times \frac{(a_i - b_i)^2}{a_i + b_i}$$

where $a_i = f_i + \alpha \cdot a_{i-1}$ and $b_i = g_i + \alpha \cdot b_{i-1}$, being $f_i = 1$ if and only if the i-th example is misclassified by the first classifier and not by the second one ($f_i = 0$ otherwise) and $g_i = 1$ if and only if the i-th example is misclassified by the second classifier and not by the first one ($g_i = 0$ otherwise).

5.3 An application: stream mining for electronic mail

In this section, data stream mining concepts are applied to a specific case of document classification: *email foldering*, that is, categorization of emails into folders. Note that this is more specific than *spam detection* (detection of fraudulent emails), where there are only two classes actually: spam and not-spam emails. In Subsection 5.3.1 we review previous work in relation with email classification, while in Subsection 5.3.2 we introduce a framework which we use for email classification. Later on, in Subsection 5.3.4 we present a study to explore different aspects of data stream mining on several email datasets from the Enron collection.

5.3.1 Background: classification of electronic mail

Email is an extremely fast, cheap and ubiquitous means of communication that can be used in personal computers, smart phones, tablets and other electronic gadgets. The task of handling large volumes of emails poses significant challenges for text mining and machine learning. One of the most challenging tasks is *email foldering* - the automatic categorization of emails into folders. This problem can be modeled as a multinomial classification problem concerned with learning a classifier that maps examples (emails) into classes (category folders). As pointed out in the pioneer work of Bekkerman et al. [144], "*email foldering is a rich and multi-faceted problem, with many difficulties that make it different from traditional topic-based categorization*". One of these difficulties, that we discuss in this section, is the dynamic, non-stationary nature of email streams.

Most of the systems that try to classify emails in an automatic way have been implemented using a batch approach [38], which, at best, can be updated only at regular intervals [145]. *RIPPER* [46] was one of the first algorithms used for email classification, making use of *TF-IDF* weighting (term frequency - inverse document frequency) to produce *if-then* rules. RIPPER was an influential rule-induction

algorithm, being able to perform efficiently in large noisy datasets. An advantage of RIPPER was that it produces easy-to-understand rules as a result, which allows users to change them as necessary. A common algorithm for email foldering is Naïve Bayes, which has been used for instance in *ifile* [38] for the purpose of general email classification, or in *SpamCop* [146] for specific spam filtering. Another popular algorithm for email foldering systems is Support Vector Machines, which was used in *MailClassifier* [147] (an extension for Thunderbird). In general, as pointed out in a work by Brutlag and Meek that compared SVM, Naïve Bayes and k-NN across several datasets [148], different email datasets cause more variation in the performance that using different classification algorithms.

With regard to online learning, little effort has been devoted to applying stream mining tools [149] to email classification. For example, Manco et al. [150] have proposed a framework for *adaptive mail classification* (*AMCo*), but they admit that their system does not actually take into account the updating of the model. Another system that incorporates incremental learning is the one proposed by Segal et al. Their system, SwiftFile [151], uses a modified version of the AIM algorithm [152] to support incremental learning. Although they have advanced in this line of research, they admit that there are limitations, like few incremental algorithms for text classification or closed integration between email clients and classifiers. Moreover, the classifier does not include concept drift detection. Chang and Poon [153] use k-NN and Naïve Bayes incrementally, although without explicitly taking into account concept drift. They use the concept of *shingles* (sequences of consecutive words) as features, selecting those with the biggest TF-IDF weight for reducing the computational complexity. Katakis et al. [154] have proposed an application of tracking recurrent contexts to spam detection, using clustering to discover concepts that reappear over time, and a different incremental classifier for each of those concepts.

5.3.2 GNUsmail: a framework for online email classification

GNUsmail [106], the main contribution of this chapter, is an open-source framework for online adaptive email classification, with an extensible *text preprocessing module* which is based on the concept of filters that extract attributes from emails, and an equally extensible *learning module* into which new algorithms, methods and libraries can be easily integrated. GNUsmail contains configurable modules for reading email, preprocessing text and learning. In the learning process, the email messages are analysed as an infinite flow of data. The source code of GNUsmail is available at `http://code.google.com/p/gnusmail/` and it is licensed under the GPLv3 license. We now explain in more detail the different modules integrated in GNUsmail.

The *reading email module* can obtain email messages from different data sources, such as a local file system or a remote IMAP server. This allows the system to process datasets like the ones from the Enron corpus or to process personal email messages from remote servers like GMail.

The *text preprocessing module* is a multi-layer filter structure, responsible for performing feature extraction tasks. Non-topic folders are skipped in the learning process. Every mail belonging to any other folder goes through a pipeline of linguistic operators which extract relevant features from it. GNUsmail performs a feature selection process using different methods for feature selection [33]. Some ready-to-use filters are implemented as part of the GNUsmail core, and new ones can be incorporated, giving developers the chance to implement their own filters, as well as to test and evaluate different techniques. We have implemented a linguistic filter to extract attributes based on relevant words. It is based on the ranking provided by the folder-wise *TF-IDF* weighting function. We have implemented several filters to extract non-linguistic features such as *CC*, *BCC*, *sender*, *number of receivers*, *domain of sender*, *size*,

number of attachments or *body/subject length*. We have also implemented filters to extract metalinguistic features, such as *capital letters proportion*, or the *language* the message is written in.

The *learning module* makes it possible to incorporate a wide variety of stream-based algorithms, such as those included in the WEKA [155] or MOA [116] frameworks. WEKA (*Waikato Environment for Knowledge Analysis*) methods are used mainly with small datasets in environments without time and memory restrictions. MOA (*Massive Online Analysis*) is a data mining framework that scales to more demanding problems, since it is designed for mining data streams.

By default, GNUsmail offers three updateable classifiers from the WEKA framework, although more can be easily added. The first one is Multinomial Naïve Bayes, a probabilistic classifier which is commonly used as a baseline for text classification. The other two classifiers belong to the family of lazy classifiers, which store all or a subset of the learning examples; when a new sample is given as input to the classifier, a subset of similar stored examples is used to provide the desired classification. IBk is one of these methods, a k-nearest neighbours algorithm, to be precise, that averages the k nearest neighbours to provide the final classification for a given input. NN-ge (*Nearest Neighbour with Generalised Exemplars*) [44] is another nearest-neighbours-like algorithm that groups together examples within the same class. This reduces the number of classification errors that result from inaccuracies of the distance function.

In the streaming scenario, GNUsmail uses MOA by including its tools for evaluation, a collection of classifier algorithms for evolving data streams, some ensemble methods, and drift detection methods. `HoeffdingTree` is the MOA implementation of VFDT. An improved learner available in MOA is the `HoeffdingTreeNB`, which is a HoeffdingTree with Naïve Bayes classifiers at leaves, and a more accurate one is the `HoeffdingTreeNBAdaptive`, which monitors the error rate of majority class and Naïve Bayes decisions in every leaf, and chooses to employ Naïve Bayes decisions only where they proved accurate in the past. Additionally, some ensemble methods from the MOA framework are included, such as `OzaBag` and `OzaBoost`. For

concept drift detection, we have included DDM [124] as the default algorithm.

5.3.3 Exploratory analysis of the Enron email corpus

In our experiments we have used the Enron [144] email corpus, which contains archived emails for 150 senior managements of the Enron Corporation. We have used ten datasets in total: the corresponding to the same seven former employees that were used in related studies [112, 144, 156], and the mail datasets for three additional employees (*rogers-b*, *germany-c* and *shapiro-r*). During the data cleaning process we have removed the non-topical folders (`Inbox`, `Sent`, `Sent Items`, `Trash`, `All documents`, `Draft`, `Discussion threads` and `Deleted`). In addition, we have also eliminated those folders that contain fewer than 10 email messages (not less than two as proposed, for instance, in Bekkerman's study [144]). Finally, each email vector is attached to its category. The messages have been arranged chronologically.

Table 5.1 summarizes the main characteristics of the datasets. We have included statistics about the number of categories and messages, as well as descriptive statistics about the number of messages by folder. The last two columns contain information that help us measuring the degree of skewness of the category distribution. The former shows the ratio between the major and minor categories. Higher ratios result in more skewed category distributions. The latter shows the entropy values; higher values of entropy imply more uncertainty in the distribution.

We can observe that there is a lack of balance in the number of emails for each category. For example, for the user `germany-c`, the biggest category is 181.98 times bigger than its smallest category.

In a previous work [106], we made some initial experiments with a test and a training set, using the first 30% for training and the remaining 70% of the stream for testing. These two scenarios were designed to compensate for the lack of the implementation of an incremental updating algorithm for feature selection. Such division is

			# of messages by folder							
User	Folders	Messages	Avg.	Min	Q1	median	Q2	Max	Max/Min	Entropy
beck-s	56	1741	31.09	10	12.8	20.0	39.0	162	16.2	5.33
farmer-d	22	3572	162.36	10	14.8	31.5	160.0	1177	117.7	3.19
kaminski-v	24	2644	110.17	14	27.3	55.0	126.3	544	38.9	3.86
kitchen-l	40	3733	93.33	11	24.0	56.0	96.8	685	62.3	4.56
lokay-m	11	2473	224.81	15	37.5	126.0	210.5	1149	76.6	2.49
sanders-r	21	1153	54.91	10	22.0	25.0	48.0	419	41.9	3.49
williams-w3	21	2738	228.17	11	15.5	18.5	81.0	1398	127.1	1.72
rogers-b	12	3323	276.9	11	32.0	89.5	212.5	1946	176.9	2.12
germany-c	26	3372	129.7	11	17.5	34.0	56.2	1992	181.1	2.55
shapiro-r	52	2966	57.04	10	16.0	25.5	51.2	1135	113.5	4.22

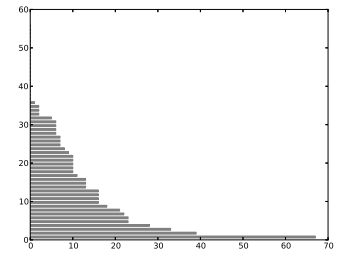
TABLE 5.1: Statistics on Enron datasets after applying the cleaning process

	first 30%					**Whole dataset**				
	# Categories	# Examples	# Features			# Categories	# Examples	# Features		
			5 lines	10 lines	All lines			5 lines	10 lines	All lines
beck-s	36	701	2877	3639	5596	56	1229	5291	6702	9874
farmer-d	14	2251	2983	3764	6400	22	2498	5889	7693	13253
kaminski-v	22	1776	3312	4548	6810	24	1856	6718	9524	12818
kitchen-l	28	1447	5029	6610	8491	40	2613	8856	12090	21933
lokay-m	8	1633	3132	4244	6270	11	1729	6786	9775	12009
sanders-r	15	608	2096	2868	4558	21	812	4868	6695	8723
williams-w3	11	531	3697	4788	6081	21	1929	4840	6287	7884
rogers-b	9	996	2023	2701	4770	12	3323	4207	5640	13079
germany-c	7	1011	1819	2226	3729	26	3372	4398	5703	10982
shapiro-r	40	889	2443	3363	8272	52	2966	5993	8070	19472

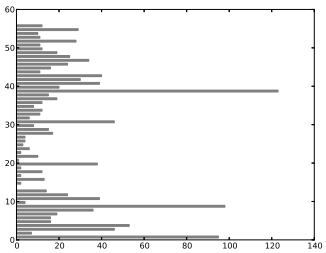
TABLE 5.2: Statistics on Enron datasets after applying splitting

useful because it demonstrates the necessity of incorporating mechanisms for updating the feature and category sets. In particular, Table 5.2 shows statistics about the resulting number of categories, messages and features from the whole dataset, as well as for the first 30% of the stream. It shows that a large amount of features and categories were not present in the first 30% of the stream.

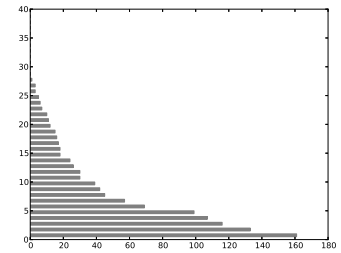
Additionally, Figure 5.4 shows the category distribution for the first 30% and 100% of some datasets. As observed, for some users, such as `lokay-m`, the distribution is very similar in the training and in the test sets, while for most users, such as `beck-s` or `kitchen-l`, the distribution is completely different. An interesting phenomenon that can be observed in `kitchen-l` is that new categories emerge in the test set that turn out to be majority categories. These observations lead us to the conclusion that new categories and features appear, and that category distributions change over time, making it necessary to introduce concept drift monitoring mechanisms.



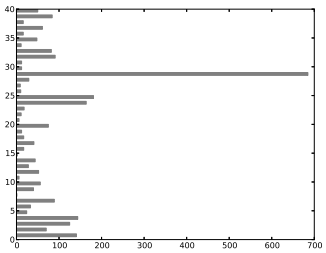
(a) Becks-s: first 30%



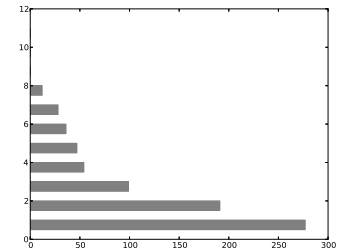
(b) Becks-s: 100%



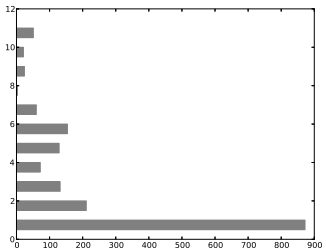
(c) Kitchen-l: first 30%



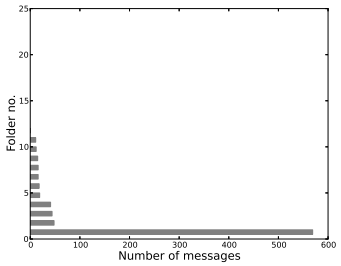
(d) Kitchen-l: 100%



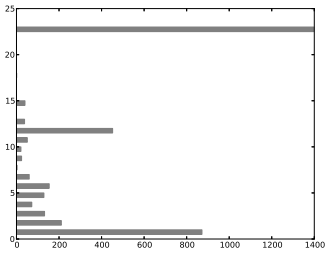
(e) Lokay-m: first 30%



(f) Lokay-m: 100%



(g) Williams-w3: first 30%



(h) Williams-w3: 100%

FIGURE 5.4: Differences in category distribution between the first 30% and the 100% of the examples in the dataset

5.3.4 Experimental study: online comparison of algorithms

In this study we use the aforementioned prequential-based error metric for online comparison of algorithms. This study takes into account how such measure evolves during run time. The rationale is that the relative performance of a set of algorithms is subject to change over time. Our objective is to use these measures to gain further insight into the dynamics of email streams, focusing on the effect of concept change.

GNUsmail has been used to carry out the comparisons. The GNUsmail source code incorporates an experimentation launcher that allows to replicate the experimentation described here.

For both the incremental and the online approach, the messages are analysed in chronological order, and they are given to the underlying learning algorithms one by one. For each dataset, we compare the online performance of the following algorithms:

- OzaBag over NN-ge , using DDM for concept drift detection
- Plain NN-ge. Recall that an incremental implementation of NNge is provided in the Weka framework.
- Hoeffding Trees (VFDT).
- Majority class, used as a baseline for comparison.

Instead of using the classical prequential error, the launcher can be configured to use sliding windows or fading factors. For each algorithm and dataset, GNUsmail plots the prequential-based metrics, to visually analyse the differences in performance. These plots also show the concept drifts when there is a concept drift detector associated with an algorithm. We carry out experiments with the DDM detector, and report the point where a concept drift warning that finally leads to an actual concept drift is detected.

TABLE 5.3: Final MOA prequential accuracies with bagging of DDM and NN-ge

Dataset	Correct/Total	Percentage
`beck-s (101 folders)`	1071/1941	55.18%
europe	131/162	80.86%
calendar	104/123	84.55%
recruiting	89/114	78.07%
doorstep	49/86	56.97%
`kaminsky-v (41 folders)`	1798/2699	66.62%
universities	298/365	81.64%
resumes	420/545	77.06%
personal	154/278	55.4%
conferences	163/221	73.76%
`lokay-m (11 folders)`	1953/2479	78.78%
tw_commercial_group	1095/1156	94.72%
corporate	345/400	86.25%
articles	152/232	65.51%
enron_t_s	86/176	48.86%
`williams-w3 (18 folders)`	2653/2778	95.5%
schedule_crawler	1397/1398	99.91%
bill_williams_iii	1000/1021	97.94%
hr	74/86	86.05%
symsees	74/81	91.36%
`farmer-d (25 folders)`	2743/3590	76.41%
logistics	1077/1177	91.58%
tufco	588/608	96.71%
wellhead	210/337	62.32%
personal	211/320	65.94%
`kitchen-l (47 folders)`	2254/3790	59.47%
esvl	640/712	89.89%
hr	159/299	53.18%
east_power	160/253	63.24%
regulatory	210/242	86.78%
`sanders-r (30 folders)`	887/1207	73.49%
iso__pricecaps	404/420	96.19%
nsm	109/119	91.6%
senator_dunn	43/83	51.81%
px	49/68	72.06%

In addition, we use the McNemar test because it can be easily adapted to deal with the online characteristic [111]. Thus, graphics are produced for each of the algorithm pairs, and the critical points of the McNemar test for a significance of 99% are shown as horizontal lines, while the zone where no significant difference is found is shown on a gray background.

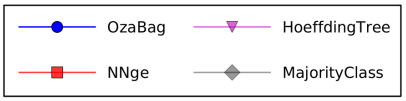
In Table 5.3, the final prequential accuracy with bagging of DDM and NN-ge is shown for each folder. As can be seen in this table, the classification results are similar to those in the works of Bekkerman et al. [144] and Bermejo et al. [156]. The final results depend largely

on each specific dataset, and, more precisely, on their folders. Folders with a big amount of messages and dealing with very specific subjects are more likely to be correctly learnt by the models. There are folders whose final prequential value goes beyond 90%, as in the cases of `farmer-d`, `kaminski-v`, `kitchen-l` and `lokay-m` (see table 5.3). The results for this dataset are illustrative, since it can be seen that the best-scored folders are precisely the ones that have a specially large amount of messages in them.

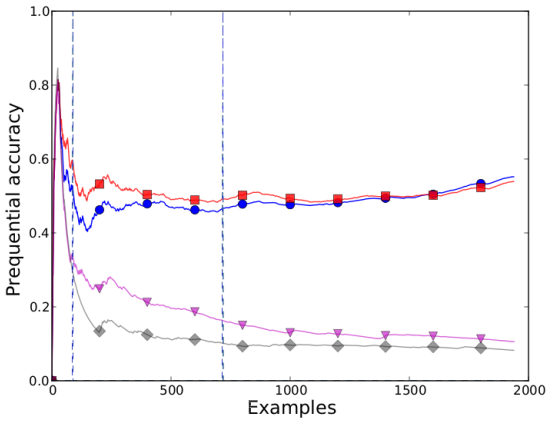
When analysing these results, one should take into account that we are dealing with a massive multi-class domain. As a result, it is difficult to get good overall performance as some folders will be learnt better than others. In the case of `beck-s`, for instance, we have over 100 different values for the folder. Another difficulty is that the classes are extremely unbalanced, which typically causes problems for learning algorithms [157]: for `williams-w3` there are folders with more than one thousand mails (`bill_williams_iii`), and folders with only a dozen mails (`el_paso`). Furthermore, it is less than obvious what the semantics of the folders intended by the user is: both `hr` and `human_resources` contain emails dealing with human resources which makes it difficult to determine how the messages should be classified. Another general problem is that some folders contain mails which have little in common (let us think about `personal` folders).

In Figures 5.5, 5.6, 5.7 and 5.8 the evolution of the prequential accuracy is shown for two specific datasets, `beck-s` and `kitchen-l`, plotting both the classical and the fading factors versions of the prequential error. As a general observation, using fading factors improves results, since the effect of poor initial performance is reduced because of the limited number of available messages at the beginning.

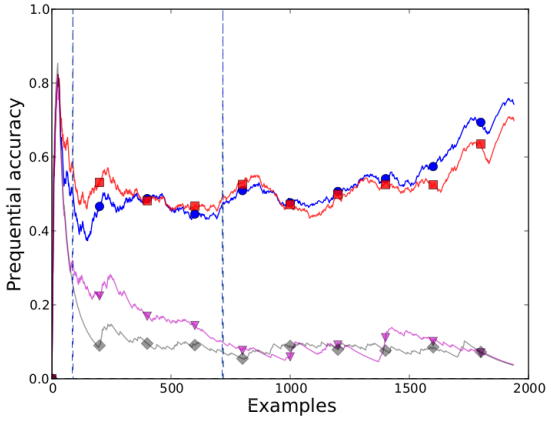
It can be seen for both datasets that a concept change is detected by DDM. This detection leads to the combination of DDM and OzaBag over NNge to show a better performance that plain NNge. Nevertheless, in the case of `beck-s` this superiority is not found to be statistically significant by the McNemar test, while, in the case of



(a) Legend

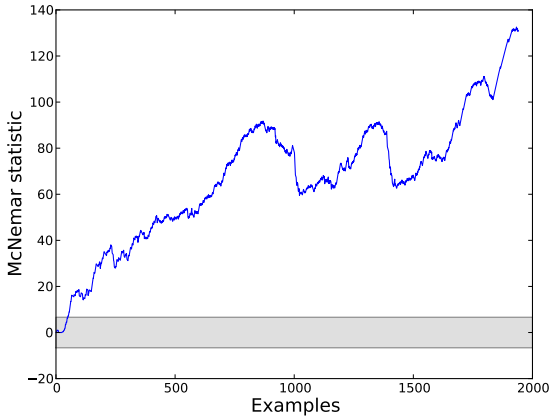


(b) Prequential error

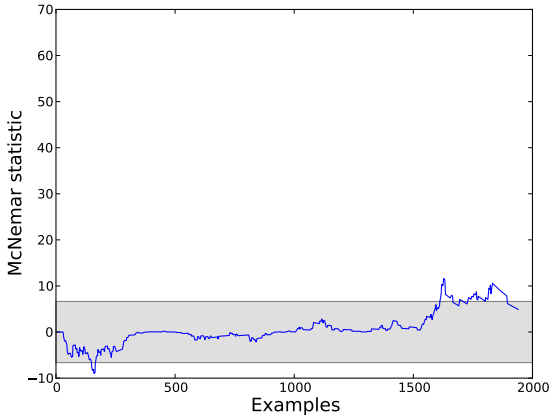


(c) Fading factors preq.

FIGURE 5.5: Prequential accuracies for beck-s. For the fading factors graphics, $\alpha = 0.995$ has been selected. The detection of concept drift by the DDM algorithm is marked with a vertical dashed line.

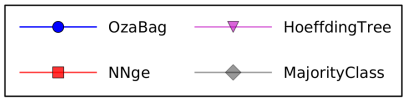


(a) McNemar test: NNge vs. HoeffdingTree. Fading factors

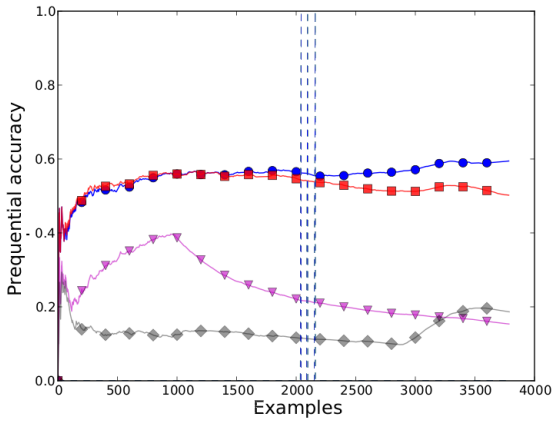


(b) McNemar test: OzaBag vs. NN-ge. Fading factors

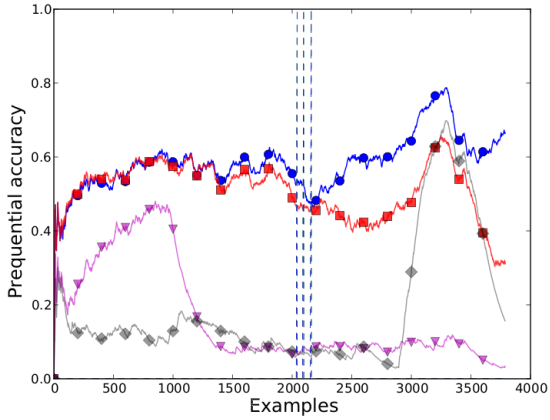
FIGURE 5.6: McNemar test for beck-s, used to compare OzaBag over NNge with DDM for concept drift detection



(a) Legend

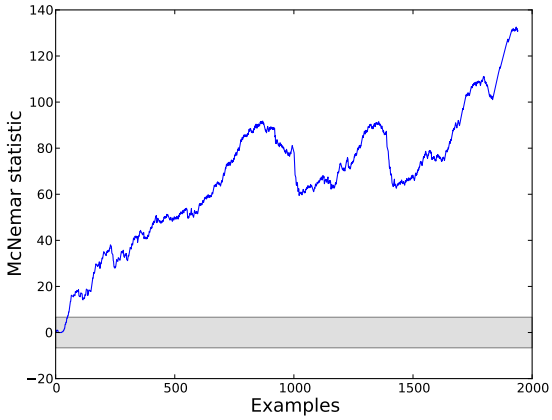


(b) Prequential accuracy

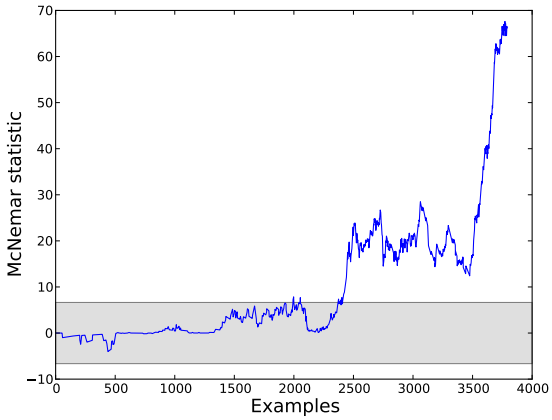


(c) Fading factors preq. accuracy

FIGURE 5.7: Prequential accuracies for kitchen-l. For the fading factors graphics, $\alpha = 0.995$ has been selected. The detection of concept drift by the DDM algorithm is marked with a vertical dashed line.



(a) McNemar test: NNge vs. HoeffdingTree. Fading factors



(b) McNemar test: OzaBag vs. NN-ge. Fading factors

FIGURE 5.8: McNemar test for kitchen-l, used to compare OzaBag over NNge with DDM for concept drift detection

`kitchen-l` the performance of the online learner that incorporates concept drift detection begins to be significantly better than plain NNge. This leads to the conclusion that concept drift is an important issue in the case of `kitchen-l`, and its detection and adaptation to it is important in order to obtain good results. Notwithstanding, in the case of `beck-s` such significant improvement is not observed. This can be caused by the concept drift being of the *virtual* type. However, there is a significant difference between the combination of OzaBag, DDM and NNge, and the plain version of NNge, although for a short period of time. Since no concept drift has been signaled, it can be concluded that the difference is due to the effect of the ensemble method, OzaBag.

On the other hand, NN-ge outperforms other basic methods, whose relative performance varies for every dataset. The performance obtained by NN-ge is comparable with the best results obtained by the aforementioned works [144, 156], which use SVM-based approaches. Specifically, the methods based on HoeffdingTree have lower accuracy than the methods based on NN-ge when directly applied to the domain of email classification. These streaming decision tree methods frequently need millions of examples to start showing good performance. The main problem of tree-based algorithms is that once a decision has been made on a node, it cannot be undone.

Chapter 6

Comparative Study on Feature Selection and Adaptive Strategies for Email Foldering Using the ABC-DynF Framework

6.1 Introduction

As we have seen in previous chapters, high dimensional data are inherently difficult to work with. In the context of supervised classification, this problem causes the number of examples needed to build a classifier to grow enormously as the dimensionality of the feature space increases. In particular, the problem of email foldering deals with natural language, and thus potentially thousands of words can be selected as features [158]. Moreover, the number of extracted words often exceeds the number of messages by more than an order of magnitude [33]. As a result, the classifier can become severely biased, especially when it is induced from small training sets [159].

Feature selection is often applied to high dimensional data as a preprocessing step in order to overcome the so-called 'curse of dimensionality' [31]. By removing irrelevant and redundant features, the classification performance and construction time can be improved.

On the other hand, in Chapter 5 we saw that email is dynamic in nature. Emails arrive in a stream over time. Users are constantly creating new folders, and letting other folders fall out of use. Even the topic associated with a folder can shift over time. Changes in the learning environment are known as *concept drift*. Concept drift scenarios require adaptive algorithms able to track such changes and adapt to them.

An additional type of concept change is presented in this chapter: *contextual concept drift*. Contextual concept drift corresponds to situations where the set of relevant features shifts over time [120]. New terms appear, other fall out of use, and, in general, the relative relevance of features changes evolves. This gives rise to *dynamic feature spaces*, which means that the features used by the classification model change over time. In order to tackle this kind of concept change, adaptive learning algorithms capable of handling changes in the feature space are desirable. Authors such as Katakis at al. [120] argue that it is useful to combine incremental feature selection methods with *feature-based learners*, that is, learning algorithms that work under evolving feature sets.

In this chapter we present ABC-DynF, an adaptive learning framework that combines the adaptive procedures proposed in the AdapPreqFr4SL framework [135] with the implementation of a dynamic feature space for a Naïve Bayes classifier in order to cope with contextual concept drift and the emergence of new categories over time. In the presented framework, data arrives to the learning system sequentially. The classification model must first make a prediction, and then the current model is updated with new data. ABC-DynF is provided with control and adaptive mechanisms that try to adjust quickly to both real concept drift and contextual concept drift, adapting to a changing feature set.

The main aim of this chapter is to evaluate how different preprocessing, control and adaptive strategies included in the ABC-DynF can affect the performance of a Bayesian-based email classification system. To this end, we have conducted experimental studies using several datasets extracted from the ENRON email corpus [32] in order to compare different settings of the ABC-DynF framework. Particularly we focused on three ways to improve the performance: *i*) dimensionality reduction of the feature space; *ii*) use of control mechanisms and adaptive methods including the implementation of a dynamic feature space to handle changing environments; *iii*) the use of an iterative algorithm called Iterative Bayes [160] for improving the probability estimators of the Naïve Bayes classifier.

This chapter is partially based on the following publication in which the author has participated:

- Carmona-Cejudo, J.M., Castillo, G., Baena-García, M., Morales-Bueno, R.: "A Comparative Study on Feature Selection and Adaptive Strategies for Email Foldering Using the ABC-DynF Framework", *Knowledge-Based Systems*. Available online 1 April 2013, ISSN 0950-7051, 10.1016/j.knosys.2013.03.006 [161]

We conclude this introduction with a brief outline of the contents of the chapter. Section 6.2 presents the ABC-DynF framework, and reviews the Naïve Bayes classifier and the Iterative Bayes procedure for parameter adaptation, as well as the adaptive and control strategies adopted to handle *concept drift* and the dynamic feature set mechanisms used by ABC-DynF. Section 6.3 presents the results of the experimental study with several datasets from the ENRON corpus. Finally, Section 6.4 provides some discussion based on the results obtained and also in the insights highlighted in related studies.

6.2 The ABC-DynF Framework: Adaptive Bayesian Classifier with Dynamic Features

This section is devoted to the ABC-DynF framework, an extension of the AdPreqFr4SL framework that combines the control strategies and adaptive procedures included in the latter with the implementation of a dynamic feature space for a Naïve Bayes classifier. In Subsection 6.2.1 we review the AdPreqFr4SL, in 6.2.2 we introduce the topic of classification in dynamic feature spaces and in Subsection 6.2.3 we discuss ABC-DynF. Finally, in Subsection 6.2.4 we explain the Iterative Bayes procedure for improved probability estimation.

6.2.1 The AdPreqFr4SL framework

The main environmental assumption that drives the design of the Adaptive Prequential Learning Framework for Supervised Learning, AdPreqFr4SL [162] is that observations do not arrive to the learning system at the same time, which allows the environment to change over time. Without loss of generality, we can assume that at each time point data arrives in batches of equal size so that we can perform model adaptation in jumps when there is some accumulated experience. Indeed, a temporal memory could be used to store the incoming examples until there are enough examples to build a batch of fixed size. The main goal is to sequentially predict the categories of the next batch.

Many adaptive systems employ regular updates while new data arrives. The AdPreqFr4SL, instead, is provided with controlling mechanisms that try to select the best adaptive actions according to the current learning goal. To this end, two performance indicators are monitored over time: the *batch error* (the proportion of misclassified examples in one batch) and the *model error* (the proportion of misclassified examples in the total of the examples that were classified using the same model). Using batches of examples and the *batch*

error as an indicator of the current performance offers the advantage to detect changes by observing the deviation from the current *batch error* value to the previous observed values without paying too much attention to outliers.

Algorithm 2 summarizes the main processes of the `AdPreqFr4SL` framework. For each batch B of examples, the current hypothesis is used to make *predictions*, the correct category is observed and the performance *indicators* (the *batch error* and the *model error*) are assessed. Then, the indicator values are used to estimate the actual systems state: $s1$-the performance is improving; $s2$ - the performance stops improving in a desirable tempo; $s3$ - a first alert of concept drift is signaled; $s4$ - there is a gradual concept change (concept drift); $s5$ - there is an abrupt concept change (concept shift); $s6$ - the performance reaches a plateau. Finally, the model is adapted according to the estimated state.

Algorithm 2 The algorithm of the the `AdPreqFr4SL`

Require: A classifier class-model $\mathcal{M}$, a dataset $\mathcal{D}$ of *i.i.d.* examples $\langle \mathbf{x}, c = f(\mathbf{x}) \rangle$ divided in batches B of M examples
Ensure: A classifier $h_{\mathcal{C}} \in \mathcal{M}$ updated at each time point
  Initialize $h_{\mathcal{C}}$ with one of the hypothesis from M
  for each batch B of m examples of $\mathcal{D}$ **do**
    for each example $\mathbf{x}$ in B **do**
      $h_{\mathcal{C}}(\mathbf{x}) \Leftarrow$ predict($\mathbf{x}$, $h_{\mathcal{C}}$)
      $f(\mathbf{x}) \Leftarrow$ getActualClass($\mathbf{x}$)
      `numIncorrect`$+ = \delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x}))$ {the 0-1 loss is used}
    end for
    indicators $\Leftarrow$ assessIndicators(`numIncorrect`, ...)
    state $\Leftarrow$ estimateState(indicators, monitoring-tools)
    adapt($h_{\mathcal{C}}$, B, state)
  end for
  end for
  return $h_{\mathcal{C}}$

Whenever it is detected that the performance continues to improve (state $s1$), the current classifier is updated using the examples from the last batch. The adaptive strategy for handling concept drift mainly consists of manipulating a short-term memory to store examples suspected to belong to a new concept. Whenever a concept drift (state $s4$) or alert (state $s3$) is signaled, the new examples are added to a short-memory. However, after signaling a concept drift, the examples of the current batch are not used to update the model

in order to force a degradation of the performance. This leads to a faster detection of a concept shift in order to build a new classifier from data that could belong to a new concept. Thus, whenever a concept shift is detected (state $s5$), the current model is discarded. In this case, a new classifier is learnt from scratch using the examples from the short-term memory. Afterwards, the short memory is cleaned for future uses. This adaptation process will continue until it is detected that the performance reaches a plateau (state $s6$). In this case we assume that it does not make more sense to continue adapting the model with new data. However, the performance will continue to be monitored. If any significant change in the behaviour is observed, then we will activate the adaptation procedures once again.

The method for monitoring concept drift is based on the Shewhart `P-Chart`. In quality control, the `P-Chart` is usually used to monitor the proportion of *non-conforming* items in a production process. In the AdPreqFr4SL, the `P-Chart` is used for monitoring the *batch error*. More details about the implementation of a `P-Chart` to control concept drift can be found in the work of Castillo and Gama [162].

The AdPreqFr4SL framework is available on-line as open source at `http://adpreqfr4sl.sourceforge.net/`.

6.2.2 Classification in dynamic feature spaces

Due to the high dimensionality of text streams and their dynamic nature, it is necessary to monitor feature relevance over time for changing the set of selected attributes when it is necessary. There are still not many algorithms in the literature able to deal with a dynamic feature space. Katakis et al [163] and Jou and Shih [12] propose the use of Naïve Bayes for this task, while Delany [164] prefers a k-NN based ensemble classifier. What both algorithms have in common is that they can learn using all the available attributes, and predict using a subset of them. This is achieved by using a subset of the features to predict posterior probabilities in the case of Naïve Bayes, and using a subset of the features to compute distances

in the case of k-NN. Another algorithm designed to work in dynamic feature spaces is FAE (*Feature Adaptive Ensemble*), proposed by Wenerstrom and Giraud-Carrier [121]). The FAE algorithm is based on a dynamically-sized ensemble of Bayesian classifiers, where each of the members of the ensemble is specialized in a subset of the attributes.

