ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DE COMPUTADORES

Comparación y análisis de métodos de clasificación con las bibliotecas scikit-learn y TensorFlow en Python

Comparison and analysis of classification methods with scikit-learn and TensorFlow libraries in Python

Realizado por
Juan Zamorano Ruiz

Tutorizado por
Daniel Garrido Márquez

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

Resumen

Hoy en día no podemos negar que vivimos rodeados de la generación de una gran cantidad de datos, más ahora que estamos en plena era digital y la mayoría de esta información se puede almacenar y procesar. Numerosas empresas buscan enfocar el uso de dicha información para sacar un beneficio a favor de los objetivos de su negocio y se ayudan de las herramientas adecuadas para llevar a cabo esta tarea.

La rama de la Inteligencia Artificial denominada “Machine Learning” o, también conocido como aprendizaje automático, es aquella que permite a las máquinas aprender a través de los datos detectando patrones y ayudando a la toma de decisiones.

Con esta rama de la inteligencia artificial lo que se pretende es dar uso al conocimiento de los datos históricos para poder llevar a cabo decisiones concretas en el futuro. Dentro del aprendizaje automático existen distintos tipos de problemas, dependiendo de su naturaleza y del objetivo buscado, así en este caso nos centraremos en aprendizaje supervisado para clasificación. Existen distintos algoritmos y técnicas matemáticas para poder llevar a cabo un problema de clasificación, donde una de las más usadas y conocidas es el de las redes neuronales.

En este proyecto vamos a hacer uso de dos bibliotecas para aprendizaje automático, profundizando en redes neuronales , ambas diseñadas para su uso en Python: una de ellas es la biblioteca Scikit-learn para *Machine Learning*, y otra parte la biblioteca de código abierto TensorFlow.

Se realizará una comparación y análisis del comportamiento de cada algoritmo y librería con distintas métricas en la predicción de 7 tipos de cubiertas forestales con el uso de variables cartográficas a través de distintos valores tomados en 4 áreas distintas del parque nacional de Roosevelt, en el norte del estado de Colorado.

Palabras clave: Scikit-learn, TensorFlow, Aprendizaje automático, Big Data, Python, Inteligencia Artificial, Keras, Análisis

Abstract

Today we can´t deny that we live surrounded by the generation of a large amount of data, indeed today that we are in the digital age and most of this information is possible to store and process. Numerous companies seek to focus on the use of such information to make a profit in favor of the objectives of their business and we help ourselves with the appropriate tools to carry out this task.

The branch of Artificial Intelligence called "*Machine Learning*" is one that allows machines to learn through data by detecting patterns and helping to make decisions.

With this branch of artificial intelligence what we intend is to use the knowledge of historical data to be able to carry out concrete decisions in the future. Within the automatic learning there are different types of problems, depending on their nature and the target, so in this case we will focus on supervised learning for classification. There are different algorithms and mathematical techniques to carry out a classification problem, one of the most used and known is neural networks.

In this project we will make use of two libraries for machine learning to implement a neural network, both designed for use in Python language: one of them is the Scikit-learn library for Machine Learning, and on the other hand the use of the open source library TensorFlow.

A comparison and analysis of the behavior of each algorithm and library will be made with different metrics in the prediction of 7 forest cover types with the use of cartographic variables different values taken in 4 different areas of the Roosevelt National Park, in the north of the state of Colorado.

Keywords: Scikit-learn, TensorFlow, Machine Learning, Big Data, Python, Artificial Intelligence, Keras, Analysis.

Índice

1

Introducción

1.1 Motivación

La motivación para realizar este Trabajo Fin de Grado viene relacionada con el auge que está experimentando el término Big Data y su aplicación en el campo del análisis de los datos para su uso en el aprendizaje automático. Vivimos en la era digital y al día se generan alrededor de 2.500 millones de gigabytes al día (PortalTic, 2017), la tendencia es a almacenarlos pero, ¿qué seríamos capaz de hacer si los analizamos y extraemos información importante para dar un valor añadido a esos mismos datos?

Los datos, una vez analizados y correctamente clasificados, son capaces de dar mucha información y una de ellas es la de seguir un patrón que nos indica un comportamiento. El ser humano es capaz de analizar y clasificar atendiendo a un número de atributos, pero, ¿qué pasaría si la cantidad de atributos y de datos a clasificar son cifras demasiado grandes?. Las máquinas están para resolver ese problema y, para ello, existen algoritmos que se encargan de identificar un patrón que siguen los datos obtener un modelo. Con ese modelo pueden ser capaces de clasificar nuevos datos que le proporcionemos, dependiendo de la técnica serán más o menos precisos. De entre esas técnicas existen las redes neuronales, de las cuales se puede dar uso para problemas de clasificación, como así han usado (Payá & Villalón, 2015), en la sanidad para la clasificación de arritmias cardíacas (Anuradha & Reddy, 2008) o como ayuda para la clasificación de la predicción y prognosis

del cáncer (Cruz & Wishart, 2006), donde se observa que ayudan a mejorar la clasificación que un doctor especialista haría usando su experiencia previa.

La motivación de este Trabajo Fin de Grado es el de utilizar un conjunto de datos de un repositorio conocido para llevar a cabo un análisis y comparación a través del uso de distintas técnicas de aprendizaje automático, y en especial las redes neuronales, extrayendo distintas métricas que permitirán analizar el comportamiento de cada uno. El paquete scikit-learn para Python tiene implementado un número importante de estos algoritmos y, a través de su API, se realizará la instanciación de estos algoritmos para, entrenarlos y analizarlos. Por otra parte se realizará la implementación de una red neuronal para clasificación en TensorFlow, otra librería para Python y que da la posibilidad del uso de aceleración gráfica para una mejor computación, ya que se basa en la realización de operaciones matemáticas de matrices, llamadas tensores, en las que se basan las redes neuronales para mejorar en sucesivas iteraciones.

Este trabajo está basado en la ampliación de uno previo realizado para scikit-learn. En esta ocasión se hace mayor hincapié en un análisis dirigido al funcionamiento y rendimiento de las redes neuronales, así como añadir la comparación de su funcionamiento para la librería TensorFlow. Se añade una nueva métrica para la comparación. El trabajo está disponible en el siguiente enlace:
https://eprints.ucm.es/48800/1/Memoria%20TFM%20Machine%20Learning_Juan_Zamorano_para_difundir%20%282%29.pdf

1.2 Objetivos

La inteligencia artificial, y en concreto el aprendizaje automático, está cada vez más extendidoa? , y lo podemos encontrar en nuestras vidas cada día. Así por ejemplo es usado para la detección de SPAM en SMS o cualquier servicio de correo (Cuenca, 2017).

Existen gran cantidad de técnicas y algoritmos que se encargan de identificar dichos patrones, a través de un conjunto histórico de datos, para generar un modelo y, a través de dicho modelo, clasificar nuevos datos sin saber su etiqueta correspondiente.

En la actualidad existen distintos tipos de bibliotecas o paquetes disponibles tanto para el lenguaje de programación Python como R. Una de las bibliotecas que se utilizará en este trabajo es la de libre uso Scikit-Learn (Pedregosa et al., 2012) orientada al lenguaje de programación Python (Raschka, 2015). En scikit-learn se dispone de numerosos algoritmos para clasificación que han sido implementados previamente y que nos permiten alimentar y extraer la información que se necesite con el uso correcto del flujo de datos, es decir, saber qué información se debe proveer y cual se puede extraer tras el procesamiento.

Otro paquete que se va a utilizar en este trabajo es la librería TensorFlow (Daniel Smilkov, Shan Carter, 2017) pensada para la generación de modelos de redes neuronales y la posibilidad de computación en distintos escenarios.

Lo que se busca así para la comparativa a llevar a cabo en este trabajo es, por un lado, entender de forma más precisa cómo funciona un clasificador y, por otro, llevar a cabo una comparativa a través de las distintas métricas que se pueden extraer de los mismos.

Las métricas que se pueden extraer para nuestra comparativa serán las que se pueden extraer del propio algoritmo para un conjunto de testeo, un conjunto de datos del que conocemos su etiqueta pero que vamos a usar para comprobar como se comporta el algoritmo una vez entrenado, estas métricas son:

- Exactitud, recuperación, precisión, Log Loss, valor F1 y tiempo de entrenamiento, entre otros, para clasificar un conjunto de test de datos sin etiqueta, métricas que se explicarán posteriormente para entender en qué consisten.

De los datos previos se generarán tablas donde almacenaremos los valores que cada algoritmo ha generado, tanto para sckit-learn como para TensorFlow. Posteriormente se usarán dichos datos para dibujar gráficas comparativas y llevar a cabo el análisis.

Para el almacenamiento y procesamiento del conjunto de datos se usará la biblioteca Pandas[1], orientada a Python, una biblioteca extensión de Numpy para manipulación y análisis de datos. Para el procesamiento de estos

[1] https://pandas.pydata.org/

datos se desarrollarán distintos programas en Python para automatizar, en la medida de lo posible, el procesamiento de los datos, el entrenamiento de los algoritmos y extracción de las métricas que nos darán la posibilidad de realizar la comparativa.

1.3 Estructura de la memoria

La estructura de la memoria de este trabajo va a consistir en varias partes que permitirán dar a conocer el uso del aprendizaje automático así como los distintos algoritmos que se pueden usar para realizar la clasificación de unos datos seleccionados, así como las respectivas gráficas y comparativas en las que el trabajo se centra. De esta forma, la memoria consta de:

- Una introducción para contextualizar el trabajo a realizar y por qué se ha propuesto su desarrollo.
- Un estudio del caso de uso a tratar y el trabajo que se pretende realizar sobre el mismo para aplicar los algoritmos de aprendizaje automático dando uso de las librerías disponibles para el objetivo buscado.
- Un capítulo donde se explica en qué consiste el aprendizaje automático y donde se analizan los algoritmos basados en redes neuronales y el porqué de su uso en nuestros días.
- Un capítulo donde se analizan y explican los pasos dados para analizar los datos, su procesamiento y entrenamiento de los algoritmos. Explicación del uso de programas para conseguir automatizar las distintas tareas que un trabajo de aprendizaje automático conlleva usando las librerías disponibles y los entornos de programación que facilitan su uso. Además se explican las métricas que se pretenden extraer.
- Un último capítulo donde se examinan las métricas obtenidas de los algoritmos seleccionados y se realiza una comparación atendiendo a cada una de ellas para posteriormente llegar a unas conclusiones atendiendo a los valores obtenidos.

2

Caso de uso y trabajo a realizar

2.2. ¿De dónde procede el conjunto de datos?

El conjunto de datos seleccionado, de tipo de cobertura forestal procede (Blackard Jock, Dean Denis, & Anderson, 1998) del depósito de Aprendizaje Automático de la UC Irvine, lugar donde mantienen aproximadamente cerca de 500 conjuntos de datos al servicio de la comunidad del aprendizaje automático.

Se necesitaba encontrar un conjunto de datos que permitiera llevar a cabo el problema de una clasificación y donde tuviera un conjunto lo suficientemente grande para entrenar los algoritmos, además de ser una fuente fiable que nos pueda asegurar que los datos son correctos y que no existe información perdida, algo que se observará en el Capítulo 2.2.

Para la elección de los datos es adecuado tener en consideración cuál será el objetivo del trabajo y qué elementos se quieren tener en cuenta para el análisis a llevar a cabo. Para poder realizar este análisis se debe tener en cuenta un conjunto de datos con un número de ejemplos significativo, que nos sirva para modelar nuestros algoritmos.

Así, las características que buscamos son:

- **Uso de almacenamiento moderado**. No podemos considerar aquellos que ocupen del orden de Terabytes.
- **Tiempo de cómputo moderado**. Esto es interesante para la extracción de modelos a través del uso de los distintos algoritmos previamente

estudiados y de donde podremos realizar el estudio y análisis de los mismos atendiendo a su comportamiento con las métricas seleccionadas.

- **Tipo de clasificación**. Podemos tener un tipo de clasificación binaria, elección entre cierto o falso, o multiclase donde tenemos 3 o más clases a clasificar. En esta ocasión es necesario que esté enfocada a multiclase para observar cómo los algoritmos hacen uso de la técnica *One-vs-Rest* (Rifkin, Klautau, & Org, 2004) .

Tras un análisis de distintos conjuntos de datos, su naturaleza, su distribución y atendiendo a que sea un problema tipo clasificación, se ha decidido finalmente la elección de un problema que se adecua y que resulta interesante para nuestro estudio, este es el de la predicción del tipo de cubierta arbórea usando solo variables cartográficas (Blackard Jock et al., 1998)

Según se puede ver en la información del repositorio, se observan las siguientes características:

Data Set Characteristics:	Multivariate	**Number of Instances:**	581012	**Area:**	Life
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	54	**Date Donated**	1998-08-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	237754

Ilustración 1. Metadatos del conjunto de datos seleccionado

Se observa un conjunto de 581012 instancias, es un problema de tipo clasificación, con 54 atributos y que no existen valores perdidos, algo que indica la fiabilidad de los datos para así proceder con confianza en el estudio, sin afectar al comportamiento de los algoritmos.

La idea de este tipo de clasificación se basa en la predicción del tipo de cobertura forestal, es decir, cual es el tipo de árbol más predominante en un área, teniendo como información variables estrictamente cartográficas. Este tipo de clasificación de cobertura forestal la determinó el área del Servicio Forestal de Estados Unidos (USFS) para cuadrículas de 30 x 30 metros. Estos datos fueron guardados tal y como se tomaron, por lo que no han sido transformados en una escala en el que todos formen parte de un mismo rango, además contienen atributos en forma binaria para variables cualitativas independientes como áreas silvestres o tipo de suelo, variables que derivaron a partir de los datos obtenidos del *US Geological Survey* y USFS. Estos datos

son importantes tenerlos en cuenta, pues influyen en los algoritmos en el momento de entrenarlos.

Los datos provienen de 4 áreas silvestres en el Bosque Nacional de Roosevelt en el norte del estado de Colorado. Estas áreas representan bosques con un mínimo de cambios provocados por el hombre por lo que los tipos de estas cubiertas forestales son representativos de procesos ecológicos naturales en lugar de repoblaciones o cambios realizados por el hombre.

