Spam Killer: Clasificador de correos mediante el uso de Naive Bayes y kNN.

José Cáceres Gómez
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
joscacgom@alum.us.es

José Antonio Márquez López
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
josmarlop16@alum.us.e

***Resumen*— Para este proyecto, la idea desde el primer momento era crear un detector de spam que realmente se adaptara a los distintos correos que recibimos cada día, y que de alguna manera, no perdamos aquellos correos que no siempre son spam, y que por alguna razón, la mayor parte de servicios de correo electrónico categorizan de esa manera.**

***Herramientas*— pandas, cPickle(pickle), sklearn, nltk, string, email, csv, bs4 y re.**

***Conclusión*— La principal conclusión obtenida ha sido que ambos métodos son muy similares, e incluso, aplicando el correcto suavizado de Laplace para Naive Bayes, este último es ligeramente mejor para un caso general en comparación con kNN; pero teniendo en cuenta otros datos obtenidos y demás métricas, kNN es mejor con respecto a este caso.**

Palabras Clave— Correo-e, Spam, Ham, Lematización, Radicalización, StopWords, NLP, Tokenización, Naive Bayes, KNN, Machine Learning.

I. Introducción

En la comunicación por correo electrónico (correo-e, de aquí en adelante) se emplea el término *spam* para designar a aquellos mensajes enviados de forma masiva y que no han sido solicitados por sus destinatarios. Por su parte, se usa el término *ham* para los correos que sí que son esperados o solicitados por sus destinatarios. Los gestores de correo-e suelen incorporar la posibilidad de crear filtros de spam que permitan identificar estos de forma automática y actuar en consecuencia (usualmente moviéndolos de la bandeja de entrada a una carpeta específica). El aumento del volumen de correos electrónicos no deseados recibidos por los usuarios ha creado la necesidad de desarrollar filtros más fiables y robustos, para lo que se aplica toda una serie de métodos, entre los que tienen una gran prevalencia aquellos provenientes del campo del aprendizaje automático.

El **objetivo principal** de esta propuesta es construir, mediante técnicas de procesamiento del lenguaje natural, concretamente mediante Naive Bayes Multinomial, y kNN como modelos clasificadores, Bolsa de términos y Tf-Idf, como modelos de lenguaje respectivamente, un sistema de filtrado de correo-e que aprenda de los correos recibidos y desechados, para tener asi una efectividad mayor que los actuales servicios de filtrado, y disminuir el número de correos fraudulentos que recibimos.

Tras una investigación sobre cómo funcionan los filtros de servicios de correo-e como Gmail o Outlook, y cómo gestionaban esas incidencias, en general, al ser la mayoria servicios con servidores enormes y gestores en cada servidor, tenían unas redes de filtrado extremadamente densas como para que nos sirviese de algo.

De todo ello sacamos algo en claro, y es que para filtrar los correos dados, era necesario obtener de cada correo las palabras que lo conforman, y a partir de ahí ya podremos filtrar los correos según una bolsa de términos, como modelo de lenguaje, que se crease a partir de los términos de cada correo-e leído.

Gracias a esto, también razonamos cómo podríamos obtener los términos y palabras de los correos recibidos, por el simple hecho de que en muchos casos, sobre todo en los casos de spam, la mayor parte de los términos eran etiquetas *html*, y se repetirían numerosas veces. A partir de aquí surge una problemática, tendremos una cantidad enorme de términos que debe de ser reducida mediante el uso de técnicas para el preprocesamiento de datos.

Por todo ello, para la obtención de la bolsa de términos, se realiza un proceso de tokenización en el que tendremos en cuenta términos alfanuméricos, descartando signos de puntuación y aquellas palabras pertenecientes al conjunto stopwords[10]. Por último, de cada uno de los términos resultantes, se obtendrán los lemas y las raíces, con el fin de obtener una mejor bolsa de términos.[1]

Todos estos procesos nos permitirán crear un clasificador haciendo uso de Naive Bayes Multinomial[2] que aprende cada vez más términos y, que al cabo del tiempo, harán al filtro lo más inteligente posible, aumentando así su precisión y disminuyendo el número de fallos que comete.

