Creacion y análisis de filtros de correo electrónico

Pablo Álvarez Caro
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
pabalvcar@alum.us.es

Diego Ruiz Gil
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
dieruigil@alum.us.es, diegoruiz037@gmail.com

***Resumen*—Este artículo describe la creación y el análisis de un filtro para designar correos electrónicos como spam. Para ambas tareas se han utilizado dos modelos de procesamiento del lenguaje natural: Bag of Words y TF-IDF.**

El estudio llegó a la conclusión de que el filtro entrenado mediante Naive Bayes produjo mejores resultados en la mayoría de situaciones. Aunque en algunos casos kNN produjese resultados similares o incluso mejores, Naive Bayes es más consistente.

***Palabras clave*—Inteligencia Artificial, Correo Electrónico.**

I. Introducción

Con el amplio uso de los correos electrónicos desde hace decádas, damos por hecho que nuestro sistemas de mensajería, como podrían ser gmail, filtren todos esos correos que consideramos spam. ¿Pero somos realmente conscientes de todas las técnicas empleadas para ello?

Primero de todo, debemos preguntarnos, ¿qué es el spam? Para ello nos remontamos a 1970, año en el que los *Monty Python's*, en uno de sus episodios de la serie *Flying Circus*, hicieron un sketch burlándose de la carne en lata de la marca SPAM, a modo de burla debido la excesiva publicidad de dicho producto [1]. Esto acabó derivando en el uso de la palabra para referirse a mensajes *basura* o no deseados.

En cuanto a la técnicas empleadas, el *Machine Learning* toma un papel fundamental, ya que el propio algoritmo es capaz de aprender y clasificar nuevos tipos de correo *spam* que aparezcan. Para más información sobre el origen y técnicas empleadas para los filtros puede consultar el artículo de Springer Link [2]. A parte, también es esencial el uso de algoritmos del *procesamiento del lenguaje natural*, los cuales nos servirán para obtener un vocabulario a partir de los correos de entreamiento, y así poder claisificar los correos entrantes [3].

En este artículo se mostrará un análisis detallado sobre los disitintos tipos de filtros que se han construido y los motivos por los que cada uno se han considerado o desestimado.

El objetivo que se buscaba era un filtro con la mayor eficacia posible y que mantuviese cierta consistencia (como se verá en el documento posteriormente, este último ha sido un factor determinante).

En este documento, primero se verán las técnicas empleadas, posteriormente se hablará de la metodología utilizada en el trabajo, acto seguido se discutirán los resultados obtenidos y finalmente se tratará la concusión a la que se ha llegado mediante los resultados obtenidos.

II. Preliminares

Para la creación de los filtros se han hecho uso de algoritmos de aprendizaje atumático, concretamente *Naive-Bayes multinomial* y *kNN*, permitiéndonos clasificar un correo de entrada como spam o legítimo. Dado que son algoritmos de aprendizaje automático, es necesario contar con un conjunto de *entrenamiento* y otro conjunto de *validación* o *prueba*. Para ello hemos decido que un 80 porciento de todos los correos que fueron prorcionados pasarán a ser parte del conjuntno de *entrenamiento*, y el 20 porciento restante al conjunto de *validación*. En ambos conjuntos se encuentra la misma proporción de correos legítmos como no deseados.
Sin embargo, necesitaremos obtener un conjunto de terminos con los cuales poder analizar los correos entrantes, y clasificarlos posteriormente. Para ello serán necesario aplicar algoritmos del *procesamiento del lenguaje natural*. Para nuestro caso emplearemos el modelo de *Bag of Words*, con su posterior clasificación con *Naive-Bayes multinomial*, y el model de *TF-IDF*, con su posterior clasificación con *kNN*.

A continuación hablaremos de los distintos métodos y técnicas que se han empleado.

