UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA INFORMÁTICA

Aceleración basada en TBB de una aplicación de Análisis de Sentimientos en Twitter

Acceleration based on TBB of a Twitter Sentiment Analysis Application

Realizado por
Susana Ledesma Jiménez

Tutorizado por
Mª Ángeles González Navarro

Departamento
Arquitectura de Computadores
UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO, 2020

Resumen

El análisis de sentimientos es una técnica que permite extraer la polaridad (i.e. positividad, negatividad o neutralidad) de un texto. Esto unido al incremento del uso de las redes sociales hace que sea una técnica ideal para monitorizarlas y conseguir una visión general de lo que el público opina. Sin embargo, esta tarea es computacionalmente muy costosa, ya que para aumentar la precisión de estos estudios se necesitan grandes cantidades de datos. Una de las soluciones a este problema es emplear técnicas de paralelización, debido a que por lo general cada uno de estos ítems se procesa de forma independiente, por lo que estas técnicas se puede aplicar sin problemas. Por lo tanto, en este proyecto, nos centramos en el estudio de una aplicación de análisis de sentimientos implementada con la librería Dispel4py (basada en Python), que se optimizará aplicando técnicas de paralelización. Específicamente usaremos la librería Intel Threading Building Blocks de C++ para acelerar una de las etapas computacionalmente más costosa de dicha aplicación. Así mismo se contribuirá a mejorar la precisión en el análisis de la aplicación original aplicando técnicas de machine learning que se han propuesto en la literatura científica relacionada con el análisis de sentimientos.

Palabras clave : Análisis de sentimientos, Aceleración, TBB, Python, C++.

Abstract

Sentiment analysis is a technique that allows the user to extract the polarity of a text (i.e. positivity, negativity or neutrality). Due to the incremented use of social media, sentiment analysis makes an ideal approach to monitor them and obtain a general overview of what the trend is. However, this task is computationally expensive because of the huge amount of data that this kind of techniques require in order to be accurate in their classification. A solution to this problem is to apply parallelization, it can be easily applied because each item is processed independently, so this technique can be employed without any issue. Therefore, in this project, we are going to study a sentiment analysis application implemented with Dispel4py, a Python library. One of this application's branches will be optimized by using parallelization supplied by C ++ Intel Threading Building Blocks. In addition, we will make a contribution by improving the accuracy of the original application by using machine learning techniques that have been explained in the scientific literature related to sentiment analysis field.

Key words : Sentiment analysis, Acceleration, TBB, Python, C++.

Índice

1 Introducción

1.1 Motivación, objetivos y organización

El análisis de datos es una técnica que actualmente se utiliza en casi todos los ámbitos, ya que la capacidad para extraer información y realizar decisiones en base a esta es muy útil. Bien sea empleando data mining para encontrar patrones, bussiness inteligence para decidir las tácticas que debe seguir una empresa, o text analytics, procesando el lenguaje natural con técnicas como por ejemplo, análisis de sentimientos. La cual es objeto de estudio en este proyecto.

Para poder procesar esta cantidad de datos es interesante explotar el concepto de paralelismo que ofrecen actualmente los procesadores multicore, y para este propósito se empleará Threading Building Blocks. Ya que se especializa en mejorar el modelo de ejecución paralela basado en tareas, por lo que aprender a desenvolverse en ambas tecnologías sin duda puede ser de gran utilidad. A partir de esta tecnología se paralelizará la aplicación de referencia de este TFG: una aplicación de análisis de sentimientos desarrollada para Dispel4py, una librería que facilita el *stream* de datos en Python.
La aplicación de analisis de sentimientos de la que partimos se optimizará en dos partes:

- Añadir una nueva etapa de clasificación de tweets basada en SVM, puesto que la apliación final solo soporta clasificación basada en lexicon.
- Aceleración de una de las etapas de nuestra aplicación, codificando en C++ las funciones correspondientes que se paralelizarán utilizando la librería Intel Threading Building Blocks.

Con lo cual en este proyecto se pretende aprender a utilizar los lenguajes de programación y librerías:

- Python, para realizar la implementaición de una técnica de análisis de sentimientos basada en machine learning.
- C++, necesario para realizar la paralelización de una de las etapas implementadas en Dispel4py. Y la extensión Cython para implementar módulos que conecten las funciones C++ con Python.

- TBB (Threading Buildding Blocks), la librería de paralelización en C++.

Por lo que el documento presenta los siguientes capítulos:

1. Introducción al análisis de sentimientos, su desarrollo a lo largo del tiempo y el estado del arte.

2. Descripción de Dispel4Py; el entorno original donde se ha desarrollado la aplicación de análisis de sentimientos objeto de este proyecto, descripción en detalle de esta aplicación

3. Introducción a Intel Threading Building Blocks, librería que se empleará para realizar la paralelización, como se ha comentado previamente.

4. A partir de las tecnologías descritas, se explican las optimizaciones realizadas sobre la aplicación original.

5. Evaluación del rendimiento de las optimizaciones propuestas en el capítulo anterior, finalizando con las conclusiones y futuras aplicaciones.

1.2 Tecnologías

1.2.1 Python

Python es un lenguaje interpretado multipropósito a alto nivel. Creado por Guido van Rossum y lanzado por primera vez en 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources.8. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: str, list, dict, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3; van Rossum describe el módulo como «una de las mayores unidades de programación de Python».3.Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. ya que pueden incluirse con tecnologías como Cython.[1]

1.2.2 Cython

Cython es un lenguaje de programación para simplificar la escritura de módulos de extensión para Python en C y C++. Siendo estrictos, la sintaxis de Cython es la misma de Python pero con algunos agregados:

Se pueden llamar funciones en C, o funciones/métodos de C++, directamente desde el código en Cython. Es posible usar tipos estáticos en las variables (enteros, flotantes, o cualquier tipo de dato). Cython compila a código en C o C++ desde Python, y el resultado puede ser usado desde Python como un "Modulo de extensión", o como una aplicación embebida en el intérprete. CPython.[2] [3]

1.2.3 C++14

C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos. Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.[4]

2 Análisis de sentimientos

2.1 ¿Qué es el análisis de sentimientos?

El análisis de sentimientos o también conocido como minería de opinión es una técnica de procesamiento de lenguaje natural, análisis de texto y lingüística computacional. La cual se utiliza para extraer de un texto su polaridad o subjetividad y así clasificarlo según si expresa una connotación positiva, negativa o neutra. Su uso está muy extendido , ya que la capacidad de poder realizar la clasificación masiva de documentos a la par que se extrae información de ellos es muy útil. Se emplea en muchos ámbitos, por ejemplo para conocer la opinión sobre un producto a través de sus *reviews* o también se puede emplear para conocer la tendencia de opinión que se sigue entre los usuarios de las redes sociales.

Para desarrollar este método se pueden utilizar técnicas basadas en lexicon, consistentes en un diccionario de palabras claves (normalmente adjetivos) con una puntuación que simboliza el peso de cada palabra, o bien basadas en machine learning, entrenando un modelo con datos previamente clasificados para que así tenga la capacidad de clasificar con más precisión.

2.2 Evolución del análisis de sentimientos

El análisis de sentimientos es un tema que lleva siendo estudiado desde hace tiempo. Sus orígenes se pueden trazar hasta 1940 cuando el primer paper sobre este fue publicado, se tituló "The Cross-Out Technique as a Method in Public Opinion Analysis"[5] el cual aborda las técnicas utilizadas y detecta un gran problema al tratar de clasificar frases ambiguas.

Tan pronto como la capacidad computacional evolucionó lo suficiente, se desarrollaron técnicas para el análisis de sentimientos basadas en computación. Las cuales quedaron reflejados en artículos como ”Elicitation, Assessment, and Pooling of Expert Judgments Using Possibility Theory”[6] . Pero no sería hasta años más tarde que alcanzaría el auge esta tecnología.

Hacia mediados de los 2000 internet ya era bastante importante y el análiss

de sentimientos vuelve a ganar importancia. Artículos con un enfoque moderno son desarrollados , como por ejemplo, “Mining and summarizing customer reviews"[7], el cual clasifica opiniones de los productos vendidos en una web usando una técnica parecida a los lexicon.

En los siguientes años el análisis de sentimientos se aborda fundamentalmente mediante dos tipos de técnicas :

- Machine Learning (Naive Bayes, maximum Entropy o support vector machine), estas se valen de *datasets* previamente clasificados para crear un modelo y poder clasificar con mayor precisión.
- Lexicon, los cuales clasifican siguiendo un diccionario que contiene una puntuación para cada una de las palabras, por lo que el resultado final se obtienen combinando cada una de estas puntuaciones.

Estas técnicas también pueden combinarse para mejorar el rendimiento, aunque todavía no obtienen el resultado deseado por culpa de los problemas que conlleva el procesamiento del lenguaje natural. Problemas tales como el sarcasmo o la interpretación basada en contexto. En el caso específico del análisis de sentimientos en Twitter el cual además es el caso elegido para este proyecto, también se enfrenta a la dificultad de la brevedad del texto escrito y las abreviaciones o las nuevas formas de expresión que surgen por culpa de esta. Para resolver estos problemas se pueden aplicar diferentes soluciones como Preprocesamiento específico (eliminar, reemplazar o corregir algunas de estas características) aunque no tienen todavía el resultado esperado [8].

El 99 % de los *papers* que hacen referencia al análisis de sentimientos están publicados después de 2004 [8], con lo cual se puede inferir que está muy relacionado a la capacidad de procesar grandes cantidades de datos.

2.3 Estado del arte

En esta sección se van a introducir las técnicas para el análisis de sentimientos que se utilizan con más frecuencia. Divididas en: tecnicas basadas en machine learning y técnicas basadas en lexicon, además de analizar el estado del arte en Twitter, ya que es la aplicación de este método que se realizará posteriormente.