In general, Naïve Bayes-based approaches are preferred in the literature, because example-based algorithms such as k-NN need to store all the previous examples, making this kind of algorithms inefficient for large text streams [163]. Moreover, in the Naïve Bayes algorithm it is trivial to add new attributes and labels, something which is necessary when dealing with text data streams. For these reasons, we use this algorithm as the basic model the ABC-DynF, presented in the next subsection.

6.2.3 The Adaptive Bayes Classifier with Dynamic Feature Space (ABC-DynF)

We have extended the AdPreqFr4SL framework with a mechanism for tracking changes in the top-k features that allows for the implementation of a dynamic feature space. The resulting framework is called ABC-DynF (Adaptive Bayes Classifier with Dynamic Feature Space) and is published on-line as open source at `http://abcdynf.sourceforge.net/`. The ABC-DynF framework is based on the use of the *Naïve Bayes* classification model, *concept drift adaptation* procedures from AdPreqFr4SL, and the chi-squared relevance score for maintaining a *dynamic feature set* to be used by the classification model.

We have chosen to use the Naïve Bayes (NB) classifier, mainly due to its incremental nature: it only needs to accumulate sufficient statistics for calculating probabilities for the Bayes classification rule. Moreover, NB allows for a simplified implementation of the adaptive algorithms not only for incorporating new incoming data, but also for adding new features and categories over time. With

this aim, ABC-DynF maintains a set of *sufficient statistics* **T** which comprises the following elements:

1. A table $\mathtt{T_c}$ containing m counters N_j, one counter for each category c_j
2. For each feature X_i with R possible values $\{r_0, \ldots, r_{|R-1|}\}$:
   - A $m \times R$ contingency table $\mathtt{CT}_i$ containing the co-occurrence counters N_{ijk}, which represent the number of examples $\langle \mathbf{x}, c \rangle \in \mathcal{D}$ such that $X_i = r_k$, and $c = c_j$, for each possible value r_k of X_i.

Note that, in our case, R=2, since we work with a bag-of-words document representation, and thus with binary attributes.

In order to handle concept drift, ABC-DynF uses the adaptation procedures described in Subsection 6.2.1. Besides, each time a batch B of examples arrives, the following three steps are carried out sequentially by the ABC-DynF framework in order to adapt to contextual concept drift and maintain a dynamic feature space:

1. The sufficient statistics **T** are updated by incrementing the counters, using the examples from B. If new features or categories appear, they are added to the tables.
2. Based on **T**, a relevance score is calculated for each feature using the chi-squared weighting function (see Eq. 2.3) .
3. The k features with the highest score are selected as the set of top-k attributes, and will be used by the NB model when predicting the category of each example from the next batch. The k parameter is selected before the execution of the algorithm, and remains constant for all the batches.

If we look at Equation 2.3, we see that, in order to calculate the chi-square weighting function, we need to estimate not only the probability of a feature value in documents from a given class, but also complementary probabilities. Nevertheless, the set of sufficient

statistics **T** provides enough information for calculating the chi-squared function. In fact, given a category c_j, a feature X_i and a possible feature value r_k, we can calculate the number of data examples $\langle \mathbf{x}, c \rangle$ such that $X_i \neq r_k \wedge c = c_j$ by using the $\mathtt{CT}_i$ and $\mathtt{T_c}$ tables, since the number of examples such that $X_i \neq r_k \wedge c = c_j$ is equal to the total number of examples where $c = c_j$ minus the number of examples where $X_i = r_k \wedge c = c_j$. In our case, since we work with binary features, we have to store counters for the `true` value of features only, that is, for term occurrences in documents. Similarly, we can use the same contingency tables to calculate the counters for the cases where $c \neq c_j$ by adding up the results for the other categories. These counters can be used to estimate the required probabilities.

NB naturally allows to use arbitrary feature subsets, so that only the top-k features are used for classification. This is achieved by constraining the set of features used in the NB classification rule:

$$category(\mathbf{x}) = \underset{c_j}{\operatorname{argmax}} P(c_j) \prod P(X_i = w_i)|c_j), X_i \in topk \quad (6.1)$$

where w_i represents the value of the X_i feature in document $\mathbf{x}$ and $topk$ is the set of the k most relevant features. That is, when learning or updating the NB model, all the existing features are updated, but when predicting the category of a new example, only the top-k features are used in the computation.

The ABC-DynF framework allows the user to choose to monitor concept drift or not. In addition, the user can also opt to maintain a dynamic feature space or not, in which case the features selected using the first batch of examples remain unchanged. Both parametrizations produce different adaptation strategies listed in Table 6.1.

6.2.4 Improving the Probability Estimates by Iterative Bayes

In order to improve the performance of Naïve Bayes its bias and/or variance has to be reduced by obtaining better probability estimates $\hat{P}(X_i = x_i \mid c_j)$ and/or $\hat{P}(c_j)$. The Iterative Bayes, proposed by J. Gama [160], improves the predictive distribution $P(C \mid \mathbf{x})$ associated with each example $\mathbf{x}$ by adjusting the estimates $\hat{P}(X_i = x_i \mid c_j)$ of the conditional probabilities.

Suppose that during a iteration we process an example $\mathbf{x}^{(l)}$ which belongs to the category $c_{\texttt{obs}}$ and that it is classified by the current classifier $h_{\mathcal{NB}}$ as belonging to $c_{\texttt{pred}}$. The Iterative Bayes updating procedure works as follows. An increment, `delta`, proportional to the difference $1 - P(c_{\texttt{pred}} \mid \mathbf{x})$ is computed. If the example is incorrectly classified, that is, $c_{\texttt{pred}} \neq c_{\texttt{obs}}$, then `delta` is multiplied by -1 in order to be a negative quantity. For each contingency table $\texttt{CT}_i$ associated to the feature X_i, the increment `delta` is added to the entry where $C = c_{\texttt{pred}}$, and then it is proportionally subtracted to the remaining entries. For avoiding zero or negative counters, the counters never goes below 1. The iterative procedure finishes when a stopping criterion is met. The experimental evaluation of Iterative Bayes in the work of Gama [160] has shown consistent reductions in the error rate. Using the bias-variance decomposition of the error, it was demonstrated that the reduction of the error is mainly due to a reduction on the bias component.

An important advantage of Iterative Bayes is that it lends itself directly to incremental learning. In [165] Adaptive Bayes was introduced, an incremental version of Iterative Bayes, that can also work in an on-line learning framework. The rationale is that after seeing a new example, the corresponding counters are first incremented and

	Concept Drift Monitoring			
	No		Yes	
Update features	No	Yes	No	Yes
	$Adapt_{00}$	$Adapt_{01}$	$Adapt_{10}$	$Adapt_{11}$

TABLE 6.1: Adaptation strategies for AdPreqFr4SL

then Iterative Bayes is executed in order to increase the probability on its current category.

6.3 Experimental Evaluation

In this section, we describe the experiments we have carried out with ABC-DynF. In Subsection 5.3.3 we present the datasets we have used. Then, we introduce our experimental settings in Subsection 6.3.1. Finally, in Subsection 6.3.2 we specify the studies which are the main focus of this chapter, and the conclusions that can be drawn from them. For the sake of replicability, we have uploaded the experimentation scripts to the project web page (`http://abcdynf.sourceforge.net/`).

6.3.1 Experimental Settings

We have carried out several experiments using 10 datasets from the Enron corpus (see Section 5.3.3 in Chapter 5). We use a binary *bag-of-words* model for email representation. All features were obtained from the message subject and the email body (from the first 5 or 10 lines, or from the whole body). After stop-words removal and stemming, we obtain a bi-dimensional array of m messages and n binary features for each email user.

In this chapter we do not aim at selecting the best weighting function, since studies already exist on this topic, e.g. the work of Forman [33], in which it is shown that the weighting function used by ABC-DynF, chi-square, is a reasonable function for text classification.

We have an email stream for each particular dataset in the Enron corpus, which we have divided into batches of 50 messages. The first batch is used to build an initial Naïve Bayes classifier, which is updated for each new batch by the ABC-DynF framework.

In order to evaluate the classification performance of the model, we have used two different evaluation measures: the *percentual classification error* and the *F1 score*. Percentual classification error is calculated as follows:

$$error = \frac{\#wrong\_predictions \times 100}{N} \tag{6.2}$$

that is, the number of wrong predictions between the total number of examples, expressed as a percentage.

The other measure we have used is the F1 measure. This measure is calculated for each category l as the harmonic mean of the precision (π) and recall (ρ) for that category:

$$F_1(l) = \frac{2\pi(l)\rho(l)}{\pi(l) + \rho(l)} \tag{6.3}$$

where l represents a category. Precision and recall are complementary measures that are calculated as follows:

$$\pi(l) = \frac{TP(l)}{TP(l) + FP(l)}, \rho(l) = \frac{TP(l)}{TP(l) + FN(l)} \tag{6.4}$$

where $TP(l)$, $FP(l)$ and $FN(l)$ represent the number of true positives, false positives and false positives for category l. Note that the π, ρ and the F1 measure are local to a given category. A global F1 is calculated by macro-averaging the measures, that is, calculating the average of the individual F1 measures for each category.

The evaluation has been carried out following a prequential-like approach: for each new incoming batch, the performance of the model is evaluated using some of the aforementioned evaluation measures, and then the examples from that batch are used to update the model. The final result corresponds to the average of the classifications for all the batches. Note that this is a pessimistic performance evaluator [114], since the evaluations for the first batches have the same weight as the evaluations for the last batches, that are probably

more representative of the performance of the learning model, since they have received more information.

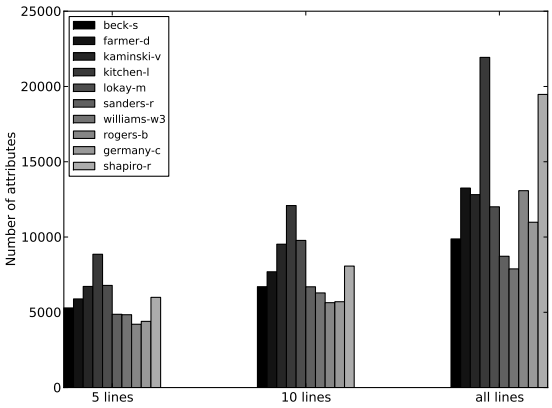
After our experiments, we have used non-parametric statistical tests in order to confirm to what extent the observed differences in performance are statistically significant, and not just statistical artifacts. To this end, we have followed the guidelines proposed by Demšar for statistical hypothesis testing [55], who advocates for the use of non-parametric tests in data mining experiments. More specifically, we have applied the Friedman test, using the statistic derived by Iman and Davenport, since the original Friedman statistic is undesirably conservative [55]. When comparing only two related samples (for example, comparing the results with or without Iterative Bayes), we have used the Wilcoxon test instead [54]. A good overview of these tests, including its associated formulas, can be found in the aforementioned work of Demšar [55]. In all of our tests, we use a significance level of $\alpha = 0.05$. When the Friedman test finds significant differences, it is useful to find out which is the source of this difference, that is, which of the compared data samples are statistically different. To this end, it is necessary to use post-hoc tests. In our case, we have used Finner's non-parametric procedure [166] as the post-hoc test for our Friedman tests.

6.3.2 Results and analysis

Tables C.4, C.5, C.6 and C.7 at the Appendix show the final classification accuracies obtained for different configuration settings for feature selection processes and adaptive strategies implemented in the ABC-DynF framework.

We can make some preliminary observations on the obtained results. First, there is a clear relationship between the accuracy for each user and the entropy of the folders (see Table 5.1 in Chapter 5). Specifically, the best accuracies are obtained for users with lower entropies, such as `williams-w3`. We can observe that accuracies are inversely related to the number of folders, but not to the total number of messages. Better accuracies are obtained when the average number

FIGURE 6.1: Number of generated features using the first five, ten or all lines



of messages by folder is bigger, since more examples are available for the learning algorithms. These observations show that the best results are obtained for datasets with a very reduced number of dominant categories containing a large number of examples.

Based on the results of Tables C.4, C.5, C.6 and C.7, we have carried three studies aimed at comparing the classification performance of classifiers learnt using different settings for: *i*) *feature selection and vocabulary size reduction*, using five, ten or all the lines of the email body, as well as using different numbers of features; *ii*) probability estimators, to study the impact of improving the classification model by using better probability estimations, and *iii*) *adaptive strategies*, in order to study the effect of the implementation of concept drift monitoring and the use of a dynamic feature set.

First study: reducing the size of the feature space

The goal of the first study is to compare how different strategies for dimensionality reduction affect the classification performance. We explore two types of strategies: *i*) using only a limited number of lines and *ii*) using a limited number of features.

The first strategy is related to the indexing phase. Previous studies have pointed out that the most relevant features for email classification are those extracted from the subject. The email body usually contains more redundant information than the subject line. Therefore, the mail subject is more likely to contain keywords [150]. Thus, we could significantly reduce the number of features if instead of processing all the lines of the message body we use only the subject and a limited number of the first lines of the body. In this study, we have compared the results when using 5, 10 or all the lines to select classification features. As depicted in Figure 6.1, increasing the number of used lines results in an increase of the number of features, which is an intuitive result.

The second strategy consists in applying feature selection, using the chi-squared weighting scheme in order to select the most relevant k features. In our study we evaluated the classification performance obtained using the feature set sizes used in a previous work [106]: 250, 300 and 350.

In order to illustrate the dynamism of the evolving feature space, in Table 6.2 we show the percentage of coincidence of the top-k feature set between every two consecutive batches, using features extracted from the first 5, 10 or all the lines. It can be observed that this coincidence is in general lower when using the complete message than when only 5 or 10 lines are used. This means that keeping a dynamic feature set is more important when larger sections of email messages are used, or, in general, with larger document sizes. It also suggests than the features appearing in the first lines of the email are more stable over time, due to the fact that they are more informative of message contents. The effect is more exaggerated in some datasets, such as *sanders-r* and *germany-c*, than in others, such as *beck-s* or *lokay-m*.

In order to evaluate the effect of using a different number of lines on performance, we have performed Friedman statistical tests to determine whether there is a statistically significant difference when five, ten or all the lines of the email body are used for feature extraction, under the null hypothesis that this has no effect on performance.

Dataset	5 lines	10 lines	All lines
beck-s	92.63%	91.5%	91.94%
farmer-d	94.45%	94.63%	92.78%
kaminski-v	97.18%	95.75%	96.42%
kitchen-l	93.14%	92.49%	91.59%
lokay-m	91.79%	90.48%	90.35%
sanders-r	87.79%	85.76%	78.75%
williams-w3	94.22%	94.24%	93.94%
rogers-b	91.07%	91.53%	88.37%
germany-c	88.13%	87.31%	82.76%
shapiro-r	87.08%	86.09%	84.48%

TABLE 6.2: Percentage of coincidence of the top-k feature set between two consecutive batches, using the first 5 or 10 lines or the complete mail message to obtain features

Thus, we have compared the performance of three classifiers resulting after applying three different configuration settings during the process of feature extraction over all the datasets, keeping fixed the settings for feature selection and adaptive strategies.

The results are in Table 6.3. It can be seen that, in most cases, the p-value of the Friedman test is larger than the significance level α (0.05). This means that the null hypothesis cannot be rejected. Moreover, in the cases where the null hypothesis is rejected, we find that using all the lines is not the best option from the point of view or performance (see the complete results in Tables C.4, C.5, C.6 and C.7).

Regarding the number of features, we have also carried out Friedman tests to analyze the effect of changing this parameter. The results can be seen in Table 6.4. We can see that the null hypothesis can almost never be rejected, which indicates that using a different number of features does not have a significant impact on performance.

To sum up, the results of our experimentation reveal that using more features or larger sections of the messages to extract features does not necessarily improve the performance. This indicates that it is possible to use less features and save computational resources without compromising performance.

Strategy	N. attrs	I.B.	p-value (error)	p-value (F1)
$Adapt_{11}$	250	True	0.0581	0.0611
$Adapt_{11}$	250	False	0.4066	0.8276
$Adapt_{11}$	300	True	0.0608	0.1225
$Adapt_{11}$	300	False	0.0821	0.1496
$Adapt_{11}$	350	True	0.2725	0.3679
$Adapt_{11}$	350	False	0.0821	0.4966
$Adapt_{10}$	250	True	0.0608	0.0665
$Adapt_{10}$	250	False	**0.0058**	**0.0013**
$Adapt_{10}$	300	True	0.0608	0.4966
$Adapt_{10}$	300	False	0.0608	0.1319
$Adapt_{10}$	350	True	**0.045**	0.0608
$Adapt_{10}$	350	False	0.1225	0.0608
$Adapt_{01}$	250	True	**0.0074**	**0.0283**
$Adapt_{01}$	250	False	0.1496	0.1988
$Adapt_{01}$	300	True	**0.0247**	0.0611
$Adapt_{01}$	300	False	0.0672	0.1988
$Adapt_{01}$	350	True	0.2319	0.2725
$Adapt_{01}$	350	False	0.0608	0.2725
$Adapt_{00}$	250	True	0.3012	0.1905
$Adapt_{00}$	250	False	**0.0071**	**0.0202**
$Adapt_{00}$	300	True	0.4066	0.332
$Adapt_{00}$	300	False	0.0821	0.0608
$Adapt_{00}$	350	True	0.2019	0.3012
$Adapt_{00}$	350	False	0.1794	0.4516

TABLE 6.3: Results of the Friedman test to check if changing the number of email lines from which the features are extracted affects the performance of the learning model. Both the error and the F1 score have been used. Significant differences have been marked in bold

Second study: using Iterative Bayes

In the second set of experiments we study the impact of improving the classification model itself. More precisely, we use the Iterative Bayes procedure to improve the probability estimates. We use the Wilcoxon test to find out if the impact on performance is statistically significant. The p-value results of the Wilcoxon tests are provided in Table 6.5.

From the results, we can conclude that using Iterative Bayes is clearly beneficial, especially when concept drift monitoring is not used (that is, with $Adapt_{00}$ and $Adapt_{01}$). When monitoring is used (this is, with $Adapt_{10}$ and $Adapt_{11}$), the Wilcoxon tests do not always reject the null hypothesis. Nevertheless, the complete results in Tables C.4, C.5, C.6 and C.7 show that, as a rule, Iterative Bayes

Strategy	N. lines	I.B.	p-value (error)	p-value (F1)
$Adapt_{11}$	5	True	0.2093	0.067
$Adapt_{11}$	5	False	0.4516	0.2847
$Adapt_{11}$	10	True	0.2019	**0.0108**
$Adapt_{11}$	10	False	0.9048	0.6703
$Adapt_{11}$	all	True	0.9747	0.4966
$Adapt_{11}$	all	False	0.6703	0.9747
$Adapt_{10}$	5	True	0.5669	0.2163
$Adapt_{10}$	5	False	0.1822	0.5899
$Adapt_{10}$	10	True	0.9048	0.8233
$Adapt_{10}$	10	False	0.4966	0.9048
$Adapt_{10}$	all	True	0.3012	0.8357
$Adapt_{10}$	all	False	0.2019	0.971
$Adapt_{01}$	5	True	0.4966	0.1773
$Adapt_{01}$	5	False	0.2847	0.1096
$Adapt_{01}$	10	True	0.7165	0.2725
$Adapt_{01}$	10	False	0.7165	0.9747
$Adapt_{01}$	all	True	0.2725	0.2093
$Adapt_{01}$	all	False	0.4516	0.4966
$Adapt_{00}$	5	True	0.5669	0.1482
$Adapt_{00}$	5	False	0.1635	0.5488
$Adapt_{00}$	10	True	0.4966	0.723
$Adapt_{00}$	10	False	0.1496	0.6144
$Adapt_{00}$	all	True	0.3872	0.7165

TABLE 6.4: Results of the Friedman test to check if changing the number of selected features affects the performance of the learning model. Significant differences have been marked in bold

improves the results. Thus, we can conclude that improving the classification algorithm (in this case, by improving the probability estimates of Naïve Bayes) has a deeper impact than augmenting the number of features.

Third study: adaptive strategies

In the third study, we compare the four adaptation strategies listed in Table 6.1. Unlike in the previous studies, we restrict the number of features to 250 and the number of lines to 5, given that our results in Subsection 6.3.2 do not justify using more features, or more lines to extract these features. We have selected to use Iterative Bayes in this experiment, given the results of Subsection 6.3.2.

Our objective is to test whether there is a statistically significant difference when using different adaptation strategies. The results

Strategy	N. attrs.	N. lines	p-value (error)	p-value (F1)
$Adapt_{11}$	250	5	**0.0367**	0.0926
$Adapt_{11}$	250	10	**0.0125**	**0.0156**
$Adapt_{11}$	250	all	0.5076	0.2207
$Adapt_{11}$	300	5	0.0593	0.0837
$Adapt_{11}$	300	10	0.0593	**0.0141**
$Adapt_{11}$	300	all	**0.0469**	**0.0093**
$Adapt_{11}$	350	5	**0.0284**	**0.0367**
$Adapt_{11}$	350	10	0.1141	**0.025**
$Adapt_{11}$	350	all	0.0593	**0.0051**
$Adapt_{10}$	250	5	**0.0218**	**0.036**
$Adapt_{10}$	250	10	0.2845	0.1122
$Adapt_{10}$	250	all	**0.0367**	**0.0367**
$Adapt_{10}$	300	5	0.0745	0.2207
$Adapt_{10}$	300	10	0.1688	0.1099
$Adapt_{10}$	300	all	0.2026	0.1731
$Adapt_{10}$	350	5	0.2026	0.2306
$Adapt_{10}$	350	10	**0.0284**	0.0523
$Adapt_{10}$	350	all	0.4446	0.3329
$Adapt_{01}$	250	5	**0.0284**	**0.025**
$Adapt_{01}$	250	10	**0.0093**	**0.0105**
$Adapt_{01}$	250	all	0.0745	**0.0385**
$Adapt_{01}$	300	5	**0.0367**	**0.0382**
$Adapt_{01}$	300	10	**0.0367**	**0.0209**
$Adapt_{01}$	300	all	**0.0051**	**0.0093**
$Adapt_{01}$	350	5	**0.0218**	**0.0214**
$Adapt_{01}$	350	10	**0.0284**	**0.0223**
$Adapt_{01}$	350	all	**0.0051**	**0.0069**
$Adapt_{00}$	250	5	**0.0166**	**0.0254**
$Adapt_{00}$	250	10	**0.0284**	**0.0284**
$Adapt_{00}$	250	all	**0.0051**	**0.0069**
$Adapt_{00}$	300	5	**0.0166**	**0.0209**
$Adapt_{00}$	300	10	**0.0469**	**0.0176**
$Adapt_{00}$	300	all	**0.0367**	**0.0499**
$Adapt_{00}$	350	5	**0.0218**	**0.0166**
$Adapt_{00}$	350	10	**0.0284**	**0.0117**
$Adapt_{00}$	350	all	**0.0284**	**0.0367**

TABLE 6.5: Results of the Wilcoxon test to find out if using Iterative Bayes significantly improves the performance of the learning model. Significant differences are marked in bold

for the F1 score and error, together with the average rankings, are given in Tables 6.6 and 6.7.

The Friedman test rejects the null hypothesis (yielding a p-value of 0.00667 for F1 and 6.03×10^{-9} for error), which means that there is a statistically significant difference in performance when different adaptation strategies are used.

Since the null hypothesis is rejected, we have carried out a post-hoc analysis to find out the causes of this difference. To this end, we

Dataset	$Adapt_{00}$	$Adapt_{10}$	$Adapt_{01}$	$Adapt_{11}$
beck-s	0.611	0.624	0.638	0.649
farmer-d	0.552	0.551	0.557	0.565
kaminski-v	0.638	0.654	0.675	0.698
kitchen-l	0.512	0.604	0.512	0.604
lokay-m	0.581	0.615	0.58	0.615
sanders-r	0.603	0.619	0.664	0.703
williams-w3	0.909	0.919	0.909	0.919
rogers-b	0.643	0.636	0.643	0.636
germany-c	0.692	0.739	0.698	0.741
shapiro-r	0.57	0.615	0.551	0.61
Average ranking	3.35	2.4	2.75	1.5

TABLE 6.6: F1 score for different adaptation strategies

Dataset	$Adapt_{00}$	$Adapt_{10}$	$Adapt_{01}$	$Adapt_{11}$
beck-s	55.68	52.18	52.04	49.32
farmer-d	54.6	54.333	53.973	52.773
kaminski-v	52.763	49.195	47.763	43.186
kitchen-l	71.333	62.024	71.333	62.024
lokay-m	51.947	48.187	51.92	48.187
sanders-r	60.057	52.343	51.029	42.971
williams-w3	14.852	13.148	14.852	13.148
rogers-b	44.831	43.477	44.769	43.446
germany-c	43.091	39.182	41.727	38.545
shapiro-r	65.138	60.966	64.828	59.897
Average ranking	3.9	2.25	2.7	1.15

TABLE 6.7: Percentual error for different adaptation strategies

have launched pair-wise comparisons of the best adaptive strategy ($Adapt_{11}$) with the others, applying Finner's procedure [166]. For both the F1 measure and the percentual error, we find that there is a significant difference between $Adapt_{11}$ and $Adapt_{00}$ (with p-value = 0.017), and between $Adapt_{11}$ and $Adapt_{10}$ (with p-value = 0.03). On the other hand, no significant difference was detected between $Adapt_{11}$ and $Adapt_{01}$.

From these results, we can conclude that the the best adaptation strategy is to use concept drift monitoring and update the feature set over time. The impact of updating the feature space is greater than the impact of concept drift monitoring. This leads us to the conclusion that the effect of concept drift is limited, probably because we do actually face *virtual* concept drift. This can occur when the order of training instances is skewed, for example, when different types of instances are not evenly distributed over the training sequence [135].

Dataset	Always	When stop improving	When reconstructing
beck-s	0.6417	0.6500	0.6358
farmer-d	0.5638	0.5712	0.5720
kaminski-v	0.7019	0.6985	0.6970
kitchen-l	0.5847	0.4975	0.4975
lokay-m	0.6018	0.5787	0.5787
sanders-r	0.6899	0.6338	0.6338
williams-w3	0.9069	0.9090	0.9088
rogers-b	0.6343	0.6435	0.6473
germany-c	0.7392	0.7016	0.7133
shapiro-r	0.6020	0.5941	0.5941
Average ranking	1.7	2.1	2.2

TABLE 6.8: F1 results for different feature updating strategies

Dataset	Always	When stop improving	When reconstructing
beck-s	49.8000	50.4200	50.9000
farmer-d	52.6400	52.5467	52.4667
kaminski-v	43.4746	43.9915	43.9746
kitchen-l	62.5000	72.8571	72.8571
lokay-m	50.0533	52.3733	52.3733
sanders-r	45.6000	53.1429	53.1429
williams-w3	14.4815	14.2963	14.3333
rogers-b	43.7846	44.2154	44.5539
germany-c	38.5455	41.8182	41.0909
shapiro-r	59.3793	60.8276	60.8276
Average ranking	1.4	2.3	2.3

TABLE 6.9: Error results for different feature updating strategies

Another interesting aspect to study is *when* to update the feature set. In principle, the feature space could be updated for every batch, so that when a new batch of examples arrives, the most relevant features from the beginning until the last batch are used. Nevertheless, there are other possibilities. One of them is to update the features only in case that the model performance stops improving. An additional strategy would be to update the features only when the learning model is rebuilt due to a concept shift, which would mean that the feature set is static, but when the model has to be rebuilt, it uses the newest top-k features.

We have launched experiments to measure the performance of these strategies, obtaining the results in Tables 6.8 and 6.9.

The results show that the best option is to always update the features. However, the Friedman test does not find statistically significant differences between always updating, updating when the model

stops improving or updating when the model is reconstructed. The p-values are 0.52 for the F1 results and 0.058 for the error results, which means that we cannot categorically affirm that updating the features for each batch is always the best option, although it is the best option in average.

As mentioned in Section 6.2, the base learner is reconstructed from scratch when a concept shift is detected. This is the most expensive step in the ABC-DynF framework, so it is interesting to analyze to how often is the model has to be reconstructed. Reconstructing a NB model involves a single pass over the training instances in order to calculate the sufficient statistics. Therefore, it depends on the number of instances and features. We have included relevant statistics in Table 6.10, where we show the percentage of batches where the model has to be reconstructed, and how many batches in the short-term memory are used for model reconstruction in average. These results are calculated using 250 features, 10 lines from each email, concept drift monitoring and Iterative Bayes. The results are different for each dataset. For users such as `williams-w3` and `rogers-b`, no concept shift is signaled. This means that we face a virtual concept change, due to the emergence of new categories, as we previously analyzed. The model does not have to be reconstructed, although the new categories have to be incorporated, and the learning models have to be incrementally updated. On the other hand, there are some datasets, such as *kaminski-v* or *lokay-m*, where concept drift is a relevant problem. For such datasets, the model has to be reconstructed more frequently. In short, the number of reconstructions from scratch depends on the nature of the occurring concept changes, which in turn depends on each specific dataset.

6.4 Discussion

As previously stated, the problem of email foldering is more difficult than *spam filtering*, since we typically have to deal with a very high number of categories. Therefore, it is important to select a subset of relevant features [167]. The skewness of the email data is another

Author	Pct. Rebuilds	Batches used to rebuild (avg.)
beck-s	2.0%	2.0
farmer-d	1.0%	1.0
kaminski-v	6.36%	2.98
kitchen-l	2.38%	3.36
lokay-m	4.0%	1.0
sanders-r	2.86%	2.80
williams-w3	0.0%	0.0
rogers-b	0.0%	0.0
germany-c	4.55%	1.54
shapiro-r	1.72%	1.74

TABLE 6.10: Percentage of model reconstructions with respect to total number of batches, and average length of the short-term memory (in number of batches) used to reconstruct the model

factor that contributes to make this problem still very challenging. That is one of the reasons why spam detection has received considerably more attention in the literature than the more difficult problem of email foldering [156].

In the previous work of Bekkerman [144], it is shown that better accuracies are obtained for datasets with one or two dominant folders. Our results are in concordance with this observation: better accuracies are obtained for email users with a few dominant folders, such as `williams-w3`. Nevertheless, for classifiers such as Naïve Bayes, the lack of balance between the number of email messages belonging to each category might derive in some overfitting of learned parameters [156]. Learning from skewed data has attracted attention recently, and applications include direct marketing, fraud detection, and text categorization [168].

Email is an example of semi-structured data, since words can be extracted either from the body or from the headers, specifically from the subject. As mentioned before, words from the subject have a higher quality as features, since they carry less redundancy. Nevertheless, Diao et al. [169] show that using exclusively words from the subject can result in important performance degradation. In this respect, Manco et al. [150] propose giving more weight to words appearing in the subject line. Since we use a binary bag-of-words

representation, it would not make sense in our case to use weighting mechanisms. Instead of this, we simply include words from the subject and from the body.

In this chapter, we have experimentally studied the influence of using features from different numbers of lines on the learning methods. Using all the lines in each message would be justifiable if a significant difference in performance could be found. Otherwise, only a reduced number of lines should be used, since this kind of optimization would result in reducing processing time. To the best of our knowledge, this is the first time this optimization is studied in the context of email foldering. Our results show that, in fact, it is justifiable to use only the five or ten first lines of each message, since this does not affect the classification performance in a significant way.

We have used the chi-squared weighting function for feature selection. It has been experimentally found that chi-squared is in general one of the most effective functions for feature selection in the domain of text mining [170]. According to this work, chi-squared favours common terms, uses categories information, and takes into account absent terms. Furthermore, results show that the chi-squared values for a given feature are highly correlated with the values resulting from using functions such as Document Frequency and Information Gain. Different functions such as Mutual Information yield poor performances in text domains.

We have shown that the optimal number of features to use is dataset-dependent, which means that no general recommendation can be given on how many features to select in this specific problem. Surprisingly, in many of the previous works on the Enron dataset [32, 144, 156] feature selection techniques are not applied. However, good feature selection techniques are crucial in this problem. As pointed out in the work of Kiritchenko and Matwin [171] *Naïve Bayes may be quite sensitive to the number of features partly because of the violation of the independence assumption*, providing experimental evidence. Rennie [38] proposes a feature selection method based on eliminating old features, while Clark et al.[172] propose to use a fixed number of features. Li et al. [173] use a fixed threshold instead.

Regarding classification models, Bayesian classifiers are among the most popular classifiers used in email classification and the more general problem of text categorization [38], [174]. These classifiers are usually generated by an explicit underlying probabilistic model, which provides the probability of being in each category rather than a simple classification. We choose Naïve Bayes as our email classifier model mainly due to its incremental nature thus allowing us an easy implementation of adaptive algorithms in order to incorporate new features and categories over time.

Our aim is not to defend the use of Bayesian classifiers over other paradigms, since past research in general favours other models such as Support Vector Machines (see for example the work of Yu and Zu [174]). Nevertheless, we do defend the necessity for adaptation in this domain, regardless of the classifier used. Naïve Bayes classifiers has the advantage of being inherently incremental, which makes it easier to implement ad-hoc solutions for incremental feature addition [120]. They make it possible to incrementally add new categories and features, which makes them ideal for our problem. Furthermore, Naïve Bayes-based approaches are generally preferred in the literature to other models that accept a dynamic feature space, such as k-NN, because they need to store all the previous examples, making them inefficient for large text streams [120]. While other classification methods such as Support Vector Machines could probably provide a better classification performance, they are not so well suited to contextual concept drift scenarios, since they cannot incrementally include new features.

Bayesian models in general, and Naïve Bayes in particular, can improve their performance if the probability estimates they use is improved. To this end, in chapter work we have use the Iterative Bayes procedure. Gama [160] demonstrated that this procedure reduces the bias component of the error. Our results confirm that this procedure improves the obtained results. A corollary of this observation is that improving the classification model in an online classification framework is more important than fine-tuning the size of the feature set.

Our results show that adaptive mechanisms effectively improve the performance of email classifiers. Dynamic data streams are subject to changes in the underlying data distributions, that is, to concept changes. The learning models have to be able to adapt to these changes, because the knowledge extracted from older examples may not be valid anymore. It is thus necessary to detect such concept drifts and react accordingly. In our case, the results confirm this observation. The use of a dynamical feature space has a larger effect on performance than concept drift monitoring, which suggests that we are facing virtual concept drift: the examples arrive from a skewed distribution over time. We refer to the work of Žliobaitė [118], which provides a panoramic view of research on learning under concept change.

Finally, there is a need for feature selection mechanisms able to take into account changes in the relevance of features during text data streams. Katakis et al. [120] show that incrementally updating features is important for real-world textual data streams, because, in most applications, the distribution of data changes over time. More specifically, they propose the coupling of an incremental feature ranking method and an incremental learning algorithm. The predictive power of features changes over time, since new features, such as new terms in our case, appear continuously and old ones fall out of use. Therefore, incremental learning algorithms have to work in combination with incremental feature selectors in the domain of text mining. Works on classifiers that accept a dynamic feature set are still rather scarce. An initial approach was proposed by Katakis et al. [120], comprising two interconnected components: `a)` an incremental feature ranking method using the chi-square weighting function, and `b)` an incremental learning algorithm (based on Naïve Bayes) that accepts dynamic feature spaces. A drawback of this approach is that the weight for every term has to be recomputed for each new document. Some partial solutions have been proposed, for instance re selecting features periodically [164], or reelecting features only when missclassifications occur [12]. In our work we have experimented with different alternatives: updating for every batch, updating when the model performance is not improving, or updating when a concept shift has occurred and the model has to be rebuilt.

In average, the best strategy was to update for every batch, although this is not the case for every individual dataset.

Chapter 7

Online Attribute Weighting for Document Vectorization and Keyword Retrieval

7.1 Introduction

The goal we set in this chapter is the efficient automatic selection of keywords that offer an accurate description of the contents of the documents in a *data stream* scenario, where there is not a fixed dataset, but an unlimited stream of documents that arrive continuously.

As we introduced in Chapter 2, the field of Information Retrieval (IR) provides a series of functions, the so-called *weighting functions*, which are used to assign numeric values to each term in a document according to its relevance. These functions represent a tool for term ranking and keyword selection. Those words which occupy the highest rank in a given document can be considered as its keywords. The simplest weighting function is the *term frequency* function, which assigns higher weights to terms occurring more frequently. Its main

weakness is that terms that occur frequently in a document are always assigned a high weight, even if they also appear frequently in other documents. As a consequence, words such as prepositions are given a high weight despite their low semantic relevance. A popular alternative is the TF-IDF (term frequencyinverse document frequency) function [175], whose rationale is to assign higher weights to terms that appear frequently in a given document but rarely in all the others. A similar weighting function, based on the same principles although slightly more complex than TF-IDF, is Okapi BM25 [176].