Las 4 áreas silvestres se encuentran representadas en el siguiente mapa y donde se pueden encontrar los siguientes tipos de cubierta arbórea, imágenes obtenidas del enlace[2] :

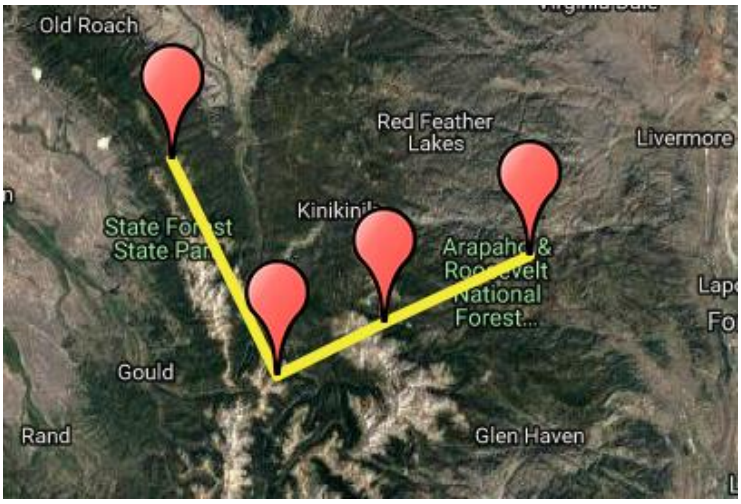


Ilustración 2. Mapa distribución de las zonas del parque

- zona Cache La Poudre:

Ilustración 3. Zona Cache La Poudre

[2] tourbuilder

- zona comanche peak

Ilustración 4. Zona Comanche Peak

- zona Neota

Ilustración 5. Zona Neota

- zona Rawah

Ilustración 6. Zona Rawah

2.2. ¿Qué atributos tiene el conjunto de datos?

Las características que tiene el conjunto de datos seleccionado son:

- Altitud: cuantitativo, expresado en metros representando la elevación en metros de cada instancia recogida.
- Aspecto: cuantitativo, expresa el aspecto en grados acimut que muestra la instancia recogida. Este es un dato usado para cartografiar.
- Sombra: cuantitativo, expresa los grados de sombra que proyecta el ejemplo donde se ha tomado.
- Distancia horizontal hasta hidrología: cuantitativo, expresa en metros la distancia horizontal hasta el recurso hidrológico más cercano.
- Distancia vertical hasta hidrología: cuantitativo, al igual que el anterior, pero en esta ocasión expresa en metros la distancia vertical.
- Distancia horizontal hasta carreteras: cuantitativo, refleja la distancia horizontal en metros hasta la carretera más próxima.
- Sombra a las 9 de la mañana: cuantitativo, indica el índice de sombra en el solsticio de verano a las 9 de la mañana, un índice que va en el rango de 0 hasta 255.
- Sombra al mediodía: al igual que el anterior, pero al mediodía.
- Sombra a las 3 de la tarde: similar que el anterior, pero en esta ocasión valor tomado a las 3 de la tarde.
- Distancia horizontal al punto de fuego más cercano: cuantitativo, muestra la distancia horizontal en metros a un posible punto de fuego.
- Área silvestre: 4 columnas binarias de tipo cualitativo para cada una de las áreas silvestres que se tratan en este problema. 1 indica presencia y 0 ausencia.
- Tipo de suelo: 40 columnas binarias de tipo cualitativo para indicar el tipo de suelo. 1 indica presencia y 0 ausencia dentro de 40 tipos de suelos posibles.
- Tipo de cobertura : 7 tipos de cobertura arbórea posibles para este estudio. Es de tipo entero donde podemos tener valores del 1 al 7. Estos son los distintos valores que se quieren predecir y en los que se centra el trabajo para la extracción de métricas. Estos 7 valores serían los distintos tipos de cobertura arbórea predominante en el área del que se han cogido los datos. Estos tipos son:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

Las siguientes imágenes representan cada uno de los tipos de cubierta arbórea que podemos encontrar en las 4 áreas y que queremos predecir con la información que tenemos del conjunto de datos.

Ilustración 7. 7 tipos de cobertura arbórea

2.2. ¿En qué va a consistir nuestro trabajo?

El objetivo de este trabajo es el de llevar a cabo el procesamiento de un problema de aprendizaje automático para clasificación, el cual involucra procesar los datos, entrenar los algoritmos de clasificación elegidos, obtener información relevante para almacenarla y, posteriormente, analizarla y compararla. Esta información es recopilada a través de los correspondientes módulos que se van a usar en las librerías del lenguaje de programación escogido, en este caso Python, y que se podrá saber a partir de ellas cómo cada algoritmo de los que se ha analizado realiza la clasificación para el conjunto de datos que se ha escogido.

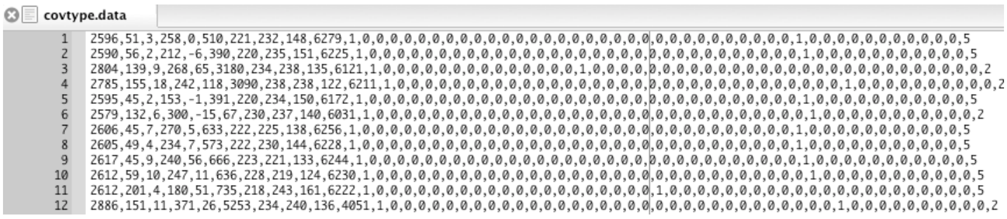
De esta forma, se pueden identificar los siguientes pasos una vez escogido el conjunto de datos a tratar, teniendo en cuenta que se quiere observar un problema de clasificación, y que se describe a continuación:

2.2.1. Descarga de los datos del repositorio

Los datos están disponibles en el repositorio online que UCI tiene disponible para descargarlos. El que se quiere tratar (predicción del tipo de

cubierta arbórea forestal) se puede encontrar en el siguiente enlace: https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/.

Una vez descargados y descomprimidos se puede examinar y comprobar que es un archivo con extensión sin formato, pero que se puede abrir con un editor de texto y poder comprobar el aspecto que tiene la información. Así se observa lo siguiente:


covtype.data

```
2596,51,3,258,0,510,221,232,148,6279,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5
2590,56,2,212,-6,390,220,235,151,6225,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5
2804,139,9,268,65,3180,234,238,135,6121,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2
2785,155,18,242,118,3090,238,238,122,6211,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2
2595,45,2,153,-1,391,220,234,150,6172,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5
2579,132,6,300,-15,67,230,237,140,6031,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,2
2606,45,7,270,5,633,222,225,138,6256,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5
2605,49,4,234,7,573,222,230,144,6228,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5
2617,45,9,240,56,666,223,221,133,6244,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5
2612,59,10,247,11,636,228,219,124,6230,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5
2612,201,4,180,51,735,218,243,161,6222,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5
2886,151,11,371,26,5253,234,240,136,4051,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2
```

Ilustración 8. Ejemplo de distribución de los datos

Se comprueba que la información está tal y como se describe, donde cada fila corresponde a cada ejemplo observado y sus respectivas columnas corresponden a los atributos que deben tener para cada ejemplo. El último valor corresponde al tipo de cobertura forestal al que pertenece cada ejemplo.

Para un mejor manejo de estos mismos e importación en el código para su tratamiento en scikit-learn, se ha optado por tenerlo en formato con valores separados por compa (CSV) y donde se ha incluido la cabecera con los títulos de cada atributo, algo que servirá para la descripción de los datos a continuación.

A partir de ahora el manejo y ejecución de comandos para el tratamiento de los datos y observación del estado en el que se encuentran se realizará a través de la aplicación Spyder[3], un entorno desarrollado para Python y que facilita el acceso a la información así como la ejecución y depuración del programa desarrollado.

2.2.2. Estadísticas de los datos

Una vez importados los datos se realiza una observación del estado de los mismos y detalle de sus estadísticas. Así, tras importar en la variable `train` el conjunto de datos, se pueden listar sus columnas con el comando `list(train.columns.values)` y obtenemos los siguientes valores:

[3] https://www.spyder-ide.org/

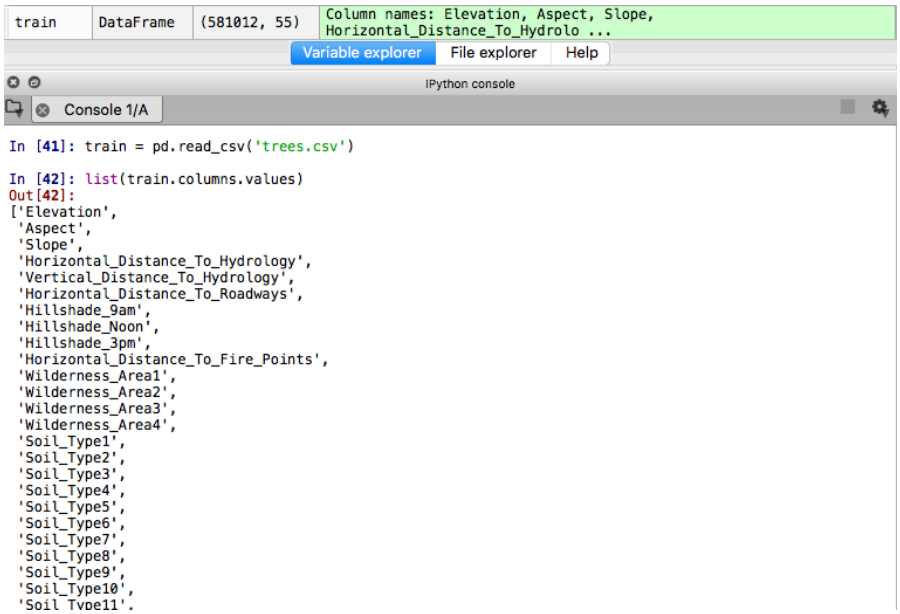


Ilustración 9. Descripción de las columnas de los datos

Con el comando `train.describe()` se podrá extraer información de cada columna, así como la suma total, la media, la desviación estándar, el valor mínimo y el máximo entre otros datos. Por ejemplo, si la desviación estándar de una columna (atributo) es 0, quiere decir que no existen variaciones, y por lo tanto esa columna o atributo se puede eliminar del conjunto de datos, ya que es irrelevante, no influye en la clasificación. Se observa que todas las columnas tienen 581012 ejemplos, por lo que no existe información perdida.

La siguiente imagen sirve de ejemplo de la salida del comando para los atributos *Elevation, Aspect* y *Slope*.

	Elevation	Aspect	Slope
count	581012.00000	581012.00000	581012.00000
mean	2959.365301	155.656807	14.103704
std	279.984734	111.913721	7.488242
min	1859.000000	0.000	0.000
25%	2809.000000	58	9
50%	2996	127	13
75%	3858	360	66
max	3858	360	66

2.2.3. Interacción de los datos

La interacción con los datos está relacionado con llevar a cabo una serie de acciones sobre los datos para así saber plantear el problema en sí y tenerla para diseñar adecuadamente el entrenamiento de los algoritmos.

Un ejemplo de esto es comprobar la distribución de los datos proporcionados y saber cuántos ejemplos se tiene de cada clase. Con la librería Seaborn[4], la cual se utilizará para visualización estadística de datos posteriormente, se puede comprobar esto. Así se importa en la variable `sns` dicha librería y se ejecuta el siguiente comando :

`sns.countplot(data=train,x=train['Cover_Type']),` la cual muestra la siguiente imagen:

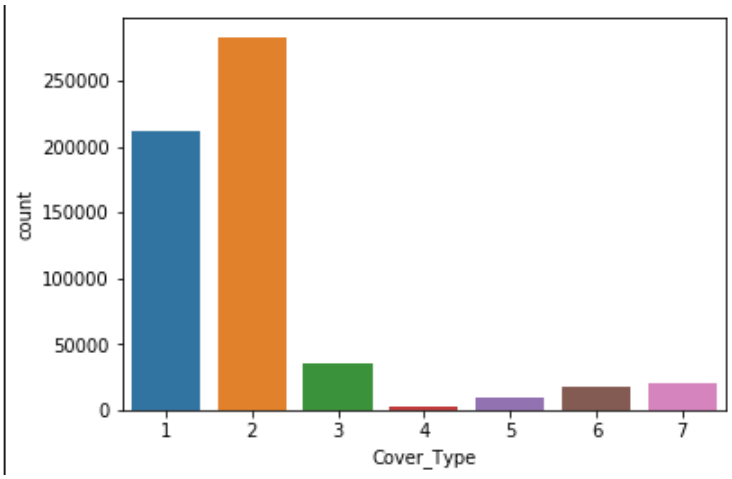


Ilustración 10. Distribución de los datos según clases

Se puede ver que existe una mayor representación del tipo 2 mientras del tipo 4 es del que menos muestras se tienen. Esto se debe tener en cuenta, pues en el conjunto de datos que se usará para entrenar los algoritmos debe haber una representación de cada clase de la forma más equitativa posible, algo que se solventará usando técnicas de validación cruzada que se explicará en el Capítulo 4.

2.2.4. Preparación de los datos

El conjunto de datos seleccionado se obtiene con una serie de atributos o características numéricas disponibles tal y como se tomaron en su momento, es lo que se puede definir como los “datos en crudo”. Por ello es necesario llevar a cabo una visualización de los mismos y adaptarlos, para así poder

[4] https://seaborn.pydata.org/

alimentar a los algoritmos que se van a entrenar de la forma más eficiente posible.

Un ejemplo de esto es que no existe un estándar de distribución de los mismos en una misma escala. Así se puede comprobar en el apartado anterior, donde se toman como ejemplo 3 de los atributos disponibles y donde se ve que la elevación, aspecto o sombra toman valores en distintas escalas.

Muchos de los algoritmos de aprendizaje automático que se van a usar en las librería seleccionadas necesitan o requieren de una normalización del conjunto de datos, algo que se explicará en el Capítulo 3 y donde se indica el motivo de porqué influye dicha escala. Se comprobará la diferencia de rendimiento para cuando se analicen las métricas que se extraen.

Así, por ejemplo hay muchos elementos que se usan para la función objetivo de algunos algoritmos de aprendizaje, ya que puede influir en la varianza de los datos y donde, en cada iteración, pueden centrarse más en datos con valores más abultados, algo que no quiere decir sean más influyentes en la búsqueda del patrón que cada algoritmo va a realizar.

La librería que se ha seleccionado posee los métodos adecuados para llevar a cabo esa transformación de los datos y es así como se hará para prepararlos antes de entrenar los algoritmos.

Por otra parte, es necesario crear los respectivos conjuntos de datos para evaluar la clasificación que hace cada algoritmo. Cada uno es un subconjunto del que partimos, en concreto serán dos, uno de ellos para entrenar los algoritmos, y el otro se utilizará para evaluar cómo se comporta cada algoritmo, una vez entrenado, para clasificar nuevos datos, aquellos para los que ya se sabe cuál es la salida correspondiente, pero que se eliminará para que cada algoritmo indique cuál es la salida que le corresponde, es decir, el tipo de cobertura a la que pertenece. Esta técnica se llama validación cruzada, o *cross-validation,* y que se explicará en el Capítulo 3.

