Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Clasificador de texto mediante técnicas de aprendizaje automático

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Christian Guardiola González

Tutor: Patricio Letelier Torres

Cotutora: Mercedes García Martínez

Curso: 2019-2020

Resumen

El aprendizaje automático es un área de la inteligencia artificial que está en auge en el ámbito empresarial. Esta área consiste en crear programas capaces de generar un comportamiento específico a partir de unos datos. El aprendizaje automático tiene múltiples aplicaciones, tales como: motores de búsqueda, diagnóstico médico, análisis del mercado de valores, juegos, reconocimiento del habla y del lenguaje escrito, coches autónomos, publicidad dirigida por historial de búsquedas o compras, etc.

Existe gran cantidad de información que se encuentra en internet de forma desordenada. En particular, hay empresas que se dedican a seleccionar y clasificar las noticias que muestran en su web de manera manual, lo que hace que no sea muy productivo, tanto por los recursos humanos necesarios como el tiempo necesario que requiere esta tarea, ya que es necesario leer todas las noticias y clasificarlas. Este TFG aborda este problema desarrollando un clasificador de texto, el cual se entrenará mediante diferentes algoritmos de aprendizaje automático en búsqueda del más eficiente. Además, se investigará cómo la cantidad de datos y la longitud del texto afecta al resultado. Para realizar dicho clasificador de texto se usará Python como lenguaje de programación ya que existen muchas librerías y módulos que facilitan su desarrollo. Además de este clasificador, se creará otro clasificador capaz de clasificar diferentes frases en un dominio específico, como puede ser legal, medicinal, etc.

Palabras clave: Aprendizaje automático, clasificador de texto, Python, algoritmos para aprendizaje.

Abstract

The machine learning area of artificial intelligence is obtaining a lot of importance in the business world. This area consists of creating programs capable of generating a specific behavior with the help of data. Machine learning has multiple applications, such as: search engines, medical diagnosis, stock market analysis, games, speech recognition and written language recognition, autonomous cars, directed advertising by search history or purchases, etc.

There is a large amount of information that we can find on the internet in a disorderly manner. In particular, there are companies dedicated to select and classify the news they show on their website manually, which makes it not very productive, both for the necessary human resources and the necessary time required for the task, since it is necessary to read all the news and classify them. This project addresses this problem by developing a text classifier, which will be trained by different automatic learning algorithms in search of the most efficient. In addition, we will investigate how the amount of data and the length of the text affect the result. To make the above mentioned text classifier, Python will be used as a programming language since there are many libraries and modules that facilitate its development. In addition to this classifier, we will create another one capable of classifying different phrases in a specific domain, such as legal, medicinal, etc.

Keywords: Machine learning, text classifier, python, learning algorithms.

Tabla de contenidos

Índice de figuras, tablas y ecuaciones

Índice de Figuras

Índice de Tablas

Índice de Ecuaciones

1. Introducción

El aprendizaje automático (en inglés *machine learning*) es un área de la inteligencia artificial que está en auge en los últimos años en el ámbito empresarial. El aprendizaje automático consiste en crear programas que sean capaces de generar un comportamiento específico a partir de unos datos. Debido a esto, se ha abierto un nuevo abanico de oportunidades ya que ofrece muchísimas opciones de uso, como pueden ser: Motores de búsqueda, diagnósticos médicos, análisis del mercado de valores, juegos, reconocimiento del habla y del lenguaje escrito, etc.

Uno de los ejemplos más emblemáticos en los últimos años en los que se usa el aprendizaje automático sería en el caso de los coches inteligentes que pueden conducir por su cuenta, estos coches usan diferentes tipos de algoritmos de aprendizaje automático, por ejemplo, cuando se encuentran en un sitio con mucho tráfico, el ordenador es capaz de predecir el comportamiento de los conductores, haciendo la conducción mucho más segura. Otro ejemplo actual es el uso de las máquinas de aprendizaje automático que usa Google para aprender a partir de tus búsquedas y poder ofrecer publicidad que se adapte a tus intereses.

1.1. Motivación

En estos momentos, existe gran cantidad de información que se encuentra en internet de forma desordenada y sin clasificar. Este problema hace que encontrar o seleccionar algo de toda esa información sea algo muy ineficiente [1], en particular podemos encontrar como ejemplo a diversas empresas que se dedican a seleccionar y clasificar las noticias que muestran en su web de manera manual, lo que hace que no sea muy productivo, tanto por los recursos humanos necesarios como el tiempo necesario que requiere esta tarea, ya que es necesario leer todas las noticias y clasificarlas.

Este trabajo de fin de grado se ha desarrollado en el marco de una práctica de empresa debido a un trabajo solicitado por una empresa, en la que hasta el día de hoy, tienen a periodistas especializados en innovación trabajando en clasificación de noticias, cosa que les ocupa mucho tiempo y dinero innecesario.

Este trabajo de fin de grado aborda este problema realizando un clasificador de texto, el cual se entrenará mediante diferentes algoritmos de aprendizaje automático para quedarnos con el más eficiente y así, no tener que depender de tantos recursos humanos para la clasificación de estas.

Esta solución no busca eliminar completamente ese puesto de trabajo, pues para entrenar una máquina, es necesario tener datos etiquetados con los que la máquina aprenderá a diferenciar. El trabajo de estas personas pasará de clasificar noticia a noticia, a la creación de un volumen de datos de noticias con el que entrenar a la máquina cada vez que se requiera que aprenda algo nuevo, el cual es un trabajo emergente gracias a la creación de dichos clasificadores y máquinas de inteligencia artificial, el cual cada vez está más en auge llamado *data labeling* [2]

Además, la clasificación de texto no sirve solo para recortar en gastos, nos ofrece diversas ventajas como se explica en [3], alguna de la ventaja más importante puede ser la protección de datos importantes, ya que el tener nuestros datos clasificados, hace que sea mucho más sencillo el decidir cómo debemos tratar con dichos datos.

Para la realización de nuestro clasificador, haremos uso del lenguaje de programación Python, ya que existen muchas librerías y módulos que facilitan la creación de este clasificador. Además, es uno de los lenguajes más usados en este ámbito y muchos ejemplos de uso que puedes encontrar se encuentran en este lenguaje de programación. Una de las librerías más importantes en las que nos vamos a apoyar es la librería Sklearn, la cual nos ofrece multitud de algoritmos pudiéndolos probar de manera sencilla y comprobar cuál de todos ellos nos conviene más usar obteniendo mejores resultados.

Además, comprobaremos diferentes datos de entrada para comprobar cómo actúan los algoritmos según el tipo de estructura de datos y la diferencia que existe a la hora de entrenar un modelo con diferente tamaño de datos.

Todo esto ayudará a automatizar la tarea y a disminuir el coste de la clasificación de texto en las empresas.

1.2. Objetivos

Los objetivos de este trabajo de fin de grado son:

1) Desarrollar un clasificador de textos estudiando los algoritmos de aprendizaje automático y estableciendo qué parámetros son los más eficientes para esta tarea.

2) Estudiar cómo la cantidad de datos y la longitud del texto (frases, párrafos, etc.) afectan el entrenamiento del modelo.

1.3. Estructura del trabajo

- Capítulo 2: Aprendizaje automático: Algoritmos y técnicas de clasificación.
    - Introducción: En esta parte del capítulo se realizará una pequeña introducción a los tipos de aprendizaje más importantes.
    - Aprendizaje supervisado: Esta segunda parte del capítulo hablaremos sobre el aprendizaje supervisado, cómo funciona y qué algoritmos hemos utilizado en nuestro proyecto.
    - Aprendizaje no supervisado: Esta tercera parte del capítulo hablaremos sobre el aprendizaje no supervisado, cómo funciona y uno de sus algoritmos más usado.
    - Aprendizaje por refuerzo: Por último, hablaremos sobre el aprendizaje por refuerzo y cómo funciona.
- Capítulo 3: Estado del arte.
    - En este capítulo se comentará el estado de los clasificadores de texto y las alternativas que existen a lo usado en nuestro proyecto junto a sus ventajas y desventajas.
- Capítulo 4: Tecnologías utilizadas
    - En este apartado hablaremos sobre todas las herramientas necesarias para realizar el proyecto y donde se pueden conseguir.
- Capítulo 5: Caso de estudio.
    - Preparación de datos: En este apartado se hablará de todos los pasos realizados para la preparación de los datos y el tipo de datos que hemos usado, esto se hará tanto para el clasificador de noticias como para el clasificador de dominio.
    - Entrenamiento del clasificador: En esta parte del capítulo se hablará de cómo hemos entrenado nuestros clasificadores y con qué algoritmos.
    - Evaluación de resultados: Este apartado es donde se hablará de los resultados que hemos obtenido y unas las conclusiones que hemos sacado de estos.
- Capítulo 6: Metodología del proyecto.
    - En este capítulo hablaremos sobre las diferentes fases por las que hemos pasado a la hora de realizar el proyecto y cuánto tiempo nos ha costado realizar cada una de las fases.

- Capítulo 7: Conclusiones y trabajo futuro
    - En este capítulo hablaremos sobre las conclusiones sacadas a partir de los resultados obtenidos en el caso de estudio, además hablaremos sobre el trabajo qué trabajo futuro puede realizarse para extender el trabajo realizado.

2. Aprendizaje automático

En este apartado hablaremos sobre los 3 tipos de aprendizaje más importantes en el aprendizaje automático y sobre los algoritmos más importantes usados en el proyecto.

2.1. Introducción

Los sistemas de aprendizaje automático son sistemas que intentan simular el proceso que realizamos los humanos a la hora de realizar una tarea. Para conseguirlo es necesario entrenar un modelo o, dicho de otro modo, hacerle aprender ese comportamiento o procesamiento. Esto lo conseguimos gracias a unos datos de entrenamiento que la máquina utilizará para aprender dicho procedimiento.

Según la información aportada en los datos existen diferentes tipos de aprendizaje, además, según el tipo de aprendizaje, podemos utilizar diferentes técnicas de clasificación según se adapte mejor a nuestro problema.

A la hora de entrenar el modelo, los datos pueden tener una etiqueta con la categoría que corresponde al segmento de texto que tenga el documento, este tipo de aprendizaje con datos ya etiquetados se llama aprendizaje supervisado.

También es posible entrenar con datos que no estén etiquetados, en el cual es el sistema el encargado de reconocer patrones en los datos para ser capaz de etiquetar las nuevas entradas, este tipo de aprendizaje se denomina aprendizaje no supervisado.

Por último, existen tipos de aprendizaje que suelen aprender mediante prueba y error, es decir, aprende gracias a la retroalimentación que recibe de sus acciones. Este tipo de aprendizaje se suele denominar aprendizaje por refuerzo.

Aunque existen otros tipos de aprendizaje, estos 3 que acabamos de comentar son los más comunes.

2.2. Aprendizaje supervisado

El aprendizaje supervisado, es aquel que trabaja con datos etiquetados. Este tipo de aprendizaje intenta encontrar una forma de reconocer similitudes entre los datos con la misma etiqueta para ser capaz de diferenciar entre los datos con etiquetas diferentes, y de esta manera, conseguir clasificar datos que no tienen etiqueta asignándoles una de las diferentes categorías [4].

Estos datos de entrenamiento tienen la siguiente estructura: dados unos datos $(x_1,y_1),\ldots,(x_n,y_n)$ donde $x = \{x_1, x_2, \ldots, x_D\}$, N es el número de ejemplos, X representa las características siendo D las dimensiones y por último Y representa las etiquetas. Un algoritmo de aprendizaje trata de obtener una función $G: X \rightarrow Y$, donde X es la entrada e Y es la salida [5].