Keyword extraction has been successfully used for document indexing and retrieval, as well as document summarization [177]. Manually annotated keywords are often not available beforehand, which makes it necessary to develop methods for automatic keyword extraction [178]. The first approaches to keyword extraction were of heuristic nature, using features such as word and phrase frequency, as in the work of Luhn [179], the position of terms in the text, as proposed by Baxendal [180] or occurrence of key phrases, as proposed in the work of Edmundson [181]. Unsupervised [182] and supervised [183] machine learning algorithms have also been used as tools for keyword detection. In this chapter, we use a different approach: the application of weighting functions to keyword extraction. This strategy has been previously employed in the work of Brutlag and Meek [148], who use TF-IDF as a tool to identify keywords in the domain of email and discuss its inherently incremental nature. Keyword extraction based on weighting functions has also been recently used in automatic generation of personalized annotation tags for Twitter users [184]. More sophisticated approaches use semantic analysis and external knowledge for document summarization, as in the work of Guo and Zhang [185]. However, such techniques are beyond the scope of this thesis. We propose an approach to keyword extraction in the absence of external knowledge, using only document content. Although using external knowledge could certainly improve results, it is not always available and it can be costly to obtain.

A graphical alternative to keyword lists for document summarization are *word clouds* [186]. A word cloud is based on the idea of showing the importance of each word by using different fonts, sizes or colors. An alternative to traditional word clouds, where all words are used, consists in selecting only the most relevant terms of each document according to a relevance metric. That is, we are only interested in the words that best describe the contents of each document. An intuitive approach for constructing word clouds for textual datasets organized by categories is to create a cloud for each category. Word clouds have been popularized by some of the first Web 2.0 sites, such as Flickr, Technorati, or Blogscope [187]. They provide a useful tool for improving the user experience when surfing web pages, as shown in the work of Champin et al. [188].

When calculating weighting functions in a stream of documents, it is necessary to maintain counters for each extracted term. The term weights in the current document depend on the terms in the previous documents, so they have to be calculated incrementally. As a result, there is a large number of different terms, which leads to a high space complexity. This problem becomes even more challenging if n-grams are used [189]. This is due to the fact that weighting functions are based on counters that store sufficient statistics about the terms. Thus, when weighting functions are used in data streams, several counters have to be stored in order to provide item counts. This requirement involves linear space complexity, something which is prohibitive in data stream scenarios [13]. In the literature, we can find solutions to this problem which consist in using only a part of the document in order to reduce vocabulary size. For instance, Brutlag and Meek suggest using only email headers when extracting keywords [148], while HaCohen-Kernel suggests extracting keywords only from abstracts [190]. There are, however, two drawbacks associated with this kind of solutions. The first one is that important semantic information can be missing if large portions of the document are ignored [191]. The second drawback is that they consider only exact solutions, that is, exact term counts, something which eventually leads to the complexity problems we have mentioned. For that reason we conclude that approximate solutions that use sublinear space are necessary when dealing with data streams [192].

In data stream scenarios, data elements are analysed only once and are not stored for future reference. There is a growing need to develop new techniques to cope with high-speed streams and deal with the space requirements we have discussed. Problems studied in this context include frequency estimation and top-k queries, among others [192]. Applications of such techniques include network traffic monitoring, network intrusion detection, sensor networks, fraudulent transaction detection or financial monitoring [193]. The earliest streaming algorithms to solve this type of queries can be traced back to the works of Munro and Paterson [194], who provided an analysis of the computational requirements of selecting and sorting in data streams, and Misra and Gries, authors of the *Frequent* algorithm [195]. The interest in streaming algorithms was boosted by some influential papers in the late 90s, such as the work of Alon et al. on the space complexity of approximating frequency moments [196].

Algorithms for frequency estimation and top-k lists in data streams can be divided into two main families: *counter-based* and *sketch-based* algorithms. Counter-based algorithms require less memory, but provide frequency estimation only for the most frequent elements, while sketch-based algorithms estimate the frequency of all the items. In this chapter, we opt to use sketch-based algorithms instead of counter-based ones because, as previously mentioned, they can estimate the frequency of all the elements, not only the most frequent ones, which is important for estimating weighting functions. Count-based and sketch-based algorithms are discussed in Subsection 7.2.1.

The use of Information Retrieval weighting functions to process non-batch textual data sources such as data streams is not new. For example, Brants et al. presented a model for new event detection based on incrementally updating term weights that allows using different TF-IDF models for different data sources [197]. This idea is also explored in the work of Kumaran and Allan, who propose a system for new event detection that makes use of incremental TF-IDF [198]. In the more recent work of Abe and Tsumoto [199], TF-IDF is used for calculating the importance of terms in order to incrementally detect temporal trends.

The main contribution of this chapter is a solution based on sketching techniques to efficiently approximate the TF-IDF and BM25 weighting functions in data stream scenarios, as well as to maintain a list of the top-k most relevant terms for each document category. We apply our proposal to two scenarios. The first one is online *keyword extraction*, i.e., assigning the most relevant keywords to each document from a stream. The second one is online construction of *word clouds*, which consists in maintaining a list of terms that represent each category from the data stream as new documents arrive. In our experiments, we compare the estimates of term weights obtained with our method with the results obtained using exact counts, using two standard Text Mining and Information Retrieval datasets. We also analyse and compare different configurations of the sketches used for top-k lists and frequency estimation.

This chapter is partially based on the following publications in which the author has participated:

- Baena-García, M.; Carmona-Cejudo, J.M.; Castillo, G., Morales-Bueno, R.: "Term Frequency, Sketched Inverse Document Frequency", *11th International Conference on Intelligent Systems Design and Applications (ISDA)* [200]
- Carmona-Cejudo, J.M.; Baena-García, M.; Castillo, G., Morales-Bueno, R.: "Online Calculation of Word-Clouds for Efficient Label Summarization", *11th International Conference on Intelligent Systems Design and Applications (ISDA)* [201]

The rest of this chapter is organized as follows. In Section 7.2, we review the theoretical basis of approximate computation of statistics in data streams using sublinear space. In Section 7.3, we present

our approach to estimation of weighting functions. Finally, in Section 7.4, we empirically evaluate our approach in two applications: keyword extraction and word-cloud summarization.

7.2 Background

7.2.1 Problem definitions and variations

As we have seen in Section 7.1, in this chapter we wish to approximate item counts using sublinear space. Moreover, we are interested in obtaining an approximate list of the top-k most relevant terms for each document or category. In this section, we formalize the tasks of finding frequent items, top-k items and item count estimation. These problems are important in themselves and as subroutines of more advanced computations. In fact, they are basic components of our proposal for calculation of approximate weighting functions, presented in Section 7.3.

The *frequent items* problem consists in processing a data stream to find all the items appearing more than a given fraction of the times. It is one of the most studied problems in data stream mining [202]. The exact *frequent items* problem is defined as follows:

Definition 7.1. Given a stream S of m items $\langle e_1, e_2, \ldots, e_m \rangle$, the frequency of an item $e \in S$ is $f(e) = |\{e_j \in S \colon e_j = e\}|$. The exact ϕ-frequent items is the set $\{e \in S \colon f(e) > \phi m\}$.

Here, ϕ is a real number that represents the ratio of the frequency of a given term with respect to the total number of items m ($\phi \leq 1$). This definition has an inherent limitation in streaming scenarios. Any streaming algorithm that solves this problem must use a linear amount of space with respect to the number of different items, as shown in the work of Cormode and Muthukrishnan [202]. When handling data streams, approximate versions of the problems are preferred if they can be solved using sublinear space. For that reason, an approximate version is defined based on error tolerance:

Definition 7.2. Given a stream S of m items, the ϵ-approximate frequent items problem consists in finding a set F of items so that for all items $e \in F$, $f(e) > (\phi - \epsilon)m$, and there is no $e \notin F$ such that $f(e) > \phi m$.

Note that the exact ϕ-frequent items problem is a particular case of the approximate function where $\epsilon = 0$. Henceforth, when we use the expression *frequent items* we refer to its ϵ-approximate version.

We encounter different alternative problem definitions in the literature concerning the *top-k* problem [203]. The definition of the exact problem is straightforward: the top-k elements in a stream are the k elements with the highest frequencies. The exact solution to this problem would require complete knowledge of the frequencies of all elements, leading to linear space. Consequently, several relaxed variations have been proposed. The `FindCandidateTop`(S, k, l) problem consists in finding l elements containing the top k elements ($l > k$), without guarantees about the ranking of the remaining elements. The `FindApproxTop`(S, k, ϵ) consists in finding a list of k elements $\langle e_1, \ldots, e_k \rangle$ with a frequency $f(e) > (1 - \epsilon)f(e_k)$ for every element e in the list, where ϵ is the error threshold defined by the user and $f(E_1) \geq f(E_2) \geq \ldots \geq f(E_{|A|})$. $|A|$ i the number of different items, and E_k is the element with the k-th rank.

A related problem is the estimation of item frequency:

Definition 7.3. Given a stream S of m items, the frequency estimation problem consists in processing a stream so that, given any element e, an approximate frequency $\hat{f}(e)$ is returned satisfying $\hat{f}(e) \leq f(e) \leq \hat{f}(e) + \epsilon m$.

There are two main families of algorithms for finding frequent and top-k items. The first one is *counter-based algorithms*, where a subset of items is kept and monitor counts associated with each of the items are used. One of the first counter-based algorithms is the aforementioned *Frequent* algorithm, originally proposed by Misra and Gries [195]. This algorithm outputs a list of top-k elements without guarantees on frequency estimations. Another simple and

well-known counter-based algorithm is Lossy Counting, proposed by Manku and Motwany [204], which periodically deletes elements with an estimated low frequency. The *Space Saving* algorithm also belongs to the counter-based family. This algorithm ensures that no false negatives are kept in the top-k list (although it allows for false positives). *Space Saving* is less demanding on memory than other counter-based algorithms [192]. A refinement of this algorithm that incorporates elements from sketch-based algorithms is described in the work of Homem and Carvalho [205].

The second class of algorithms are the sketch-based ones. Sketch-based algorithms are able to provide frequency estimations for all elements in the stream, not only for the most frequent ones [206]. Each element is hashed into the space of counters using a family of hash functions, and the counters are updated for every hit of this element [192]. The disadvantage with respect to counter-based algorithms is that more memory is needed, since a big enough number of counters has to be used to minimize hash collisions. Two prominent examples of sketch-based algorithms are Count-Sketch [207] and Count-Min Sketch [202], both of them based on maintaining a collection of hash functions that map each element to a counter. Both algorithms are explained in depth in Subsection 7.2.2.

7.2.2 Sketch-based techniques

In the context of stream mining algorithms, a *sketch* may be defined as a matrix data structure which represents a linear projection of the input. This linear projection is implicitly defined by a set of hash functions. Sketch algorithms are designed to solve the frequency estimation problem, although they can also be applied to the top-k items problem, using a heap-like structure, as shown in the work of Charikar et al [207].

A sketching algorithm consists of three elements: *i*) a *matrix data structure* that summarizes the stream using a linear projection of the input; *ii*) an *update procedure* which modifies the structure each

time a new element arrives, and *iii*) a *set of queries* that the structure supports and their associated procedures (e.g. the sum of the elements, their average, variance, or extreme values).

There are different sketching models. Amongst them, there are frequency-based sketches, such as Count-Min Sketch and Count-Sketch, which are compact summary data structures used to compute the frequencies of all items in the input stream. We now proceed to describe both algorithms, and show that the Count-Min Sketch has better space and time properties.

The first algorithm is the *Count-Sketch*, proposed by Charikar et al [207]. A Count-Sketch with parameters (w, h), where h is the height and w is the width, comprises the following elements:

- **Data structure**: a two-dimensional array C of counters with h rows of length w. Each row r_j is associated with two hash functions: h_j that maps the input domain $\mathscr{U} = \{t_1, t_2, \ldots, t_n\}$ onto the range $\{1, 2, \ldots, w\}$, and g_j, which maps input items onto $\{-1, +1\}$.
- **Update procedure**: At the beginning, the entire sketch array is initialized with zeros. Whenever an item i arrives, $h_j(i)$ is computed in row j for $1 \leq j \leq h$, and the value of $C[j, h_j(i)]$ is incremented by $g_j(i)$ in the sketch array.
- **Estimation procedure**: let $\mathbf{A}$ be an array of n counters. Each $\mathbf{A}[i]$ registers the number of appearances of the $t_i \in \mathscr{U}$ element in the stream. The estimate of the counter $\mathbf{A}[i]$ is calculated as:

$$\hat{\mathbf{A}}[i] = \mu_{\frac{1}{2}}(g_j(i)C[k, h_k(i)]), i \leq j \leq h \tag{7.1}$$

where $\mu_{\frac{1}{2}}(x)$ represents the median of x. The total space used by the algorithm has complexity $\mathcal{O}(\frac{1}{\epsilon^2} log \frac{1}{\delta})$, while the complexity associated with the time per update is $\mathcal{O}(log \frac{1}{\delta})$ [207].

The second sketch-based algorithm is *Count-Min Sketch*, which is the one we have used in our proposal because of its better space

and time properties. It was proposed by Cormode and Mathukrishnan [202] and comprises the following elements:

- **Data structure**: same as in *Count-Sketch* except for the hash functions g, which are not used. See Figure 7.1.
- **Update procedure**: initially the entire sketch array is initialized with zeros. Whenever an item i arrives, $h_j(i)$ is computed for each row, and the value of $C[j, h_j(i)]$ is incremented by one in the sketch array. Therefore, processing each item takes time $\mathcal{O}(h)$.
- **Estimation procedure**: the estimate of the counter $\mathbf{A}[i]$ of a given item i in the input stream is:

$$\hat{\mathbf{A}}[i] = min_k C[k, h_k(i)] \tag{7.2}$$

The following Theorem is proven in the aforementioned work of Cormode and Muthukrishnan [202]:

Theorem 7.4. *The Count-Min Sketch algorithm uses space* $\mathcal{O}(\frac{1}{\epsilon} log \frac{1}{\delta})$ *with update time of* $\mathcal{O}(log \frac{1}{\delta})$ *per each update and query operation, and estimates the frequency of element i (*$\mathbf{A}[i]$*) by means of an estimate* $\widehat{\mathbf{A}}[i]$ *such that* $\mathbf{A}[i] \leq \widehat{\mathbf{A}}[i]$ *and with probability at least 1 -* δ*:*

$$\widehat{\mathbf{A}}[i] \leq \mathbf{A}[i] + \epsilon ||\mathbf{A}||_1. \tag{7.3}$$

Recall that $\mathbf{A}$ represents the vector of frequencies. It can be proven that in order to get an ϵ approximation with probability δ we need $width = \mathrm{e}/\epsilon$ and $height = log(1/\delta)$ [202], where e is the base of the natural logarithm. Therefore, to select the sketch size, we have to choose a given ϵ and δ. From these equations we observe that the algorithm has a space complexity of $\mathcal{O}(\frac{1}{\epsilon} log \frac{1}{\delta})$ regardless of the vocabulary size. That is, once the sketch size is fixed for a given ϵ and δ, the space complexity is sublinear. The same is true for update and estimation time.

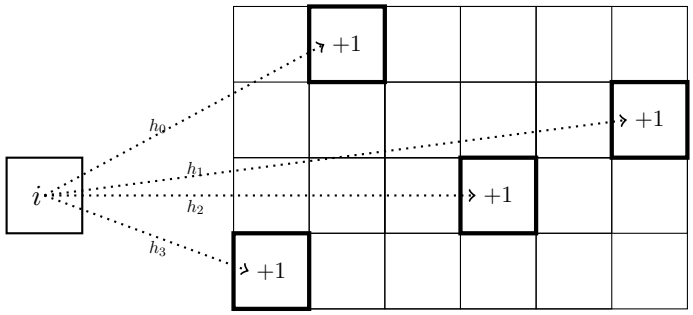


FIGURE 7.1: Count-Min Sketch structure, with $w = 6$ and $d = 4$. When a new item i arrives, for each row r of the structure the k-th element (where $k = h_r(i)$) is selected and its value is incremented

The space can be further reduced when items follow a Zipf distribution, which is usually the case in natural language. A Zipf distribution with parameter z has the property that the relative frequency of the i-th most frequent item is given by $f_i = \frac{c_z}{i^z}$, where c_z is a scaling constant. If the data follow a Zipf distribution with parameter z, the following Theorem holds:

Theorem 7.5. *Using a Count-Min Sketch, the space required to answer point queries with error* $\epsilon||\mathbf{A}||_1$ *with probability at least 1 - δ has complexity:*

$$\mathcal{O}(\frac{1}{\epsilon^{min\{1,\frac{1}{z}\}}} ln \frac{1}{\delta}) \tag{7.4}$$

The proof of Theorem 7.5 is also provided in the work of Cormode and Mathukrishnan [202]. Given these guarantees, it is obvious that the Count-Min Sketch represents a significant improvement over previous sketch-based model such as Count-Sketch, which has an $\mathcal{O}(\epsilon^{-2})$ space complexity with respect to ϵ, while Count-Min Sketch has $\mathcal{O}(\epsilon^{-1})$. These properties make Count-Min Sketch a suitable approach for approximate term count estimation in sublinear space.

Let us now move on to our approach for approximating weighting functions in sublinear space using the Count-Min sketch, which represents the main contribution of this chapter.

7.3 Approximate calculation of weighting functions using sketches: TF-SIDF and SBM25

As seen in Section 7.1, two of the most popular functions in Information Retrieval are *TF-IDF* and *BM25*, for which the importance of a term is proportional to the number of times it appears in the document, and inversely proportional to the number of documents in which that term appears. In this section, we present the main contribution of this chapter: a method for approximating weighting functions using sketches, which we call *TF-SIDF* (for calculating TF-IDF) and *SBM25* (for BM25).

7.3.1 Previous definitions

Let us first provide some definitions that we will need afterwards. Let $\mathscr{U} = \{t_1, t_2, \dots t_n\}$ the set of different observed terms (the *vocabulary*). The size of $\mathscr{U}$ is denoted as n. This set is expanded when a new term is observed. Let $D = \{d_1, d_2, \dots, d_{|D|}\}$ be a stream of documents. Each document $d \in D$ contains a list of $|d|$ terms $\langle w_1, w_2, \dots, w_{|d|} \rangle$. Note that each term can appear multiple times in a document. We can define $f(t_i, d_j) = |\{w \in d_j \colon w = t_i\}|$ as the number of times t_i appears in the document d_j. For a given document d_j and a term t_i the *term frequency* $\mathrm{TF}(t_i, d_j)$ is the number of times the term t_i appears in the document d_j divided by the length of the document:

$$\mathrm{TF}(t_i, d_j) = \frac{f(t_i, d_j)}{|d_j|} \tag{7.5}$$

Let us also define $df(t_i)$ as the number of documents where t_i appears: $df(t_i) = |\{d \in D \colon t_i \in d\}|$. For a given term t_i, the *inverse document frequency* $\mathrm{IDF}(i)$ is the number of documents divided by the number of documents containing this term:

$$\mathrm{IDF}(t_i) = log \frac{|D|}{df(t_i)} \tag{7.6}$$

The TF-IDF weighting function gives a higher weight to a term in a document if it appears frequently in that document and rarely in the others [10]. TF-IDF combines term frequency and inverse document frequency. For a given document d_j and a term t_i, the *term frequency-inverse document frequency* TF-IDF(i, j) is defined as follows:

$$\text{TF-IDF}(t_i, d_j) = \text{TF}(t_i, d_j)\text{IDF}(t_i) \tag{7.7}$$

Regarding the BM25 function, it is computed as follows:

$$\text{BM25}(t_i, d_j) = \frac{3 \cdot f(t_i, d_j)}{f(t_i, d_j) + 0.75 + \frac{2.25 \cdot |D|}{avgdl}} \cdot log \frac{|D| - df(t_i) + 0.5}{df(t_i) + 0.5} \tag{7.8}$$

where *avgdl* is the average document length and the other terms have the same meaning as in Equations 7.5 and 7.6.

In the next Subsection, we show how these weighting functions can be adapted for sketch-based approximate calculation.

7.3.2 Proposed approach

For computing the TF-IDF score of a term in a document, Equation 7.7 is used. If this formula is applied in an incremental setting, with documents arriving one after another, we first need to extract information from the current document (d_j) in order to obtain the TF(t_i, d_j) and then to process information from all the previously seen documents $d_1, \ldots, d_j$ in order to compute the IDF(t_i). Thus, we need to maintain a counter for each different term that has been extracted from the documents seen so far, which results in linear complexity with respect to the vocabulary size.

Given the space complexity of storing exact counts of how many times each term has occurred, we propose to use a Count-Min Sketch to approximate these counts. This Count-Min Sketch is thus used to get an approximation of the times each term has appeared in the

document, in order to provide an approximate value for the IDF factor.

We call our solution *TF-SIDF* (Term Frequency - Sketched Inverse Document Frequency). This approximation of TF-IDF is calculated as follows:

$$\text{TF-SIDF}(i,j) = \text{TF}(i,j) log \frac{|D|}{\widehat{df}(t_i)} \tag{7.9}$$

where $\widehat{df}(t_i)$ represents the approximation of $df_(t_i)$ obtained by the Count-Min Sketch algorithm.

We can employ the same strategy for calculating different weighting functions. For example, the sketched version of BM25, *SBM25*, is calculated as follows:

$$\text{SBM25}(i,j) = \frac{3 \cdot f(t_i,d_j)}{f(t_i,d_j) + 0.75 + \frac{2.25 \cdot |D|}{avgdl}} \cdot log \frac{|D| - \widehat{df}(t_i) + 0.5}{\widehat{df}(t_i) + 0.5} \tag{7.10}$$

Therefore, the TF-SIDF / BM25 system comprises the following elements:

- An exact counter D which stores the number of documents seen so far
- A *Count-Min Sketch* structure that approximates $df(t)$ for every term t in the vocabulary.
- In the case of BM25, we need a real number $avgdl$ that maintains the average document length.

The TF term is independent for each document (it only depends on the current document), so no information associated with it has to be permanently stored. That is, the TF information can be discarded once it has been used for each document. Every time a

document arrives, the sketch used to calculate the IDF term has to be updated.

After describing the elements that comprise our approach for approximate attribute weighting, we provide an empirical evaluation in the next Section.

7.4 Experimentation

We have carried out experiments for two different applications: online *keyword extraction* and online *word clouds* construction. In the first application, we validate the feasibility of using sketches for weighting function calculation, using the TF-IDF function. In the second application, we add the BM25 function for the calculation of word clouds.

Our hypothetical scenario consists of an unrestricted stream of text documents. For each document, we want to extract the most relevant words with respect to the past documents, using sketches to reduce the space complexity.

7.4.1 Experimental setup and evaluation measures

In our experiments, we compare the results using several sketch configurations (i.e. height and weight) with the results using exact counters. We force the size of the sketch to be smaller than the number of different terms, because the objective is to reduce space. Therefore, sketches larger than the space needed for storing the exact counts would not make sense. We can define the *compression reduction* of a given sketch $C_{w,h}$ of width w and height h with respect to a dataset D as $compression\_ratio(s_{w,h}, D) = w \times h / ||D||_0$, where $||D||_0$ represents the size of dataset D in number of distinct terms (that is, the vocabulary size). The compression ratio is, therefore, a real value that ranges between 0 (when the sketch has size 0) and 1 (when the sketch has the same number of counters that would be necessary to store exact counts for every term in the dataset).

For this reason we have previously calculated this number of counters and have added the restriction that the total size of the sketches has to be $< \frac{3}{4}N$.

In our experiments, we measure the performance of different sketch configurations concerning memory size. In order to measure this performance, we have used two different measures: the *recall* and the *Spearman distance*. The recall is defined as the size of the overlapping between two top-k lists:

$$recall(L_{sk}, L_{ex}) = \frac{|L_{sk} \cap L_{ex}|}{k} \tag{7.11}$$

where L_{sk} and L_{ex} are the top-k terms calculated by the sketched and exact versions of the algorithms, respectively, and k is the number of top-k terms (in our case, we have opted to use $k = 10$). This measure identifies the number of co-occurrences for both lists, without taking into account the order of the lists.

On the other hand we are also interested in the *order* of the lists, since it affects the order of the keywords and the representation of the word cloud. To this end, we use the *Spearman distance* (Δ_{sp}) [208] between the top-k terms:

$$\Delta_{sp}(L_{sk}, L_{ex}) = \frac{1}{k(k+1)} \sum_{i \in D} |pos(i, L_{sk}) - pos(i, L_{ex})| \tag{7.12}$$

where D is the set of terms in $L_{sk} \cup L_{ex}$, and pos(i,L) is the position at which i appears in L, or $|L| + 1$, if i does not occur in L.

The rationale behind this kind of measures over top-k lists is that we are not interested in the performance for all of the terms, but only in the most relevant ones, so the ranking of the elements is an important factor. Therefore, we include only the most relevant terms in our evaluation measures. As a final indicator of the performance of configuration for a given dataset, we use the average values of these measures over the documents.

Dataset	Number of documents	Number of terms
Reuters	9159	206543
PMC	212374	26903638

TABLE 7.1: Dataset statistics in number of documents and different terms

Additionally, we need a measure of the correlation between size reduction obtained by the sketches and performance with respect to the exact version. To this end, we use the Spearman correlation coefficient (not to be confused with the Spearman distance), which measures how well the relationship between two variables can be described by a monotonic function [209]. The relationship does not have to be linear, as is the case with the Pearson correlation coefficient. More specifically, it measures the correlation between the element ranks. The Spearman correlation coefficient between lists X and Y can be calculated as follows:

$$\rho(X,Y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \tag{7.13}$$

where x_i is the rank of X_i and $\bar{x}$ is the average rank of the elements in X.

Finally we have used the Friedman test [55], introduced in Subsection 2.9.2 to check if there is statistically significant difference in the performance of the models presented in Subsection 7.4.4.

7.4.2 Datasets

We have used two datasets for our experimental evaluation: **Reuters** and **PMC**, both of them presented in Subsection 2.9.3. We have preprocessed these initial datasets. First, the original XML documents in the dataset were parsed in order to extract raw text from their titles, abstracts and bodies. The raw text was converted into lower case, and stop-words were deleted, as well as words with a length of only one or two characters. Numbers and other non-alphabetical symbols have also been deleted. Stemming has been

applied to the remaining words, using the Porter algorithm [210]. Finally, the dataset has been reordered chronologically in order to simulate the real-world scenario. We show the statistics of these datasets in Table 7.1.

7.4.3 First application: online keyword extraction

In our first application, keywords are extracted using a term-weighting scheme that assigns higher weights to terms that are more relevant according to the TF-SIDF function.

When a document arrives, the weighting function is calculated for each of its terms, and the top-k terms with the highest value are selected as keywords.We set $k = 10$. We use the approximate weighting functions TF-SIDF and compare the obtained keywords with the keywords we obtain when using exact weighting function. As evaluation measures, we use the Spearman distance and recall to compare the exact list of keywords and the list of keywords obtained by different Count-Min Sketch structures, with different widths and heights.

In Figures 7.2 and 7.3, we show the Spearman distance and recall results for different compression ratios and sketch heights, for the Reuters and PMC datasets, respectively. We have calculated the Spearman correlation coefficient between the compression ratio and the Spearman distance for the Reuters dataset, obtaining a coefficient of -0.999 for $height = 2$, -1 for $height = 5$ and -0.999 for $height = 10$, which indicates a very high inverse correlation. We have also calculated the Spearman correlation between the compression ratios and the recall values, obtaining as a result 0.999, 1 and 0.999 for $height = 2$, $height = 5$ and $height = 10$ respectively. Hence, it is clear that bigger compression ratios lead to worse results. This is an expected result, since smaller sketches have more hash collisions, which results in more errors. The same applies to the PMC dataset, where the Spearman correlation coefficients between the Spearman distances and the size reductions are -1 for all heights, and 1 between recall and size reductions, showing a perfect

	Reuters			PMC		
Size reduction	**h = 2**	**h = 5**	**h = 10**	**h = 2**	**h = 5**	**h = 10**
0.025	0.7312	0.7538	0.7773	0.619	0.7204	0.8213
0.05	0.6844	0.7084	0.7396	0.4867	0.5983	0.6992
0.075	0.6548	0.6839	0.7142	0.4059	0.5162	0.6231
0.1	0.63	0.6651	0.692	0.3497	0.4556	0.5636
0.125	0.6107	0.6457	0.676	0.3077	0.4043	0.5168
0.15	0.5947	0.6317	0.6653	0.2729	0.3676	0.4762
0.175	0.5826	0.6176	0.6543	0.2429	0.3322	0.4428
0.2	0.5657	0.6046	0.6418	0.2191	0.3018	0.4112
0.225	0.5511	0.5955	0.6336	0.1999	0.2776	0.3848
0.25	0.5395	0.5821	0.624	0.1831	0.2524	0.3574
0.275	0.528	0.5706	0.6136	0.1688	0.2347	0.3369
0.3	0.5177	0.5642	0.6068	0.1558	0.2172	0.3172
0.325	0.5038	0.554	0.5974	0.1453	0.2031	0.2964
0.35	0.4966	0.5452	0.5923	0.1343	0.1857	0.2815
0.375	0.4865	0.5398	0.5845	0.1234	0.1705	0.2634
0.4	0.4785	0.5284	0.5773	0.1161	0.1581	0.2493
0.425	0.4788	0.5255	0.5777	0.1073	0.1469	0.2376
0.45	0.4599	0.5116	0.5654	0.1042	0.1345	0.222
0.475	0.4512	0.5063	0.5588	0.09982	0.1271	0.2095
0.5	0.4434	0.5013	0.5519	0.09113	0.117	0.1966
0.525	0.4358	0.4947	0.5451	0.08601	0.109	0.1864
0.55	0.4262	0.4856	0.5389	0.08321	0.1026	0.177
0.575	0.4194	0.4777	0.5349	0.07743	0.09411	0.1685
0.6	0.4179	0.473	0.5305	0.0747	0.08938	0.1603
0.625	0.4064	0.4673	0.5238	0.07013	0.08359	0.1484
0.65	0.3961	0.462	0.519	0.06679	0.0778	0.1445
0.675	0.3942	0.4524	0.5154	0.06332	0.07443	0.1335
0.7	0.3875	0.4458	0.5093	0.0604	0.06871	0.128
0.725	0.3778	0.4427	0.5056	0.05775	0.06453	0.1199

TABLE 7.2: Spearman distance between the exact top-k words (k=10) and its approximate equivalent calculated using a sketch with a given compression ratio with respect to the exact number of counters (shown in the first column), for the Reuters and PMC datasets. Sketch configurations with height 2, 5 and 10 have been used. The results show that reducing the size results in worse results, whereas smaller heights result in improving the distance

non-linear correlation (see also Tables 7.2 and 7.3). If we compare the results for the Reuters and PMC datasets, we see that better results are obtained with the latter. This indicates that our approach is more beneficial for bigger datasets, where the need to reduce space is more pressing.

The second observation is that better results are obtained when using sketch configurations with smaller heights, that is, with fewer hash functions. We have conducted a Friedman test in order to check if there is a statistically significant difference between using $height =$

	Reuters			PMC		
Size reduction	**h = 2**	**h = 5**	**h = 10**	**h = 2**	**h = 5**	**h = 10**
0.025	0.473	0.4548	0.4325	0.5011	0.4126	0.3358
0.05	0.5139	0.4932	0.4675	0.6088	0.518	0.4321
0.075	0.5384	0.5163	0.4906	0.6743	0.5848	0.4975
0.1	0.5583	0.5304	0.5092	0.7194	0.6338	0.5465
0.125	0.5749	0.5485	0.5241	0.7532	0.6747	0.5845
0.15	0.5863	0.5599	0.5329	0.781	0.7048	0.6167
0.175	0.5956	0.5714	0.5425	0.8057	0.7329	0.6442
0.2	0.6095	0.5805	0.5516	0.8239	0.7574	0.6691
0.225	0.6202	0.5888	0.5599	0.8402	0.777	0.6905
0.25	0.629	0.5982	0.566	0.8537	0.7966	0.7122
0.275	0.6369	0.6074	0.5746	0.8654	0.812	0.7294
0.3	0.6436	0.6126	0.5806	0.8758	0.8263	0.7445
0.325	0.6536	0.6192	0.5877	0.8844	0.8375	0.7614
0.35	0.6596	0.6267	0.592	0.8932	0.851	0.7734
0.375	0.6664	0.6309	0.597	0.9017	0.863	0.7875
0.4	0.6718	0.6387	0.6034	0.9076	0.8739	0.7993
0.425	0.6707	0.641	0.6027	0.9149	0.8828	0.8087
0.45	0.685	0.6507	0.612	0.9173	0.8925	0.8213
0.475	0.6906	0.6563	0.6179	0.9218	0.8982	0.8308
0.5	0.6962	0.6571	0.6225	0.9282	0.9064	0.8411
0.525	0.7013	0.6645	0.6272	0.9324	0.913	0.8497
0.55	0.7085	0.6697	0.6325	0.9343	0.9185	0.858
0.575	0.7125	0.6748	0.6361	0.939	0.9251	0.8644
0.6	0.714	0.6792	0.638	0.9407	0.9294	0.8716
0.625	0.7218	0.6825	0.6421	0.9449	0.9332	0.8807
0.65	0.7285	0.6862	0.647	0.9473	0.9385	0.884
0.675	0.7307	0.6942	0.6487	0.9505	0.9406	0.8926
0.7	0.7351	0.6987	0.653	0.9526	0.9461	0.8971
0.725	0.7406	0.7	0.6568	0.9548	0.949	0.9038

TABLE 7.3: Recall between the exact top-k words (k=10) and its approximate equivalent (See Table 7.2)

2, $height = 5$, and $height = 10$. The null hypothesis that there is no difference among the rankings in the results produced by these three configurations is rejected with a p-value of 2.2437×10^{-13} in both cases under a significance level of $\alpha = 0.01$. This is also confirmed in the case of the PMC dataset, where the null hypothesis is rejected under $\alpha = 0.01$, obtaining a p-value of 2.5437×10^{-13}. This shows that it is recommendable to use sketch configurations with a small height, that is, with few hash functions. This is explained by the properties of the Count-Min Sketch shown in Section 7.2.2. We are primarily interested in elements with low counts, since the values we estimate with sketches are in the denominator of the IDF term. Error is proportional to the 1-norm of the count array of all observed elements (the vocabulary size), so elements with large counts will

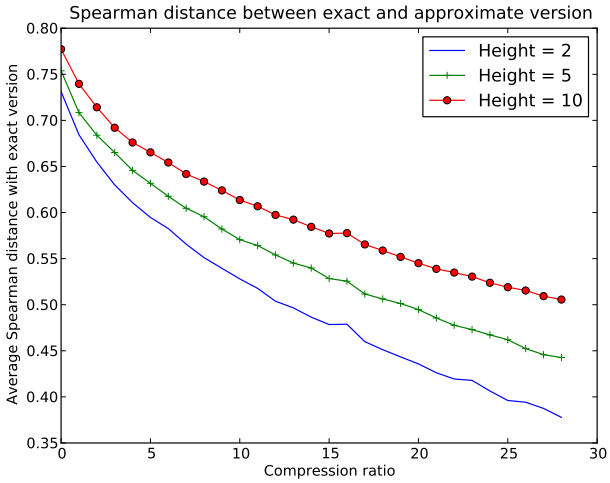
have lower errors. Error depends on width (more precisely, $\epsilon = \mathrm{e}/width$ with probability $1-\delta$), whereas δ depends on the number of hash functions (the height). Therefore, when working with elements having small counts it is desirable to use sketches with a width as big as possible, at the expense of height, as this reduces the expected error.

In Figure 7.4, we show the evolution of the Spearman distance using moving averages and sketches of sizes corresponding to a compression ratio of 0.3, 0.5 and 0.7, for the Reuters and PMC datasets. Initially, there is a better term coincidence, since few hash collisions have taken place. Later collisions result in a descending accuracy trend that nevertheless ends up stabilizing. This shows that the approximation is consistent.

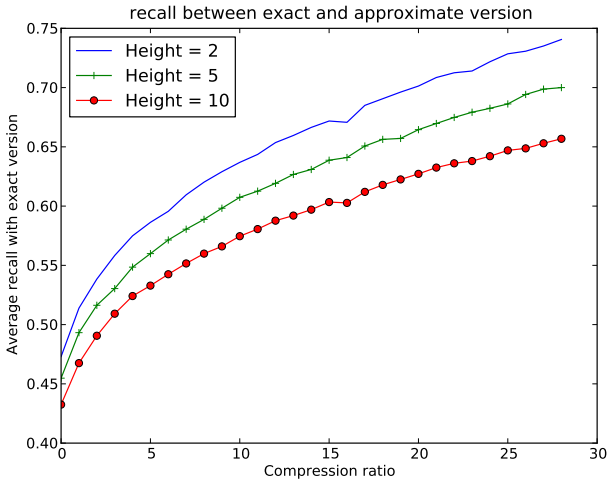
7.4.4 Second application: word cloud summarization

In the second application, our hypothetical scenario consists of a stream of categorized documents, where each document d has an associated *category* from a set of categories. Our problem consists in summarizing the contents of each of these categories by means of word clouds, using only the most relevant terms of the documents. The objective is to provide an approximate solution to this problem using Count-Min Sketch in order to reduce the space requirements.