1) *Técnicas de preprocesamiento:*
   Mediante la aplicación de distintas técnicas de preprocesamiento de datos de texto, entre las que destacan la *eliminación de ruido*,la *tokenización* y la *normalización*, hemos limpiado los correos del corpus y realizado un vocabulario mediante la normalización de sus términos [4]. Todas estas técnicas nos ayudarán a reducir el vocabulario, mejorando así la eficiencia de los filtros, pero sin perder información relevante.
   - *Eliminación de ruido:* Se puede realizar antes o después de la tokenización, y consiste en limpiar el texto de aquellos elementos que no nos sean útiles a la hora de analizarlo, por lo que podríamos decir que es una actividad específica de la tarea a realizar. Esta actividad puede incluir la eliminacion de encabezados de archivos de texto, pies de página, ..., la eliminación de HTML, XML, es decir lenguaje de marcado y metadatos, o la extracción de cualquier otro tipo de datos que se considere poco útil. Para

nuestro caso, nos vale con la eliminación de HTML y reemplezar las contracciones, motivo por el cual hemos decidio hacerlo antes de la tokenización.
- *Tokenización:* Antes o tras realizar la eliminación de ruido, se procede con la tokenización. Consiste en la división del texto en palabras, que posteriormente formarán parte del vocabulario de nuestros modelos del procesamiento del lenguaje natural.
- *Normalización:* Una vez realizadas las dos actividades anteriores, se finaliza con la normalización del vocabulario obtenido tras la tokenización. Esta tarea consiste en poner en 'igualdad de condiciones' todo el texto, es decir dejar todo el texto en minúsculas, convertir los números a su equivalente en string, eliminación de puntuación, etc.

2) *Bag of Words:*
Este modelo, dado un vocabulario de términos, nos permite representar los documentos del corpus como un vector en el que se representa la cantidad de veces que aparece cada término del vocabulario en dicho documento. Esto es muy útil para la posterior clasificación de documentos mediante Naive-Bayes multinomial, modelo de clasificación del que hablamos a continuación.

3) *Naive-Bayes multinomial:*
Este modelo de clasificación nos permite asignar una etiqueta a cada documento a clasificar, que viene dada por la siguiente fórmula:

$$\hat{c} = argmáx_{c \in C}(\log(\mathbf{P}(c)) + \sum_{t \in v}(n_{D,t} * \log(\mathbf{P}(t|c)))) \tag{1}$$

donde $\hat{c}$ es la clase elegida, *c* una clase perteneciente al conjunto de todas las clases *C*, *D* el documento, *v* el vocabulario de términos (y por tanto t un término del vocabulario), $\mathbf{P}(t \in c)$ la probabilidad de que se dé el término *t* en un documento de clase *c*, y $n_{D,t}$ el número de ocurrencias del término *t* en el documento *D*.
Para evitar el caso de que alguna estimación pueda dar cero, se puede aplicar la téncia del suavidado de Laplace, por lo que la probabilidad de un termino dada una clase será:

$$\mathbf{P}(t|c) = \frac{N_{c,t} + k}{\sum_{s \in V} N_{c,s} + |V| * k} \tag{2}$$

donde $N_{c,t}$ es el número de ocurrencias del termino *t* en los documentos etiquetados como *c*, *k* es el hiperparámetro de suavizado, es decir, cuantos terminos de un tipo adicionalaes se añaden, *s* serían los terminos pertenecientes al vocabulario *V*, $N_{c,s}$ es el número de ocurrencias del termino *s* en los documentos etiquetados como *c* [5].

Un ejemplo de los datos obtenidos al aplicar *Bag of Words* junto con el suavizado de Laplace igual a 1, podría ser el siguiente:

TABLA I
CLASE *correo legítimo*

	D1	D2	D3	Suavizado	$\mathbf{P}(t\|legitimo)$
charset	3	1	1	1	$\frac{1}{2}$
texthtml	0	1	0	1	$\frac{1}{6}$
option	0	2	1	1	$\frac{1}{3}$
	3	4	2	3	

4) *TF-IDF:*
Este modelo, a diferencia del *Bag of Words*, no todos los términos reciben la misma relevancia. Tras aplicar este modelo, aquellas palabras que aparecen con bastante frecuencia en el documento, serán dotadas de mayor relevancia, al contrario que las palabras muy comunes en el corpus de entrenamiento, que recibirán una menor relevancia.
Para poder entender este modelo, tendremos que ver cada documento como un vector de *pesos*, en el que cada coordenada representaría un término del vocabulario, es decir, ese *peso* nos indica la relevancia de dicho término en el modelo. Por lo tanto nuestros documentos quedarán representados de la siguiente forma:

$$D_1 = (tf - idf_{t_1,D_1}, ..., tf_i df_{t_n,D_1})^T \tag{3}$$

Para el cálculo de dicha relevancia haremos usos de la siguiete fórmula:

$$tf - idf_{t,D} = tf_{t,D} * idf_t \tag{4}$$

donde $tf_{t,D}$ es la frecuencia del término *t* en el documento *D*, df_t el número de documentos en los que aparece el término *t*. Por último, idf_t, es el inverso de df_t, que se calularía como:

$$idf_t = \log(\frac{N}{df_t}) \tag{5}$$

donde *N* indica el número total de documentos en el corpus. Cabe mencionar que la base del logaritmo es indiferente, siempre y cuando todos los calculos se relaicen con la misma base.