En este tipo de aprendizaje, se diferencian 2 categorías según el tipo de etiqueta: clasificación y regresión.

Clasificación: En esta categoría del aprendizaje supervisado, el algoritmo se centra en clasificar los datos en diferentes categorías específicas. Los algoritmos intentan definir a qué categoría pertenece los datos de entrada a clasificar según las características de estos.

Regresión: En esta categoría del aprendizaje supervisado, en vez de clasificar los datos en diferentes categorías, las etiquetas suelen ser un valor numérico y el algoritmo consiste en intentar aproximar a un número según las características de los datos de entrada [6].

Debido a todas estas características, el aprendizaje supervisado suele ser usado en problemas sobre la clasificación como identificación de la categoría de una noticia, saber si un mensaje es spam o no, etc. También se suele usar en problemas de regresión como puede ser predicciones meteorológicas, detección del margen de error a la hora de traducir un texto, etc.

A continuación, hablaremos de 4 de los algoritmos más frecuentes en este tipo de aprendizaje: Máquinas de soporte vectorial, *naive bayes*, bosques aleatorios y regresión lógica.

2.2.1. Máquinas de soporte vectorial (SVM)

Las máquinas de soporte vectorial [7] (SVM del inglés *Support Vector Machines*) es un algoritmo de aprendizaje supervisado desarrollado por Vladimir Vapnik. Este algoritmo suele relacionarse tanto con problemas de clasificación como de regresión.

Una SVM se puede representar como puntos en un espacio, el objetivo de este algoritmo es crear un plano, llamado hiperplano, que se encarga de dividir estos puntos para crear particiones homogéneas.

El objetivo de este algoritmo es encontrar un hiperplano óptimo, es decir, siempre intentará que el hiperplano que separa las clases tenga el mayor margen entre este y los puntos más cercanos de cada clase. En la figura 1 podemos ver una representación de diversos hiperplanos creados en un espacio donde podemos comprobar como H1 y H2, aunque separan las clases, no las separa de forma óptima, sin embargo, H3 sí.

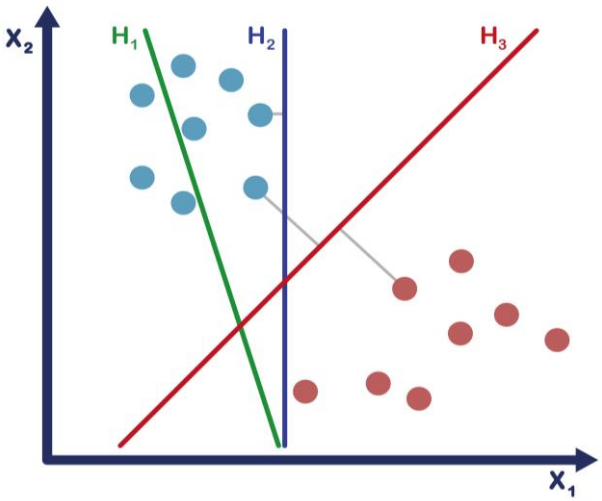

Figura 1: Diferentes hiperplanos en un SVM

El nombre de máquinas de soporte vectorial viene dado por los puntos más cercanos al hiperplano, donde cada clase debe tener por lo menos uno. Usando estos vectores de soporte, es posible definir el hiperplano de máximo margen.

Debido a que es posible que el hiperplano no sea capaz de separar todos los puntos de una clase de manera precisa, el algoritmo da un parámetro C que se encarga de compensar estos errores que aparecen en el momento de entrenamiento, esto permite algunos errores de clasificación a la vez que los penaliza (cuando usamos el parámetro para compensar errores se denomina SVC) [8].

Este algoritmo sirve para separar 2 clases, pero existen formas de usar este algoritmo para la clasificación de diversas clases, esto se consigue mediante el uso de la estrategia *one-vs-one* o *one-vs-all*. En nuestro proyecto hemos utilizado la estrategia *one-vs-one*, ya que es con la que trabaja la librería *sklearn* [9].

La forma más sencilla de crear un hiperplano es hacerlo de forma lineal. Sin embargo, puede ocurrir que no siempre sea la mejor solución para nuestros datos. Debido a las limitaciones de cómputo de las máquinas de aprendizaje lineal, no suelen usarse en problemas del mundo real. Sin embargo, existen soluciones a este problema, como pueden ser las representaciones por medio de funciones *kernel*, en la cual se proyecta la información a un espacio con gran dimensionalidad, esto permite que la capacidad computacional de dicha máquina de aprendizaje lineal aumente [10]. Según el tipo de *kernel* que usemos, se obtendrán diferentes hiperplanos como podemos observar en la figura 2.

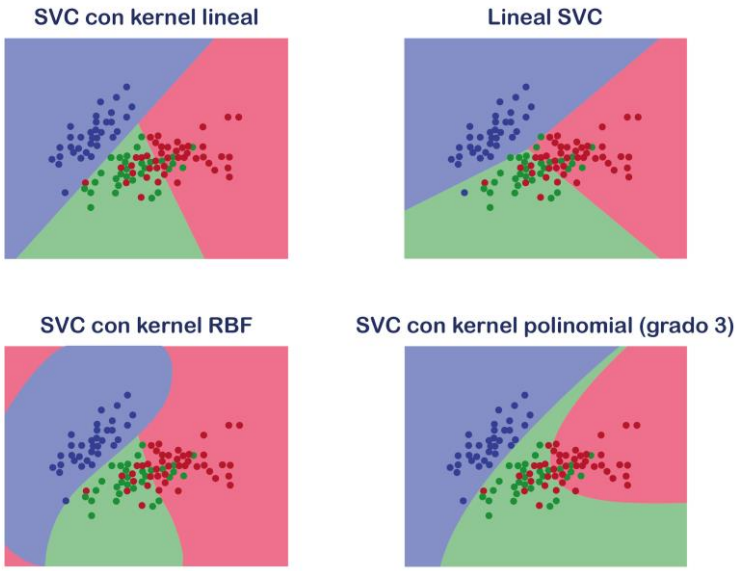

Figura 2: SVM con diferentes kernels

Este algoritmo suele destacar debido a que es más fácil de usar que las Redes Neuronales, además de no ser muy influenciable por los datos ruidosos y suele ajustarse bien a modelos relativamente complejos, sin embargo, tiene varias desventajas como el hecho que para encontrar el mejor hiperplano para separar los datos es necesario probar diferentes *kernels*, esto puede hacer que el modelo sea lento de entrenar.

2.2.2. Naive Bayes

Naive bayes [11] es un algoritmo el cual se basa en el Teorema de Bayes y los Métodos Bayesianos. Los clasificadores basados en estos métodos funcionan utilizando los datos para poder calcular la probabilidad de que dicho dato de entrada pertenece a la clase que lleva como etiqueta. Cuando el clasificador es aplicado a datos que no se encuentran etiquetados, este usa esas probabilidades para poder encontrar la clase más probable.

Los clasificadores Bayesianos suelen usarse cuando existen problemas donde es necesario considerar simultáneamente numerosos atributos de la información para poder estimar la probabilidad de dicho evento [12].

Naive bayes no es el único algoritmo que usa los métodos Bayesianos para poder clasificar, pero sí que es el más común a la hora de clasificar texto, donde se podría decir que se ha convertido en el algoritmo estándar.

En nuestro proyecto usaremos el algoritmo *Multinomial Naive Bayes*, ya que es el que más se adapta a nuestro clasificador y es uno de los 2 clásicos algoritmos de *Naive Bayes* que se usa en la clasificación de textos. Este algoritmo hace uso de la siguiente fórmula:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Ecuación 1: Naive Bayes

Donde θ_{yi} es la probabilidad del evento, N_{yi} es el número de veces que la característica i aparece en el conjunto de datos de la clase y, N_y es el total de las clases y, por último, el valor $\propto$ es un valor que varía de 0 a 1 y sirve para normalizar los resultados.

El nombre de *Naive Bayes*, más concretamente la parte de *Naive* (ingenuo), se debe a que el algoritmo trabaja con la idea de que las clases son totalmente independientes, lo cual es algo que raramente sucede.

Este algoritmo tiene diversas fortalezas y debilidades que pueden afectar a la hora de decidir si usarlo o no, las fortalezas más importantes se podrían resumir en que es un algoritmo rápido y simple capaz de manejar datos ruidosos, necesita relativamente pocos ejemplos a la hora de entrenar y es fácil obtener las probabilidades estimadas para una predicción. En lo que a debilidades se refiere, las más importantes serían que presupone que son igual de importante todas las clases, no suele ser óptimo para un conjunto de datos que tengan una gran cantidad de clases numéricas y las probabilidades estimadas son menos fiables que las clases que predice.

2.2.3. Bosques aleatorios

El algoritmo Bosques Aleatorios [15] fue desarrollado por Breitman y Cutlet en 2001. Se trata de un algoritmo que funciona utilizando diferentes modelos que al final se combinan para poder obtener un modelo de todo el conjunto.

El método que usa Bosques Aleatorios para generar diferentes modelos a partir de los datos se conoce como *boostrap aggregatting, o bagging*. Este método genera una serie de conjunto de datos de entrenamiento a partir de los datos originales mediante *bootstrap sampling,* para luego ser usados a la hora de entrenar un modelo único con cada uno de los datos de entrenamiento. Posteriormente, las predicciones de estos modelos son combinadas mediante votos si es un clasificador o mediante medias si es de regresión, podemos ver un ejemplo de este funcionamiento en la figura 3.

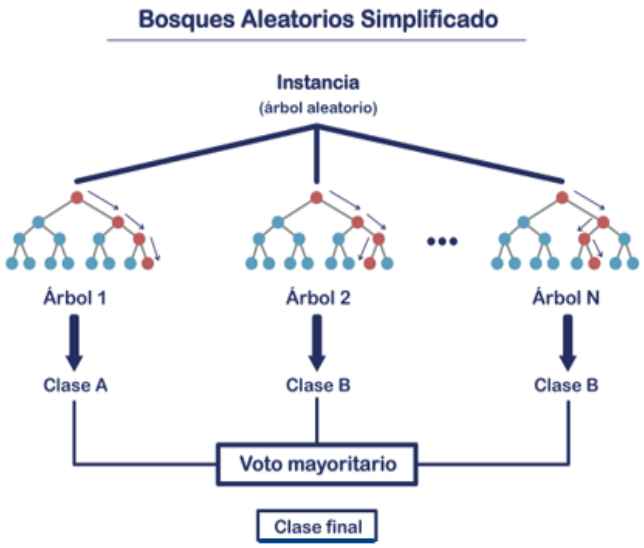

Figura 3: Ejemplo del funcionamiento de los bosques aleatorios

Este algoritmo tiene diversas fortalezas y debilidades. En lo que a fortalezas se refiere, podemos destacar que este modelo puede manejar datos ruidosos, se puede usar con datos que contengan muchas clases diferentes y es capaz de procesar una gran cantidad de datos en poco tiempo. Las debilidades más importantes de este algoritmo son que no es fácilmente interpretable y es difícil de ajustar a los datos.

2.2.4. Regresión logística

La regresión logística [13] es un algoritmo de clasificación que se suele utilizar para conocer la probabilidad de que una variable se encuentra en una categoría. En la regresión logística, la variable se trata de una variable binaria que tiene datos codificados como 1 o como 0.