2.3.1 Machine Learning

- Naive Bayes es un clasificador probabilístico basado en el teorema de Bayes [1], el cual resulta muy útil para categorizar texto.
  Entre los clasificadores basados en este modelo el más utilizado es el modelo multinomial. En este, un documento está formado por una secuencia ordenada de palabras, todas pertenecientes a un conjunto V. Cada uno de estos es escogido siguiendo el modelo bolsa de palabras [2].

  Para estimar la probabilidad de cada palabra dada su clase se sigue:

$$P(w_t|c_k)\ \frac{\sum_{i1}^{|D|} N_{t,i} * P(c_k|d_i)\ 1}{\sum_{t1}^{|V|}\sum_{t1}^{|D|} N_{t,i} * P(c_k|d_i)\ |V|} \tag{1}$$

  Donde $N_{t,i}$ es el número de apariciones de la palabra w_t en el documento d_i, |V| y |D| se refieren al cardinal del vocabulario y del dataset respectivamente. Siendo $P(c_k|d_i)$ la probabilidad condicionada del suceso c dado anteriormente el suceso d. Por lo que finalmente el clasificador se puede definir como [11]:

$$max_{ck}\,(P(c_k) \prod_{t\in|V|} (P(w_t|c_k))^{N_{t,i}}) \tag{2}$$

- Support Vector Machine, esta técnica sirve generalmente para resolver problemas de clasificación en general linealmente separables. Funciona bastante bien ya que tiene facilidad para manejar grandes cantidades de datos incluso cuando estos están muy dispersos entre sí.

  SVM realiza la clasificación encontrado el hiperplano óptimo que separa a las diferentes clases. Se ayuda de la función kernel que pueden ir desde la función lineal (soluciones lineales) hasta la función radial (soluciones no lineales) la cual busca un hiperplano en n+1 dimensiones computando la transformación del producto escalar entre los puntos para hallarlo.

[1] https://es.wikipedia.org/wiki/Teorema_de_Bayes
[2] https://es.wikipedia.org/wiki/Modelo_bolsa_de_palabras

La forma lineal sirve para resolver problemas binarios ya que separa los datos en dos grupos a través del hiperplano anteriormente mencionado. Existen muchos hiperplanos que separen los puntos pero se busca el que tenga el máximo margen, es decir, que maximice la distancia entre los puntos de las distintas clases. Para calcularlo se ayuda de los vectores de soporte,que son los puntos más cercanos al hiperplano.

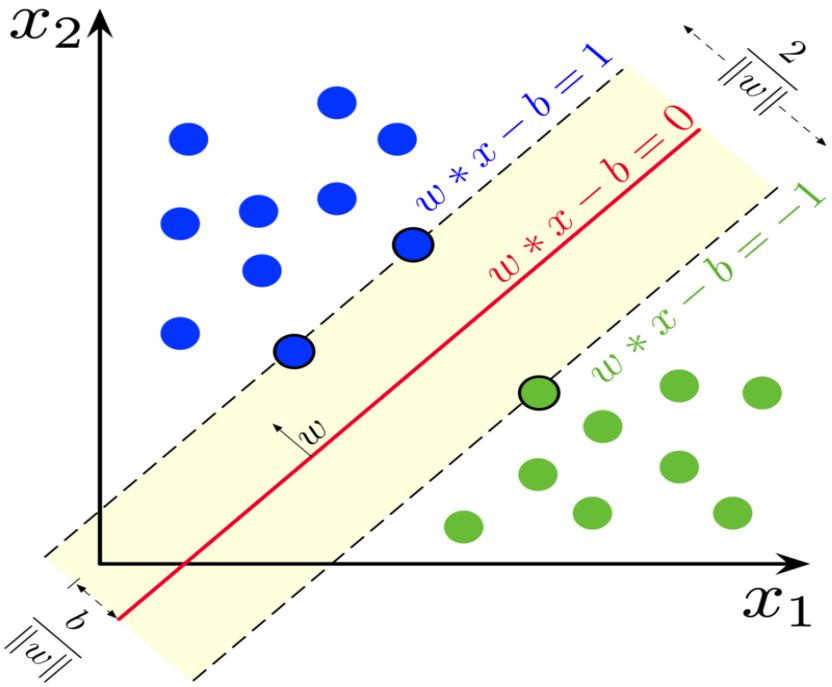

Figura 1: Explicación gráfica SVM [13]

Por tanto son estos puntos los que determinan dónde corta el hiperplano. Y para calcularlos se utiliza la función de pérdida que ayuda a máximizar estos márgenes, en este caso es la *hinge loss*. La cual penaliza tanto los fallos, como acercarse demasiado a los vectores de soporte.

2.3.2 Lexicon

Las técnicas basadas en lexicon tienen como objetivo calcular la polaridad de un documento según la semántica de las frases o palabras en este, empleando diccionarios. Estos diccionarios se pueden realizar de forma manual o bien de forma automática utilizando palabras clave para expandir la lista de estas.

Por lo que, al centrarse tanto en la semántica pueden destarcase los siguientes puntos.[16]

- Los adjetivos son la fuente principal para extraer la subjetividad del texto, aunque también se hace uso de adverbios, verbos, o frases claves.
  Una versión simple de este proceso es seleccionar cada una de estas palabras y hallar la media de sus puntuaciones, como podría ser el caso de AFINN. Sin embargo este método obviamente depende mucho de la forma en la que esté hecha el diccionario. Pese a que los diccionarios realizados de forma automática o semi-automática tienen muchas ventajas, algunos términos no terminan de estar bien ponderados y eso puede tener mucho impacto en la puntuación final. Por el contrario, los diccionarios manualmente realizados suelen obtener mejores clasificaciones.

- Si solo se clasifican a partir de su significado pueden cometerse errores como por ejemplo con las negaciones. "Not good" tiene un significado completamente opuesto a "good" pero si solo se clasificase esta, tendría una puntuación positiva. Por lo tanto es importante tener en cuenta el contexto. Para solucionar este problema se puede realizar una búsqueda hacia atrás hasta el inicio de la oración.

- Los modos irreales, los cuales expresan hechos hipotéticos, por ejemplo "I thought that it would be a great movie, but it wasn't". Great es un adjetivo con significado positivo pero al estar en una frase en forma irreal, está obteniendo una connoatación negativa, cambiando radicalmente su sentido. Por lo que, o bien se ignoran al estar en esta forma, o se invierte su puntuación.

A partir de las características detalladas, el método AFINN podría verse como un método muy simple el cual puede llegar a obtener un resultado tan bajo de precisión como del 42.45 % analizando el dataset *Comments_NYT* [17].

Por el contrario SentiWordNet 3.0 (SWN3) es un clasificador algo más complejo, ya que evalúa todos los anillos de sinónimos[3] (synsets) parte fundamental de esta técnica. Cada uno de estos anillos tiene asociado tres valores:

[3]https://es.wikipedia.org/wiki/Anillo_de_sin%C3%B3nimos

positivo, negativo y "objetivo".Estas puntuaciónes van en un intervalo $[0,1]$ y la combinación de los tres resultados suma 1. El diccionario que utiliza está generado de forma semi-supervisada ya que emplea el uso de "semillas" para formar los synsets.

2.4 Análisis de sentimientos en Twitter

El análisis de sentimientos aplicado a Twitter es un área especializado que está recibiendo mucha atención últimamente. Debido a que Twitter tiene 134 millones de usuarios activos, es decir, contiene una población considerable que llega a generar 500 millones de tweets al día. Por lo tanto esto hace de Twitter una red social ideal para realizar esta técnica. Sin embargo, cada uno de estos tweets tiene como máximo 280 caracteres, aunque anteriormente este número se limitaba a 140. Debido a esto, se desarrollan nuevas expresiones y formas de comunicarse, haciendo que el análisis de sentimientos tradicional no resulte del todo efectivo. Surgen nuevas dificultades a la hora de clasificar estos textos, a continuación se presentan algunas de estas y propuestas para resolverlas [8]:

- Brevedad de los tweets y nuevas formas de expresarse, para esto se proponen distintas soluciones. Borrar, reemplazar o corregir propiedades específicas de los tweets como pueden ser emoticonos, hashtags, hipervínculos, menciones, acrónimos o jerga propia. O bien añadir estas propiedades como características a la hora de clasificarlos.

- Poca expresividad, los tweets suelen ser neutrales con pocas expresiones relacionadas a la positividad o negatividad que está reflejando. Como resultado se produce un gran desbalanceo a la hora de clasificar. Para solucionarlo se debe expandir el set de entrenamiento utilizado para calibrar los clasificadores basados en machine learning, ya que cuanto mayor sea el set de entrenamiento mejor podrá clasificar las expresiones relacionadas con la polaridad en los tweets.

- Dependencia temporal de los tweets, estos se suelen clasificar como ítems independientes sin embargo, puede no ser así. Los tweets pueden estar conectados entre sí, formando un texto más complejo. En este caso algunos investigadores han optado por clasificar el *stream* completo en vez de individualmente.