Un ejemplo de datos obtenidos tras aplicar el modelo de *tf-idf* podría ser el siguiente:

TABLA II
TF-IDF

	$tf_{t,D}$		df_t	idf_t	$tf - idf_{t,D}$	
	D1	D2			D1	D2
charset	3	1	2	$\log_2(\frac{2}{2})$	0	0
texthtml	0	1	1	$\log_2(\frac{2}{1})$	0	1
option	2	0	1	$\log_2(\frac{2}{1})$	0	2

Una vez obtenidos los vectores correspondientes a cada documento, será necesario calcular la distancia entre dos documentos para poder calisficar posteriormente con el

algoritmo de *kNN*. Para el cálculo de esta similitud se aplicará la siguiente fórmula:

$$sim(D_1, D_2) = \frac{D_1 * D_2}{|D_1| * |D_2|} \tag{6}$$

$$= \frac{\sum_{i=1}^{n} tf - idf_{t_i,D_1} tf - idf_{t_i,D_2}}{\sqrt{\sum_{i=1}^{n} tf - idf_{t_i,D_1}^2} \sqrt{\sum_{i=1}^{n} tf - idf_{t_i,D_2}^2}} \tag{7}$$

5) *kNN:*
Este modelo de clasificación nos permite asociar una clase a un documento dado. Para ello serán necesarios determinar ciertos hiperparámetros, dado que no se ajustarán durante el aprendizaje.
   - *Distancia:* Nos permitirá determinar que documentos están más cernanos al nuestro. Dado que trabajaremos también con el modelo *TF-IDF*, nuestro cálculo de la distancia se realizará medienta la fórmula de la *similitud* mostrada anteriormente.

     Sin embargo tambíen existen otros tipos de distancia igual de útiles según las necesidades del problema. Entre ellos podemos destacar, para atributos numéricos, la distancia *Manhattan*:

$$d(\mathbf{x}, \mathbf{x'}) = \sum_{i=1}^{n} |x_i - x'_i| \tag{8}$$

     o incluso la distancia *Euclídea*:

$$d(\mathbf{x}, \mathbf{x'}) = \sum_{i=1}^{n} (x_i - x'_i)^2 \tag{9}$$

     Para aquellos casos en los que se trabaje con atributos discretos, una de las distancias más usadas es la de *Hamming*, la cual determina una mayor distancia cuanto menor sea el número de atributos en común:

$$d(\mathbf{x}, \mathbf{x'}) = \sum_{i=1}^{n} \mathbf{1}(x_i \neq x'_i) \tag{10}$$

     donde **1** representa la función indicador, que toma el valor 1 si el argumento es verdadero o 0 si es falso:

$$\mathbf{1}(x_i, x'_i) = \begin{cases} 1 & si \quad x_i = x'_i \\ 0 & si \quad x_i \neq x'_i \end{cases} \tag{11}$$

     En cuanto a la normalización de los atributos, no será necesario en nuestro caso, dado que ningún atributo dominará sobre el resto, pues todos ellos representan los pesos de los términos del vocabulario.
   - *Cantidad de vecinos (k):* Inidicará el número de vecinos más cercanos a nuestro documento a la hora de realizar la clasificación. El mayor número de documentos pertenecientes a una misma clase determinará la predicción del algoritmo.

Para más información sobre los modelos de aprendizaje autómatico y del procesamiento del lenguaje natural, se pueden consultar los siguientes documentos [5] [6]

III. Metodología

A. Preprocesamiento y creación del vocabulario

Para la gran mayoría de este proceso, nos hemos servido del paquete nltk de python [7].

Una de las primeras decisiones que tuvimos que tomar al empezar el trabajo fué decidir como constituir el vocabulario que emplearíamos para entrenar el modelo. Una opción que barajamos fue usar todos los términos que aparecieran en los correos, pero nos pareció demasiado exagerado, ya que quedaría un vocabulario demasiado grande. Por tanto, decidimos usar solamente los 500 términos más comunes de cada tipo de correo, es decir, unificar los 500 términos más comunes en los correos legítimos y los 500 términos más comunes en los correos marcados como spam, de forma que hubiera un equilibrio entre la cantidad de términos pertenecientes a cada tipo de correo, y que de la misma forma no hubiera términos muy poco usados que pudieran desestabilizar el modelo, es por esto que nos pareció una opción más consistente elegir estos términos más frecuentes.

