UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

GRADO DE ESTADÍSTICA EMPRESARIAL

Clasificación de textos mediante algoritmos de Machine Learning

Trabajo Fin de Grado

Junio 2021

Autor: Sukhwinder Singh Kaur

Tutor: Jose L. Saniz-Pardo Auñón

Índice

1.- RESUMEN

El presente trabajo se centra en el estudio de algunos algoritmos que se engloban bajo el concepto de Machine Learning aplicados a la clasificación de textos, y hacer una comparativa entre ellos en cuanto a la eficacia y precisión a la hora de realizar las agrupaciones en cada uno de los casos.

El primero de ellos consiste en la clasificación de los emails, si son spam o no, y el segundo se centra en la clasificación de las noticias en sus correspondientes categorías. Los algoritmos que se han elegido son el de Naive Bayes y el de Support Vector Machine.

Tras el correspondiente análisis de los resultados se llegaron a varias conclusiones. Primero de todo, hay que destacar la necesidad e importancia de un correcto preprocesamiento de los datos. Si se lleva a cabo de una forma correcta y recomendable, las diferencias en los resultados finales son notables.

Segundo, según el tipo de datos y de acuerdo con la forma de prepararlos, existen funciones que se pueden adaptar mejor al modelo que otras, y de esta forma influir de una manera directa sobre los resultados deseados.

Por último, indicar que de entre los modelos que se emplearon para la resolución de los casos, aquel que mejor se adaptó y que realizó las clasificaciones con una mayor precisión fue el de Support Vector Machine.

2.- INTRODUCCIÓN

El siguiente trabajo de fin de grado se va a centrar en el estudio de distintos algoritmos para la clasificación de textos, valorando las ventajas y desventajas de los mismos. Además, se detallarán la aplicación de los mediante código en Phyton, comparando los resultados de los distintos métodos y haciendo una valoración final.

Se va a comenzar con una pequeña introducción de algunas ideas básicas de conceptos claves, que van a ser de ayuda para la mejor comprensión del trabajo desarrollado.

Primero comenzaremos definiendo qué es el Maching Learning (ML), en qué consiste este concepto que está tan de moda en la actualidad. Explicaremos cuáles son sus diferencias y relaciones con otros conceptos con los que se relacionan.

Seguidamente, nos dispondremos a explicar qué es el procesamiento del lenguaje de texto (NLP), y qué utilidades y aplicaciones puede tener el mismo. Dentro de este apartado veremos una serie de métodos, de entre los cuales vamos a desarrollar dos de ellos. El método de clasificación Naïve Bayes y el Support Vector Machine.

Antes de meternos de lleno en el primero de los modelos, Naïve Bayes, comentaremos las principales diferencias que existen entre la estadística clásica y la bayesiana., ya que el método Naïve se basa en la teoría de la probabilidad bayesiana.

Una vez tenemos todos los principales conceptos y detalles comentados, pasaremos a profundizar en los dos métodos seleccionados. Primeramente, se hará una introducción teórica de en qué consiste el método y su funcionamiento, tanto en un modelo como en el otro.

Seguidamente, los pondremos en práctica mediante una serie de aplicaciones prácticas para valorar su efectividad. El software que se va a utilizar para la programación de dichos métodos es Phyton.

Por último, haremos una comparativa de los resultados que se han obtenido en cada uno de los supuestos prácticos con la aplicación de cada uno de los métodos, haciendo una valoración final de cuál de ellos ha sido el mejor.

3.- ESTADO DE LA CUESTIÓN Y MARCO TEÓRICO

3.1- ¿Qué se entiende por machine learning?

Machine Learning es aquel campo de estudio que otorga la habilidad de aprender al ordenador sin tener la necesidad de que todo sea programado explícitamente, lo que se denomina self-learning. A partir de unos datos, busca relaciones y gracias a estas, se crea un modelo para poder tomar las decisiones finales.

Comparativa con otros campos con los que se relaciona:

Otros campos que se relacionan y están unidos con el ML son la Inteligencia Artificial y el Data Mining. Todos ellos están englobados dentro de lo que se denomina la Ciencia de los Datos (Figura 1).

Por una parte, tenemos la Inteligencia Artificial, que busca el desarrollo de las capacidades de las máquinas para tomar decisiones cognitivas, como el ser humano. Esta rama está formada por subramas, como pueden ser el propio Machine Learning o Natural Language Processing (NLP).

Por otra parte, tenemos el Machine Learning y el Data Mining. Ambos campos buscan patrones y relaciones en grandes cantidades de datos. Además, las técnicas y algoritmos que utilizan son similares, como pueden ser el análisis de componentes principales, árboles de decisión y distintas técnicas de clasificación. Una de las diferencias es que, mientras que el Data Mining se centra en el estudio únicamente de los inputs (no se conoce el resultado final, no se conocen los datos), para poder dar una solución final, ML estudia ambos casos, input y output para hacer el aprendizaje.

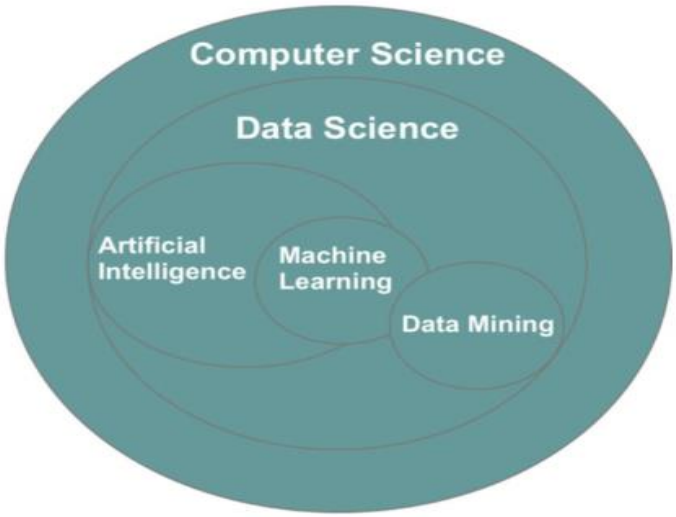

Figura 1: Campos del Data Science

Fuente: Oliver Theobald- Machine Learning For Absolute Beginners [2]

Profundizando en el Machine Learning:

Sabemos que gracias al Machine Learning, la máquina aprende por sí sola. Pero,¿Cómo lo consigue?

Esto es posible gracias a la experiencia que va consiguiendo la máquina. A partir de una serie de datos iniciales, se encarga de buscar los patrones y relaciones existentes entre las variables, con los que se creará un modelo para tomar de manera posterior una decisión final.

Los datos que le son proporcionados son divididos en dos partes, Train y Test Data (Figura 2). Es decir, del conjunto de datos que se le proporciona, se guarda una parte (Train), para realizar el aprendizaje, buscar patrones y relaciones entre los datos y variables, y la otra parte (Test), para poner en práctica el modelo. De esta forma, de una manera previa a probar el modelo con nuevos datos, se comprueba la fiabilidad con los datos ya conocidos. En el caso de que el modelo no tenga suficiente eficacia o no cumpla con las condiciones establecidas, se vuelve a entrenar buscando otro modelo.

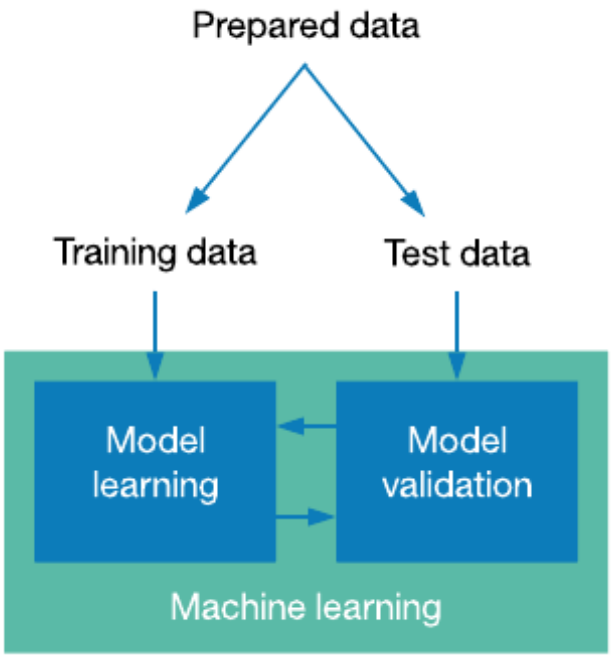

Figura 2 Train y Test Set

Fuente: TowardsDatascience.com [3]

Normalmente el porcentaje para dividir dichos datos está en un 60-70% para el Train, y un 30-40% para Test, dependiendo de la cantidad de datos que se tengan.

Una vez se tienen construidos los modelos, es necesario tener una función loss or cost (loss / cost function) que nos permita evaluar nuestro modelo.

Ventajas y Desventajas:

Se ha llegado a pensar que no es de mucha utilidad y que estas normas o patrones se le pueden introducir manualmente, pero eso lleva aparejado el problema de la actualización de dichas normas cada vez que se produzca un cambio en alguna de las características en algún momento en el tiempo. Por tanto, aumenta el retrabajo para la remodelación y el reajuste a los nuevos patrones.

Es por ello que, una de las ventajas de ML es su facilidad para adaptarse a los entornos cambiantes en el tiempo. Además, otro factor importante que juega a su favor es la gran cantidad de datos que se manejan hoy en día, además de la variedad de los mismos, no solo son del tipo de numéricos, también los textos, sonidos e imágenes.

Por otra parte, existen una serie de problemas que pueden surgir cuando se emplea algún algoritmo de machine learning. Uno de ellos es que se cuente con bad data.

Esto se refiere a problemas como la insuficiente cantidad de datos que hacen que la máquina no tenga suficiente cantidad de información sobre la que aprender, la existencia de valores que no son del todo representativos debido a la existencia de valores atípicos o de valores ausentes.

Categorías del Machine Learning:

Una vez que contamos con una visión global del funcionamiento del Machine Learning, vamos a adentrarnos y ver las distintas categorías en las que se pueden clasificar todos los algoritmos de este campo.

En función de la naturaleza de los datos (inputs), los algoritmos se clasifican en:

- Aprendizaje Supervisado (Supervised Learning)
- Aprendizaje No Supervisado (Unsupervised Learning)
- Aprendizaje Reforzado (Reinforcement Learning)

Supervised Learning:

En este tipo de aprendizaje se caracteriza por que tanto el input, datos iniciales que son proporcionados, como el output, la solución que se quiere obtener, son conocidos.