Aunque este algoritmo sirve para la clasificación de 2 clases, *sklearn,* la librería que vamos a usar en nuestro proyecto nos permite utilizar este algoritmo para clasificar un modelo con múltiples clases mediante el uso de la estrategia *one vs rest*. Esta estrategia consiste en entrenar un clasificador por cada clase, con el que el modelo los usará para comprobar si el dato de entrada es de dicha clase o no, ya que cada clasificador dará positivo si es de dicha clase (1) o negativo (0) si es de una de las demás clases. De esta forma, este algoritmo binario, se puede utilizar para clasificar más de 2 clases diferentes [14].

Aunque esta estrategia es popular a la hora de clasificar, tiene diversos problemas como, por ejemplo, debido a que los clasificadores solo ven como positivo los datos de su misma clase y negativo todos los demás, hay un desbalance a la hora de entrenar el clasificador que usará el modelo, ya que en el conjunto de datos de entrenamiento existirán muchos más negativos que positivos.

2.3. Aprendizaje no supervisado

El aprendizaje no supervisado es aquel que trabaja con datos no etiquetados, es decir, no hay un conocimiento a priori del tipo de datos. Este tipo de aprendizaje intenta encontrar una forma de reconocer similitudes entre los datos para así poder agruparlos según sus características.

Los datos de entrada tienen la siguiente estructura: Dados unos datos de entrada $(X_1 \dots X_N)$ donde X es un vector de D dimensiones o características y N es el número de ejemplos [5].

Debido al funcionamiento de este tipo de aprendizaje, el aprendizaje no supervisado suele usarse principalmente para agrupación o compresión de datos.

2.3.1. K-medias

Resumiendo lo contado en la tesis de Daniel Jiménez González [18], K-medias es un algoritmo que usa una serie de cálculos iterativos para llegar a obtener el mejor resultado posible, es decir, usa una técnica de refinamiento iterativo. El nombre de este algoritmo se debe a que la letra k se refiere al número de clúster, los cuales deben ser definidos, después de ser definidos, el algoritmo asigna y reparte los N ejemplos a su correspondiente clúster. Este agrupamiento se hace con la idea de minimizar las diferencias entre los ejemplos agrupados y aumentarlas entre los clústeres para que pueda haber una clasificación lo más clara posible.

Debido a que a menos que existan pocos clústeres y pocos ejemplos, es muy difícil conseguir los clústeres óptimos entre las diferentes combinaciones que pueden aparecer, por tanto, el algoritmo hace uso de una aproximación heurística que le servirá para conseguir las soluciones óptimas locales. Esto significa que siempre empieza con una estimación inicial, y con las consiguientes iteraciones modifica ligeramente la aproximación y comprueba si esa modificación mejora la homogeneidad dentro de cada clúster.

Para poder comparar esta homogeneidad, el algoritmo hace uso de un cálculo de distancias, en el que el algoritmo recoge la coordenada en el hiperplano y luego se calcula la distancia euclídea entre los puntos, es decir, a la hora de ejecutar el algoritmo se ejecutaría de esta forma.

Primero, se selecciona los k puntos al azar, los cuales harán de centro de clúster, una vez el algoritmo tenga dichos centros, cada punto se asigna al clúster más cercano. Esa distancia se calcula mediante un Diagrama de Voronoi el cual se genera a partir de los centros asignados.

$$S_i^{(t)} = \left\{ x_p : ||x_p - m_i^{(t)}|| \leq ||x_p - m_j^{(t)}|| \forall 1 \leq j \leq k \right\}$$

Ecuación 2: Diagrama de Voronoi

Seguidamente, como se ha mencionado antes, el algoritmo utiliza la distancia euclídea para comparar la homogeneidad de cada clúster.

$$d(x,y) = \sqrt{\sum_i^n (x_i - y_i)^2} = \sqrt{\sum_i x_i^2 + \sum_i y_i^2 - 2\sum_i x_i y_i}$$

Ecuación 3: Fórmula para calcular la distancia euclídea

Por último, una vez que se haya terminado de asignar cada punto, el algoritmo pasa a actualizar. En lo que consiste esta actualización es cambiar la localización de los k puntos a otra localización. Esta localización se calcula consiguiendo la posición media de los puntos asignados a cada clúster, esta localización se denomina centroide.

$$m_i^{(t+1)} = 1/|S_i^{(t)} \sum_{x_j \epsilon S_i^{(t)}} x_j$$

Ecuación 4: Media para encontrar el centroide

Debido a todo esto, el algoritmo tiene ventajas como pueden ser que es muy flexible y puede ser adaptado con ajustes sencillos y se comporta de forma bastante competente en la mayoría de los problemas del mundo real, aunque también tiene diversas desventajas como que no es tan sofisticado como otros algoritmos de *clustering* moderno y no suele ser muy eficiente para clústeres no esféricos o con una densidad variable. Aun con todo esto, es uno de los algoritmos de clúster más populares.

2.4. Aprendizaje por refuerzo

El aprendizaje por refuerzo [16] es un tipo de aprendizaje que se usaba hace mucho tiempo en trabajos de estadística, neurociencia e informática, pero desde hace poco tiempo ha ganado un gran interés en el entorno del aprendizaje automático. Este tipo de aprendizaje es una forma de programar agentes a través de recompensas y castigos sin necesidad de especificar desde un principio como cumplir la tarea a aprender.

En el modelo de aprendizaje por refuerzo, el agente en cada iteración recibe un *input* "I", un indicador del estado actual del entorno "S", con los que el agente decide una acción "A" para poder generar la salida. Esta acción cambia el estado del entorno y el valor de este estado se comunica al agente a través de una señal que indica si el cambio es bueno o malo (Recompensa o castigo) "R". El agente con el tiempo elegiría las acciones que aumentase dicha señal y así, poco a poco, iría aprendiendo mediante prueba y error.

Debido a esto, el aprendizaje por refuerzo suele usarse para enseñar a máquinas a jugar juegos como puede ser el ajedrez o robots para que tengan un comportamiento específico.

2.5. Conclusión

Como conclusión, existen diversas maneras de entrenar un modelo y según cual sea tu objetivo, será mejor utilizar un tipo de aprendizaje u otro y según la complejidad o tamaño de tus datos será mejor utilizar un tipo de algoritmo u otro.

Debido a que nuestro modelo se ha entrenado con datos etiquetados, hemos utilizado el aprendizaje supervisado, con el que hemos entrenado nuestro modelo con los 4 algoritmos explicados anteriormente: Máquinas de soporte vectorial (tanto con kernel lineal como con kernel rbf), *naive bayes*, bosques aleatorios y regresión lógica.

3. Estado del arte: clasificadores de texto

En estos momentos, la clasificación de texto es un tema de gran interés, muchas empresas están haciendo uso de clasificadores automáticos para poder reunir información sobre textos o mensajes escritos en plataformas como Twitter, Facebook, etc. Estos clasificadores pueden llegar a saber cosas como: si un mensaje es racista, si un texto habla sobre política o habla sobre tecnología, si un comentario en Twitter sobre una conferencia es positivo o negativo, el idioma en el cual un texto está escrito, etc.

En nuestro caso se ha requerido que se realicen 2 clasificadores específicos, los cuales han sido: un clasificador de noticias y un clasificador de dominio.

A la hora de realizar nuestro proyecto se ha utilizado Python como lenguaje de programación, ya que este lenguaje ofrece la mayor variedad de librerías en C que sirven como ayuda a la hora de crear nuestros clasificadores y todas las nuevas tecnologías de clasificación como se puede comprobar con Bert, el nuevo método de preentrenamiento de modelos de *Google* el cual a la hora de entrenar comprueba el contexto de forma bidireccional [17]. Además del lenguaje de programación Python, también se ha hecho uso de la librería Scikit-Learn [15], la cual, como hemos comentado anteriormente, ayuda enormemente a la hora de crear los clasificadores, ya que nos ofrece todo tipo de algoritmos para poder entrenarlos.

Además de lo usado en nuestro proyecto, estos clasificadores se pueden crear en otros tipos de lenguajes. Podemos encontrar clasificadores en java como el clasificador lingüístico de textos [19], en el cual unos estudiantes de la universidad de Madrid crearon un clasificador que tiene como entrada un texto y como salida el idioma en el que está escrito. También podemos usar el lenguaje de programación C++ con ciertas librerías como MeTA [20], la cual nos da acceso a diferentes algoritmos de la misma manera que Scikit-Learn nos permite hacer en Python. Además de estos 2 lenguajes de programación orientados a objetos, también podemos encontrar clasificadores realizados con Matlab, el cual junto a herramientas como *Text Analytics Toolbox [21]* es capaz de analizar texto y crear modelos de clasificación con diferentes algoritmos.

Aún con todas estas posibilidades en lo que a lenguaje de programación se refiere, nos hemos decantado por Python gracias a la gran información y facilidades que ofrece.

No obstante, además de programar nosotros el modelo, también existen alternativas online que nos ofrecen el poder crear un clasificador de texto sin escribir ni una sola línea de código, estos servicios online han sido creados tanto por Amazon como por Microsoft, en los que podemos encontrar Microsoft Azure por parte de Microsoft y Amazon Machine Learning por parte de Amazon. Estos servicios funcionan como una caja negra, en la cual te encargas de introducir los datos de entrada etiquetados con los que quieres que tu clasificador aprenda, y Amazon o Microsoft se encargan de crear un modelo con el cual podrás clasificar los datos que necesites. La

gran ventaja de todo esto, como es obvio, es la comodidad y la facilidad de creación del modelo de clasificación, pero tiene una gran desventaja, y es que son servicios de pago, además de que no son reutilizables, es decir, una vez realizado nuestro proyecto, si queremos actualizarlo con más ejemplos lo único que necesitaríamos es ponerlo en marcha y reentrenarlo, pero en el caso de Microsoft y de Amazon deberíamos de volver a pagar todo el tiempo necesario para volver a entrenar el modelo.

Por ejemplo, en el caso del servicio ofrecido por Mircrosoft, si nos fijamos en la tabla de precios de la página oficial de Microsoft Azure[1], en el proceso de alto rendimiento nos encontramos diferentes precios pero el que más se acercaría a lo utilizado en este trabajo sería la instancia H16 que cuenta con 16 núcleos y 112GB de Ram, con un coste de 2,282€ por hora, teniendo en cuenta que el entrenamiento que realizamos para todos los algoritmos y los diferentes tamaños de ejemplos utilizados tuvo una duración aproximado de un mes, el precio de haberlo utilizado con este método habría sido aproximadamente de 1700€. Este precio claramente solo es una suposición ya que para darse este caso debería de tardar exactamente lo mismo que nuestro trabajo en el servidor Nvidia utilizado. Además de todo esto, los algoritmos permitidos en este servicio son clasificación, regresión, *clustering,* detección de anomalías, y *ranking.*

En el caso de Amazon, según la página oficial[2], el precio sería un cómputo de 0,37€ por hora, por lo que un mes sería 275,28€, el problema con este método que nos ofrece Amazon es que más adelante, cada vez que queramos realizar predicciones, es decir, utilizar nuestro modelo, es necesario pagar 0,08€ por cada 1000 predicciones, por lo que cada vez que queramos utilizar el modelo, será necesario pagar conforme a lo necesario, además de todo esto, según hemos comprobado en la página de Amazon, los algoritmos permitidos son solo de clasificación, de regresión y de *clustering.*

Como podemos ver, el precio varía bastante, el modelo de Microsoft es bueno debido a que si siempre utilizamos el mismo sin volver a entrenar es un pago único, el de Amazon para utilizarlo debemos pagar por cada 1000 predicciones que necesitemos hacer, con respecto al nuestro podemos decir que el precio pagado por la empresa fue de unos 500€ (sueldo pagado en un mes de prácticas), con la ventaja que nuestro modelo se puede entrenar de nuevo ya que el código está y solo habría que cambiar los datos de entrada, se pueden hacer todas las predicciones que se necesiten y se ha ajustado a las necesidades de la empresa probando diferentes algoritmos, debido a todo eso, al final dará mejores resultados.