Para ello, hemos procedido a dividir los datos de entrenamiento en dos partes: los correos legítimos y los correos de spam. De cada respectiva parte, hemos iterado cada correo individualmente, añadiendo a una lista la clase a la que corresponde el correo, lista que se usará más tarde para entrenar al modelo. Esto se ha podido hacer con facilidad basándonos en la ubicación en la que estaban almacenados, usando una ruta relativa al fichero .ipynb. Además, se han tokenizado las palabras de cada correo, y según el tipo de correo se ha añadido a una lista de palabras tokenizadas, distinguimos por el tipo de correo ya que estas son las listas que usaremos para construir nuestro vocabulario, por tanto las hemos separado en dos para poder calcular los 500 términos más comunes de cada una de ellas. También añadimos cada correo a una lista de todos los correos en conjunto, que es la que se usará para entrenar los modelos.

Partiendo de las dos listas que contenían las palabras tokenizadas, hemos realizado una normalización a dichas listas, y acto seguido a cada una de ellas le hemos realizado una distribución frecuencial quedándonos con las 500 más comúnes de cada una. Luego, hemos creado un vocabulario de términos, del que hemos eliminado los términos que se repitieran en ambos documentos, para así eliminar conjunciones u otras expresiones comunes y quedarnos solamente con términos determinantes a la hora de decidir de qué tipo es un correo.

También hemos dividido los datos de prueba en dos partes: nuevamente divididas en correos legítimos y correos marcados como spam, y los hemos iterado añadiendo a una lista la clase

de cada correo, para una vez clasifiquemos estos correos se puedan comparar los resultados esperados (es decir, esta lista), con los resultados obtenidos. También se ha añadido cada correo de prueba tokenizado a una lista, que posteriormente se ha normalizado. Esta es la lista que emplearemos para probar el modelo.

B. Entrenamiento y clasificación mediante Bag of Words y Naive-Bayes multinomial

El entrenamiento y la prueba del modelo se ha realizado mediante el paquete scikit-learn (sklearn) de python

En primer lugar, hemos codificado las listas que contienen las clases de los correos de entrenamiento y prueba para poder usarlas como parámetros en las funciones del paquete sklearn.

Para entrenar el modelo Bag of Words, hemos utilizado el objeto *CountVectorizer* [8] del paquete sklearn, inicializándolo con el vocabulario definido y ajustándolo a los correos del conjunto de entrenamiento, de este modo obtendremos un vector por cada correo con la cantidad de veces que se repite cada término del vocabulario en ese correo, representado como una lista de listas de enteros, donde cada lista es un correo y los enteros representan el número de veces que se repite la palabra que esté en el vocabulario en la misma posición que el entero en la lista, es decir, si la primera palabra del vocabulario es 'hola' y en el primer correo se dice dos veces, el primer entero (correspondiente a la palabra 'hola') de la primera lista (el primer documento) debe ser 2 (el número de veces que se repite la palabra 'hola' en el primer documento). También repetiremos este proceso para los correos de prueba.

Acto seguido, definiremos un objeto *MultinomialNB* [9] del paquete sklearn, y le pasaremos como parámetro el hiperparámetro de suavizado K. Entrenaremos este modelo con la lista de listas de enteros calculada anteriormente para el conjunto de entrenamiento, y con las clases de cada correo del mismo conjunto de entrenamiento codificadas. De tal modo, se calcularán las probabilidades que tiene cada palabra de aparecer en un correo legítimo y en un correo de spam.

Una vez que hemos entrenado nuestro modelo Naive-Bayes multinomial, es hora de obtener una predicción. Para ello usaremos el método *predict*, pasándole la lista de listas de enteros calculada sobre el conjunto de prueba, lo que nos devolverá una lista de resultados codificados. Una vez decodifiquemos esta lista, tendremos una lista con strings que serán o bien 'legítimo' o bien 'spam'. Cada string representa la clase asignada al correo de prueba que estviese en la misma posición que la string en la lista, es decir, si la 4a string de la lista es 'spam', el 4o correo ha sido clasificado como spam.