Con la misión de obtener el clasificador más óptimo posible y como se ha mencionado en el presente documento, se construirá un clasificador kNN mediante el uso como modelo de lenguaje Tf-Idf, esta medida hace referencia a la frecuencia de aparición de cada término en una colección de

documentos, el valor Tf-Idf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia del término en la colección de documentos, lo que nos permite observar el hecho de que algunos términos son más comunes que otros.

Para construir y entrenar dichos clasificadores se dispone de un fichero con dos conjuntos de correos clasificados como spam y legítimos.

El presente documento se dividirá en varios apartados:

1. *Preliminares:* donde se introducirán las técnicas empleadas en la construcción de dichos modelos de clasificación.
2. *Metodología:* dedicado a la descripción del método implementado en el proyecto, correspondiente con el desarrollo del mismo.
3. *Resultados:* donde se detalla y analiza tanto los experimentos realizados como los resultados obtenidos en ambos modelos de clasificación.

II. Preliminares

En esta sección se hace una breve introducción de las técnicas empleadas y también trabajos relacionados.

A. Métodos empleados

Se han empleado, tal y como se introduce en el documento, Bolsa de términos como modelo de lenguaje para un modelo clasificador basado en Naive Bayes Multinomial y Tf-Idf para un kNN clasificador:

- **Bolsa de términos**[1]: Técnica en la que cada documento se representa como el vector de la cantidad de veces que ocurre cada término en el documento. Este vector no conserva el orden de términos en la oración, siendo una de las características principales de este modelo. Este tipo de representación es aplicable al filtrado de correos electrónicos.

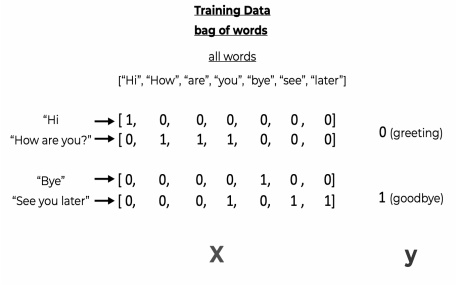

Figura 1: Ejemplo de una bolsa de palabras

- **Naive Bayes Multinomial**[2]: El algoritmo Naive Bayes multinomial es un método de aprendizaje probabilístico utilizado principalmente en procesamiento del lenguaje natural. Está basado en el teorema de Bayes y predice la etiqueta de un texto, como en este caso, un correo electrónico. Calcula la probabilidad de cada etiqueta para una muestra dada y devuelve como resultado la etiqueta con probabilidad más alta. El teorema de Bayes[6] calcula la probabilidad de que ocurra un evento en base al conocimiento previo de las condiciones relacionadas con dicho evento. Naive Bayes supone la independencia de las variables predictoras.

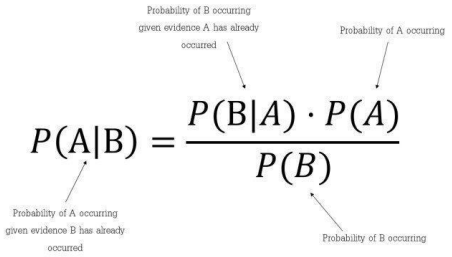

Figura 2: Teorema de Bayes

- **Lematización**[3]: Es un proceso lingüístico que consiste en, dada una forma flexionada (es decir, en plural, en femenino, conjugada, etc), hallar el lema correspondiente. El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra. Es decir, el lema de una palabra es la palabra que nos encontraríamos como entrada en un diccionario tradicional: singular para sustantivos, masculino singular para adjetivos, infinitivo para verbos, etc. Por ejemplo, *decir* es el lema de *dije*, pero también de *diré* o *dijéramos*; *guapo* es el lema de *guapas*; *mesa* es el lema de *mesas*.

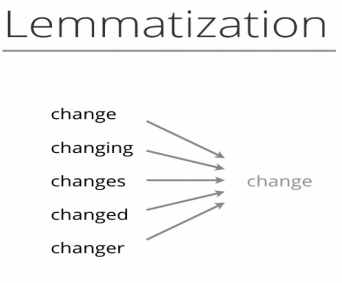

Figura 3: Ejemplo de lematización

- **Radicalización**[3]: Es el procedimiento de convertir palabras en raíces. Estas raíces son la parte invariable de palabras relacionadas sobre todo por su forma. Nos asegura que la morfología de las palabras no penalice la frecuencia de estas, nos permite realizar la extracción del morfema, lo que implica identificar e ignorar prefijos y sufijos con el fin de obtener la raíz común de la palabra.