2.2.5. Evaluación y predicción

Una vez realizados los pasos anteriores se procederá a entrenar cada uno de los algoritmos seleccionados, para obtener un modelo y posteriormente evaluar su comportamiento, para ambas librerías. Se guardarán los valores de las métricas para, posteriormente, llevar a cabo el análisis y comparación que

se propone en el trabajo, métricas que se explicará de donde proceden en el Capítulo 3.

Se centrará sobre todo en los algoritmos que se basan en redes neuronales artificiales y que se usará tanto para la librería scikit-learn como para TensorFlow, así se pueda ver el comportamiento, partiendo de la misma base pero implementadas de distinta forma.

2.2.6. Visualización y análisis de los datos

Una vez obtenidas las métricas y adecuadamente almacenadas, se procederá a su visualización donde se hará uso de las respectivas librerías que permitirán mostrar gráficas para así poder comparar con mayor precisión y con una visualización más amigable. Para ello se usará la librería Seaborn, comentada previamente, que nos permite llevar a cabo esta visualización estadística de los datos, y tenerla gráficamente para una mejor experiencia visual.

3

Aprendizaje automático

El aprendizaje automático (Cruz & Wishart, 2006) se encarga de dar uso a una variedad de técnicas estadísticas, probabilísticas y de optimización para dar la capacidad a las máquinas de aprender del pasado y usar dicha información para clasificar o identificar patrones. Esta rama de la inteligencia artifical se usa para analizar e interpretar datos como se hace en la estadística. Sin embargo en el aprendizaje automático se usa lógica booleana como AND, OR, NOT, o por ejemplo condicionales absolutos como IF, THEN, ELSE, además de otro tipo de condicionales que ayudan al reconocimiento de patrones dependiendo de los datos que se estén tratando, algo parecido a los métodos que los humanos usamos para aprender y clasificar. El aprendizaje automático también funciona de esta forma pero es más potente debido a que es capaz de tomar decisiones que las estadísticas no pueden o el mismo humano no es capaz de detectar. En muchos casos los métodos estadísticos que conocemos se basan en análisis de correlación de variables y es donde mejor funcionan, pero, ¿qué ocurre si las variables son independientes entre sí y no existe una correlación lineal entre ellas o existe alguna inter-dependencia entre ellas?. Es aquí donde el aprendizaje automático mejor funciona.

Para nuestro caso de uso, se puede observar como, debido a la cantidad de los datos que se poseen, la cantidad de atributos con los que contamos y la independencia de los mismos entre ellos, hace que el uso de técnicas de aprendizaje automático aporten el valor añadido para poder llevar a cabo un entrenamiento y clasificación de los datos.

3.1. La clasificación en el aprendizaje automático

Como se ha comentado previamente, el aprendizaje automático puede ser usado en varios campos. Uno de ellos es el de la clasificación, donde podemos usar diferentes técnicas o algoritmos que ayudarán a extraer información relevante de la que podemos dar uso dependiendo del objetivo buscado, por ejemplo para la toma de decisiones.

Dentro de las numerosas técnicas que el aprendizaje automático nos ofrece, este proyecto se va a centrar en la clasificación. Se parte de un conjunto de datos previamente etiquetado y con valores discretos. El uso de algoritmos de aprendizaje automático para clasificación serán capaces de identificar patrones y "aprender". Se partirá, inicialmente, de un conjunto de entrenamiento con un número de características y con el conocimiento de su etiqueta o salida con el que se obtendrá un modelo al entrenar. Cuando se provea a dicho modelo con un nuevo dato sin clasificar, será capaz de indicar o no qué etiqueta debe, algo que dará lugar a las métricas que se quieren analizar.

Existen distintos algoritmos, o técnicas matemáticas, de aprendizaje automático para clasificación, dependiendo de su implementación, la precisión de acierto con cada dato será mejor o peor, ya que la distribución de los datos y la naturaleza de los mismos podrá afectar de una manera u otra los cálculos internos que el mismo realice.

Un ejemplo a destacar de este tipo de aprendizaje automático es el de la clasificación binaria para detectar si un paciente tiene una enfermedad o no (Schapire, 2008).

En este trabajo se buscará que el algoritmo encuentre un modelo que se adecue al patrón que los datos siguen y, a partir de ese patrón, se proveerá al modelo datos sin conocer su etiqueta para que realice una predicción.

3.2. Aprendizaje supervisado

Este trabajo está basado en aprendizaje supervisado o con clase, (Hormozi, Hormozi, & Nohooji, 2012; Kotsiantis, 2007), y es aquel donde el

objetivo es aprender o identificar un patrón a partir de un conjunto de datos que utilizaremos de entrenamiento, el cual nos permitirá realizar predicciones de conjuntos de datos no observados previamente, una vez realizado el entrenamiento, es decir, se tiene conocimiento previo de los datos. De esta forma, podemos indicar que el concepto de “supervisado” proviene de que a partir de datos usados de ejemplo, se usarán como entrada para entrenar o alimentar nuestro modelo, y donde se sabe la salida deseada, se van a usar para entrenar el algoritmo, para así ajustar el mismo con el objetivo de conseguir la salida deseada.

Otros tipos de aprendizaje que también se puede encontrar es el aprendizaje sin clase, o también conocido como aprendizaje no supervisado (Dayan, 2009; Ghahramani, 2004), y el aprendizaje reforzado (R. S. Sutton & Barto, 2015).

3.4. Las redes neuronales

Las redes neuronales se han usado para el aprendizaje automático desde hace mucho tiempo, un ejemplo se encuentra en el reconocimiento de voz (Chen, Chen, & Lin, 1996). Aunque previamente se usaban como una ayuda ahora se le está dando mayor importancia para tomar decisiones, y es por ello que su uso en varios campos de la medicina y de la investigación se le está dando mayor importancia. Las redes neuronales son capaces de clasificar o de reconocer patrones atendiendo a unas características. Fueron diseñadas para modelar el cerebro humano llevando a cabo la conexión de las distintas neuronas que tenemos en el mismo, para así simular cómo somos capaces de aprender a través de un entrenamiento previo. Podemos encontrar numerosas referencias donde se usan redes neuronales tanto para el área médica como en otras, como la detección de correo spam (Awad & ELseuofi, 2011; Balakrishnan & Puthusserypady, 2016; Kumar & Duraipandian, 2013) y además demuestran que en muchos casos mejoran la predicción y clasificación que un experto podría hacer, así es el caso de la detección y prognosis de distintos tipos de cáncer (Cruz & Wishart, 2006). Para la mayoría de casos existen numerosas variables y distintos patrones que en ocasiones pueden llevar a confusión por mucha experiencia que se tenga, además de la fatiga que un ser humano pueda tener. Las redes neuronales llevan a cabo gran cantidad de

operaciones hasta llegar a conseguir la salida deseada, o lo más parecido, cuando está realizando el entrenamiento.

El número de capas ocultas o la cantidad de neuronas internas a usar en cada una de ellas es un número que no se sabe a priori cual usar, requiere de cierta experiencia y un estudio adecuado del tipo de problema para ajustar adecuadamente dichos valores para poder conseguir el mejor resultado, así como el usar más capas o más neuronas no quiere decir que se consigan mejores resultados.

Una red neuronal se trata de un conjunto de neuronas con varias capas ocultas y conectadas entre sí todas para conseguir un resultado. El número de neuronas de entrada se corresponde con la cantidad de características del conjunto de datos, y el número de neuronas de salida se corresponderá con las distintas clases que tiene el problema. Se tratará de vectores de números y con las que se tendrán que realizar operaciones de multiplicación de matrices. Por esto toma tanta importancia y consigue un buen rendimiento la librería de TensorFlow, ya que está enfocada en el uso de este tipo de operaciones, multiplicaciones matriciales, y que sacan partido del rendimiento de los procesadores gráficos (GPU) enfocados en esas tareas.

Se verá en el Capítulo 6 como ese rendimiento mejora cuando usamos Tensorflow con la ventaja del uso de GPU´s, más si cabe si se compara con scikit-learn que solo usa el rendimiento de la CPU.

3.3.1. Las redes neuronales y su uso para clasificación

Como se ha comentado previamente, existen distintas técnicas y algoritmos de aprendizaje automático para clasificación. Dependiendo de la técnica y la naturaleza del mismo, se pueden clasificar en distintas categorías: lineales, máquinas de vectores de soporte (SVM), análisis discriminante, conjunto, árboles de decisión o, las que ya se han mencionado, redes neuroanles. En este trabajo se va a centrar el estudio en las redes neuronales y, por lo tanto, se va a realizar un estudio de su funcionamiento y en qué están basadas.

Para poder desarrollar el concepto de red neuronal es necesario previamente explicar en qué consiste una neurona artificial y para ello se va a

basar la explicación del Perceptrón Simple que el psicólogo Rosenblat desarrolló basándose en otros conceptos.

3.3.1.1. Perceptrón Simple

Se puede decir que el Perceptrón es un modelo de neurona simple. En el año 1958 el psicólogo Frank Rosenblat (Rosenblatt, 1958) desarrolló este modelo de neurona basándose en el presentado por McCulloch y Pitts (McCulloch & Pitts, 1943) y en una regla del aprendizaje que se basaba en la corrección del error. A los científicos de aquella época les llamó la atención la capacidad que tenía este modelo para aprender patrones y, por lo tanto, usarlo para clasificación atendiendo a esos patrones. Este tipo de neurona, el perceptrón, está basado en una cantidad de sensores, o también se pueden llamar entradas, desde donde recibe los datos a reconocer. También se encuentra una neurona de salida que, dependiendo del valor obtenido, indica si el dato con las características introducidas pertenece a una clase u otra.

Para poder entender en que consiste el algoritmo del Perceptrón simple, es necesario también entender las funciones en las que se basa para poder llevar a cabo esa clasificación de datos, basadas en el concepto de Perceptrón simple.

Se supone que tenemos la función *f* de R^n en {-1, 1}, a la que le podemos aplicar un patrón de entrada $x =(x_1,x_2,...,x_n)^T \in R^n$ y donde tendremos una salida deseada $z \in$ {-1, 1}, o lo que es lo mismo, f(x)= *z.*

Dicho patrón de entrada, es lo que se va a considerar como los atributos que cada dato del conjunto de datos tiene y que se le pasa a la función previamente mencionada. Al tener un número de patrones de entrada al cual queremos llevar acabo la clasificación, tendríamos la siguiente relación $\{x^1, z^1\}, \{x^2, z^2\}....\{x^p, z^p\}$, donde x^i es el patrón de entrada $i \in R^n$ y $z = f(x^i)$.

Esta función lo que hace es una partición del conjunto de entrada en dos espacios, lo que se dice una clasificación binaria. Por una parte se tendrían aquellos patrones cuya salida da +1 y por otro lado aquellos que da como salida -1. Esto indica que la función previamente comentada es capaz de distinguir entre dos clases.

El siguiente paso sería el de construir un modelo que cumpla con la función previamente mencionada. Para ello vamos a partir de una unidad de proceso bipolar la cual cumple la siguiente función:

$$\left\{ \begin{array}{c} 1\ si\ w_1x_1 + w_2x_2 + \cdots + w_nx_n \geq \theta \\ -1\ si\ w_1x_1 + w_2x_2 + \cdots + w_nx_n < \theta \end{array} \right\}$$

Aquí se tiene que los parámetros w_i son los llamados pesos sinápticos. Estos pesos suponen la importancia que le vamos a dar a cada característica o valor de entrada. Por otro lado, se encuentra la suma ponderada que se llamará potencial sináptico y finalmente encontramos el umbral que es el símbolo θ. Si la salida de dicha función es 1 entonces se dice que está activa y en caso contrario será -1, o que está inactiva. En la siguiente ilustración se muestra cómo funciona este modelo explicado:

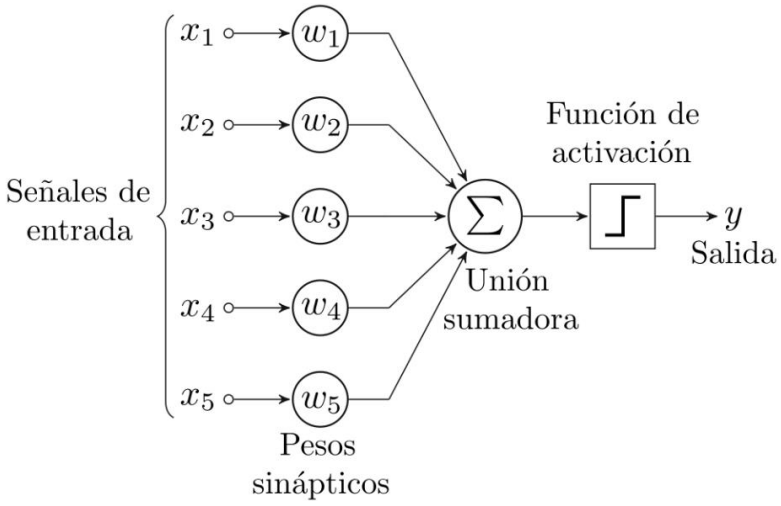


Ilustración 11. Ejemplo de Perceptrón
https://es.wikipedia.org/wiki/Perceptr%C3%B3n#/media/File:Perceptr%C3%B3n_5_unidades.svg

Pero, ¿cuál es el valor inicial de los pesos sinápticos y el umbral?. Esto es algo que se realiza a través de un proceso adaptativo con unos valores iniciales aleatorios y que se irán modificando según obtenga o no la salida deseada en el entrenamiento. Dicha modificación es la que se conoce como **Regla de aprendizaje del Perceptrón Simple** (Chakraverty, Sahoo, & Mahato, 2019)**.**

3.3.1.2. Perceptrón Multicapa

Se puede decir que el Perceptrón multicapa es una generalización del Perceptrón simple y el cual surgió como consecuencia de las limitaciones que

tenía a la hora de clasificar conjuntos de datos que no son linealmente separables. En su momento, Minsky y Papert (Multicapa, 1969) fueron capaces de demostrar en el año 1969 que con la combinación de varios Perceptrones simples, usándolos en capas ocultas, podrían solucionar el problema de clasificación para aquellos conjuntos que no son linealmente separables. Sin embargo, se encontraron con el problema de que la regla del perceptrón simple no es posible aplicarla a este problema, al existir dichas capas ocultas que se han comentado. A pesar de esto, la idea de combinar varios Perceptrones simples sirvió de ayuda para los estudios realizados por Rummelhart, Hinton y Williams cuando en 1986 (Rumelhart, Hinton, & Williams, 1986) presentaron una forma de retro-propagación del error que ha cometido la red y de cómo adaptar los pesos sinápticos a través de una regla. Esta regla es la conocida como regla delta o de retro-propagación.