C. Entrenamiento y clasificación mediante TF-IDF y Knn

El entrenamiento y la prueba del modelo nuevamente se ha realizado mediante el paquete sklearn.

Se han vuelto a usar las mismas listas codificadas que se explican en el apartado anterior como parámetros de las funciones, pues aunque el modelo que se va a entrenar sí que varía, los datos con los que se va a entrenar al modelo son los mismos.

Para entrenar el modelo TF-IDF, se ha utilizado un objeto tipo *TfIdfVectorizer* [10], que funciona de manera muy similar al *CountVectorizer*, ya que inicializamos el *TfIdfvectorizer* con el vocabulario que se ha creado y se ajusta a los correos de entrenamiento de la misma forma que el *CountVectorizer*. La principal diferencia es que no obtendremos una lista de listas de enteros, si no una lista de listas de decimales, donde al igual que con el *CountVectorizer*, cada lista representa un correo, y el decimal la TF-IDF de la palabra cuya posición en el vocabulario sea la misma que la posición del decimal en la lista. Utilizando el mismo ejemplo que se utilizó para explicar el CountVectorizer, si la primera palabra del vocabulario es 'hola', el primer decimal (correspondiente a la palabra 'hola') de la primera lista (el primer documento) será la TF-IDF de la palabra hola en el primer documento. Al igual que con Bag of Words, repetiremos este proceso para los correos de prueba.

Tras esto, definiremos un objeto del tipo *KNeighborsClassifier* [11], al que le introduciremos como parámetro el hiperparámetro k (el número de vecinos), y le pasaremos a este la lista de listas de decimales y las clases codificadas de los correos del conjunto de entrenamiento.

Por último obtendremos la clasificación de los correos de pruebas mediante este modelo Knn, que se realizará del mismo modo que con el objeto *MultinomialNB*, o sea, utilizaremos el método *predict* de *KNeighborsClassifier*, pasándole la lista de listas de enteros que se calculó con los correos de prueba, y el resultado descodificado será una lista con la clase asignada, 'legítimo' o 'spam', a cada correo del conjunto de prueba.

IV. Resultados

Unas vez entrenado los algoritmos y realizado las predicciones con los conjuntos de validación, hemos procedido a calcular ciertas métricas que nos permitiesen hacer una comparación.

Todos los calculos realizados se aplicarán sobre los datos obtenidos por los filtros tras incorporar las mejoras, puesto que son la versión final de los algoritmos.

Las métricas empleadas para la comparativa serán *accuracy*, *recall*, *precision* y *especificidad* [12].

Pero para poder entender estas métricas, debemos tener claro que es una *matriz de confusión*, que dado que nuestro problema es de clasificación binaria, nos será muy útil.

Primero asignaremos una de las clases de nuestro problema como *positiva*, en nuestro caso la clase de correo *spam* o *no deseado*, y como clase *negativa* aquellos correos *legítimos*. Una vez hecho esto, se calculará lo siguente:

- *Verdaderos Positivos:* número de correos asignados correctamente como *positivos*.
- *Falsos Negativos:* número de correos asignados incorrectamente como *positivos*.
- *Verdaderos Negativos:* número de correos asignados correctamente como *negativos*.
- *Falsos Positivos:* número de correos asignados incorrectamente como *negativos*.

TABLA III
MATRIZ DE CONFUSIÓN

		Clase predicha	
		Positiva	Negativa
Clase correcta	Positiva	VP	FN
	Negativa	FP	VN

Una vez conocemos la *matriz de confusión*, podremos calcular las siguentes métricas que se mencionaron anteriormente:

- *Accuracy:* nos indica el porcentaje total de elementos clasificados correctamente. Se calcula de la siguiente forma:

$$Accuracy = \frac{VP + VN}{VP + FP + VN + FN} \tag{12}$$

A pesar que esta métrica no es del todo fiable en los casos en los que el número de elementos pertenecientes a una clase sea superior a la otra, pues si decimos que todos son positivos, y de ese misma clase hay un 90 porciento, nos dará una precisión del 0.9. Sin embargo, para nuestro caso, al estar repartidos de forma equitativa, es una métrica bastante fiable.