Por tanto, la forma de trabajar de estos modelos es que buscan relaciones que existen entre una serie características y la decisión final.

Una familia de algoritmos que se incluyen en este grupo son los métodos de regresión. En ellos se busca el posible efecto que tenga las características, denotadas como X's, sobre la decisión final, denotada como Y.

Un ejemplo muy común que se utiliza para la compresión de este tipo de métodos es el de calcular la valoración de un coche. Se trata de, a partir de una serie de características como son su modelo, potencia, accesorios o el consumo (X), intentar predecir su precio final (Y).

Algunas técnicas que se engloban en este ámbito son:

- Regresión Lineal / Regresión Logística
- Árboles de decisión y Random forests
- Support Vector Machine (SVM)
- Redes Neuronales
- K-Nearest Neighbours

Unsupervised Learning:

En este caso, los datos finales son desconocidos, por lo que no se pueden buscar relaciones entre input y output. Se centra por tanto en la búsqueda de relaciones y patrones escondidos entre los inputs.

Un ejemplo que define claramente este grupo es la clasificación. Pongámonos en el caso de un blog, que contiene información sobre distintos ámbitos y temas, y va recogiendo información sobre qué es lo que leen las personas, frecuencia de conexión y otras más. En este caso no se conoce la clasificación final, por tanto, tiene que buscar relaciones entre estas variables y poder hacer así una buena agrupación.

Otra aplicación muy frecuente de estos métodos o técnicas no supervisadas es la detección de fraudes. De igual forma que se agrupaba a las personas del blog con características similares, cuando alguien tiene un comportamiento en internet sospechoso, ya sea la hora de conexión, la información que consulta, entre otras cosas, se puede detectar que no se adapta a las características del resto de personas y es un comportamiento anómalo.

Algunas técnicas que se engloban en este ámbito son:

- K-medias (Clasificación)
- Análisis de Componentes Principales (Reducción Dimensión y Visualización)

Reinforcement Learning:

La tercera categoría de clasificación, a diferencia de las otras dos, construye modelos basados en la experiencia que va ganando por las decisiones aleatorias que toma.

Por tanto, se basa en aprender basándose en una gran cantidad de combinaciones posibles entre los datos que se le proporcionan.

Este tipo de ML está en auge, y un ejemplo claro es el de los self-driving cars. Otro de los ejemplos que se conocen es el juego de ajedrez AlphaGo. En 2017, una máquina, que se había entrenado utilizando este método con una gran cantidad de combinaciones y movimientos posibles, se enfrentó al campeón del mundo Ke Jie.

3.2-Text Analysis: Clasificación de textos y NLP

Text analysis es una de las categorías de entre las muchas que se engloban bajo el término de la Inteligencia Artificial. Se trata de procesos para la extracción de información relevante a partir de unos textos, pudiéndose estos encontrarse en muchas formas de la vida cotidiana, como pueden ser libros, páginas web, revistas y demás.

El objetivo principal es el de transformar el texto original en datos para poder realizar análisis con los mismos, y para ello se usan técnicas como el Natural Language Processing, técnicas del Data Mining y algoritmos propios del machine Learning.

El Text Analysis engloba acciones como el análisis léxico, la frecuencia o conteo de las palabras, reconocimiento de patrones, etc. En cuanto a sus las

aplicaciones, son variadas y entre ellas destacan el análisis de sentimientos, agrupación de textos del mismo tipo o la clasificación en categorías. Tal y como se puede apreciar en la Figura 3.

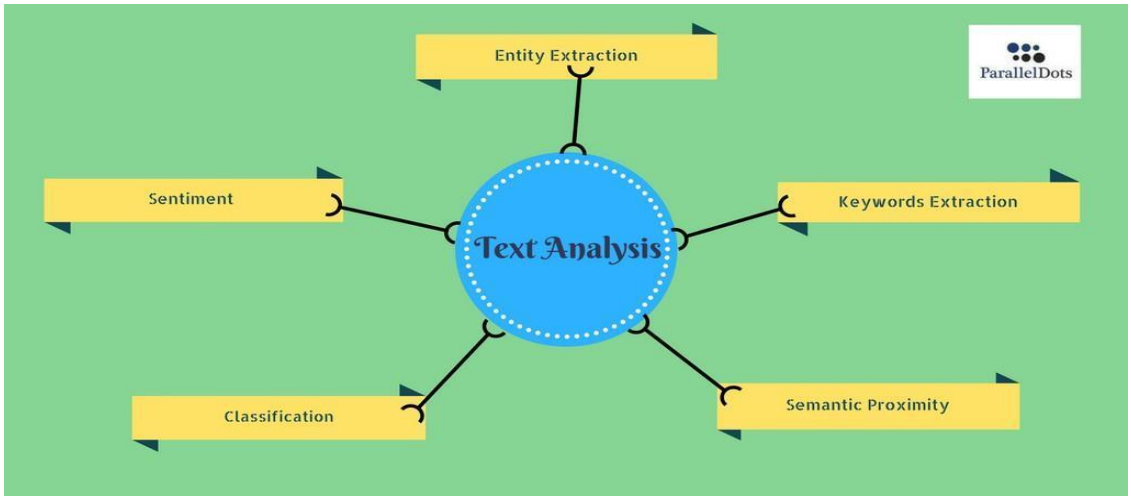

Figura 3 Aplicaciones del Text Analysis

Fuente: Medium.com [4]

Existen muchas ventajas para la aplicación de esta categoría al mundo empresarial, ya que permite la recopilación de la información en tiempo real, y la extracción de lo más relevante en cuestión de minutos utilizando algoritmos adecuados. Se caracteriza por su rapidez y la consistencia de los datos.

En este trabajo nos vamos a centrar en la clasificación de textos, que es un proceso automático de ordenación de una serie de textos en una serie de categorías predefinidas.

Esto se consigue con la ayuda de algoritmos de Machine Learning y de Natural Language Processing (NLP). Vamos a comentar brevemente en qué consiste este concepto.

Natural Language Processing es una rama de la inteligencia artificial que se concentra en conseguir que la máquina sea capaz de entender e interpretar el lenguaje humano. La información se da en distintos formatos, no solo en el

lenguaje hablado si no también en el lenguaje de los signos. Algunos ejemplos son comentarios en las redes sociales, páginas web o mensajes de voz.

La información no está en forma binaria, con lo que máquina no puede trabajar y obtener una interpretación rápida del mismo, si no que está en distintos formatos, como los anteriormente mencionados. Por tanto, es más difícil trabajar con los algoritmos tradicionales. Para ello, contamos con los algoritmos de ML, Deep Learning and Neural Networks que cuentan con distintos modelos que facilitan el procesamiento de los textos.

En cuanto a las aplicaciones y usos que se le han dado, uno de los primeros es el de la detección si un email es spam o no. Se trata de la intentar detectar si un mensaje es spam o no en función de la frecuencia y presencia de algunas palabras clave, como son dinero o premios, o mensajes que contiene saludos antes de empezar a contar algo.

Otra de las aplicaciones del NLP son los chatbox. En los que las máquinas son capaces de mantener unas conversaciones con los usuarios. Esta aplicación se está desarrollando y extendiendo en la actualidad, debido a su gran utilidad, en una gran cantidad de páginas como puede ser asesoramiento para las compras online, algunas oficinas y demás.

Pasos desarrollar:

Una vez comentado el significado de NLP, vamos a ver cuáles son los principales pasos para llevar a cabo la clasificación de los textos con ayuda del NLP.

Se puede clasificar en 4 grandes pasos:

1. Preparación de los datos
2. Selección de características
3. Entrenamiento del Modelo
4. Mejorar el modelo de clasificación

Primero de todo se tiene que obtener los datos con los que van a trabajar. El primer paso importante es la transformación de los datos a un formato limpio y organizado para que sea comprendido e interpretado de una manera correcta por los modelos NLP.

Para la reordenación de los datos y su transformación, se llevan a cabo pasos como la eliminación del ruido, normalización del léxico o la estandarización de los objetos, y otros un poco más complejas como Tokenization, Word Stemming o Lemmatization.

La primera técnica consiste en la partición de las frases o líneas de los textos en pequeñas partes, ya sean por palabras, letras o incluso en símbolos, que tengan un significado o valor para la interpretación.

Los dos siguientes tienen el mismo objetivo, reducir la dimensión de la palabra y dejarla en su formato original. La diferencia es que stemming no tiene en cuenta el contexto en el que se utiliza dicha palabra, de manera que palabras utilizadas en distintos contextos tiene un único significado para él.

De todas formas, esto no supone una razón para no utilizar esta técnica y recurrir a otras, debido a que esta falta de precisión no es tan relevante frente a su facilidad de implementación y la rapidez.

El segundo de los pasos es la selección de las características principales que van a ser usadas para la preparación y entrenamiento de los modelos. Los factores o elementos que de verdad tiene una importancia y definen a los textos son aspectos que no vienen dados de por sí, y tiene que ser definidos utilizando distintas técnicas como pueden ser Syntactical Parsing, Entities, N-grams, word-based features Statistical features y word embeddings.

Syntatic Parsing (Figura 4), consiste en hacer un análisis gramatical de las palabras de una frase, así como las relaciones que haya entre las mismas. Para ver las relaciones gramaticales de entre las palabras se utilizan técnicas como árboles de dependencia. Del mismo modo que los árboles de decisión, se trata de colocar cada una de las palabras con una estructura de árbol, y así poder apreciar las relaciones.