The lists of words to be used in the word cloud for each category is calculated as follows. For each arriving document d, the list of its top-k words is obtained as in the first application (Subsection 7.4.3), setting $k = 10$. Such elements are marked as keywords for the document. We consider the terms that have been selected the most times as keywords in documents belonging to a given *category* to be the most representative terms for that category. In other words, in the exact version of the problem a counter $c_{category,t}$ has to be maintained for each category and term combination $(category, t)$. This counter is used to count how many times a term has been marked as a keyword in documents belonging to that category. The

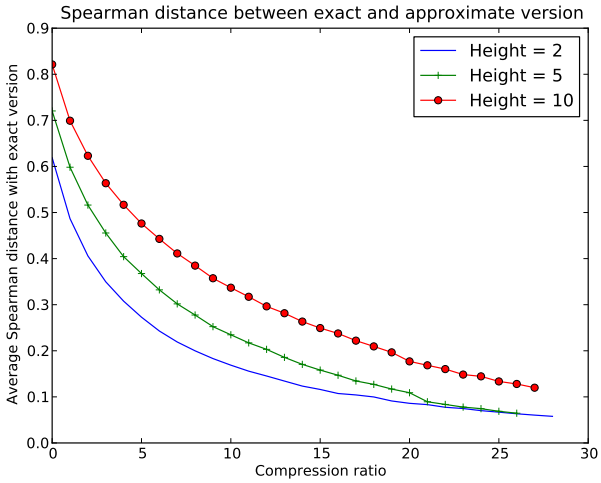


(a) Spearman distance

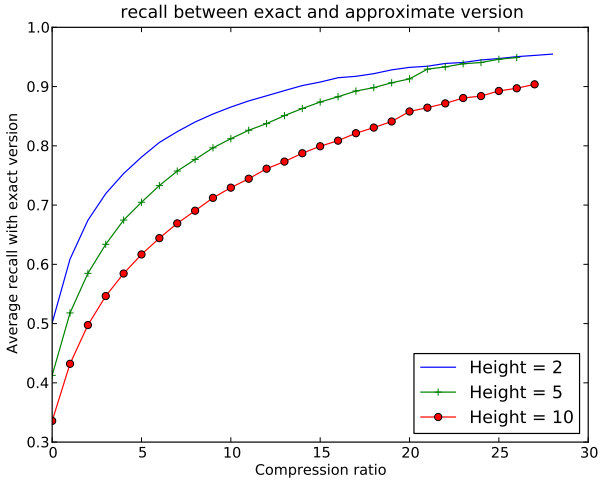


(b) Recall

FIGURE 7.2: Evolution of the Spearman distance and recall between the exact and approximate top-k words for different sketch sizes for the Reuters dataset. The x axis represents the compression ratio of the sketches with respect to the number of counters needed for exact calculation, while the y axis represents the Spearman distance. Three graphics are shown, for heights 2, 5, and 10 respectively. As explained in the paper, lower heights bring better results (see Friedman test results). The size of the sketch is inversely correlated with the Spearman distance, yielding correlation factor close to -1 (see the text for a more detailed explanation)

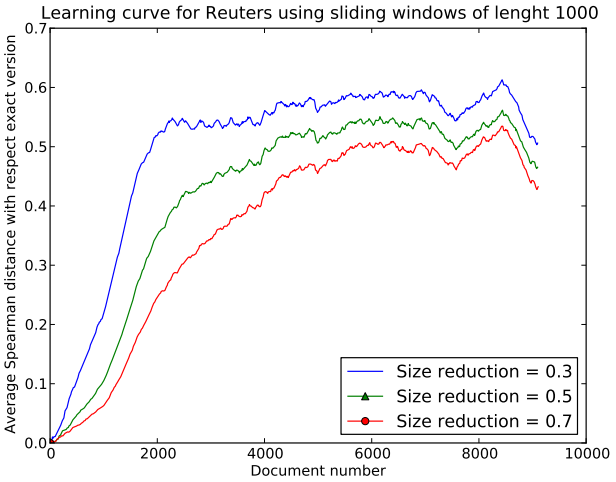


(a) Spearman distance

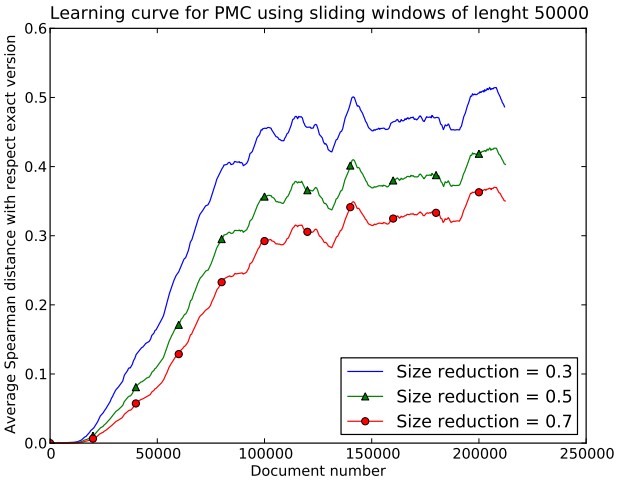


(b) Recall

FIGURE 7.3: Evolution of the Spearman distance and recall between the exact and approximate top-k words for different sketch sizes for the PMC dataset. The results show the same tendency as in the Reuters dataset, depicted in Figure 7.2



(a) Reuters



(b) PMC

FIGURE 7.4: Learning curve for the Reuters and PMC datasets. The lines represent the evolution of the Spearman distance (y axis) for sketches of different sizes (indicated in the legend). The x axis represents the number of documents analysed until now. The first documents show a better performance, something that could be expected given the absence of hash collisions at the beginning. The results stabilize and even improve as time goes by. This graphic uses moving averages over the Spearman distance with a moving window of width 1000 for Reuters and 50000 for PMC

terms with the highest counter in a category will be used as its representatives in the word cloud.

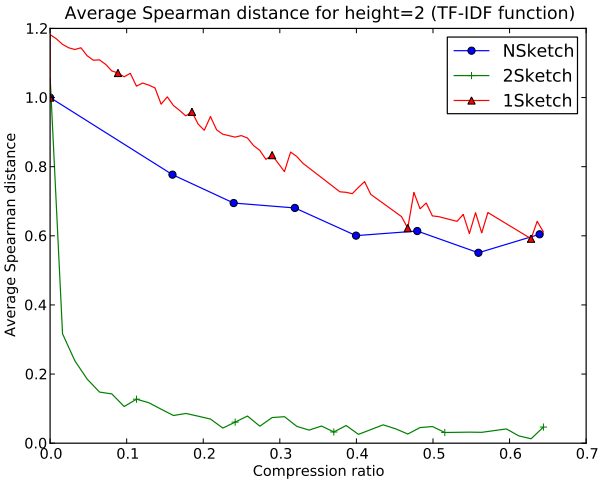
In the approximate version, all the needed counters are approximated using the Count-Min Sketch algorithm, in order to reduce the space complexity of the exact version. Sketches are used for two different tasks in our approach:

- Approximation of weighting functions (*frequency estimation problem*) for obtaining the keywords of a document.
- Approximation of the top-k problem to select the most frequent keywords in each word cloud corresponding to a category (*top-k problem*).

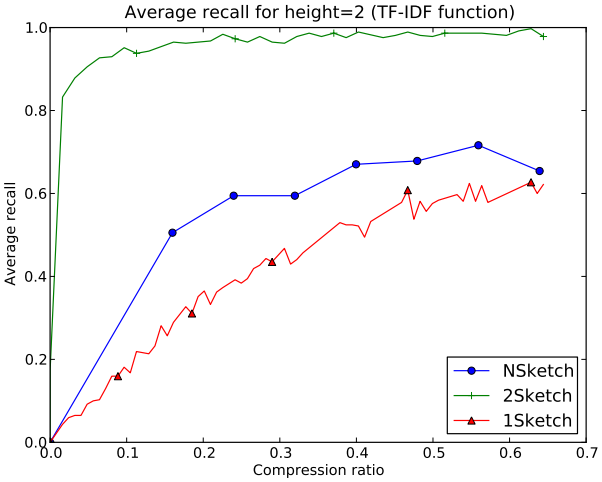
We consider three different options for the top-k calculation:

1. Using a different sketch to calculate the top-k for each label. An additional sketch is used for weighting function calculation. We call this configuration $NSketch$.
2. Using a shared sketch to calculate the top-k for each label. To this end, we have used the following method: the label is concatenated to the beginning of each term as a prefix, to avoid confusions when the same term appears in different labels. Again, different sketch is used for weighting function calculation. We call this configuration $2Sketch$.
3. Using the same sketch for both weighting function calculation and top-k calculation for all labels. We call this configuration $1Sketch$.

The memory requirements are different for each of these options. If we define a sketch S of width w and height h, its size $|S|$ will be $w \cdot h$. If we use a single sketch to calculate the weighting functions and the top-k words, $w \cdot h$ counters will be needed. If we use a sketch for the top-k problem for all labels and a different sketch for weighting function calculation, $2w \cdot h$ counters will be necessary. But if we opt

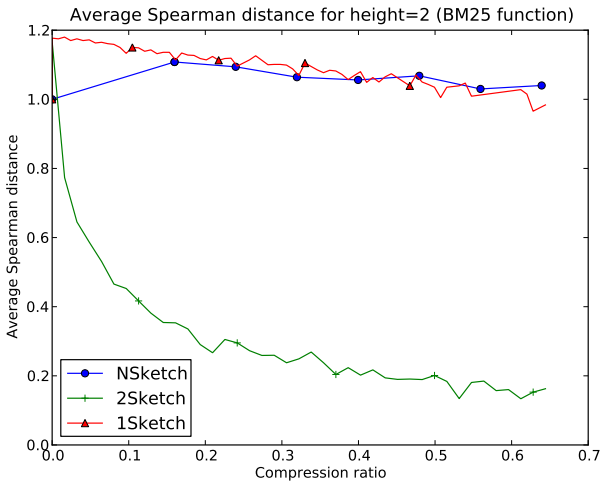


(a) Spearman distance

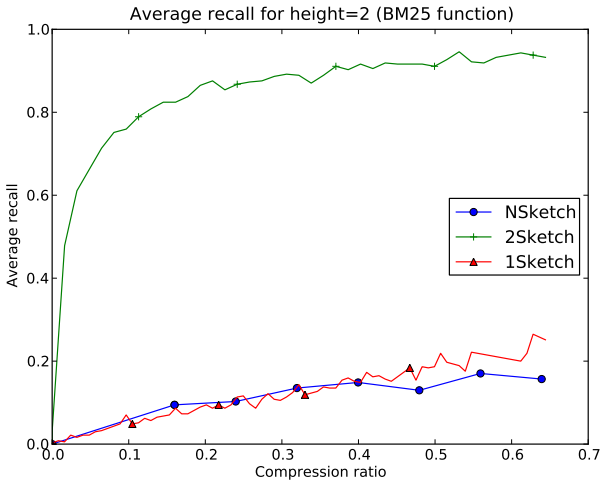


(b) Recall

FIGURE 7.5: Comparison of Spearman distance and recall evolution for the three models (Sketch height=2), using TFIDF as weighting function and PMC as dataset. We see that the optimal strategy is to use configuration 2Sketch, a sketch for weighting function calculation, and another one shared between the labels for top-k calculation. The best results are obtained for the 2Sketch configuration, because it is efficient in space while avoiding many collisions. See text for more details.



(a) Spearman distance



(b) Recall

FIGURE 7.6: Comparison of Spearman distance and recall evolution for the three models (Sketch height=2), using BM25 (PMC dataset). The results are similar to those depicted in Figure 7.5 for the TF-IDF function

for using a different sketch for each label, $(1+|labels|)\cdot w\cdot h$ counters in total will be necessary, where $|labels|$ is the number of labels.

For comparing the proposed models we use the Spearman distance and recall of the top-k elements with respect to the exact calculation. We use $k = 10$ in all the experiments. The comparison of the three proposed models is graphically depicted in Figures 7.5 and 7.6. All of them show the comparison for a given measure (Spearman distance or recall) for TF-IDF and BM25 with respect to the compression ratio. The figures show that the best results are obtained with the $2Sketch$ model: one sketch for calculating the weighting function, and another one for top-k. We have carried out a Friedman test in order to check if there is enough statistical evidence for this difference. The obtained p-values are 9.12×10^{-4} for both TF-IDF graphics, and 3.36×10^{-4} for the BM25 graphics. These results reject the null hypothesis under a significance level of $\alpha = 0.01$, which implies that there is a statistically significant difference. $2Sketch$ yields better results because, on the one hand, $NSketch$ is very inefficient in space since small sketches have to be employed so as not to use up too much memory, which rapidly causes saturation. On the other hand, $1Sketch$ causes big errors in the top-k calculation, given that sketches are more likely to be more accurate for greater item counts. For these reasons, $2Sketch$ is found to be the optimal configuration: it is memory-efficient while alleviating the problem of hash collisions that lead to errors in the top-k calculation.

See Figures 7.7, 7.8 and 7.9 for an example of word clouds for three specific journals. The graphics were produced by the online tool Wordle[1] using the terms and frequencies provided by the BM25 function.

[1] http://www.wordle.net

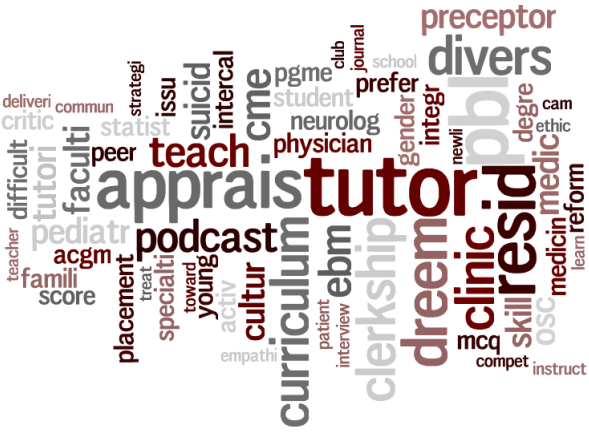


FIGURE 7.7: Word cloud for BMC Medical Education, obtained using SBM25 function. The most relevant terms (that is, the ones that are represented with a larger font size) are terms related to education, such as *tutor*, *curriculum*, *clerkship*, *appraise* or *faculty*, together with terms from the domain of medicine, such as *physician* or *pediatrician*. Note that the terms appearing in this graph are the result of applying the Porter stemming algorithm.

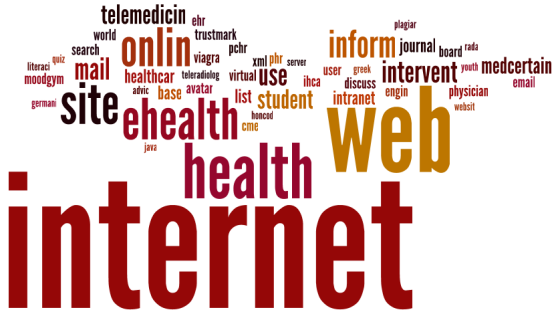


FIGURE 7.8: Word cloud for JMIR, using SBM25 weighting. As expected, the most relevant words correspond to Internet-related terms, such as *Internet*, *e-Health*, *web*, *site*, *online* or *telemedicine*. Note that the terms have been stemmed using the Porter algorithm.

FIGURE 7.9: Word cloud for HBP Surg, using SB25 weighting. The most relevant terms correspond to hepatopancreatobiliary surgery, such as *liver*, *pancreatic* or *hepatic*.

Chapter 8

STFSIDF: Efficient Online Feature Selection for Categorization in Text Streams

8.1 Introduction

As we have discussed, an important difficulty faced by text categorization applications is the inherently high dimensionality of the datasets. In order to classify documents, the bag-of-words representation is commonly used, which means that words appearing in the documents are used as classification features. Given the number of possible words, it is necessary to select only a subset of the words as attributes, in order to be able to learn with less computational cost and to avoid overfitting problems resulting from high dimensionality. A common approach is the application of weighting functions that associate numeric values or weights to attributes according to their relevance, selecting the words with the highest weight as attributes. We introduced to topic of dimensionality reduction in Chapter 2.

In this chapter we are in the data stream scenario. In data stream scenarios, data instances arrive from a hypothetically infinite stream, and the learning models have to be continuously updated. The available space is also limited, and documents cannot be stored for future reference. Sublinear algorithms are widely recommended in text stream mining scenarios [211].

As we saw in Chapter 5, a particularity of data streams is concept change [124]. The distribution of data changes over time, which means that the most important attributes at a given moment might not be the most important later on. This is especially problematic for high-dimensional text streams. Incrementally updating the set of features is necessary to maintain and improve the performance of text classification, as pointed out by Katakis et al. [163]. In order to dynamically change the classification attributes, the necessary statistics for computing weighting functions have to be continuously updated. In general, weighting functions such as chi-squared or Gini index have the disadvantage that the observation of a given term affects not only the weight for that term, but also for every term in the vocabulary. This means that, in order to calculate weighting functions, the weight of every word in the vocabulary has to be updated each time a new document arrives. This is not a feasible approach to feature space reduction in text streams, because of the time complexity. For this reason, we need weighting functions that minimize this problem.

An additional difficulty we face in the context of text streams is space complexity, due to the fact that statistics have to be kept for each of the terms that have appeared so far. For this reason, it is desirable to work with algorithms that use sublinear space complexity, even at the expense of controlled precision losses.

Therefore, there is a need for feature selection mechanisms able to take into account changes in the relevance of attributes during text data streams. Such mechanisms should be efficient in time and space, given the computational restrictions of data streams. Works on classifiers that accept a dynamic feature set are still rather scarce. An initial approach was proposed by Katakis et al.[163], comprising

two interconnected components: a) an incremental feature ranking method using the chi-square weighting function, and b) an incremental learning algorithm (based on Naïve Bayes) that accepts dynamic feature spaces. A drawback of this approach is that the weight for every term has to be recomputed for each new document. Some partial solutions have been proposed, for instance reselecting features periodically [164], or reelecting features only when missclassifications occur [12]. See also Chapter 6. However, these solutions are still expensive, since they still need many feature list recalculations, and, except in the proposal by Katakis et al., the feature set is not always kept updated. Moreover, they involve large space requirements given the vocabulary sizes in text streams, so they are not suitable for data stream processing. Henceforth, it would be interesting to have mechanisms for dynamic feature selection in data streams with reduced space complexity, which do not need to recalculate all the weights continuously in order to maintain an updated feature set.

In this chapter we propose such a mechanism for incrementally selecting the most relevant attributes in text streams. To this end, we propose the use of a modified version of TF-IDF which is able to efficiently maintain the updated weight of all the words, as well as the use of approximate algorithms for frequency estimation. We present the STFSIDF approach, based on the application of sketch-based algorithms and Bloom filters in order to reduce the space needed for the weighting function computation without significantly affecting the performance of the classification algorithms.

The rest of this chapter is structured as follows. In Section 8.2 we discuss the theoretical background and provide some relevant references. We discuss weighting functions for attribute selection, showing that the exact calculation and update of their values is unfeasible in data stream scenarios, and then we discuss some approximate algorithms for data streams. In Section 8.3 we explain our proposal for weighting functions approximation, using the concepts from the previous section. Finally, in Section 8.4 we provide the experimental evaluation.

8.2 Background

In this Section we present the main concepts used by our proposal and provide relevant references to the literature. In Subsection 8.2.1 we deal with feature space reduction, and in 8.2.2 we explain the approximate algorithms for data streams that we have used.

8.2.1 Feature selection in data streams with dynamic feature spaces

A common strategy for dimensionality reduction of text data is selecting the most relevant terms according to a given *weighting function* as classification features [212]. As we saw in Chapter 2, there are many weighting functions in the literature, such as the *document frequency* function, or functions originating from the Information Theory area [10], for example Information Gain [213], Mutual Information [214], Chi-square [213], DIA association factor [29], NGL coefficient [215], relevancy score [216], odds ratio [213] or GSS coefficient [217]. An empirical comparison of several function is provided in the work of Čehovin and Bosnić [218]. In general, these functions capture the intuition that the best terms for a given category are the ones distributed most differently in the sets of positive and negative examples [10].

TF-IDF [219] is a different alternative which captures the same intuition. For a given category c_j and a term t_i, the *term frequency-inverse document frequency* *TF-IDF*(t_i, c_j) is defined as:

$$TF - IDF(t_i, c_j) = \frac{f(t_i, c_j)}{|c_j|} log \frac{|C|}{|\{c \in C : t_i \in c\}|} \qquad (8.1)$$

where $f(t_i, c_j)$ is the number of appearances of term t_i in the category c_j, C is the set of existing categories, and $|c \in C : t_i \in d|$ is the number of categories in which the term t_i appears.

All of these functions provide *local* weights, that is, weights with respect to a given category. In order to obtain a global weight, the

local values have to be aggregated [10]. As we mentioned in Section 8.1, weighting functions in general have to be recalculated for every term in the vocabulary every time a new document arrives. Otherwise, it can occur that a term has a high weight at a given time, and after that is not observed anymore. Its actual relevance diminishes over time, but, since it is not updated, the term is still associated with a high weight. We need functions that reduce the number of necessary recalculations. TF-IDF has the advantage over other functions that it can be easily modified to cope with this requirement, as we will see in Section 8.3.

8.2.2 More approximate algorithms for stream summarization: Bloom filters

In stream mining contexts it is necessary to maintain certain statistics about the data. Nevertheless, the time and memory requirements of algorithms for streaming scenarios make it unfeasible to use exact calculations, since this would necessarily involve linear complexity, something which is not acceptable in streaming scenarios, as discussed in the work of Papapetrou et al. [220].

For this reason, several efficient algorithms have been proposed for approximating online statistics. Two important examples are *Bloom filters* and *sketches*. In Chapter 7 we saw that sketch-based algorithms are more adequate for function approximation than counter-based algorithms.

There are several works that discuss the application of approximate algorithms to text streams. See for example the work of Fung et al. [221], which uses document and category summaries within a similarity-based online multilabel classifier. Note that they use the term *sketch* in a different sense than we do. A recent related article is the one by Goyal et al. [222], which uses the Count-Min Sketch algorithm in order to compute semantic similarities between word pairs. Ravichandran et al. [223] use hash functions for compressing word-pair similarities in large text collections, while Levenberg and Osborne use approximate techniques for building language

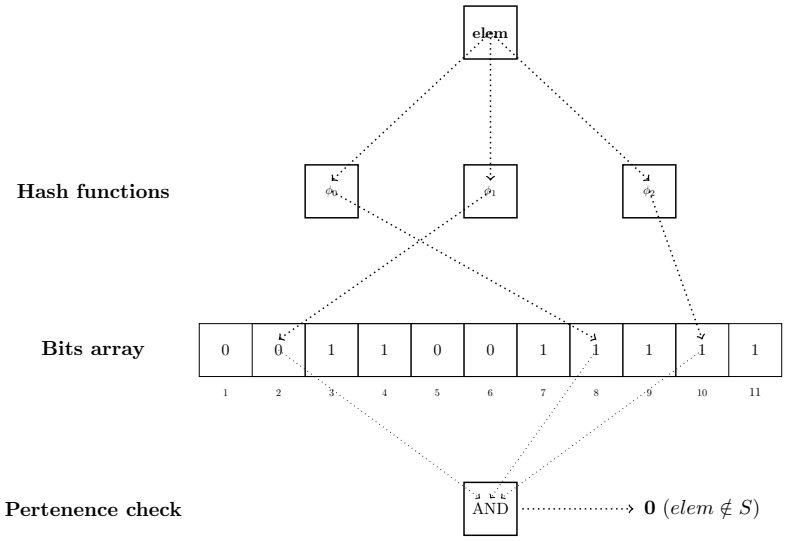


FIGURE 8.1: Schema of a simple Bloom filter structure. When adding $elem$ to set S, the positions $\phi_0(elem)$, $\phi_1(elem)$ and $\phi_2(elem)$ are set to 1 in the bits array. In order to check if $elem \in S$ the corresponding positions are checked. If all of them are set to 1, the element is in the set. If there is a single i such that $\phi_i(elem) = 0$, the element is not in the set

models in [224]. Bloom filters have been applied to multilingual keyword search in the context of cloud computing in the work of Pal et al. [225].

In Chapter 7 we have already reviewed Sketch-based algorithms. We now discuss *Bloom filters*. A Bloom filter is a space-efficient probabilistic data structure that can be used to determine whether an element belongs to a set S [226], and consists of a bit array b with m bits, and k hash functions $\phi_1, \ldots, \phi_k$, each one of them mapping an element into a position of the array with a uniform distribution. The optimal value of k is given by $k = \frac{m}{n} ln\ 2$ [227], where n is the number of inserted items. All the elements in b are initialized with 0.

Two operations are defined on a Bloom filter:

- $add\_to\_bloom\_filter(bf, elem)$ adds element $elem$ to the Bloom

filter bf, in case that $elem \in S$. For each hash function ϕ, $b[\phi(elem)] \Leftarrow 1$.

- $bloom\_filter\_contains(bf, elem)$ checks if element $elem$ belongs to the set S, using Bloom filter bf. In order to query if an element $elem$ is contained in S, all the corresponding positions in b are checked (that is, for every hash function $\phi_i \in \{\phi_1, \ldots, \phi_k\}$, $b[\phi_i(elem)]$ is checked). If any of the corresponding positions is 0, the element does not belong to S. That is, $\prod_{i=0}^{k} b[\phi_i(elem)] = 0 \Rightarrow elem \notin S$. If all the positions are 1, nothing can be assured.

See Fig. 8.1 for a graphical depiction of this structure. A Bloom filter may return false positives, but never false negatives. In spite of this disadvantage, Bloom filters have a clear space advantage over other data structures for representing sets that require the data items themselves to be stored. Moreover, the operations related to Bloom filters present a constant time complexity $O(k)$, independently of the number of elements in the set.

In the next Section we describe how we use sketch-based algorithms and Bloom filters to approximate the TF-IDF weighting functions.

8.3 STFSIDF for online feature selection

In this section we present the main contribution of this chapter, the STFSIDF approach for online attribute selection. First we explain the use of TF-IDF for feature selection, then we describe the use of approximate algorithms for approximating the TF-IDF weighting function and finally we explain more formally our proposal.

8.3.1 TF-IDF for efficient incremental feature selection

Our strategy for selecting features for classification works a follows. We keep a list q_j for each category c_j containing the top-k words

with the highest relevance according to a weighting function. These lists have to be permanently updated. In order to select the features that the classifier is going to use, we use the union of the lists, $\bigcup_j q_j$.

As previously stated in Section 8.1, using weighting functions to incrementally update the list of most relevant attributes has an inherent complexity problem, since, in general, the weights for all the terms and categories have to be continuously updated. Some partial solutions to this problem were presented in Subsection 6.2.2. By contrast, in this chapter we propose using weighting functions for which the number of recalculations of all the weights (*complete recalculations*) can be minimized. More specifically, we propose the TF-IDF weighting function since it allows us to reduce the number of necessary complete recalculations.

The TF-IDF weighting function can be modified to satisfy this property, by ignoring the $|c_j|$ term in Equation 8.1, and performing an update of all the weights only when it is indispensable (see Subsection 8.3.3). This results in a reduction of the number of weight recalculations. We denote this modified function as $TFIDF'$:

$$TFIDF'(t_i, c_j) = f(t_i, c_j) \left(log \frac{|C|}{|\{c \in C : t_i \in c\}|} \right) \tag{8.2}$$

Let us see why this function avoids most complete recalculations. Imagine that a document d with category c_j is observed, and that the category c was previously known. Then, for each word $w \in d$ one of the following applies:

- The word w had been previously observed in conjunction with category c_j. This implies that the $|\{c \in C : w \in c\}|$ term in Eq. 8.2 remains unchanged. The only weight that was to be modified is the $TFIDF'(w, c)$, since the only change has been $f(w, c)$. Thus, no complete recalculation is needed in this case
- Category c_j is new for word w. This implies that the $|\{\{c \in C : w \in c\}|$ term is incremented for each category $c \in C$ where w has occurred. This makes it necessary to recompute the

weights, which will diminish for word w in other categories. However this recalculation can be avoided for a category c in two cases: *a)* if w is not in the top-k list for category c (since its weight will diminish, it will not enter the top-k list if it is not previously there) and *b)* if w is in the top-k list, but its new weight is larger than the minimum weight in the top-k list. Hence, a complete recalculation is necessary only when the weight reduction of w makes it fall off the top-k list, because it is not possible to know which other term will occupy the vacant place. From the computational point of view, this is more affordable than running complete recalculations for every new document.

- It can also happen that a previously unknown category arrives. In this case, a complete recalculation has to be carried out for all the other categories. However, the number of new classes is expected to be low with respect to the number of documents.

This means that, in most cases, when a document arrives, the only weights that have to be updated correspond to the words appearing in the document, for the category with which the document has been tagged. Weights for other words and categories have to be recalculated only if a new category is observed, or if a word that appears in a document appears as a relevant attribute for other category and its new value is smaller than the smallest weight in the corresponding attribute list.

8.3.2 Weighting function approximation using sketches and Bloom filters

Let us start by providing some previous definitions. Let $\mathscr{S}$ be a stream of categorized documents, represented by a list of document - category pairs, $\{(d_1, c_1), (d_2, c_2), \ldots\}$. The stream contains a total number of terms $\#terms_{\mathscr{S}} = \sum_i |d_i|$, where $|d_i|$ is the length in number of terms of document d_i. The number of different terms is $\#diff\_terms_{\mathscr{S},c} = |\{t : \exists d(d, c) \in \mathscr{S} \wedge t \in d\}|$.

The number of counters needed for calculating the *exact* TFIDF for category c_j and stream $\mathscr{S}$ is

$$size_{tfidf}(\mathscr{S}, c_j) = 2 \times \#diff\_terms_{\mathscr{S},c} + 2 \tag{8.3}$$

That is, it is necessary to store the number of appearances and number of categories where each distinct term appears, and two additional counters (for the total category size and for the number of categories). This represents a linear complexity with respect to the number of terms in the stream.

Our idea for overcoming this complexity consists in using *approximate* instead of *exact* statistics for the calculation of weighting functions, making use of auxiliary Count-Min Sketch data structures. The challenge is to calculate accurate enough approximations of the functions while reducing the necessary space.

In order to calculate $TFIDF(t_i, c_j)$, the following statistics have to be maintained: *a)* total number of different terms; *b)* number of categories, *c)* number of appearances of each term in each category and *d)* number of categories where each term occurs.

The number of different terms and categories is not a problem from the point of view of space complexity, since it can be calculated using a single counter. The last two types of statistics are approximable by sketches, using the approximate version of the TFIDF function, which we call STFSIDF:

$$STFSIDF(t_i, c_j) = \frac{\widehat{f}(t_i, c_j)}{|c_j|} log \frac{|C|}{|\{c \in \widehat{C : t_i} \in c\}|} \tag{8.4}$$

where $|\{c \in \widehat{: t_i} \in c\}|$ represents the approximation of $|\{c \in C : t_i \in c\}|$ provided by the Count-Min sketch and $\widehat{f}(t_i, c_j)$ is the approximation of $f(t_i, c_j)$.

For approximating the number of appearances of each term in each category (the STF factor) we need a single sketch, which is used for

storing $(category, term)$ pairs, so that we do not have to maintain a different sketch for each category. In a pilot study (see Chapter 7) we have also tried using a different sketch for each category, but the results were worse than when using a shared sketch.

Approximating the number of categories where each term occurs (for the SIDF factor) is more tricky. An observed category for a term is added to the sketch only if it is the first time that a specific $(term, category)$ pair has appeared (because we are measuring the number of *different* categories in which a term has appeared). In order to assert if a given $(term, category)$ has appeared in the data stream, we use a Bloom Filter. In summary, we use a Count-Min Sketch in conjunction with a Bloom filter to approximate the SIDF term, and an additional Count-Min Sketch to approximate the STF factor.

8.3.3 STFISDF formalization

We now present our approach more formally. The STFSIDF calculation algorithm uses a structure $\Psi_{h_t,w_t,h_c,w_c,b}$ that comprises the following elements:

- A set of $|C|$ counters $CL = \{cl_1, \ldots, cl_{|C|}\}$ for calculating the size, in number of terms, for each category $c \in C$
- A Count-Min Sketch structure CM_{terms} that maintains the counts of the $(category, term)$ combinations, with width w_t and height h_t
- A counter c_{cat} that maintains the number of different categories
- A Count-Min Sketch structure CM_{cat} that maintains the number of different categories in which each term has appeared, with width w_c and height h_c.
- A Bloom Filter BF to decide whether a given $(category, term)$ combination has appeared, with b bits and f functions.

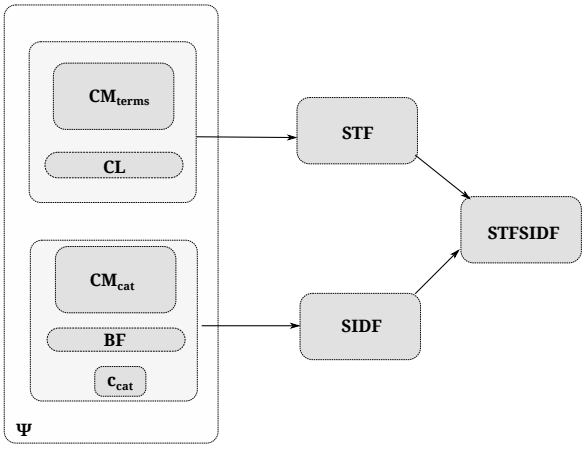


FIGURE 8.2: STFSIDF elements and their relationship with the final calculation

- A set of observed categories C

Additionally, we need a priority queue q_i of maximum size k for each category $c_i \in C$, in order to store the list of its top-k terms (sorted by TFIDF). Such queues can be restarted ($resetQueue(q)$), and it is possible to change the weight for a term in the queue ($changeWeightInQueue(q, w, weight)$). An element w with weight $weight$ can be added to queue q by means of *add_to_priority_queue(q, (w, weight))*. This means that the element will be kept in q if its weight is larger than the lower weight already present in the queue.

Figure 8.2 represents the relationship between the mentioned elements and the STFSIDF calculation: the CM_{terms} sketch and CL counters determine the STF term, while the Bloom filter BF, the CM_{cat} sketch and the c_{cat} counter determine the SIDF factor.

The size in number of counters of the structure is given by the following formula:

$$size(\Psi_{h_t,w_t,h_c,w_c,b}) = h_t w_t + h_c w_c + |C| + b + 1 \qquad (8.5)$$

This size depends on the selected sizes for the sketches and the Bloom filter. Obviously, we are interested in parameters for which

the size of the structure is lower than the size that would be necessary for the exact calculation, as long as good approximations are obtained. In Section 8.4 we show that it is in fact possible to obtain good approximations while reducing space.

The algorithm works as follows. First, all the relevant data structures are initialized: the set of observed categories is initialized as an empty set, each of the counters corresponding to category length are initialized with 0, as well as the sketches and the Bloom filter. Each time a new document d with category c arrives, the data structures are updated, following Algorithm 3. The set of observed categories is updated, as well as the associated lengths. The sketch structures associated with term counts are updated for each term $w \in d$, as well as the sketch and Bloom filter used for the $SIDF$ approximation. The priority queues are also modified, using the values returned by Algorithm 5. More specifically, if no new category has been observed for a given term, its weight is recomputed and offered to the queue corresponding to category c. Otherwise, other queues might have to be reset, following algorithm 4. This is by far the slowest operation of the approach, since all the words in the vocabulary have to be covered. There are two situations when this operation may take place: *a)* when a new category occurs in the dataset, or *b),* when an existent category is observed in conjunction with a term for the first time. As a result of this procedure, each queue q_j contains the top-k elements for category c_j according to the approximate TFIDF function.

Finally, the set of attributes to be used by the classifiers is the union of the set of attributes for each category, which are the ones provided by the queues.

Figure 8.3 shows graphically the procedure we have described for carrying out the STFSIDF calculation.

After the presentation of our approach, we provide the experimental results in the next Section.

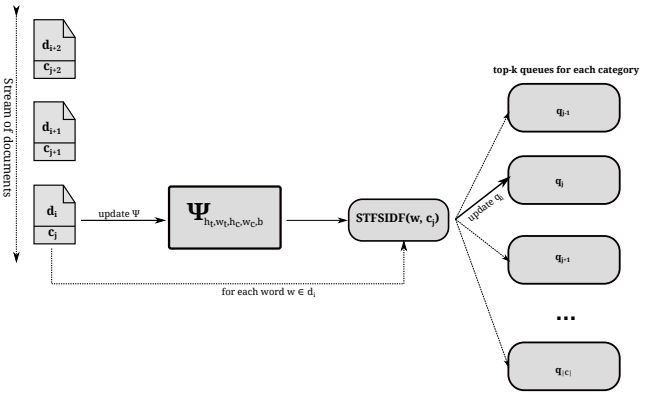


FIGURE 8.3: Schema of the STFSIDF elements and procedures. When a document arrives, the STFSIDF elements are updated using the documents words and category. These statistics are used to calculate the TFIDF function for each category, and the values are used to maintain a top-k queue for the categories.