Como podemos ver en la tabla 1, existen diversas ventajas a la hora de realizar nuestro proyecto de la manera que la hemos realizado en comparación al uso de las diferentes alternativas que nos ofrece tanto Microsoft como Amazon.

[1] Página oficial Microsoft Azure: https://azure.microsoft.com/es-es/pricing/details/machine-learning-service/
[2] Página oficial de Amazon Machine Learning: https://aws.amazon.com/es/getting-started/projects/build-machine-learning-model/services-costs/

	Trabajo realizado (TFG)	Amazon	Microsoft
Decisión del algoritmo de aprendizaje	Libre	Impuesto (Caja Negra)	Impuesto (Caja Negra)
Precio	500€	Aproximadamente 275€ + 0,08€ por cada 1000 predicciones	Aproximadamente 1700€
Dificultad	Moderada	Fácil	Fácil
Adaptabilidad	Grande	Moderada	Moderada

Tabla 1: Comparativa del trabajo con sus alternativas

Para concluir, existen muchas alternativas a la tomada para poder crear un modelo de clasificación, pero en el momento de la realización de este proyecto o estas alternativas no tienen tanta información y facilidades como las ofrecidas por Python, o son muy caras de usar en un largo periodo de tiempo.

4. Tecnologías utilizadas

En este apartado se presentan todas las herramientas que se han usado para realizar el proyecto, el motivo de porqué las hemos usado y donde podemos descargarlas.

4.1. Ubuntu

Como sistema operativo en este proyecto se ha utilizado Ubuntu, esto se debe a la gran comodidad que ofrece el poder trabajar con una interfaz de líneas de comandos a la hora de lanzar scripts y la facilidad de conectarse a un servidor Nvidia, el cual incorpora 6 CPUs mediante un ssh para así aumentar la velocidad de nuestros entrenamientos de los modelos. El uso de este servidor no es necesario para realizar el proyecto, pero como se ha dicho, ayuda mucho con la velocidad de procesamiento.

En cuanto a la versión que se ha usado es la versión 14.04, ya que es la última versión estable en producción en la fecha de la realización de este proyecto (febrero de 2019). Podemos encontrar este sistema operativo en su página oficial[3].

4.2. Python

Python es el lenguaje de programación que se ha utilizado en nuestro proyecto, esto se debe a, como hemos comentado antes, este lenguaje de programación ofrece muchas librerías que son de ayuda a la hora de crear un modelo, además de que es el lenguaje de programación que más se usa a la hora de la creación de modelos.

En este proyecto hemos usado la versión 3.7.3 de python, la cual se puede encontrar en su página web oficial[4].

[3] Página web de Ubuntu: https://www.ubuntu.com/
[4] Página web Python: https://www.python.org/downloads/

4.3. PyCharm community

En cuanto a entorno de desarrollo, hemos usado *PyCharm community* versión 2019.2 debido a que programar modelos sin uno sería muy tedioso además que nos sirve para tener todo organizado.

Como podemos comprobar en la figura 4, este entorno de programación es muy parecido a otros en su distribución de las herramientas como pueden ser Visual Studio o *IntelliJ IDEA*, donde este último está realizado por la misma compañía que *PyCharm*. Disponemos de un resumen gráfico a mano izquierda donde podemos movernos por el directorio donde tenemos nuestro proyecto, además, como podemos comprobar en la parte inferior, dispone de un apartado el cual nos permite tener diferentes terminales para poder lanzar uno o más scripts a la vez además de una consola Python, todo esto hace que sea muy cómodo trabajar en este entorno de programación.

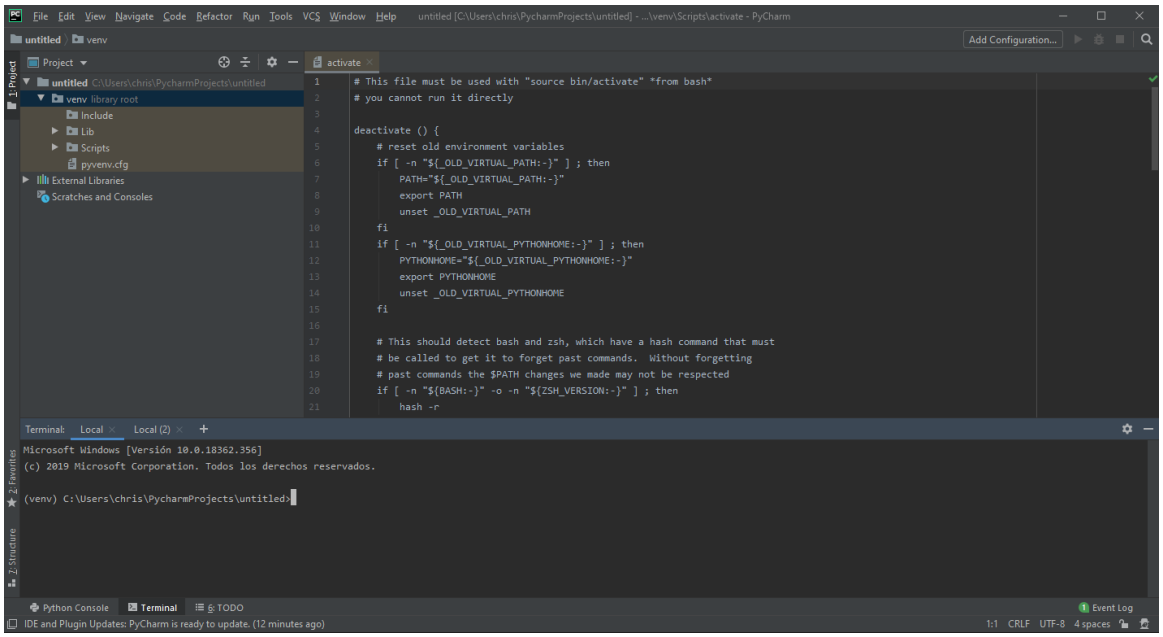

Figura 4: PyCharm Community interfaz

Este entorno de desarrollo en su versión *community* es gratis y se puede encontrar en la aplicación Centro de software de *Ubuntu* que viene de serie en nuestro sistema operativo, o en su página web oficial[5].

[5] Página web PyCharm: https://www.jetbrains.com/pycharm/

4.4. Librerías

Para la realización de este proyecto nos hemos apoyado en diversas librerías que han hecho que muchas de las cosas más complicadas sean accesibles y sea sencillo trabajar con ellas. A continuación, hablaremos de las más importantes y necesarias para realizar los clasificadores.

4.4.1. Numpy 

Numpy se trata de una librería para *Python* la cual se encarga de crear y manejar arrays N-dimensionales de datos de forma eficiente y sencilla, lo que nos permite implementar computaciones numéricas de nuestros datos sin mucha dificultad [22].

Hacemos uso de esta librería en el clasificador de noticias para poder conseguir de la forma más sencilla y eficiente posible la media de cada clase predicha, eso nos permitirá conocer la eficiencia de nuestro clasificador mediante la función `np.mean`, el cual como se muestra en el ejemplo de la figura 5 sacada de la página de numpy[4], podemos ver cómo funciona la función y con qué facilidad nos ayuda a sacar la media de un array.

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([ 2.,  3.])
>>> np.mean(a, axis=1)
array([ 1.5,  3.5])
```

Figura 5: Ejemplo de numpy, media de un array.

Podemos descargar esta librería mediante la línea de comandos con *pip*, o mediante otras distribuciones como pueden ser *Anaconda*, *WinPython* para Widows, etc. Podemos encontrar más información de donde descargarla en su página oficial[6].

[6] Página oficial Numpy: https://www.numpy.org/

4.4.2. Pandas 

Pandas es una librería para *Python* la cual ofrece una forma sencilla de crear estructuras de datos y herramientas para analizar y manipular datos. Esta librería es buena para el manejo de datos perdidos (Los cuales se representan como NaN) Además de que se encarga del alineamiento de datos de forma automática de modo que ayuda a aumentar el rendimiento de nuestro entrenamiento [23].

Hacemos uso de esta librería en el clasificador de dominio para la creación de una estructura de datos que usaremos para entrenar nuestro modelo. Esto lo haremos ya que como hemos dicho, se encarga del alineamiento de datos automáticamente y este clasificador se ha realizado con muchos datos de entrenamiento, lo cual nos ayudará a mejorar un poco el rendimiento de este.

Podemos encontrar un ejemplo de uso en la figura 6, donde leemos un archivo csv (*Comma Separated Values*), el cual contiene 1 millón de frases etiquetadas de la siguiente manera [Texto, Categoría]. En el ejemplo podemos ver cómo asignamos un nombre a cada columna, y se la introducimos a pandas para que realice la asignación, por lo que, desde ahora, tenemos un *dataframe* en nuestra variable llamada *trainDF que* podemos utilizar para entrenar nuestro modelo, ya que ha asignado a cada texto la columna *Text* y a cada categoría correspondiente la columna *Category*.

```
trainDF = pandas.read_csv('new_data_1000000.csv')
# print(df.head())
col = ['Text', 'Category']
trainDF = trainDF[col]
```

Figura 6: Ejemplo de creación de un Dataframe con un archivo csv

Podemos descargar esta librería mediante *pip* como la anterior, o mediante *Conda* como nos indican en su página oficial[7].

[7] Página oficial Pandas: https://pandas.pydata.org/

4.4.3. Pickle 

Pickle es una librería que ofrece una forma de poder guardar y cargar los modelos que entrenemos para poder usarlos después para clasificar otros datos. Gracias a esta librería nada más terminar de entrenar, podemos guardar el modelo en un binario el cual nos dejará cargarlo más adelante en cualquier otro script para clasificar datos.

Podemos observar un ejemplo de cómo usar Pickle en la figura 7 sacada de la página oficial de Sklearn[8], donde se muestra la creación de un pequeño modelo usando datos proporcionados por sklearn (`dataset.load_iris()`), luego del entrenamiento, podemos observar cómo se guarda un modelo mediante la función `pickle.dumps`, y vuelve a cargarlo para poder predecir mediante la función `pickle.loads`.

```
>>> from sklearn import svm
>>> from sklearn import datasets
>>> clf = svm.SVC(gamma='scale')
>>> iris = datasets.load_iris()
>>> X, y = iris.data, iris.target
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

>>> import pickle
>>> s = pickle.dumps(clf)
>>> clf2 = pickle.loads(s)
>>> clf2.predict(X[0:1])
array([0])
>>> y[0]
0
```

Figura 7: Ejemplo de guardado y carga de un modelo mediante la librería Pickle

Podemos descargar esta librería tanto con pip o Conda al igual que las anteriores.

4.4.4. Sklearn

Sklearn es la librería de software libre la cual ofrece varios algoritmos, tanto de clasificación y como de regresión. Usaremos esta librería junto a todas las demás para así entrenar el modelo con los diferentes algoritmos que ofrece (Máquinas de soporte vectorial, bosques aleatorios, Naive Bayes y regresión lógica) para así comprobar qué algoritmo es el mejor para cada clasificador.

Esta librería también ofrece funciones para poder preparar los datos, ya que nos permite crear una matriz para contar cuantas veces aparece cada palabra, nos permite quitar las palabras vacías (*Stopwords* en inglés), es decir, las palabras sin significado como pueden ser los artículos, pronombres, etc. Todo eso nos permite limpiar el corpus de datos de entrenamiento para así mejorar dicho entrenamiento y mejorar el clasificador [15].

Podemos encontrar un ejemplo sacado de la página oficial de sklearn de lo sencillo que es poner a entrenar un modelo mediante esta librería en la figura 8, donde se decide usar la regresión lineal como algoritmo de entrenamiento, se le añaden los datos a la función fit para entrenarlo (*X_train* e *Y_train*) siendo *X_train* los datos e *Y_train* las etiquetas de dichos datos, y por último se usa el modelo para predecir unos cuantos datos que nos hemos guardado, los cuales sabemos su etiqueta de antemano, con esto se consigue un porcentaje de acierto que se usará para saber la eficacia de dicho modelo.