Con esta idea, se han diseñado distintas soluciones comerciales o académicas, ya sea bien empleando machine learning o lexicon. Estas pueden dividirse en aplicaciones de propósito general o bien específico, las cuales han demostrado tener un mejor rendimiento.[8]

System	Academic / Commercial	General-Purpose / Domain-Specific
AiApplied	Commercial	General-Purpose
Anonymous	Commercial	General-Purpose
BPEF [Hassan et al. 2013]	Academic	Domain-Specific
ChatterBox	Commercial	General-Purpose
EWGA [Abbasi et al. 2008]	Academic	Domain-Specific
FRFF [Sharif et al. 2014]	Academic	Domain-Specific
FRN [Abbasi et al. 2011]	Academic	Domain-Specific
GU-MLT-LT [Gunther and Furrer 2013]	Academic	Domain-Specific
Intridea	Commercial	General-Purpose
KLUE [Proisl et al. 2013]	Academic	Domain-Specific
LightSIDE [Mayfield and Rose 2012] (Version 2.0)	Academic	Domain-Specific
Lymbix	Commercial	General-Purpose
MLAnalyzer	Commercial	General-Purpose
NRC [Mohammad et al. 2013] (Version EmoLex 0.92)	Academic	Domain-Specific
OpinionFinder [Riloff and Wiebe 2003] (Version 1.5)	Academic	General-Purpose
Repustate	Commercial	General-Purpose
RNTN [Socher et al. 2013] (Version CoreNLP 3.4)	Academic	Domain-Specific
Semantria	Commercial	General-Purpose
Sentiment140 [Go et al. 2009]	Academic	General-Purpose
SentimentAnalyzer	Commercial	General-Purpose
SentiStrength [Thelwall et al. 2010] (Version .NET)	Academic	General-Purpose
SVM Baseline [Pang et al. 2002] (Version RapidMiner 5.3)	Academic	Domain-Specific
TeamX [Miura et al. 2014]	Academic	Domain-Specific
Textalytics	Commercial	General-Purpose
TextProcessing	Commercial	General-Purpose
uClassify	Commercial	General-Purpose
ViralHeat	Commercial	General-Purpose
Webis [Hagen et al. 2015] (Version SemEval 2015)	Academic	Domain-Specific

Figura 2: Muestra de aplicaciones utilizadas en [8]

Entre las aplicaciones que se seleccionaron. Para evaluar estas aplicaciones y comprobar la preción que tienen, se utilizaron tweets anotados manualmente por humanos mediante Amazon Mechanical Turk, evaluados según sean positivos, negativos o neutros. Se emplearon cuatro datasets distintos referidos a los temas: farmacéutico, ventas, seguridad, tecnología o telecomunicaciones. Como conclusión de este estudio, las aplicaciones de propósito específico obtuvieron de media un 67.53 % de aciertos, frente al 56.76 % de precisión que marcaron las aplicaciones de propósito general. Siendo webis la que mejor media obtiene con un 71.41 % de tweets clasificados correctamente por parte de las aplicaciones de dominio específico y ChatterBox con un 67.43 % en las de propósito general.

3 Dispel4Py y código Original

En este capítulo se procede a realizar una introducción sobre la librería Dispel4Py y a introducir el código original presentado junto al artículo "dispel4py: A Python framework for data-intensive scientific computing"[10].

3.1 Dispel4Py

Dispel4Py es una librería desarrollada en Python cuyo objetivo es facilitar el procesamiento intensivo de datos centrándose en cumplir los requerimientos de los usuarios y que estos se puedan centrar en diseñar sus workflows a un nivel abstracto. Describiendo acciones, entradas y salidas necesarias y cómo están conectados. Mediante su sistema de nodos se pueden programar aplicaciones y ejecutarlas de forma escalable en distintas plataformas, estos nodos se interconectan formando un grafo dirigido [4] y pueden adoptar las siguientes formas [10]:

1. Processing element (PE) corresponde a la idea general de un nodo y se pueden extender con múltiples clases, pero la diferencia fundamental es el número de entradas y salidas, desglosandose :

   (a) Generic PE : representa el tipo básico de nodo , tiene un número variable de entradas y salidas.

   (b) IterativePE : cuenta con exactamente una entrada y una salida, se suele usar para encapsular funciones de filtro o transformación.

   (c) ProducerPE : la raíz del árbol con cero entradas y una salida, sirve los datos al resto de nodos.

   (d) ConsumerPE : es un nodo hoja ya que con una entrada y cero salidas representa el final de la computación.

2. Una instancia es una copia ejecutable del PE, cada una de estas previo a la ejecución se traduce en una o más instancias, siendo el último caso aplicado si la misma acción se requiere en más partes del código o también

[4] https://es.wikipedia.org/wiki/Grafo_dirigido

podría ser necesario para paralelizar e incrementar el procesamiento de datos.

3. Al ser un sistema basado en streaming, cuando el buffer está lleno, los procesos generadores se paran hasta que todos los datos hayan sido consumidos.

4. Cuenta con dos modos de ejecución: Sistema secuencial, se ejecuta en un solo proceso sin optimizar por lo que se procesan los métodos de uno en uno, es un sistema útil para depurar . Multiproceso , para implementar este método se emplea la librería de Python ya que crea un pool de procesos y asigna cada uno de estos a una instancia de un PE.

Para poner a prueba esta librería se diseñaron tres tipos de aplicaciones para distintos campos :

- Sismología : realiza la correlación cruzada entre el ruido sísmico ambiental (Las primeras fases están desarrolladas con Dispel4py. La primera fase procesa las series temporales contínuas dada una estación sísmica y al ser independiente se puede realizar de forma paralela. En la fase 2 empareja todas las estaciones y calcula la correlación cruzada de cada par)

- Astrofísica : calcula la extinción interestelar interna de las galaxias (coeficiente de corrección necesario para calcular la luminosidad óptica de una galaxia) del catálogo AMIGA. Lee un fichero que contiene los valores de ascensión recta (AR) y declinación (Dec) de 1051 galaxias. A continuación consulta un servicio del observatorio virtual para cada galaxia usando sus valores AR y Dec, estos resultados se filtran y seleccionan solo un valor que corresponde al tipo morfológico y aplanado.

- Análisis de sentimientos : Analiza tweets para determinar si la opinión que se muestra en estos es positiva o negativa y los clasifica según el estado de EEUU al que pertenezcan, clasificando finalmente estos estados según la puntuación total que obtengan.
  La cantidad y variedad de datos es enorme y está en constante crecimiento por lo que una librería como dispel4py es muy útil para tratar tal cantidad de datos al tener implementado *streaming* y procesamiento intensivo de datos, encaja a la perfección con el modelo de Twitter.

Por lo que queda patente que esta librería puede usarse en campos tan dispares que van desde la geología a la lingüistica, por tanto demuestra su facilidad para implementarse. Aunque este proyecto se centrará en la aplicación para el análisis de sentimientos.

3.2 Código original

El código implementado con Dispel4py corresponde a una modificación de una aplicación para el análisis de sentimientos [5], la cual se ha adaptado para ejecutarse usando la librería y el resultado adopta la siguiente forma [10] :

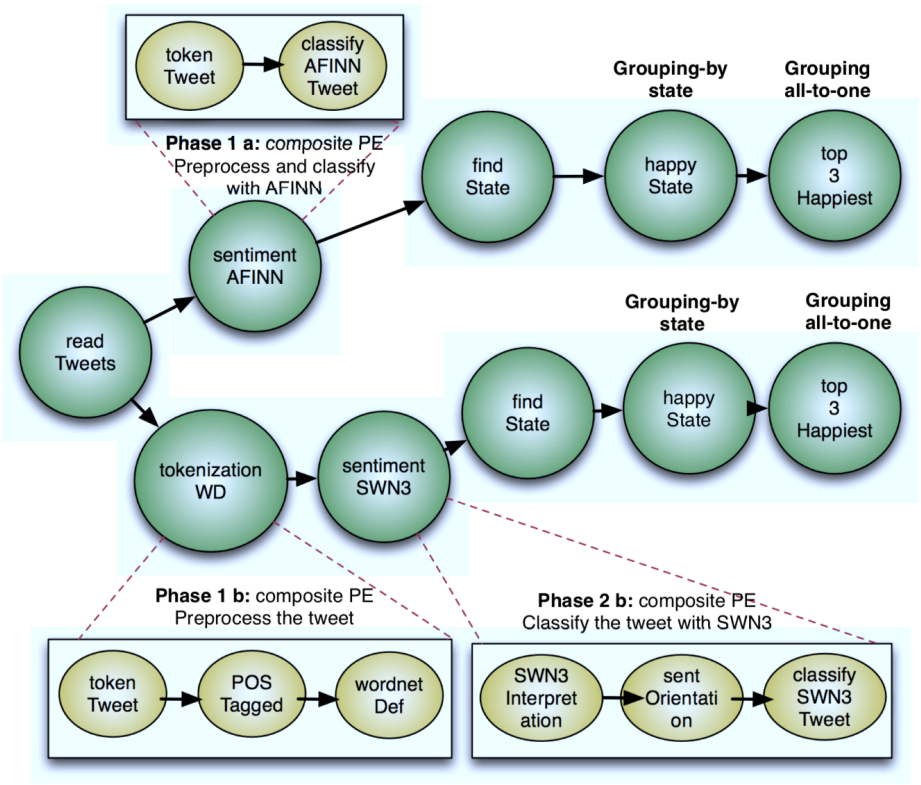

Figura 3: Distribución de la aplicación

El grafo mostrado para determinar la estructura de la aplicación tiene la siguiente forma traducido a código:

[5] https://github.com/linkTDP/BigDataAnalysis_TweetSentiment

```
ROOT_DIR="./"
tweets= ReadData()
tweets.name='read'
sentiment_afinn= AFINNSentimeScore("AFINN-111.txt")
findstate1=FindState()
findstate2=FindState()
happystate1=HappyState()
happystate2=HappyState()
findhappystate1 = GlobalHappyState()
findhappystate2 = GlobalHappyState()

preprocess_sentiword=Tokenization_WD()
sentiment_sentiword=SentiWordNetScore("SentiWordNet_3.0.0_20130122.txt")
sentiwordscore=ComputeSentiWordNetScore()
```

Grafo dirigido para el método AFINN :

```
graph = WorkflowGraph()
graph.connect(tweets, 'output', sentiment_afinn, 'input')
graph.connect(sentiment_afinn, 'output', findstate1, 'input')
graph.connect(findstate1, 'output', happystate1, 'input')
graph.connect(happystate1, 'output', findhappystate1, 'input')
```

Grafo dirigido para SWN-3:

```
graph.connect(tweets, 'output', preprocess_sentiword, 'input')
graph.connect(preprocess_sentiword,'output', sentiment_sentiword,
    'input')
graph.connect(sentiment_sentiword, 'output',sentiwordscore , 'input')
graph.connect(sentiwordscore , 'output', findstate2, 'input')
graph.connect(findstate2, 'output', happystate2, 'input')
graph.connect(happystate2, 'output', findhappystate2, 'input')
```

A continuación se describen las siguientes etapas correspondientes a los métodos que interconectan el grafo :

1. Lectura.

   En este apartado se utiliza un nodo Genérico en este caso la entrada es el

nombre del fichero y la salida es un diccionario conformado por el texto contenido en el tweet, las coordenadas, el lugar y la localización.