Algorithm 3 *Update* procedure when a document arrives

Require: A document $\mathbf{d_i}$ with $|\mathbf{d_i}|$ terms, with category c_j
Ensure: Updated structures C, CL, CM, BF, CM_{cat}, q_i
  $C \Leftarrow C \cup \{c_j\}$
  $cl_j \Leftarrow cl_j + |d_i|$
  for each term w in $\mathbf{d_i}$ **do**
    $update\_sketch(CM_{terms}, \{w\, c_j\})$
    if $\neg bloom\_filter\_contains(c_j)$ **then**
      $update\_sketch(CM_{cat}, \{w\})$
      $add\_to\_bloom\_filter(BF, c_j)$
    end if
  end for
  $CategoriesToReset \Leftarrow \emptyset$
  for each term w in $\mathbf{d_i}$ **do**
    $tfidf \Leftarrow calculate\_stfstidf(w, c_j)$
    $add\_to\_priority\_queue(q_j, (w, tfidf))$
    if $isNewLabelForWord(w, c_j)$ **then**
      for each category c_k in $C - \{c_j\}$ **do**
        $weight \Leftarrow calculate\_stfsidf(w, c_k)$
        if $(w \in q_k \wedge weight < minWeight(q_k)) \vee |q_k| = 0$ **then**
          $CategoriesToReset \Leftarrow CategoriesToReset + c_k$
        else
          $changeWeightInQueue(q_k, w, weight)$
        end if
      end for
    end if
  end for
  for $c_k \in CategoriesToReset$ **do**
    $resetQueue(q_k)$
  end for

Algorithm 4 Reset queue

Require: A queue q for category c
  $clearQueue(q)$
  for each word w in the vocabulary **do**
    $weight \Leftarrow calculate\_stfstidf(w, c)$
    $add\_to\_priority\_queue(q, (w, weight))$
  end for

Algorithm 5 $STFSIDF$ calculation

Require: A term w and a category c_j
Ensure: $STFSIDF(w, c_j)$
  $stf \Leftarrow estimate\_from\_sketch(CM_{terms}, \{c_j, w\})$
  $estimate \Leftarrow estimate\_from\_sketch(CM_{cat}, w)$
  $sidf \Leftarrow \log \frac{|C|}{estimate}$
  $stfsidf \Leftarrow stf * sidf$
  return $stfsidf$

8.4 Experimentation

In this Section we evaluate our approach from two different perspectives. We first provide an *intrinsic* evaluation, where we study how different the features selected by the exact and approximate weighting functions are, and afterwards an *extrinsic* evaluation, in order to analyze how the classification algorithm is affected by the use of approximate feature selection functions, as well as the impact of attribute list recalculations.

The rest of this section is structured as follows. In Subsection 8.4.1 we describe the datasets and measures we have used in our evaluation. In Subsection 8.4.2 we explain the intrinsic evaluation and our parameter optimization process, and in Subsection 8.4.3 we provide the extrinsic evaluation.

8.4.1 Experimental setting: datasets and measures

We have launched an empirical evaluation using the Naïve Bayes algorithm, as in the work of Katakis et al. [163], with Laplacian probability estimators. We have used two datasets. We have used the Reuters and PubMed datasets, as we did in Chapter 7. Both datasets are distributed as plain text, so we have preprocessed them

in order to map the documents into a more suitable representation for machine learning. To this end, we have used a *bag-of-words* representation, after applying stop-word removal and stemming. A representation based on bigrams (adjacent word pairs) has been used. For classification purposes, we use the top 150 attributes for each category.

The PubMed dataset is distributed in an unlabelled form. Since we are interested in a supervised learning scenario, we have divided the articles into 15 clusters, using the *repeated bisections* method (top-down clustering) and the cosine distance as a distance function, with the help of the CLUTO library [228] for clustering.

We use various measures to evaluate our approach for two different purposes: comparison of lists of ordered elements (intrinsic evaluation) and classifier performance (extrinsic evaluation). For evaluating differences between ordered lists we use the Spearman distance (Δ_{Sp}) [208], as in Chapter 7

For evaluating classifier performance we use the prequential accuracy with fading factors, proposed by Gama et al. [114] as a mechanism to overcome some deficiencies of the plain prequential accuracy. Additionally, for evaluating to what extent the performances of two online classifiers over a data stream are different, we use the McNemar statistical test with fading factors, also proposed by Gama et al. [114]. See Section 5.2.5 for more information on this test.

8.4.2 Parameter optimization and intrinsic evaluation

The STFSIDF data structures for weighting function approximation have several parameters that affect their performance: the width and height of the terms sketch (w_t and h_t), the width and height of the categories sketch (w_c and h_c) and the size b of the Bloom filter. These parameters determine the total size of the structure (see Eq. 8.5). We pose the following constraint on the parameters: the total size of the data structures involved in STFSIDF has to be lower than the number of counters that would be necessary for the exact calculation. That is, $size_{tfidf}(\mathscr{S}, c_j) > size(\Psi_{h_t,w_t,h_c,w_c,b,f})$

reduction	Best Δ_{Sp}	% CM_{terms}	% CM_{cat}	% BF
0.5	0.001653	64.44%	26.09%	9.47%
0.25	0.1091	65.24%	31.09%	3.66%
0.125	0.3132	76.69%	17.90%	5.4%
0.0625	0.4471	80.62%	15.92%	3.46%
0.03125	0.6095	85.23%	13.79%	0.97%

TABLE 8.1: Best results for the Reuters dataset, according to the maximum allowed size

reduction	Best Δ_{Sp}	% CM_{terms}	% CM_{cat}	% BF
0.5	0.07758	21.53%	71.68%	3.69%
0.375	0.1745	28.97%	66.86%	4.16%
0.25	0.3188	15.57%	53.92%	7.94%
0.1875	0.423	35.48%	59.17%	14.58%
0.125	0.5745	26.02%	54.19%	17.15%

TABLE 8.2: Best results for the PubMed dataset, according to the maximum allowed size

For a given maximum $size(\Psi_{h_t,w_t,h_c,w_c,b,f})$, which we can abbreviated as MAX_SIZE, our objective is to find the parameters h_t, w_t, h_c, w_c, b, f that optimize the average Spearman distances between the exact and approximate lists of top-k attributes for each category, after processing the whole dataset.

The maximum size should be a fraction of the numbers of counters needed for the exact calculation, that is, $MAX\_SIZE = reduction * size_{tfidf}(\mathscr{S}, c_j)$, $reduction \in (0, 1)$. In order to find the optimal parameters, we have launched an optimization process for the Reuters dataset, using the Luus-Jakola optimization algorithm [229]. As a result, we have obtained the parameters in Tables 8.1 and 8.2.

We have used increasingly large size reductions until the Spearman distance has become larger than 0. $\widehat{45}$, since this distance ensures that there is at least one term that appears in one list and not in the other one.

Slight modifications of the obtained parameters do not necessarily yield statistically significant changes of the Spearman distance values. Figure 8.4 uses boxplots to represent the optimal range of values for the parameters associated with the optimal configurations we have found, that is, the range of values for which the average Spearman values (averaged over the categories) is not significantly

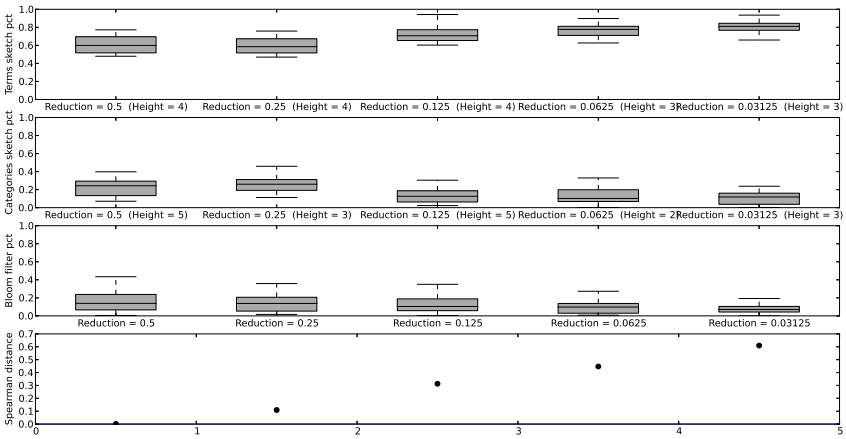
different from the obtained optimal value. We have used a parametric Student's t-test [209] with a significance level of $\alpha = 0.01$ to test significant differences.

We can extract several conclusions from these results. The first one is that bigger size reductions imply worse approximations, something which is intuitive. The second conclusion is that the relative sizes of the data structures depend on the number of different labels. This happens because the most difficult component to approximate for both TF-IDF is SIDF (the number of classes in which each component appears), given that IDF is optimized by a small number of appearances, which, as previously pointed out in Theorem 7.4, is more error-prone than in the case of more frequent elements. The Reuters dataset contains a high number of labels, which means that small modifications of the approximated values of IDF do not modify the relative term ranking. Nevertheless, the difference is smaller when there are fewer labels, as in the PubMed dataset, so bigger sketches are necessary to successfully approximate the SIDF term. As a corollary of this, when applying our approach to datasets with a high number of labels, we have to give more space to CM_{terms}, whereas when the number is not so high, more space has to be given to the CM_{cat} component.

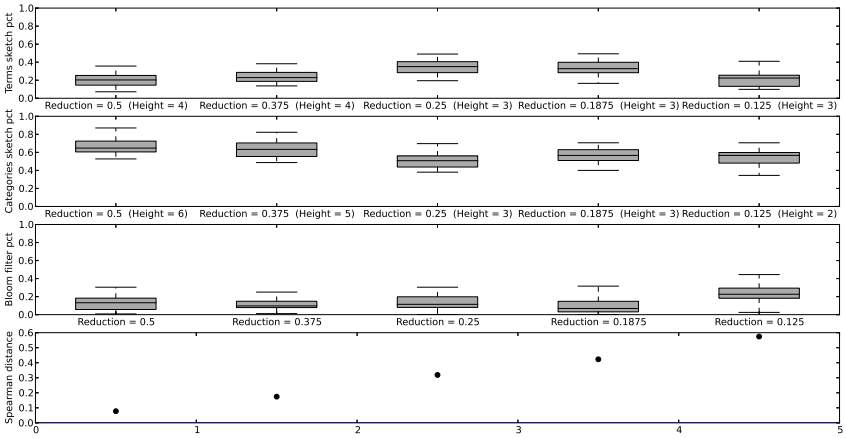
8.4.3 Application to classification (extrinsic evaluation)

The objective of the extrinsic evaluation is to test whether there are significant differences in the performance of the classifier with exact or approximate functions, as well as the impact of attribute list recalculations. We have used the best parameters for a space reduction of 0.5, shown in Table 8.2. In the case of the Reuters dataset, we have removed categories with less than 100 documents.

In Figure 8.5 we see the evaluation of the McNemar statistic corresponding to the comparison of the behaviour of the classification algorithm for the PubMed dataset, using exact or approximate values. Fading factors with $\alpha = 0.99$ are employed in the figure. It can be observed that the McNemar statistic lies within the $(-6.635, 6.635)$



(a) Reuters dataset



(b) PubMed dataset

FIGURE 8.4: Result of the parameter optimization for the Reuters and PubMed datasets. The first three rows in the figures represent the range of values of the size proportion of the terms sketch, the categories sketch and the Bloom filter which lead to an optimal Spearman distance, while the last row represents the obtained Spearman distance

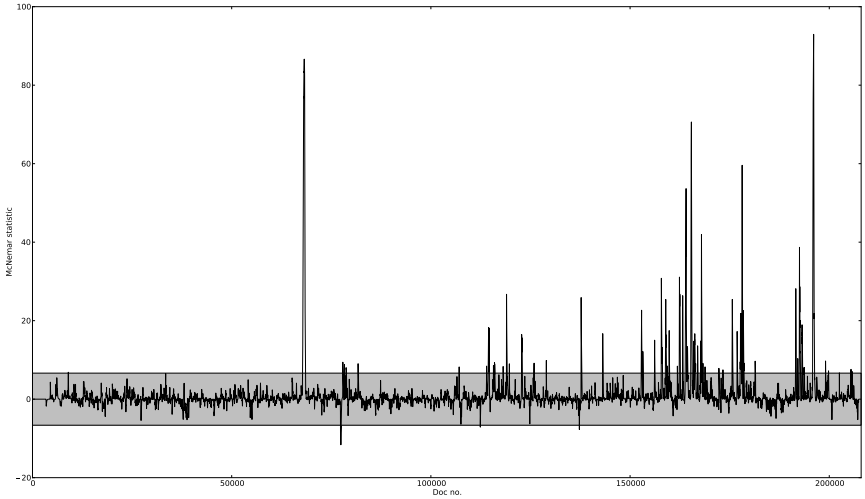
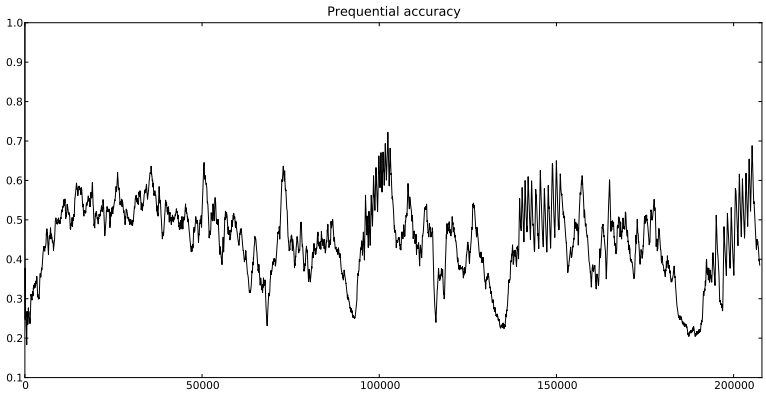
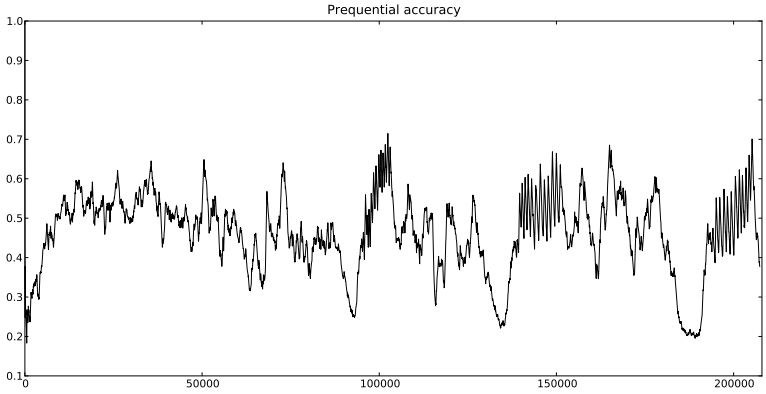


FIGURE 8.5: McNemar statistic for the PubMed dataset with fading factor α=0.99. The grey zone represents the area without significant differences in performance with a significance level of 0.01. The figure shows that the performance of the classifiers is not statistically different for the 95.5% of the documents

range during the 95.5% of the examples, which means that, in general, the difference of the performances using exact and approximate attributes is not statistically significant. Put in other words, we have succeeded in computing approximate statistics using a reduced space, without significantly modifying the behaviour of the algorithm. Note that in the last quarter of the execution for the PubMed dataset there are more and more points where a significant difference is detected. This is due to the phenomenon of *sketch saturation*: when many examples with low count are registered in the sketches, there are more hash function collisions, which yields worse approximations over time. This is an inherent limitation of sketch-based algorithms, which makes them work better for approximating large counters than smaller ones, as seen in Chapter 7. Regarding the Reuters dataset, we have found that the value of the McNemar statistic is 0 during the complete execution, which means that the behaviour of the exact and approximate algorithms is the same.



(a) Exact version. The last value of the prequential accuracy is 0.6351



(b) Sketch version. The last value of the prequential accuracy is 0.6094

FIGURE 8.6: Prequential accuracies of the classifier using the features provided by the exact and approximate calculation of the TF-IDF measure (PubMed dataset). The curves show a similar behaviour for the exact and sketch version. The performance drops are due to concept drifts to which the classifiers have to adapt. Fading factors with $\alpha = 0.99$ have been applied to the accuracy curve in order to give more weight to the most recent examples.

This was expected due to the low Spearman distance obtained in

Subsection 8.4.2.

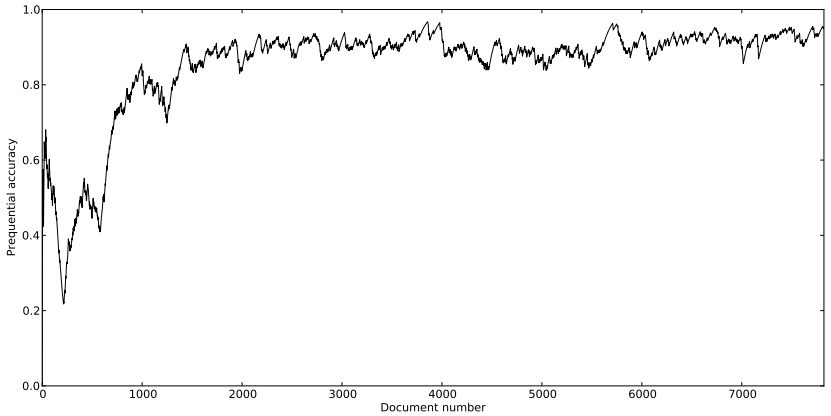
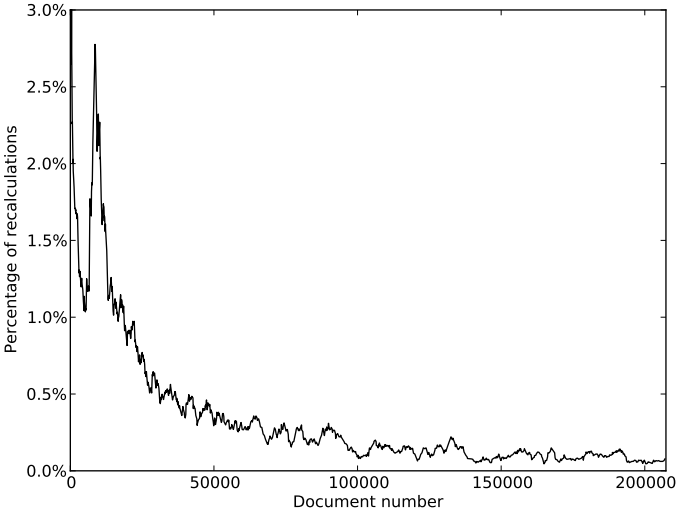


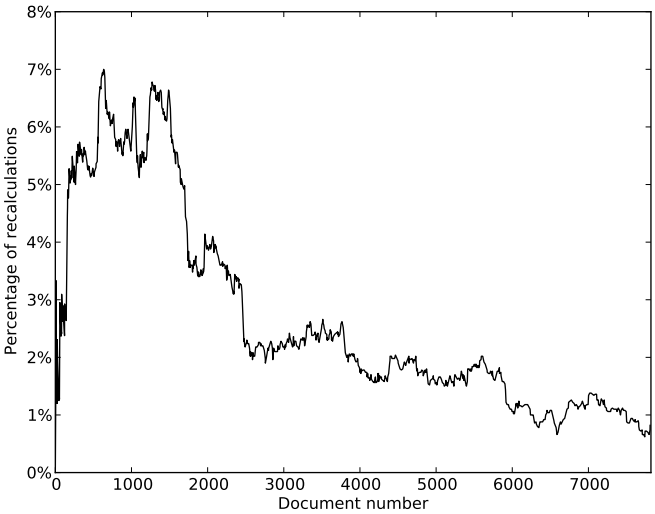
FIGURE 8.7: Prequential accuracies of the classifier for the Reuters dataset. The curves are identical for the exact and approximate version of the weighting function. The last value of the prequential accuracy is 0.9241

Figure 8.6 depicts the evolution of the prequential accuracy over time for the exact and approximate size for the PubMed dataset. The curves are remarkably similar. The sudden performance drops correspond to changes in the data distributions. The behaviour of the classifier is similar in the exact and approximate feature scenarios, which was already corroborated by the McNemar test depicted in Fig. 8.5. This adds further evidence to our observation that our approximation of weighting functions does not significantly affect the classifiers. Concerning the Reuters dataset, as we have seen, the behaviour of the exact and approximate algorithm is identical. The prequential accuracy for this dataset is shown in Figure 8.7.

We would also like to know the number of complete attribute list recalculations, since this is the most expensive step of the algorithm. To this end, Figures 8.8 depicts the percentage of the number of performed top-k attribute list recalculations with respect to the number of recalculations that would be necessary in the worst case, that is, if all the attribute lists had to be updated for every new document. In this figure we use moving averages over a sliding window of size



(a) PubMed dataset. Size of sliding window = 2500



(b) Reuters dataset. Size of sliding window = 500

FIGURE 8.8: Percentage of the number of complete top-k attribute list recalculations with respect to the worst case (that is, when all the lists had to be updated for each document). The figure shows the moving average of the percentages, using sliding windows of 2500 documents for the PubMed dataset and 500 for Reuters

2500 for PubMed and 500 for Reuters, in order to give more weight to the most recent examples. The figure shows that, although the number of recalculations is high at the beginning of the stream, it diminishes as new documents arrive over time, stabilizing at about 0.08% in the case of the PubMed dataset and 1% for the Reuters dataset.

To wrap up, from the experiments we can conclude that using the sketched version of the statistics for calculating weighting functions does not significantly change the behaviour of the algorithms, but reduces the used space, which is a desirable property for online learning scenarios. Moreover, the number of complete recalculations is small and diminishes over time. This is better than recalculating it for each document, given the complexity of this task, and also better than recalculating it periodically, since our approach guarantees that the attribute lists are always updated.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The current trend to increasingly large data sources is a double-edged sword. On the one hand, such data allow interesting new applications which combine research interest with an important socioeconomic impact. On the other hand, data sources are more and more complex and voluminous, which makes it hard to process them using standard technologies and tools. For this reason, new techniques are being researched in the fields of machine learning and data mining in order to tackle these challenges.

In particular, text mining is a discipline of paramount importance, given that a large proportion of the stored information is in the form of natural text: web pages, news articles, blogs, social networks, and others. As we have discussed, the automatic processing of natural text is a difficult task. One of the reasons for this is high dimensionality: systems that analyze text have to be able to reduce the space needed for representing the textual datasets without compromising their semantic contents.

Many interesting current data sources are open-ended, which means that there is not a closed dataset, but an unbounded stream of data that arrives continuously. This leads to several interesting challenges. Learning algorithms have to be incremental, that is, they have to be able to incorporate new information without learning the model from scratch. Moreover, they must comply with strict computational constraints in terms of time and space needed. The models have to be able to adapt to new concepts that emerge over time, since data streams do not have to be necessarily stationary from the point of view of the statistical distributions that generate the data.

In the case of text streams, we face some specific difficulties. As we have stated, it is necessary to reduce the dimensionality of textual data sources, given their high dimensionality. This has to be done in an incremental fashion, continuously incorporating new information to the preprocessing model. This is challenging from the point of view of the required computer resources. Moreover, this has the consequence that the selected attributes may change over time, which makes it necessary for learning models to be able to work under the assumption of a dynamic attribute space.

In this thesis, we have studied the problem of *text stream mining* and propose new techniques for dealing with massive, evolving and high-dimensional text streams in an efficient way. We now summarize the contents of the individual chapters.

9.2 Conclusions by chapter

In Chapter 2 we have provided an introduction to the field of text classification, and have discussed related fields of study and the main algorithms and techniques involved in this field. We have presented text classification as a subfield of text mining, the application of data mining techniques to natural language datasets. In this chapter we have discussed that the complexity of such data makes it necessary to use techniques from the field of information retrieval and natural language processing, especially for transforming the documents

from their raw form into a more adequate representation that captures the underlying semantics of the documents while reducing the dimensionality of the data. Once the documents have been transformed into a suitable representation, data mining algorithms can be used to obtain the desired models. Standard evaluation techniques can be used for studying the effectiveness of the obtained models, taking into account that the nature of textual data makes some measures more adequate than others. We have also mentioned some commonly used benchmarks for comparing text classification systems.

After that, in Chapter 3 we have shown that text mining techniques can be applied to feature extraction in the domain of DNA strings. We have proposed a parallel algorithm that is able to extract meaningful substrings from complete DNA strings. Such substrings can be used as words of different length, which allows to treat DNA strings as documents, that is, as word sequences. In contrast with previous works, the length of the features does not have to be fixed beforehand. We have shown that this approach can be used for string visualisation, as well as for mitochondrial DNA haplogroup classification, where competitive results are obtained using text mining techniques exclusively, without attributes provided by experts in biology.

Text classification is extended to the multilabel scenario in Chapter 4, focusing on email classification. We have argued that multilabel classification is important for many data sources, and have presented a study of the performance of different multi-label methods in combination with standard single-label algorithms using several specific multi-label metrics. We have shown that reducing the dimensionality spaces improves the learning performance, and that, most of the times, the difference in performance is statistically significant. The results achieved by SVM were uniformly better than for any other algorithm. The best multilabel methods were found to be RAkEL, EPPT and CLR. Overall, the best combination was RAkEL over SVM, but CLR over decision trees worked better when we only want to optimize Micro Precision. We found that label dependency was more important for recall than for precision.

In Chapter 5, we have introduced the concept of text stream mining. We have presented the GNUsmail framework for online email classification, which was used to carry out a study about the effect of different types of concept drift in several email datasets. From this study we could conclude that, although the process of adaptation improves the accuracy of the learning models, the concept drift was virtual rather than real, which means that it is caused by the skewness of the data distribution over time.

In Chapter 6 we have focused on the comparison of several feature selection and adaptive strategies for email foldering using the Enron corpus and our proposed ABC-DynF framework for adaptation. The ABC-DynF framework can work under a dynamic feature set, keeping a list of the top-k features which will be used by the learning model, and allowing to add new categories. We have not found evidence that allows us to assert that the classification accuracy is significantly different when a different number of lines are used to extract features. We can corroborate that the use of Iterative Bayes significantly improves the performance of the incremental Naïve Bayes. Updating the classification features also shows a significantly positive impact on classification. On the other hand, our results show that adapting to conceptual concept change by allowing a dynamical feature space consistently improves the performance in a statistically significant way. Thus, we conclude that keeping track of the most important features over time and including them in the classification model has a great importance when improving classification performance.

In Chapter 7, we have dealt with the process of vectorization of documents arriving from a stream, and have proposed weighting functions as a mechanism to summarize documents. More specifically, we have used weighting functions to extract the most relevant words from documents, in order to incrementally construct keyword lists and word clouds. We show that there is a severe complexity problem involved in this computation, so we have proposed TF-SIDF / SBM25 as a novel solution that uses sketch-based algorithms to cope with this. We have found that sketch configurations with few hash functions are preferable given the same total size. The results

show that using a sketch for computing the weighting function and another one to compute top-k of all labels is the most appropriate solution for maintaining word clouds.

Finally, in Chapter 8 we have dealt with an online learning scenario where text documents arrive from a stream, and only the most important words should be used as features in order to avoid complexity problems. It is important to use algorithms able to deal with the dynamic feature spaces in an efficient way, reducing the necessary space. Our main contribution in this chapter was STFSIDF, an approach to approximating a modified version of the TFIDF function using the sketches and Bloom filters as a basic tool for reducing the needed space, making it possible to maintain dynamic feature spaces in data streams with reduced time and space requirements. The experimental results show that, in general, there is no significant difference in the behaviour of the algorithm when using exact or approximate functions. Additionally, we have found that the number of attribute list recalculations is affordable. Thus, we can conclude that our proposal is of practical utility for text data streams, since it enables a noticeable complexity reduction without significantly affecting the performance of classification algorithms.

9.3 Future work

There are several issues that are still open, and different improvements and extensions can be made over proposals presented in this thesis. Some possibilities for future work are the following:

- A possibility for future research is to extend the GNUsmail framework to other text-based domains, such as blog entries or tweets. The current architecture of the framework, although designed with a low level of coupling, is specialized in electronic mail, and includes out-of-the-box components for email parsing and feature extraction. The architecture could be extended by allowing other data sources, such as web pages, blog feeds, tweets or others. This would involve implementing other

data extractors, as well using attributes relevant to each kind of data.

- In the future, we plan to extend the study presented in this thesis to different item frequency estimation algorithms that use sublinear space.

- We are interested in studying the performance of the TF-SIDF approach with alternative weighting measures (currently, only TF-IDF and Okapi BM25 are used).

- Another line of future work is the adoption of measures to handle the degradation of the sketch performance due to saturation. This saturation is inherent in sketch-based algorithms because as time goes by more elements are stored in the sketch, which results in more collisions taking place due to the effect of having many elements with small counts. This saturation can be delayed by selecting different sketch configurations or using larger sketches, but it cannot be totally avoided in the long term. We plan to study measures to alleviate this problem, such as periodically resetting the counters.

- As future work, we aim to study the multilabel classification problem under the optic of on-line multi-label learning.

- We plan to use different weighting functions for feature selection. We also plan to use other classifiers are base learners for the ABC-DynF framework.

- We are also interested in developing new stream mining methods for learning in the dynamic feature set scenario, specially at the presence of concept drift.

- Finally, we also consider the use of other kinds of features for classification, such as semantic annotations.

Appendix A

Resumen en español

A.1 Introducción

En los últimos años ha tenido lugar un vertiginoso incremento en la cantidad de datos disponibles en forma electrónica, en gran parte debido al surgimiento de la Web 2.0 y las aplicaciones móviles. Gran parte de estos datos están disponibles en forma de lenguaje natural, ya sea páginas web, artículos científicos, noticias de periódicos, correo electrónico, o contenidos de redes sociales. Esta explosión de datos ha hecho posible la aparición de aplicaciones tan novedosas como recomendadores personalizados, análisis automático de sentimiento respecto a un producto, detección automática de correo no deseado, segmentación de mercado o incluso análisis automático de literatura médica para obtener conocimiento nuevo sobre enfermedades. Estas aplicaciones son importantes no sólo desde el punto de vista de los desafíos científicos asociados a ellas, sino también por su impacto económico y social. El hecho de poder extraer conocimiento y patrones que no estaban disponibles con anterioridad es de gran importancia para las empresas, ya que las sitúa en una mejor posición para tomar las medidas más adecuadas para continuar con éxito su crecimiento.

Por estas razones, cada vez es más importante el desarrollo de técnicas capaces de explotar y dar así un valor añadido la gran cantidad de

datos disponibles. Esta tarea está lejos de ser trivial, y necesita la implicación de campos científicos de diversa índole, tales como la estadística, la inteligencia artificial, tecnologías de bases de datos y almacenamiento, sin olvidar conocimiento de los expertos del dominio del problema concreto a ser tratado, ya que, como hemos visto, una gran variedad de dominios de conocimiento se pueden beneficiar de la explotación automática de datos. Como condición previa para dar utilidad a los datos es necesario explorar y mejorar la calidad de estos, ya que pueden estar incompletos e incluso presentar inconsistencias internas y ambigüedades en el mensaje. También es importante unificar fuentes de datos heterogéneas, ya que en aplicaciones reales puede haber documentos de muy distinto tipo que, sin embargo, hablen de lo mismo. Por ejemplo, una noticia de alcance generará artículos de opinión, tweets o noticias en periódicos, por lo que es necesario unificar todas estas fuentes para obtener la mayor cantidad posible sobre el hecho de interés. Por otra parte, es necesario aplicar técnicas adecuadas de modelado de documentos, de forma que puedan ser procesados automáticamente. Es necesario disponer de herramientas para evaluar el éxito de los modelos desarrollados con respecto a los problemas de interés, sobre todo para poder comparar diferentes estrategias, algoritmos y tecnologías. En resumen, nos enfrentamos a una tarea altamente multidisciplinar. La minería de datos ha demostrado ser una disciplina esencial para conseguir la extracción de información útil a partir de datos almacenados, proporcionando un amplio abanico de herramientas para las distintas tareas que se necesitan: preprocesado de datos, extracción de información, análisis predictivo o agrupamiento de instancias de datos similares, y evaluación. Desde un punto de vista general, el proceso de minería de datos consiste en el análisis automatizado de grandes cantidades de datos para extraer patrones interesantes que, hasta ahora, no eran conocidos. Estos patrones pueden consistir en grupos de registros de datos (mediante el análisis clúster), instancias poco usuales (detección de anomalías) y dependencias (reglas de asociación), así como modelos que predicen a qué grupo corresponde una instancia dada (clasificadores), y funciones que son capaces de asignar un valor numérico a cada instancia (regresión). La minería de datos ha sido aplicada con éxito en ámbitos muy diversos, como

análisis de patrones de fraude en entidades bancarias o análisis de comportamientos de usuarios en páginas web, analizando las páginas visitadas o los clicks que ha hecho el usuario. En nuestro caso, las instancias a estudiar son documentos en lenguaje natural, y nuestra tarea consiste en generar modelos capaces de predecir a qué categoría debe ser asignado cada documento, de entre un conjunto de categorías preestablecidas.

No resulta sorprendente que la mayor parte de la información se encuentre almacenada hoy en día en forma de texto en lenguaje natural. Gracias a Internet es posible acceder a una inmensa cantidad de páginas web, blogs y artículos sobre cualquier tema que podamos imaginar. Una de las virtudes de la Web 2.0 es que prácticamente cualquier persona con una conexión a Internet puede generar contenidos. El uso de correo electrónico es hoy en día universal entre los usuarios de internet, y la escritura en blogs o Twitter está también muy extendida. Desgraciadamente, el análisis computacional de información no estructurada, como es el caso del lenguaje natural, es una tarea compleja, ya que implica que la máquina pueda de alguna forma captar el significado que el emisor pretendía darle a su mensaje. Las razones para esta dificultad inherente al procesamiento de lenguaje natural son variadas. Por una parte está el hecho de que los idiomas humanos, en contraste con los lenguajes de programación, están plagados de ambigüedades en términos de gramática y vocabulario. El conocimiento del dominio, que en muchos casos podría ayudar a resolver estas ambigüedades, no siempre es fácil de obtener, o, cuando está disponible, no siempre está claro cómo aplicarlo para mejorar el desempeño de los algoritmos.

En esta tesis exploramos diversos aspectos de la clasificación de textos, incluyendo su aplicación a datos biológicos (cadenas de ADN representadas como documentos de texto), médicos (artículos de literatura médica) y correo electrónico, y afrontamos diversas dificultades que surgen al analizar fuentes de texto voluminosas, que posiblemente no están disponibles de forma completa, sino que sus documentos llegan continuamente a lo largo del tiempo, debiendo ser incorporados a los modelos de aprendizaje. En la siguiente sección

desgranamos con más detalle nuestra motivación y las dificultades asociadas a los problemas con los que nos enfrentamos en esta tesis.

A.2 Motivación

Una dificultad notable de los conjuntos de datos en lenguaje natural en general es su alta dimensionalidad inherente, ya que los principales atributos de clasificación con los que contamos son las palabras que conforman los documentos. El gran tamaño de los vocabularios asociados a las fuentes de datos hace necesario el desarrollo de técnicas escalables capaces de hacer frente a espacios de atributos de alta dimensionalidad, ya que, por una parte, esta alta dimensionalidad causa problemas desde el punto de vista de la complejidad computacional (tanto en tiempo como en memoria), y por otra parte, perjudica a la capacidad predictiva de los modelos de aprendizaje, debido al efecto del sobreajuste provocado por un espacio de atributos demasiado complejo.

En esta tesis nos planteamos además la aplicación de técnicas de minería de textos a datos no estrictamente lingüísticos, si no más bien al lenguaje biológico por excelencia: las cadenas de ADN. Intuitivamente, podemos pensar en el ADN como en un lenguaje: está formado por cadenas de símbolos de un alfabeto (los cuatro tipos de nucleótidos) con una semántica (el ADN determina el fenotipo, es decir, las características observables de los seres vivos). En general, la extracción de palabras a partir de documentos es la parte más sencilla del procesado de lenguaje natural, ya que las palabras están delimitadas por espacios u otros símbolos de puntuación, con la excepción de algunos lenguajes como el chino o el japonés. Esto no es así en el ADN, ya que a priori no tenemos ninguna forma de saber cuáles son sus subcadenas más significativas de entre todas las posibles subcadenas que se encuentran contenidas en la secuencia de ADN. Por eso, es necesario desarrollar técnicas que permitan extraer atributos a partir de las cadenas de ADN para poder usarlos para aplicar minería de datos. Dada la gran longitud de este tipo de cadenas, se necesitan algoritmos eficientes para llevar a cabo esta

extracción de atributos, ya que existe una gran cantidad de subcadenas posibles a analizar. Una condición que deben cumplir las subcadenas para poder ser consideradas atributos de clasificación es que aparezcan con frecuencia en la secuencia de ADN. Sin embargo, esto no es suficiente por sí mismo, ya que, si una cadena es frecuente, sus subcadenas también lo serán necesariamente. Por lo tanto hay que emplear filtros adicionales, que permitan evaluar hasta qué punto una subcadena es relevante para poder ser considerada como una "palabra" dentro de ADN, y, por lo tanto, como un atributo de clasificación para las técnicas de minería de texto.