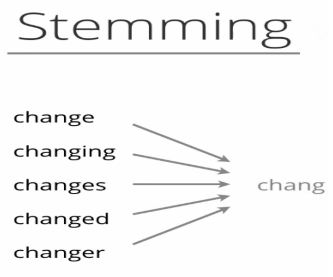

Figura 4: Ejemplo de radicalización

- **Tf-Idf**[4]: Es una medida que puede cuantificar la relevancia de las representaciones de cadenas de texto en un documento entre una colección de documentos. El valor tf-idf aumenta proporcionalmente al número de veces que aparece una palabra en un documento, pero es compensada por la frecuencia de la palabra en

la colección de documentos, lo que permite observar el hecho de que algunas palabras son más comunes que otras. Tf-Idf se puede dividir en dos partes:

- *Frecuencia de términos(TF):* Funciona observando la frecuencia de un término en concreto en relación con el documento, existen múltiples maneras de definir la frecuencia, por ejemplo, el número de veces que aparece el término en el documento.

- *Frecuencia de documento inversa (IDF):* Analiza qué tan común es un término en la colección de documentos. Se obtiene dividiendo el número total de documentos, por el número de documentos que contiene al término.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Figura 5: Fórmula de tf-idf

- **kNN**[5]: Es un algoritmo de aprendizaje supervisado que clasifica cada dato nuevo en el grupo que corresponda, buscando los puntos de datos más similares aprendidos en la etapa de entrenamiento. Para ello calcula la distancia del elemento nuevo a cada uno de los existentes para seleccionar al grupo al que pertenece, dicho grupo será el de menores distancias. Hay que tener en cuenta dos aspectos importantes:

  - *k:* Con valores k distintos, podemos obtener resultados muy distintos.

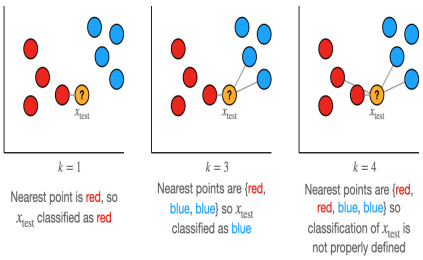

Figura 6: Ejemplos de valores que toma k

  - *Métrica de distancia:* La métrica utilizada en el modelo influye de gran manera en las relaciones de cercanía que se irán estableciendo en el algoritmo con los *k* vecinos.

$$Manhattan\ Distance = \sum_{i=1}^{d} |x_{1i} - x_{2i}|$$

$$Euclidean\ Distance = \left(\sum_{i=1}^{d} (x_{1i} - x_{2i})^2\right)^{\frac{1}{2}}$$

$$Minkowski\ Distance = \left(\sum_{i=1}^{d} |x_{1i} - x_{2i}|^p\right)^{\frac{1}{p}}$$

Figura 7: Métricas de cercanía

B. Trabajo Relacionado

Como material de apoyo a la hora de realizar el proyecto, hemos tenido en cuenta todos los enlaces y bibliografías que venían en el documento de explicación del proyecto, especialmente el artículo *Applicability of machine learning in spam and phishing email filtering: review and approaches*[12], además de distintos vídeos de usuarios de Youtube, donde explican de sencilla cómo emprender la tarea de realizar un modelo clasificador de spam usando Naive Bayes y kNN. Por su parte, también buscamos información en internet de manera general sobre los distintos filtros y aplicaciones de estos sobre correos y demás documentos. En sí, tuvimos muy en cuenta la información brindada por la propia asignatura, pero al tener que aplicar un método concreto en un proyecto más específico, tuvimos que buscar más información en internet.

Por descontado queda que toda esta información está reflejada en la bibliografía de este documento, además de referenciadas en las diferentes definiciones ya brindadas.

También hemos utilizado distintas librerías recomendadas por el documento de explicación del proyecto, tales como *scikit-learn* o *nltk.*

III. Metodología

Aquí explicaremos de manera más detallada, cuál ha sido la metodología, la idea principal de los filtros y del procesado de los correos, y junto con todo ello, algunas métricas principales de ambos métodos junto con un breve análisis de ellas.