2. Clasificación.

   En esta función se ramifica en dos , una rama siguiendo el procesamiento de AFINN-111 y la otra siguiendo SWN-3.

   - AFINN-111.

     Para realizar la clasificación se emplea un nodo iterativo, la entrada conecta con el método de lectura por lo que recoge el diccionario, procede a *tokenizar* las palabras del tweet, quitar todos los caracteres innecesarios y clasificar cada palabra con el diccionario AFINN-111, una vez clasificadas todas las palabras, calcula la puntuación media y pasa esta información al siguiente método.

   - SWN-3.
     Este nodo es algo más complejo que el de la otra rama, ya que se necesita clasificar las distintas palabras según su clase (sustantivo, verbo, adjetivo y advervio). Desglosando las siguientes funciones principales :

     – tag_tweet : tokeniza las frases , dentro de cada una de estas elimina todos los signos de puntuación y procede a clasificar cada palabra en las categorías anteriormente enunciadas.

     – wordnet_definitions : procede a la lematización de la palabra donde a partir de una forma flexionada se halla el lema (ej. decir es el lema de diré), una vez completo , asigna un tipo de palabra.

     Dentro de este último se emplea el análisis de cada palabra por lo tanto se tiene que escoger un significado para esta, con *word_sense disambiguate* se proporciona una lista de sinónimos dada su clase (anillo de sinónimos), con la cual se pretende desambiguar el significado cruzando la frecuencia de cada una de las palabras con el

contexto y quedándose con el que mejor se ajuste.

Si la palabra tiene un significado válido y está contenido en la lista de SWN-3 se le asigna tres calificaciones distintas positiva, negativa y objetiva , si no es válido, se intenta con sus sinónimos, por último si no se encuentra ninguna puntuación, se desecha.

Con estos resultados se pasa al siguiente nodo *ComputeSentiWordNetScore* en el cuál se procede a hallar la media de las puntuaciones negativas y positivas, y se asigna como definitiva la que mayor valor obtenga.

3. Delimitar localización.

   Los métodos para determinar si un tweet se encuentra en una localización válida son iguales para ambas técnicas.

   Para empezar a clasificar, se procede a hallar si el tweet procesado pertenece a la región sobre la que se quiere realizar el estudio, esta información se intenta averiguar de dos formas :

   - coordenadas : intenta precisar a través de estas si la localización está dentro de la zona determinada haciendo uso del método *coord2state*.
   - Lugar o localización : si tiene un sitio asignado, busca si está en la lista de estados proveídos .

4. Puntuación final.
   Una vez conseguidos los tweets que están dentro de la región de interés , se procede a encontrar los estados con mejor puntuación en los nodos HappyState y GlobalHappyState.

   - HappyState: en este método cada vez que se procesa un tweet se procede a actualizar la puntuación que tiene el estado al que pertenece, si esta puntuación excede a la máxima previamente calculada para ese estado, se manda al siguiente método.
   - GlobalHappyState: cada vez que la puntuación máxima de un estado es actualizada, se recalcula el top y se muestra el resultado.

Terminada la ejecución, la última salida muestra el resultado con el top tres que más puntuación ha obtenido.

3.3 Support Vector Machine aplicado a Twitter

Ya que todas las técnicas implementadas con Dispel4py son basadas en lexicon, se propone en este TFG implementar una solución basada en Support Vector Machine, que como se explcó en el anterior capítulo es una de las aproximaciones que se puede utilizar en este contexto. Al utilizar entrenamiento, con esta aproximación se alcanzan niveles de precisión mucho más altos que los lexicon. Por ejemplo, empleando Support Vector Machine, a partir de 2000 reviews de películas alcanzó un 85.1% de aciertos. Gran parte de los clasificadores realizados con machine learning están basados en esta técnica [16], por lo que, para comprobar su efectividad se procede a implementarlo utilizando la librería sklearn [6].

Adaptándolo en lo posible a Dispel4py, empleando un nodo genérico para realizar la lectura del dataset de entrenamiento y un nodo consumidor donde se realiza la clasificación. Reutilizando los nodos para calcular el top, clasificación por estados y la lectura inicial:

```
tweets = ReadData()
classifier = SVM()

graph.connect(tweets, 'output', findstate1, 'input')
graph.connect(findstate1, 'output', tweets2, 'input')
graph.connect(tweets2, 'output', classifier, 'input')
graph.connect(classifier, 'output', happystate1, 'input')
graph.connect(happystate1, 'output', findhappystate1, 'input')
```

Para entrenar el clasificador se utiliza :

- *Dataset*, para poder entrenar el clasificador es necesario utilizar uno. El elegido es el "sentiment-140 dataset", el cual cuenta con 1.600.000 tweets preprocesados y clasificados con 0 si es negativo o con 1 si es positivo. Está desarrollado utilizando un clasificador de máxima verosimilitud [14].

[6] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm

- Los tweets se vectorizan siguiendo el método TFIDF (term frequency - inverse document frequency), el cual mide la frecuencia del término dentro del documento, por lo que su valor incrementa proporcionalmente a estos, con lo que se puede determinar qué palabras aparecen más frecuentemente. Este método es muy utilizado ya que más del 83 % de los sistemas de recomendación basado en texto lo usan [15]. Para implementar el vectorizador se utiliza el método *TfidVectorizer*, con la configuración :

```
vectorizer = TfidfVectorizer(min_df = 5,
                             max_df = 0.8,
                             sublinear_tf = True,
                             use_idf = True)
```

  Con **min_df** se determina que ignore los términos que aparezcan en menos de 5 documentos, y con **max_df** los términos que aparezcan en más del 80 % de los documentos. Con **use_idf** a true los términos que aparezcan demasiado son ponderados de forma que obtengan menos puntuación ya que estos pueden ser un mal indicador.

- Una vez obtenido el vectorizador se le aplica a los sets de entrenamiento y de test :

```
train_vectors = vectorizer.fit_transform(train['SentimentText'])
test_vectors = vectorizer.transform(test['SentimentText'])
```

- A continuación, se construye el clasificador, y se entrena con el vector que contiene el 80 % de los datos :

```
classifier_svm = svm.SVC(kernel='linear')

classifier_svm.fit(train_vectors, train['Sentiment'])

prediction = classifier_svm.predict(test_vectors)

report = classification_report(test['Sentiment'], prediction,
    output_dict=True)
```

Con *report* se obtiene un resumen de los resultados.

```
'neg ', {'recall': 0.7285085965613755, 'f1-score':
   0.7439011942431356, 'support': 5002, 'precision':
   0.7599582898852972}, ' pos :', {'recall':
   0.7697078831532613, 'f1-score': 0.754091933744977,
   'support': 4998, 'precision': 0.7390970220941403}
```

Se observa que ha clasificado para un 72 % de los datos negativos, con un 74.3 % de precisión, y las opiniones positivas para un 75 % de los datos, con un 73.9 % de precisión. Cuenta con 5002 muestras positivas y con 4998 muestras negativas. Frente al 56 % que se obtiene con AFINN o al 63 % con SWN-3, los cuáles son resultados bastante buenos teniendo en cuenta que no son métodos supervisados.

4 Intel Threading Building Blocks

El paralelismo es la capacidad de procesar múltiples instrucciones simultaneamente con el afán de reducir el tiempo de computación y mejorar el rendimiento por ejemplo haciendo uso de los procesadores multinúcleo. Para aplicar paralelización se va a emplear la librería threading building blocks de C++.
Una de las principales optimizaciones de la aplicación descrita en 3.2 se basa en la paralelización de la misma para aprovechar los recursos disponibles que nos ofrecen los procesadores multinúcleo.

4.1 ¿Qué es TBB?

Intel Threading Building Blocks es una librería lanzada en 2006 por primera vez, un año después del lanzamiento del primer procesador multinúcleo de arquitectura x86 de Intel, el Pentium D[7], además en esta época C++ no contaba con soporte para programación paralela, facilitando así la implementación del paralelismo en este. Actualmente cuenta con la librería STL, aún así TBB posee una abstracción de alto nivel que facilita mucho su uso.

Una característica a destacar es que TBB no garantiza la ejecución paralela (en la ejecución puede participar un subconjunto de threads, todos o únicamente el master) solo comunica que está permitida.

Es una librería que provee plantillas a alto nivel, las cuales sirven para casos más generales, pero también provee interfaces a bajo nivel para crear soluciones más específicas . Estos dos modos no son independientes ya que pueden mezclarse, por ejemplo se pueden crear aplicaciones que tienen Flow Graph a alto nivel con nodos que usan paralelismo genérico. También la implementación puede realizarse de forma progresiva ya que se pueden localizar las partes del código que necesiten paralelización, realizarla, y posteriormente añadir más paralelizaciones[19].