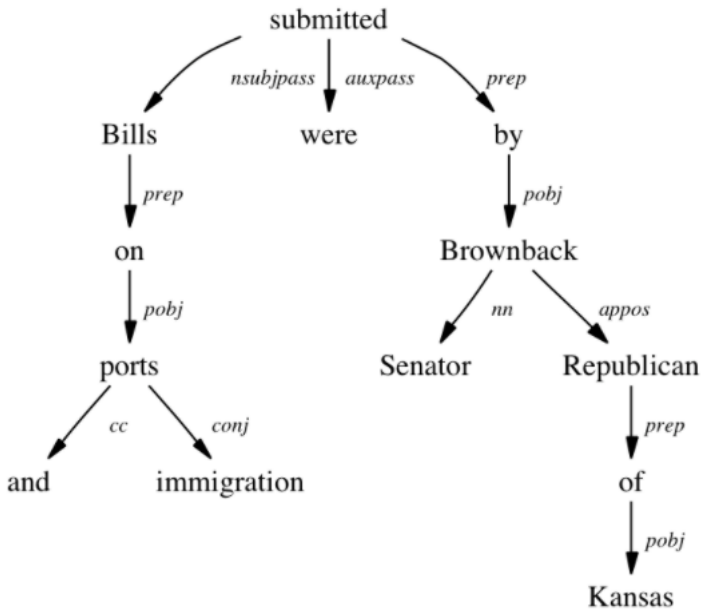

Figura 4 Ejemplo de Syntactic Parsing

Fuente: The Stanford Natural Language Processing Group [5]

Otra de las técnicas conocida como POS (Part of speech) tagging, se utiliza para complementar el NLP e intentar corregir los posibles fallos que pueda cometer este, como puede ser el caso de dos palabras similares. En este supuesto, el proceso de lematización no tenía en cuenta los contextos en los que se incluía y conducía a interpretaciones erróneas. De esta forma, mediante el POS, se tiene en cuenta el contexto y se evitan dichas confusiones.

De igual forma, otro de los puntos que se incluyen dentro de la selección de las características a estudiar, es el reconocimiento de los nombres de entidades o personas, parejas de dos nombres o más, o también puede darse el caso de que sea tiempos verbales complejos, formados por varios verbos.

Del mismo modo, se cuenta con los Diagramas N, que son representaciones de conjuntos de palabras. Es decir, a partir de una frase se obtienen las parejas, en el caso de que los diagramas sean de N=2 elementos, o agrupaciones de palabras.

Otra característica importante y paso previo a la generación del modelo es la transformación de los bloques de textos o palabras en vectores, para que puedan reducir la dimensión y ser fácilmente tratables por los modelos.

Entre los modelos que más se utilizan para llevar a cabo dicha transformación están el **Word2Vec** y **GloVe.**

Word2Vec es un algoritmo que utiliza las neural network para aprender las asociaciones entre las palabras a raíz de grandes bloques de texto. Cada una de dichas palabras representa un vector. Esta herramienta es útil y capaz de detectar sinónimos y realizar sugerencias conforme se van redactando frases.

Por otro lado, el algoritmo de GloVe es un modelo que se engloba bajo los denominados unsupervised learning. La conversión en su caso del texto a vectores se logra haciendo una representación de las palabras en un espacio determinado, dónde la distancia entre cada uno de los puntos refleja el parecido semántico que presentan.

Una vez que se han dado los primeros pasos y se cuenta con unos datos preparados para su utilización en los modelos, se selecciona algunas de las técnicas o algoritmos de machine learning para preparar correctamente el modelo.

Finalmente, se lleva a cabo la comprobación del modelo, si los resultados obtenidos son adecuados y correctos, y el modelo es capaz de realizar la clasificación de una eficaz y con una determinada eficacia.

3.3- Diferencias entre la estadística clásica y bayesiana

En la estadística, podemos encontrar varios enfoques o subconjuntos, como son la estadística inferencial o clásica y la bayesiana.

La estadística inferencial es la parte que incluye los métodos y procedimientos para la obtención de conclusiones de la población en su conjunto, a partir de una pequeña muestra de la misma.

Por su parte, la estadística bayesiana se basa en la evidencia, es decir en los grados de creencia o lo que es lo mismo, en las probabilidades bayesianas.

Por tanto, ¿dónde reside la diferencia principal entre estos dos subconjuntos de la estadística?

La clave está en la manera en la que conciben el concepto de probabilidad.

Para los frecuentistas, la probabilidad se define en términos de experimentación. Es decir, está relacionada con el número de veces que ocurre un suceso cuando se realiza el experimento x veces bajo las mismas condiciones. Se suele decir que intentan estudiar la probabilidad real de las cosas.

Para tomar las decisiones, se basan únicamente en los datos de los que se dispone, es decir, no tiene en cuenta aspectos relacionados que hayan ocurrido con anterioridad ni parámetros influyentes.

Por otro lado, están los bayesianos que afirman que la probabilidad es la certeza de que ocurra un cierto evento. Está basado en el Teorema de Bayes, y que se diferencia de la estadística frecuentista debido a la inclusión de información externa dentro del estudio.

Los resultados de estos dos tipos de métodos pueden ser iguales o no, en función del tipo de experimento en los que se aplica. En experimentos sencillos, que no presentan mucha complejidad, los resultados pueden ser muy parejos si no iguales, pero en otros casos que incluyen más parámetros discrepan.

Existen supuestos, que no son tan sencillos de repetir infinitas veces, debido a su complejidad o debido a la existencia de parámetros que son desconocidos pero que son necesarios para la realización de dichos experimentos para los frecuentistas.

En lo que respecta a la subjetividad, en ambos aspectos existe cierto punto. Los bayesianos al hacer las suposiciones o creencias sobre los hechos, y en el caso de los frecuentistas en cuanto a la fijación del nivel de confianza en los contrastes de hipótesis.

3.4- Software a emplear

Para el desarrollo de este proyecto vamos a trabajar con el programa Phyton. Es uno de los programas más desarrollados en el mundo de la estadística y se está convirtiendo en unos de los más empleados para su uso en temas relacionados con la Inteligencia Artificial y sus ámbitos. Cuenta con numerosas librerías o módulos de entre los que vamos a emplear la scikit-learn.

3.5- Naive Bayes

El primero de los modelos de clasificación de textos que se van a estudiar es el conocido como clasificador Naive Bayes. Es uno de los algoritmos más antiguos usados en Machine Learning, cuyo pionero es Thomas Bayes. Fue un matemático y estadístico que introdujo el uso de la probabilidad de una manera inductiva, es decir, una forma de calcular la probabilidad de que un suceso vuelva a ocurrir, basándose en los hechos anteriores, el número de veces que ha ocurrido un suceso.

Uno de los aspectos que se introdujeron con esta filosofía fue la inclusión del parámetro de la incertidumbre. Siguiendo esta forma de pensar, no hay ninguna opción que sea correcta y el resto no lo sean. Se refiere al hecho de que se seleccionará aquella con la mayor probabilidad de ocurrencia.

Este algoritmo se basa en, teniendo en cuenta la probabilidad de ocurrencia de los sucesos, hacer las clasificaciones en distintas clases. Lo que caracteriza este método son dos cosas:

- Bayes: basado en el teorema de Bayes.
- Naive: Debido a la complejidad de cálculo de algunas probabilidades condicionadas, se asume la independencia de los sucesos para facilitar los cálculos. Aunque este no sea el caso en los acontecimientos reales.

El teorema de Bayes se utiliza para calcular la probabilidad de un determinado suceso, contando con información previa sobre el mismo. Se parte de la existencia de dos sucesos, A y B.

Dicho teorema cuenta con un enfoque distinto al de la probabilidad inversa o total, que pretende hacer inferencia sobre un suceso B a partir de la información o resultados del A. Por otra parte, Bayes busca la probabilidad condicionada de A sobre B.

Los elementos clave o base del teorema de Bayes son:

- P (Suceso A): Probabilidad de ocurrencia de A
- P (Suceso B): Probabilidad de ocurrencia de B
- P(A|B): Probabilidad de que ocurra A, teniendo en cuenta que sucede B.
- P(B|A): Probabilidad de que ocurra B, teniendo en cuenta que sucede A.

Dichas probabilidades de los eventos o sucesos son calculadas a posteriori, es decir basándose en las frecuencias observadas.

Otro elemento que se requiere para el teorema de Bayes es la regla del producto. Esta regla se basa en el simple hecho de que, si los dos sucesos ocurren de manera simultánea e indistintamente del orden, el producto se puede escribir como:

$$P(A * B) = P(A|B) * P(B)$$
$$P(B * A) = P(B|A) * P(A)$$

Aplicando estas dos reglas, se llega a la famosa fórmula del teorema de Bayes es la siguiente:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Una vez que sabemos cómo funciona el teorema base, podemos pasar a ver el funcionamiento del algoritmo. No siempre se da el caso de tener únicamente dos sucesos o eventos, y más en el caso de la clasificación de textos en la que suelen existir más categorías o clases.

Por tanto, se define un primer conjunto X, formado por los inputs que se quieren clasificar. Son los datos que se le proporcionan al algoritmo para que nos haga la posterior agrupación.

Por otro lado estaría el conjunto de los K clases, siendo cada k1 …kn las posibles clases en las que se vana a agrupar los x1….xn elementos dados.

Asumiendo la hipótesis de independencia de los sucesos se puede llegar a la fórmula final de:

$$P(y_k|x) = \frac{P(x|y_k) * P(y_k)}{P(x)}$$

Y haciendo las oportunas simplificaciones la fórmula del clasificador naive bayes es:

$$v_{NB} = arg\ arg\ max_{v_j \epsilon V} P(v_j) \prod_i \quad P(a_i|v_j)$$

Otra de las asumpciones que hace dicho modelo es la conocida como la regla Laplace. Es decir, para realizar la clasificación de los elementos, se requiere la probabilidad de ocurrencia de uno de los sucesos. Sin embargo, surge un problema cuando uno de los eventos no ha ocurrido ninguna vez, por tanto su probabilidad es 0 y el resultado final sería erróneo. Pero el hecho de que no haya aparecido un evento en nuestro set de entrenamiento, no supone que no vaya a salir o que no exista la posibilidad de que pueda salir en futuros análisis.

Es por ello que se suele añadir una unidad a cada uno de los recuentos, y de esta manera los resultados son proporcionales y no se produce ninguna modificación. En supuesto de que un elemento no ocurra, al menos al hacer el producto de probabilidades, los valores son inferiores y evitan que se hagan clasificaciones erróneas.

Dicha aplicación se puede entender mejor con el ejemplo más común de clasificar los e-mail según sean spam o no. En este supuesto, si contamos en el conjunto de emails que se usa para el entrenamiento, las frecuencias de las palabras, puede darse el caso que luego en el email nuevo que entra para su clasificación no contenga ninguna de esas palabras. En este caso, al hacer el producto de las probabilidades nos da un valor de 0, y por tanto el programa interpreta que al no contener ninguna de dichas palabras clave, el email no es engañoso y lo clasifica como no spam. Por tanto, si se le suma una unidad a todas las palabras que se estudian, se consigue hacer una corrección y evitar hacer una mala clasificación.