Un tipo diferente de desafío con el que frecuentemente nos encontramos al trabajar con conjuntos de textos es la clasificación multietiqueta. En muchas ocasiones, asignar una sola etiqueta a cada documento no es suficiente, ya que un mismo documento puede estar asociado de forma natural a diferentes temáticas o categorías, y seleccionar sólo una puede ser demasiado restrictivo. Esto da origen al paradigma de clasificación *multietiqueta*. Aunque una estrategia simple para clasificación multietiqueta es entrenar a un clasificador por cada etiqueta de forma independiente, esto no tiene en cuenta que las etiquetas no son independientes entre sí, sino que la aparición de unas etiquetas tiene influencia en la posibilidad de que otras etiquetas aparezcan también. Por ejemplo, las noticias relacionadas con economía pueden ir relacionadas también con política, pero más raramente con entretenimiento. Por lo tanto, los métodos de clasificación multietiqueta deben incorporar esta información sobre dependencias en su proceso de aprendizaje. Igualmente, los procedimientos de evaluación tienen que tener en cuenta la naturaleza multietiqueta de estas fuentes de datos, por lo que hay que usar medidas de evaluación específicas.

En esta tesis nos enfrentamos a fuentes de datos no acotadas, es decir, a documentos procedentes de un flujo de datos, en contraste con los conjuntos de datos estáticos. En otras palabras: no disponemos a priori de todo el conjunto de datos, sino que los documentos van llegando poco a poco, debiendo ser incorporados al modelo de clasificación conforme aparecen. Esto de pie a hablar de clasificación de *flujos de datos*. La minería de flujos de datos surge de forma natural

en escenarios tales como tráfico de red, datos de sensores, búsquedas web, o correos electrónicos. Este tipo de escenarios está asociado a fuertes restricciones computacionales, tanto desde el punto de vista del tiempo como del espacio necesarios. Las instancias no pueden ser almacenadas, por provenir de un flujo de datos ilimitado, lo que hace que cada una sólo se pueda procesar una vez, o, a lo sumo, un número limitado de veces. Los algoritmos usados deben ser incrementales, es decir, deben ser capaces de procesar las instancias una a una, sin tener que volver a revisar instancias anteriores. Además, el modelo de aprendizaje debe estar listo para predecir en cualquier momento, por lo que el aprendizaje y las predicciones se intercalan en el tiempo. Por estas razones, los algoritmos de minería de datos tradicionales no nos sirven en general para analizar flujos de datos.

Una dificultad asociada a los flujos de datos es la aparición de cambios de concepto. Este fenómeno consiste en que la distribución de los datos, así como la relación subyacente entre datos y etiquetas, está sujeta a cambios a lo largo del tiempo. Esta problemática se da por ejemplo en el correo electrónico, donde nuevos temas de conversación surgen a lo largo del tiempo, o en flujos de noticias, donde aparecen temas de actualidad que deben ser incorporados a los modelos de clasificación ya existentes. Este fenómeno hace necesario que los modelos de aprendizaje deban ser capaces de, por una parte, detectar los cambios que ocurran, y, por otra, adaptarse a estos cambios, modificando los modelos de forma adecuada, sin que sea necesario recalcular el modelo desde cero. En esta tesis nos planteamos un estudio del efecto que tienen distintos tipos de cambio de concepto en el desempeño de clasificadores de correo electrónico, ya que el efecto será diferente dependiendo de si estamos ante un cambio de concepto "real" o "virtual", es decir, originado por desbalanceos en la distribución de las clases a lo largo del tiempo.

En el caso particular de los flujos de textos tenemos además el problema de los cambios de concepto de tipo *contextual*. Esto significa los términos del vocabulario más relacionados con las categorías, y por lo tanto usados como atributos de clasificación, cambian a lo largo del tiempo. Es natural que en flujos de texto aparezcan

nuevas palabras, así como que haya palabras que pierden su relevancia. Por ejemplo, durante la celebración de un cónclave para la elección del nuevo papa surgirán muchos términos relacionados con la iglesia católica, que perderán en parte su frecuencia en las noticias pasadas unas semanas. Esto es una consecuencia de la alta dimensionalidad de este tipo de datos, derivada del elevado número de palabras diferentes que aparecen en el lenguaje natural. Por lo tanto, se hace necesario contar con mecanismos para selección de atributos de forma incremental, así como clasificadores incrementales capaces de trabajar en un escenario de atributos dinámicos, es decir un espacio de atributos cambiante a lo largo del tiempo. Una de las principales motivaciones de esta tesis es precisamente la implementación y evaluación de mecanismos de selección incremental de atributos, reduciendo en la medida de lo posible la complejidad computacional para poder ser aplicadas a escenarios de flujos de datos con las restricciones mencionadas. También nos planteamos el estudio de estrategias para coordinar la actualización del espacio de atributos, ya que este se podría actualizar para cada nuevo documento, o bien sólo cada vez que se detecta un cambio de concepto.

Debido a las restricciones espaciales con las que nos enfrentamos en los flujos de datos, es necesario desarrollar estrategias para procesar los datos de forma que se ahorre espacio, a costa de pérdidas de precisión controladas. En particular, es necesario contar con algoritmos sublineales respecto al tamaño del vocabulario. Históricamente se han desarrollado diferentes algoritmos de compresión del espacio necesario para procesar flujos de datos, permitiendo pérdidas controladas de precisión como contrapartida de funcionar con una complejidad espacial más razonable. Un ejemplo importante de este tipo de estrategias son los algoritmos basados en *sketches*, así como los *filtros de Bloom*. Este tipo de algoritmos usan estructuras de datos probabilísticas, que proyectan los datos a un espacio de menor dimensionalidad mediante el uso de funciones hash. Estas estructuras son utilizadas a lo largo esta tesis para el resumen automático de documentos y etiquetas en flujos de textos, así como para la implementación de estrategias de extracción incremental de atributos.

En la siguiente sección resumimos cada uno de los capítulos que componen esta tesis.

A.3 Resumen de los capítulos individuales

- En el Capítulo 1 sirve de introducción a la tesis. En él introducimos nuestras motivaciones y objetivos. El hecho de que la mayor parte de los datos almacenada hoy en día en forma informática consista en lenguaje natural nos sirve como motivación para justificar la importancia que tiene desarrollar nuevas técnicas y algoritmos que permitan extraer información. Introducimos los conceptos más importantes que desarrollamos a lo largo de la tesis, y exploramos los desafíos que presenta el análisis automatizado de lenguaje natural, tales como la ambigüedad lingüística, cambio de concepto, alta dimensionalidad y los altos requisitos computacionales. Por último, proponemos los objetivos de esta tesis doctoral e introducimos a grandes rasgos cada uno de los capítulos de la tesis.

- En el Capítulo 2 proporcionamos una introducción general a la clasificación de textos en lenguaje natural. El objetivo principal de este capítulo es que esta tesis sea lo más autocontenida posible. Presentamos la minería de textos como una disciplina encuadrada dentro de la minería de datos, y explicamos la relación entre minería de datos, aprendizaje natural y extracción de información, definiendo los términos que serán utilizados a lo largo del resto de la tesis, y proporcionando ejemplos de las principales tareas de la minería de datos. Explicamos de forma general las fases que constituyen un proyecto KDD (selección de datos, preprocesamiento, transformación de los datos, modelado, evaluación y despliegue). A continuación repasamos la historia del procesamiento de lenguaje natural y sus tareas más destacadas. Hacemos también un repaso breve de algunos conceptos básicos de extracción de información.

Por último, formalizamos la disciplina de clasificación de textos, haciendo mención a modelos de representación de documentos, técnicas de indexación y reducción de dimensionalidad, así como los algoritmos más importantes para la clasificación en sí misma, tales como los algoritmos probabilísticos, clasificadores lineales, máquinas de vectores de soporte, aprendices basados en ejemplos y árboles de decisión. Por último, hablamos de las principales técnicas de evaluación, incluyendo medidas de evaluación y procedimientos como la validación cruzada.

- En el Capítulo 3 exploramos la aplicación de las técnicas de minería de texto a la extracción de características, visualización y clasificación de ADN. El ADN puede ser visto como cadenas extremadamente largas de símbolos. Estas cadenas tienen además una semántica asociada, ya que determinan las características de cada ser vivo, es decir, su fenotipo. Con esto en mente, podemos afirmar que el ADN es una especie lenguaje biológico al que le podemos aplicar técnicas que han sido desarrolladas para lenguaje natural. Al igual que algunos lenguajes humanos como el chino o el japonés, sus unidades constitutivas (sus "palabras") no están delimitadas por separadores, por lo que es necesario aplicar técnicas de naturaleza estadística para descubrir las subcadenas más relevantes de una cadena del ADN. Dada la gran longitud de estas cadenas, nosotros proponemos una versión paralela del algoritmo SANSPOS para extraer factores frecuentes, junto con una función de interés usada en el aprendizaje de reglas de asociación (AV, iniciales de *Added Value*) para filtrar las subcadenas más interesantes, que serán usadas como las palabras que conforman la cadena de ADN. Como resultado de este procedimiento obtenemos una representación del ADN como una bolsa de palabras, o, visto de otra forma, como un documento al que podemos aplicar técnicas de minería de textos. En este capítulo usamos esta representación para dos tareas. La primera consiste en visualizar las cadenas de ADN en función de sus palabras más significativas, lo que permite tener una representación gráfica de los atributos más importantes de una secuencia de ADN.

Adicionalmente, se puede usar esta misma estrategia para representar la diferencia entre dos secuencias de ADN, lo que permite visualizar cuáles son los atributos que mejor diferencian una secuencia de otra. La segunda tarea consiste en clasificar cadenas de ADN mitocondrial según su haplogrupo, usando los atributos seleccionados como entrada a los algoritmos de clasificación.

- El Capítulo 4 está dedicado al aprendizaje multietiqueta para clasificación de textos. En este capítulo presentamos las principales tareas y algoritmos, y las usamos para analizar el efecto de la selección de atributos en un conjunto de datos multietiqueta. Este capítulo nos sirve para presentar las principales tareas y métodos de aprendizaje multietiqueta, centrándonos en los métodos denominados *de transformación de problema*, que consisten en reducir los problemas multietiqueta en una colección de problemas monoetiqueta, integrando los clasificadores individuales en el clasificador multietiqueta final. Discutimos también las principales medidas de evaluación para aprendizaje multietiqueta, y proponemos una evaluación de diferentes algoritmos multietiqueta a la clasificación de correo electrónico. Además, estudiamos el efecto de reducir la dimensionalidad del problema con respecto a diferentes medidas de evaluación, usando técnicas clásicas de selección de atributos.

- En el Capítulo 5 introducimos fuentes de datos textuales no acotadas (flujos de documentos de texto), las cuales presentan limitaciones en cuanto al tiempo y espacio para procesar los ejemplos. Además, estás fuentes de datos están sujetas a cambios de concepto. En primer lugar presentamos los principales conceptos teóricos de la minería de flujos de datos, incluyendo una discusión sobre cambio de concepto gradual o repentino, así como virtual y real. Repasamos también algunos de los algoritmos más destacados para aprendizaje en flujos de datos, incluyendo una descripción de detectores clásicos de cambio de concepto y las medidas de evaluación más adecuadas para flujos de datos. En la segunda parte

del capítulo ofrecemos un estudio experimental sobre la ocurrencia de cambio de concepto en diversos conjuntos de datos de correo electrónico. Presentamos literatura relevante sobre la clasificación de correo electrónico, y pasamos a describir la plataforma de código abierto presentada en este capítulo, GNUsmail. Tras un análisis exploratorio de conjuntos de datos del corpus Enron de correo electrónico, comparamos distintas configuraciones de GNUsmail, utilizando diferentes algoritmos y detectores de cambio de concepto. Llegamos a la conclusión de que el cambio de concepto observado es de tipo virtual, lo cual hace que, aunque los algoritmos de detección y adaptación a cambio de concepto sean capaces de mejorar el desempeño de lo algoritmos en muchos casos, no lo hagan siempre de forma significativa desde el punto de vista estadístico. Utilizamos el test de McNemar como herramienta estadística de comparación de algoritmos en flujos de datos. Observamos también que los algoritmos basados en árboles de decisión, tales como VFDT (árbol de Hoeffding), no funcionan bien en estos conjuntos de datos.

- En el Capítulo 6 introducimos la problemática de los espacios de atributos dinámicos junto con cambios de concepto en flujos de texto. Introducimos un nuevo tipo de cambio de concepto, el contextual. Presentamos ABC-DynF, un framework para clasificación de flujos de datos que mantiene una lista de los atributos más relevantes en cada momento (usando una función de relevancia de atributos, concretamente chi-cuadrado), y que es capaz de actualizar el espacio de atributos usado según la evolución de esta lista. Adicionalmente, ABC-DynF hereda del framework de clasificación AdPreqFr4SL un mecanismo de detección de cambio de concepto basado en las P-Charts de Shewart, que busca mantener un equilibrio entre sesgo y sobreajuste. Se usa Iterative Bayes para mejorar las estimaciones de probabilidad del algoritmo bayesiano en el que está basado ABC-DynF. En la sección de experimentación, ABC-DynF es evaluado en el problema de clasificación de correo electrónico usando varios datasets del corpus

Enron. Se prueban diferentes estrategias de reducción de dimensionalidad (incluyendo poner un límite en el número de líneas usadas por cada documento), y se estudia el efecto de usar Iterative Bayes para mejorar las estimaciones de probabilidad (mejorando por tanto el algoritmo base). Se comparan diversas estrategias de control de cambio de concepto, para estudiar el efecto de la monitorización de cambio de concepto. Por último, se estudia si el espacio de atributos debe ser actualizado continuamente, o sólo cuando se ha detectado un cambio de concepto.

- El Capítulo 7 se centra en el uso de funciones de pesado sublineales para resumen de flujos de datos. Más específicamente, usamos las funciones TF-IDF y BM25 para seleccionar los términos más importantes de cada documento del flujo, para, por una parte, extraer palabras clave (resumen del documento) y, por otra, construir nubes de palabras para cada categoría a partir de las palabras relevantes más frecuentes (resumen automatizado de categorías). Ofrecemos un repaso a la literatura relevante sobre resumen automatizado de documentos, para a continuación definir formalmente las tareas relacionadas, tales como el problema de encontrar ítems frecuentes (en su versión exacta y aproximada), o el problema de encontrar los k elementos con mayor frecuencia dentro de un flujo de datos. Mostramos que esta es una tarea para la que se necesita una gran cantidad de espacio, debido a la alta dimensionalidad de los datos, producida por el tamaño de los vocabularios, por lo que se necesitan algoritmos específicos (basados en contadores o basados en sketches). Posteriormente nos centramos en las técnicas basadas en sketches para aproximación de frecuencias en flujos de datos, especialmente el algoritmo Count-Min Sketch, junto con sus propiedades teóricas más importantes. Proponemos una nueva metodología, que denominamos TF-SIDF para resumen aproximado de flujos de documentos de texto, aplicándolos a conjuntos de datos médicos y de correo electrónico. Esta metodología se basa en usar sketches para aproximar los contadores necesarios para evaluar diferente funciones de relevancia de atributos, en concreto TF-IDF y BM25.

Demostramos empíricamente que esta técnica obtiene buenas aproximaciones ahorrando espacio, sin que haya una diferencia estadísticamente significativa con los resultados obtenidos usando contadores exactos. Por lo tanto, resulta una estrategia escalable a vocabularios de gran tamaño, y, en consecuencia, esta propuesta es adecuada para ser usada en resúmenes automáticos de flujos de texto.

- En el Capítulo 8 exploramos la selección de atributos incremental en flujos de texto. Justificamos la utilidad de los algoritmos sublineales para almacenar información relativa a los atributos de clasificación, y mostramos que los cambios de concepto en flujos de documentos son problemáticos para la selección de atributos. Introducimos algunas técnicas adicionales para resumen automatizado de flujos de datos y presentamos STFSIDF, una propuesta para llevar a cabo esta selección de forma eficiente en cuanto al tiempo y al espacio necesarios para la computación. STFSIDF está basado en aplicar algoritmos basados en sketches para reducir el espacio necesario para almacenar los contadores para calcular los pesos de los distintos atributos, disminuyendo además el número de actualizaciones completas de la lista de atributos, evitando que se recalculen continuamente todos los atributos, a diferencia de otras propuestas anteriores para selección incremental de atributos. Tras formalizar nuestra propuesta, presentamos una evaluación desde dos perspectivas diferentes: una evaluación intrínseca del modelo, con el objetivo de estudiar la mejor parametrización, de forma que los atributos obtenidos mediante el algoritmo exacto sean lo más parecidos posibles a los obtenidos mediante STFSIDF, y una evaluación extrínseca, donde aplicamos nuestra metodología para clasificación de correo electrónico y de artículos de literatura médica. Estudiamos mediante tests estadísticos la diferencia de desempeño de los clasificadores que se obtiene de usar los atributos obtenidos por los algoritmos exactos y por los aproximados. Mostramos que esta diferencia no es significativa desde el punto de vista estadístico. Al ser los algoritmos aproximadas más eficientes en tiempo y espacio, podemos concluir

que hemos obtenido un esquema de selección de atributos adecuado para su uso en flujos de datos.

- Por último, en el Capítulo 9 ofrecemos las conclusiones de esta tesis, y anticipamos el trabajo futuro que se deriva de ella.

A.4 Aportaciones de esta tesis

Los objetivos que perseguimos en esta tesis están relacionados con la minería de flujos de texto en lenguaje natural, en escenarios con espacio de atributos dinámico y cambios de en la distribución estadística seguida por los datos a lo largo del tiempo. Se estudian técnicas para atajar los problemas anteriormente mencionados, y los comparamos con el estado del arte en clasificación de flujos de texto. Nuestras principales aportaciones han sido las siguientes:

- Revisión del estado actual de diferentes áreas de trabajo relacionadas con clasificación de textos, incluyendo clasificación multietiqueta, minería de flujos de datos, detección de cambio de concepto, evaluación de clasificadores online y algoritmos aproximados para procesado de flujos de datos. Este estudio sirve como base para el desarrollo posterior de nuestras soluciones algorítmicas y en forma de estructuras de datos.

- Propuesta y evaluación de una versión paralela de SANSPOS para extraer subcadenas frecuentes a partir de secuencias de ADN, junto al uso de la función AV para filtrar las palabras más significativas.

- Uso de la estrategia anteriormente mencionada para visualizar cadenas de ADN según sus subcadenas más significativas, así como la diferencia entre organismos a nivel de subcadenas de ADN.

- Uso del espacio de atributos extraído siguiendo esta misma metodología para clasificación de cadenas de ADN mitocondrial en diferentes haplogrupos. A diferencia de otras propuestas anteriores, no es necesario proporcionar a los algoritmos un conjunto prefijado de atributos de clasificación, sino que son descubiertos automáticamente, obteniendo resultados competitivos con el estado del arte.

- Evaluación de diferentes algoritmos multietiqueta con respecto a diferentes medidas de desempeño, así como del efecto de reducir el espacio de atributos. Nos centramos en el dominio de la clasificación de correo electrónico. Estudiamos la relación existente entre diferentes medidas y exploramos la influencia de los procesos de selección de atributos y preprocesamiento. Observamos que el uso de preprocesamiento lingüístico mejora significativamente el desempeño de los algoritmos. Concluimos que es más importante tener en cuenta las dependencias entre etiquetas en las medidas de evaluación basadas en recall.

- Presentamos GNUsmail, una plataforma escrita en Java para clasificación de flujos de correo electrónico. Se incluyen diferentes herramientas para extracción de correo electrónico, selección de atributos y algoritmos dinámicos para clasificación y gestión del cambio de concepto. Esta herramienta se publica en forma de código abierto, poniéndola a disposición de la comunidad.

- Usamos esta misma herramienta para estudiar diferentes estrategias de selección de atributos y adaptación a cambios de concepto. Vemos que en los flujos de correo hay cambio de concepto, que en muchas ocasiones es de tipo virtual.

- Se propone ABC-DynF, un framework de clasificación para flujos de datos que, por una parte, suporta un espacio de atributos dinámico a los largo del tiempo, y, por otra parte, ofrece estrategias de control de cambio que previenen la sobreadaptación y el sesgo (bias) de los clasificadores.

- Estudiamos la reducción de líneas usadas en un documento como estrategia para reducción de dimensionalidad. Concluimos que es una estrategia aceptable.

- Usando ABC-DynF, estudiamos el efecto de mejorar el clasificador (refinando los parámetros relativos a la probabilidad en un clasificador Bayesiano) con respecto a mejorar el espacio de atributos, concluyendo que es más importante mejorar el clasificador de cara al desempeño.

- Estudiamos si vale la pena actualizar continuamente el espacio de atributos usados por ABC-DynF, o si es suficiente con invocar actualizaciones sólo en caso de que se detecte cambio de concepto, o cuando se detecte que el clasificador ha dejado de mejorar su desempeño. Descubrimos que, aunque los resultados son mejores si se actualiza continuamente el espacio de atributos, la diferencia no es significativa.

- Estudiamos soluciones sublineales para vectorización incremental de documentos, para cumplir con los requisitos computacionales que exhiben los flujos de datos, especialmente los textuales, dada su alta dimensionalidad intrínseca. Proponemos y analizamos el uso de algoritmos basados en la noción de *sketches* con el objetivo de mantener estadísticas sobre términos del vocabulario, admitiendo pérdidas controladas de precisión para lograr funcionar con complejidad espacial sublineal. Concluimos que esta estrategia ahorra espacio sin afectar significativamente al desempeño. Estudiamos diferentes estrategias de configuración de nuestra propuesta, a la que le damos el nombre de TF-SDIDF / SBM25, observando que es mejor mantener sketches separados para el problema de los top-k atributos y el de la estimación de frecuencias.

- Proponemos el mecanismo citado anteriormente para extracción de palabras clave y computación de nubes de palabras en flujos de texto, como estrategia para resumen automatizado de documentos.

- Asimismo, proponemos STFSIDF, un nuevo algoritmo eficiente en tiempo y espacio para selección incremental de atributos

en flujos de texto. Usamos funciones de pesado que evitan el recálculo continuo de las estadísticas asociadas a cada término del vocabulario.

- En relación con el último punto, se propone y estudia el uso de algoritmos aproximados para reducir la complejidad espacial asociada a la tarea anterior, y estudiamos cómo optimizar la parametrización de nuestra propuesta para conseguir aumentar la precisión usando el mismo espacio. Concluimos que el uso de algoritmos aproximados no afecta negativamente al desempeño de los clasificadores.

Appendix B

Conclusiones y trabajo futuro (español)

B.1 Conclusiones

La tendencia actual a fuentes de largos cada vez más largas es un arma de doble file. Por una parte, esos datos hacen posibles aplicaciones novedosas e interesantes, que combinan interés desde el punto de vista de la investigación con un importante impacto socioeconómico. Por otra parte, la fuentes de datos son cada vez más complejas y voluminosas, lo que dificulta su procesado usando tecnologías y herramientas clásicas. Por esta razón, en los campos de la minería de datos y aprendizaje computacional se están investigando técnicas que permitan hacer frente a estos desafíos.

En particular, la minería de textos es de gran importancia, dado que una gran parte de la información almacenada hoy en día está en forma de lenguaje natural: páginas web, artículos de noticias, blogs, redes sociales, y otras. Como hemos discutido previamente, el procesado automático de texto natural es una tarea difícil por varias razones. Una de ellas es su alta dimensionalidad: los sistemas que analizan textos deben ser capaces de reducir el espacio necesario para representar las fuentes de datos de textos sin comprometer el contenido semántico.

Muchas fuentes de datos de la actualidad son ya no sólo cada vez más grandes, sino que además no están acotadas, lo que significa que no hay un conjunto de datos cerrado, sino un flujo de datos que llega continuamente. Esto llega a desafíos interesantes para la minería de datos. Los algoritmos de aprendizaje tienen que ser incrementales, esto es, tienen que ser capaces de incorporar información nueva sin tener que reaprender el modelo. Es más, deben procesar los datos de forma que se respeten limitaciones computacionales en términos del espacio y tiempo necesarios. Los modelos deben ser capaces de adaptarse a los nuevos conceptos que surgen a lo largo del tiempo, ya que los flujos de datos no tienen que estar gobernados por distribuciones estadísticas de tipo estático.

En el caso de los flujos de texto, nos enfrentamos a algunas dificultades específicas. Como hemos dicho previamente, es necesario reducir la dimensionalidad de las fuentes de datos de texto, a causa de su alta dimensionalidad. Pero, en el caso de los flujos de texto, esto debe ser hecho en forma incremental, incorporando nueva información al modelo de preprocesado de forma continua. Esto es un desafío desde el punto de vista de los recursos computacionales requeridos. Además, esto tiene como consecuencia que los atributos seleccionados pueden cambiar a lo largo del tiempo, por lo que los modelos de aprendizaje deben ser capaces de asumir un espacio de atributos dinámico.

En esta tesis estudiamos el problema de la *minería de flujo de textos* y proponemos nuevas técnicas para tratar eficientemente con flujos de datos cambiantes y de alta dimensionalidad. A continuación resumimos las principales conclusiones de cada capítulo.

B.2 Conclusiones por capítulo

En el capítulo 2 hemos introducido el campo de la clasificación de textos, discutiendo campos de estudios relacionados, así como las principales técnicas y algoritmos. Presentamos la clasificación de textos como una subdisciplina de la minería de textos, es decir, la aplicación de técnicas de minería de datos a datasets compuestos

de documentos en lenguaje natural. En este capítulo hemos visto que la complejidad de este tipo de datos hace necesario hacer uso de técnicas de extracción de información y procesamiento de lenguaje neutral, especialmente para transformar los documentos de su forma original a un tipo de representación más adecuada desde el punto de vista informático, que capture la semántica subyacente de estos documentos mientras se reduce la dimensionalidad de los datos y se eliminan atributos irrelevantes y redundantes. Una vez que los documentos han sido transformados a una representación adecuada, se pueden usar algoritmos de minería de datos. Las técnicas de evaluación estándar pueden ser usadas para estudiar la efectividad de los modelos obtenidos, teniendo en cuenta que la naturaleza de las fuentes de datos textuales hacen que unas medidas de evaluación puedan tener más sentido que otras. Hemos mencionado también algunos benchmarks comunes para comparar sistemas de clasificación de texto.

A continuación, en el capítulo 3 hemos mostrado que las técnicas de minería de textos pueden ser aplicadas para extracción de características en el dominio de las secuencias de ADN. Hemos propuesto un algoritmo paralelo capaz de extraer subsecuencias significativas de las cadenas completas de ADN. Estas subcadenas pueden ser usadas como palabras, lo que permite trabajar con cadenas de ADN como si se trataran de documentos, es decir, como una secuencia de palabras. A diferencia de otros trabajos, la longitud de los atributos de clasificación no tiene que ser fijado previamente. Hemos mostrado que esta idea puede ser usada para visualización de cadenas, así como para clasificación de haplogrupos de ADN mitocondrial, obteniendo resultados competitivos usando exclusivamente técnicas de minería de textos, sin atributos propuestos por expertos en biología.

Después, en el capítulo 4 hemos extendido la clasificación de textos al escenario multietiqueta, explicando la importancia de esta paradigma para muchas fuentes de datos y centrándonos en el caso de la clasificación de correo electrónico. Hemos presentado un estudio del desempeño de diferentes métodos multietiqueta en combinación con algoritmos que trabajan con una sola etiqueta, usando diversas métricas especificas para conjuntos de datos multietiqueta.

Hemos mostrado que reducir la dimensionalidad mejora el desempeño de aprendizaje. La mayoría de las veces se consigue una mejoría significativa desde el punto de vista estadístico. Los resultados conseguidos por SVM son uniformemente mejores que los de los demás algoritmos. En conjunto, los mejores algoritmos multietiqueta son RAkEL, EPPT y CLR. En conjunto, hemos encontrado que la mejor combinación es usar RAkEL sobre SVM, excepto cuando se optimizar la micro-precisión, caso en el que es preferible usar árboles de decisión. Asimismo, hemos encontrado que tener en cuenta la dependencia entre etiquetas es más importante para el recall que para la precisión.

En el capítulo 5 hemos presentado el concepto de clasificación de flujos de texto, así como los cambios de concepto. Hemos presentado la plataforma GNUsmail para clasificación online de correo electrónico, y la hemos usado para llevar a cabo un estudio sobre el efecto de diferentes tipos de cambio de concepto en diferentes datasets de correo electrónico. A partir de este estudio hemos podido concluir que, aunque el proceso de adaptación mejora la precisión de los modelos de aprendizaje, el cambio de concepto resulta ser virtual y no real, al estar causado por la asimetría estadística de la distribución de los datos a lo largo del tiempo.

En el capítulo 6 nos hemos centrado en la comparación de varias estrategias de selección de atributos y estrategias adaptativas para clasificación de correo electrónico usando el dataset de Enron con una plataforma de clasificación adaptativa que hemos propuesto, ABC-Dynf. Esta plataforma soporta conjuntos cambiantes de atributos, manteniendo una lista de los top-k atributos, que serán usados por el modelo de aprendizaje. ABC-DynF también permite añadir nuevas categorías conforme pasa el tiempo. No hemos encontrado evidencia que nos permita afirmar que el desempeño es significativamente diferente cuando se usa un número diferente de líneas para extraer atributos. Podemos corroborar que el uso de Iterative Bayes mejora de forma significativa el desempeño del Naïve Bayes incremental. Actualizar los atributos de clasificación, para hacer frente a cambio de concepto contextual, también tiene un impacto positivo

en la clasificación de forma significativa. Por lo tanto podemos concluir que monitorizar las categorías más importantes a lo largo del tiempo, incluyéndolas en el modelo de clasificación, tiene una gran importancia a la hora de mejorar el desempeño de los modelos.

En el capítulo 7 nos hemos enfrentado con el proceso de vectorización de documentos que llegan de un flujo de datos, y hemos propuesto funciones de relevancia como mecanismo para resumir documentos. De manera más específica, usamos funciones de pesado para extraer las palabras más relevantes de los documentos, y usamos estas palabras para construir incrementalmente listas de palabras clave y nubes de palabras. Mostramos que los cálculos necesarios presentan un grave problema de complejidad, por lo que proponemos TF-SIDF como solución novedosa. Mostramos que, para un mismo tamaño total del sketch, son preferibles configuraciones que tengan relativamente pocas funciones hash. En lo relativo a los resúmenes basados en nubes de etiquetas, hemos probado que usar un sketch para computar la función de relevancia y otro para computar los top-k atributos de cada etiqueta es la solución más apropiada.

Finalmente, en el capítulo 8 hemos lidiado con el escenario de aprendizaje online donde sólo las palabras más importantes deben ser usadas como atributos, para evitar problemas de complejidad. Es importante tener algoritmos eficientes que acepten un espacio dinámico de atributos. Nuestra contribución principal en este capítulo ha SIDO STFSIDF, una estrategia para calcular un valor aproximado de la función TFIDF usando el algoritmo Min-Count Sketch como herramienta básica para reducir el espacio necesario, posibilitando la gestión de espacios dinámicos de atributos con unos requisitos de tiempo y memoria reducidos. Los resultados experimentales muestran que, en general, no hay una diferencia significativa en el comportamiento del algoritmo cuando se usan funciones exactas o aproximadas, lo que demuestra que nuestra propuesta tiene utilidad práctica en los flujos de datos de texto, al hacer posible reducir la complejidad sin afectar significativamente a los algoritmos de clasificación.

B.3 Trabajo futuro

Hay varias cuestiones que todavía están abiertas, así como diferentes mejoras y extensiones a las propuestas presentadas en esta tesis. Algunas posibilidades para trabajo futuro son las siguientes:

- Una posibilidad de trabajo futuro es extender GNUsmail a otros dominios basados en texto, tales como entradas de blogs o tweets. La arquitectura actual está especializada en correo electrónico, y trae incluidos componentes para análisis y extracción de atributos a partir de emails. Esta arquitectura se podría extender a otras fuentes de datos. Esto conllevaría implementar otros módulos de extracción de datos, así como usar atributos especializados en cada tipo de datos.

- En el futuro planeamos extender los estudios presentados en esta tesis a diferentes algoritmos de estimación de frecuencia que usen espacio sublineal.

- Estamos interesados en estudiar el desempeño de TF-SIDF con otros tipos de funciones de relevancia diferentes a las propuestas.

- Otra línea de trabajo futuro es la adopción de medidas para gestionar la degradación del desempeño de los sketch debida a la saturación. Esta saturación es inherente a los algoritmos basados en sketches, debido a las colisiones producidas por la gran cantidad de elementos con cuentas pequeñas. Esta saturación se podría reiniciando periódicamente los contadores contenidos en los sketches.

- Pretendemos estudiar el problema de clasificación multietiqueta bajo la óptica del aprendizaje online, donde los ejemplos llegan de uno en uno, de forma que no tenemos un dataset fijo al principio del proceso de aprendizaje.

- Respecto a ABC-DynF, planeamos experimentar con funciones de relevancia diferentes a chi-cuadrado, así como con otros clasificadores base.

- Estamos también interesados en el desarrollo de nuevos algoritmos para el escenario de espacios dinámicos de atributos con presencia de cambio de concepto.
- Finalmente, consideramos el uso de otros tipos de atributos para clasificación, tales como las anotaciones semánticas.

Appendix C

Tables

TABLE C.1: (

Measures for different combinations of algorithms and multilabel problem transformation methods (Chapter 4). The simbols ⊕ and ⊖ are used when there is a statistically significant difference.