[7] https://es.wikipedia.org/wiki/Pentium_D

4.2 Algoritmos Genéricos

TBB ofrece una serie de algoritmos que pueden usarse en casos generales, por ejemplo :

- parallel_for ejecuta paralelamente el cuerpo de la función sobre un cada elemento dentro de un rango. Si el número de pasos a aumentar no está determinado se toma 1.
- parallel_do procesa los distintos elementos en un contenedor en paralelo, con la capacidad de añadir elementos dinamicamente, este termina cuando la función principal devuelve todos los items que había en la secuencia de entrada.
- parallel_invoke evalúa $f_1(), f_2()...f_n()$ de forma paralela, los argumentos pueden ser o bien funciones, expresiones lambda o punteros a funciones. El número de parámetros es mínimo dos pero no tiene límite gracias a C++11.
- parallel_reduce genera de forma dinamica conjuntos para reducir en paralelo.Tiene dos versiones, determinista (algo más lenta pero tiene los resultados esperados y es más fácil debuggear) y no determinista. Cada una de estas tiene dos plantillas. Funcional, es fácil de usar junto a las lambda expresiones e imperativa la cual minimiza el número de datos copiados.
- parallel_sort ordena una estructura de datos de forma paralela, pero no es ni estable ni determinista, es decir no garantiza que la siguiente ejecución repita el mismo resultado.

Pero no todos los algoritmos encajan en uno de estos patrones, por lo que para este tipo de aplicaciones conviene usar flow graphs los cuales permiten expresar programas a través de grafos. Suele usarse para aplicaciones con *stream* de datos los cuales van pasando una serie de filtros.Parallel_do tiene la misma funcionalidad a excepción de que solo tiene una función a aplicar a cada item.

4.3 Flow Graph

Para poder usar flow graph primero hay que crear un objeto *grafo*, dentro de este se crean los nodos que operan entre ellos, para que puedan comunicarse las aristas detallan estos canales de comunicación y las dependencias entre los nodos, unificando el grafo ya que cuando un nodo es creado es un grafo independiente. Para crear un grafo se siguen cinco pasos fundamentales :

- Crear el objeto grafo, este se utiliza para llamar a sus operaciones tales como esperar hasta la completación de todas las tareas, resetear los estados de los nodos o cancelar su ejecución.

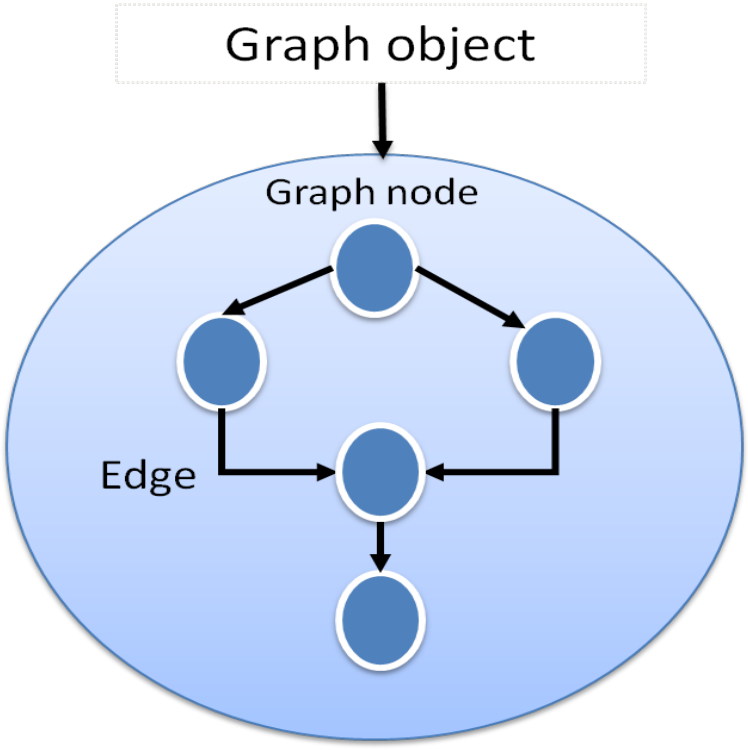

Figura 4: Ejemplo estructura de un grafo [20]

- Crear los nodos, estos se pueden dividir en tres grupos fundamentales functional nodes (computa los datos de entrada y manda el resultado o señal a sus sucesores. Toma como entrada una función lambda), control flow nodes y buffering nodes (están diseñados para acumular mensajes de entrada y mandarlos a los sucesores).

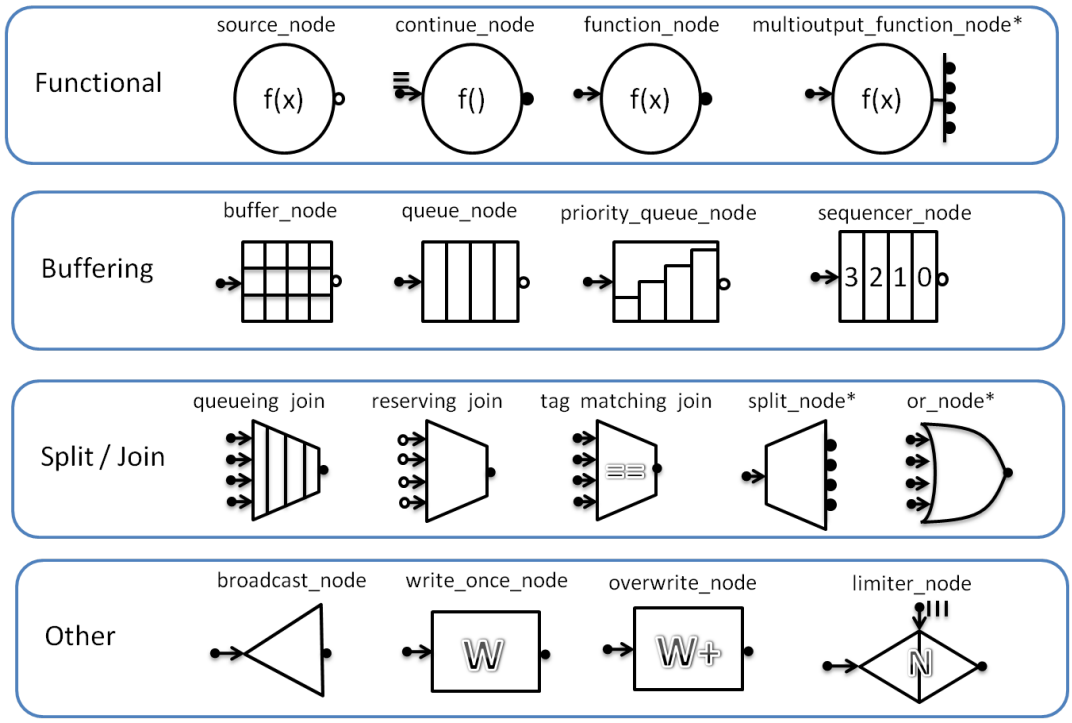

Figura 5: Tipos de nodos [20]

- Añadir aristas, para construir el grafo es necesario unir los distintos nodos que lo conforman. Al tratarse de un grafo dirigido para conectarlos constan de un *input_port* y *output_port*.
- Ejecutar el grafo, hay dos formas de comenzar la ejecución, o bien a través de un try_put a un nodo o a partir de la salida de un nodo fuente.
- Ejecución completa, el grafo espera a que termine la ejecución con la instrucción wait_for_all(). Mostrando el resultado final.

4.4 Estructuras de datos

Para poder realizar programación paralela es necesario estructuras de datos concurrentes:

Class name *and C++11 connection notes*	Concurrent traversal and insertion.	Keys have a value associated with them.	Support concurrent erasure	Built-in locking.	No visible locking (lock-free interface).	Identical items allowed to be inserted.	[ ] and at accessors
`concurrent_hash_map` *Predates C++11.*	✓	✓	✓	✓	✗	✗	✗
`concurrent_unordered_map` *Closely resembles the C++11 `unordered_map`.*	✓	✓	✗	✗	✓	✗	✓
`concurrent_unordered_multimap` *Closely resembles the C++11 `unordered_multimap`.*	✓	✓	✗	✗	✓	✓	✗
`concurrent_unordered_set` *Closely resembles the C++11 `unordered_set`.*	✓	✗	✗	✗	✓	✗	✗
`concurrent_unordered_multiset` *Closely resembles the C++11 `unordered_multiset`.*	✓	✗	✗	✗	✓	✓	✗

Figura 6: Propiedades de las principales estructuras [19]

- Concurrent vector: es un array que crece de forma dinámica, y es seguro ampliar el vector incluso cuando el resto de hebras están operando sobre él, para realizar esta operación se vale de tres métodos fundamentales : push_back, grow_by, grow_to_at_least. Para garantizar que es seguro, nunca mueve elementos cuando está aumentando, y para ello el contenedor asigna un número de arrays contínuos. El número de reservas, crecimiento o asignaciones determina el tamaño del primer array por lo que utilizar un número de elementos pequeño al inicio puede provocar fragmentación a medida de que el vector va creciendo. Para solucionar esto shrink_to_fit une los distintos arrays en un solo array contínuo lo cual puede mejorar el tiempo de acceso.

- Unordered map: son contenedores asociativos, los cuales tienen pares clave (únicas) - valor y permite inserción e iteración concurrente aunque no permite borrar concurrentemente,pero sí puede realizar accesos directos a posición, lo que permite acceder a un valor por su clave.

- Unordered set: son contenedores que permiten guardar elementos únicos sin orden en particular, solo permite las funciones para iterar e insertar sobre la estructura de forma concurrente.