A. Preprocesado y procesado de correos.

En primera instancia, dadas dos carpetas con los correos legítimos y spam separados, pasar estas carpetas a un intervalo de entrenamiento y uno de test, con el objetivo de ver si los resultados obtenidos eran los correctos, y si todo funcionaba como se esperaba y habíamos entrenado bien el filtro. Comenzamos creando *email_csv_parser.py,* donde se hallan los métodos de preprocesamiento y la posterior creación de los archivos .csv sobre los que crear el *dataframe* para trabajar con los datos de manera más sencilla.

Se procede con la creación de varios métodos en los que limpiamos esos correos, quitando posibles etiquetas html que podrían ser conflictivas, y eliminamos también espacios excesivos, saltos de línea y demás elementos no deseados o que podrían dificultar el correcto funcionamiento del modelo clasificador. Al final de todo ello, almacenamos los correos en dos archivos distintos, para poder trabajar más fácilmente e identificar posibles fallos.

Principalmente, lo primero que hacemos con los correos es decodificarlos en *latin-1*, ya que nos permite decodificar

todos los correos sin problemas de *charset,* posteriormente, procederemos a lo más importante: limpiar el correo.

Este proceso es un poco más sencillo, ya que aplicando un patrón, simplemente sustituimos caracteres como puntos, comas, exclamaciones o corchetes, con espacios en blanco. También haremos que todo el correo sea pasado a texto plano.

Una vez ya tenemos el texto plano, lo pasamos todo a minúsculas y eliminamos los saltos de línea. Con todo ello, sustituimos tanto cifras, como correos electrónicos y enlaces, por términos normalizados como *"number", "httpaddress", "emailaddress"*. Todo este proceso, nos ha servido para pasar de correos con cabeceras complejas y demás etiquetas, a correos mucho más simples con términos concretos, lo cual nos facilitara el trabajo a la hora de hacer la bolsa de términos.

Todo ello era lo que consideramos como el preprocesamiento de los correos, que es una parte fundamental, ya que esto es un factor determinante a la hora de aplicar el filtro y obtener diferentes términos.

En términos de complejidad, el filtro en sí no es muy complejo, aplicando el propio Naive Bayes y teniendo una buena bolsa de términos con la que testear, y sobretodo, entrenar el filtro; por todo esto, teníamos muy claro desde el principio la importancia de preprocesar y trabajar con los correos.

Al revisar los correos legítimos, no encontramos muchas complicaciones en primera instancia: sabíamos que teníamos que eliminar las etiquetas y demás información del correo, es decir, todo aquello que no fuese el cuerpo del mensaje. Obviamente tuvimos en cuenta que, más adelante, durante el desarrollo del algoritmo, sería útil conocer ciertos parámetros de estas cabeceras, pero en un primer momento, para crear esta bolsa de términos, no era necesario.

Cuando comenzamos a procesar y manipular los correos de spam, nos dimos cuenta de que lo complicado era separar y procesar todos esos, ya que la mayoría eran prácticamente muchísimas etiquetas html, enlaces y términos con codificación extraña o poco común. Al darnos cuenta de esto, decidimos que era extremadamente importante procesar correctamente los correos, por ello, buscamos información sobre el procesado de etiquetas html para solo obtener los propios términos o las etiquetas más específicas, además de las distintas codificaciones que haya dentro de los correos..

Este proceso no fue desde luego sencillo, ya que con la cantidad de correos y de distintas codificaciones que poseían estos, fue bastante complicado procesar correctamente sin eliminar información demás, y tampoco sobrecargando las bolsas de términos de palabras repetidas que fuesen etiquetas o caracteres extraños.

Técnicamente, es decir, refiriéndonos más al código de manera más explícita, una vez teníamos procesados los correos, los almacenamos en dos archivos .csv, uno para correos legítimos y otro para correos spam, en el que almacenamos en una columna el propio cuerpo del correo ya procesado, y en la otra columna, un valor de 0 o 1 reflejando si ese correo era legítimo o spam respectivamente.

B. Implementación de Naive Bayes Multinomial.

Una vez teníamos los dos archivos de los correos, para poder entrenar correctamente el filtro, los unimos en un único *dataframe*, para poder trabajar con todos los correos de manera conjunta, crear una bolsa de términos más completa, y además, ahorrar bastante tiempo.