```
# Creat a linear regression object
model = linear_model.LinearRegression()

# Fit the model using the training set
model.fit(X_train, y_train)

# Make prediction using test set
y_test_pred = model.predict(X_test)
```

Figura 8: Ejemplo entrenamiento de un modelo mediante sklearn

Para poder descargar esta librería podemos hacerlo mediante *pip* o *conda* como se indica en la página oficial de la librería[8].

[8] Página oficial Scikit-learn: https://scikit-learn.org/stable/install.html

5. Caso de estudio

En este apartado se presentan los 2 clasificadores que hemos realizado, tanto el clasificador de noticias, el cual nos servirá de introducción a nuestro mejor modelo y más completo, el clasificador de dominio. Hablaremos sobre las diferentes herramientas que hemos usado para la realización de este caso de estudio y cómo hemos configurado y construido nuestros clasificadores, además podemos encontrar un pequeño esquema de los pasos seguidos para la obtención de este proyecto en la figura 9.

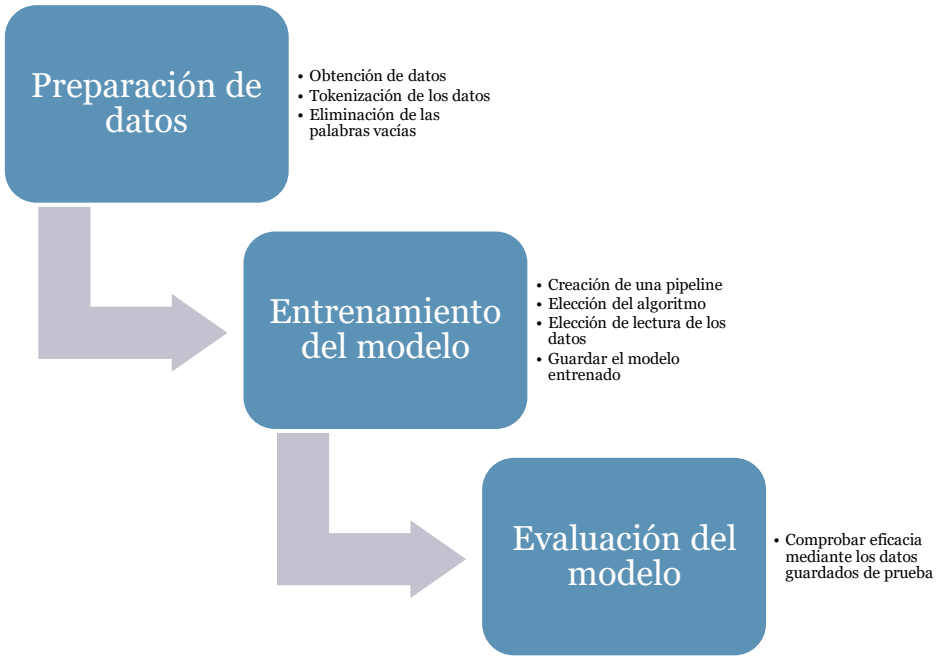

Figura 9: Esquema de pasos para la creación del modelo

5.1. Preparación de datos

La preparación de los datos es la parte más importante a la hora de crear un clasificador, ya que el clasificador deberá de aprender a partir de ellos, y cualquier fallo, desbalance entre el número de ejemplo de cada clase o ambigüedad en las frases (palabras vacías, etc.) puede afectar al aprendizaje de estos y, por ende, haría que nuestro clasificador fuese menos preciso. Por eso, lo primero que haremos es preparar los datos que nos permitirán empezar a clasificar nuestros clasificadores.

5.1.1 Clasificador de noticias

Primero empezaremos explicando la preparación de datos de entrenamiento del clasificador de noticias. Este clasificador fue propuesto por una empresa como trabajo, por lo que es necesario que las clases que dará como salida a la hora de clasificar nuestro clasificador, tiene que coincidir con las clases que la empresa tiene en su base de datos, las cuales son: 3D, biomedicina, *blockchain*, ciberseguridad, *data analytics*, IA (Inteligencia artificial), IOT (*Internet of things*), nanotecnología, robótica y VR (Realidad virtual)

Lo primero que necesitamos para poder empezar a limpiar los datos, es conseguir nuestros datos de entrenamiento etiquetados, los cuales fueron dados en un fichero en formato json por la empresa, la cual nos dio antiguas noticias ya clasificadas que tenían en su base de datos, y finalmente obtuvimos una distribución de datos como se muestra en la tabla 2.

Categoría	N.º De Noticias
3D	178
Biomedicina	955
Blockchain	150
Ciberseguridad	198
Data Analytics	216
IA	181
IOT	176
Nanotecnología	150
Robótica	169
VR	211

Tabla 2: Datos de entrenamiento para cada clase del clasificador de noticias

Ahora que tenemos los datos, los metemos en un diccionario el cual se estructura de la siguiente manera: “Clase/categoría de la noticia”; “Titulo”; “Resumen”; “Noticia escrita”. Podemos encontrar un ejemplo de una noticia utilizada en la figura 10

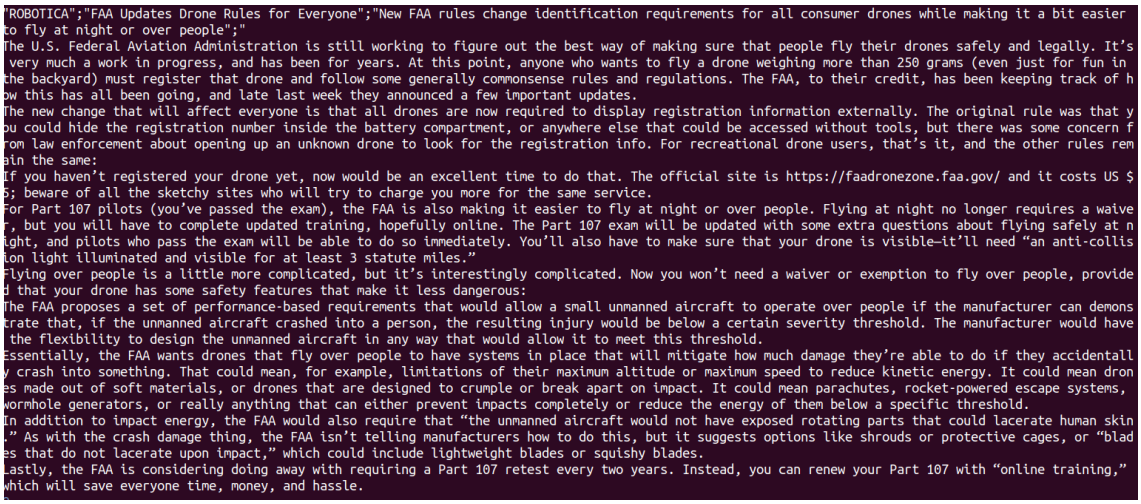

Figura 10: Ejemplo de noticia con categoría robótica

Con dicho diccionario creado, para poder leer los datos, nuestro programa usa la clase `load_files()` de sklearn, la cual funciona de la siguiente manera: Primero, debemos de introducir la dirección de una carpeta donde se encuentran todos los datos para el entrenamiento. Dentro de esa carpeta, existirá una carpeta individual para cada categoría de noticia, que tendrá como nombre la categoría que es. Por último, dentro de esa carpeta existirá un documento de texto por cada noticia de esa categoría, conteniendo solo lo que necesitamos para que aprenda, que es la noticia sin el título, ni su resumen, ni su categoría, ya que eso lo recoge del nombre de la carpeta en la que se encuentra, por lo que solo necesitamos la noticia escrita. Para esto, gracias a que lo tenemos en un diccionario, solo es necesario crear una pequeña función que lo recorra y nos lo organice como se ha explicado.

Una vez terminada la separación y la creación de las carpetas, fue necesario crear una misma estructura de carpetas y archivos con el 10% de las noticias de esa separación anterior, la cual nos servirá como datos de test para comprobar que el clasificador funciona bien, los datos de test (nunca visto en entrenamiento) se clasifican usando el modelo y se comprobará si la predicción es correcta comparando la etiqueta con la que tenía.

Ahora que tenemos la estructura creada y podemos leer los datos mediante la función `load_files()` que nos ofrece la librería *Sklearn.datasets*, podemos empezar a limpiar los datos para que el clasificador se pueda entrenar con los datos más precisos y limpios posibles.

Lo primero que tenemos que hacer con los datos es *tokenizar* los datos, es decir, contar cuantas veces aparece una palabra en cada documento. Esto permitirá al clasificador saber qué palabras suelen ser frecuentes en cada categoría. Esto hemos

realizado mediante la función que nos ofrece Sklearn llamada *`CounterVectorizer()`*. Esto nos devolverá una matriz con cada palabra y las veces que sale en cada documento, pero, existe un problema y es que no queremos tener como hemos dicho anteriormente las palabras vacías, por lo que esta función nos ofrece una forma de quitarlas a la hora de hacer la *tokenización,* por lo que elegimos el idioma en el que está nuestro texto, y le decimos a la función que nos elimine las palabras vacías a la hora de hacer el conteo, quedándonos con la función de la siguiente manera. *`CounterVectorizer(stop_words = "english").`*

Con todo esto ya tendríamos el número de veces que aparece cada palabra en un texto, pero hacerlo de esta manera tiene un problema, y es la descompensación que puede haber en el conteo de una palabra debido a que un texto es mucho más largo que otro, por lo que es necesario sacar la frecuencia en la que aparece la palabra según la longitud del texto, cosa que Sklearn también nos ofrece con la función `TfidfVectorizer()`, que sustituirá a la función anterior, o podemos usar `TfidfTransformer()`, en la salida que nos da la función `CounterVectorizer()`, cualquiera de las 2 son correctas y daría el mismo resultado, pero por limpieza en el código, nosotros hemos utilizado la función `TfidfVetorizer(stop_words = stopwords.words('english')")` y hemos eliminado el conteo normal.

Con el conteo ya hecho y la estructura de los datos realizada de forma correcta, ya podemos pasar a la parte de entrenamiento de este clasificador de noticias.

5.1.2 Clasificador de dominio

Segundo tenemos la preparación de datos de entrenamiento que hemos usado a la hora de crear nuestro clasificador de dominio. Este clasificador fue propuesto por la empresa en el que se ha desarrollado este trabajo. Los datos etiquetados que hemos usado para el entrenamiento de este clasificador fueron dados por la empresa, estos datos venían con las siguientes categorías: Legal, Vernacular, Medicinal, Finanzas y Tecnológico.

Para este clasificador, el número de ejemplos que nos dio la empresa fue mucho mayor que el que teníamos en el clasificador de noticias, como podemos comprobar en la figura 11.

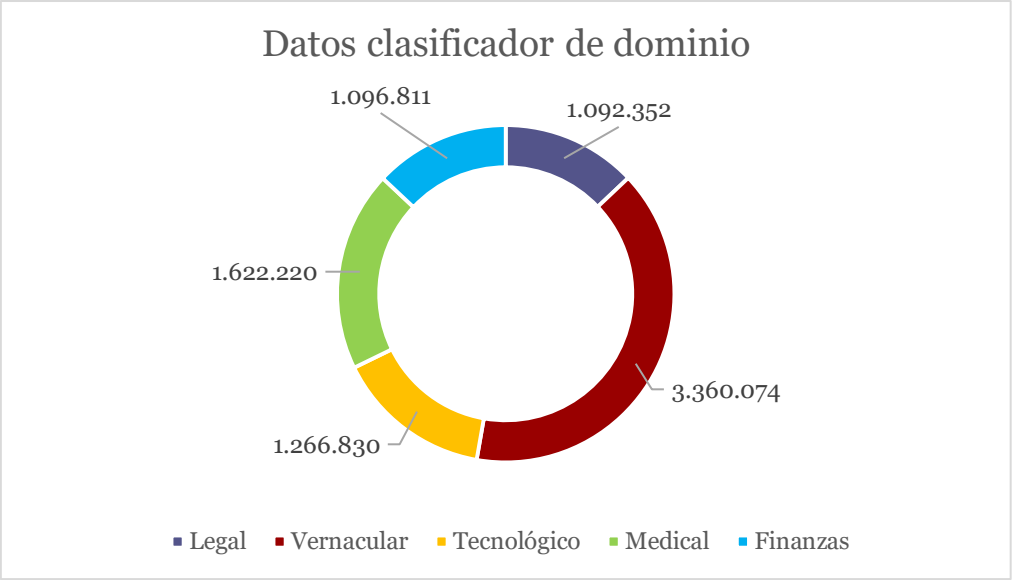

Figura 11: Datos clasificador de dominio

Como podemos observar en la figura 11, tenemos más de 1 millón de datos de cada categoría, esto se debe en parte a que, en este clasificador, en vez de clasificar noticias enteras clasificaremos frases, lo cual hace que buscar datos de entrenamiento sea más sencillo. También, como tenemos tantos datos de entrenamiento, nos ha venido perfecto para comprobar la mejora de un clasificador cuantos más datos de aprendizaje tiene. Para hacer esto, entrenaremos el clasificador con 250.000 ejemplos, con 500.000 datos y con 1.000.000 de ejemplos de cada categoría.

Los datos etiquetados se encuentran en un archivo CSV, en el que la estructura de cada dato es: “Frase”, “Categoría”, como podemos comprobar en las figuras 12 y 13. Con esto conseguimos que, con un pequeño script, leyendo la categoría de cada frase, sacamos el número equivalente de los datos de todas las categorías para evitar desbalance en los datos a la hora de entrenar y lo metemos en otro CSV. En este clasificador hay muchos más datos que en el clasificador de noticias.