3.6- Support Vector Machine

Otro de los algoritmos que muestra buenos resultados en aplicaciones del mundo real es el Support Vector Machine. Este método, basado en la teoría del álgebra lineal, se basa en la idea de separar dos clases usando una línea recta, en una dimensión, o un hiperplano, en dos dimensiones o un hiperplano de n-1 cuando existen n dimensiones o categorías. Vamos a ver el funcionamiento básico en dos dimensiones, tal y como se muestra en la siguiente Figura 5.

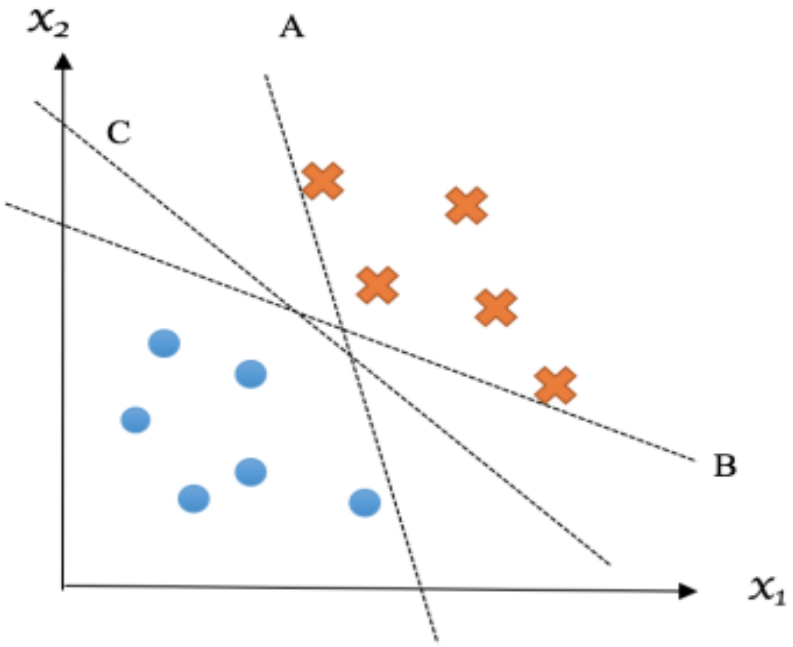

Figura 5 Posibles hiperplanos de corte del algoritmo

Fuente: Python Machine Learning By Example-The easiest way to get into machine learning [1]

Contamos con una serie de observaciones, que van a ser nuestro conjunto de datos para el entrenamiento del modelo. Si queremos separar las cruces y los círculos, podemos imaginar muchas combinaciones que nos hacen la separación o división clara en los dos conjuntos (Figura 6). Pero, imaginemos que nos dan una nueva observación para que clasifiquemos en una de las categorías y ocurre lo siguiente.

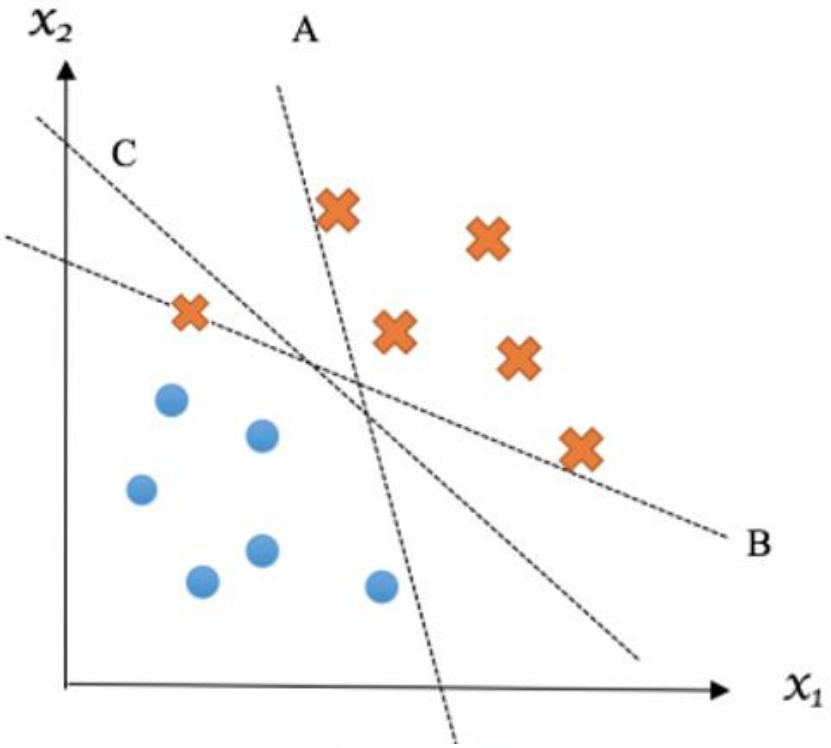

Figura 6 Situación cuando sale outlier

Fuente: Python Machine Learning By Example-The easiest way to get into machine learning [1]

Con ninguna de las opciones anteriores se haría la agrupación de una manera correcta. Por tanto, surge ahí la siguiente pregunta, ¿qué hay que tener en cuenta para la selección del hiperplano correcto?

La respuesta es la siguiente: Se trata de seleccionar aquellos puntos o soportes más cercanos al hiperplano seleccionado, tanto por el lado negativo como por el positivo. De esta forma se obtienen los hiperplanos de cada uno de los lados. Y la distancia entre los puntos soportes y el hiperplano, es lo que se denomina margen. La forma de seleccionar el hiperplano definitivo es que se maximice todo lo posible dicho margen (Figura 7).

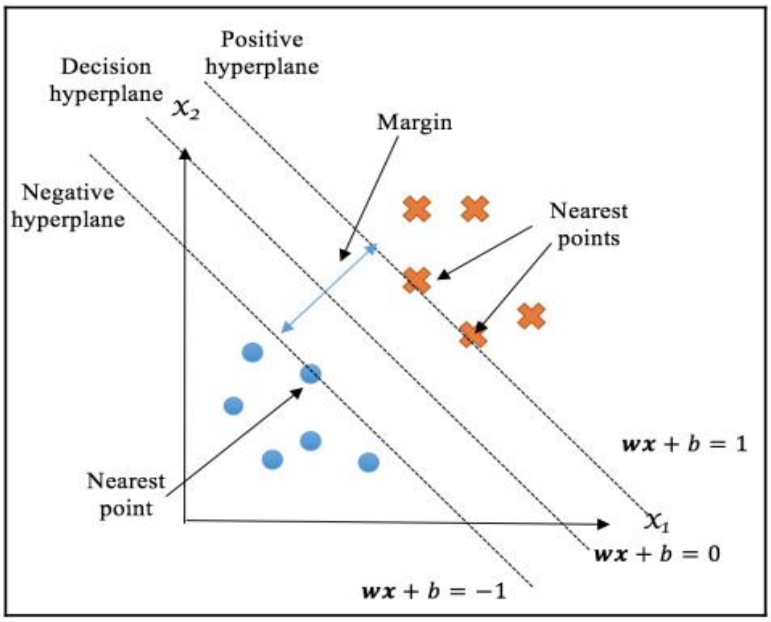

Figura 7 Elementos del algoritmo Support Vector Machine

Fuente: Python Machine Learning By Example-The easiest way to get into machine learning [1]

De esta forma se consigue solucionar dicho problema. Ahora bien, esto era un ejemplo de clasificación en dos clases, pero pueden darse situaciones en la vida real, y de hecho se dan, donde se requiere de clasificar en más de dos clases.

Para ello existen dos métodos que pueden ayudar a realizar este tipo de clasificaciones. Ambos de ellos se basan en la transformación de los datos a los casos binarios a través de:

- Comparaciones one-against-one: es decir se van comparando cada uno de los pares entre ellos.
- Comparaciones one-against-all: en este caso, una pareja concreta se va comparando con cada una del resto.

Cabe destacar que ambos modelos requieren de un esfuerzo computacional muy grande. A pesar de ello, los resultados que se obtienen son muy buenos y

vale la pena su aplicación. Dependiendo de qué función de la librería sklearn utilicemos, se aplicará uno u otro modelo de manera automática. El que mejor resultados aporta, y además se consigue reducir el esfuerzo computacional es el de one-against-all, que es el utilizado por la función LinearSVM por defecto.

4.- OBJETIVOS

En este trabajo nos vamos a centrar en estudiar dos casos concretos de entre los muchos que nos rodean en la vida cotidiana, como son por una parte detectar si un email es spam o no, y por otro lado la clasificación de noticias en unas categorías preestablecidas. Además, analizar como pueden ser tratados y realizados de manera automática y sin necesidad de tener a alguien en todo momento supervisando, gracias a los algoritmos de machine learning.

Lo que se pretende es, en cada una de las situaciones, ver la eficacia con la que se consiguen resolver los problemas mediante la aplicación de los dos algoritmos seleccionados para este propósito, el método de Naive Bayes y el método de Support Vector Machine.

Decidiremos cuál de ellos es el más adecuado en base a la precisión con la hagan las clasificaciones, es decir, se seleccionará cada uno en base a que presente el mayor acierto en las agrupaciones, y por consiguiente cometa menos errores.

Además, se pretende comparar los resultados, en el supuesto de la clasificación de las noticias, prescindiendo de unas operaciones importantes en el pre procesamiento de textos como es la eliminación de los ‘stopwords’. Estas son las palabras clave en un idioma, que son muy comunes y necesarias para la comunicación, pero que no nos aportan información a la hora de hacer las agrupaciones en categorías como son los artículos o los pronombres.

5.- APLICACIÓN PRÁCTICA

El primero de los supuestos es el de decidir si un email es spam o no lo es. Vamos a aplicar los dos métodos, y veremos cuál de ellos nos proporciona mejores resultados. Para ello usaremos un conjunto de datos de un repositorio que se denomina UCI Machine Learning Repository. En esta página se encuentran una gran cantidad de bases sobre distintos temas de la realidad y de nuestro día a día que nos pueden ser de ayuda.

La base seleccionada se denomina SMSSpamCollection y está formada por un total de 1393 registros, de los cuales 1208 son con contenido spam, y los 185 restantes son correos válidos.