Posteriormente a esto, se crea la bolsa de palabras o bolsa de términos, donde eliminamos los signos de puntuación y cualquier término que pertenezca al conjunto de stopwords o bien no sea alfanumérico, para quedar aún más procesado el correo. Pero esta tarea, que ya hemos dejado claro lo fundamental que es, no acaba aquí, pues de nuevo, los términos ya obtenidos en esa bolsa, son procesados de nuevo, lematizandolos, es decir, obteniendo el lema o palabra común.

Al principio de la implementación, sí que se radicalizaron los términos, pero al evaluar los primeros resultados, nos dimos cuenta de que apenas cambiaban los resultados, y de hecho, obteníamos una bolsa de términos más consistente si sólo aplicamos la lematización. Posteriormente, durante la mejora del filtro, decidimos radicalizar en primer lugar, con el *algoritmo de porter*[12], y posteriormente lematizamos estos términos, todo ello aumentaba la precisión del filtro.

Gracias a todo este proceso, hemos pasado de unos correos en unas carpetas, totalmente sin procesar y con una cantidad de términos triviales innecesarios, a correos limpios, solo con los términos más importantes y sus lemas, y listos para manejarlos con los filtros.

Una vez tenemos la ansiada bolsa de términos, es necesario vectorizarla, es decir, hacer una matriz con los términos y el número de veces que aparecen en todo el documento, en este caso, en la bolsa de palabras.

Con todo esto, simplemente había que inicializar distintas variables y aplicar el propio Naives Bayes.

Por supuesto, este método es generado por el propio paquete *sklearn*, que posee además un gran conjunto de métricas, que considerábamos útiles, tales como la precisión del filtro, o la matriz de confusión. Gracias a estas métricas, podíamos evaluar lo preciso que era el método, dependiendo del suavizado de laplace que se aplicase. También nos fueron útiles a la hora de evaluar cuál de los dos métodos era el correcto, ya que, principalmente, las dos métricas dichas eran las más importantes a la hora de comparar ambos métodos del proyecto.

Con todo ello, tenemos unos resultados generales del método que nos arrojaban los siguientes datos:

Tabla 1: Detalles de precisión por cada suavizado

Suavizado	Precisión	Precisión (SPAM)	Precisión (HAM)
n=1	0.99525597944	1.00	0.89
n=5	0.99525597944	0.99	0.90
n=10	0.98970423062	0.99	0.93

Y además, nos arrojan la siguiente matriz de confusión:

Tabla 2: Matriz de confusión con Laplace a 1

n=1	**Valores Reales**		
		HAM	*SPAM*
Valores Predichos	*HAM*	4639	24
	SPAM	592	5035

Tabla 3: Matriz de confusión con Laplace a 5

n=5	**Valores Reales**		
		HAM	*SPAM*
Valores Predichos	*HAM*	4632	41
	SPAM	525	5102

Tabla 4: Matriz de confusión con Laplace a 10

n=10	**Valores Reales**		
		HAM	*SPAM*
Valores Predichos	*HAM*	4608	55
	SPAM	340	5287

Como se puede ver, cada vez que se aplica un suavizado mayor, mejoran las métricas y el número de correos filtrados correctamente. También podemos ver que los resultados no son malos, pero considerábamos que eran muy mejorables.

C. Implementación del modelo kNN.

El funcionamiento del método, en un inicio es igual que el de Naive Bayes, es decir, a partir de los dos archivos *.csv* de los correos separados, los juntamos en un solo set de datos, y a partir de ahí, lo hacemos de manera similar: aplicamos el vector, pero en este caso, el vector de *tf-idf*. Por supuesto, aplicamos el mismo proceso de lematización y radicalización en los términos de la bolsa, para obtener métricas más fiables y evaluar ambos métodos bajo igualdad de condiciones. Con todo ello, obtenemos las mismas métricas, es decir, precisión, sensibilidad o tasa de verdaderos positivos o la matriz de confusión entre otros. Todas estas métricas son evaluadas con 10, 15 y 20 de *k*.