	Hamming Loss		Examp. Accuracy		Examp. Recall		Micro Precision		Micro Recall	
Method / Alg.	**Orig.**	**Preproc.**	**Orig.**	**Preproc.**	**Orig.**	**Preproc.**	**Orig.**	**Preproc.**	**Orig.**	**Preproc.**
RAkEL / J48	0.07	0.12 ⊖	0.39	0.34 ⊖	0.59	0.76 ⊕	0.45	0.31 ⊖	0.56	0.75 ⊕
RAkEL / IBk	0.07	0.07 ⊖	0.30	0.34 ⊕	0.40	0.47 ⊕	0.45	0.43 ⊖	0.37	0.44 ⊕
RAkEL / N. Bayes	0.10	0.07 ⊕	0.28	0.41 ⊕	0.59	0.57 ⊖	0.33	0.49 ⊕	0.55	0.55
RAkEL / SMO	0.06	0.06 ⊖	0.44	0.46 ⊕	0.58	0.68 ⊕	0.55	0.51 ⊖	0.55	0.65 ⊕
RAkEL / NNge	0.06	0.07 ⊖	0.37	0.39 ⊕	0.50	0.64 ⊕	0.52	0.45 ⊖	0.49	0.62 ⊕
RAkEL / LibSVM	0.05	0.06 ⊖	0.45	0.48 ⊕	0.55	0.65 ⊕	0.61	0.55 ⊖	0.52	0.63 ⊕
PPT / J48	0.07	0.06 ⊕	0.35	0.38 ⊕	0.44	0.46 ⊕	0.49	0.52 ⊕	0.40	0.43 ⊕
PPT / IBk	0.07	0.07 ⊕	0.29	0.33 ⊕	0.35	0.39 ⊕	0.48	0.50 ⊕	0.32	0.35 ⊕
PPT / N. Bayes	0.06	0.06 ⊕	0.36	0.37 ⊕	0.42	0.44 ⊕	0.55	0.56 ⊕	0.37	0.39 ⊕
PPT / SMO	0.05	0.05	0.42	0.43 ⊕	0.50	0.51 ⊕	0.60	0.60	0.46	0.47 ⊕
PPT / NNge	0.06	0.06 ⊕	0.37	0.38	0.47	0.48 ⊕	0.50	0.52 ⊕	0.45	0.46 ⊕
PPT / LibSVM	0.05	0.05	0.42	0.44 ⊕	0.50	0.51 ⊕	0.60	0.60	0.46	0.48 ⊕
CLR / J48	0.05	0.05 ⊕	0.39	0.41 ⊕	0.46	0.48 ⊕	0.68	0.69 ⊕	0.44	0.46 ⊕
CLR / IBk	0.07	0.07 ⊕	0.30	0.34 ⊕	0.40	0.45 ⊕	0.44	0.45 ⊕	0.38	0.42 ⊕
CLR / N. Bayes	0.19	0.12 ⊕	0.22	0.30 ⊕	0.68	0.63 ⊖	0.20	0.29 ⊕	0.64	0.60 ⊖
CLR / SMO	0.06	0.05 ⊕	0.43	0.44 ⊕	0.55	0.57 ⊕	0.58	0.59 ⊕	0.53	0.55 ⊕
CLR / NNge	0.05	0.05 ⊕	0.33	0.37	0.40	0.44	0.63	0.65 ⊕	0.41	0.44
CLR / LibSVM	0.05	0.05 ⊕	0.43	0.44 ⊕	0.52	0.54 ⊕	0.64	0.64	0.49	0.52 ⊕
LP / J48	0.08	0.07 ⊕	0.32	0.34 ⊕	0.43	0.46 ⊕	0.41	0.44 ⊕	0.40	0.43 ⊕
LP / IBk	0.07	0.07 ⊕	0.30	0.33 ⊕	0.39	0.43 ⊕	0.44	0.46 ⊕	0.37	0.40 ⊕
LP / N. Bayes	0.06	0.06	0.33	0.34 ⊕	0.39	0.40 ⊕	0.53	0.52	0.33	0.35 ⊕
LP / SMO	0.06	0.06	0.41	0.41	0.50	0.50 ⊕	0.57	0.56	0.45	0.46 ⊕
LP / NNge	0.06	0.06 ⊕	0.38	0.39	0.47	0.48	0.51	0.52	0.44	0.44
LP / LibSVM	0.06	0.06 ⊕	0.40	0.41 ⊕	0.48	0.50 ⊕	0.56	0.57	0.43	0.45 ⊕
BR / J48	0.05	0.05 ⊕	0.38	0.40 ⊕	0.46	0.48 ⊕	0.61	0.64 ⊕	0.44	0.45 ⊕
BR / IBk	0.07	0.07 ⊕	0.30	0.34 ⊕	0.40	0.45 ⊕	0.44	0.45 ⊕	0.38	0.42 ⊕
BR / N. Bayes	0.19	0.12 ⊕	0.22	0.30 ⊕	0.68	0.62 ⊖	0.20	0.29 ⊕	0.64	0.58 ⊖
BR / SMO	0.06	0.06 ⊕	0.39	0.40 ⊕	0.51	0.52 ⊕	0.54	0.54	0.48	0.49 ⊕
BR / NNge	0.05	0.05	0.33	0.35	0.39	0.41	0.62	0.63	0.39	0.41
BR / LibSVM	0.05	0.05	0.41	0.42 ⊕	0.50	0.51 ⊕	0.63	0.63 ⊖	0.47	0.49 ⊕
EPPT / J48	0.14	0.12 ⊕	0.31	0.33 ⊕	0.77	0.76 ⊖	0.29	0.31 ⊕	0.75	0.74 ⊖
EPPT / IBk	0.09	0.09 ⊕	0.30	0.34 ⊕	0.55	0.57 ⊕	0.36	0.39 ⊕	0.51	0.54 ⊕
EPPT / N. Bayes	0.07	0.06 ⊕	0.37	0.38 ⊕	0.49	0.50	0.48	0.50 ⊕	0.46	0.47 ⊕
EPPT / SMO	0.07	0.07 ⊖	0.42	0.43	0.66	0.68 ⊕	0.46	0.45 ⊖	0.63	0.66 ⊕
EPPT / NNge	0.12	0.10 ⊕	0.31	0.35 ⊕	0.71	0.70	0.32	0.35 ⊕	0.69	0.68
EPPT / LibSVM	0.07	0.07 ⊖	0.42	0.44 ⊕	0.63	0.66 ⊕	0.47	0.47 ⊖	0.61	0.64 ⊕

TABLE C.2: Average ranking of the multilabel methods using the original version of the dataset (Chapter 4)

Method	Alg.	Hamming Loss	Examp. Accuracy	Examp. Recall	Micro Precision	Micro Recall	Av. ranking
RAkEL	LibSVM	0.053 (4)	0.448 (1)	0.552 (11)	0.613 (6)	0.518 (11)	6.6
CLR	LibSVM	0.051 (2)	0.426 (4)	0.518 (13)	0.644 (2)	0.494 (13)	6.8
CLR	SMO	0.055 (6)	0.427 (3)	0.554 (10)	0.583 (10)	0.527 (10)	7.8
RAkEL	SMO	0.058 (11)	0.438 (2)	0.58 (9)	0.551 (14)	0.55 (8)	8.8
BR	LibSVM	0.052 (3)	0.411 (10)	0.495 (19)	0.633 (3)	0.469 (16)	10.2
PPT	LibSVM	0.055 (6)	0.424 (5)	0.5 (17)	0.602 (8)	0.461 (18)	10.8
PPT	SMO	0.055 (6)	0.424 (5)	0.502 (16)	0.6 (9)	0.463 (17)	10.6
CLR	J48	0.05 (1)	0.39 (13)	0.458 (25)	0.677 (1)	0.441 (23)	12.6
EPPT	LibSVM	0.07 (23)	0.424 (5)	0.634 (6)	0.473 (23)	0.61 (6)	12.6
LP	SMO	0.058 (11)	0.412 (9)	0.497 (18)	0.566 (11)	0.455 (20)	13.8
EPPT	SMO	0.072 (28)	0.424 (5)	0.659 (5)	0.459 (24)	0.633 (5)	13.4
BR	SMO	0.061 (15)	0.389 (14)	0.507 (14)	0.537 (15)	0.478 (15)	14.6
BR	J48	0.055 (6)	0.378 (15)	0.461 (24)	0.606 (7)	0.442 (22)	14.8
RAkEL	NNge	0.062 (16)	0.371 (18)	0.504 (15)	0.524 (17)	0.491 (14)	16
LP	LibSVM	0.059 (13)	0.4 (11)	0.477 (21)	0.564 (12)	0.433 (25)	16.4
RAkEL	J48	0.073 (29)	0.392 (12)	0.585 (8)	0.451 (25)	0.557 (7)	16.2
CLR	NNge	0.054 (5)	0.334 (22)	0.402 (29)	0.632 (4)	0.41 (26)	17.2
EPPT	NNge	0.117 (33)	0.307 (26)	0.709 (2)	0.317 (33)	0.685 (2)	19.2
EPPT	J48	0.135 (34)	0.305 (27)	0.772 (1)	0.291 (34)	0.753 (1)	19.4
BR	NNge	0.055 (6)	0.329 (24)	0.39 (33)	0.625 (5)	0.394 (29)	19.4
PPT	NNge	0.065 (19)	0.375 (17)	0.472 (22)	0.501 (19)	0.446 (21)	19.6
LP	NNge	0.064 (18)	0.377 (16)	0.47 (23)	0.509 (18)	0.439 (24)	19.8
EPPT	NB	0.067 (21)	0.366 (19)	0.491 (20)	0.483 (21)	0.459 (19)	20
PPT	NB	0.06 (14)	0.356 (20)	0.418 (28)	0.552 (13)	0.374 (32)	21.4
BR	NB	0.19 (35)	0.218 (35)	0.684 (3)	0.201 (35)	0.644 (3)	22.2
PPT	J48	0.066 (20)	0.352 (21)	0.436 (26)	0.491 (20)	0.402 (27)	22.8
RAkEL	NB	0.101 (32)	0.279 (34)	0.588 (7)	0.332 (32)	0.55 (8)	22.6
CLR	NB	0.192 (36)	0.216 (36)	0.68 (4)	0.199 (36)	0.641 (4)	23.2
EPPT	IBk	0.091 (31)	0.297 (32)	0.546 (12)	0.36 (31)	0.512 (12)	23.6
LP	NB	0.063 (17)	0.334 (22)	0.386 (35)	0.528 (16)	0.335 (35)	25
LP	J48	0.076 (30)	0.322 (25)	0.431 (27)	0.414 (30)	0.401 (28)	28
RAkEL	IBk	0.07 (23)	0.304 (28)	0.397 (32)	0.451 (25)	0.374 (32)	28
CLR	IBk	0.071 (25)	0.299 (30)	0.401 (30)	0.442 (28)	0.384 (30)	28.6
PPT	IBk	0.067 (21)	0.295 (33)	0.353 (36)	0.478 (22)	0.323 (36)	29.6
BR	IBk	0.071 (25)	0.299 (30)	0.401 (30)	0.442 (28)	0.384 (30)	28.6
LP	IBk	0.071 (25)	0.303 (29)	0.389 (34)	0.444 (27)	0.368 (34)	29.8

TABLE C.3: Average ranking of the multilabel methods using the preprocessed dataset (Chapter 4)

Method	Alg.	Hamming Loss	Examp. Accuracy	Examp. Recall	Micro Precision	Micro Recall	Av. ranking
RAkEL	LibSVM	0.057 (10)	0.476 (1)	0.654 (7)	0.551 (13)	0.628 (7)	7.6
CLR	LibSVM	0.05 (2)	0.443 (3)	0.54 (14)	0.641 (4)	0.517 (14)	7.4
CLR	SMO	0.054 (6)	0.443 (3)	0.573 (12)	0.59 (9)	0.549 (12)	8.4
RAkEL	SMO	0.064 (19)	0.46 (2)	0.68 (5)	0.506 (19)	0.654 (5)	10
BR	LibSVM	0.052 (3)	0.424 (9)	0.513 (17)	0.625 (6)	0.487 (16)	10.2
PPT	LibSVM	0.054 (6)	0.436 (5)	0.514 (16)	0.603 (7)	0.475 (17)	10.2
PPT	SMO	0.055 (9)	0.432 (7)	0.51 (18)	0.601 (8)	0.472 (18)	12
CLR	J48	0.049 (1)	0.412 (12)	0.478 (24)	0.687 (1)	0.457 (22)	12
BR	J48	0.052 (3)	0.403 (14)	0.479 (23)	0.643 (3)	0.455 (23)	13.2
EPPT	LibSVM	0.071 (24)	0.435 (6)	0.662 (6)	0.467 (23)	0.638 (6)	13
EPPT	SMO	0.074 (28)	0.429 (8)	0.681 (4)	0.452 (25)	0.655 (4)	13.8
BR	SMO	0.06 (14)	0.402 (15)	0.522 (15)	0.54 (14)	0.494 (15)	14.6
LP	SMO	0.058 (11)	0.415 (10)	0.503 (19)	0.563 (12)	0.46 (21)	14.6
LP	LibSVM	0.058 (11)	0.414 (11)	0.497 (20)	0.568 (10)	0.453 (24)	15.2
RAkEL	NB	0.066 (22)	0.405 (13)	0.575 (11)	0.491 (22)	0.55 (11)	15.8
CLR	NNge	0.052 (3)	0.368 (22)	0.438 (31)	0.647 (2)	0.436 (26)	16.8
RAkEL	NNge	0.074 (28)	0.393 (16)	0.635 (8)	0.45 (28)	0.619 (8)	17.6
EPPT	NNge	0.102 (32)	0.353 (23)	0.702 (3)	0.353 (32)	0.68 (3)	18.6
PPT	NNge	0.063 (16)	0.382 (18)	0.484 (22)	0.517 (18)	0.462 (20)	18.8
LP	NNge	0.062 (15)	0.389 (17)	0.478 (24)	0.523 (16)	0.44 (25)	19.4
EPPT	NB	0.065 (20)	0.381 (19)	0.497 (20)	0.503 (20)	0.467 (19)	19.6
RAkEL	J48	0.124 (35)	0.34 (27)	0.76 (2)	0.31 (33)	0.746 (1)	19.6
BR	NNge	0.054 (6)	0.349 (24)	0.41 (34)	0.631 (5)	0.407 (32)	20.2
EPPT	J48	0.125 (36)	0.331 (33)	0.763 (1)	0.308 (34)	0.742 (2)	21.2
PPT	J48	0.063 (16)	0.376 (20)	0.464 (27)	0.52 (17)	0.427 (28)	21.6
PPT	NB	0.059 (13)	0.375 (21)	0.436 (32)	0.564 (11)	0.395 (34)	22.2
EPPT	IBk	0.085 (31)	0.339 (29)	0.572 (13)	0.388 (31)	0.543 (13)	23.4
CLR	NB	0.12 (34)	0.305 (35)	0.634 (9)	0.292 (35)	0.604 (9)	24.4
BR	NB	0.119 (33)	0.299 (36)	0.616 (10)	0.291 (36)	0.582 (10)	25
LP	NB	0.063 (16)	0.344 (25)	0.398 (35)	0.525 (15)	0.35 (36)	25.4
LP	J48	0.073 (27)	0.344 (25)	0.458 (28)	0.436 (29)	0.425 (29)	27.6
BR	IBk	0.071 (24)	0.337 (30)	0.448 (29)	0.451 (26)	0.417 (30)	27.8
RAkEL	IBk	0.074 (28)	0.34 (27)	0.467 (26)	0.429 (30)	0.436 (26)	27.4
CLR	IBk	0.071 (24)	0.337 (30)	0.448 (29)	0.451 (26)	0.417 (30)	27.8
LP	IBk	0.069 (23)	0.334 (32)	0.426 (33)	0.46 (24)	0.396 (33)	29
PPT	IBk	0.065 (20)	0.328 (34)	0.389 (36)	0.498 (21)	0.352 (35)	29.2

TABLE C.4: F1 results using monitoring in the ABC-DynF framework (Chapter 6)

		Update attributes						Do not update attributes					
		250 at.		300 at.		350 at.		250 at.		300 at.		350 at.	
Dataset	N. lines	IB	Not IB	IB	Not IB	IB	Not IB	IB	Not IB	IB	Not IB	IB	Not IB
beck-s	5	0.642	0.64	0.637	0.635	0.637	0.636	0.643	0.642	0.645	0.644	0.643	0.647
beck-s	10	0.649	0.635	0.64	0.627	0.637	0.626	0.638	0.642	0.643	0.644	0.645	0.646
beck-s	all	0.636	0.635	0.631	0.623	0.628	0.623	0.572	0.598	0.576	0.624	0.578	0.633
farmer-d	5	0.564	0.537	0.559	0.541	0.561	0.542	0.567	0.562	0.567	0.565	0.567	0.565
farmer-d	10	0.565	0.539	0.563	0.545	0.559	0.549	0.557	0.547	0.561	0.552	0.557	0.553
farmer-d	all	0.549	0.514	0.538	0.516	0.545	0.527	0.529	0.52	0.515	0.515	0.528	0.523
kaminski-v	5	0.702	0.697	0.702	0.692	0.692	0.685	0.685	0.678	0.676	0.673	0.676	0.673
kaminski-v	10	0.698	0.689	0.698	0.684	0.695	0.678	0.675	0.67	0.673	0.671	0.662	0.662
kaminski-v	all	0.714	0.714	0.712	0.71	0.701	0.697	0.649	0.619	0.646	0.616	0.641	0.598
kitchen-l	5	0.585	0.533	0.581	0.54	0.576	0.538	0.497	0.45	0.491	0.443	0.495	0.439
kitchen-l	10	0.604	0.573	0.599	0.558	0.603	0.564	0.512	0.508	0.518	0.458	0.524	0.461
kitchen-l	all	0.578	0.478	0.586	0.479	0.58	0.484	0.543	0.485	0.531	0.483	0.525	0.471
lokay-m	5	0.602	0.589	0.6	0.592	0.602	0.595	0.576	0.573	0.574	0.577	0.586	0.582
lokay-m	10	0.615	0.605	0.609	0.603	0.611	0.608	0.58	0.568	0.582	0.572	0.585	0.574
lokay-m	all	0.602	0.594	0.608	0.597	0.606	0.603	0.563	0.544	0.571	0.569	0.582	0.573
sanders-r	5	0.69	0.672	0.678	0.664	0.689	0.68	0.631	0.641	0.631	0.642	0.642	0.652
sanders-r	10	0.703	0.692	0.703	0.677	0.692	0.663	0.664	0.647	0.664	0.661	0.658	0.653
sanders-r	all	0.755	0.766	0.754	0.76	0.756	0.755	0.655	0.639	0.686	0.665	0.636	0.639
williams-w3	5	0.907	0.917	0.911	0.921	0.915	0.923	0.909	0.9	0.9	0.9	0.906	0.905
williams-w3	10	0.919	0.922	0.923	0.927	0.925	0.932	0.909	0.905	0.909	0.902	0.917	0.909
williams-w3	all	0.903	0.9	0.919	0.906	0.921	0.909	0.9	0.887	0.888	0.887	0.888	0.887
rogers-b	5	0.634	0.631	0.635	0.631	0.638	0.632	0.647	0.632	0.64	0.627	0.638	0.626
rogers-b	10	0.636	0.631	0.633	0.633	0.63	0.629	0.643	0.638	0.631	0.63	0.631	0.627
rogers-b	all	0.586	0.482	0.588	0.523	0.591	0.5	0.643	0.609	0.636	0.606	0.626	0.592
germany-c	5	0.739	0.732	0.736	0.728	0.738	0.736	0.705	0.702	0.692	0.679	0.692	0.679
germany-c	10	0.741	0.73	0.737	0.733	0.736	0.732	0.698	0.704	0.697	0.707	0.691	0.672
germany-c	all	0.705	0.734	0.721	0.694	0.713	0.703	0.702	0.696	0.706	0.696	0.703	0.693
shapiro-r	5	0.602	0.613	0.592	0.601	0.588	0.591	0.559	0.556	0.56	0.563	0.557	0.564
shapiro-r	10	0.611	0.613	0.608	0.602	0.598	0.597	0.551	0.557	0.554	0.556	0.539	0.545
shapiro-r	all	0.614	0.617	0.624	0.614	0.616	0.609	0.519	0.515	0.524	0.536	0.526	0.541

TABLE C.5: F1 results without using monitoring in the ABC-DynF framework (Chapter 6)

		Update attributes						Do not update attributes					
		250 at.		300 at.		350 at.		250 at.		300 at.		350 at.	
Dataset	N. lines	IB	Not IB	IB	Not IB	IB	Not IB	IB	Not IB	IB	Not IB	IB	Not IB
beck-s	5	0.618	0.612	0.614	0.61	0.617	0.614	0.609	0.617	0.614	0.62	0.619	0.624
beck-s	10	0.624	0.61	0.621	0.606	0.615	0.604	0.611	0.622	0.615	0.623	0.623	0.629
beck-s	all	0.607	0.573	0.596	0.569	0.596	0.568	0.559	0.555	0.555	0.566	0.567	0.577
farmer-d	5	0.556	0.525	0.551	0.528	0.558	0.532	0.555	0.54	0.555	0.539	0.555	0.539
farmer-d	10	0.551	0.528	0.553	0.534	0.552	0.539	0.552	0.542	0.554	0.546	0.552	0.544
farmer-d	all	0.541	0.509	0.539	0.51	0.538	0.514	0.519	0.506	0.506	0.506	0.523	0.512
kaminski-v	5	0.631	0.61	0.63	0.608	0.62	0.605	0.632	0.601	0.622	0.597	0.622	0.597
kaminski-v	10	0.654	0.624	0.65	0.616	0.644	0.609	0.638	0.612	0.632	0.61	0.624	0.605
kaminski-v	all	0.614	0.612	0.638	0.637	0.629	0.623	0.623	0.559	0.617	0.552	0.606	0.536
kitchen-l	5	0.575	0.533	0.581	0.54	0.576	0.538	0.497	0.45	0.491	0.443	0.495	0.437
kitchen-l	10	0.604	0.573	0.599	0.558	0.603	0.564	0.512	0.454	0.518	0.458	0.524	0.461
kitchen-l	all	0.525	0.456	0.545	0.481	0.542	0.485	0.543	0.485	0.531	0.483	0.525	0.471
lokay-m	5	0.603	0.589	0.6	0.592	0.603	0.595	0.57	0.573	0.579	0.576	0.586	0.582
lokay-m	10	0.615	0.605	0.609	0.603	0.612	0.608	0.581	0.568	0.581	0.572	0.585	0.574
lokay-m	all	0.607	0.594	0.608	0.597	0.606	0.603	0.563	0.544	0.571	0.569	0.582	0.573
sanders-r	5	0.601	0.539	0.61	0.583	0.622	0.59	0.59	0.584	0.591	0.589	0.597	0.59
sanders-r	10	0.619	0.564	0.648	0.593	0.645	0.607	0.603	0.579	0.611	0.592	0.601	0.593
sanders-r	all	0.743	0.717	0.75	0.718	0.747	0.716	0.637	0.57	0.651	0.596	0.617	0.598
williams-w3	5	0.907	0.917	0.911	0.921	0.915	0.923	0.909	0.9	0.9	0.9	0.906	0.905
williams-w3	10	0.919	0.922	0.923	0.927	0.925	0.932	0.909	0.905	0.913	0.902	0.917	0.909
williams-w3	all	0.902	0.9	0.911	0.905	0.917	0.907	0.9	0.887	0.888	0.887	0.888	0.888
rogers-b	5	0.634	0.631	0.634	0.631	0.638	0.633	0.647	0.632	0.64	0.627	0.64	0.627
rogers-b	10	0.636	0.631	0.633	0.633	0.63	0.629	0.643	0.638	0.631	0.63	0.631	0.627
rogers-b	all	0.583	0.491	0.587	0.501	0.589	0.515	0.643	0.609	0.636	0.606	0.626	0.592
germany-c	5	0.739	0.736	0.737	0.736	0.742	0.742	0.704	0.701	0.696	0.675	0.696	0.675
germany-c	10	0.739	0.732	0.734	0.735	0.738	0.735	0.692	0.681	0.691	0.683	0.688	0.655
germany-c	all	0.726	0.733	0.723	0.725	0.721	0.719	0.698	0.689	0.696	0.691	0.7	0.69
shapiro-r	5	0.603	0.601	0.598	0.598	0.598	0.59	0.564	0.548	0.561	0.553	0.563	0.559
shapiro-r	10	0.615	0.612	0.612	0.598	0.603	0.596	0.57	0.56	0.576	0.56	0.545	0.545
shapiro-r	all	0.623	0.626	0.624	0.62	0.617	0.607	0.535	0.533	0.549	0.549	0.544	0.546

TABLE C.6: Error results using monitoring in the ABC-DynF framework (Chapter 6)

		Update attributes						Do not update attributes					
		250 at.		300 at.		350 at.		250 at.		300 at.		350 at.	
Dataset	**N. lines**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**
beck-s	5	49.8	49.86	50.22	50.9	50.6	50.88	51.36	51.48	51.0	51.22	51.24	50.86
beck-s	10	49.32	50.12	50.14	50.86	50.4	51.36	52.04	51.92	51.3	51.44	50.98	51.08
beck-s	all	50.9	50.56	51.64	52.18	52.04	52.52	60.5	58.04	60.0	54.34	59.8	53.0
farmer-d	5	52.64	55.12	52.64	54.4667	52.2667	54.3867	52.96	54.0133	53.2533	53.32	53.2533	53.32
farmer-d	10	52.7733	55.4133	52.4533	54.52	52.68	54.0933	53.9733	55.08	53.36	54.16	53.6267	54.0667
farmer-d	all	54.0667	58.4133	55.0	57.8	54.2	56.7067	57.5733	57.8933	58.2533	58.0533	57.0933	57.5067
kaminski-v	5	43.4746	43.7373	43.2542	44.2119	43.5678	45.3136	45.5169	46.5593	46.1271	46.8644	46.1271	46.8644
kaminski-v	10	43.1864	44.6441	43.661	45.3051	43.839	46.0593	47.7627	48.6949	47.7881	48.2458	48.3898	49.0678
kaminski-v	all	41.5169	41.2627	42.1271	41.9661	42.7542	42.7458	48.3729	51.7881	48.5169	51.7797	48.5763	53.1017
kitchen-l	5	62.5	68.6667	63.8333	67.9524	64.5714	68.2619	72.8571	77.619	72.7619	77.4762	72.3095	76.9286
kitchen-l	10	62.0238	65.381	61.9524	65.4524	61.7619	65.3095	71.3333	69.6905	70.5	76.5714	70.5238	75.5952
kitchen-l	all	66.7143	76.0	66.6667	76.1667	67.5476	75.7143	72.7619	80.3333	72.6667	79.7143	71.9762	78.5
lokay-m	5	50.0533	51.68	50.08	51.2267	49.76	50.8533	52.6667	54.3733	52.8533	54.0533	52.0	53.4933
lokay-m	10	48.1867	49.5733	48.9067	49.8933	48.5333	49.2267	51.92	54.48	51.3333	53.9733	51.4133	53.84
lokay-m	all	49.52	50.24	49.2	50.2667	49.36	49.8667	52.5867	57.0133	51.7333	53.8933	50.9867	52.8
sanders-r	5	45.6	47.3714	46.5143	48.6286	45.8857	46.8571	53.5429	53.2571	54.1714	52.9143	53.3714	51.8857
sanders-r	10	42.9714	43.7714	43.9429	46.2286	45.1429	47.7143	51.0286	54.1143	51.3714	51.1429	50.9143	52.8571
sanders-r	all	40.6857	40.6286	40.8571	40.5714	40.1714	39.7714	50.4	51.8286	47.4857	49.8286	52.1714	51.6
williams-w3	5	14.4815	13.037	14.2222	12.8519	13.8519	12.7407	14.3333	15.6667	15.3333	15.7407	14.8148	15.1481
williams-w3	10	13.1481	12.8519	13.3333	12.2963	13.0	12.1481	14.8519	15.6667	15.2963	16.1481	14.7778	15.5926
williams-w3	all	16.1111	16.2963	13.7778	15.2963	13.1852	13.7037	16.5926	18.2593	17.8148	17.8519	17.7407	17.8148
rogers-b	5	43.785	44.923	43.969	45.2	43.754	45.015	44.554	46.031	44.8	46.462	45.262	46.646
rogers-b	10	43.446	44.0	43.6	43.938	43.846	44.185	44.769	45.754	45.538	45.938	44.8	45.692
rogers-b	all	49.2	60.4	49.015	57.169	48.923	58.492	45.569	49.446	46.492	50.092	46.431	50.338
germany-c	5	38.545	39.606	38.727	39.606	38.606	39.212	40.818	41.545	42.091	43.848	42.091	43.848
germany-c	10	38.545	39.061	39.061	39.0	39.152	38.788	41.727	41.758	42.0	41.576	42.727	44.939
germany-c	all	42.667	40.667	40.879	43.939	41.758	43.152	43.697	43.455	43.667	43.939	43.212	43.697
shapiro-r	5	59.379	59.345	60.31	59.448	58.565	59.655	63.897	63.759	63.379	63.069	63.759	62.552
shapiro-r	10	59.897	59.586	59.621	59.483	60.724	59.552	64.828	63.414	64.517	63.0	65.414	64.0
shapiro-r	all	60.172	59.069	61.034	59.724	61.034	60.207	69.138	69.483	68.276	67.138	68.241	66.379

TABLE C.7: Error results without using monitoring in the ABC-DynF framework (Chapter 6)

		Update attributes						Do not update attributes					
		250 at.		300 at.		350 at.		250 at.		300 at.		350 at.	
Dataset	**N. lines**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**	**IB**	**Not IB**
beck-s	5	52.58	53.1	53.18	53.68	53.14	53.28	55.72	54.64	55.0	54.18	54.42	53.8
beck-s	10	52.18	53.14	53.0	53.66	53.24	54.16	55.68	54.46	55.3	54.08	54.3	53.4
beck-s	all	53.9	56.92	55.06	57.5	55.42	57.84	61.92	63.2	61.9	61.54	61.28	60.12
farmer-d	5	53.88	56.773	53.973	56.013	53.307	55.773	54.613	56.133	54.853	56.173	54.853	56.173
farmer-d	10	54.333	57.013	53.827	55.893	53.76	55.347	54.6	56.013	54.267	54.973	54.44	55.213
farmer-d	all	54.987	59.213	55.253	58.72	55.04	58.253	59.0	59.533	59.493	59.32	58.053	58.773
kaminski-v	5	51.5	53.653	51.78	54.068	52.441	54.542	52.424	55.729	52.941	56.119	52.941	56.119
kaminski-v	10	49.195	53.169	49.644	53.949	50.153	54.881	52.763	56.22	53.195	56.195	53.847	56.373
kaminski-v	all	51.686	51.695	49.5	49.729	50.339	50.568	52.203	59.161	52.593	59.678	53.322	61.169
kitchen-l	5	64.714	68.667	63.905	67.952	64.571	68.262	72.857	77.619	72.762	77.476	72.31	77.19
kitchen-l	10	62.024	66.214	61.952	66.429	61.762	65.31	71.333	77.19	70.5	76.571	70.524	75.595
kitchen-l	all	74.381	79.571	74.238	78.976	72.643	77.929	72.762	80.333	72.667	79.714	71.929	78.5
lokay-m	5	50.267	51.68	50.16	51.227	49.707	50.853	53.76	54.373	52.48	54.053	52.0	53.493
lokay-m	10	48.187	49.573	48.907	49.893	48.4	49.227	51.947	54.48	52.053	53.973	51.413	53.84
lokay-m	all	49.387	50.24	49.2	50.267	49.36	49.867	52.587	57.013	51.733	53.893	50.987	52.8
sanders-r	5	55.657	61.429	55.543	58.057	54.457	56.857	59.943	61.143	60.171	60.286	60.229	60.171
sanders-r	10	52.343	57.371	50.686	56.229	51.086	54.571	60.057	61.943	59.314	60.914	59.371	60.4
sanders-r	all	42.229	45.086	42.286	45.086	42.057	44.343	53.714	60.743	52.4	58.971	54.857	57.771
williams-w3	5	14.481	13.037	14.222	12.852	13.852	12.741	14.333	15.667	15.333	15.741	14.815	15.148
williams-w3	10	13.148	12.852	13.333	12.296	13.0	12.148	14.852	15.667	14.815	16.148	14.778	15.593
williams-w3	all	16.37	16.296	15.259	15.556	13.852	14.074	16.593	18.259	17.815	17.852	17.741	17.741
rogers-b	5	43.785	44.923	43.969	45.2	43.754	44.923	44.554	46.031	44.8	46.462	45.015	46.431
rogers-b	10	43.477	44.0	43.6	43.938	43.846	44.185	44.831	45.754	45.538	45.938	44.8	45.692
rogers-b	all	49.446	61.262	49.292	59.908	49.323	58.431	45.569	49.446	46.492	50.092	46.431	50.338
germany-c	5	39.152	39.697	38.818	39.576	39.0	39.455	42.636	43.424	43.212	45.455	43.212	45.455
germany-c	10	39.182	39.515	39.576	39.515	39.576	39.424	43.091	43.576	43.182	43.424	43.273	46.394
germany-c	all	42.0	41.303	41.727	41.97	42.061	42.212	44.424	45.303	44.364	45.03	43.939	45.455
shapiro-r	5	61.207	61.448	61.793	61.379	62.172	62.069	64.586	65.345	64.241	65.0	64.0	64.31
shapiro-r	10	60.966	61.241	60.69	61.552	61.483	61.724	65.138	64.724	64.966	64.483	66.069	65.69
shapiro-r	all	61.69	60.759	61.034	61.241	61.31	61.966	68.931	70.828	68.586	68.621	68.586	68.069

Bibliography

[1] Kwok-Wai Cheung, James T. Kwok, Martin H. Law, and Kwok-Ching Tsui. Mining customer product ratings for personalized marketing. *Decis. Support Syst.*, 35(2): 231–243, May 2003. ISSN 0167-9236. doi: 10.1016/S0167-9236(02)00108-2. URL `http://dx.doi.org/10.1016/S0167-9236(02)00108-2`.

[2] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, January 2008. ISSN 1554-0669. doi: 10.1561/1500000011. URL `http://dx.doi.org/10.1561/1500000011`.

[3] Carlos Castillo, Debora Donato, Aristides Gionis, Vanessa Murdock, and Fabrizio Silvestri. Know your neighbors: web spam detection using the web topology. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 423–430, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7. doi: 10.1145/1277741.1277814. URL `http://doi.acm.org/10.1145/1277741.1277814`.

[4] Don R. Swanson and Neil R. Smalheiser. An interactive system for finding complementary literatures: A stimulus to scientific discovery. *Artif. Intell.*, 91(2):183–203, 1997.

[5] William E. Spangler, Mordechai Gal-Or, and Jerrold H. May. Using data mining to profile tv viewers. *Commun. ACM*, 46(12):66–72, December 2003. ISSN 0001-0782. doi: 10.

1145/953460.953461. URL http://doi.acm.org/10.1145/953460.953461.

[6] Philip Russorn. Bi search and analytics: New additions to the bi technology stack, 2007. URL http://download.101com.com/pub/tdwi/Files/TDWI_RRQ207_lo.pdf.

[7] Bhavani M. Thuraisingham. Data mining, national security, privacy and civil liberties. *SIGKDD Explorations*, 4(2):1–5, 2002.

[8] V. Gadia and G. Rosen. A text-mining approach for classification of genomic fragments. In *IEEE International Workshop on Biomedical and Health Informatics*, 2008.

[9] Alan M. Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, October 1950. ISSN 00264423. doi: 10.2307/2251299. URL http://dx.doi.org/10.2307/2251299.

[10] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):147, 2002. ISSN 0360-0300.

[11] G. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Trans. Inf. Theor.*, 14(1):55–63, September 2006. ISSN 0018-9448. doi: 10.1109/TIT.1968.1054102. URL http://dx.doi.org/10.1109/TIT.1968.1054102.

[12] Chichang Jou and Yung-Yu Shih. A spam email classification based on the incremental forgetting bayesian algorithm. In *2012 International Conference on Business and Information (BAI 2012)*, pages 19–27, 2012.

[13] Odysseas Papapetrou, George Papadakis, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient term cloud generation for streaming web content. In *Proceedings of the 10th international conference on Web engineering*, ICWE'10, pages 385–399, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13910-8, 978-3-642-13910-9. URL http://portal.acm.org/citation.cfm?id=1884110.1884142.

[14] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of data mining*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0-262-08290-X.

[15] Igor Kononenko and Matjaz Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited, West Sussex, 2008. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/1904275214`.

[16] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. CRISP-DM 1.0 Step-by-step data mining guide. Technical report, The CRISP-DM consortium, August 2000. URL `http://www.crisp-dm.org/CRISPWP-0800.pdf`.

[17] T. M. Mitchell. *Machine learning*. McGraw Hill, New York, 1997.

[18] W. John Hutchins. The georgetown-ibm experiment demonstrated in january 1954. In Robert E. Frederking and Kathryn Taylor, editors, *AMTA*, volume 3265 of *Lecture Notes in Computer Science*, pages 102–114. Springer, 2004. ISBN 3-540-23300-8.

[19] W.J. Hutchins. Alpac: the (in)famous report. *MT News International*, 14:9–12, 1996.

[20] *Proceedings of the 6th Conference on Message Understanding, MUC 1997*, 1997.

[21] Thierry Poibeau and Leila Kosseim. Proper name extraction from non-journalistic texts. In *In Computational Linguistics in the Netherlands*, pages 144–157, 2001.

[22] Steven J. DeRose. Grammatical category disambiguation by statistical optimization. *Comput. Linguist.*, 14(1):31–39, January 1988. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=49084.49087`.

[23] José Luis Triviño-Rodríguez and Rafael Morales Bueno. Using multiattribute prediction suffix graphs for spanish part-of-speech tagging. In Frank Hoffmann, David J. Hand, Niall M. Adams, Douglas H. Fisher, and Gabriela Guimarães, editors, *IDA*, volume 2189 of *Lecture Notes in Computer Science*, pages 228–237. Springer, 2001. ISBN 3-540-42581-0.

[24] William J. Rapaport. A history of the sentence 'buffalo buffalo buffalo buffalo buffalo', 2000. URL `http://web.archive.org/web/20070320205923/http://www.cse.buffalo.edu/~rapaport/buffalobuffalo.html`.

[25] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. *CoRR*, cs.CL/0205070, 2002.

[26] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=944919.944937`.

[27] M. E. Maron. Automatic indexing: An experimental inquiry. *J. ACM*, 8(3):404–417, 1961.

[28] Philip J. Hayes and Steven P. Weinstein. Construe/TIS: A system for content-based indexing of a database of news stories. In *2nd Annual Conference on Innovative Applications of Artificial Intelligence.*, pages 1–5, 1990.

[29] Norbert Fuhr, Stephan Hartmann, Gerhard Lustig, Michael Schwantner, Konstadinos Tzeras, Technische Hochschule Darmstadt, Fachbereich Informatik, and Gerhard Knorz. Air/x - a rule-based multistage indexing system for large subject fields. In *Proceedings of RIAO'91*, pages 606–623, 1991.

[30] Javier Sánchez-Monedero, Manuel Cruz-Ramírez, Francisco Fernández-Navarro, Juan Carlos Fernández, Pedro Antonio Gutiérrez, and César Hervás-Martínez. On the suitability of extreme learning machine for gene classification using feature selection. In *ISDA*, pages 507–512. IEEE, 2010.

[31] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.

[32] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning (ECML-2004)*, 2004.

[33] George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, 2003. ISSN 1533-7928. URL `http://portal.acm.org/citation.cfm?id=944919.944974`.

[34] Francisco Herrera and José-Luis Verdegay. *Genetic Algorithms and Soft Computing*. Physica-Verlag, 1996.

[35] Mehdi Hosseinzadeh Aghdam, Nasser Ghasem-Aghaee, and Mohammad Ehsan Basiri. Text feature selection using ant colony optimization. *Expert Syst. Appl.*, 36(3):6843–6853, April 2009. ISSN 0957-4174. doi: 10.1016/j.eswa.2008.08.022. URL `http://dx.doi.org/10.1016/j.eswa.2008.08.022`.