```
"the new member states shall be required to accede, under the conditions laid down in this protocol, to the agreements or conventions concluded or provis
ionally applied by the present member states and the union or the european atomic energy community, acting jointly, and to the agreements concluded by th
ose states which are related to those agreements or conventions.  ",Legal
```

Figura 12: Ejemplo frase Legal para clasificador de dominio.

```
"if you need more information about your medical condition or your treatment, read the package leaflet (also part of the epar) or contact your doctor or
pharmacist.",Medical
```

Figura 13: Ejemplo frase Medical para clasificador de dominio.

Una vez sacado los datos en su respectivo CSV, usaremos Pandas para leer los datos y crear una estructura de datos que nos ayude con el rendimiento. Al igual que en clasificador de noticias, empezamos a limpiar los datos. Primero, como hemos dicho, usamos pandas para que nos cree un array bidimensional en que se estructura de la siguiente manera: [Frase][Categoría].

Podemos encontrar un ejemplo de cómo se realizó este array en la figura 14, donde primero se lee el archivo csv, donde como hemos anteriormente, está dividido el texto y la categoría mediante una ",". La función `pandas.read_csv()` nos crea un array con tantas dimensiones como datos separados por una coma existan, por lo que solo nos quedaría asignarle un nombre a cada columna del array, la cual en el ejemplo se le llama "Text" y "Category".

El usar Pandas como se ha explicado anteriormente, ayuda a mejorar el rendimiento gracias a todos los procesos automáticos que realiza con los datos.

```
# create a dataframe using texts and lables
print('Reading CSV...')
trainDF = pandas.read_csv('new_data_1000000.csv')
# print(df.head())
col = ['Text', 'Category']
trainDF = trainDF[col]
```

Figura 14: Código para la creación de array bidimensional de datos

Teniendo la estructura de datos creada, el siguiente paso es separar todos los datos de entrenamiento que tenemos y dejar un porcentaje para test, el cual se eligió que sería de un 10%. Debido a que en este clasificador solo tenemos una estructura de datos con todos estos juntos, la separación se ha realizado mediante la función que nos ofrece la librería *Sklearn*: `model_selection.train_test_split()`. Esta función ofrece la posibilidad de cambiar el porcentaje de prueba que queremos cambiando solo una variable que se le pasa llamada `test_size`, en el que 0.1 es el valor que representa el 10% de los datos.

Podemos observar un ejemplo del funcionamiento de esta función en la figura 15, donde separamos el 10% de los datos para validar posteriormente nuestro modelo. También podemos ver como para que el modelo sea constante en todos los tipos de algoritmos que hemos usado y no dependa el resultado de los datos que ha cogido para entrenar y para validar, le pasamos a la función un número a su valor `random_state` para que siempre sean los mismos datos, donde los textos de entrenamiento se guardarán en la variable *train_x* y sus respectivas categorías en la variable *train_y* y lo mismo con los datos de test con las variables *valid_x* y *valid_y*.

```
train_x, valid_x, train_y, valid_y = model_selection.train_test_split(trainDF['Text'], labels, test_size=0.10,
                                                                      random_state=3)
```

Figura 25: Código para la separación de datos

Con los datos separados, como hemos hecho en el clasificador anterior, realizamos el conteo de las palabras mediante la función `TfidfVectorizer()`, en la que no solo quitamos las palabras vacías, sino que también nos pone todos los verbos en infinitivo para así evitar diferencias entre palabras como corría y correría, ya que no nos interesa el tiempo verbal en el que está escrito el texto.

Debido a que en este clasificador tenemos muchos más datos, podemos comprobar también si es mejor que el clasificador aprenda a nivel de frase, a nivel de varias palabras, a nivel de palabra, o a nivel de letra.

Una vez tengamos la función con los ajustes necesarios, sacamos el conteo con la configuración que hemos decidido mediante la función `fit_transform()` de `TfidfVecrorizer()`. Esto nos da la matriz del conteo y lo ajusta a nuestros datos.

Para poder entender mejor la salida de la matriz que recibimos a la hora de usar la función `TfidfVecrorizer().fit_transform(Datos)` podemos ver cómo nos quedaría la matriz introduciendo las frases: Frase 1: *I'm having so much luck* y Frase 2: *Money, i have so much money*, en la tabla 3.

	HAVE	*LUCK*	*MONEY*
Frase 1	1	1	0
Frase 2	1	0	2

Tabla 3: Ejemplo matriz de conteo

Como podemos comprobar en la tabla 3, las palabras *"I", "am", "so", "much"* son recogidas como palabras vacías que no aportan nada al contexto para la clasificación, y la palabra "*having*" se ha contado en infinitivo y no se ha creado una palabra para ella, ahorrando espacio y mejorando la eficiencia. Con todo esto, hemos conseguido un vector para la frase 1: 1-1-0 y para la frase 2: 1-0-2, gracias a esto, sabremos en todo momento la proporción de aparición de palabras con significado en el texto.

5.2. Entrenamiento del clasificador

Una vez tenemos los datos preparados, toca pasar a la fase de entrenamiento de nuestros clasificadores. En este apartado hablaremos sobre los diferentes algoritmos con los que hemos entrenado nuestros clasificadores y los resultados que nos han dado cada uno de estos.

5.2.1 Clasificador de noticias

El clasificador de noticias fue el primero que se creó, y como hemos visto anteriormente en la preparación de datos, existen muy pocos datos de entrenamiento, por lo que no es un clasificador que vaya a dar muchas oportunidades de experimentar con él, aun así, son suficientes datos para poder crear una primera versión del clasificador.

Como ya tenemos los datos preparados, es hora de clasificar, y para ello, es necesario elegir primero con qué clasificadores queremos entrenar nuestro clasificador. Debido a que es un clasificador con pocos datos, hemos elegido 2 clasificadores sencillos para el entrenamiento: Naive Bayes y máquinas de soporte vectorial lineales.

Para poder entrenar el modelo lo único que es necesario hacer es llamar a nuestra función para realizar el conteo y transformar la matriz del conteo a nuestros datos, y a continuación, pasarle uno de los algoritmos que hemos elegido para realizar el clasificador. Para realizar fácilmente los pasos de Vectorizar → Transformar → Clasificar más fácil de usar, la librería Sklearn nos ofrece una clase *pipeline*, en la que podemos introducir nuestras funciones de preparación de datos y el algoritmo para realizar todo esto de una sola llamada (puedes encontrar más información de cómo usarla en la página oficial[9]).

Con la pipeline construida, lo único que hay que hacer es introducir la función del algoritmo con el que deseamos enseñar a nuestro clasificador y llamar a la función `fit(datos con los que entrenamos)` a nuestra pipeline y con esto, nuestro clasificador aprenderá mediante nuestros datos, las diferentes categorías y cómo diferenciarlas, podemos ver el código utilizado para crear la *pipeline* y el entrenamiento del modelo con máquinas de soporte vectorial en la figura 16. Lo único que nos queda es evaluar la precisión que tiene a la hora de clasificar las noticias y guardar nuestro modelo.

[9] Sklearn.pipeline: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

```
text_clf = Pipeline([('vect', CountVectorizer(stop_words=noise_list, ngram_range=(1, 1), strip_accents='ascii')),
                     ('tdifd', TfidfTransformer(use_idf=True)),
                     ('dlc', SGDClassifier(loss='hinge', penalty='l2',
                                           alpha=0.001, random_state=42,
                                           max_iter=200, tol=None)),
                     ])