El segundo de los ejemplos es el de la clasificación de noticias en las temáticas correspondientes. Para este supuesto los datos fueron recogidos de manera manual, a lo largo de una semana. Se recogió una noticia de cada categoría por día, hasta obtener un total de 40 noticias, 10 por cada una de las clases. Fueron categorizadas de la forma en que se muestran en la Tabla 1:

Categoría	Código
1	Deportes
2	Economía
3	Ciencia
4	Educación

Tabla 1 Categorías de las noticias recogidas

Un paso importante, es que las noticias fueron recogidas y guardadas según la categoría, es decir, los datos no son aleatorios. Con esto nos referimos a lo siguiente: en la base de datos aparecen primero las 5 noticias de la categoría 1, después las 5 de la categoría 2, y así con las 4 categorías. A pesar de que el train y test realiza la división de manera aleatoria, puede darse el caso que el train seleccione el 75-80% registros de las primeras 3 categorías, y las pertenecientes a la categoría 4 se queden en el test. En este caso, no se

aprende ningún vocabulario clave de la categoría 4 y no va a ser capaz de distinguirla del resto. Es por ello que se recuerda la importancia de hacer una reordenación.

Para ello, tenemos dos opciones: la primera de ellas es aleatorizar desde el propio Python empleando el comando necesario. Y la segunda opción, es barajar las opciones desde el archivo de Excel.

Como en todo análisis de textos, hay que hacer un preprocesamiento previo del input, para que la información que se extraiga sea lo más relevante para nosotros. Las funciones de la librería scikit-learn, como son el Countvectorizer o la TfidfVectorizer, ya nos realizan las transformaciones necesarias como puede ser la conversión de las letras mayúsculas a minúsculas, entre muchas otras.

Pero en este caso realizaremos este preprocesamiento previo para intentar reducir el esfuerzo computacional y que no requiera de mucha espera la clasificación.

Para ver cómo realmente sí que influye la eliminación de las palabras clave en el resultado, en el ejemplo de la clasificación de las noticias vamos a realizar las comparaciones entre tres casos distintos:

1- Agrupación sin la eliminación de los stopwords. Es decir, únicamente trabajando los caracteres especiales, pero manteniendo los stopwords, a ver cómo trabajan las funciones por defecto.
2- Agrupación descartando las stopwords y aplicando la función Countvectorizer.
3- Agrupación descartando las stopwords y aplicando la función TfidfVectorizer.

Una vez que tenemos comentados las aplicaciones que vamos a realizar, y de manera previa al análisis de los resultados obtenidos, es necesario que introduzcamos los indicadores que vamos a utilizar para la selección del modelo adecuado. Su correcta comprensión es importante para que las interpretaciones que se hagan no sean erróneas.

El indicador más utilizado, y que generalmente es la forma más adecuada y fiable para extraer conclusiones sobre la eficacia de nuestros modelos, es la denominada matriz de confusión (Tabla 2). En dicha matriz se recogen en las filas los valores reales de las n clases, y en las columnas se representan los valores que han sido predichos en sus respectivas n categorías.

Matriz de Confusión	Positivos	Negativos
Positivos	TRUE POSITIVE	FALSE POSITIVE
Negativos	FALSE NEGATIVE	TRUE NEGATIVE

Tabla 2 Ejemplo Matriz de Confusión y sus elementos

A partir de dicha matriz se pueden calcular distintos ratios o errores. En el campo de la medicina se suelen utilizar mucho la nomenclatura de Error del tipo I o Error del tipo II. Vamos a comentar los distintos ratios que se pueden derivar de dicha matriz.

El primero que nos viene a la cabeza cuando pensamos en hacer la evaluación de un modelo de clasificación es el de la precisión. Este ratio nos indica cuál es la número de predicciones correctas que realiza nuestro modelo de una categoría y se define como:

$$\frac{True\ Positives}{True\ Positives + False\ Positives}$$

Pero este ratio no siempre es el más adecuado para la evaluación de nuestro modelo, ya que puede darse el caso de que un alto valor de precisión no se traduzca en un alto porcentaje de acierto en la clasificación.

Imaginemos el siguiente ejemplo: contamos con un almacén donde se fabrican dos clases de productos, el producto A y el producto B, cada uno de ellos con unas características determinadas y que requieren de ciertas condiciones de almacenaje. Hemos entrenado el modelo de tal forma que, con observar las características propias, tenga que decidir si se guarda en un compartimento u otro.

Le damos un total de 100 productos para clasificar, siendo 90 de ellos del grupo 1 y 10 de ellos del grupo 2. Si el algoritmo nos presenta una precisión del 90%, en un principio podemos pensar que funciona bastante bien. Pero en el fondo, puede darse el caso que al tener una mayor frecuencia de una categoría determinada, diga que todas pertenecen a ese grupo y ninguno al otro grupo.

La precisión es alta, pero no nos clasifica ningún producto del grupo 2. Simplemente nos dice que todos van al grupo 1 y listo. Es por ello, que es necesario buscar otros indicadores.

El segundo de ellos es el Recall, que se define como:

$$\frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Este indicador también se suele denominar sensibilidad, se refiere a la capacidad de encontrar todos los True Positives values.

Una forma de simplificar estos dos ratios y facilitar la interpretación en las comparativas es el F1-Score. Se trata de una media armónica de la precisión y recall. Es importante destacar que no es una media estándar, donde se le otorga el mismo peso a sus componentes, sino que en este caso se le otorga un mayor peso a los valores más bajos. Esto tiene como consecuencia directa que, únicamente van a tener valores de f1-score altos aquellos que posean altos porcentajes tanto de precisión como de recall.

Esto ayuda entre otras cosas a solucionar la problemática que se creaba en el cálculo de la precisión, cuando una categoría presentaba una mayor frecuencia, y por tanto obtenía un alto porcentaje de precisión, y en cambio no era capaz de hacer la diferenciación, lo que se traduce en un menor recall.

La fórmula que usa el F1-score es la siguiente:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall} = \frac{True\ Positives}{True\ Positives + \frac{FN + FP}{2}}$$

Este indicador también presenta un dilema, ya que una alta precisión en muchas ocasiones se traduce en un bajo porcentaje de recall, y viceversa. Dependiendo del caso en que nos encontremos, nos será de mayor importancia el distinguir de forma clara y contundente dos clases o categorías, como puede ser en la predicción si un cliente, conforme a sus características, me va a devolver un préstamo bancario que le concedo como entidad bancaria.

Es mucho más importante saber si tendré dificultades en el cobro o no, que el porcentaje de personas que me lo devuelven. Es decir, es mejor incurrir en este caso en un menor porcentaje de precisión, y mayor recall.

Por tanto, la selección del indicador se utilizará en función de los intereses en cada determinada situación.

Por último, uno de los indicadores que se pueden utilizar para la evaluación de forma gráfica de clasificadores o modelos binarios es la curva de ROC.

La Curva de Roc consiste básicamente en la representación de dos ratios: True Positives y False Positives, o también conocidos como sensibilidad y 1-especificidad.

Tal y como se puede apreciar en la Figura , hay una curva diagonal que divide el gráfica de esquina inferior izquierda a esquina superior derecha (línea de no-discriminación). Cualquier punto que se predijera de manera aleatoria tendría que caer en algún punto a lo largo de dicha curva. Cualquier punto por encima de dicha diagonal, significa que es un buen resultado y la interpretación cambia cuando el punto está por debajo.

Cuanto más cercano esté el punto a la esquina superior izquierda, mejor va a ser nuestro modelo. Significa que nuestro modelo no comete ningún error, y es lo que se llama una clasificación perfecta.

El indicador se traduce en valor como el área que queda debajo de la curva ROC (Area Under the Curve). Los valores próximos del AUC a partir del 0.75 se consideran que los modelos realizan buenas clasificaciones. A partir de ahí, se puede utilizar para realizar las comparativas.

Una vez que tenemos todas las herramientas para hacer las evaluaciones y elegir los mejores modelos, podemos proceder a comentar los resultados obtenidos.

6.- RESULTADOS

En este apartado vamos a comentar los resultados obtenidos en nuestros dos experimentos. La forma de proceder ha sido en líneas generales similar, con un primer preprocesamiento y transformación del texto en un lenguaje fácil de manejar de forma computacional y más efectivo.

Cabe indicar que, en cuanto a la división entre el conjunto de datos en train y test, se ha utilizado unos porcentajes del 75 y 25 por ciento respectivamente. Estos parámetros han sido comunes en ambos supuestos.

Supuesto 1

En la primera aplicación, referida a la determinación de si un email es spam o no, una vez que se entrenó el modelo y se realizaron las predicciones, los resultados más relevantes que se obtuvieron fueron los siguientes:

Naive Bayes	Spam	No spam
Spam	1190	18
No spam	11	174

Tabla 3 Matriz de confusión modelo Naive Bayes

Con la observación de la matriz de confusión, vemos como únicamente el modelo se equivoca en la clasificación de 29 emails, de un total de 1393. Lo cual es un error muy pequeño en función de la cantidad de emails que se tenían que clasificar.

Si nos fijamos en los siguientes ratios de la Tabla 5, vemos como el modelo es bastante bueno. Atendiendo a los ratios referenciados, resalta que en media presenta un 98% de precisión, es decir, que la clasificación de los emails buenos es casa segura, con un error muy pequeño. Y a su vez, la

diferenciación entre las dos clases también lo es, con un 98%. Por tanto, el modelo de Naive Bayes funciona bastante bien en dicho supuesto.

Ahora realizamos la misma clasificación, pero en este caso utilizando el algoritmo de Support Vector Machine los resultados son incluso mejor.

SVM	Spam	No spam
Spam	1204	4
No spam	15	170

Tabla 4 Matriz de confusión modelo SVM

El número de falsos positivos se reduce de manera significativa con respecto al Naive Bayes, mientras que el de falsos negativos aumenta de manera muy ligera, en 4 unidades (Tabla 4).

Ratios	Precision	Recall	F1 Score	ROC/AUC
Naive Bayes	0.979607	0.979182	0.979345	0.987059
SVM	0.986276	0.98636	0.986182	0.986567

Tabla 5 Comparativa de los ratios entre los dos algoritmos

Además, los ratios son superiores en un 1% respecto al modelo anterior. La capacidad del modelo de hacer la diferenciación entre las clases aumenta, además de la correcta clasificación de los emails buenos.