- Concurrent hash map: mapea las claves y los valores de una forma que le permite acceder a valores con métodos para encontrar, insertar y borrar. Las claves están desordenadas, actúa como contenedor para elementos de tipo *std::pair<const Key, T>*, y permite tanto leerlos como actualizarlos concurrentemente.

- Concurrent queues: son estructuras de datos donde los items se añaden o eliminan de la cola con operaciones como push y pop. En este caso se utiliza un "try pop" el cual dictamina si la cola está vacía por lo que ningún valor puede extraerse de la cola, es un método útil para evitar tener que bloquear la cola y comprobar si está vacía, algo que no sería seguro concurrentemente y perjudicaría la ejecución.

5 Optimización

En este capítulo se detallan las distintas opciones que se han seguido para realizar la optimización del código.

5.1 Reorganización

Como se muestra en la sección 3.2, el código sigue una estructura *lectura-clasificación-filtrado-resultado*, la cual no es una secuencia óptima ya que se filtra después de clasificar, por lo que una gran cantidad de datos han sido clasificados innecesariamente, lo que conlleva una pérdida de tiempo.

Por lo tanto la siguiente propuesta es reorganizar la estructura de la siguiente forma:

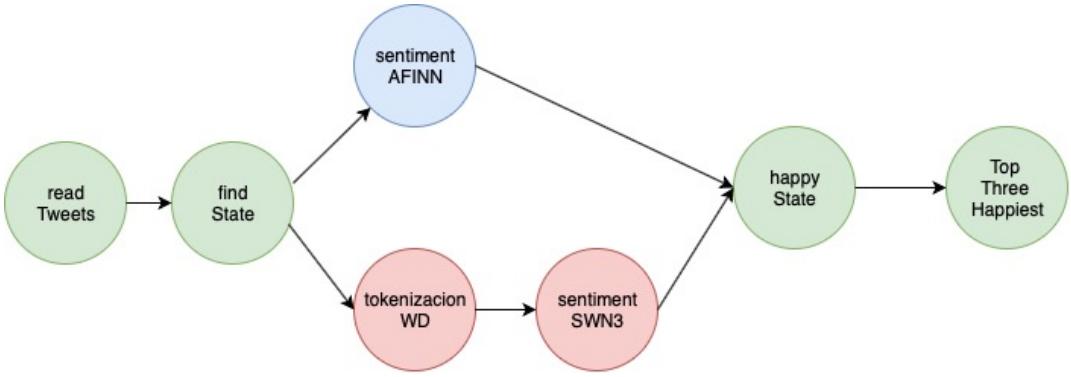

Figura 7: Nodos reorganizados

Con esta nueva estructura se procedería a realizar primero el filtro para posteriormente realizar la clasificación de los tweets válidos.

Para poder implementar esta reorganización, las entradas y salidas de los nodos han sufrido ligeras modificaciones. Y para obtener aún más eficiencia en la lectura también se descartan los tweets de los cuáles no se pueda obtener una localización.

Obteniendo finalmente los grafos de conexión para:
AFINN

```
graph = WorkflowGraph()
graph.connect(tweets, 'output', findstate1, 'input')
graph.connect(findstate1, 'output', sentiment_afinn, 'input')
graph.connect(sentiment_afinn, 'output', happystate1, 'input')
graph.connect(happystate1, 'output', findhappystate1, 'input')
```

SWN-3

```
graph.connect(findstate1,'output', preprocess_sentiword, 'input')
graph.connect(preprocess_sentiword, 'output', sentiment_sentiword,
    'input')
graph.connect(sentiment_sentiword, 'output', sentiwordscore, 'input')
graph.connect(sentiwordscore, 'output', happystate2, 'input')
graph.connect(happystate2, 'output', findhappystate2, 'input')
```

5.2 Threading Building Blocks

Para realizar esta optimización se elige implementar una de las ramas en C++ para aumentar su rendimiento. Siendo AFINN la elegida ya que la rama que implementa SWN-3 está realizada íntegramente con Wordnet [8] el cual proviene del paquete nltk [9].

Tras las modificaciones para integrar la computación AFINN en C++ el grafo para este quedaría de la forma:

Figura 8: Nodos AFINN

Por lo que los pasos de lectura y filtrado por estados permanecen sin

[8] https://www.nltk.org/howto/wordnet.html
[9] https://www.nltk.org

modificación. Pero a la hora de clasificarlos se pasaría a c++ el cual está conectado con python a través de cython, con la siguiente llamada en el método afinn de python:

```
class AFINNSentimeScore(ConsumerPE):
    def __init__(self, sentimentData):
        ConsumerPE.__init__(self)
        afinnfile = open(ROOT_DIR + sentimentData)
        self.sentiment= {}
        for line in afinnfile:
            term, score = line.split("\t")
            self.sentiment[term] = float(score)



    def _process(self, data):
        csolution(data,self.sentiment)
```

El método csolution tiene como entrada una lista de tweets y su correspondiente estado, el diccionario correspondiente a la clasificación AFINN. Los cuales traducidos a C++ con cython corresponden a un vector de pares (*string,string*) y un map (*string,int*) respectivamente. Con estos parámentros se llama a la función en C++.

```
from libcpp.string cimport string
from libcpp.pair cimport pair
from libcpp.map cimport map
from libcpp.vector cimport vector

cdef extern from "map_reduce.h":
    void _csolution "allways"(vector[pair[string,string]] tweet,
        map[string,int])
def csolution(list s, dict d):
    cdef vector[pair[string,string]] ss = s
    cdef map[string,int] sc = d
    _csolution(ss,sc)
```

Para asegurarnos de que la paralelización se está ejecutando con el máximo número de *threads*, se indica como parámetro para inicializar el planificador.

```
int n = tbb::task_scheduler_init::default_num_threads();
tbb::task_scheduler_init init(n);
```

Siguiendo el mismo esquema que en Python, el estado junto a la puntuación se almacena en un vector concurrente.

```
tbb::concurrent_vector<std::pair<double,std::string>> v;
```

Los items del diccionario se procesan con un parallel_for, a cada uno de estos primero se le aplica un regex

```
std::regex r("[^a-zA-z ]+");
```

para dejar el texto sin signos de puntuación, a continuación se tokeniza el tweet con el siguiente método, en el que cada una de las palabras del string se guardan en un vector

```
std::vector<std::string> tokenize(const std::string& input) {
  std::stringstream ss(input);
  std::vector<std::string> output;
  std::string item;

  while(std::getline(ss, item,' ')) {
    output.push_back(item);
  }
  return output;
}
```

Con *parallel_for* se paraleliza que a cada uno de los items del nuevo vector se busque si está en el diccionario AFINN y sume su puntuación. Cuando finaliza realiza la media y se almacena en otro vector, concurrentemente.

```
std::smatch m;
int l = state.size();
```

```
tbb::parallel_for(0,l,[&](int i){
     int res = 0 ;
     int count = 0;
     std::string result = regex_replace(state[i].first, r,"");
     std::vector<std::string> results = tokenize(result);
     for(int i = 0 ; i < results.size();i++){
        auto found = pru.find(results[i]);

       if(found != pru.end() ){
         res += found->second;
         count ++;
       }
     }
     if(count != 0 ) res = res/count;
     v.push_back(std::make_pair(res,state[i].second));
   });
```

A continuación se procede a calcular la puntuación de todos los estados. Recorriendo el vector con un *parallel_for* de forma paralela, pero en esta ocasión el resultado se guarda en un *concurrent_hash_map* ya que permite borrar e insertar concurrentemente. Necesario para realizar la actualización de la puntuación.

```
    l = v.size();
    tbb::concurrent_hash_map< std::string,double> final ;
    tbb::parallel_for(0,l,[&](int i){

       tbb::concurrent_hash_map< std::string, double>::accessor a ;
       final.insert(a,v[i].second);
       a->second += v[i].first;

    });
   for (auto& it: final) {
    std::cout << it.first << " con: " << it.second << std::endl;
   }
}
```

Ya que el módulo debe poder usarse en python se realiza el siguiente archivo de configuración para proceder a la compilación de la función Cython y la función en C++ .

```
from distutils.core import setup, Extension
from Cython.Build import cythonize
from Cython.Distutils import build_ext
# First create an Extension object with the appropriate name and
    sources
ext = Extension(name="cppsol",
                sources=["map_red.pyx", "map_reduce.cpp"],
                language="c++",
                extra_compile_args=["-std=c++11" ],
                extra_link_args=["-std=c++11", "-ltbb"])
```

```
setup(name="cppsol",
      ext_modules=cythonize(ext))
```

Finalmente la única instrucción que realiza el nodo en python es llamar a la función con el diccionario AFINN y los tweets.

```
class AFINNSentimeScore(ConsumerPE):
    def __init__(self, sentimentData):
        ConsumerPE.__init__(self)
        afinnfile = open(ROOT_DIR + sentimentData)
        self.sentiment= {}
        for line in afinnfile:
            term, score = line.split("\t")
            self.sentiment[term] = float(score)

    def _process(self, data):
        csolution(data,self.sentiment)
```

6 Resultados, conclusiones y futuras aplicaciones

6.1 Resultados

El entorno en el que se han desarrollado las pruebas posee las siguientes características:

Procesador	Nº de procesadores	Nº de cores	Caché de nivel 2	Caché de nivel 3	Memoria RAM
Intel Core i5 2,3 GHz LPDDR3	1	4	256KB	6MB	8GB

El sistema operativo utilizado es Mojave versión 10.14.6. Compilando con C++14 y Python 2.7 (la implementación del código original está realizada en este, aunque esté obsoleto).
Por lo que para ejecutar se utiliza los comandos:

```
python -m dispel4py simple nombre_archivo.py -d '{"read" : [
    {"input" : "nombre_archivo.json"} ]}'
```