Estas métricas, nos sirven para medir cómo de bueno es el filtro que se está aplicando, además de ver si es realmente diferencial el uso de uno u otro valor de k. Por ejemplo, nos arroja los siguientes resultados generales:

Tabla 5: Detalles de precisión por cada valor de k

Valor de *k*	**Precisión**	**Precisión (SPAM)**	**Precisión (HAM)**
k=10	0.9041518001	0.90	0.99
k=15	0.9576655052	0.96	0.97
k=20	0.9713203463	0.97	0.95

Y además, nos arrojan la siguiente matriz de confusión:

Tabla 6: Matriz de confusión con k = 10

k=10	**Valores Reales**		
		HAM	*SPAM*
Valores Predichos	*HAM*	4072	591
	SPAM	52	5575

Tabla 7: Matriz de confusión con k = 15

k=15	**Valores Reales**		
		HAM	*SPAM*
Valores Predichos	*HAM*	4420	243
	SPAM	130	5497

Tabla 8: Matriz de confusión con k = 20

k=20	**Valores Reales**		
		HAM	*SPAM*
Valores Predichos	*HAM*	4504	159
	SPAM	242	5385

Como podemos ver en la *tabla 5*, hay una clara estabilidad en la precisión de los correos que son spam, sin embargo, la precisión fluctúa un poco en las dos últimas iteraciones del valor de k, al igual que podemos ver en las distintas matrices de confusión previas, que por cada valor de k, aumentan los casos en lo que se acierta y disminuyen los fallos, pero aun así, se puede ver a simple vista, que los resultados generales son claramente mejores, sobretodo en el número de fallos.

D. Serialización y deserialización.

Con respecto a esto, hemos considerado muy importante el uso de la librería pickle[13], puesto que era fundamental para el desarrollo del archivo *clasificador_seleccionado.py*. Esto se debe a que, era necesario llamar al modelo clasificador seleccionado, pero el uso de un simple *"import"* de este modelo hace que esta clase no se ejecute en un tiempo aceptable, ya que volvería a realizarse el proceso de entrenamiento del propio modelo. Todo esto nos obligó a usar pickle, que empaqueta el modelo previamente entrenado(*serialización*), y permite posteriormente desempaquetarlo(*deserialización*) y usarlo sin necesidad de volver a entrenar el modelo clasificador, lo que nos lleva a una ejecución de la predicción en un tiempo muy reducido.

IV. Resultados

Primeramente, explicaremos de una manera muy resumida, los métodos utilizados, junto con una explicación muy breve de sus beneficios.

- **Naive Bayes Multinomial:** es un método de aprendizaje probabilístico utilizado principalmente

en procesamiento del lenguaje natural. Está basado en el teorema de Bayes y predice la etiqueta de un texto, como en este caso, un correo electrónico. Calcula la probabilidad de cada etiqueta para una muestra dada y devuelve como resultado la etiqueta con probabilidad más alta. Se usará el concepto de **bolsa de términos** como modelo de lenguaje, que es una técnica en la que cada documento se representa como el vector de la cantidad de veces que ocurre cada término en el documento. Este vector no conserva el orden de términos en la oración, siendo una de las características principales de este modelo. Este tipo de representación es aplicable al filtrado de correos electrónicos. Teniendo esto en cuenta, se pueden elegir distintos valores de k para el ***suavizado de Laplace***, lo cual nos aporta siempre mejores resultados con respecto al mismo conjunto de datos. Concretamente, elegimos como valores iniciales 1, 2 y 3, pero durante el desarrollo del filtro, decidimos aumentar los valores y la distancia entre ellos, para poder ilustrar mejor la evolución del suavizado, y por ello elegimos 1, 5 y 10, para poder mostrar una diferencia clara. Podríamos aumentar estos valores, pero la diferencia no era sustancial, y además quedaban algo desvirtuados los resultados.

- **kNN**: Es un algoritmo de aprendizaje supervisado que clasifica cada dato nuevo en el grupo que corresponda, buscando los puntos de datos más similares aprendidos en la etapa de entrenamiento. Para ello calcula la distancia del elemento nuevo a cada uno de los existentes para seleccionar al grupo al que pertenece, dicho grupo será el de menores distancias. Por su parte, usa **tf-idf** como modelo de lenguaje, que es una medida que puede cuantificar la relevancia de las representaciones de cadenas de texto en un documento entre una colección de documentos. Al igual que en el método de Naive Bayes, aquí tenemos distintas iteraciones, utilizando los valores de ***k vecinos*** en este caso. La decisión de elegir 5, 10 y 15 es la misma, es decir, elegimos estos valores con el fin de mostrar mejor la evolución del filtro y su aprendizaje.