- *Recall:* nos indica el porcentaje de elementos indentificados como positivos del total de verdaderos positivos. Se calcula de la siguiente forma:

$$Recall = \frac{VP}{VP + FN} \tag{13}$$

- *Precision:* nos indica el porcentaje de elementos identificados correctamente como positivos del total de elementos indentificados como positivos. Se calcula de la siguient forma:

$$Precision = \frac{VP}{VP + FP} \tag{14}$$

- *Especificidad:* nos indica el porcentaje de elementos correctamente identificados como negativos del total de negativos. Se calcula de la siguiente forma:

$$Especificidad = \frac{VN}{VN + FP} \tag{15}$$

Tras calcular las métricas para ambos modelos clasificatorios, e iterando el valor de sus hiperparámetros, mostrando unicamente los valores para k mayor que 7, dado que el modelo *kNN* no se estabiliza hasta alcanzar esos valores en el hiperparámetro, disminuyendo así la precisón de la gráfica por valores tan dispares, hemos obtenido los siguientes resultados:

- Comparativa accuracy:

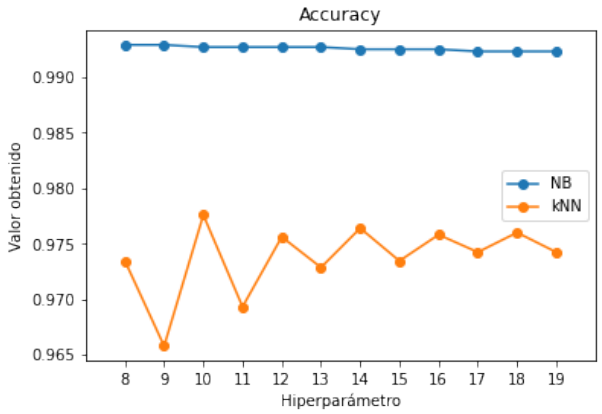

Fig. 1. Comparación en la métrica accuracy

- Comparativa recall:

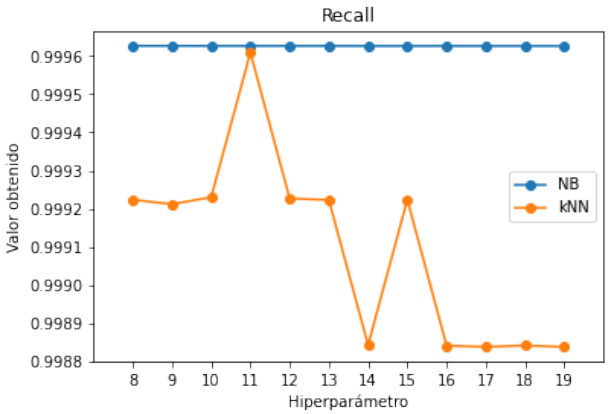

Fig. 2. Comparación en la métrica recall

- Comparativa precision:

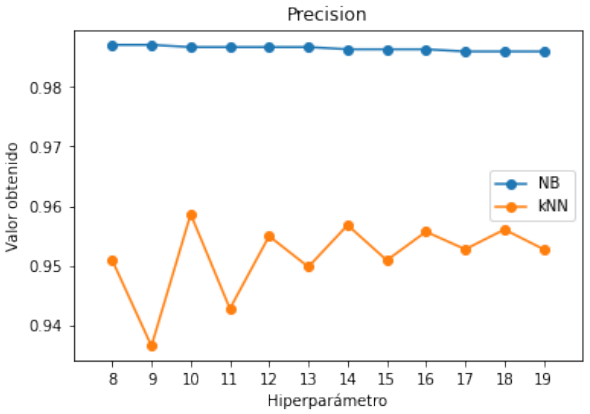

Fig. 3. Comparación en la métrica precision

- Comparativa especificidad:

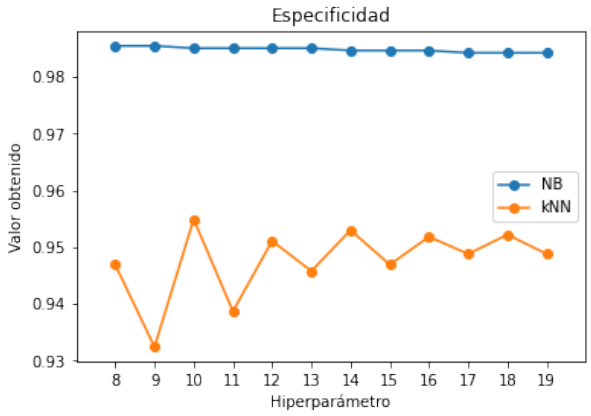

Fig. 4. Comparación en la métrica especificidad

Tras haber calculado todas las métricas, deberemos excoger una de ellas para la evaluación, ya no solo para determinar cual de los dos modelos a resultado ser más eficaz, sino para una futura mejora del algoritmo, teniendo claro que hay que optimizar.