Por último, vamos a echar un vistazo a la curva ROC y el área que queda por debajo de la misma.

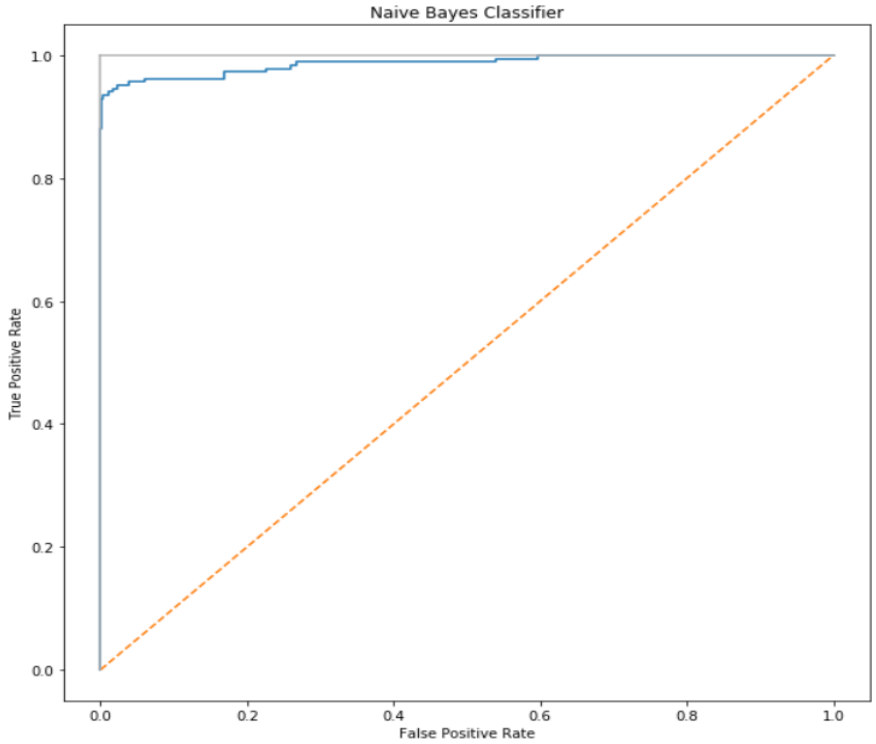

Figura 8 Representación de la Curva de Roc algoritmo Naive Bayes

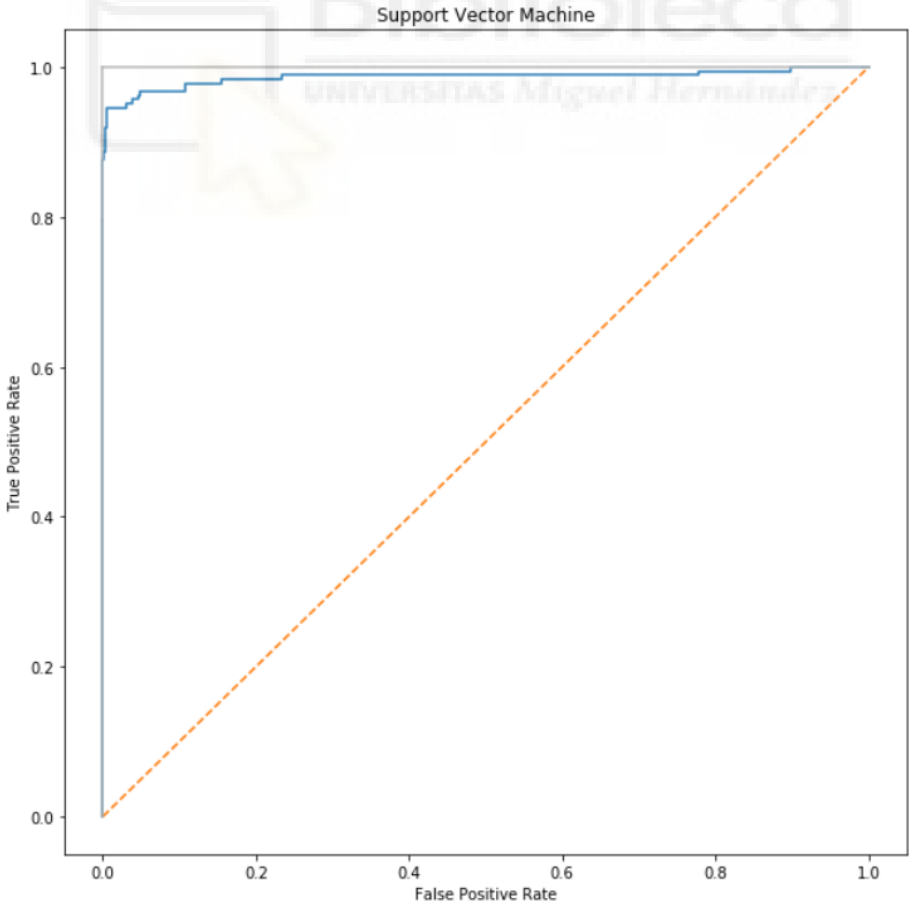

Figura 9 Representación de la Curva de Roc algoritmo SVM

En cuanto a la curva de ROC, vemos como no existen muchas diferencias,las Figuras 8 y 9, son prácticamente idénticas, y como consecuencia el área bajo la curva (AUC) es similar en ambos casos, con un valor de 0.987.

Podemos concluir entonces que en este caso la curva de Roc no nos ayuda en la selección de alguno de los modelos ya que proporciona resultados similares.

El mejor modelo, aunque por poca diferencia sería el de SVM, ya que mejora los porcentajes de precisión y recall del Naive Bayes.

Supuesto 2

Para este segundo supuesto, en la detección de las categorías de los emails, vamos a hacer la comparativa de los tres casos comentados anteriormente: El caso uno corresponde a la clasificación sin hacer el filtrado de los stopwords. El caso dos hace referencia aplicando el filtrado de los stopwords y aplicando la función CountVectorizer y el caso tres consiste en aplicar el filtrado y el uso de la función TfdifVectorizer.

Caso 1

Los resultados obtenidos son bastantes claros, y se pueden apreciar en las siguientes matrices de decisión y de ratios.

SVM	Deportes	Economía	Ciencia	Educación
Deportes	2	0	0	0
Economía	1	1	0	3
Ciencia	1	0	1	0
Educación	1	0	0	0

Tabla 6 Matriz de decisión SVM sin filtrado de stopwords

Naive Bayes	Deportes	Economía	Ciencia	Educación
Deportes	2	0	0	0
Economía	0	0	0	5
Ciencia	0	0	1	1
Educación	0	0	0	1

Tabla 7 Matriz de decisión Naive Bayes sin filtrado de stopwords

Ratios	Precision	Recall	F1 Score
SVM	0.78	0.4	0.414286
Naive Bayes	0.414286	0.4	0.358333

Tabla 8 Resumen Ratios caso 1

Observando las tablas 6 y 7, queda bastante claro que los modelos no son capaces de adaptarse bien a los casos prácticos y hacer las predicciones de una manera razonable.

Fijándonos en la Tabla 8, podemos apreciar como los porcentajes no alcanzan ni el 50% de precisión ni recall, es decir, no son capaces en algunos casos de distinguir las distintas categorías.

Un ejemplo claro de ello es, en el modelo de Naive Bayes, no es capaz de distinguir y clasificar los registros en la categoría Economía. Por su parte, ocurre algo similar con el modelo SVM, pero con la categoría Educación.

Habiendo visto estos resultados, en los dos modelos, nos hace suponer que el paso es necesario para garantizar cierta efectividad en las clasificaciones.

Caso 2

Las clasificaciones son mucho más acertadas en este caso, tal y como se aprecia en las tablas 9 y 10, respectivamente.

El modelo de SVM únicamente comete un único error, clasifica una noticia económica en la categoría de educación. A pesar de ese caso concreto, el resto las agrupa bien. (Tabla 9)

En cambio, el modelo Naive Bayes presenta una menor precisión en la clasificación, ya que comete 3 errores, y no es capaz de identificar ninguna observación correctamente en la categoría de educación. (Tabla 10)

SVM	Deportes	Economía	Ciencia	Educación
Deportes	2	0	0	0
Economía	0	4	0	1
Ciencia	0	0	2	0
Educación	0	0	0	1

Tabla 9 Matriz de confusión modelo SVM y CountVectorizer

Naive Bayes	Deportes	Economía	Ciencia	Educación
Deportes	2	0	0	0
Economía	1	4	0	0
Ciencia	1	0	1	0
Educación	0	1	0	0

Tabla 10 Matriz de confusión modelo Naive y CountVectorizer

Ratios CountVec	Precision	Recall	F1 Score
SVM	0.95	0.9	0.911111
Naive Bayes	0.7	0.7	0.666667

Tabla 11 Resumen Ratios caso 2

Las diferencias entre los dos algoritmos usando la función Countvectorizer nos deja claro que los mejores resultados nos los aporta el modelo de SVM, con una precisión por encima del 90% (Tabla 11). Esto ya es más del doble que lo que nos daba sin el filtrado. Además, la diferenciación entre las categorías no es tan difícil de apreciar por el algoritmo y la clasificación es mucho más eficaz.

Caso 3

El tercero de los casos, a diferencia del caso 2, aplicamos la función TfidfVectorizer, que es una forma distinta de codificación que incluye la librería sklearn.

Tal y como se puede apreciar en las tablas 12 y 13, los modelos pierden eficacia a la hora de realizar las agrupaciones. Cometen más errores que en el caso 2.

SVM	Deportes	Economía	Ciencia	Educación
Deportes	2	0	0	0
Economía	0	1	0	4
Ciencia	0	0	2	0
Educación	0	0	0	1

Tabla 12 Matriz de confusión modelo SVM y TfidfVectorizer

NB	Deportes	Economía	Ciencia	Educación
Deportes	2	0	0	0
Economía	0	0	0	5
Ciencia	0	0	1	1
Educación	0	0	0	1

Tabla 13 Matriz de confusión modelo Naive Bayes y TfidfVectorizer

Ratios	Precision	Recall	F1 Score
SVM	0.92	0.6	0.6
Naive Bayes	0.414286	0.4	0.358333

Tabla 14 Resumen Ratios caso 3

Si nos fijamos en la tabla 14, podemos ver como el F1 score se reduce mucho con respecto al caso 2. Es decir, que los resultados varían bastante según la función que se aplique. El modelo SVM presenta un alto porcentaje de precisión, 92%, mientras que el de recall es de sólo un 60%. Por otro lado, el modelo Naive Bayes presenta unos valores muy bajos en ambos casos.