[36] Alex Alves Freitas. A review of evolutionary algorithms for data mining. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 371–400. Springer, 2010. ISBN 978-0-387-09822-7.

[37] David D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 4–15, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-64417-2. URL `http://dl.acm.org/citation.cfm?id=645326.649711`.

[38] Jason D. M. Rennie. ifile: An application of machine learning to e-mail filtering. In *Proc. KDD Workshop on Text Mining*, 2000.

[39] Joseph M. Hilbe. *Logistc Regression Models*. Chapman & Hall/CRC Press, 2009. ISBN 1420-075756.

[40] Hinrich Schütze, David A. Hull, and Jan O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '95, pages 229–237, New York, NY, USA, 1995. ACM. ISBN 0-89791-714-6. doi: 10.1145/215206.215365. URL `http://doi.acm.org/10.1145/215206.215365`.

[41] I. Dagan, Y. Karov, and D. Roth. Mistake-Driven Learning in Text Categorization. In *EMNLP-97, The Second Conference on Empirical Methods in Natural Language Processing*, pages 55–63, August 1997.

[42] Robert H. Creecy, Brij M. Masand, Stephen J. Smith, and David L. Waltz. Trading mips and memory for knowledge engineering. *Commun. ACM*, 35(8):48–64, August 1992. ISSN 0001-0782. doi: 10.1145/135226.135228. URL `http://doi.acm.org/10.1145/135226.135228`.

[43] Yiming Yang and Christopher G. Chute. An example-based mapping method for text categorization and retrieval. *ACM Trans. Inf. Syst.*, 12:252–277, July 1994. ISSN 1046-8188. URL `http://dx.doi.org/10.1145/183422.183424`.

[44] Brent Martin. Instance-based learning: Nearest neighbour with generalization. Master's thesis, University of Waikato, 1995.

[45] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. ISSN 0885-6125. URL `http://dx.doi.org/10.1007/BF00116251`.

[46] William W. Cohen. Learning Rules that Classify E-Mail. In *In Papers from the AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25, 1996. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.4129`.

[47] Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship

with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.

[48] Ray Liere and Prasad Tadepalli. Active learning with committees for text categorization. In *In proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 591–596, 1997.

[49] R. E. Schapire and Y. Singer. BoosTexter: a boosting-based system for text categorization. *Machine Learning*, 39:135–168, 2000. ISSN 0885-6125. URL `http://dx.doi.org/10.1023/A:1007649029923`.

[50] Gonzalo Ramos-Jiménez, José del Campo-Ávila, and Rafael Morales Bueno. Fe-cidim: fast ensemble of cidim classifiers. *Int. J. Systems Science*, 37(13):939–947, 2006.

[51] José del Campo-Ávila, Gonzalo Ramos-Jiménez, Jesús Pérez-García, and Rafael Morales Bueno. Studying the hybridization of artificial neural networks in hecic. In Joan Cabestany, Ignacio Rojas, and Gonzalo Joya Caparrós, editors, *IWANN (2)*, volume 6692 of *Lecture Notes in Computer Science*, pages 137–144. Springer, 2011. ISBN 978-3-642-21497-4.

[52] José del Campo-Ávila, Gonzalo Ramos-Jiménez, and Rafael Morales Bueno. Incremental learning with multiple classifier systems using correction filters for classification. In Michael R. Berthold, John Shawe-Taylor, and Nada Lavrac, editors, *IDA*, volume 4723 of *Lecture Notes in Computer Science*, pages 106–117. Springer, 2007. ISBN 978-3-540-74824-3.

[53] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1:69–90, May 1999. ISSN 1386-4564. doi: 10.1023/A:1009982220290. URL `http://portal.acm.org/citation.cfm?id=357367.357383`.

[54] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945. ISSN 00994987. doi: 10.2307/3001968. URL `http://dx.doi.org/10.2307/3001968`.

[55] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[56] William Hersh, Chris Buckley, T. J. Leone, and David Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 192–201, New York, NY, USA, 1994. Springer-Verlag New York, Inc. ISBN 0-387-19889-X. URL `http://dl.acm.org/citation.cfm?id=188490.188557`.

[57] Ken Lang. Newsweeder: Learning to filter netnews. In *in Proceedings of the 12th International Machine Learning Conference (ML95)*, 1995.

[58] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, December 2004. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1005332.1005345`.

[59] Keh-Jiann Chen and Shing-Huan Liu. Word identification for mandarin chinese sentences. In *COLING*, pages 101–107, 1992.

[60] Joon Ho Lee and Hyun Jung Lee. Combing different statistical evidence for chinese word segmentation.

[61] M. Baena-García and R. Morales Bueno. New data structures for analyzing frequent factors in strings. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 1294 –1299, nov. 2011.

[62] Manuel Baena-García and Rafael Morales Bueno. Mining interestingness measures for string pattern mining. *Knowl.-Based Syst.*, 25(1):45–50, 2012.

[63] Xin Chen, Ming Li, Bin Ma, and John Tromp. Dnacompress: fast and effective dna sequence compression. *Bioinformatics*, 18(12):1696–1698, 2002. URL `http:`

//dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics18.html#ChenLMT02.

[64] David W. Mount. Using the Basic Local Alignment Search Tool (BLAST). *Cold Spring Harbor Protocols*, 2007(7): pdb.top17+, July 2007. doi: 10.1101/pdb.top17. URL http://dx.doi.org/10.1101/pdb.top17.

[65] K. Rutherford, J. Parkhill, J. Crook, T. Horsnell, P. Rice, M. A. Rajandream, and B. Barrell. Artemis: sequence visualization and annotation. *Bioinformatics*, 16(10):944–945, Oct 2000.

[66] Ramu Chenna, Hideaki Sugawara, Tadashi Koike, Rodrigo Lopez, Toby J. Gibson, Desmond G. Higgins, and Julie D. Thompson. Multiple sequence alignment with the clustal series of programs. *Nucleic Acids Res*, 31:3497–3500, 2003.

[67] Carol Wong, Yuran Li, Chih Lee, and Chun-Hsi Huang. Ensemble learning algorithms for classification of mtdna into haplogroups. *Briefings in Bioinformatics*, 12(1):1–9, 2011.

[68] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171 (4356):737–738, April 1953. ISSN 0028-0836. doi: 10.1038/171737a0. URL http://dx.doi.org/10.1038/171737a0.

[69] Spencer Wells. *Deep ancestry: Inside the genographic project.* National Geographic, 2006.

[70] Anita Kloss-Brandsttter, Dominic Pacher, Sebastian Schnherr, Hansi Weissensteiner, Robert Binna, Gnther Specht, and Florian Kronenberg. Haplogrep: a fast and reliable algorithm for automatic classification of mitochondrial dna haplogroups. *Human Mutation*, 32(1):25–32, 2011. ISSN 1098-1004. doi: 10.1002/humu.21382. URL http://dx.doi.org/10.1002/humu.21382.

[71] Aaron M. Cohen and William R. Hersh. A survey of current work in biomedical text mining. *Briefings in Bioinformatics*, 6(1):57–71, 2005.

[72] Alberto Faro, Daniela Giordano, and Concetto Spampinato. Combining literature text mining with microarray data: advances for system biology modeling. *Briefings in Bioinformatics*, 13(1):61–82, 2012.

[73] P. Weiner. Linear pattern matching algorithm. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[74] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[75] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *LATIN*, pages 374–390, 1998.

[76] Essam Mansour, Amin Allam, Spiros Skiadopoulos, and Panos Kalnis. Era: Efficient serial and parallel suffix tree construction for very long strings. *PVLDB*, 5(1):49–60, 2011.

[77] Pal Saetrom. *Hardware accelerated genetic programming for pattern mining in strings*. Dr.philos thesis, Faculty of Information Technology, Mathematics and Electrical Engineering Department of Computer and Information Science, Norwegian University of Science and Technology, NTNU, Norway, 2005.

[78] Steven De Rooij. Methods of statistical data compression. Master's thesis, Institute for Logic, Language and Computation, University of Amsterdam, 2003.

[79] Breanndán Ó Nualláin and Steven de Rooij. Online suffix trees with counts. *Data Compression Conference*, 0:555, 2004. ISSN 1068-0314.

[80] Edward Fredkin. Trie memory. *Commun. ACM*, 3(9): 490–499, September 1960. ISSN 0001-0782. doi: 10.1145/367390.367400. URL `http://doi.acm.org/10.1145/367390.367400`.

[81] Ferenc Bodon. A survey on frequent itemset mining. Technical report, Budapest University of Technology and Economics, 2006.

[82] Jaak Vilo. Discovering frequent patterns from strings. Technical report, Department of Computer Science, University of Helsinki, Finland, 1998.

[83] Neng wen Lo, Hsuan T. Chang, S. W. Xiao, C. H. Li, Chung, and J. Kuo. Visualization and comparison of dna sequences by use of three-dimensional trajectories. In *In First Asia-Pacific Bioinformatics Conference, (APBC)*, pages 81–85. Australian Computer Society, Inc, 2003.

[84] Joan Hérisson, Pierre-Emmanuel Gros, Nicolas Férey, Olivier Magneau, and Rachid Gherbi. Dna *in virtuo* visualization and exploration of 3d genomic structures. In Lynette van Zijl and Patrick Marais, editors, *Afrigraph*, pages 35–40. ACM, 2004. ISBN 1-58113-863-6.

[85] John A. Berger, Sanjit K. Mitra, Marco Carli, and Alessandro Neri. Visualization and analysis of DNA sequences using DNA walks. *J. Franklin Inst.*, 341(1-2):37–53, 2004. doi: 10.1016/j.jfranklin.2003.12.002.

[86] Scott Schwartz, Laura Elnitski, Mei Li, Matthew Weirauch, Cathy Riemer, Arian Smit, Eric D. Green, Ross C. Hardison, and Webb Miller. Multipipmaker and supporting tools: alignments and analysis of multiple genomic dna sequences. *Nucleic Acids Research*, 31(13):3518–3524, 2003.

[87] Manuel Baena-García and Rafael Morales Bueno. New data structures for analyzing frequent factors in strings. In Sebastián Ventura, Ajith Abraham, Krzysztof J. Cios, Cristóbal Romero, Francesco Marcelloni, José Manuel Benítez, and Eva Lucrecia Gibaja Galindo, editors, *ISDA*, pages 900–905. IEEE, 2011. ISBN 978-1-4577-1676-8.

[88] G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 667–685. Springer US, 2010. ISBN 978-0-387-09823-4. URL `http://dx.doi.org/10.1007/978-0-387-09823-4_34`.

[89] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. A machine learning approach to building domain-specific search engines. In *IJCAI*, pages 662–667, 1999.

[90] N. Ueda and K. Saito. Parametric mixture models for multi-labeled text. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 721–728. MIT Press, Cambridge, MA, 2003.

[91] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7:1601–1626, 2006. ISSN 1532-4435. URL `http://portal.acm.org/citation.cfm?id=1248547.1248606`.

[92] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004. ISSN 0031-3203. doi: DOI:10.1016/j.patcog.2004.03.009. URL `http://www.sciencedirect.com/science/article/B6V14-4CF14JX-1/2/a17089f241a1d23f218e55d2c8d9f763`.

[93] G. J. Qi, X. S. Hua, Y. Rui, J. Tang, T. Mei, and H. J. Zhang. Correlative multi-label video annotation. In *Proceedings of the 15th international conference on Multimedia*, pages 17–26, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-702-5. doi: http://doi.acm.org/10.1145/1291233.1291245. URL `http://doi.acm.org/10.1145/1291233.1291245`.

[94] Z. Barutcuoglu, R. E. Schapire, and O. G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006. URL `http://view.ncbi.nlm.nih.gov/pubmed/16410319`.

[95] T. Li and M. Ogihara. Toward intelligent music information retrieval. *IEEE Transactions on Multimedia*, 8(3):564–574, 2006.

[96] J. Read, B. Pfahringer, and G. Holmes. Multi-label classification using ensembles of pruned sets. In *Proceedings of the 2008*

Eighth IEEE International Conference on Data Mining, pages 995–1000, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3502-9. doi: 10.1109/ICDM.2008.74. URL `http://portal.acm.org/citation.cfm?id=1510528.1511358`.

[97] G. Tsoumakas and I. Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Proceedings of the 18th European conference on Machine Learning*, ECML '07, pages 406–417, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74957-8. doi: http://dx.doi.org/10.1007/978-3-540-74958-5_38. URL `http://dx.doi.org/10.1007/978-3-540-74958-5_38`.

[98] Andrew McCallum. Multi-label text classification with a mixture model trained by em. In *AAAI 99 Workshop on Text Learning*, 1999.

[99] W. Daelemans, B. Goethals, K. Morik, A. Streich, and J. Buhmann. Classification of multi-labeled data: A generative approach. In *Machine Learning and Knowledge Discovery in Databases*, volume 5212 of *Lecture Notes in Computer Science*, pages 390–405. Springer Berlin / Heidelberg, 2008. URL `http://dx.doi.org/10.1007/978-3-540-87481-2_26`.

[100] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Trans. on Knowl. and Data Eng.*, 18:1338–1351, October 2006. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2006.162. URL `http://dx.doi.org/10.1109/TKDE.2006.162`.

[101] Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. *J. Mach. Learn. Res.*, 3:1025–1058, March 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=944919.944962`.

[102] Fadi A. Thabtah, Peter Cowling, and Yonghong Peng. Mmac: A new multi-class, multi-label associative classification approach. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, ICDM '04, pages 217–224, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2142-8. URL `http://dl.acm.org/citation.cfm?id=1032649.1033457`.

[103] Adriano Veloso, Wagner Meira, Jr., Marcos Gonçalves, and Mohammed Zaki. Multi-label lazy associative classification. In *Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD 2007, pages 605–612, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74975-2. doi: http://dx.doi.org/10.1007/978-3-540-74976-9_64. URL `http://dx.doi.org/10.1007/978-3-540-74976-9_64`.

[104] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier. On label dependence in multi-label classification. In *Second International Workshop on Learning from Multi-Label Data*, pages 5–12, Haifa, Israel, June 2010.

[105] Concha Bielza and Pedro Larrañaga. Multi-label classification methods, 2010. URL `http://www.dia.fi.upm.es/~concha/muia-aa-multilabel.pdf`.

[106] José M. Carmona-Cejudo, Manuel Baena-García, José del Campo-Ávila, Rafael Morales Bueno, and Albert Bifet. Gnusmail: Open framework for on-line email classification. In *ECAI*, pages 1141–1142, 2010.

[107] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Multilabel text classification for automated tag suggestion. In *In: Proceedings of the ECML/PKDD-08 Workshop on Discovery Challenge*, 2008.

[108] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11

(1):10–18, 2009. ISSN 1931-0145. URL `http://dx.doi.org/10.1145/1656274.1656278`.

[109] José M. Carmona-Cejudo, Manuel Baena-García, and R. Morales Bueno, Rafael. Amc-gnusmail: an extensible machine learning based framework for email classification. In *Proceedings of TTIA '09*. IASK, 2009.

[110] Charu C. Aggarwal and ChengXiang Zhai, editors. *Mining Text Data*. Springer, 2012. ISBN 978-1-4419-8462-3.

[111] João Gama. *Knowledge Discovery from Data Streams*. CRC Press, 2010.

[112] José M. Carmona-Cejudo, Gladys Castillo, Manuel Baena-García, and Rafael Morales Bueno. A comparative study on feature selection and adaptive strategies for email foldering. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 1294 –1299, nov. 2011. doi: 10.1109/ISDA.2011.6121838.

[113] José M. Carmona-Cejudo, Manuel Baena-García, José del Campo-Ávila, Albert Bifet, João Gama, and Rafael Morales Bueno. Online evaluation of email streaming classifiers using gnusmail. In João Gama, Elizabeth Bradley, and Jaakko Hollmén, editors, *IDA*, volume 7014 of *Lecture Notes in Computer Science*, pages 90–100. Springer, 2011. ISBN 978-3-642-24799-6.

[114] João Gama, Raquel Sebastião, and Pedro Rodrígues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2009)*, pages 329–338, 2009.

[115] Albert Bifet and Richard Kirkby. Data stream mining a practical approach, 2011.

[116] Albert Bifet and Richard Kirkby. http://sourceforge.net/projects/moa-datastream/. URL `http://sourceforge.net/projects/moa-datastream/`.

[117] Jürgen Beringer and Eyke Hüllermeier. Efficient instance-based learning on data streams. *Intell. Data Anal.*, 11:627–650, December 2007. ISSN 1088-467X. URL `http://dl.acm.org/citation.cfm?id=1368018.1368022`.

[118] Indrė Žliobaitė. Learning under concept drift: an overview. *CoRR*, abs/1010.4784, 2010.

[119] Gladys Castillo. *Adaptive Learning Algorithms for Bayesian Network Classifiers*. PhD thesis, Departamento de Matemática, Univeridade de Aveiro, 2006.

[120] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis P. Vlahavas. On the utility of incremental feature selection for the classification of textual data streams. In *Panhellenic Conference on Informatics*, pages 338–348, 2005.

[121] Brent Wenerstrom and Christophe Giraud-Carrier. Temporal data mining in dynamic feature spaces. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 1141–1145, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2701-9. doi: 10.1109/ICDM.2006.157. URL `http://dx.doi.org/10.1109/ICDM.2006.157`.

[122] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000. URL `citeseer.ist.psu.edu/article/domingos00mining.html`.

[123] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. doi: http://doi.acm.org/10.1145/502512.502529. URL `http://doi.acm.org/10.1145/502512.502529`.

[124] João Gama, Pedro Medas, and Ricardo Rocha. Forest trees for on-line data. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 632–636, New York, NY,

USA, 2004. ACM. ISBN 1-58113-812-1. doi: http://doi.acm.org/10.1145/967900.968033. URL http://doi.acm.org/10.1145/967900.968033.

[125] Qiang Ding, Qin Ding, and William Perrizo. Decision tree classification of spatial data streams using peano count trees. In *Proceedings of the 2002 ACM symposium on Applied computing*, SAC '02, pages 413–417, New York, NY, USA, 2002. ACM. ISBN 1-58113-445-2. doi: http://doi.acm.org/10.1145/508791.508870. URL http://doi.acm.org/10.1145/508791.508870.

[126] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. doi: http://doi.acm.org/10.1145/956750.956778. URL http://doi.acm.org/10.1145/956750.956778.

[127] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explor. Newsl.*, 3:1–10, January 2002. ISSN 1931-0145. doi: http://doi.acm.org/10.1145/507515.507517. URL http://doi.acm.org/10.1145/507515.507517.

[128] Spiros Papadimitriou, Anthony Brockwell, and Christos Faloutsos. Adaptive, hands-off stream mining. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, pages 560–571. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL http://dl.acm.org/citation.cfm?id=1315451.1315500.

[129] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. On demand classification of data streams. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 503–508, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: http://doi.acm.org/10.1145/1014052.1014110. URL http://doi.acm.org/10.1145/1014052.1014110.

[130] Mohamed Medhat Gaber, Arkady B. Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.

[131] Mark Last. Online classification of nonstationary data streams. *Intell. Data Anal.*, 6:129–147, April 2002. ISSN 1088-467X. URL `http://dl.acm.org/citation.cfm?id=1293986.1293988`.

[132] Francisco Ferrer-Troyano, Jesús S. Aguilar-Ruiz, and Riquel José C. Discovering decision rules from numerical data streams. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 649–653, New York, NY, USA, 2004. ACM. ISBN 1-58113-812-1. doi: http://doi.acm.org/10.1145/967900.968036. URL `http://doi.acm.org/10.1145/967900.968036`.

[133] Yan nei Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In *In PKDD*, pages 108–120. Springer, 2005.

[134] Nikunj C Oza and Stuart Russell. Online bagging and boosting. *Artificial Intelligence and Statistics*, 1(4):105–112, 2001. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1571498`.

[135] Gladys Castillo and João Gama. An Adaptive Prequential Learning Framework for Bayesian Network Classifiers. In *Knowledge Discovery in Databases: PKDD 2006*, volume 4213, chapter 11, pages 67–78. Springer Berlin Heidelberg, 2006.

[136] John S Oakland. *Statistical Process Control, Fifth Edition*. Butterworth-Heinemann, 2003. ISBN 0750657669.

[137] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*, 2007.

[138] Raquel Sebastião, João Gama, Pedro Pereira Rodrígues, and João Bernardes. Monitoring incremental histogram distribution for change detection in data streams. In *KDD Workshop on Knowledge Discovery from Sensor Data*, pages 25–42, 2008.

[139] E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41 (1/2):100–115, 1954. ISSN 00063444. doi: 10.2307/2333009. URL `http://dx.doi.org/10.2307/2333009`.

[140] João Gama, Pedro Medas, Gladys Castillo, and Pedro Pereira Rodrigues. Learning with drift detection. In Ana L. C. Bazzan and Sofiane Labidi, editors, *SBIA*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 2004. ISBN 3-540-23237-0.

[141] Manuel Baena-García, José Del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales Bueno. Early drift detection method. *Fourth International Workshop on Knowledge Discovery from Data Streams*, 6:77–86, 2006. URL `http://eprints.pascal-network.org/archive/00002509/`.

[142] Jeonghoon Lee and F. Magoules. Detection of concept drift for learning from stream data. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 241–245, 2012.

[143] A. P. Dawid. Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A*, 147 (2):278–292, 1984.

[144] R. Bekkerman, A. Mccallum, G. Huang, and Others. Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora. *Center for Intelligent Information Retrieval, Technical Report IR*, 418, 2004. URL `http://www.cs.umass.edu/~{}mccallum/papers/foldering-tr05.pdf`.

[145] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[146] Patrick Pantel and Dekang Lin. Spamcop: A spam classification & organization program. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[147] Emanuele Sabellico and Daniele Repici. http://mailclassifier.mozdev.org/. URL `http://mailclassifier.mozdev.org/`.

[148] Jake D. Brutlag and Christopher Meek. Challenges of the email domain for text classification. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 103–110, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2.

[149] N. Chaudhry, K. Shaw, and M. Abdelguerfi, editors. *Stream Data Management*. Advances in Database Systems. Springer, 2005.

[150] G. Manco, E. Masciari, and A. Tagarelli. A framework for adaptive mail classification. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2002)*, pages 387–392, 2002.

[151] R. B. Segal and J. O. Kephart. Incremental learning in SwiftFile. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pages 863–870, 2000. ISBN 1-55860-707-2.

[152] R. Barrett and T. Selker. AIM: A new approach for meeting information needs. Technical report, IBM Almaden Research Center, Almaden, CA, 1995.

[153] Matthew Chang and Chung Keung Poon. Using phrases as features in email classification. *Journal of Systems and Software*, 82(6):1036 – 1045, 2009. ISSN 0164-1212. doi: 10.1016/j.jss.2009.01.013. URL `http://www.sciencedirect.com/science/article/pii/S0164121209000089`.

[154] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis P. Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowl. Inf. Syst.*, 22(3):371–391, 2010.

[155] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. In *SIGKDD Explorations*, volume 11-1, pages 10–18, 2009.

[156] P. Bermejo, J. A. Gámez, J. M. Puerta, and R. Uribe-Paredes. Improving KNN-based e-mail classification into folders generating class-balanced datasets. In *Proceedings of the 12th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Sytems (IPMU-2008)*, pages 529–536, 2008.

[157] Francisco Fernández-Navarro, César Hervás-Martínez, and Pedro Antonio Gutiérrez. A dynamic over-sampling procedure based on sensitivity for multi-class problems. *Pattern Recognition*, 44(8):1821–1833, 2011.

[158] Qin Zhang, Feifei Li, and Ke Yi. Finding frequent items in probabilistic data. In *SIGMOD Conference*, pages 819–832, 2008.

[159] P. Viswanath, M. Narasimha Murty, and S. Kambala. An efficient parzen-window based network intrusion detector using a pattern synthesis technique. In *PReMI*, pages 799–804, 2005.

[160] João Gama. Iterative bayes. *Theor. Comput. Sci.*, 292(2): 417–430, 2003.

[161] José M. Carmona-Cejudo, Gladys Castillo, Manuel Baena-García, and Rafael Morales Bueno. A comparative study on feature selection and adaptive strategies for email foldering using the abc-dynf framework. *Knowledge-Based Systems*, (0): –, 2013. ISSN 0950-7051.

[162] Gladys Castillo and João Gama. Adaptive bayesian network classifiers. *Intell. Data Anal.*, 13(1):39–59, 2009.

[163] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Dynamic feature space and incremental feature selection for the classification of textual data streams. In *in ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams. 2006*, page 107. Springer Verlag, 2006.

[164] Sarah Jane Delany, Padraig Cunningham, and Alexey Tsymbal. A comparison of ensemble and case-base maintenance techniques for handling concept drift in spam filtering. In Geoff Sutcliffe and Randy Goebel, editors, *FLAIRS Conference*, pages 340–345. AAAI Press, 2006.

[165] João Gama and Gladys Castillo. Adaptive bayes. IBERAMIA 2002, pages 765–774, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-00131-X. URL `http://portal.acm.org/citation.cfm?id=645991.674817`.

[166] H. Finner. Stepwise multiple test procedures and control of directional errors. *The Annals of Statistics*, 27(1):274–289, 1999. ISSN 0090-5364.

[167] Harun Uğuz. A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm. *Knowledge-Based Systems*, 24 (7):1024 – 1032, 2011. ISSN 0950-7051.

[168] Lei Tang and Huan Liu. Bias analysis in text classification for highly skewed data. In *ICDM*, pages 781–784. IEEE Computer Society, 2005. ISBN 0-7695-2278-5. URL `http://doi.ieeecomputersociety.org/10.1109/ICDM.2005.34`.

[169] Yanlei Diao, Hongjun Lu, and Dekai Wu. A comparative study of classification based personal e-mail filtering. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, PADKK '00, pages 408–419, London, UK, 2000. Springer-Verlag. ISBN 3-540-67382-2. URL `http://portal.acm.org/citation.cfm?id=646418.693345`.

[170] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of*

the Fourteenth International Conference on Machine Learning, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-486-3. URL http://dl.acm.org/citation.cfm?id=645526.657137.

[171] Svetlana Kiritchenko, Stan Matwin, and Suhayya Abu-Hakima. Email classification with temporal features. In *Intelligent Information Systems*, pages 523–533, 2004.

[172] J. Clark, I. Koprinska, and J. Poon. A neural network based approach to automated e-mail classification. In *Web Intelligence*, pages 702–705, 2003.

[173] Li-Ping Jing, Hou-Kuan Huang, and Hong-Bo Shi. Improved feature selection approach tfidf in text mining. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, volume 2, pages 944 – 946, 2002.

[174] B Yu and Z Xu. A comparative study for content-based dynamic spam classification using four machine learning algorithms. *Knowledge-Based Systems*, 21(4):355–362, May 2008. ISSN 09507051. doi: 10.1016/j.knosys.2008.01.001. URL http://linkinghub.elsevier.com/retrieve/pii/S0950705108000026.

[175] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513 – 523, 1988. ISSN 0306-4573. doi: DOI:10.1016/0306-4573(88)90021-0. URL http://www.sciencedirect.com/science/article/B6VC8-469WV05-1/2/d251fa7251ec6c4247f833f88efd3068.

[176] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at trec-3. In *TREC*, pages 0–, 1994.

[177] Xiaojun Wan, Jianwu Yang, and Jianguo Xiao. Towards an iterative reinforcement approach for simultaneous document summarization and keyword extraction. In *ACL*, 2007.

[178] Fei Liu, Feifan Liu, and Yang Liu. Automatic keyword extraction for the meeting corpus using supervised approach and bigram expansion. In *Spoken Language Technology Workshop, 2008. SLT 2008. IEEE*, pages 181 –184, dec. 2008.

[179] H. P. Luhn. The automatic creation of literature abstracts. *IBM J. Res. Dev.*, 2(2):159–165, April 1958. ISSN 0018-8646.

[180] P. B. Baxendale. Machine-made index for technical literature: an experiment. *IBM J. Res. Dev.*, 2(4):354–361, October 1958. ISSN 0018-8646.

[181] H. P. Edmundson. New methods in automatic extracting. *J. ACM*, 16(2):264–285, April 1969. ISSN 0004-5411.

[182] Alberto Muñoz. Compound key word generation from document databases using a hierarchical clustering art model. *Intell. Data Anal.*, 1(1-4):25–48, 1997.

[183] Bruce Krulwich and Chad Burkey. The infofinder agent: Learning user interests through heuristic phrase extraction. *IEEE Expert*, 12(5):22–27, 1997.

[184] Wei Wu, Bin Zhang, and Mari Ostendorf. Automatic generation of personalized annotation tags for twitter users. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 689–692, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. ISBN 1-932432-65-5.

[185] Qinglin Guo and Ming Zhang. Multi-documents automatic abstracting based on text clustering and semantic analysis. *Knowledge-Based Systems*, 22(6):482 – 485, 2009. ISSN 0950-7051.

[186] Thomas Gottron. Document word clouds: Visualising web documents as tag clouds to aid users in relevance decisions. In Maristella Agosti, Jos Borbinha, Sarantos Kapidakis, Christos Papatheodorou, and Giannis Tsakonas, editors, *Research and Advanced Technology for Digital Libraries*, volume 5714

of *Lecture Notes in Computer Science*, pages 94–105. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-04345-1.

[187] Nilesh Bansal and Nick Koudas. Blogscope: spatio-temporal analysis of the blogosphere. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 1269–1270, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7. doi: http://doi.acm.org/10.1145/1242572.1242802. URL `http://doi.acm.org/10.1145/1242572.1242802`.

[188] Pierre-Antoine Champin, Peter Briggs, Maurice Coyle, and Barry Smyth. Coping with noisy search experiences. *Know.-Based Syst.*, 23(4):287–294, May 2010. ISSN 0950-7051.

[189] Samuel Huston, J. Shane Culpepper, and W. Bruce Croft. Sketch-based indexing of n-words. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 1864–1868, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1156-4. doi: 10.1145/2396761.2398533. URL `http://doi.acm.org/10.1145/2396761.2398533`.

[190] Yaakov HaCohen-Kerner. Automatic extraction of keywords from abstracts. In Vasile Palade, Robert Howlett, and Lakhmi Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 2773 of *Lecture Notes in Computer Science*, pages 843–849. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-40803-1.

[191] Parantu K. Shah, Carolina Pérez-Iratxeta, Peer Bork, and Miguel A. Andrade. Information extraction from full text scientific articles: Where are the keywords? *BMC Bioinformatics*, 4:20, 2003.

[192] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, pages 398–412, 2005.

[193] Xiao-Dong Zhu and Zhi-Qiu Huang. Conceptual modeling rules extracting for data streams. *Knowledge-Based Systems*, 21(8):934 – 940, 2008. ISSN 0950-7051.

[194] J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

[195] Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

[196] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[197] Thorsten Brants and Francine Chen. A system for new event detection. In *SIGIR*, pages 330–337, 2003.

[198] Giridhar Kumaran and James Allan. Text classification and named entities for new event detection. In *SIGIR*, pages 297–304, 2004.

[199] Hidenao Abe and Shusaku Tsumoto. Detecting temporal patterns of importance indices about technical phrases. In Juan Velsquez, Sebastin Ros, Robert Howlett, and Lakhmi Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, volume 5712 of *Lecture Notes in Computer Science*, pages 252–258. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-04591-2.

[200] Manuel Baena-García, José M. Carmona-Cejudo, Gladys Castillo, and Rafael Morales Bueno. Term frequency, sketched inverse document frequency. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 1294 –1299, nov. 2011. doi: 10.1109/ISDA.2011.6121838.

[201] José M. Carmona-Cejudo, Manuel Baena-García, Gladys Castillo, and Rafael Morales Bueno. Online calculation of word-clouds for efficient label summarization. In *Intelligent*

Systems Design and Applications (ISDA), 2011 11th International Conference on, pages 1294 –1299, nov. 2011. doi: 10.1109/ISDA.2011.6121838.

[202] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[203] Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *VLDB J.*, 19(1):3–20, 2010.

[204] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357. Morgan Kaufmann, 2002.

[205] Nuno Homem and João Carvalho. Estimating top-k destinations in data streams. In Eyke Hllermeier, Rudolf Kruse, and Frank Hoffmann, editors, *Computational Intelligence for Knowledge-Based Systems Design*, volume 6178 of *Lecture Notes in Computer Science*, pages 290–299. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-14048-8.

[206] Amit Goyal, Hal Daumé, III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1093–1103, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[207] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[208] Luca Bortolussi, Liviu Petrisor Dinu, and Andrea Sgarro. Spearman permutation distances and shannon's distinguishability. *Fundam. Inform.*, 118(3):245–252, 2012.

[209] R. R. Johnson, editor. *Elementary statistics*. PWS Publishing Co., Boston, MA, USA, 1988. ISBN 0-534-91719-4.

[210] M. F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130137, 1993. ISSN 0033-0337.

[211] T. Snowsill, F. Nicart, M. Stefani, T. De Bie, and N. Cristianini. Finding surprising patterns in textual data streams. In *Cognitive Information Processing (CIP), 2010 2nd International Workshop on*, pages 405 –410, june 2010. doi: 10.1109/CIP.2010.5604085.

[212] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1:131–156, 1997.

[213] Maria Fernanda Caropreso, Stan Matwin, and Fabrizio Sebastiani. Text databases & document management. chapter A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization, pages 78–102. IGI Publishing, Hershey, PA, USA, 2001. ISBN 1-878289-93-4. URL `http://dl.acm.org/citation.cfm?id=374247.374254`.

[214] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, CIKM '98, pages 148–155, New York, NY, USA, 1998. ACM. ISBN 1-58113-061-9. doi: 10.1145/288627.288651. URL `http://doi.acm.org/10.1145/288627.288651`.

[215] Hwee Tou Ng, Wei Boon Goh, and Kok Leong Low. Feature selection, perceptron learning, and a usability case study for text categorization. *SIGIR Forum*, 31(SI):67–73, July 1997. ISSN 0163-5840. doi: 10.1145/278459.258537. URL `http://doi.acm.org/10.1145/278459.258537`.

[216] Erik D. Wiener, Jan O. Pedersen, and Andreas S. Weigend. A neural network approach to topic spotting. In *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*, pages 317–332, Las Vegas,

US, 1995. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.6608.

[217] Luigi Galavotti, Fabrizio Sebastiani, and Maria Simi. Experiments on the use of feature selection and negative evidence in automated text categorization. In *Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries*, ECDL '00, pages 59–68, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-41023-6. URL http://dl.acm.org/citation.cfm?id=646633.699638.

[218] Luka Cehovin and Zoran Bosnic. Empirical evaluation of feature selection methods in classification. *Intell. Data Anal.*, 14 (3):265–281, 2010.

[219] Li-Ping Jing, Hou-Kuan Huang, and Hong-Bo Shi. Improved feature selection approach tfidf in text mining. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, volume 2, pages 944 – 946 vol.2, 2002. doi: 10.1109/ICMLC.2002.1174522.

[220] Odysseas Papapetrou, George Papadakis, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient term cloud generation for streaming web content. In *Proceedings of the 10th international conference on Web engineering*, ICWE'10, pages 385–399, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13910-8, 978-3-642-13910-9. URL http://portal.acm.org/citation.cfm?id=1884110.1884142.

[221] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, and Hongjun Lu. Classifying high-speed text streams. In *WAIM*, pages 148–160, 2003.

[222] A. Goyal, J. Jagarlamudi, H. Daum III, and S. Venkatasubramanian. Sketching techniques for large scale NLP. In *Proceedings of the NAACL HLT 2010 Sixth Web as Corpus Workshop*, page 1725, 2010.

[223] Deepak Ravichandran, Patrick Pantel, and Eduard H. Hovy. Randomized algorithms and nlp: Using locality sensitive hash

functions for high speed noun clustering. In Kevin Knight, Hwee Tou Ng, and Kemal Oflazer, editors, *ACL*. The Association for Computer Linguistics, 2005.

[224] Abby Levenberg and Miles Osborne. Stream-based randomised language models for smt. In *EMNLP*, pages 756–764. ACL, 2009. ISBN 978-1-932432-59-6, 978-1-932432-62-6, 978-1-932432-63-3.

[225] S.K. Pal, P. Sardana, and K. Yadav. Efficient multilingual keyword search using bloom filter for cloud computing applications. In *Advanced Computing (ICoAC), 2012 Fourth International Conference on*, pages 1–7, 2012.

[226] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[227] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better bloom filter. In *In Proc. the 14th Annual European Symposium on Algorithms (ESA 2006)*, pages 456–467, 2006.

[228] George Karypis. *CLUTO: A Clustering Toolkit*, 2003. URL `http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download`.

[229] R. Luus and T. H. I. Jaakola. Optimization by Direct Search and Systematic Reduction of the Size of the Search Region. *American Institute of Chemical Engineers (AIChE) Journal*, 19:760–766, 1973.

Index