Con todo lo visto, procederemos a dar una vista más detallada de todas las métricas de cada método, así como una comparación visual de ellas, para poder así, dar una explicación más clara y objetiva.

Tabla 9: Comparativa entre Naive Bayes y kNN

Método	Precision	Accuracy	Recall
NB (n=1)	0.99525597944	0.94013605442	0.89479296250
NB (n=5)	0.99525597944	0.94499514091	0.90669984005
NB (n=10)	0.98970423062	0.96161321671	0.93957703927
kNN (k=10)	0.9041518001	0.93751214771	0.99075884130
kNN (k=15)	0.9576655052	0.96375121477	0.97689710325
kNN (k=20)	0.9713203463	0.96103012633	0.95699306913

Principalmente, se han seleccionado las métricas de *precision, accuracy* y *recall,* que se refiere a:

1. *Precision*: es la proporción de ejemplos realmente positivos entre los clasificados como positivos
2. *Accuracy*: es la proporción de ejemplos clasificados correctamente. También denominado tasa de acierto.
3. *Recall*: es la proporción de ejemplos positivos clasificados correctamente. También conocidos como tasa de verdaderos positivos o recuerdos.
4. *F1-score:* es la media armónica entre *precision* y *recall.*
5. *Classification report:* que no es más que un breve resumen de las métricas previas, en una tabla. Es una métrica que se muestra en el código, pero que no hemos creído relevante poner en la documentación, pues no es más que un resumen de lo previo.

Para nosotros, las métricas más significativas que arrojan ambos métodos son *precision* y las matrices de confusión, de las cuales vamos a hablar a continuación.

Como se puede ver, no hay una gran diferencia entre ambos métodos a nivel de métricas, sin embargo, si comparamos las matrices de confusión de los dos mejores casos de ambos métodos, tenemos:

Tabla 10: Matriz de confusión con n = 10 para Naive Bayes

n=10	**Valores Reales**		
Valores Predichos		*HAM*	*SPAM*
	HAM	4608	55
	SPAM	340	5287

Tabla 11: Matriz de confusión con k = 15 para kNN

k=15	**Valores Reales**		
Valores Predichos		*HAM*	*SPAM*
	HAM	4420	243
	SPAM	130	5497

Teniendo en cuenta esta comparativa, explicaremos la decisión sobre el método elegido.

Como podemos ver, hay un cambio muy grande entre los correos que no son SPAM y que se clasifican como SPAM, es decir, de todos los correos probados, solo 130 son clasificados como SPAM, siendo estos correos legítimos. Comparándolo con el método de Naive Bayes, hay 210 correos mal clasificados en ese sentido, y por ello, tomamos la decisión valorando todos estos resultados que el mejor clasificador es el de kNN, pues lo importante de estos métodos clasificadores es que los correos de spam sean correctamente clasificados en su gran mayoría, y que los casos legítimos, no sean clasificados erróneamente, y en ambos casos, el clasificador de kNN es mejor, en concreto, kNN con valor 15 para k vecinos.

En definitiva y como resumen final: ambos métodos son muy similares tanto en métricas como en matrices de confusión, sin embargo, el método de kNN es mejor que el de Naive Bayes para el caso concreto que nos requiere el proyecto, que los correos de spam sean correctamente clasificados en su gran mayoría, y que los casos legítimos, no sean clasificados erróneamente como spam.

V. Conclusiones

Finalmente, creemos que hemos elegido correctamente el filtro, pues si es verdad que para un caso más general, en el que no haya un objetivo más claro que la simple comparación de métricas de ambos métodos, Naive Bayes arroja mejores resultados, sin embargo, para el caso particular que nos atiende, es mejor el ***método de kNN*** por lo explicado anteriormente, y en concreto, con un valor de vecino ***k en 15,*** ya que de los valores elegidos, es el que mejor resultados muestra.

Hemos aprendido de manera mucho más práctica ambos métodos, como funcionan y compararlos, además de trabajar con ellos en python, lo cual nos ha servido para comprender mejor cómo funciona un método de aprendizaje automático.