7.- CONCLUSIONES Y PROPUESTAS

En este trabajo se han estudiado dos algoritmos de Machine Learning y su aplicación a la clasificación de textos de dos situaciones comunes y del día a día. La función principal que se perseguía era hacer una comparativa de que modelo era el más adecuado para cada caso.

Una vez realizamos los cálculos y obtuvimos los resultados, llegamos a dos conclusiones principales. Primero, la necesidad de hacer una buena preparación de los datos, para que luego los resultados sean más acordes a lo que se intentó predecir, y que no se de el caso que nos lleven a descartar nuestro modelo. El correcto procesamiento de los datos incluye la eliminación de caracteres especiales y de palabras clave (stopwords).

Por otro lado, la selección de las funciones a emplear es también de gran importancia. Hemos visto como la función CountVectorizer es la que mejores resultados de cara a la clasificación nos daba. Por su parte, la función TfidfVectorizer no nos daba unos resultados buenos ya que emplea una ponderación inversa, es decir que a las palabras que más veces se repiten les

otorga un menor peso ya que interpreta que se consideran de menor importancia. Por eso, cuando eliminamos las palabras clave los resultados con esta función no son del todo acertados. En cambio, si no eliminamos los stopwords, el CountVectorizer proporcionaría malos resultados y estos serían mejores con el TfidfVectorizer.

Por último, hemos demostrado cómo en los supuestos seleccionados el mejor algoritmo para la clasificación es el de Support Vector Machine.

En la clasificación de emails, las diferencias no eran tan significativas entre los dos modelos, por lo que no importaría si se emplea uno u otro. Sin embargo, en la clasificación de las noticias, sí que la elección del modelo influye en los resultados.

8.- ANEXOS

Anexo 1

Clasificación de los emails en función de si son spam o no.

```
# Importamos la base de datos

import pandas as pd
messages = pd.read_csv('smsspamcollection/SMSSpamCollection', sep='\t',
names=["label", "message"])

# cargamos las librerías necesarias para hacer el tratamiento del input
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

# Descargamos los stopwords,que nos van a ayudar a mejorar la limpieza de
texto irrelevante
# nltk.download('stopwords')
# Ejecutamos la función para que nos divida los mensajes y transforme todas
```

```
# las letras a minúsculas.
ps = PorterStemmer()
corpus = []
for i in range(0, len(messages)):
    review = re.sub('[^a-zA-Z]', ' ', messages['message'][i])
    review = review.lower()
    review = review.split()

    review = [ps.stem(word) for word in review if not word in
stopwords.words('english')]
    review = ' '.join(review)
    corpus.append(review)

# Creamos lo que se denomina la mochila de palabras, es decir el diccionario
que forma y del que aprende el modelo.

from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform(corpus).toarray()


y=pd.get_dummies(messages['label'])
y=y.iloc[:,1].values


# Una vez preparado el dataset, dividimos en dos partes, el conjunto para
entrenar el modelo (train) y el que
# utilizaremos para probar su efectividad (test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)


# Aplicamos primero el modelo Multinomial de Naive Bayes
from sklearn.naive_bayes import MultinomialNB
```

```
spam_detect_model = MultinomialNB().fit(X_train, y_train)

# hacemos la predicción
y_pred=spam_detect_model.predict(X_test)

# obtenemos la precisión de nuestro modelo, accuracy
accuracy1=spam_detect_model.score(X_test, y_test)

# obtenemos la descomposición de la precisión, en sus tres variantes
(precision, recall, f1score)

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
spam_detect_model.score(X_test, y_test)


# obtenemos la matriz de decisión del primer modelo
from sklearn.metrics import confusion_matrix
matriz1=confusion_matrix(y_test, y_pred)

from tabulate import tabulate
encabezado=['Naive Bayes','Spam','No spam']
indice=['Spam','No spam']
print(tabulate(matriz1, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))


# Ahora hacemos la agrupación con el segundo modelo, el Support Vector
Machine
from sklearn import svm
from sklearn.calibration import CalibratedClassifierCV

SS = svm.LinearSVC()

# Para hacer la posterior representación de la curva de ROC, necesitamos
```

```
obtener las probabilidades, y para ello usamos
# la función CalibratedClassifier

clf_svm = CalibratedClassifierCV(base_estimator=SS)

# Ajustamos el modelo y realizamos la predicción
modelo2=clf_svm.fit(X_train, y_train)
pred_mod2=modelo2.predict(X_test)

# Descomposición de la precisión del segundo modelo
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_mod2))

# Matriz de decisión modelo SVM
matriz2=confusion_matrix(y_test, pred_mod2)


encabezado=['SVM','Spam','No spam']
indice=['Spam','No spam']
print(tabulate(matriz2, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

accuracy2=modelo2.score(X_test, y_test)

# Para hacer la comparativa con la curva de ROC, primero tenemos que
obtener las probabilidades y después obtenemos los scores

from sklearn.metrics import roc_curve, roc_auc_score

y_score1 = spam_detect_model.fit(X_train, y_train).predict_proba(X_test)[:,1]
y_score2 = clf_svm.fit(X_train, y_train).predict_proba(X_test)[:,1]

false_positive_rate1, true_positive_rate1, threshold1 = roc_curve(y_test,
y_score1)
false_positive_rate2, true_positive_rate2, threshold2 = roc_curve(y_test,
y_score2)
```

```
roc1=roc_auc_score(y_test, y_score1)
roc2=roc_auc_score(y_test, y_score2)


# Graficamos los valores de la curva ROC
import matplotlib.pyplot as plt

plt.subplots(1, figsize=(10,10))
plt.title('Naive Bayes Classifier')
plt.plot(false_positive_rate1, true_positive_rate1)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

plt.subplots(1, figsize=(10,10))
plt.title('Support Vector Machine')
plt.plot(false_positive_rate2, true_positive_rate2)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()


# Tabla resumen y comparativa entre la precisión y la curva ROC de los dos
modelos
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score

precision_mod1=precision_score(y_test, y_pred, average='weighted')
precision_mod2=precision_score(y_test, pred_mod2, average='weighted')
```

```
recall_mod1=recall_score(y_test, y_pred, average='weighted')
recall_mod2=recall_score(y_test, pred_mod2, average='weighted')

f1_mod1=f1_score(y_test, y_pred, average='weighted')
f1_mod2=f1_score(y_test, pred_mod2, average='weighted')

summary1=[[precision_mod1,recall_mod1,f1_mod1,roc1],[precision_mod2,recal
l_mod2,f1_mod2,roc2]]
encabezado=['Ratios','Precision','Recall','F1 Score','ROC/AUC']
indice=['Naive Bayes','SVM']
print(tabulate(summary1, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))
```