Sin embargo puede que la decisión de excoger una única métrica no se fácil, ya que no cubriría todos los aspectos que se consideran importantes a la hora de determinar si se ha realizado una buena clasificación.

Para estos casos, es recomendable usar la media armónica, dando valores más cercanos de aquellas métricas que tengan un valor más bajo. Su cálculo es el siguiente:

$$MediaArmonica = \frac{N}{\frac{1}{M_1} + \dots \frac{1}{M_n}} \tag{16}$$

donde N es el número total de elementos incluidos en la media, es decir el número de métricas, y M_n representa el valor obtenido de una métrica.

Sin embargo, dado que el porcentaje de correos del conjunto de prueba clasificados como *legítimos* es similar a los clasificados como *no deseados*, hemos considerado que la mejor métrica para evaluar los modelos es la *accuracy*, ya que tiene encuenta los aciertos y fallos para ambas clases. Por tanto nuestro algoritmo seleccionado para la realización del filtrado de correos de spam es el modelo de *Naive-Bayes multinomial*, demostrando tener unos resultados bastante estables frente a *kNN*, para cualquier valor del hiperparámetro del suavizado.

V. Conclusiones

Tras los resultados obtenidos en la comparativa mediante la métrica *accuracy*, podemos determinar que el modelo de *Naive-Bayes multinomial* es el más eficaz para la tarea de claisficación de correos. No solo es más eficaz, sino que más eficiente, pues el coste de aumentar su hiperparámetro de suavizado, sólo inluye en el cálculo de las probabilidades condicionadas de los términos del vocabulario. Sin embargo, en el modelo *kNN*, aumentar su hiperparámetro, el cual determina la cantidad de vecinos más cercanos a tener en cuenta, incrementa el coste computacional, pues debe calcular más distancias entre los distintos vecinos y realizar las comparativas para obtener una predicción.

Cabe también mencionar la evolución de ambos modelos de aprendizaje automático tras aplicar las mejoras en los filtros. Mientras que el modelo de *Naive-Bayes multinomial* presenta una mejoría, el modelo *kNN* presenta unos resultados bastante iteresantes.

Estos son los resultados obtenidos, donde el eje X representa el valor del hiperparámetro, y el eje Y el valor obtenido de la métrica *accuracy*:

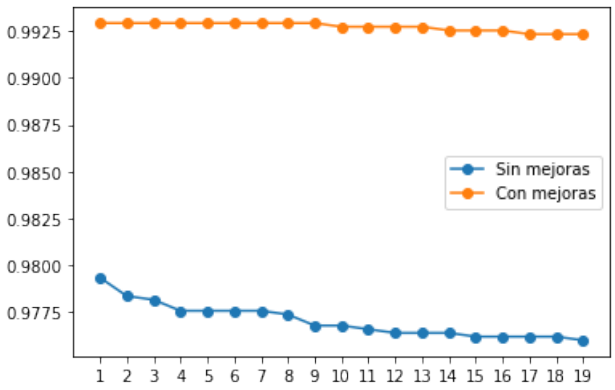

Fig. 5. Comparativa Navie-Bayes multinomial

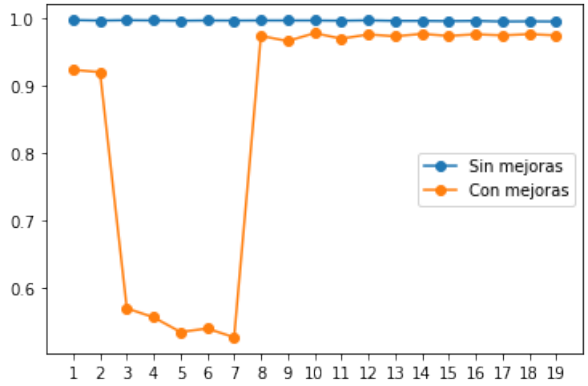

Fig. 6. Comparativa kNN

Podemos observar que en este útlimo modelo, entre los valores 3 y 7 del hiperparámetro, reduce el valor de la métrica obtenida incluso a la mitad con respecto a los valores obtenidos antes de aplicar las mejoras en el filtro. Lo más seguro es que esto se deba a que el número de vecinos no es ni muy pequeño ni muy grande; con un número grande lo más lógico es que se produzca un menor error ya que al tener una muestra tan grande donde los correos son parecidos entre sí, las menores distancias deben ser por lo general a los correos de la clase adecuada, con un número más pequeño debería haber mayor margen de error puesto que puede que

un correo que sea spam se parezca mucho a algunos que no lo sea y viceversa, y al tener un menor número de vecinos esos correos que se parezcan serán determinantes a la hora de clasificar. Sin embargo, esto empieza a pasar a partir de k=3, lo cuál nos hace pensar que, si bien es cierto que algunos correos spam se asemejan con otros no spam, con un número de vecinos muy pequeño la menor distancia o las dos menores distancias siguen siendo a correos de su respectiva clase, y se necesita una cantidad de vecinos ligeramente mayor para que empiece a clasificar erróneamente. Esta similitud entre los correos principalmente se debe a las técnicas de preprocesamiento introducidas, que mediante eliminación de ruido y otras técnicas deja el correo limpio de tags HTML y elementos similares, es por esto que el modelo Knn clasificaba mejor antes de aplicar las mejoras, ya que las tags HTML y otros elementos similares prácticamente solo estaban presentes en correos spam.

Sin embargo, a paritr del valor 8 del hiperparámetro, se estabilizan más los valores de la métrica. En esta gráfica podemos observar con más detalle la diferencia de valores:

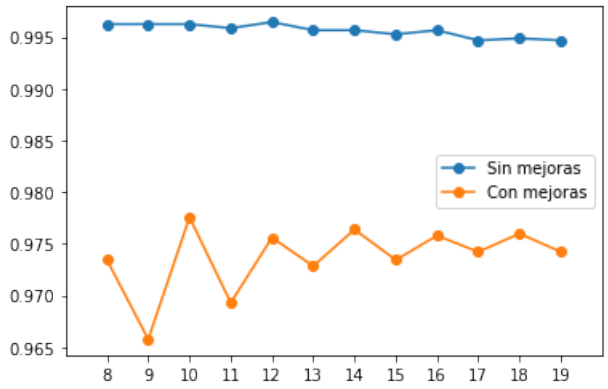

Fig. 7. Caption

REFERENCIAS

[1] Daven Hiskey (2010, Sept.) How the word "SPAM" came to mean "junk message". Today I Found Out. Recuperado de: http://www.todayifoundout.com/index.php/2010/09/how-the-word-spam-came-to-mean-junk-message/.

[2] Tushaar Gangavarapu, C. D. Jaidhar, Bhadesh Chanduka (2020 Feb.) Applicability of machine learning in spam and phishing email filtering: review and approaches. Recuperado de: https://link--springer--com.us.debiblio.com/article/10.1007/s10462-020-09814-9.

[3] Elizabeth D. Liddy (2001) Matural Language Processing. Syracuse University. Recuperado de: https://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub.

[4] Matthew Mayo (2018, May.) Preprocesamiento de datos de texto: un tutorial en Python. CienciaDatos. Recuperado de: https://medium.com/datos-y-ciencia/preprocesamiento-de-datos-de-texto-un-tutorial-en-python-5db5620f1767.

[5] Dpto. de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla (2022, Apr.) Procesamiento del Lenguaje Natural. Recuperado de: https://www.cs.us.es/docencia/aulavirtual/pluginfile.php/13015/mod_resource/content/3/Procesamiento_del_lenguaje_natural.pdf.

[6] Dpto. de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla (2022, Mar.) Aprendizaje automático. Recuperado de: https://www.cs.us.es/docencia/aulavirtual/pluginfile.php/13004/mod_resource/content/2/Aprendizaje_autom%C3%A1tico.pdf

[7] NLTK Project (2022, Mar.) Source code for NLTK Vocabulary. Recuperado de: https://www.nltk.org/_modules/nltk/lm/vocabulary.html

[8] Scikit-learn developers (2022) The CountVectorizer class Recuperado de: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[9] Scikit-learn developers (2022) The MultinomialNB class Recuperado de: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[10] Scikit-learn developers (2022) The TfIdfVectorizer class Recuperado de: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[11] Scikit-learn developers (2022) The KNeighborsClassifier class Recuperado de: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[12] (2019 Jan.) Maching Learning: Selección Métricas de clasificación. Recuperado de: https://sitiobigdata.com/2019/01/19/machine-learning-metrica-clasificacion-parte-3/