text_clf = text_clf.fit(file_train.data, file_train.target)
```

Figura 16: Código de pipeline y entrenamiento con máquinas de soporte vectorial.

Primero, para guardar nuestro clasificador lo que hicimos fue usar la librería *pickle*, la cual nos ofrece la función `pickle.dump(Modelo a guardar, sitio donde guardarlo)`, con esto la librería nos crea un binario en el lugar indicado que podremos cargar en cualquier momento en cualquier programa mediante la función `pickle.load(archivo a cargar)`.

Una vez tenemos nuestro clasificador guardado en un archivo binario, para poder determinar la precisión de nuestro esté a la hora de clasificar las noticias en las diferentes categorías, haremos uso de los datos de test que hemos reservado. Lo que hacemos con estos datos, es pasarlos todos por nuestro clasificador para que este nos indique la categoría a las que él cree que pertenece mediante la función `predict()`.

Debido que estamos realizando un aprendizaje supervisado, sabemos la categoría de esos datos reservados a priori gracias a que todos los datos están etiquetados con ella. Una vez tenemos todos los datos clasificados, los comparamos y comprobamos cuales han acertado y cuáles no y sacamos una puntuación que nos indica la precisión a la hora de clasificar el cual varía entre 0 y 1, 0 siendo que falla en todas y 1 que acierta todas. Podemos ver una representación de la puntuación general de cada categoría en las figuras 17 y 18 donde podemos comprobar las etiquetas predichas en nuestros ejemplos y las etiquetas que realmente tienen.

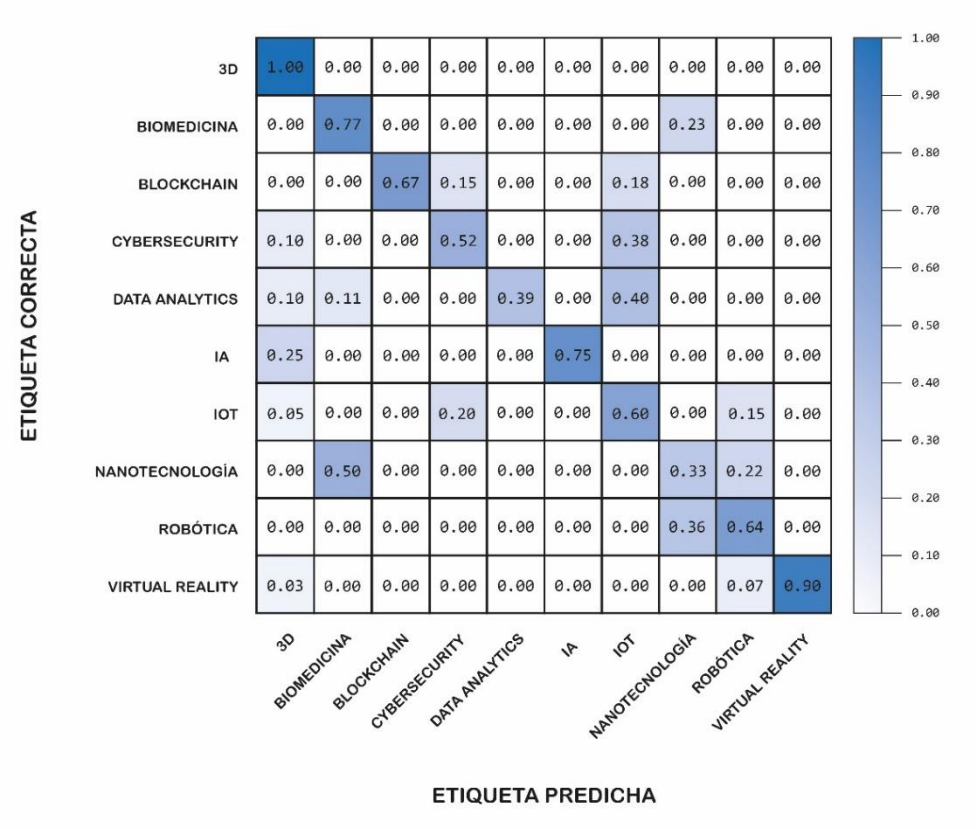

Figura 17: Matriz de confusión de nuestro modelo con Máquinas de soporte vectorial

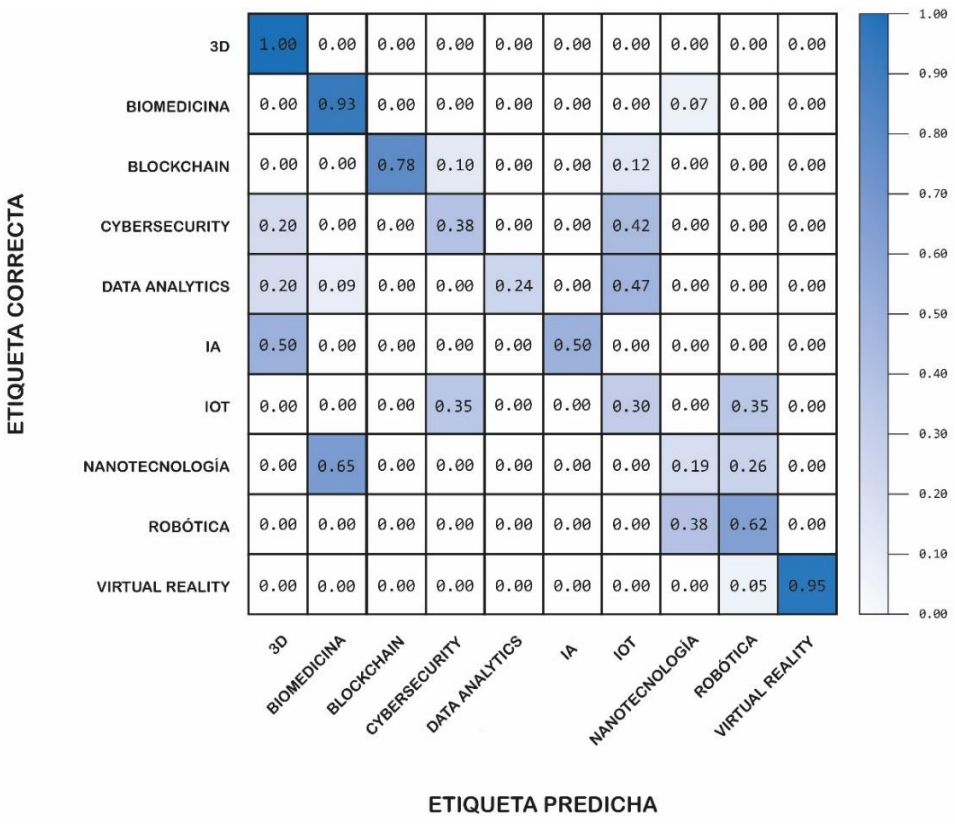

Figura 18: Matriz de confusión de nuestro modelo con Naive Bayes

Como hemos comentado antes, hemos realizado el entrenamiento mediante 2 algoritmos, Naive Bayes y máquinas de vectores lineales, y los resultados obtenidos por ambos como podemos comprobar haciendo la media de aciertos en las figuras 17 y 18 han sido: 58,9% de acierto para el clasificador entrenado mediante Naive Bayes y 65,7% en el clasificador entrenado con máquinas de soporte vectorial. Podemos sacar varias conclusiones de estas figuras como, por ejemplo, la clase “nanotecnología” no la clasifica bien debido a que es una clase que suele contener contenido ambiguo, además comprobamos que SVM funciona mejor que Naive Bayes debido a que es más compleja.

5.2.2 Clasificador de dominio

En la fase de entrenamiento del clasificador de dominio hemos realizado diversos experimentos sobre la diferencia que hay al entrenar un clasificador con diferentes tamaños de datos, con diferentes algoritmos y con diferentes formas de leer los datos.

Debido a que la base del entrenamiento del clasificador es la misma que el clasificador de noticias, no se explicará de nuevo, en cambio nos centraremos en las diferencias que existen con respecto al entrenamiento explicado anteriormente y los resultados obtenidos.

En cuanto a los algoritmos utilizados en nuestro clasificador, hemos optado por 5 algoritmos para entrenarlo. Al igual que el clasificador de noticias, hemos usado Naive Bayes y máquinas de soporte vectorial, en este caso con *kernel* lineal y con *kernel rbf (Radial basis function),* además de regresión logística y bosques aleatorios. Todos estos algoritmos se pueden encontrar en la librería Sklearn, y para poder implementarlos, como hemos dicho anteriormente, solo hace falta meter la función en nuestro pipeline.

Otro de los cambios que hemos realizado en este clasificador es entrenarlos todos con diferentes formas de estructurar los datos en la matriz creada mediante `Tfidf`. Como hemos dicho anteriormente, las formas de estructurar los datos con las que hemos entrenado nuestro clasificador son: Leyendo letra a letra (*character level)*, en la que el clasificador tiene en cuenta cada letra, la proximidad entre ellas, etc. Leyendo palabra a palabra *(word level)*, en la que en vez de tener en cuenta cada letra se mira por cada palabra. Leyendo varias palabras a la vez (*Ngram level*), en el que se va estructurando en intervalos de palabras que nosotros le especificamos, es decir, si tenemos por ejemplo la frase: “El código no funcionaba en la puerta”, si elegimos un intervalo de 2 palabras, el clasificador leerá primero “El código”, después repetirá la última y cogerá la siguiente, quedando “código no”, y así sucesivamente hasta tener toda la frase.

Por último, hemos entrenado este clasificador con diferentes tamaños de datos, para así poder comprobar la diferencia en eficiencia que hay a la hora de entrenar un modelo con diferentes tamaños de datos para entrenar. Los tamaños han sido para cada categoría: 250 mil ejemplos, 500 mil ejemplos y 1 millón de ejemplos.

5.3. Evaluación de resultados

Con todos los clasificadores entrenados y teniendo todos los resultados como podemos comprobar en las tablas 4, 5 y 6, por fin podemos comparar las diferencias según las características con las que hemos entrenado nuestro clasificador.

	Naive Bayes	Regresión logística	Máquina de soporte vectorial con kernel rbf	Máquina de soporte vectorial con kernel lineal	Árboles aleatorios
Character Level	43,18%	77,65%	35.60%	**87,50%**	79.16%
Word Level	67,8%	84,1%	35.60%	**87,50%**	81.44%
Ngram Level	70,8%	75,76%	35,60%	82,57%	76.51%

Tabla 4: Porcentaje de acierto con 250.000 ejemplos de cada categoría

	Naive Bayes	Regresión logística	Máquinas de soporte vectorial con kernel rbf	Máquinas de soporte vectorial con kernel lineal	Árboles aleatorios
Character Level	72.71%	85,64%	29.85%	90,08%	91,33%
Word Level	39.01%	49,99%	49,75%	49,82%	85,36%
Ngram Level	80.40%	88,68%	27.04%	90,85%	**92,06**%

Tabla 5: Porcentaje de acierto con 500.000 de ejemplos de cada categoría

	Naive Bayes	Regresión logística	Máquina de soporte vectorial con kernel rbf	Máquina de soporte vectorial con kernel lineal	Árboles aleatorios
Character Level	71.2%	86,34%	44.15%	89.81%	91.71%
Word Level	38.9%	50,41%	50,41%	49.92%	85.62%
Ngram Level	79.8%	88,70%	42.66%	90.42%	**92.22%**

Tabla 6: Porcentaje de acierto con 1.000.000 ejemplos de cada categoría

En cuanto al clasificador de dominio, que es el clasificador en el que hemos realizado los experimentos para la comprobación de estas diferencias, podemos comprobar la gran importancia que tiene el entrenar mediante una gran cantidad de datos. Como podemos comprobar en la tabla 4, a la hora de entrenar con 250 mil datos de cada categoría, ya conseguimos un muy buen resultado, siendo 87.5% la tasa de acierto usando el algoritmo de máquinas de soporte vectorial con el *kernel* lineal, sin embargo, en el momento que doblamos el número de datos de entrenamiento a 500k, como podemos comprobar en la tabla 5, prácticamente todos los clasificadores mejoran una cantidad considerable con respecto a entrenarlos con solo la mitad, llegando a ser el mejor resultado mediante árboles aleatorios con una tasa de acierto de 92.06%. También podemos comprobar cómo cuanto más complejo se hace, y teniendo en cuenta que estamos clasificando frases, al aumentar los datos de entrenamiento, el clasificar estructurando palabra a palabra empeora, llegando a ser peor en un 30% en los algoritmos más sencillos como Naive Bayes y regresión logística con respecto al anterior. Por último, en la tabla 6, volvemos a sacar los resultados con 1 millón de datos en cada categoría, pero esta vez, comprobamos como la mejora cada vez es menor, siendo los resultados prácticamente iguales a los conseguidos con el entrenamiento con 500 mil, haciendo que el mejor, siga siendo mediante árboles aleatorios con una mejora solo de 0.16 en la tasa de acierto, dándonos una tasa de acierto del 90.22% estructurando los datos con un intervalo de palabras.

Comprobando las tablas 4, 5 y 6, también podemos observar que nuestro clasificador funciona muy bien mediante algoritmos que separan los datos linealmente. Esto se puede ver claramente en la diferencia que hemos obtenido a la hora de usar el kernel lineal con las máquinas de vectores de soporte y el obtenido usando el kernel rbf, el cual podemos ver cómo se comporta en la figura 2, sección 2.2.1.

Por último, hemos observado que en nuestro clasificador los mejores resultados los hemos obtenido mediante la estructuración en la matriz de varias palabras (2 en nuestro caso). El estructurarlo letra a letra también ha dado muy buenos resultados, acercándose a la tasa de acierto que hemos conseguido con varias palabras a la vez, pero, por el contrario, cuando lo hemos realizado palabra por palabra ha dado peores resultados, esto puede deberse a que nuestros datos son frases no muy largas, y por ello el ir carácter a carácter o varias palabras a la vez sea mucho más efectivo.