La arquitectura de un Perceptrón multicapa (Multicapa, 1969), como su nombre indica, se basa en una disposición de sus neuronas en varios niveles o capas. Esta arquitectura es una de red de alimentación hacia adelante (*feedforward*) y en la cual se encontrará una capa de entrada con un número, dependiendo de la cantidad de características de nuestro conjunto de datos, otra capa de salida, que serán los distintos tipos a clasificar, y un número determinado de capas intermedias de proceso, que se pueden llamar ocultas, ya que no existe conexión con el exterior. El papel que desempeña la capa oculta o intermedia es la de una proyección de los patrones de entrada en un cubo cuya dimensión viene dada por el número de unidades de la capa oculta.

Las unidades de salidas están conectadas sólo con la última capa oculta. Con este tipo de red lo que se pretende es establecer una relación entre un conjunto de entrada y otro de salida, así tenemos la siguiente relación: $(x_1,x_2,x_3...x_n) \in R^n \rightarrow (y_1,y_2,y_3....y_m) \in R^m$. De esta forma se parte de un conjunto *p* de patrones de entrenamiento donde sabemos que para el patrón de entrada $(x^k_1,x^k_2,...x^k_n)$ le corresponde la salida $(y^k_1, y^k_2,..y^k_m)$ con k =1,2,..p. En la siguiente figura se muestra cómo sería una representación de este tipo de red neuronal:

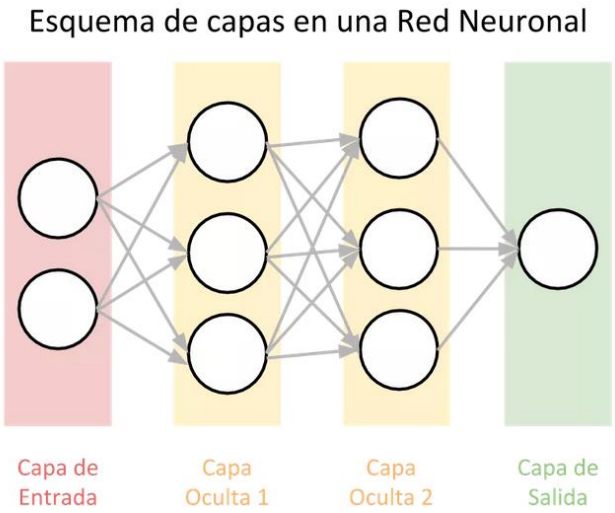


Ilustración 12. Estructura de una red neuronal
https://www.aprendemachinelearning.com/aprendizaje-profundo-una-guia-rapida/

Función de red neuronal multicapa

El procesamiento que realiza este tipo de red neuronal para poder extraer la salida y_i suponiendo una red con una sola capa oculta se encuentra de la siguiente manera:

$$y_i = g_1\left(\sum_{j=1}^{L} w_{ij} s_j\right) = g_1\left(\sum_{j=1}^{L} w_{ij}\left(g_2\left(\sum_{r=1}^{N} t_{jr} x_r\right)\right)\right)$$

Aquí se tiene que w_{ij} es el peso sináptico de la conexión entre la unidad de salida *i* y la unidad de proceso oculta *j*. *L* sería el número de unidades de proceso en la capa oculta; g_1 sería la función de transferencia para las unidades de proceso de la capa de salida, las cuales pueden ser la función identidad, la tangente hiperbólica o una función logística; t_{jr} es el peso sináptico que conecta la unidad de proceso *j* de la capa oculta con la entrada *r.* Por último, tenemos la función g_2 que es la función de transferencia de las unidades de proceso de la capa oculta, las cuales también pueden ser del tipo mencionado previamente para las unidades de proceso de la capa de salida, identidad, tangente hiperbólica y logística. Los pesos sinápticos asociados a cada entrada son también necesarios identificar para el inicio de la red neuronal.

3.4. Aprendizaje automático con scikit-learn

Existen numerosos lenguajes de programación que se pueden utilizar en la rama del aprendizaje automático. Entre ellos podemos encontrar Python, el más usado y conocido dentro de la comunidad de las ciencias de los datos. En

él existen bibliotecas de fácil uso e intuitivas que pueden ayudar al procesamiento, visualización y manejo de conjuntos de datos, los protagonistas en los algoritmos de aprendizaje automático. Como ejemplo son las librerías NumPy y SciPy, que están disponibles sobre otras capas como Fortran o C, para realizar operaciones vectorizadas de gran rendimiento en arrays multidimensionales, el tipo de estructura de datos en el que se centra este trabajo.

Una ventaja que permite el desarrollo en Python es el de poder modularizar el código y por lo tanto puede ser reusado en futuros desarrollos.

Existen gran cantidad de bibliotecas disponibles para usar en Python. En el trabajo que concierne se va a basar en aquellas que sirvan de uso para aprendizaje automático. De entre ellas se tomará de referencia a la de **Scikit-learn** (Pedregosa et al., 2012), una de las bibliotecas de libre uso de aprendizaje automático más utilizadas y populares en el día de hoy para aprendizaje automático.

Se podría decir que scikit-learn es como una caja de herramientas orientada a su uso en Python y pensada para ser utilizada para la minería de datos y, lo que comúnmente se conoce como ciencias de los datos, enfocada al aprendizaje automático.

La misma librería dispone de una API de la cual se pueden usar los módulos necesarios para importar en el proyecto y realizar las llamadas a los métodos requeridos. En el trabajo que concierne se hará uso de los distintos módulos que dan acceso a los algoritmos que se usarán para aprendizaje automático enfocados en clasificación, además de aquellos que se usarán para la extracción de métricas.

La API posee muchas características interesantes, las cuales se usarán para llevar a cabo el proceso de aprendizaje automático y extraer la información necesaria (Buitinck et al., 2013).

3.4.2. Establecer un protocolo de evaluación

Una vez conocido el objetivo a perseguir, hay que decidir cómo se va a medir el progreso para perseguir ese objetivo. Es aquí donde entra en juego el término de validación cruzada o también conocido como *cross-validation* (Arlot

& Celisse, 2009; Kohavi, 1995). Es una técnica para evaluar los resultados de un análisis estadístico y poder garantizar que son independientes de la partición entre el conjunto de datos que se utiliza para entrenamiento y el conjunto de prueba. Es utilizado en entornos donde el objetivo es la predicción y se quiere estimar cómo de preciso es el modelo generado.

Esta técnica surgió para resolver el problema del método de retención que consiste en dividir en dos subconjuntos los datos con los que contamos, realizar el entrenamiento con uno de ellos que se llama “Conjunto de entrenamiento” y validar el análisis con el otro, llamado “Conjunto de prueba”.

Lo normal es dividir el conjunto de datos en un 70% para entrenamiento y un 30% para prueba o también un 80/20, aunque este ratio debe ser considerado dependiendo del tamaño de nuestro conjunto de datos.

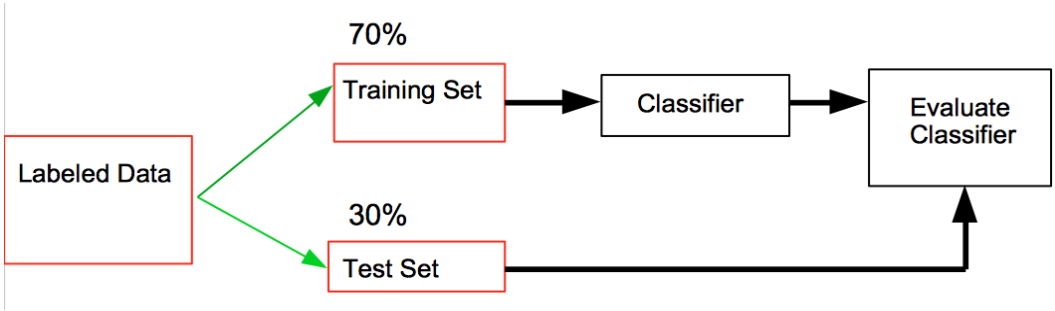


Ilustración 13. Validación cruzada

El problema de esta técnica es que el modelo que se genera solo se ajusta al conjunto de datos de entrenamiento y a partir de esto calcula los valores de salida para el conjunto de datos de prueba. Suele ser muy rápido pero no es del todo fiable porque es algo que depende cómo estén posicionados los datos. Si los datos están ordenados y en ese conjunto de entrenamiento que se ha creado solo hay información 4 clases de 7 posibles, el modelo que va a generar será muy preciso para esas 4, pero para las 3 restantes no tiene información y por lo tanto no sabrá como clasificarlos.

Es adecuado que exista una distribución equitativa de cada clase en cada conjunto, para que así pueda entrenar todos los patrones para cada tipo y pueda encontrar un modelo general para todo el conjunto de datos. Es aquí donde la validación cruzada entra en juego.

El módulo `model_selection` del paquete scikit-learn provee de distintos tipos de validación cruzada. Permitirán llevar a cabo esta división del conjunto de datos para proceder a su entrenamiento y posterior testeo para la extracción de las métricas y evaluar el comportamiento que ha tenido cada algoritmo o técnica. En scikit-learn se usará el tipo *StratiffiedShuffleSplit* que se encarga de conservar el porcentaje de muestras de cada clase para cada uno de los pliegues en los que se divide el conjunto de entrenamiento.

3.5. Aprendizaje automático con TensorFlow

3.5.1. ¿Qué es?

TensorFlow (Abadi et al., 2016) es un sistema de aprendizaje automático que funciona en entornos a gran escala y de forma heterogénea. Es una librería de código abierto para cálculo numérico y que usa como forma de programación grafos de flujo de datos. Nace del trabajo de Google Brain, que es un grupo de investigadores e ingenieros dedicados a la Inteligencia Artificial y que en 2011 desarrollaron DistBelief, llamado el predecesor de TensorFlow. En 2017 Google liberó la primera versión de este código con mejoras, como el rendimiento con el uso de GPU´s. El código que Google liberó es como software libre bajo licencia Apache 2 y por lo tanto una comunidad muy extensa está colaborando para mejorarlo e implementar nuevas características.

Este sistema de aprendizaje automático es hoy en día más usado de lo que creemos, así por ejemplo se encuentra en las respuestas automáticas que GMAIL ofrece para los correos o por ejemplo la famosa aplicación para traducir Google Translate.

El motivo de usar la palabra TensorFlow viene de la principal estructura de datos que conforma esta librería, y que son los “tensores”, y deriva de las operaciones que las redes neuronales realizan sobre arrays multidimensionales de datos. Al estar basado en grafos, los nodos en el grafo representan operaciones matemáticas, y por otro lado, las conexiones del grafo representan los conjuntos de datos multidimensionales, llamados tensores. Un tensor es un conjunto de datos primitivos, suponiendo números flotantes o números enteros,

organizados en un array de 1 o N dimensiones, el rango del tensor sería el número de dimensiones en las que se compone la estructura. Estas estructuras de datos son las que se van a usar para hacer que fluyan los datos entre las distintas capas de una red neuronal. Así se realizarán las operaciones necesarias para llevar a cabo el ajuste necesario y entrenar la red para clasificación.

3.5.2. ¿Cómo funciona?

TensorFlow usa grafos de flujo de datos para representar la computación, en un estado compartido, y las operaciones que cambian dicho estado. De lo que se encarga es de mapear los nodos de un flujo de datos a través de muchos equipos que están en clúster y dentro de muchos dispositivos para realizar computación, aquí se pueden incluir CPU con multicores, unidades dedicadas exclusivamente a GPU para procesamiento gráfico e incluso dispositivos modificados exclusivamente para funcionar con TensorFlow conocidos como Tensor Processing Units o TPU´s.

TensorFlow se encarga de proporcionar esto al programador a través del lenguaje de programación Python. Al fin y al cabo los nodos y los tensores son objetos de Python, y por otro lado, las aplicaciones de TensorFlow son en sí las mismas aplicaciones de Python. Sin embargo, las operaciones matemáticas que requieren estos procesamientos no se realizan en Python. Las bibliotecas a las que se referencian para realizar dichas operaciones matemáticas están escritas en lenguaje C++ de alto rendimiento. Python solo se encarga de dirigir el tráfico entre las distintas piezas y lo que hace es proporcionar la abstracción de programación de alto nivel necesaria para conectarlas.

TensorFlow se encarga de usar eficientemente gran cantidad de servidores con habilitación para usar GPU´s para un entrenamiento rápido, y se encarga de ejecutar modelos ya entrenados y ajustados para que se ejecuten en gran cantidad de servidores. Los nodos en el grafo los llama “ops”, el cual puede tomar cero o más tensores y que desempeña una operación computacional que da lugar a cero o más tensores.

Las aplicaciones que TensorFlow permite son varias y es que se puede ejecutar en la mayoría de destinos que necesitemos conveniente, ya sea una máquina local, un clúster en la nube, un dispositivo con iOS o Android e incluso cualquier CPU o GPU. Los modelos resultantes que creemos podremos guardarlos y ser importados en cualquier dispositivo donde se usarán para realizar predicciones.

El beneficio que ofrece para el aprendizaje automático es la posibilidad de abstracción. En vez de tener que llevar a cabo la implementación de los algoritmos o saber cómo debe conectar el flujo de los datos de una función en otra, simplemente tiene que centrarse en la lógica de la aplicación.

3.5.3. Uso de la librería Keras

Para implementar una red neuronal en TensorFlow se va a utilizar la librería para aprendizaje profundo (*deep learning*) Keras[5]. Esta librería es una API para redes neuronales y que permite ejecutarse en varias herramientas de aprendizaje automático, una de ellas es TensorFlow, que es nuestro objetivo.

Como principales características podemos destacar:

- Facilidad de uso: está pensado para que su uso sea lo más sencillo posible, con el mínimo de acciones requeridas por el usuario y con una API simple y consistente.
- Fácilmente extensible: se pueden añadir nuevos módulos como funciones y clases que nos permite acceder a más ejemplos de modelos.
- Funciona con Python: ésta es una característica principal ya que el trabajo se centra en el uso de las distintas librerías basadas en este lenguaje de programación.

3.5.5. Uso de Google Colaboratory

Colaboratory es una herramienta de investigación para la educación y la exploración del aprendizaje automático, en un entorno de bloc de notas de Jupyter Notebook y sin necesidad de configuración, gratuito debido a que está

[5] https://keras.io/

pensado como proyecto de investigación. Permite escribir nuestro código, guardarlo y compartirlo como si fuera un documento de Google Drive.