Anexo 2

Clasificación noticias en categorías.

```
# Cargamos las primeras librerías que nos vamos a necesitar para la limpieza y

# preparación de los datos

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import re
import string

# En el caso de que no tuviéramos los registros aleatorizados, cargamos los
originales y después los aleatorizamos
# usando la función sample e indicando un valor cualquiera, en este caso el 1.
# una vez que esté estandarizado, podemos escribirlo a un archivo nuevo de
```

```
excel y así para la próxima vez que queramos usarlo
# únicamente tendremos que cargar la base.

# df = pd.read_excel (r'datos_noticias.xlsx')
# df= df.sample(frac=1)
# newsshuffled=pd.DataFrame(df)
# df.to_excel('./news2.xlsx', sheet_name='Topics', index=False)


# Cargamos los datos nuevos.
df = pd.read_excel (r'news2.xlsx')

df.shape

(40, 2)

# A continuación, creamos una función que nos elimine los caracteres
especiales #como son las barras, los signos de porcentajes, transformar todas
las mayúsculas a #minúsculas.

def corregir(text):
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub("\\W"," ",text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text

# Ahora aplicamos esta función a nuestra data frame. Haremos una réplica de
la base, # ya que vamos a analizar el efecto de la eliminación de los stopwords.

df["descripcion"] = df["descripcion"].apply(corregir)
df1=df
```

```
#Primero estudiamos el caso en el que no eliminamos los stopwords o palabras clave.

# Ahora que tenemos los datos libres de caracteres especiales, tenemos que #transformar las frases y palabras a un lenguaje con el que phyton pueda tomarlo #como vectores, de forma que pueda aprender nuevo vocabulario y que lo pueda #codificar para su posterior uso en las clasificaciones.

#Cuando aplicamos esta función, Countvectorizer, ya por defecto se realizan las #transformaciones que hemos realizado de forma previa, pero para agilizarlo lo #hacemos antes. Además, al hacerlo de manera manual los resultados son mejores #ya que se realiza con una mayor precisión.

#Dividimos en train y test y aplicamos Countvectorizer.

from sklearn.model_selection import train_test_split
x = df["descripcion"]
y = df["clase"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=1)

from sklearn.feature_extraction.text import CountVectorizer



# Guardamos la función en variable
vectorizer = CountVectorizer()

# Codificamos el train y test de la variable x, es decir, el texto
x_train = vectorizer.fit_transform(x_train)
x_test = vectorizer.transform(x_test)

# Con el siguiente comando obtendríamos un resumen de las palabras que ha aprendido, es decir, del diccionario que la función
# se ha creado.
# print(vectorizer.vocabulary_)
```

```
# Las siguientes líneas son útiles para ver si la conversión se ha realizado bien y obtener el tamaño de los vectores
#print(vector_x_train.shape)
#print(type(vector_x_train))
#print(vector_x_train.toarray())

# Ahora que tenemos los datos preparados, importamos la función SVM.

from sklearn import svm
clf_svm = svm.LinearSVC()

# ajustamos el modelo al train y test
modelo1=clf_svm.fit(x_train, y_train)

# Intentamos hacer la predicción del test para ver cómo lo hace
pred_mod1=modelo1.predict(x_test)

# obtenemos el score o porcentaje de accuracy
accuracysvm=clf_svm.score(x_test, y_test)

#print(classification_report(y_test, pred_mod1))

# Importamos y realizamos la matriz de confusión.

from sklearn.metrics import confusion_matrix
mat1=confusion_matrix(y_test,pred_mod1)

from tabulate import tabulate
encabezado=['SVM','Deportes','Economia','Ciencia','Educación']
indice=['Deportes','Economia','Ciencia','Educación']
print(tabulate(mat1, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

# Lo podemos hacer también con el modelo de Naive Bayes, alomejor este modelo es #mejor para este tipo de clasificación.
```

```
from sklearn.naive_bayes import MultinomialNB

# Importamos la función, ajustamos el modelo y hacemos la predicción.
nb= MultinomialNB()
modelo1_1=nb.fit(x_train, y_train)
pred_modelo1_1=nb.predict(x_test)

# Obtenemos la matriz de confusión
mat2=confusion_matrix(y_test,pred_modelo1_1)

from tabulate import tabulate
encabezado=['SVM','Deportes','Economia','Ciencia','Educación']
indice=['Deportes','Economia','Ciencia','Educación']
print(tabulate(mat2, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

# Creamos una tabla resumen con los principales ratios de comparación entre
los dos #modelos, y vemos las diferencias

from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score

precision_mod1=precision_score(y_test, pred_mod1, average='weighted')
precision_mod1_1=precision_score(y_test, pred_modelo1_1,
average='weighted')

recall_mod1=recall_score(y_test, pred_mod1, average='weighted')
recall_mod1_1=recall_score(y_test, pred_modelo1_1, average='weighted')

f1_mod1=f1_score(y_test, pred_mod1, average='weighted')
f1_mod1_1=f1_score(y_test, pred_modelo1_1, average='weighted')

summary1=[[precision_mod1,recall_mod1,f1_mod1],[precision_mod1_1,recall_
mod1_1,f1_mod1_1]]
encabezado=['Ratios','Precision','Recall','F1 Score']
```

```
indice=['SVM','Naive Bayes',]
print(tabulate(summary1, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))


# Ahora vamos a eliminar las palabras que no aportan información como
pueden ser #los determinantes, artículos y otras palabras que no van a ser
relevantes para la #clasificación y que pueden ralentizar la clasificación y
incluso empeorar los #resultados.

#import nltk
#nltk.download('punkt')

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Establecemos el idioma de los stopwords, en este caso están en español
stop_words = set(stopwords.words('spanish'))

# iteramos sobre los 40 registros que tenemos, aplicando una partición de las
palabras # y frases con la función word_tikenize
# y eliminar los stopwords.
Xedited= []
for i in range(0,40):

    word_tokens = word_tokenize(df1.iat[i,1])
    filtered_sentence = []
    for w in word_tokens:
        if w not in stop_words:
            filtered_sentence.append(w)
    filtered_sentence = ' '.join(filtered_sentence)
    Xedited.append(filtered_sentence)

from sklearn.feature_extraction.text import CountVectorizer
```

```
# almacenamos la función que realiza la conversión a vectores
vectorizer1 = CountVectorizer()

# crear la libreria a partir del train.
X1 = vectorizer1.fit_transform(Xedited).toarray()

# Una alternativa para hacer la transformación a vectores es la librería
TfidfVectorizer. #En este caso se toman las frecuencias como referencia, es
decir el conteo, pero con #la peculiaridad de que hace una asignación de pesos
invertida. Es decir, si en algún #caso la mayor frecuencia se asocia con una
menor influencia como puede ser los #artículos o determinantes.

from sklearn.feature_extraction.text import TfidfVectorizer

# create the transform
vectorizer2 = TfidfVectorizer()

# tokenize and build vocabulary
X2= vectorizer2.fit_transform(Xedited).toarray()

#Ahora ya tenemos los datos preparados y podemos pasar a separar en train y
test

from sklearn.model_selection import train_test_split

## Esto eliminando las palabras como determinantes y demás y aplicando
Countvectorizer
x_train1, x_test1, y_train1, y_test1 = train_test_split(X1, y,
test_size=0.25,random_state=1)

## Esto eliminando las palabras como determinantes y demás y aplicando
TFIDVectorizer
x_train2, x_test2, y_train2, y_test2 = train_test_split(X2, y,
test_size=0.25,random_state=1)

#Pasamos a aplicar los algoritmos y realizar las predicciones.
```

```
from sklearn import svm
clf_svm = svm.LinearSVC()

#ajustamos los 2 modelos que tenemos
modelo2=clf_svm.fit(x_train1, y_train1)
modelo3=clf_svm.fit(x_train2, y_train2)

## hacemos las predicciones
pred_mod2=modelo2.predict(x_test1)
pred_mod3=modelo3.predict(x_test2)

# Matriz de confusión para el SVM con función CountVectorizer.
mat3=confusion_matrix(y_test1,pred_mod2)

encabezado=['SVM CountVec','Deportes','Economia','Ciencia','Educación']
indice=['Deportes','Economia','Ciencia','Educación']
print(tabulate(mat3, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

# Matriz de confusión para el SVM con función TfidfVectorizer.
mat4=confusion_matrix(y_test2,pred_mod3)

encabezado=['SVM TfidfVec','Deportes','Economia','Ciencia','Educación']
indice=['Deportes','Economia','Ciencia','Educación']
print(tabulate(mat4, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

#Ahora aplicamos el segundo de los modelos, el de Naive Bayes. Ajustamos y
#hacemos la predicción.

from sklearn.naive_bayes import MultinomialNB

nb= MultinomialNB()
nb.fit(x_train1, y_train1)
predichos=nb.predict(x_test1)
```

```
# Matriz de confusión para el Naive Bayes con función CountVectorizer.
mat5=confusion_matrix(y_test1,predichos)

encabezado=['Naive Bayes','Deportes','Economia','Ciencia','Educación']
indice=['Deportes','Economia','Ciencia','Educación']
print(tabulate(mat5, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

nb2= MultinomialNB()
nb2.fit(x_train2, y_train2)
predichos2=nb2.predict(x_test2)

# Matriz de confusión para el Naive con función TfidfVectorizer.
mat6=confusion_matrix(y_test2,predichos2)

encabezado=['NB TfidfVec','Deportes','Economia','Ciencia','Educación']
indice=['Deportes','Economia','Ciencia','Educación']
print(tabulate(mat6, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

# Vamos a incluir las tablas comparativas de Ratios entre los dos modelos con
los dos # funciones distintas que se utilizan.

# Primera tabla resumen con los indicadores entre los dos algoritmos y la
función #CountVectoriozer.

precision_mod1=precision_score(y_test1, pred_mod2, average='weighted')
precision_mod2=precision_score(y_test1, predichos, average='weighted')


recall_mod1=recall_score(y_test1, pred_mod2, average='weighted')
recall_mod2=recall_score(y_test1, predichos, average='weighted')


f1_mod1=f1_score(y_test1, pred_mod2, average='weighted')
f1_mod2=f1_score(y_test1, predichos, average='weighted')
```

```
summary2=[[precision_mod1,recall_mod1,f1_mod1],[precision_mod2,recall_mo
d2,f1_mod2]]
encabezado=['Ratios CountVec','Precision','Recall','F1 Score']
indice=['SVM','Naive Bayes',]
print(tabulate(summary2, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))

#Segunda tabla resumen con los indicadores entre los dos algoritmos y la
función #TfidfVectoriozer.

precision_mod1=precision_score(y_test2, pred_mod3, average='weighted')
precision_mod2=precision_score(y_test2, predichos2, average='weighted')

recall_mod1=recall_score(y_test2, pred_mod3, average='weighted')
recall_mod2=recall_score(y_test2, predichos2, average='weighted')

f1_mod1=f1_score(y_test2, pred_mod3, average='weighted')
f1_mod2=f1_score(y_test2, predichos2, average='weighted')

summary3=[[precision_mod1,recall_mod1,f1_mod1],[precision_mod2,recall_mo
d2,f1_mod2]]
encabezado=['Ratios TfdifVec','Precision','Recall','F1 Score']
indice=['SVM','Naive Bayes',]
print(tabulate(summary3, headers=encabezado,showindex=indice,
tablefmt='fancy_grid'))
```

Anexo 3

En este apartado se comentan las principales funciones que se emplean a lo largo de la resolución de los supuestos planteados.

- **Countvectorizer ():** Una de las funciones que se integran en el módulo sklearn y consiste en la transformación de un texto a vector basándose en la frecuencia de las veces que aparece en nuestro texto. El texto se convierte en una matriz, donde cada una de las palabras se representa en forma de columna en la matriz y en las filas se representan los ejemplos de documentos.

-**TfidfVectorizer():** Las siglas de esta función hacen referencia a Term Frequency and Inverse Document Frequency. Es una combinación de dos métodos, que se complementan para dar un mayor peso a las palabras que con menos frecuencia ocurren, ya que el hecho de que salgan con mayor frecuencia no debe influir en la importancia que presenta. Esto representa una mejora en algunos casos respecto al CountVectorizer, que sólo se basa en el conteo de las palabras.

- **PorterStemmer():** Función empleada para encontrar la raíz de las palabras. Esto ayuda a que palabras que son de la misma familia no sean contadas como distintas, y de esta forma la necesidad de un menor número de operaciones.

- **MultinomialNB():** Función implementada referida a nuestro modelo de Naive Bayes, adaptado para aplicarlo a múltiples categorías. Ya que, en un principio, es un modelo planteado a raíz de dos sucesos A y B.

- **SVM ():** Implementación del algoritmo Support Vector Machine, que cuenta con múltiples argumentos que se pueden modificar para adaptarse a los distintos sucesos. Entre los más importantes están el kernel o probability.

9.- BIBLIOGRAFIA

[1] Yuxi Liu - Python Machine Learning By Example_ The easiest way to get into machine learning (2017, Packt Publishing)

[2] Oliver Theobald - Machine Learning For Absolute Beginners- A Plain English Introduction (2017, Scatterplot Press)

[3] Towards Data Science,Train and Test Set: https://towardsdatascience.com/6-amateur-mistakes-ive-made-working-with-train-test-splits-916fabb421bb

[4] Medium.com, Text Analysis:

https://medium.com/@ParallelDots/everything-you-need-to-know-about-text-analysis-a730acf31647

[5] The Stanford Natural Language Processing:

https://nlp.stanford.edu/software/stanford-dependencies.shtml