Como conclusión, el mejor resultado lo hemos obtenido mediante el algoritmo de bosques aleatorios, y como cabía esperar, ha sido con el mayor número de datos que se ha podido realizar 1 millón en cada categoría, concluyendo que cuantos más datos para aprender siempre será mejor, pero que el aprendizaje llega un momento de estabilidad en el que, aunque introduzcamos más datos, la mejora es mínima.

6. Cronología del proyecto

Como podemos observar en la figura 19, a la hora de realizar este proyecto se ha pasado por diferentes fases que hemos representado en la línea temporal.

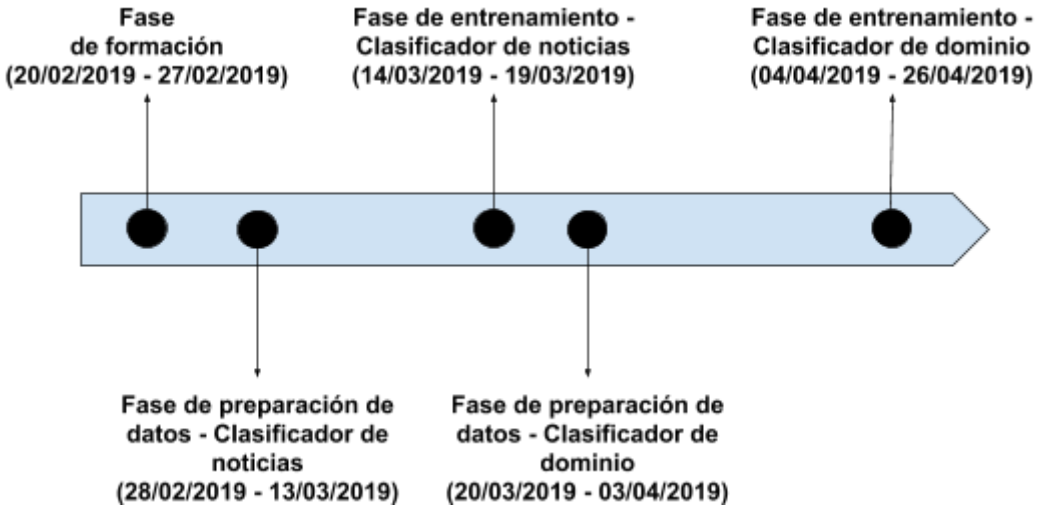

Figura 19: Línea temporal del proyecto

La primera fase fue la fase de formación, en la que debido a que nunca había visto ni python ni nada relacionado con los algoritmos de aprendizaje, fue necesario una semana de investigación y búsqueda en internet para conocer las herramientas que podía utilizar a la hora de crear el clasificador y conocer cómo funcionan algunos de los algoritmos de entrenamiento.

La segunda fase fue la fase de preparación de datos del clasificador de noticias, aquí fue donde más problemas se encontró debido a que fue la primera vez que utilizaba estos conceptos como la vectorización de texto. Por suerte, no fue extremadamente difícil debido a que *sklearn* tiene una muy buena documentación y se puede encontrar todo muy fácilmente.

La tercera fase de este proyecto fue la fase de entrenamiento del clasificador de noticias, debido a que este clasificador tiene pocos datos, duró poco en entrenarse y fue muy sencillo gracias a las librerías utilizadas, por lo que esta fase fue muy corta.

La cuarta y penúltima fase fue la fase de preparación de datos del clasificador de dominio, en el que se repitieron los pasos de la preparación de datos del clasificador de noticias, la única diferencia fue el cómo leímos los datos y la preparación de los datos previa para meterlos en el *script* python. Debido a esto, fue una fase rápida y sencilla.

Como última fase, tuvimos la fase de entrenamiento del clasificador de dominio, donde se tardó un tiempo considerable debido a que se entrenaron diferentes modelos con muchos datos, lo que hizo que algunos entrenamientos con un millón de datos tardasen de 3 a 4 días, dando un total de casi un mes para entrenar todas las categorías con los diferentes tamaños de datos.

7. Conclusiones y trabajo futuro

Como conclusión, en este proyecto hemos conseguido desarrollar dos clasificadores de texto, uno de noticias y uno de dominio. El clasificador de noticias nos ha servido para aprender las cosas básicas sobre la creación de clasificadores, las cuales hemos aplicado a la hora de crear el clasificador de dominio, en el que, debido a que en este clasificador hemos tenido mucho más texto para trabajar, es el que hemos utilizado para experimentar tanto con los algoritmos como con la forma de estructurar el texto.

Hemos cumplido los objetivos de este TFG ya que hemos conseguido crear un clasificador de texto lo suficientemente bueno como para ser usado en la empresa como clasificador de dominio. Además, se ha cumplido también el segundo objetivo, el cual era conocer las diferencias que hay a la hora de entrenar los mismos datos con diferentes algoritmos, cantidad de datos y manera de leer datos a la hora de entrenar un modelo.

Con los datos obtenidos, podemos sacar como conclusión que la cantidad de datos es uno de los factores más importantes a la hora de crear un modelo de clasificación. El clasificador aumenta en precisión cuantos más datos tiene y suele ser un aumento logarítmico donde llega un momento que la mejora es mínima por más datos extra que introducimos.

Además, este trabajo se ha usado en la empresa para poder clasificar cada uno de los datos que tenían a una de las categorías del modelo de dominio, con los que posteriormente, se ha utilizado para reentrenar y mejorar algunas de las máquinas que ya tenían en funcionamiento.

También hemos comprobado las diferencias en precisión entre los algoritmos que hemos utilizado para entrenar el clasificador, en el que, en nuestro caso, los algoritmos lineales han funcionado mejor, seguramente debido a que el clasificador de dominio tenía como datos de entrenamiento frases cortas, lo que hace que sea más complicada la ambigüedad entre 2 clases que en datos de entrenamiento con frases más largas.

Además de la conclusión del trabajo, en lo personal puedo destacar lo difícil que ha sido para mí el aprendizaje del funcionamiento de los diferentes algoritmos, además de la forma que tiene las librerías usadas a la hora de modificar los datos, ya que este proyecto es sobre inteligencia artificial, asignatura en la carrera que se dio en tercero de carrera de una forma muy breve e introductoria, y que se expande en la rama de computación, la cual yo no elegí ya que cursé la rama de ingeniería del software. Aun así, ha sido una forma de extender ese conocimiento de tercero por mi cuenta y de poder aprender sobre este ámbito de la informática que es tan interesante e inmenso.

Como trabajo futuro podría ser en lo que el clasificador de noticias se refiere, conseguir muchos más datos de entrenamiento para mejorar su precisión y comprobar muchos más algoritmos para ver cuál se adapta mejor a él. En cuanto al clasificador de dominio, se podría intentar usar algoritmos de redes neuronales, ya que la librería usada en el proyecto (*sklearn)* nos ofrece la posibilidad de usarlas siempre y cuando se utilicen los datos necesarios.

8. Referencias

[1] Unclassified Data is a disaster waiting to happen (2019) http://www.cio-connect.co.uk/data-classification/

[2] Data labeling for AI is set to become a billion-dollar market by 2023 (2019) https://www.axios.com/ai-data-labeling-billion-dollar-market-409704bc-e63c-4af0-b0d0-44424abcd561.html

[3] Jason Andress CISSP, ISSAP, CISM, GPEN, Mark Leary CISSP, CISM, CGIET, PMP, in Building a Practical Information Security Program (2017) https://doi.org/10.1016/B978-0-12-802042-5.00007-X

[4] Técnica de Machine Learning para crear modelos predictivos a partir de datos de entrada y respuesta conocidos (Marzo 2019) https://es.mathworks.com/discovery/supervised-learning.html

[5] Garcia Martinez, Mercedes. (2012). *Selecting translations to be post-edited by Sentence-Level Automatic Quality Evaluation*. Tesis de master en Universidad Politecnica de Valencia. http://hdl.handle.net/10251/27256.

[6] Tipos de Machine Learning. Clasificación vs Regresión (Marzo 2019) http://heroesdeldato.com/tipos-de-machine-learning-clasificacion-vs-regresion

[7] Extensiones y aplicaciones (Máquinas de vectores soporte, SVM) http://verso.mat.uam.es/~joser.berrendero/cursos/Matematicas-IO/io-tema6-svm-16.pdf

[8] Enrique J. Carmona Suárez. Tutorial sobre Máquinas de Vectores Soporte. (2014) http://www.ia.uned.es/~ejcarmona/publicaciones/%5B2013-Carmona%5D%20SVM.pdf

[9] Libería sklearn SVC (Mayo 2019) https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[10] Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. The annals of statistics, 1171-1220. https://projecteuclid.org/download/pdfview_1/euclid.aos/1211819561

[11] H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf

[12] Juan Zamorano Ruiz. (2018). COMPARATIVA Y ANÁLISIS DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO PARA LA PREDICCIÓN DEL TIPO PREDOMINANTE DE CUBIERTA ARBÓREA https://eprints.ucm.es/48800/1/Memoria%20TFM%20Machine%20Learning_Juan_Zamorano_para_difundir%20%282%29.pdf

[13] José R. Berrendero. Clasificación y regresión logística (2019) http://verso.mat.uam.es/~joser.berrendero/cursos/Matematicas-e2/e2-tema4-16.pdf

[14] Sklearn Logistic Regression (Abril 2019) https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[15] D. Daniel Jiménez González. ALGORITMOS DE CLUSTERING PARALELOSEN SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN DISTRIBUIDOS (Mayo 2011) http://www.dsic.upv.es/docs/bib-dig/tesis/etd-11182010-091738/Tesis.pdf

[16]Tsz Kin Lam, Julia Kreutzer, Stefan Riezler. (2018). A reinforcement Learning Approach to Interactive-Predictive Neural Machine Learning http://arxiv.org/abs/1805.01553

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://arxiv.org/abs/1810.04805

[18] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. (2012). Scikit-learn: Machine Learning in Python. https://arxiv.org/abs/1201.0490

[19] Alberto, Óscar & Pérez, García & Martinez Fernandez, Ignacio. (2019). CLASIFICADOR LINGÜÍSTICO DE TEXTOS EN JAVA. https://www.researchgate.net/publication/267789873_CLASIFICADOR_LINGUISTICO_DE_TEXTOS_EN_JAVA

[20] Massung, Sean and Geigle, Chase and Zhai, Cheng Xiang. (2016) MeTA: A Unified Toolkit for Text Retrieval and Analysis http://anthology.aclweb.org/P16-4016

[21] Text Analytics Toolbox. (marzo 2019). https://es.mathworks.com/products/text-analytics.html

[22] Stéfan van der Walt, S. Chris Colbert, Gaël Varoquaux. (2011). The NumPy array: a structure for efficient numerical computation https://arxiv.org/abs/1102.1523

[23] Página Github de Pandas (mayo 2019). https://github.com/pandas-dev/pandas