El motivo de usar este entorno es por el uso de la potencia de cómputo que puede dar un entorno con capacidad de uso de GPU (nVidia Tesla K80[6]). El código se ejecuta en una máquina virtual exclusiva para nuestra cuenta y que se recicla cuando se ha dejado de usar durante un tiempo prolongado, además de una vida útil máxima que determina el sistema, 12 horas en uso y 90 minutos si está ocioso. Este entorno en la nube permite usar estas máquinas de forma totalmente gratuita y nos ayudará a sacar mejor partido del procesamiento para TensorFlow para la multiplicación de matrices en redes neuronales. Al no necesitar de configuración tendremos un entorno listo y con las librerías necesarias sin tener que perder tiempo en hacerlo nosotros mismos en nuestra máquina, con los problemas de configuración e incompatibilidades que podamos encontrarnos.

3.6. Métricas a extraer

Es necesario saber controlar algo, que debe ser observable, y para conseguir éxito necesitamos también saber qué es lo que consideramos éxito. De los algoritmos de clasificación se pueden extraer métricas que nos ayudarán a saber en qué cantidad ese porcentaje lo hemos alcanzado, para ello se usan métricas de evaluación como precisión, exactitud o recuperación, aquellas que nos dicen como de buenas son las predicciones del algoritmo entrenado.

Las métricas son una parte fundamental de este trabajo, tanto para la librería de scikit-learn como para TensoFlow se podrán hacer uso de las mismas y se invocarán desde el mismo módulo (`sklearn.metrics`) que el paquete scikit-learn permite para ello.

Para entender las distintas métricas que se van a plantear a continuación es necesario entender previamente de donde se extraen las mismas. Para ello se procede a explicar lo que es la matriz de confusión

[6] https://www.nvidia.com/es-es/data-center/tesla-k80/

(Santra & Christy, 2012), una matriz que ayuda a saber cómo ha realizado la clasificación un algoritmo. Esta matriz simplemente muestra en un cuadro los siguientes valores, una vez realizado el entrenamiento y posterior validación con aquellos datos que tenemos para testear usando la validación cruzada: falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos. Para entender mejor este concepto se va a mostrar cómo sería esta matriz para una clasificación binaria:

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos(FP)	Verdaderos Negativos(VN)

Ilustración 14. Matriz de confusión para clasificación binaria

Como se puede observar, las etiquetas conocidas corresponden a las filas, y las que el algoritmo predice a las columnas. Para definir correctamente estos valores podemos indicar que:

Verdaderos Positivos (VP): es la cantidad de aquellas observaciones que fueron clasificados como pertenecientes a una clase y acertó.

Falsos Positivos (FP): Esas observaciones que no pertenecían a una clase, pero se llegaron a considerar perteneciente a ellas, son considerados positivos pero como sabemos la salida también sabemos que se ha equivocado.

Falsos Negativos (FN): Aquellas observaciones que pertenecían a una clase, pero no se consideraron en ella, considerados como negativos.

Verdaderos Negativos (VN): es la cantidad de aquellas observaciones que no pertenecían a una clase y las clasificó correctamente.

De todo algoritmo en scikit-learn podemos sacar esta matriz de confusión, así con el método `confusion_matrix` del paquete `sklearn.metrics` podemos extraer los 4 valores indicados previamente.

A partir de estos 4 valores previos podemos sacar nuevas métricas de rendimiento que nos ayudarán a extraer más información.

Estas métricas sirven para medir clasificaciones binarias. Como nuestro problema está enfocado a clasificación multiclase es necesario sacar el valor promedio para cada una, así para cada métrica tenemos VP_i que son los Verdaderos Positivos para la clase i, VN_i = Verdaderos Negativos para la clase i, FP_i= Falsos Positivos para la clase i y por último FN_i = Falsos Negativos para la clase i.

Las métricas que vamos a tener en cuenta en este trabajo son las siguientes:

Exactitud

Extraemos el porcentaje de observaciones que ha clasificado correctamente, ¿Qué porcentaje de predicciones fue correcta?. Cuanto más cercano sea su valor a 1 mejor será.

Su fórmula es la siguiente:

$$\text{Exactitud} = \frac{VP \ + \ VN}{VP + VN + FP + FN}$$

$$\text{Exactitud promedio} = \frac{1}{C}\sum_{i=1}^{C} \frac{VP_i + VN_i}{VP_i + FP_i + VN_i + FN_i}$$

Donde C es el número de clases cuando realizamos clasificación multiclase.

Recuperación

Mide la habilidad del clasificador de encontrar todas las observaciones positivas, buscamos que este valor sea lo más alto posible en una escala del 0 al 1. Su fórmula es como sigue:

$$\text{Recuperación} = \frac{VP}{(VP \ + \ FN)}$$

$$\text{Recuperación promedio} = \frac{1}{C}\sum_{i=1}^{C} \frac{VP_i}{VP_i + FN_i}$$

Donde C es el número de clases, que en nuestro caso serán 7

Precisión

La precisión mide la habilidad de un clasificador de no etiquetar como positivo una observación que se debe considerar como negativa. Cuanto mayor sea este valor mejor será. Su fórmula es la siguiente:

$$Precision\ = \frac{VP}{VP + FP}$$

$$\text{Precision promedio} = \frac{1}{C}\sum_{i=1}^{C}\frac{VP_i}{VP_i + FP_i}$$

Donde C es el número de clases, que en nuestro caso serán 7.

Tanto la precisión como la recuperación se basan en una compresión y medida de la relevancia:

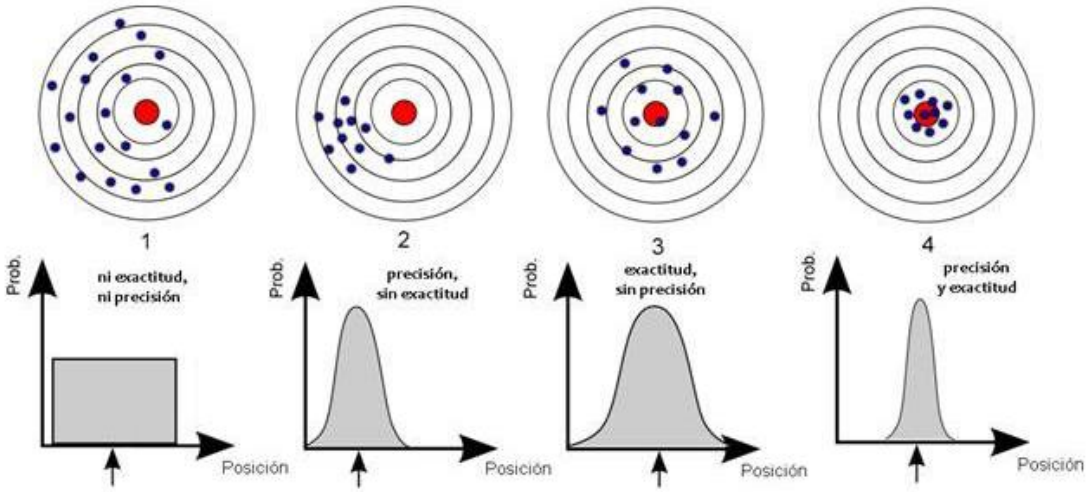


Ilustración 15.

https://medium.com/@gogasca_95055/precisi%C3%B3n-y-recuperaci%C3%B3n-precision-recall-dc3c92178d5b

Valor F

Esta métrica mide la media ponderada de la precisión y la recuperación. Así la fórmula es:

$$F1\ = 2\frac{Precisión\ x\ Recuperación}{Precisión + Recuperación}$$

Log Loss

Esta métrica (Figini, Pavia, Felice, Pavia, & Maggi, 2014) mide la incertidumbre que ha tenido nuestro clasificador con respecto a la etiqueta real que correspondía, es decir, si la clasificación ha variado o desviado mucho o poco con respecto a lo que debía ser. Lo que se pretende es que este valor sea lo más cercano a 0. Lo que hace es cuantificar la precisión del clasificador penalizando las clasificaciones falsas. Minimizando este valor quiere decir que maximizamos la precisión del clasificador.

Para calcular este valor, el clasificador asigna una probabilidad de pertenecer a cada clase para cada ejemplo. Matemáticamente la fórmula para calcular este valor es:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log p_{ij}$$

donde N es el número de instancias en nuestro conjunto de datos, M es el número de etiquetas posibles, y_{ij} es un valor binario que nos dice si para la instancia i su etiqueta es j, y p_{ij} es la probabilidad con el que el clasificador le ha asignado la etiqueta j a la instancia i.

4

Procesamiento de los datos y entrenamiento de los algoritmos

A continuación se va a explicar cómo se lleva a cabo la interacción con los datos y su procesamiento en los algoritmos de aprendizaje automático para las distintas técnicas que hemos incluido en este trabajo, tanto para la herramienta Scikit-learn como para TensorFlow. Para cada uno se han definido distintos programas en Python que permiten llevar a cabo todo el proceso que el aprendizaje automático necesita y se pueda llevar a cabo la comparativa y análisis:

Scikit-learn

El código está disponible para su descarga en el repositorio público en Github. El enlace para acceder a él y su descarga es el siguiente: https://github.com/juzaru18/forest_cover_sklearn

En este se realiza la importación de los paquetes necesarios a usar en el trabajo, tanto para el manejo de estructuras de datos con el paquete Pandas, como el procesamiento de los mismos para aplicar aprendizaje automático con los distintos métodos que el paquete scikit-learn incluye, o el uso de librerías para la visualización estadística de datos, como por ejemplo la librería Seaborn.

Como se comentaba en el Capítulo 2, para llevar a cabo el procesamiento de los datos y su análisis tras el entrenamiento de los algoritmos de aprendizaje automático, se van a realizar una serie de pasos para que el flujo de este aprendizaje sea el adecuado. Así los tres primeros, que son la descarga, estadística e interacción con los datos, se harán manualmente, a través del intérprete de comandos de Python para saber cómo están los datos y su distribución.

Para los siguientes pasos, que son los de preparación de los datos, evaluación y predicción, y por último, la visualización y análisis de los datos, se ha desarrollado un archivo con código en Python que permita desarrollar, a través de funciones y parámetros, el proceso por el que queremos llevar a cabo el aprendizaje automático haciendo uso de las distintas librerías disponibles y sus respectivos métodos. El poder hacerlo a través de un archivo con código permitirá una mejor depuración, control de versiones e iterar a través de los distintos algoritmos que se quieren analizar, y un mejor entendimiento del enfoque que se esté dando a este desarrollo. La idea es de dar la posibilidad a mejoras futuras y que permita un entendimiento de su funcionamiento con una sencilla lectura, a través de la ayuda de comentarios que indiquen el objetivo de cada función definida.

Para tener un orden y mejor comprensión del uso del script se han definido varias funciones, que pueden ser llamadas desde la función principal, para adecuarse a lo que se quiera obtener cuando se ejecute. Así se han definido funciones para llevar a cabo los pasos indicados. Para llevar a cabo las fases mencionadas se tiene lo siguiente:

1. **Importación de las librerías correspondientes**: Se importan las librerías que se usarán en el programa: pandas, scicikt-learn, numpy…etc.
2. **Importación de los datos**: Desde el programa principal se importa el conjunto de datos que tenemos guardado en formato CSV, y que se almacena en la correspondiente estructura de datos, en este caso :
   `train = pd.read_csv('trees.csv')`
3. **Preparación de los datos**: Tal y como se indicó en el Capítulo 2, los datos son descargados tal y como están, transformados a formato CSV, y, finalmente con el paquete `pandas` importados en una estructura de

datos con el correspondiente método `pd.read_csv`. Posteriormente se divide el conjunto en dos subconjuntos, uno con todos los atributos y otro con las etiquetas que le corresponde a cada uno. Así está definido el siguiente método:

```
def preprocess(dataset):
    labels = dataset.Cover_Type.values
    dataset= dataset.drop(['Cover_Type'], axis=1)
    return dataset, labels
```

Asi, la llamada a este método desde el programa principal quedaría de la siguiente forma:

```
train = pd.read_csv('trees.csv')
```

4. **<u>Validación cruzada para poder evaluar</u>**: Como se indicó en el Capítulo 4, es necesario poder medir el progreso de cómo se consigue el objetivo con las métricas que se quieren extraer, algo que la validación cruzada ayuda a ello. Así se realiza en los siguientes pasos en el programa principal:
    - Se define el tipo de cross validation que se quiere hacer donde se indica el número de pliegues (10) y cuanta cantidad de datos se va a usar para el testeo, en este caso un 30% (test_size=0.3), y por lo tanto el 70% se usará para entrenamiento:

      ```
      sss = StratifiedShuffleSplit(10, test_size=0.3,random_state=0)
      ```

    - Se divide el conjunto de datos en el siguiente código:

      ```
      for train_index, test_index in sss.split(X, y):
            X_train,
            X_test=X.values[train_index], X.values[test_index]
            y_train, y_test = y[train_index], y[test_index]
      ```

      Donde se tiene en `X_train` el conjunto de entrenamiento con los atributos, `X_test` el conjunto de testeo con sus atributos, `y_train` las etiquetas correspondientes al conjunto de entrenamiento y por último en la variable `y_test` están las etiquetas correspondientes al conjunto de testeo. Todas estas variables y sus datos se utilizarán a posteriori para, por un lado entrenar y por otro evaluar con el conjunto de test.

      Debido a que los datos no están en la misma escala, se ha definido un método para estandarizar los 10 primeros atributos, ya

que los restantes solo toman valores binarios, 0 ó 1. Así se realiza un entrenamiento sin estandarizarlos y posteriormente estandarizados, para ver cómo afecta al rendimiento de los algoritmos seleccionados. Esto se realiza en el método `standardize (train)`

5. **Evaluación y predicción**: Esta es otra de las fases que se estudió en el Capítulo 2, y fundamental para poder realizar la evaluación de cada algoritmo, pues previamente hay que entrenarlos, y así consigan tener un modelo de como los datos siguen un patrón. Para esto se ha definido una función que se invocará desde el programa principal y facilitar el procesamiento de los datos. Desde la misma se entrenan los algoritmos que se pasan por argumento (lista de algoritmos que queremos entrenar), se extraen las métricas, además del tiempo empleado para entrenar, y se almacena todo en una estructura de datos tipo tabla, que se utilizará para guardar en formato CSV, la cual servirá para posteriormente mostrar las gráficas que ayudarán a visualizar las métricas obtenidas. El método para el entrenamiento de los algoritmos es el siguiente:

   ```
   def compare_classifiers(classifiers, X_train, y_train,
   X_test, y_test, conf_matrix=False):
   ```

   Donde se pasa en el argumento `classifiers` una lista con los clasificadores que se quieren entrenar y evaluar. Se irá iterando para entrenar y evaluar con el conjunto de prueba para extraer las métricas. Para ello, primero se ha importado:

   ```
   from sklearn.neural_network import MLPClassifier
   ```

   Y se ha instanciado definiendo además sus hiper-parámetros en las siguientes líneas:

   ```
   MLPClassifier(warm_start=False, shuffle=True,
   nesterovs_momentum=True, hidden_layer_sizes=(1024, 512, 256, 128),
                         validation_fraction=0.333,
                         solver = 'adam',
                         learning_rate='constant', max_iter=162,
   batch_size=200, random_state=1,
                         momentum=0.11593,
                         tol=0.081977,
   ```

```
                    alpha=0.01, activation='relu',
early_stopping=False)
```

En el script existen instanciaciones de otras técnicas y algoritmos que también se entrenarán y que se usarán para tomar referencias en las gráficas comparativas.

6. **Visualización y análisis de los datos**: Para la última fase, la de visualización de los datos y análisis de los mismos, se ha desarrollado un método que permite mostrar los datos en las gráficas que se han seleccionado desde el paquete Seaborn. Así, desde el programa principal, se importa primero la tabla con las métricas que se guardaron en formato CSV en el paso previo, para pasarla como argumento al método de visualización de gráficas. Este método es el siguiente:

   ```
   def show_graphs(table)
   ```

   El cual se llamará desde el programa principal de la siguiente forma:

   ```
   graph = pd.read_csv('multilayer.csv')
   show_graphs(graph)
   ```

   En el archivo 'multilayer.csv' se tiene la información de cada algoritmo que

   hemos entrenado y del que se han sacado sus métricas.

Los otros algoritmos seleccionados para usar de la librería scikit-learn y que servirán para análisis y comparación son:

- **Perceptrón Simple**: es el tipo en el que se basa el perceptrón multicapa para simular una red neuronal. (Rosenblatt, 1958)
- **K Vecinos Cercanos (KNN)** : es uno de los más conocidos para el reconocimiento de patrones y uno de los más fáciles de implementar. Apenas usa datos para entrenamiento ya que la clasificación se basa en la búsqueda de características similares. (O. Sutton, 2012)
- **Arboles de decisión**: sus reglas para clasificación son fácilmente comprensibles para el ser humano y también son bastante eficientes. Se podrá comprobar una vez obtengamos las métricas. (Cha & Tappert, 2009)

- **Conjunto**: uso de la combinación de varios clasificadores para construir uno más robusto y preciso. (Geurts, Ernst, & Wehenkel, 2006)

TensorFlow

El código está disponible para su descarga en el repositorio público en Github. El enlace para acceder a él es el siguiente:

https://github.com/juzaru18/forest_cover_TensorFlow

Para TensorFlow se ha realizado un proceso parecido para el procesamiento de los datos y entrenamiento de los modelos que vamos a usar con esta herramienta, en este caso es el de dar uso a la librería Keras para crear un modelo de red neuronal, y posteriormente se ajuste al conjunto de datos que hemos seleccionado.

Los pasos tomados en el código realizado en Google Colaboratory haciendo uso de estas librerías refleja lo siguiente:

1. **Importación de las librerías correspondientes**: Importamos las librerías que vamos a usar en el programa, pandas, scicikt-learn, numpy…etc, para poder utilizarlas en nuestro programa.
2. **Importación de los datos:** Importamos los datos directamente desde el repositorio. Esto se realiza en la siguiente línea:

   ```
   !wget https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz
   !gzip -d "covtype.data.gz"
   ```

3. **Preparación de los datos:** En esta parte se realiza el proceso para la creación del conjunto de datos que se va a utilizar.
   - Para ello se importa la información en una estructura de datos pandas: `df = pd.read_csv("covtype.data", header=None)`
   - Se definen las constantes necesarias, con el número de clases que vamos a clasificar, el tamaño del conjunto de entrenamiento, el de testeo y por último el de validación, que forman parte de la

validación cruzada que se explicó en el Capítulo 3 :

```
N_CLASSES = 7
TRAIN_SIZE = int(0.7 * df.shape[0])
TEST_SIZE = int(0.15 * df.shape[0])
VALIDATION_SIZE = int(0.15 * df.shape[0])
```

- Se estandarizan los valores numéricos de los primeros 10 atributos, ya que los siguientes son todos de tipo 0 ó 1 y no es necesario estandarizarlos. Se la función que hemos definido `standardize` y que hemos usado también en la parte de scikit-learn:

```
features = standardize(df.iloc[:, 0:54])
labels = df.iloc[:, 54:].values
```

4. **Validación cruzada para poder evaluar:** Debido a que TensorFlow no posee métodos para la validación cruzada, es algo que se programará en el código que se está realizando para esta parte. Así se tiene:
   - Se definen los distintos conjuntos que se usarán para validación cruzada:

```
X_train, Y_train = features[:TRAIN_SIZE], y_one_hot[:TRAIN_SIZE]
X_test, Y_test = features[TRAIN_SIZE: TRAIN_SIZE + TEST_SIZE],
y_one_hot[TRAIN_SIZE: TRAIN_SIZE + TEST_SIZE]
X_validation, Y_validation = features[TRAIN_SIZE + TEST_SIZE : ],
y_one_hot[TRAIN_SIZE + TEST_SIZE : ]
```

   - Se crean los tensores necesarios para su entrenamiento en TensorFlow:

```
dataset_train = tf.data.Dataset.from_tensor_slices((X_train,
Y_train)).batch(256).shuffle(buffer_size=1000)
dataset_test = tf.data.Dataset.from_tensor_slices((X_test,
Y_test)).batch(256)
dataset_validation  =  tf.data.Dataset.from_tensor_slices((X_validation,
Y_validation)).batch(256)
```

5. **Evaluación y predicción:** Se selecciona el algoritmo, en este caso será el uso de keras para TensorFlow y que nos encargamos de definir en las siguientes líneas:
   - Se define el modelo, con :

```
model = tf.keras.Sequential([
 layers.Dense(1024, activation = 'relu'),
```

```
 layers.Dense(512, activation = 'relu'),
 layers.Dense(256, activation = 'relu'),
 layers.Dense(128, activation = 'relu'),
 layers.Dense(N_CLASSES, activation = 'softmax'),
])
```

- Se compila:

```
model.compile(optimizer=tf.train.AdamOptimizer(0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy'])
```

- Se entrena el modelo:

```
model.fit(dataset_train, epochs = 10, validation_data =
dataset_validation)
```

- Una vez se tiene el modelo, se predicen los datos para el conjunto de prueba con el método `predict`, y así extraer las métricas, tal y como se hizo para la librería scikit-learn, y a partir de esas predicciones se pueden extraer las métricas que hemos comentado en el Capítulo 3, como por ejemplo la recuperación:

```
y_pred = model.predict(dataset_test)
rec = recall_score(Y_test, y_pred_one_hot)
```

6. **Visualización y análisis:** Para esto, una vez extraído los mismos datos que para scikit-learn, se guardarán todos juntos en un mismo archivo CSV para importarlos y proceder a la visualización de los mismos. Esto es algo que se hace conjuntamente con los datos recopilados de scikit-learn, con todos los datos recopilados en un mismo archivo.

5

Métricas, comparativa y análisis

En este capítulo se mostrarán las métricas que se han obtenido de cada algoritmo y donde se procederá a una evaluación de los mismos indicando, según dichas métricas, cuál de ellos ha tenido mejor rendimiento para el conjunto de datos seleccionado. Se mostrará una gráfica de cada una de las métricas que se ha elegido y que se van a comparar, estas son: exactitud, recuperación, precisión, valor F, *log loss* y tiempo para entrenamiento.

Los valores se ordenarán según el objetivo buscado, así por ejemplo, para la exactitud es mejor cuanto más cerca del 100% esté, mientras que para log loss lo mejor es que esté más cerca del 0. Se mostrará en primer resultado aquel algoritmo que ha conseguido mejor resultado en cada una de las métricas.

Como ya se indicó en el Capítulo 2, los datos se obtienen tal y como fueron recogidos, en bruto y donde cada atributo está en una escala distinta. Por ello se va a tener en cuenta la estandarización de los 10 primeros atributos para así conseguir que estén en el rango [-1,1], los restantes atributos toman valores 0 ó 1, por lo que no son necesarios transformarlos. Esto es algo importante a tener en cuenta para muchos algoritmos ya que se consigue darle la misma importancia a todos los atributos y se evita que el algoritmo desvíe en la búsqueda del modelo hacia atributos que tienen más peso por rangos de

valores más dispares. Se comprobará que ésta estandarización afecta, tanto para mejor como para peor, a los algoritmos que se estudian, y es algo que se verá en las gráficas que se muestran a continuación.

Previamente, se mostrará en una tabla los datos obtenidos de cada algoritmo y donde se encontrarán las siguientes columnas, tal y como se guardaron los datos en formato CSV, para su posterior tratamiento con la librería *Seaborn,* y que representan las distintas métricas que hemos extraído para evaluar cada uno de los algoritmos: Clasificador, Ex (Exactitud), Prec (Precisión), Rec (Recuperación), F1 (Valor F), LL (Log Loss), Te(Tiempo entrenamiento).

A la hora de saber qué clasificador ha rendido mejor es necesario tener en cuenta varias cosas, por ejemplo, si los datos con los que entrenamos no están balanceados con una representación de todas las clases, no tendremos un valor real de exactitud. Es por ello que la validación cruzada nos va a ayudar a tener dicha representación en la mejor medida posible. Otra métrica a tener en cuenta es la de Log Loss que se encarga de penalizar aquellos fallos que el algoritmo ha cometido para un conjunto de prueba, lo que buscamos es un valor lo más cercano al 0, que indica un menor número de errores.

La matriz de confusión es otra métrica que ayuda a saber cómo ha clasificado el algoritmo para un conjunto de prueba y nos da una información relevante para así saber en cuales clases se equivoca más, algo que servirá para ver a nivel gráfico en una matriz el número de ejemplos que ha clasificado para cada clase. Estas matrices se mostrarán para algunos algoritmos y así se pueda hacer una comparación entre el mejor y el peor.

El tiempo que tarda el algoritmo en entrenar para el conjunto de entrenamiento es algo que también se verá a continuación, donde se comprobará como el uso de TensorFlow con aceleración por GPU consigue mejorar para entrenar y conseguir el modelo.

5.1. Valores obtenidos

Tras realizar el entrenamiento de los algoritmos y su evaluación con el conjunto de prueba se han obtenido los siguientes valores:

Clasificador	Ex(%)	Prec(%)	Rec(%)	Valor F(%)	LL	Te
KNN	96,72	96,71	96,72	96,71	0,13	5,87
ExtraT(std)	93,96	93,96	93,96	93,96	0,26	15,79
TF(delta)	93,44	93,43	93,44	93,73	2,27	67,95
ExtraT	93,63	93,62	93,63	93,60	0,27	16,15
TF(ada)	93,10	93,08	93,10	93,35	2,38	66,67
DecisionT(std)	93,19	93,19	93,19	93,19	2,35	5,65
DecisionT	93,14	93,15	93,14	93,15	2,37	7,47
TF(rms)	92,90	92,88	92,90	93,11	2,45	68,52
KNN(std)	93,00	92,98	93,00	92,98	0,43	6,09
TF(sgd)	92,17	92,19	92,17	92,59	2,70	66,40
MLP(std)	91,23	91,20	91,23	91,20	0,22	1052,17
TF(adam)	89,36	89,36	89,36	89,55	3,67	69,84
MLP	75,67	77,70	75,67	74,26	0,55	1270,74
Perceptron(std)	57,45	58,32	57,45	54,74	8,81	8,37
Perceptron	49,26	50,58	49,26	33,34	14,40	11,48
TF(adamax)	36,57	13,38	36,57	19,44	21,91	69,63
TF(nadam)	36,57	13,38	36,57	19,44	21,91	85,44

En la anterior tabla se muestran los datos obtenidos de las métricas que se han comentado previamente en el Capítulo 3. En ella se han incluido todos aquellos que han sido objeto de observación para este trabajo. Así aquellos que son precedidos con “TF” es para indicar donde se ha usado la librería de TensorFlow y con cada uno de los optimizadores que se han utilizado para comparar, con los parámetros por defecto. Estos optimizadores:

- Optimizador **SGD** : optimizador Stochastic Gradient Descent (R. S. Sutton & Barto, 2015).
- Optimizador **RMS**: optimizador RMSprop, el cual divide el gradiente a través de una media de su magnitud reciente.
- Optimizador **Adagrad**
- Optimizador **Adadelta:** una extensión más robusta de Adagrad
- Optimizador **Adam**
- Optimizador **Adamax**
- Optimizador **Nadam**(Dozat, n.d.)

El resto de algoritmos que se encuentran son los que se han instanciado usando la librería de scikit-learn. Se indican los valores obtenidos con los datos sin estandarizar y estandarizados, aquellos que tienen añadido “std”, para comprobar cómo afecta.

La tabla ha sido ordenada tomando como referencia el valor F(%) ya que se basa en la media ponderada entre la precisión y la recuperación, dos valores que también se tienen en cuenta.

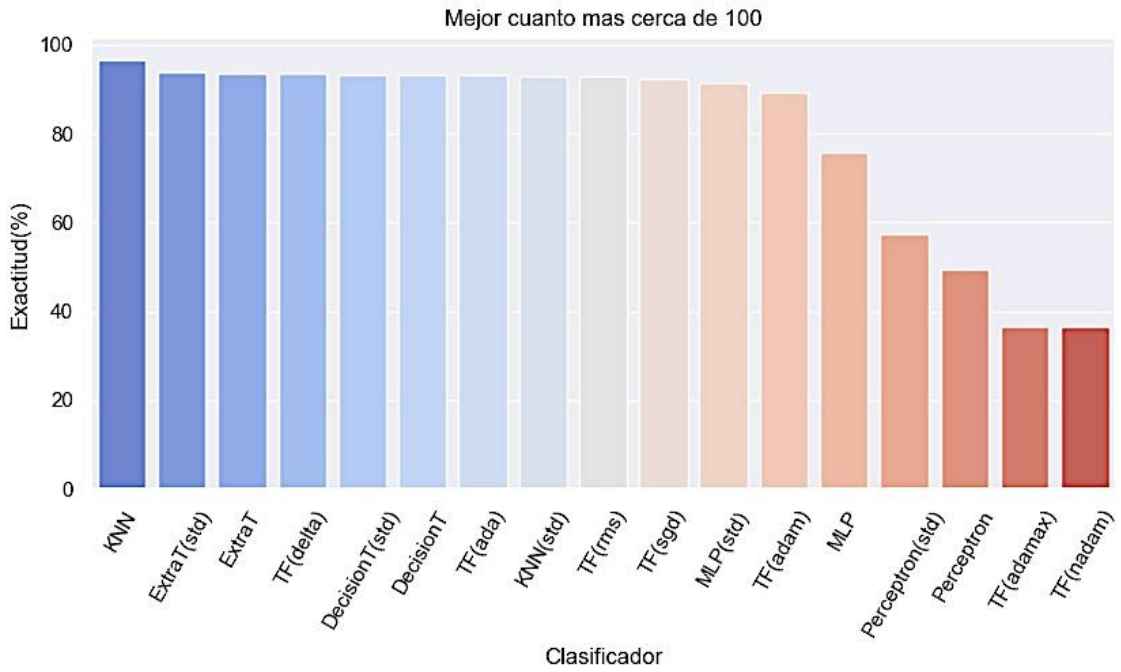
Se puede comprobar como el algoritmo KNN es el que ha obtenido mejor resultado con casi un 97% en valor F en el conjunto de datos de testeo, algo bastante bueno y que da bastante fiabilidad a la hora de clasificar.

Si atendemos a la estandarización de los datos, se puede observar cómo hay algunos algoritmos que mejoran y otros no. Así por ejemplo para KNN con los datos sin estandarizar se obtiene una Exactitud de un 96,72% mientras que si se estandarizan baja al 93%. Sin embargo, si se toma de referencia el Perceptrón multicapa (MLP) se puede observar como con los datos estandarizados mejora un 15% en dicha exactitud, algo a tener en cuenta si tomamos un conjunto de datos mucho más grande.

5.2. Gráficas

A continuación se van a analizar las distintas métricas que se han obtenido y se muestra en formato gráfica para una mejor visualización y comparación de los datos obtenidos:

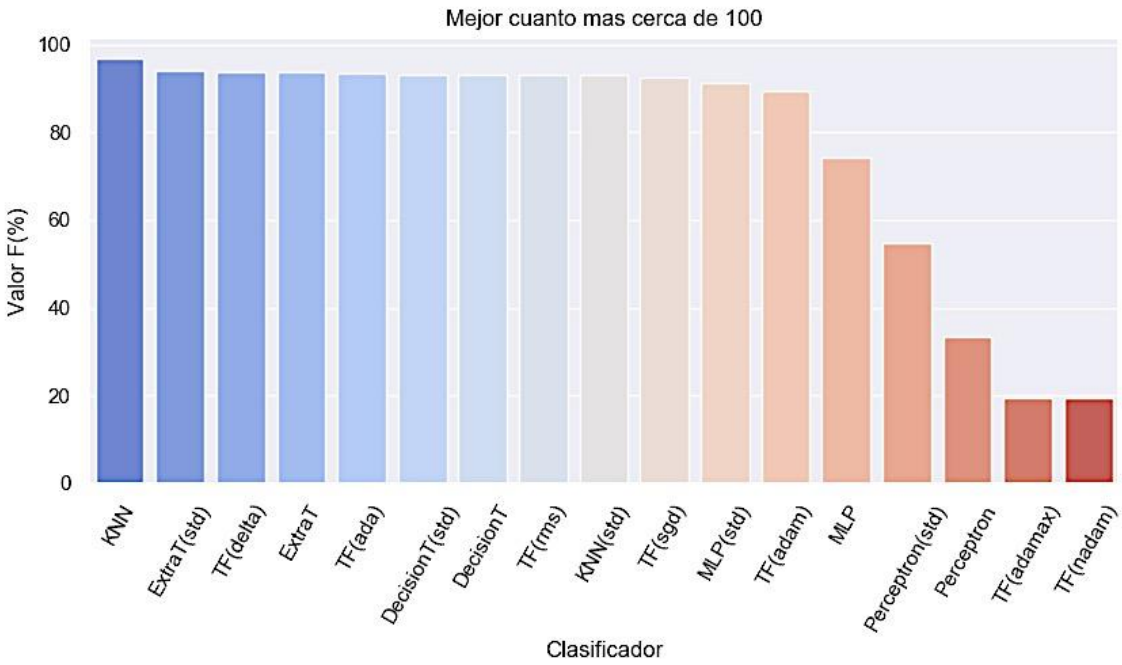
Exactitud



Analizando la anterior gráfica con respecto a la Exactitud, se puede decir que el mejor es el clasificador KNN seguido de ExtraTrees con los datos normalizados. En esta gráfica se observa sobre todo la diferencia que existe entre el uso de los datos estandarizados o no para el Perceptrón multicapa (MLP) y cómo mejora la exactitud desde el Perceptrón simple al multicapa, con una diferencia de un 33% para los datos estandarizados.

Si se tiene en cuenta para TensorFlow se puede observar cómo la selección del optimizador influye, así por ejemplo para TF(delta) se obtiene un 4% de diferencia que para TF(adam), aunque no parezca demasiado es una diferencia importante cuando hablamos de clasificaciones de miles de datos.

Valor F

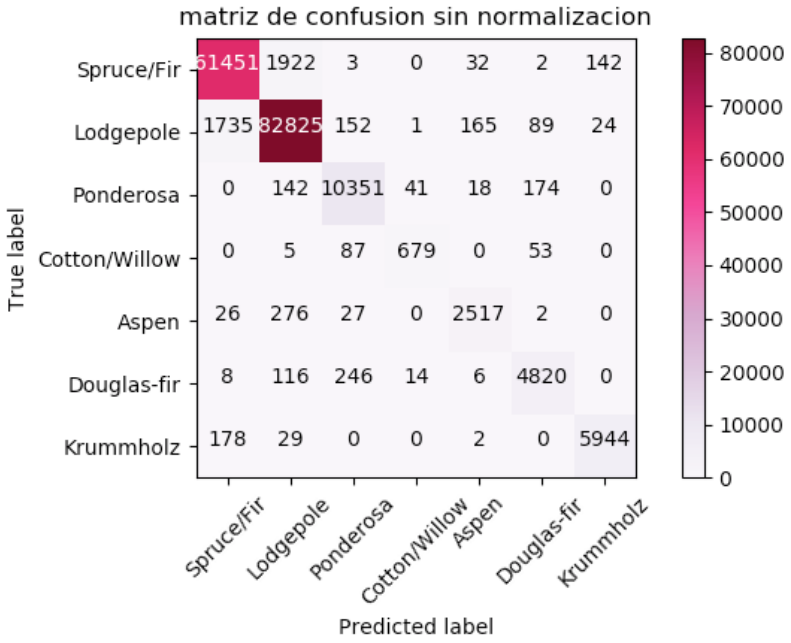


En aquellas distribuciones de datos en las que no se encuentran las clases balanceadas (mismo número de cada clase) es adecuado tener en cuenta métricas como la Precisión o la Recuperación. Por ello el uso del valor F consigue atender ese problema ya que es la media entre la Precisión y la Recuperación, y además tienen también en cuenta los falsos negativos y los falsos positivos que el modelo ha predicho.

Observando la gráfica se puede ver como KNN consigue la mejor puntuación seguido de ExtraT (árboles de decisión aleatorios). Si se compara con el valor obtenido para las redes neuronales, en la mayoría de los casos en TensorFlow se consigue mejor puntuación, scikit-learn MLP solo supera a TensorFlow con el optimizador Adam.

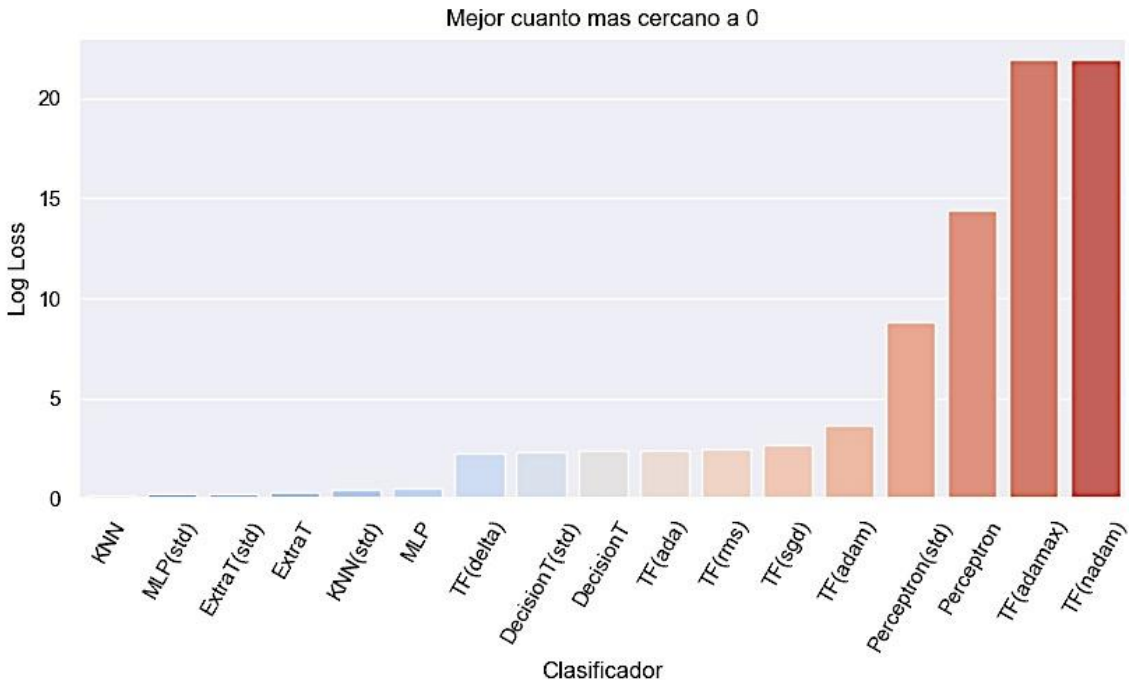
En el mismo gráfico se puede observar como con los datos estandarizados en scikit-learn los valores son siempre mejores, donde por ejemplo para el perceptrón multicapa en scikit-learn (MLP) hay una diferencia de casi el 17% para el valor F entre los datos estandarizados y sin hacerlo.

En la siguiente imagen se puede ver la matriz de confusión para el algoritmo KNN:



En dicha imagen se puede observar cómo de bien ha clasificado el algoritmo. Las filas muestran las etiquetas que corresponden, mientras que las columnas son las que el algoritmo ha predicho. Lo que se busca es que solo existan números en la diagonal, lo que indicaría un clasificador perfecto. Esto sirve también para observar qué clases le producen mayor confusión, debido a atributos parecidos que lleven a eso.

Log Loss



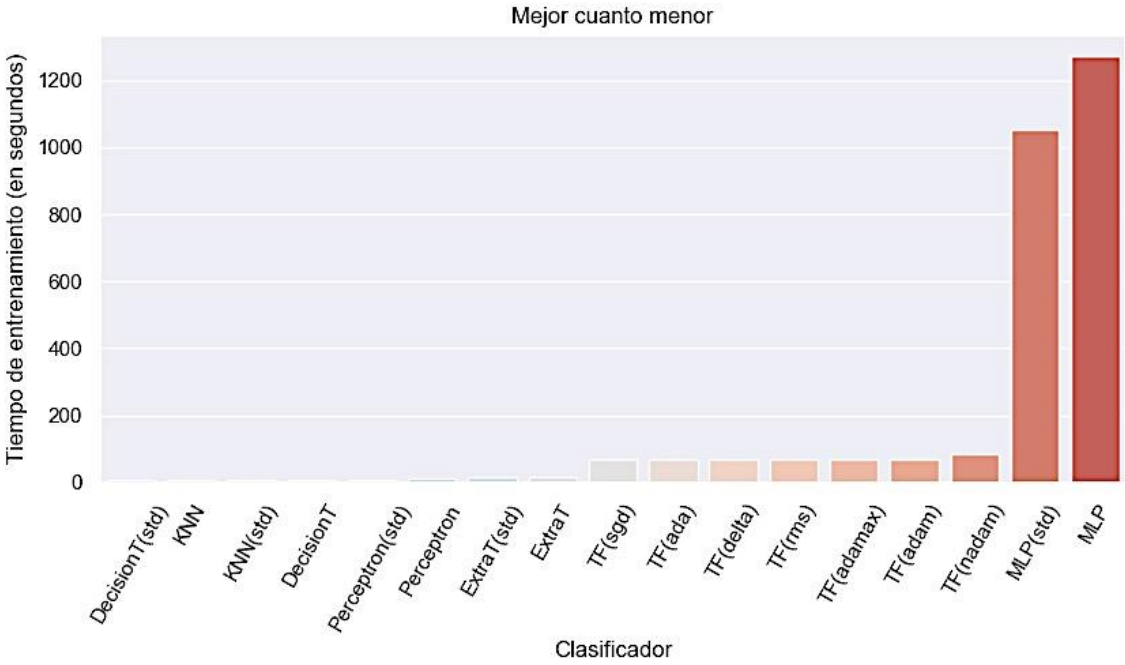
Esta métrica se puede decir que es de las más importantes cuando se quiere basar el criterio en probabilidades, es una métrica que se referencia en la función probabilidad. Ésta se encarga de decir “¿cómo de probable creía el

modelo que era el conjunto de testeo observado?”, es decir, ¿cómo de seguro se mostró el modelo para predecir cada uno de los ejemplos del conjunto de testeo?. En palabras más técnicas, se puede decir que es una transformación logarítmica de la suma de las probabilidades que el modelo asigna a cada ejemplo de forma errónea.

Para la gráfica anterior se puede observar que para el algoritmo KNN y para MLP con los datos estandarizados se consiguen valores de 0,13 y 0,22 respectivamente. Si comparamos con los valores obtenidos en TensorFlow se puede observar una diferencia considerable pues en TF(delta) se observa un LogLoss de 2,27, que indica menor confianza en dicha librería usando los optimizadores por defecto.

Estos valores se pueden mejorar si se realiza una búsqueda exhaustiva de cada uno de los hiper-parámetros para el menor valor posible de esta métrica.

Tiempo de ajuste



En esta gráfica se puede observar la diferencia existente entre el tiempo de ajuste que cada algoritmo necesita para entrenar y la obtención de un modelo atendiendo al patrón que siguen los datos que se les han proporcionado, los datos de entrenamiento. Así se puede observar la diferencia que existe entre ellos, por ejemplo tiempos muy cortos de entrenamiento para los algoritmos basados en árboles y el KNN, esto es debido a que la

clasificación la realizan en tiempo de ejecución y es por ello que son tiempos muy cortos, de apenas unos segundos.

Sin embargo sí que se observa diferencia entre los algoritmos basados en redes neuronales. Para TensorFlow con optimizador SGD (Stocasthic Gradient Descent), donde existe computación GPU, el tiempo se reduce en 1200 segundos con respecto a scikit-learn MLP. El uso de GPU para redes neuronales influye mucho, debido a que para el cálculo del gradiente descendiente o la regla delta de retropropagación de las redes neuronales requiere de numerosas multiplicaciones matriciales, algo para lo que las tarjetas gráficas están diseñadas y optimizadas.

Adecuado mencionar como el tiempo en scikit-learn MLP decrece en 200 segundos con los datos estandarizados o normalizados, afectando también al resto de las métricas, donde se observa mejora significativa.

6

Conclusiones

En este capítulo se evalúa el trabajo realizado y los objetivos conseguidos, incluyendo una valoración personal. También se incluyen posibles mejoras y ampliaciones que podrían desarrollarse en el futuro.

Objetivos

Para cerrar esta memoria, se va a hacer una retrospectiva de los objetivos fijados al inicio de esta, indicando para cada uno de ellos en qué grado se ha cumplido:

- **Aprendizaje uso librerías aprendizaje automático.** Durante este trabajo se ha realizado un estudio y uso en profundidad de las librerías de aprendizaje automático en Python que se han utilizado para su desarrollo, así como los distintos aspectos necesarios para su implementación tanto para scikit-learn como para TensorFlow.
- **Estudio del conjunto de datos.** Se han analizado distintos conjuntos de datos disponibles de varios repositorios, donde se han tanteado distintos aspectos para llevar a cabo el estudio. Se ha realizado un análisis del estado del conjunto de datos así como su formateo para llevar a cabo la siguiente fase de preparación de los datos. Así se comprobó la estructura que tienen, la existencia o no de errores, la distribución de los mismos, y otros aspectos que influyen a la hora de entrenar y clasificar los algoritmos seleccionados. Así se realizó una interacción con los datos para mostrar en gráfica la distribución en clases y así saber cómo enfocar el entrenamiento de los algoritmos.

- **Preparación de los datos.** Se realizó un estudio para saber cuál es la distribución de los datos, y cuál debían tener, algo que se implementó en los programas desarrollados tanto para scikit-learn como para TensorFlow con la correspondiente normalización de los mismos para que estuvieran en la misma escala. Además se estudió la forma de distribuir adecuadamente los datos de entrenamiento y de testeo para que la distribución de todas las clases fuera la más equitativa posible, y de esta forma los algoritmos aprendieran un modelo general para todo el conjunto de datos, algo que se consiguió con el uso de *cross-validation* y que en TensorFlow se solucionó con una mezcla de los mismos antes de proceder al particionado.
- **Estudio y evaluación de los algoritmos en scikit-learn.** Se desarrolló un programa para llevar a cabo el proceso de tratamiento de los datos, uso de cross-validation, entrenamiento de varios algoritmos (con estudio especial para redes neuronales), extracción de métricas y guardado de las mismas en formato CSV para su posterior importación y visualización en gráficas.
- **Estudio y evaluación de TensorFlow, su uso con keras y optimizadores.** Se desarrolló un programa para entrenar y clasificar el mismo conjunto de datos. En esta ocasión haciendo uso de la librería TensorFlow, donde se hizo uso exclusivo de rendimiento gráfico GPU con el entorno de desarrollo Google Colaboratory. Éste permitió extraer métricas para distintos optimizadores que iteran y ajustan los pesos para una red neuronal creada para clasificar con la librería keras. Se observó que el tiempo de cómputo es mucho menor que el uso de redes neuronales en sciki-learn, donde solo se usa computación CPU, de propósito más general y sin optimización para el cálculo matricial en el que se basan las redes neuronales.
- **Extracción de métricas y visualización de gráficas.** Una vez realizado el entrenamiento de los algoritmos tanto en scikit-learn como TensorFlow, se realizó una importación de los datos desde el archivo CSV correspondiente para así mostrar las métricas que se comentaron en el Capítulo 3. Se mostraron las gráficas correspondientes y se analizaron los resultados obtenidos en cada una de ellas.

Valoración personal

Durante el desarrollo de este trabajo se ha podido comprobar el funcionamiento del aprendizaje automático en dos librerías que hacen uso de esta rama de la inteligencia artificial y cómo se extraen distintas métricas dependiendo de la técnica que se use. Dependiendo de cómo se "alimente" al algoritmo a la hora de entrenarlo, por ejemplo si los datos están normalizados en escala o no, se ha podido comprobar que el rendimiento cambia.

Se ha podido comprobar como la elección de un algoritmo u otro puede dar lugar a diferente rendimiento. Se ha observado que las redes neuronales tienen un potencial importante y que, adecuadamente implementadas, pueden usarse para clasificar conjuntos de datos grandes. Así se pueden tomar como referencia a la hora de tomar decisiones o de predecir datos, así encontramos por ejemplo su uso para la predicción de la quiebra bancaria a través del uso de redes neuronales (Serrano Cinca, 1993), o por ejemplo para la predicción del tráfico (Torres Alvarez, Hernández, & Pedraza, 2011). Si contamos con conjuntos de datos muy grandes, el tiempo de computo necesario para que la red clasifique es importante, y por ello la opción de su implementación usando TensorFlow ayudará a conseguir el modelo 10 veces más rápido que si se implementa la misma red neuronal con la librería scikit-learn pero con cómputo en CPU, algo considerable a tener en cuenta.

El aprendizaje automático está muy presente en nuestros días y se usa en muchos campos. En la era digital en la que vivimos estamos siendo clasificados continuamente. Un ejemplo podemos encontrarlo en la clasificación que los bancos hacen del perfil de cada cliente para así saber cómo enfocar las campañas de marketing que les ayudarán a mejor éxito en ellas, así vemos un ejemplo en la competición que el grupo Banco Santander lanzó en el año 2018 en el portal Kaggle[7]. Este ejemplo también se ha podido ver con el escándalo de Cambridge Analytica[8] donde hace uso de aprendizaje

[7] https://www.kaggle.com/c/santander-value-prediction-challenge/overview
[8] http://theconversation.com/how-cambridge-analyticas-facebook-targeting-model-really-worked-according-to-the-person-who-built-it-94078

automático para clasificar la información recopilada a través de encuestas en distintas redes sociales y que ayudan a direccionar una campaña política.

Como resumen, no podemos dar de lado este tipo de inteligencia artificial pues está más presente de lo que pensamos y que forma parte de nuestras vidas. Adecuadamente usada y adaptada a nuestras necesidades el aprendizaje automático será capaz de facilitar ciertas tareas que ayuden a mejorar nuestro día a día.

Futuras mejoras

Las posibilidades que ofrece el aprendizaje automático son inmensas. Siguiendo la línea de este trabajo, enfocado en redes neuronales, a continuación se describen algunas posibilidades que su continuación nos ofrece.

En primer lugar, una de las posibilidades es la de comparar el comportamiento y rendimiento con los competidores de TensorFlow, como pueden ser Pytorch[9], CNTK (Microsoft Cognitive Toolkit)[10] o Apache MxNET[11]. Otra opción sería la de comprobar el funcionamiento y rendimiento de las posibilidades que ofrecen las grandes compañías en computación para aprendizaje automático, como pueden ser Amazon AWS o Microsoft Azure.

Debido a que TensorFlow está pensado en procesamiento gráfico, otra de las posibilidades de mejora sería el uso de la librería para saber el comportamiento que tiene para clasificación de imágenes, algo muy demandado hoy en día por ejemplo para la investigación e implantación del coche autónomo (Bojarski et al., 2017).

Otra posibilidad sería la de realizar una búsqueda exhaustiva de los hiperparámetros de los optimizadores en TensorFlow, así conseguir un mejor ajuste y mayor confianza en el clasificador, algo que se puede extrapolar a otro tipo de problemas de clasificación y conjuntos de datos disponibles.

[9] https://pytorch.org/
[10] https://github.com/microsoft/CNTK
[11] https://mxnet.apache.org/

Bibliografía

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zheng, X. (2016). TensorFlow : A System for Large-Scale Machine Learning This paper is included in the Proceedings of the TensorFlow : A system for large-scale machine learning.

Anuradha, B., & Reddy, V. C. V. (2008). ANN FOR CLASSIFICATION OF CARDIAC ARRHYTHMIAS, *3*(3), 1–6.

Arlot, S., & Celisse, A. (2009). A survey of cross-validation procedures for model selection, *4*, 40–79. https://doi.org/10.1214/09-SS054

Awad, W. A., & ELseuofi, S. M. (2011). Machine Learning Methods for Spam E-Mail. *International Journal of Computer Science & Information Technology*, *3*(1), 173–184.

Balakrishnan, D., & Puthusserypady, S. (2016). Multilayer perceptrons for the classification of brain computer interface data. *Proceedings of the IEEE 31st Annual Northeast Bioengineering Conference, 2005.*, (November), 118–119. https://doi.org/10.1109/NEBC.2005.1431953

Blackard Jock, A., Dean Denis, J., & Anderson, C. (1998). Cover Type repository. Retrieved October 20, 2017, from https://archive.ics.uci.edu/ml/datasets/covertype

Bojarski, M., York, N., Yeres, P., Firner, B., Muller, U., Choromanaska, A., & Jackel, L. (2017). Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car, 1–8.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., … Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project, 1–15. Retrieved from http://arxiv.org/abs/1309.0238

Cha, S., & Tappert, C. (2009). A Genetic Algorithm for Constructing Compact Binary Decision Trees, *1*, 1–13.

Chakraverty, S., Sahoo, D. M., & Mahato, N. R. (2019). *Perceptron Learning Rule. In: Concepts of Soft Computing*. Singapore.

Chen, W. Y., Chen, S. H., & Lin, C. J. (1996). A speech recognition method based on the sequential multi-layer perceptrons. *Neural Networks*, *9*(4), 655–669. https://doi.org/10.1016/0893-6080(95)00140-9

Cruz, J. A., & Wishart, D. S. (2006). Applications of machine learning in cancer prediction and prognosis. *Cancer Informatics*, *2*, 59–77. https://doi.org/10.1177/117693510600200030

Cuenca, D. (2017). Filtrado de SPAM en SMS mediante algoritmos de aprendizaje automático, 109–117.

Daniel Smilkov, Shan Carter, A. K. (2017). Simulation of Neural Network using Tensorflow library.

Dayan, P. (2009). Unsupervised learning. *The MIT Encyclopedia of the Cognitive Sciences*, 1–7. https://doi.org/10.1007/BF00993379

Dozat, T. (n.d.). Incorporating Nesterov Momentum into Adam.

Figini, S., Pavia, U., Felice, V. S., Pavia, I.-, & Maggi, M. (2014). Performance of credit risk prediction models via proper loss functions, *64*(January).

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees, (June 2005). https://doi.org/10.1007/s10994-006-6226-1

Ghahramani, Z. (2004). Unsupervised Learning BT - Advanced Lectures on Machine Learning. *Advanced Lectures on Machine Learning*, *3176*(Chapter

5), 72–112. https://doi.org/10.1007/978-3-540-28650-9_5
Hormozi, H., Hormozi, E., & Nohooji, H. R. (2012). The Classification of the Applicable Machine Learning Methods in Robot Manipulators. *International Journal of Machine Learning and Computing*, *2*(5), 560–563. https://doi.org/10.7763/IJMLC.2012.V2.189
Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Appears in the International Joint Conference on Articial Intelligence (IJCAI)*, *5*, 1–7. https://doi.org/10.1067/mod.2000.109031
Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, *31*, 249–268. https://doi.org/10.1115/1.1559160
Kumar, S. S., & Duraipandian, N. (2013). Artificial Neural Network Based Method for Classification of Gene Expression Data of Human Diseases along with Privacy Preserving Email : Sathish_tri@yahoo.com, *4*(2), 722–730.
McCulloch, W. S., & Pitts, W. (1943). A Logical Calculus of the Idea Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133. https://doi.org/10.1007/BF02478259
Multicapa, P. (1969). Perceptron Multicapa, 1–49. Retrieved from http://bibing.us.es/proyectos/abreproy/12166/fichero/Volumen+1+-+Memoria+descriptiva+del+proyecto%252F3+-+Perceptron+multicapa.pdf
Payá, Y. J., & Villalón, A. (2015). Grado en Matemática Computacional Redes neuronales . Un modelo de clasificación para la detección de dominios DNS maliciosos .
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2012). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. https://doi.org/10.1007/s13398-014-0173-7.2
PortalTic. (2017). Cada día se generan 2.500 millones de GB de datos: IBM crea una plataforma para que las empresas los aprovechen. *EuropaPress*. Retrieved from https://www.europapress.es/portaltic/internet/noticia-cada-dia-generan-2500-millones-gb-datos-ibm-crea-plataforma-empresas-aprovechen-20170323162319.html
Raschka, S. (2015). *Python Machine Learning*. *Bangladesh Journal of Plant Taxonomy* (Vol. 22). https://doi.org/10.1007/s13398-014-0173-7.2
Rifkin, R., Klautau, A., & Org, K. (2004). In Defense of One-Vs-All Classification. *Journal of Machine Learning Research*, *5*, 101–141. https://doi.org/10.1007/BF00718004
Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in …. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533. Retrieved from http://dx.doi.org/10.1038/323533a0
Santra, a. K., & Christy, C. J. (2012). Genetic Algorithm and Confusion Matrix for Document Clustering. *International Journal of Computer Science*, *9*(1), 322–328. Retrieved from http://ijcsi.org/papers/IJCSI-9-1-2-322-328.pdf
Schapire, R. (2008). Machine Learning Algorithms for Classification, 6. https://doi.org/10.13140/RG.2.1.2044.4003
Serrano Cinca, C. (1993). Predicción de la quiebra bancaria a través del uso de

redes neuronales, *XXIII*, 153–176.
Sutton, O. (2012). Introduction to k Nearest Neighbour Classification and Condensed Nearest Neighbour Data Reduction. *Introduction to k Nearest Neighbour Classification*, 1–10.
Sutton, R. S., & Barto, A. G. (2015). Reinforcement Learning : An Introduction.
Torres Alvarez, N. S., Hernández, C., & Pedraza, L. F. (2011). Redes neuronales y predicción de tráfico, *V*(29), 90–97.