Para obtener los tiempos totales de la ejecución se coloca el comando *time* al comienzo:

```
time python -m dispel4py simple nombre_archivo.py -d '{"read" : [
    {"input" : "nombre_archivo.json"} ]}'
```

A partir de esta configuración, se realizan 10 pruebas para cada uno de los tamaños de entrada con los 3 códigos diferentes: *Original*, *Reorganizado* y *Paralelizado*; siendo este último la versión final. *Original* representa la versión inicial de la aplicación en Dispel4Py, *Reorganizado* corresponde a la versión de código optimizada que se explica en la sección 5.1, mientras que *Paralelizado* se explica en 5.2. Para esta segunda versión se muestran los tiempos de ejecución cuando se utilizan todos los cores del procesador y se habilita hyperthreading (2 threads hardware por core). El código paralelizado se puede ejecutar por lo tanto hasta con 8 *threads*. Para calcular los tiempos se realiza la media aritmética, descartando los valores atípicos si ocurren. A continuación se muestran los

resultados globales:

Tamaño de entrada	***Original***	***Reorganizado***	***Paralelizado***	**Ratio 1**	**Ratio 2**
10.000	65.32s	12.04s	10.02s	5.42	6.52
20.000	117.95s	14.10s	12.52s	8.36	9.42
50.000	331.88s	29.59s	26.10s	11.22	12.71
100.000	600.03s	62.10s	51.19s	9.66	11.72

Tabla 1: Tiempo medio de ejecución en segundos para cada implementación y ratios de mejora: Ratio 1 representa la mejora de *Reorganizado*, y Ratio 2 la mejora de *Paralelizado*, con respecto a *Original*

Como se ve en la Tabla 1, se observa una tendencia creciente en las mejoras cuando aumenta el tamaño del problema. La reorganización consigue mejoras substanciales (hasta 11x), que se ven ampliadas hasta en un 21 % gracias a la paralelización.

Desglosando los resultados en más detalle, en la Figura 9 se muestran los tiempos correspondientes el código *Original* y a la versión *Reorganizado*.

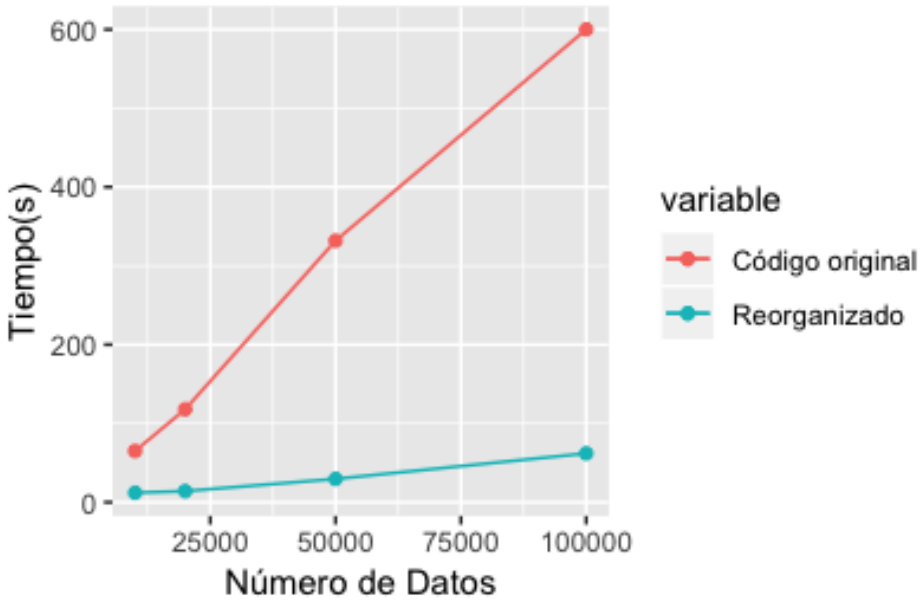

Figura 9: Comparación código *Original* vs. código *Reorganizado*

El tiempo de ejecución en el código original alcanza los 600 segundos, mientras que en el código reorganizado apenas llega a los 60 segundos. Por lo tanto, se puede inferir que la mayoría de tweets estaban siendo clasificados sin utilizarse posteriormente, como se había supuesto al realizar la optimización. Se observa que el incremento del tiempo sigue una tendencia lineal con el tamaño del problema, pero el factor de crecimiento es mayor en el código *Original*.

Prosiguiendo con la optimización realizada con TBB, la paralelización, en la Figura 10 se compara el código *Reorganizado* con el *Paralelizado*. No se realiza la comparación con la invocación multiproceso del código original en dispel4Py ya que en este caso, la ejecución presenta errores.

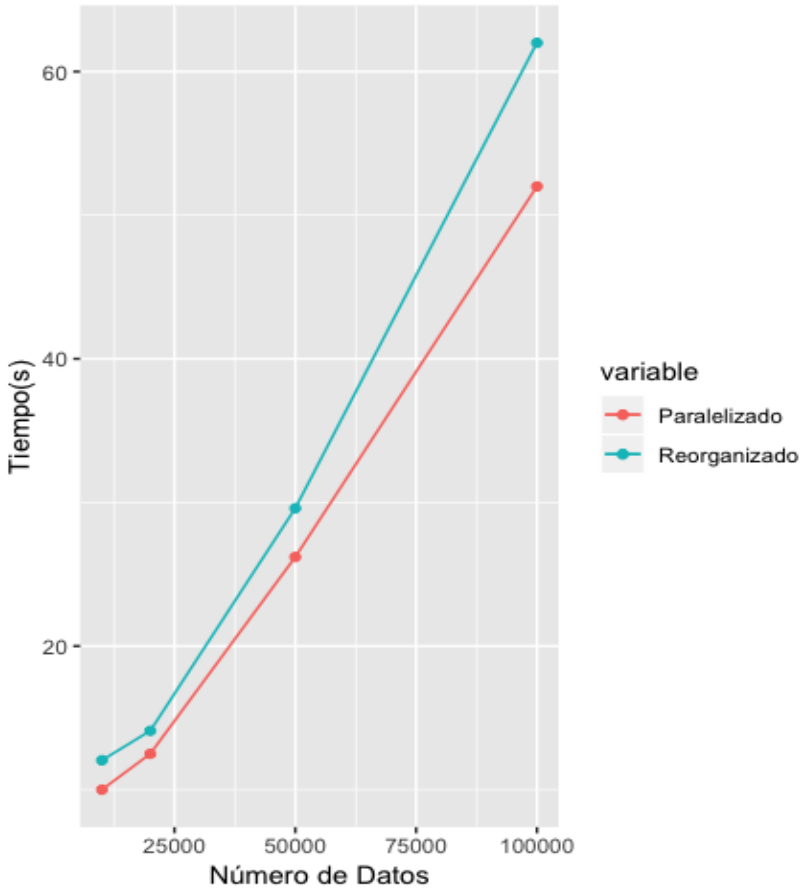

Figura 10: Comparación código *Reorganizado* vs. código *Paralelizado*

Conforme aumenta el tamaño de los datos de entrada, la paralelización muestra mayor eficiencia. Ya que está realizada sobre la computación del algoritmo

AFINN, cuántos más datos se puedan procesar en paralelilo, más significativa será la diferencia.

La Figura 11 visualiza la escalabilidad para cada invocación de la función AFINN (que es la que se ha paralelizado) con respecto al número de *threads*, para el mayor tamaño de entrada disponible. En este caso, se mide el tiempo dentro de la aplicación con la función `tick_count` de TBB.

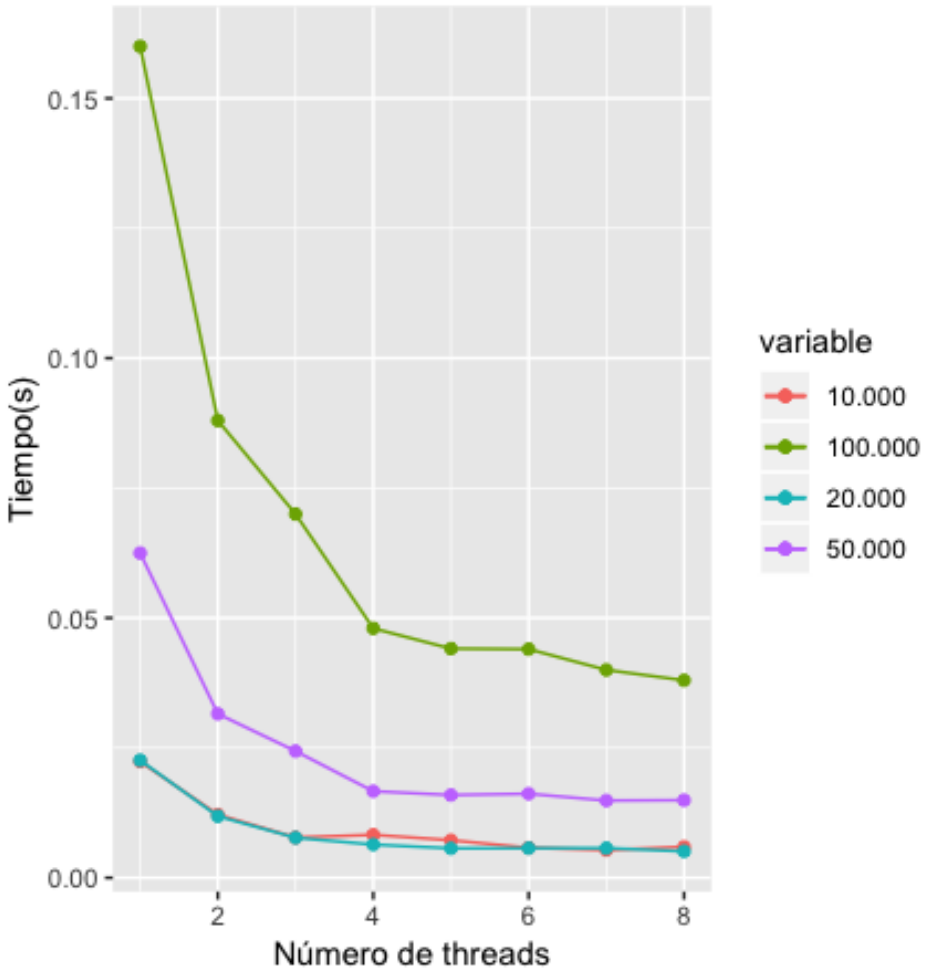

Figura 11: Tiempo según el número de threads para el nodo AFINN

Cuánto mayor es el tamaño de los datos de entrada, mejor escala la función paralela. Como podemos ver, el hyperthreading (número de threads por encima de 4) supone una mejora pequeña cuando el tamaño de datos es lo suficientemente grande y tiene un impacto marginal para tamaños medianos o pequeños.