Básicamente, cuando nosotros elegimos este proyecto, nuestra intención era hacer un filtro de spam realmente fiable, y además, que aprendiese de los correos que vamos recibiendo y los que se van filtrando y clasificando como no deseado.

Por ello, investigando las distintas formas que había de plantearlo, y buscando información sobre métricas, otros métodos similares e incluso diferentes artículos de análisis sobre Naive Bayes y kNN como clasificadores de spam, nos dimos cuenta de lo complejo que esto podía ser, y más aún en un lenguaje que no controlamos del todo.

Finalmente, decidimos ponernos con todo desde el principio, y apoyándonos tanto en lo aprendido en la asignatura, como en información externa, creemos que hemos realizado el mejor trabajo posible según lo especificado y teniendo en cuenta tanto las limitaciones planteadas por el proyecto, como las limitaciones de tiempo propias.

En general, estamos muy satisfechos con el trabajo realizado, tanto a nivel de código, del método seleccionado, de documentación e investigación, como a nivel práctico de lo que es el clasificador en sí.

Al final del presente documento se detallan las diferentes referencias, fuentes y artículos consultados durante el desarrollo e investigación del proyecto.

Referencias

[1] Global, Z. (s. f.). Brutalk - Una suave introducción al modelo de bolsa de palabras.Brutalk. https://www.brutalk.com/es/noticias/brutalk-blog/ver/una-suave-introduccion-al-modelo-de-bolsa-de-palabras-6047169dd22e9

[2] S. (2022, 11 enero). Multinomial Naive Bayes Explained: Function, Advantages & Disadvantages, Applications in 2022. upGrad Blog. https://www.upgrad.com/blog/multinomial-naive-bayes-explained/

[3] Fernández, L. A. U. (2021, 9 diciembre). *Reducir el número de palabras de un texto: lematización y radicalización (stemming) con Python*. Medium. https://medium.com/qu4nt/reducir-el-n%C3%BAmero-de-palabras-de-un-texto-lematizaci%C3%B3n-y-radicalizaci%C3%B3n-stemming-con-python-965bfd0c69fa

[4] *Understanding TF-IDF for Machine Learning*. (2021, 6 octubre). Capital One. https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

[5] El algoritmo K-NN y su importancia en el modelado de datos | Blog. (2020, 1 septiembre). Merkle. https://www.merkleinc.com/es/es/blog/algoritmo-knn-modelado-datos

[6] Colaboradores de Wikipedia. (2022, 23 mayo). Teorema de Bayes. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Teorema_de_Bayes

[7] Anello, E. (2021, 19 agosto). A friendly guide to NLP: Bag-of-Words with Python example. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/08/a-friendly-guide-to-nlp-bag-of-words-with-python-example/

[8] Bd, N. (2018, 2 abril). Detect E-mail Spam Using Python. CodeProject. https://www.codeproject.com/Articles/1232758/Detect-E-mail-Spam-Using-Python

[9] Data, S. B. (2019, 27 octubre). Machine Learning: Seleccion Métricas de clasificación. sitiobigdata.com. https://sitiobigdata.com/2019/01/19/machine-learning-metrica-clasificacion-parte-3/

[10] GeeksforGeeks. (2022, 18 mayo). Removing stop words with NLTK in Python. https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

[11] Applicability of machine learning in spam and phishing email filtering: review and approaches. https://idp.debiblio.com/adas/fed/federation/production/US_SAML/PAPI/gpoa.php?ACTION=CHECK&DATA=1e57038e1dfe7618e3a86805e4231e85fb383227&URL=https%3A%2F%2Flink--springer--com.us.debiblio.com%2Farticle%2F10.1007%2Fs10462-020-09814-9

[12] López, R. (2018, 26 mayo). Algoritmo de Porter para el Español en Java* - Roque López. Medium. https://medium.com/@roquelopez/algoritmo-de-porter-para-el-espa%C3%B1ol-en-java-dd44ea7b0a10

[13] pickle — Python object serialization — Python 3.10.5 documentation. (s. f.). Pickle - Serialization. https://docs.python.org/3/library/pickle.html

[14] V. (s. f.). Victor MARTIN ~ How to make pickle faster? How to Make Pickle Faster. https://vmartin.fr/how-to-make-pickle-faster.html