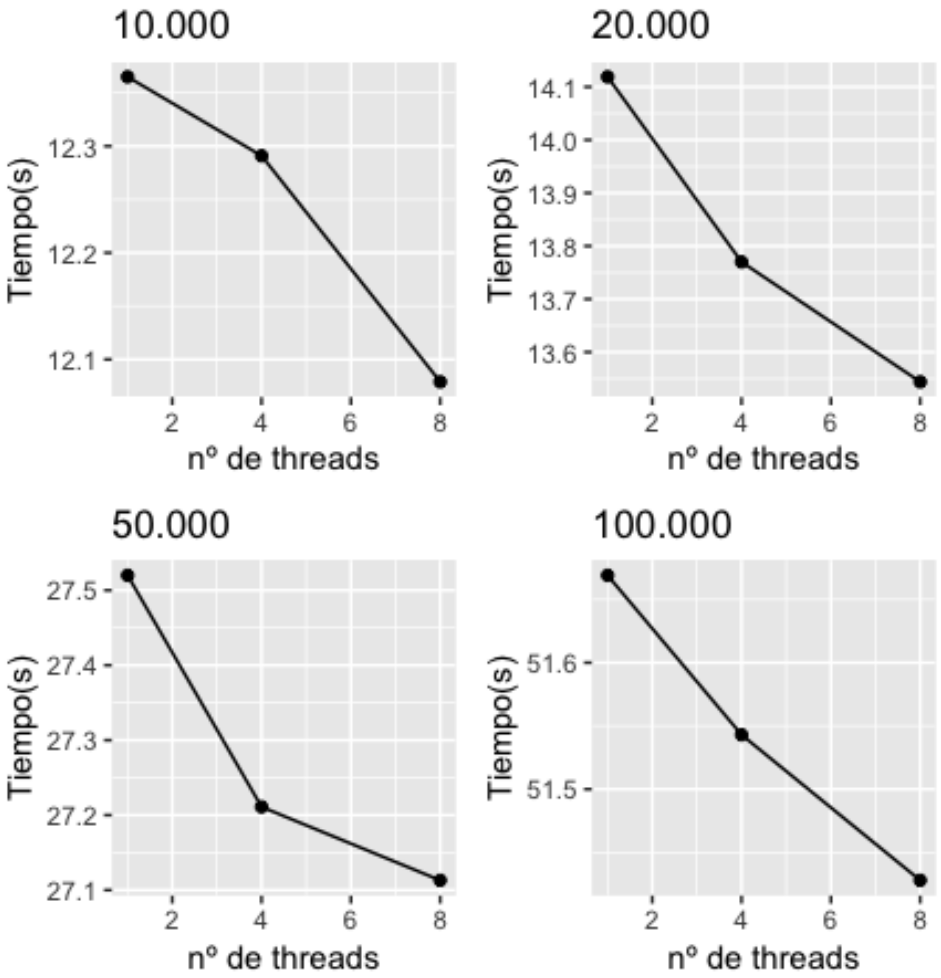

Figura 12: Tiempo según el número de threads para la aplicación completa y en función del tamaño de los datos de entrada

Finalmente, en la Figura 12 mostramos los tiempos y la escalabilidad en función del número de threads para la ejecución de toda la aplicación, y para cada uno de los tamaños de los datos de entrada. De nuevo se observa una mejora, en este caso más pequeña, cuando incrementan el número de threads empleados. Esto se debe a que la parte paralelizada solo supone el 24 % del tiempo de ejecución, por lo que la aceleración máxima que se puede obtener está acotada por la ley de Amhdahl idealmente al 31 %, cercana al 21 % (Ratio 2/ Ratio 1) que obtenemos para el mayor tamaño de entrada.

6.2 Conclusión

En este trabajo se han tratado múltiples temas, comenzando por el análisis de sentimientos y el uso de la informática aplicado a la lingüística. Cómo ha ido evolucionando a lo largo del tiempo y su incremento de uso con el paso de este. Para ello se ha realizado una introducción a algunas de las técnicas actuales que se utilizan en este dominio de aplicación. En este TFG nos centramos en el caso específico de Twitter. Proponemos una implementación de un Support Vector Machine para el análisis de sentimiento en Twitter empleando un dataset de tweets utilizando Python con la librería sklearn, con el cual se consigue un buen rendimiento en la clasificación: hasta un 74 % de precisión.

La aplicación de análisis de sentimientos que usamos como referencia proviene del framework de investigación Dispel4Py, desarrollado por la Universidad de Edimburgo que tiene como uno de sus objetivos dar soporte de streaming a los programadores de aplicaciones en Python. En este TFG optimizamos dicha aplicación Python. Para optimizarla se han seguido distintas fases. Comenzando por la reorganización del código para que no se traten datos de manera redundante, prosiguiendo por la paralelización del método AFINN, que es el de mayor carga computacional. Para ello utilizamos la librería de paralelización TBB de C++, junto con Cython para invocar C++ desde Python. Con ello consiguimos una primera mejora de hasta 11x, al pasar de 600s a 60s el código reorganizado, y adicionalmente una aceleración de 1.2x (60s a 51s) con la paralelización.

6.3 Futuras aplicaciones

Este proyecto podría extenderse implementándose por completo la rama AFINN con FlowGraph de TBB, para aumentar la cobertura del código paralelizado y por lo tanto el factor de mejora que se conseguiría con la aceleración. Otra extensión del proyecto consisitiría en añadir una nueva rama con el modo SVM descrito en 3.3 y con ello mejorararíamos la precisión de nuestra aplicación de análisis de sentimientos, rama que posteriormente se podría paralelizar también.

Referencias

[1] *Python (lenguaje de programación).* https://es.wikipedia.org/wiki/Python . 2018

[2] *Cython .*https://es.wikipedia.org/wiki/Cython

[3] *Cython. C-extensions for python* https://cython.org

[4] *C++ (lenguaje de programación)* https://es.wikipedia.org/wiki/C%2B%2B

[5] R. Stagner, *"The cross-out technique as a method in public opinion analysis,"* The Journal of Social Psychology, vol. 11, no. 1, pp. 79–90, 1940.

[6] S. A. Sandri, D. Dubois, and H. W. Kalfsbeek, *"Elicitation, assessment, and pooling of expert judgments using possibility theory,"* IEEE transactions on fuzzy systems, vol. 3, no. 3, pp. 313–335, 1995.

[7] Minqing Hu and Bing Liu,*Mining and summarizing customer reviews"*, KDD , 2004.

[8] David Zimbra, Ahmed Abbasi, Daniel Zeng and Hsinchun Chen, *The State-of-the-Art in Twitter Sentiment Analysis: A Review and Benchmark Evaluation*, ACM Transactions on Managment Information Systems, 2018.

[9] C. D. Manning and H. Schutze. *Foundations of statistical natural language processing.* MIT Press, 1999.

[10] Rosa Filgueira, Amrey Krause, Malcolm Atkinson, Iraklis Klampanos, Alessandro Spinuso, Susana Sanchez-Exposito . *FDispel4py: A Python Framework for Data-Intensive eScience*, November 2015.

[11] Songbo Tan, Xueqi Cheng, Yuefen Wang, and Hongbo Xu . *Adapting Naive Bayes to Domain Adaptation for Sentiment Analysis* Key Laboratory of Network, Institute of Computing Technology, China 2 Information Center, Chinese Academy of Geological Sciences, China, 2009.

[12] Nurulhuda Zainuddin,Ali Selamat. *Sentiment Analysis Using Support Vector Machine.* Faculty of Computing, Malaysia . 2014

[13] *Support-vector machine* https://en.wikipedia.org/wiki/Support-vector_machine

[14] Alec Go,Richa Bhayani, Lei Huang. *STwitter Sentiment Classification using Distant Supervision.*

[15] Breitinger, Corinna; Gipp, Bela; Langer, Stefan . "*Research-paper recommender systems: a literature survey".* International Journal on Digital Libraries. 17 (4): 305–338. 2015.

[16] Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, Manfred Stede . *Lexicon-Based Methods for Sentiment Analysis* ,2010.

[17] Filipe N Ribeiro ,Matheus Araújo, Pollyanna Gonçalves,Marcos André Gonçalves,Fabrício Benevenuto. *SentiBench - a benchmark comparison of state-of-the-practice sentiment analysis methods.* 2016.

[18] Fabrizio Sebastiani . *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining.* 2010.

[19] *Pro TBB C++ Parallel Programming with Threading Building Blocks.* Rafael Asenjo, Michael Voss, James Reinders. 2019

[20] Michael V. , *The Intel® Threading Building Blocks flow graph is now fully supported.* 2011

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA