DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Métodos Numéricos

Integrante	LU	Correo electrónico
Rodrigo Laconte	193/18	rola1475@gmail.com
Macarena Piaggio	212/18	piaggiomacarena@gmail.com
Julia Rabinowicz	48/18	julirabinowicz@gmail.com
Amalia Sorondo	281/18	sorondo.amalia@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
https://exactas.uba.ar

Resumen

En este trabajo se busca estudiar el método de k vecinos más cercanos combinado con el análisis de componentes principales para clasificar reseñas sobre películas en IMDB como positivas o negativas. Modificando la elección de los parámetros que influyen en la precisión de los algoritmos elegidos, analizaremos la combinación de los mismos que resulta más eficiente. Con esta motivación realizamos pruebas experimentales sobre los métodos, de las que concluimos que para obtener los resultados óptimos en cuanto a la predicción de las reseñas es necesario un balance entre los parámetros; moderar los umbrales al filtrar palabras del modelo utilizado, una cantidad justa de vecinos al aplicar el método de los k vecinos así como en la elección del α del método de componentes principales y en el tamaño del conjunto de entrenamiento de las instancias.

Palabras claves:

- K vecinos más cercanos
- Análisis de componentes principales
- Análisis de sentimiento
- Conjunto de entrenamiento

Índice

Página

1. Introducción

En la actualidad poder extraer información representativa de la opinión de los usuarios de una plataforma o sitio web es un tema de gran interés por diversos motivos. Las redes sociales y las páginas web proveen un espacio de libre expresión en donde la gente suele compartir sus opiniones sobre los temas de interés del momento. Esto sumado a la interacción entre diferentes usuarios genera que en estos espacios las empresas o grupos políticos encuentren las ideas que las personas a las que están dirigidas sus publicidades tienen sobre su producto o su candidato. Muchas veces, al haber discusiones entre varios usuarios, se presentan críticas que probablemente mediante una encuesta u otro medio, no se podrían haber obtenido.

Esta capacidad de procesar y estudiar de forma eficaz la opinión de los usuarios representa una gran ventaja en el ámbito comercial. Al conocer los intereses de los usuarios y tener feedback sobre sus productos, las empresas pueden explorar nuevos mercados o modificar sus estrategias de venta o su línea de productos para satisfacer las necesidades de los clientes. De esta misma forma, las publicidades pueden dirigirse de forma más específica, variando según lo que los usuarios expresaron en comentarios o basadas en sus historiales de búsqueda.

El estudio y análisis de la opinión pública sobre temas de actualidad puede ser también de gran uso en la política. De esta forma, en la búsqueda de aceptación pública una campaña política puede verse influenciada por la tendencia en las opiniones de los usuarios en las redes sociales.

Para facilitar esta extracción de información de las plataformas virtuales y el análisis de la misma, existen algoritmos, generalmente relacionados con la idea de inteligencia artificial y machine learning. En este trabajo vamos a sustraer datos de críticas a películas en IMDB, con el objetivo de analizar la connotación emocional de una reseña. Queremos, dada una crítica, poder predecir si ésta es "positiva" o "negativa". Este procedimiento pertenece al análisis de sentimiento: busca clasificar documentos de manera automática en función del estado de ánimo de un sujeto. Para realizar el estudio implementamos la técnica de clasificación de vecinos más cercanos (K-Nearest Neighbors) con la ayuda del análisis de componentes principales (Principal Component Analysis). Previo a la explicación de estos métodos es necesario presentar el concepto de "bolsa de palabras".

Como ya mencionamos, vamos a trabajar con datos en forma de texto, por lo que contaremos con un vocabulario. Se vuelve necesario un método como el de "bolsa de palabras" para representar el texto y poder comparar dos entradas de forma cuantitativa. Tomaremos como base un vector cuyas posiciones representan cada una de las palabras pertenecientes al vocabulario, luego cada texto será tomado como un vector escrito en coordenadas de la base. Esto lleva a la pregunta de qué ocurre si contamos con un vocabulario muy extenso: habría que trabajar con vectores de gran dimensión. En este contexto, para evitar este problema implementaremos el análisis de componentes principales o PCA [1]. Éste nos permitirá reducir la dimensión de la muestra, es decir, la base de datos con la que contamos, para poder predecir si una reseña es positiva o negativa. Gracias a este procedimiento se logra conservar un conjunto de datos más acotado y por lo tanto más fácil de procesar, pero que contenga información lo suficientemente representativa como para poder clasificar objetos de la base de entrada.

El método de KNN [2] consiste en, mediante una base de datos ya clasificados, catalogar una nueva instancia, es decir, uno de los vectores, según los k datos a los que más se asemeja. Con esos datos se elige para el nuevo la clase a la que la mayoría pertenece.

[1] https://es.wikipedia.org/wiki/Análisis_de_componentes_principales
[2] https://www.analiticaweb.es/algoritmo-knn-modelado-datos/

En la siguiente sección se incluye una descripción de los métodos utilizados para el análisis de las reseñas: K-Nearest Neighbors, Método de la potencia y PCA. Presentaremos también la forma en la que fueron implementados los algoritmos correspondientes a cada uno. Además se incluirán detalles sobre la forma en que fue realizada la experimentación.

2. Desarrollo

2.1. Sobre los métodos implementados

Como mencionamos anteriormente, en este trabajo estudiaremos el estado de ánimo de las personas al momento de escribir sus reseñas, con el fin de poder predecir, dada una crítica, si ésta es positiva o negativa. Para esto implementamos distintos métodos que describiremos a continuación, así como la forma en la que éstos fueron implementados. En el funcionamiento de estos métodos influyen distintos parámetros. Experimentaremos entonces con estos parámetros y cómo estos y sus posibles combinaciones afectan la precisión del método de predicción.

2.1.1. Análisis de componentes principales y método de la potencia

Para poder comparar dos textos de forma cuantitativa es necesario contar con algún modelo de representación para los mismos. Utilizamos en este trabajo el modelo de bolsa de palabras. En este modelo se ignora el orden de las palabras de un texto y únicamente se especifica, dado un vocabulario base, la cantidad de veces que aparece cada palabra de ese vocabulario en el texto bajo análisis. Luego, si contamos con un vocabulario de m palabras, los textos van a ser representados como vectores de m coordenadas, donde la i-ésima coordenada representa la cantidad de apariciones de la palabra número i en el texto.

Este modelo presenta el problema de la elección del tamaño del vocabulario ya que en general éstos pueden ser muy extensos y contar con más de 160.000 palabras. Trabajar con vectores tan grandes implica costos temporales y espaciales considerables. Para resolver esto, no consideraremos palabras presentes en estos vocabularios que no aporten información significativa a la hora de extraer información sobre un texto: es el caso de las palabras con alta frecuencia como preposiciones. También se excluirán palabras con baja frecuencia ya que será raro encontrarlas tanto en los textos pertenecientes al conjunto de datos que sirven para el entrenamiento, como en los textos cuya connotación queremos predecir. Logramos de esta forma reducir el tamaño de los vectores, aunque éstos siguen siendo muy grandes y por eso métodos como el análisis de componentes principales resultan útiles.

El método de análisis de componentes principales o PCA se utiliza para reducir la dimensión de los datos conservando las α componentes más representativas para una buena clasificación de las instancias de entrada. Dada una matriz que contiene los datos muestrales, el método de análisis de componentes principales calcula primero la matriz de covarianza de la muestra. Luego tomando los α mayores autovalores en módulo, se crea la matriz de cambio de base conformada por los autovectores asociados a los mismos. Esta matriz permite pasar los datos de entrada a un espacio cuyas dimensiones resultan más apropiadas. Detallamos a continuación de qué manera elegimos implementar el método de análisis de componentes principales.

El algoritmo de PCA cuenta con dos funciones: `fit` y `transform`. La primera recibe una matriz y asigna al método "autovectores" de la clase PCA la matriz formada por los autovectores de la matriz de covarianza de la entrada; a continuación nos explayamos sobre cómo implementamos esta función. Sea X la matriz de entrada, a cada fila de la matriz, es decir a cada instancia del conjunto de datos, se le resta el promedio coordenada a coordenada de los datos. Luego multiplicamos la matriz obtenida en el paso anterior por su transpuesta y dividimos este producto por (n-1) siendo n la cantidad de datos en el conjunto. Finalmente, utilizamos el método de la potencia para obtener la matriz conteniendo los autovectores asociados a los α mayores autovalores en módulo, siendo α la cantidad de componentes de los datos con la que queremos trabajar. Le asignamos al método esta matriz de autovectores. La

segunda función, `transform`, recibe la matriz de autovectores calculada por `fit` y la convierte en una matriz densa. Luego devuelve el resultado de multiplicar esta matriz por la matriz autovectores de la clase PCA.

Implementamos además una función extra, `newAlpha`, para evitar recalcular innecesariamente la matriz de autovectores al experimentar, cuando queremos cambiar el alpha a utilizar sobre el mismo conjunto de entrenamiento. Si inicialmente se transforma la base de entrenamiento utilizando el mayor α a observar, para utilizar la base entrenada con un α menor no es más que recortar los autovectores innecesarios de la matriz de autovectores. Esto evita tener que recalcular los mismos autovectores más de una vez, generando una mejora en el tiempo de cómputo.

Como dijimos anteriormente, para calcular los primeros α autovalores utilizamos el método de la potencia acompañado de un proceso de deflación de la matriz en cuestión. El algoritmo implementado del método de la potencia toma una matriz (la matriz de covarianza), un número que representa la cantidad de iteraciones y un epsilon que será utilizado para determinar el criterio de salida del ciclo principal. El algoritmo consiste en tomar como x_0 un vector aleatorio de la misma dimensión que las filas de la columna, a partir del cual se busca aproximar el autovector asociado al autovalor de mayor módulo guardando para cada iteración el vector multiplicado por la matriz y normalizado. Luego compara el vector obtenido con el de la iteración anterior y si la norma de la diferencia es menor al epsilon, finaliza. En caso de que nunca se llegue a esta instancia, finaliza cuando realice el número de iteraciones indicado por parámetro. Finalmente calcula el autovalor. La implementación que calcula el resto de los autovalores y autovectores consiste en aplicar el método de la potencia a la matriz, pero aplicándole el método de deflación en cada iteración, es decir que a la matriz se le resta la matriz resultante de multiplicar el autovector por su traspuesto multiplicado por el autovalor.

2.1.2. K vecinos más cercanos

Una vez preprocesados los datos muestrales, utilizamos el algoritmo de k vecinos más cercanos para clasificar una reseña de entrada no presente en el conjunto de entrenamiento. Este método busca las k instancias más cercanas según sus bolsas de palabras y le asigna a la instancia bajo análisis la clase que representa la moda dentro de los k vecinos.

Con el fin de clasificar un conjunto de instancias, pasadas como entrada en forma de matriz, implementamos el método de los k vecinos más cercanos gracias a cuatro funciones. La primera función, `distanceToRow`, toma una instancia y devuelve el vector conteniendo las normas de la distancia entre la entrada y las instancias pertenecientes al conjunto de entrenamiento. La norma que decidimos utilizar para determinar la cercanía de dos instancias es la norma 2. Elegimos esta norma ya que nos pareció adecuada para determinar la distancia entre dos instancias representadas por el modelo de bolsa de palabras donde cada componente de los vectores que las representan contiene la cantidad de apariciones de cada palabra del vocabulario. Esta función convierte la matriz de *sparse* a densa para poder utilizar operaciones exclusivas para matrices densas como `rowwise`.

Luego este vector de normas es utilizado para predecir la clase de una instancia utilizando la función `predictRow`. Esta función calcula la moda de las clases de los k vecinos más cercanos de una instancia. Para efectuar esto último, crea un vector temporal conteniendo para cada instancia de entrenamiento, un par cuya primera componente es la norma de la distancia con la instancia de entrada y la segunda, la posición original de la reseña en el conjunto de entrenamiento. Ordenamos este vector de menor a mayor utilizando el algoritmo de ordenamiento de C++ en función a la primera componente. Los primeros elementos representan entonces las instancias del conjunto de entrenamiento que presentan las menores distancias con la instancia de entrada. Finalmente, mediante dos contadores recorremos las primeras k posiciones del vector calculando, gracias a la segunda componente de los pares, la cantidad de reseñas positivas y negativas presentes. Guardamos los "votos" de cada una de las primeras k instancias en un vector, en el cual los votos estan ordenados desde el del vecino más cercano al del

más lejano, y lo colocamos en una "matriz de votos" (`_vote_mat`) que tiene tantas filas como reseñas hay en la matriz de reseñas cuya valoración se quiere predecir, la cual `predict` recibe como entrada. Esta última función aplica las dos anteriores y genera un vector con las predicciones para cada reseña de su entrada.

Decidimos guardar los votos de cada uno de los vecinos más cercanos en una matriz en lugar de guardar únicamente el resultado de su votación para poder calcular los resultados de las experimentaciones más rápidamente. Cuando para un clasificador queremos ver qué ocurre con las predicciones para un conjunto de reseñas si tomamos cantidades variables de vecinos, lo único que tenemos que hacer es calcular primero la matriz de votos para la mayor cantidad de vecinos a observar, y luego recortar columnas de esta matriz según cuántos vecinos menos queramos observar (utilizando la función `predictNewK`). A partir de la matriz, calcular la clasificación con mayor presencia no es más que calcular el promedio para cada fila de la matriz. Utilizamos una matriz de tipo Sparse para implementar la matriz de votos ya que la clasificación es binaria, por lo que aproximadamente la mitad de sus elementos serán 0 (representando un voto para clasificar la reseña como negativa) y la otra mitad, 1 (voto para clasificación positiva).

2.2. Sobre la experimentación

Como ya señalamos, el objetivo de este trabajo es analizar los parámetros que inciden en la precisión y en la calidad de los resultados de nuestro algoritmo de clasificación de reseñas de IMDB.

Los parámetros sobre los que nos interesó experimentar para alcanzar resultados óptimos son:

- Número k de vecinos tomados
- Cantidad de componentes principales considerados (α de PCA)
- Tamaño de la base de entrenamiento (y, en menor medida, relación entre este y el tamaño del conjunto de test)
- Criterio de parada del método de la potencia
- Umbrales del modelo de bolsa de palabras

Para realizar la experimentación implementamos tests que, en el comienzo, mantenían el tamaño de la base de entrenamiento y de test fijos, y variaban la cantidad de vecinos en KNN y solo el α para PCA. Con estos resultados provisorios formamos una primera idea de en qué rangos nos interesaba más experimentar para ambos parámetros, y con esto en mente complejizamos los experimentos, haciendo que cantidad de vecinos y α variaran en conjunto. Finalmente, hicimos nuevas corridas de los mismos experimentos cambiando el tamaño de las bases de entrenamiento y test y observamos las diferencias entre todos los resultados obtenidos, y para los mejores resultados afinamos la precisión de nuestras medidas.

Como ya mencionamos, con el objetivo de optimizar la experimentación al usar PCA, decidimos no recalcular la matriz de cambio de base, es decir la matriz que contiene los autovectores de la matriz de covarianza, al experimentar con α diferentes. Comenzamos entonces el experimento calculando esta matriz para el mayor α_1 y luego al tomar un α_2 menor en lugar de recalcular la matriz simplemente recortamos la matriz ya calculada, tomando únicamente las primeras α_2 columnas.

En cuanto a experimentar con distintos tamaños de las bases de entrenamiento, nos pareció que no tenía sentido experimentar con bases de entrenamiento extremadamente chicas o grandes, ya que en el primer caso las predicciones obtenidas serían demasiado arbitrarias y en el segundo no tendríamos suficiente base de test restante como para medir la precisión de manera confiable. Por esto elegimos dividir la base de datos de IMDB en 1/4 (3125 reseñas), 1/2 (6225 reseñas) y 3/4 (9375 reseñas) de

las instancias para la base de entrenamiento y lo restante para test. Para tomar estas cantidades de instancias, primero reordenamos en forma aleatoria el conjunto de todas las instancias y agarramos 1/4 de las mismas para formar la base de entrenamiento. Este procedimiento lo repetimos para todos los tamaños: antes de formar el conjunto de entrenamiento para cualquier proporción reordenamos aleatoriamente todas las instancias. Realizamos esto para evitar estar optimizando los parámetros para un conjunto de instancias en un orden específico y de esta forma perder generalidad en nuestras deducciones.

Como reordenamos de manera aleatoria la base de instancias al realizar cada experimento, decidimos promediar varias corridas de cada uno de éstos para asegurarnos que los resultados obtenidos representen adecuadamente los fenómenos estudiados. Luego, los experimentos que analizan las distintas proporciones de las bases de entrenamiento y test, así como el test de umbrales de filtro de palabras, fueron corridos 5 veces y luego promediados. En cuanto al experimento de las normas, como éste mide cuestiones temporales además de la precisión de resultados elegimos correrlo 25 veces.

3. Resultados y discusión

3.1. KNN: Cantidad de vecinos

En primer lugar estudiaremos cómo varían los resultados al elegir para la votación diferentes cantidades de vecinos. Con un dataset de entrenamiento fijo, analizaremos esta modificación, en primer lugar, sin el método de análisis de componentes principales para recortar la dimensión de los datos de entrada.

Hipótesis

Si bien podría resultar intuitivo que cuantos más vecinos mejor funciona el algoritmo de clasificación, creemos que una de las desventajas de tomar tantas instancias del dataset de entrenamiento es que se estaría comparando la entrada con algunos datos que quizás ya se encuentran muy alejados de la misma. Es decir, que si consideramos una cantidad muy alta de vecinos, probablemente los primeros sean una buena aproximación pero el resto no, lo que llevaría a resultados erróneos al estar considerando los más alejados en la votación, ya que, al haber tomado tantos vecinos, éstos podrían superar en cantidad a los que sí se encontraban más cerca de la instancia en consideración. En contraste con esto, si elegimos muy pocos puede ser que estemos dejando afuera a las instancias que se encontraban más cerca de nuestro dato de entrada. Con esto nos referimos a que puede pasar que estemos agarrando instancias cercanas respecto de la norma pero que no pertenezcan a la misma clase, por lo que si tomábamos un k mayor, probablemente hubieran entrado en consideración más instancias que sí compartían la misma clase con la que estamos analizando. En conclusión, esperamos que el k que optimiza el algoritmo en función de la calidad del resultado sea relativamente chico pero lo suficientemente grande como para abarcar las instancias más cercanas.

Sumado a esto, al no utilizar el método de PCA, los datos, tanto del conjunto de entrenamiento como los de entrada que vamos a clasificar, tendrán información que no será de gran relevancia al momento de comparar las instancias para elegir los k vecinos más cercanos. Por está razón esperamos que se necesite un k más grande que al combinar KNN con PCA.

Experimento

Para verificar cuál es el k que optimiza el algoritmo lo corrimos con una base de entrenamiento (`X_train`) que representa 3/4 de todas las instancias con las que contamos. Para cada una de las instancias restantes en el conjunto de test, ejecutamos el KNN con k múltiplo de 10, desde 10 hasta 200. Observamos la accuracy de cada resultado y obtuvimos lo siguiente:

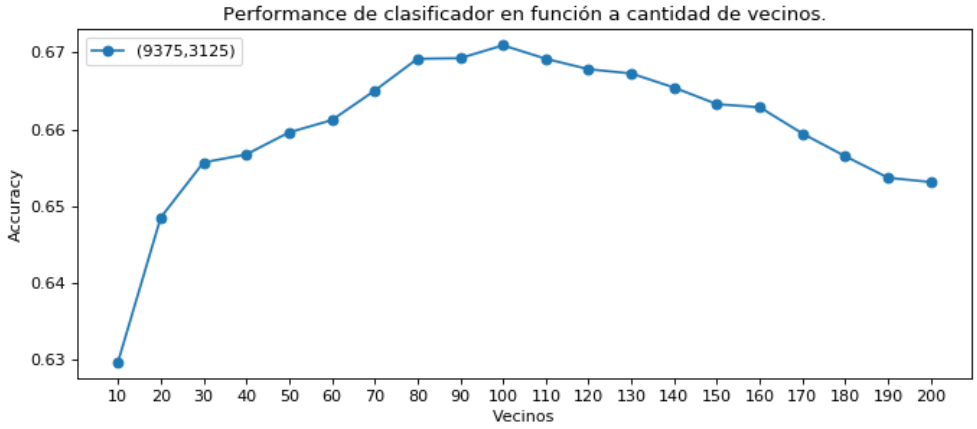

Figura 1: Resultados del experimento sobre la cantidad de vecinos.

Resultados y conclusiones

Podemos ver en la figura 1 que la curva que realiza la accuracy en función a la cantidad de vecinos comienza creciendo hasta alcanzar un máximo en $k = 100$, luego del cual la exactitud de la predicción comienza a disminuir. Nuestras hipótesis se ven entonces confirmadas. Resulta necesario considerar una cantidad suficiente de vecinos como para poder clasificar una instancia en base a un número representativo de instancias cercanas. En este caso $k = 100$ parece ser una cantidad de vecinos lo suficientemente significativa. Sin embargo, seguir aumentando el k no garantiza mejores resultados en cuanto a la accuracy, como podemos ver en el gráfico. Al tomar demasiados vecinos, éstos pueden representar instancias alejadas de la que se encuentra bajo estudio, por lo que pueden pertenecer a clases diferentes y perjudicar la predicción. En nuestro experimento, si tomamos más de 100 vecinos la exactitud de nuestra predicción comienza a disminuir debido a este fenómeno. Podemos constatar también que para los valores de k entre 70 y 140, si bien el valor de la accuracy varía, ésta se mantiene alta.

3.2. KNN y PCA: Cantidad de componentes principales

Analizaremos ahora qué ocurre al combinar el algoritmo de KNN con el de análisis de componentes principales: para diferentes valores de k, estudiaremos variaciones en los resultados con distintas cantidades de componentes principales (α) que consideraremos.

Hipótesis

Como ya aclaramos, el α representa la cantidad de componentes principales que tendremos en cuenta al momento de clasificar instancias. Al tener un α demasiado pequeño creemos que no se va a estar considerando suficiente información relevante, por lo que es probable que se clasifiquen incorrectamente las instancias. Por otra parte, si tomamos un α considerablemente grande, el objetivo del PCA se vería perjudicado ya que seguiríamos teniendo en cuenta componentes que no aportan datos importantes, por lo que esperamos, igual que con el k, que el α que optimice KNN en combinación con PCA sea moderado, pero lo suficientemente grande.

Experimento

Al igual que en el experimento anterior, corrimos el algoritmo (ahora utilizando el de PCA también) con una base de entrenamiento que representa 3/4 de todas las instancias. Para cada una de las

instancias del conjunto de test, ejecutamos el KNN con k múltiplo de 10, desde 10 hasta 200 y con α múltiplo de 20 entre 20 y 100. Observamos la accuracy de cada resultado y obtuvimos lo siguiente:

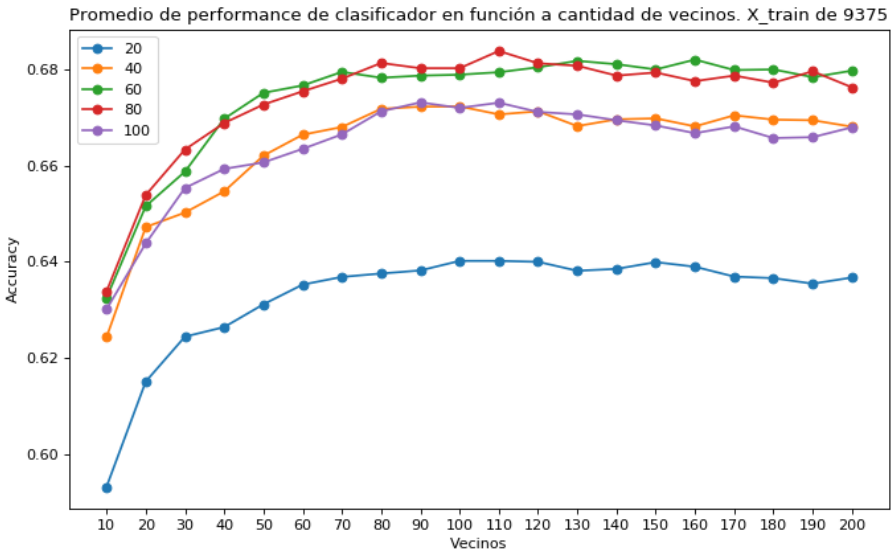

Figura 2: Resultados del experimento sobre cantidad de componentes principales combinado con cantidad de vecinos. Cada línea del gráfico representa el valor de α que muestra el recuadro de la esquina.

Resultados y conclusiones

Podemos constatar que en la figura 2 los α más chicos, como 20, resultan ser los de menor accuracy, y que la diferencia entre sus respectivas precisiones con las de los α mayores, es mayor que la que se obtiene entre los α entre 40 y 100. Es interesante observar que, por más que en algunas corridas individuales (no figuradas) a $\alpha = 100$ le iba mejor que a 60 y 80, en promedio resultó ser peor que estos otros. Analicemos ahora el caso de $\alpha = 60$ y $\alpha = 80$. Podemos ver que, tal como habíamos observado en el experimento de KNN sin PCA, para los k menores, más precisamente entre 10 y 40, se obtiene poca accuracy. Sin embargo, para el resto de las cantidades de vecinos, la accuracy presenta ciertas variaciones pero siempre se mantiene relativamente alta. Mirando con más detenimiento, podemos decir que la mayor accuracy se encuentra con la combinación de $\alpha = 80$ y $k = 110$.

Podemos concluir entonces que, como esperábamos inicialmente, los mayores α presentan mejor calidad en los resultados obtenidos, aunque la mayor precisión no se alcanza para el mayor α sino que en un valor moderado. Sin embargo, observamos también que la mejor combinación con la cantidad de vecinos se obtiene en un k cercano al conjunto de los que optimizaban el algoritmo de KNN sin PCA. Esto contradice nuestra hipótesis de la sección anterior, que planteaba que los k que optimizaban el algoritmo sin PCA serían mayores que los que analizamos en este experimento. La aplicación de PCA parecería haber vuelto más confiables los resultados para cantidad de vecinos mayores, aproximadamente a partir de los 80 vecinos.

Estos resultados nos llevaron a preguntarnos si había algún k cerca de 110 que optimizara aún más el algoritmo utilizando PCA. Por esta razón hicimos una nueva corrida, focalizando los valores de α y de k en los valores para los que dio mejor accuracy.

3.2.1. KNN y PCA: valores óptimos

Corrimos el mismo experimento que el inicial, pero con valores de k entre 100 y 200 avanzando de a 3, y con valores de α múltiplos de 20 entre 40 y 100. Los resultados obtenidos se encuentran en el siguiente gráfico:

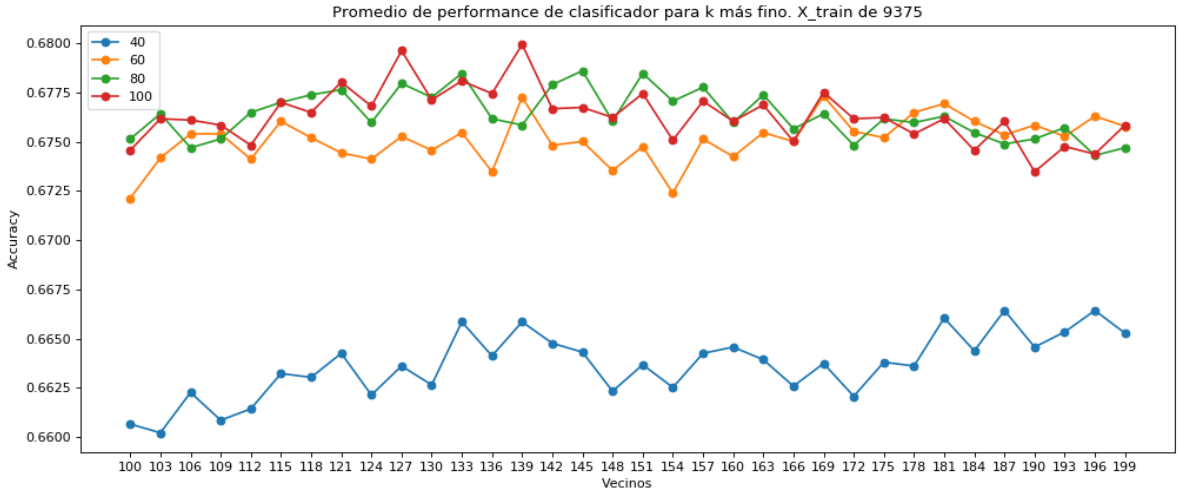

Figura 3: Resultados del experimento sobre cantidad de componentes principales combinado con cantidad de vecinos focalizado. Cada línea del gráfico representa el valor de α que muestra el recuadro de la esquina.

Estudiando los resultados de este nuevo experimento podemos decir que el algoritmo se optimiza con $\alpha = 100$, para ciertos valores de k como 115, 136 o 139, aunque la accuracy obtenida con los valores mencionados no difiere demasiado con la obtenida con ciertos k para $\alpha = 80$. Vemos entonces que los k para los cuales el resultado tiene una mejor calidad están entre los que habíamos obtenido en el experimento de KNN sin PCA. Con esto en mente, queremos indagar todavía más en estos valores que parecen hacer nuestro clasificador más eficiente. Por eso comparamos los últimos resultados con una nueva corrida para $\alpha = 120$ para ver si un mayor α nos llevará a aún mejores resultados en este rango de vecinos. Los resultados a los que llegamos fueron:

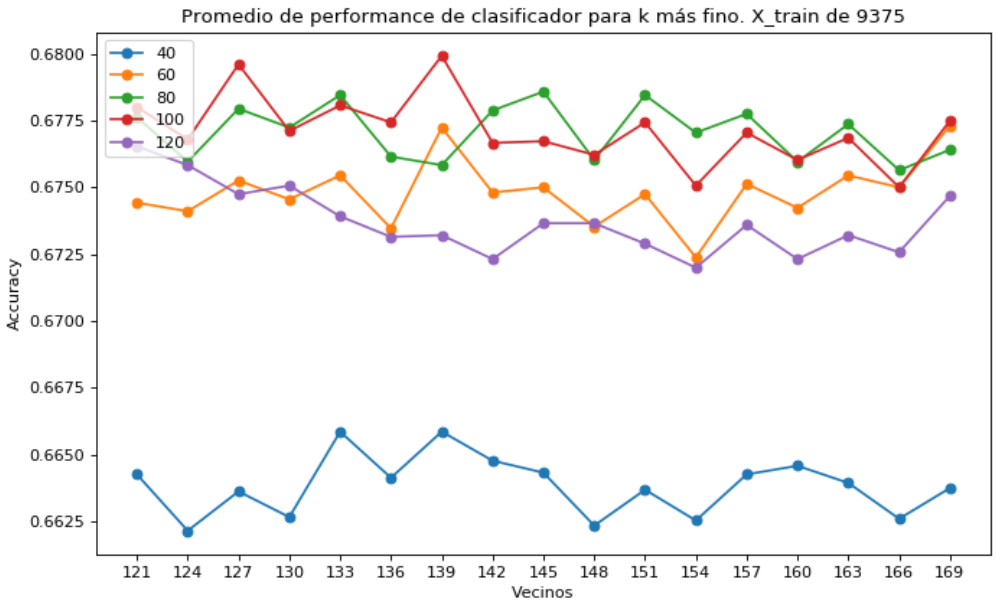

Figura 4: Resultados del experimento sobre cantidad de componentes principales combinado con cantidad de vecinos focalizado. Cada línea del gráfico representa el valor de α que muestra el recuadro de la esquina.

Podemos ver entonces que a $\alpha = 120$ le va peor que a $\alpha = 100$ y que en $k = 139$ se obtiene la máxima precisión que alcanza nuestro algoritmo de clasificación. Otra observación interesante es que ese k estaba entre los que permitían una mayor precisión en el experimento de KNN sin utilizar PCA. Finalmente, podemos ver que a medida que aumentamos el k aumenta disminuye la precisión lo que confirma nuestra hipótesis de que la cantidad de vecinos considerada debe ser moderada, ya que al ser demasiado grande entran en juego instancias cada vez más alejadas de nuestra reseña bajo análisis.

Por otro lado, observamos además que la precisión fue globalmente mayor para los α mayores en este experimento, por lo que en la siguiente subsección presentaremos uno nuevo (experimento 3.2.2) con α más grandes para un gran rango de vecinos, con el objetivo de observar hasta qué punto se sigue dando esta mejora.

3.2.2. KNN y PCA: alphas más grandes

Como ya mencionamos, nos interesa ver hasta qué punto podemos mejorar nuestro algoritmo aumentando la cantidad de valores principales tomados. Buscamos estudiar si hay un punto en el que aumentar el α comienza a generar resultados menos confiables, y cuál es. Creemos que encontraremos un punto en el que aumentar el α implique resultados menos confiables y por ende menor eficiencia. En este contexto, el experimento fue correr KNN y PCA con valores de α múltiplos de 10 entre 100 y 200 y para k también múltiplos de 10 pero entre 30 y 200. Por razones de simplicidad del gráfico y porque al fin de este experimento no aportaban mayor información, en la figura 5 se muestran los α de 20 en 20. Los resultados fueron los siguientes:

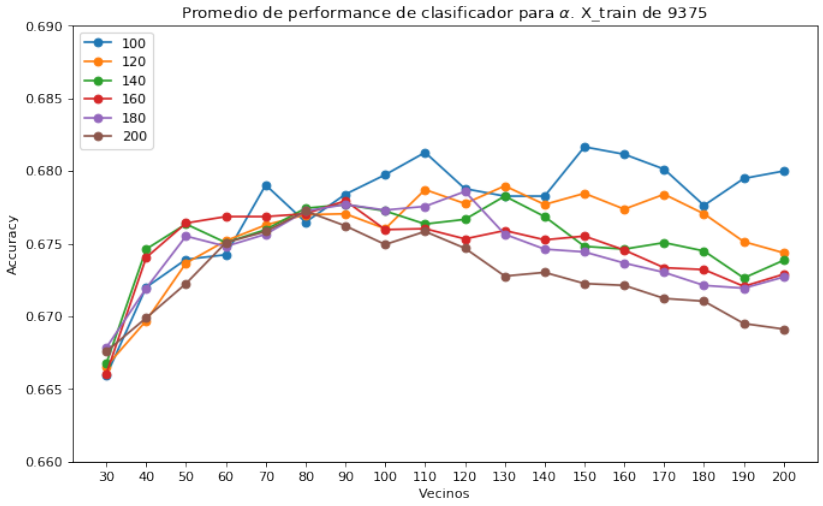

Figura 5: Resultados del experimento sobre cantidad de componentes principales combinado con cantidad de vecinos para mayores valores de α. Cada línea del gráfico representa el valor de α que muestra el recuadro de la esquina.

Podemos observar que el mejor resultado global se alcanzó con $\alpha = 100$, seguido por $\alpha = 120$. Es interesante notar que la mejoría de eficiencia entre 100 y 140 es ligeramente mayor que aquella entre 160, 180 y 200, que se comportan de manera muy similar, como podemos observar en este recorte del gráfico (figura 6):

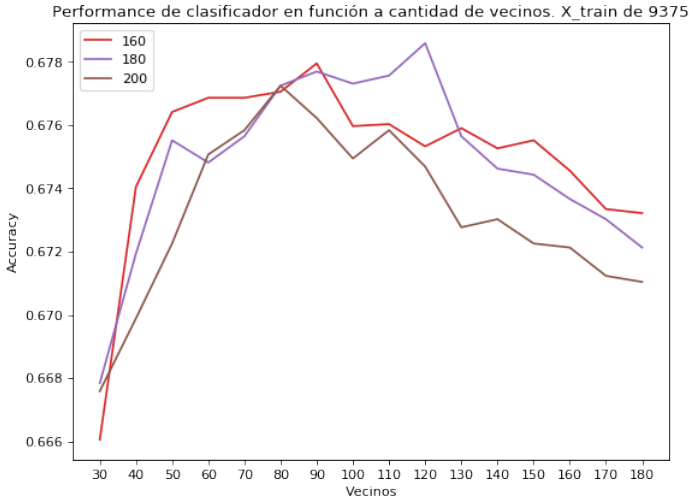

Figura 6: Recorte de los resultados de la figura 5. Cada línea del gráfico representa el valor de α que muestra el recuadro de la esquina.

Esto implicaría que los componentes posteriores a 100 ya no son muy relevantes a la clasificación de resultados, por lo que la mejoría en α aún más grandes parecería ser despreciable. En conclusión, podemos afirmar que se confirma nuestra creencia inicial, ya que aparentemente la precisión empeora con con α mayor a 120.

3.3. KNN: relación entre cantidad de vecinos y tamaño del conjunto de entrenamiento

Luego de haber examinado los cambios en la calidad de los resultados con diferentes variaciones de la cantidad de vecinos elegidos y con diferentes tamaños del conjunto de entrenamiento, vamos a indagar ahora sobre la mejor combinación entre los dos parámetros. Es decir que analizaremos para los tamaños del dataset de entrenamiento, cuál es el k que mejora la precisión del algoritmo sin utilizar PCA.

Hipótesis

Con la misma idea de balancear los parámetros y la combinación de éstos para optimizar la exactitud de nuestras predicciones planteamos la siguiente hipótesis. Como mencionamos anteriormente, el conjunto de entrenamiento debe representar una proporción considerable de la base de datos total pero también debe dejar suficientes instancias para conformar el conjunto de test y de esta forma analizar de forma correcta la precisión de nuestro algoritmo. De la misma manera, la cantidad de vecinos debe representar un balance para poder quedarnos con una cantidad suficiente de los mismos para que los tomados pertenezcan en mayor medida a la clase real de la instancia que se desea predecir.

Pero ¿cuál es la relación entre estos dos parámetros? Si el tamaño de la base de entrenamiento es chico y tomamos una cantidad de vecinos que sean una proporción grande de esta base, entonces es más probable tomar instancias que, a pesar de ser las más cercanas a la nueva, no sean verdaderamente cercanas a esta y por lo tanto que pertenezcan a una clase diferente. Si mantenemos esa misma cantidad de vecinos y agrandamos el conjunto, esperamos que se mitigue este efecto ya que la cantidad de vecinos tomada representaría una menor proporción del total.

Si este fenómeno se da como creemos, nos interesa ver cuál es la proporción óptima de la base total de entrenamiento que deberíamos tomar para obtener resultados precisos y no "ensuciarlos" con instancias que, a pesar de ser los vecinos próximos, no son lo suficientemente cercanas como para guiarnos por ellas.

Experimento

Como indicaremos en la siguiente figura (7), realizamos cuatro corridas diferentes en las que establecimos distintos tamaños del conjunto de entrenamiento: con 3125, 6225 y 9375 instancias, dejando las restantes del total para el conjunto de test. En cuanto a los parámetros principales, k va a variar entre 10 y 200, saltando de 10 en 10. En esta observación, α no va a participar ya que experimentamos con KNN sin PCA.

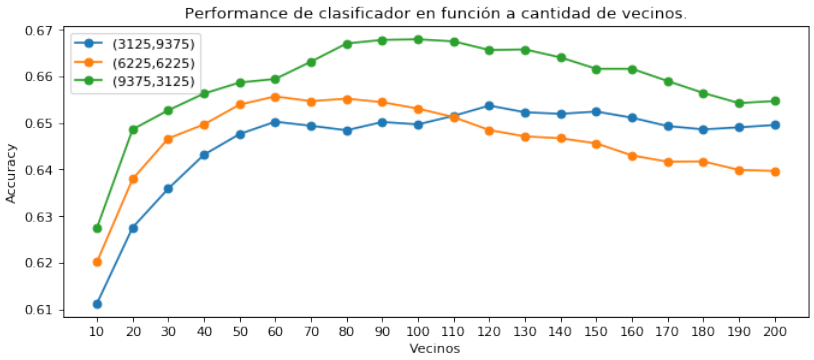

Figura 7: Resultados de la precisión en base a los tres tamaños del conjunto de entrenamiento y en función de distintos k vecinos. Cada una de las líneas del gráfico representa la cantidad de instancias del conjunto de entrenamiento que muestra el recuadro de la esquina, donde la primera posición indica el tamaño del conjunto de entrenamiento y la segunda, el del conjunto de test.

Resultados y conclusiones

Si llamamos puntos de inflexión a la cantidad de vecinos k en la cual la precisión alcanza un máximo local y a partir de la cual la precisión no se vuelve a superar, podemos observar que para la base de entrenamiento de 3125 instancias, este punto, aunque menos marcado que para las otras dos, se encuentra en $k = 120$, que representa un 3,84 % del conjunto total de entrenamiento; para la base de 6225 instancias, $k = 60$ representa un 0,96 % del total y para la de 9375 instancias, $k = 100$ representa un 1,07 %. Podríamos pensar entonces en una tendencia a que la cantidad de vecinos óptima para predecir correctamente la valorización de una reseña sea un 1,96 % de la base de entrenamiento total, por lo que parecería estar dándose el fenómeno que describimos en la hipótesis, ya que luego del punto de inflexión la precisión empeora al agregar nuevas instancias.

3.3.1. PCA: relación entre cantidad de componentes principales y tamaño del conjunto de entrenamiento

Otro análisis que realizamos fue estudiar cómo variaba el α según diferentes tamaños de la muestra de entrenamiento. Es decir que vamos a buscar el α que optimiza el algoritmo al variar tanto la cantidad de vecinos como el tamaño del conjunto de instancias de entrenamiento.

Hipótesis

Nuestra hipótesis para este experimento es que al tomar una muestra de entrenamiento de gran tamaño, el α podrá ser menor ya que al estar bien entrenado el clasificador, tomando una cantidad de componentes principales más chica igualmente se obtendrá una buena precisión. Dicho de otro modo, si tenemos un α relativamente chico, podríamos perder información al no tener en cuenta ciertos componentes relevantes, pero esta falta podría compensarse teniendo una buena base de entrenamiento.

Experimento

Para realizar este análisis corrimos el algoritmo con k múltiplo de 5 en el rango de 10 a 200, con α de 10 en 10 hasta 200, y con un dataset de entrenamiento de 1/4, 1/2 y 3/4 del total de las instancias. De

esta manera, obtuvimos estos resultados (notar que el resultado de este experimento con un conjunto de entrenamiento de tamaño 3/4 del conjunto de todas las instancias es la figura 2):

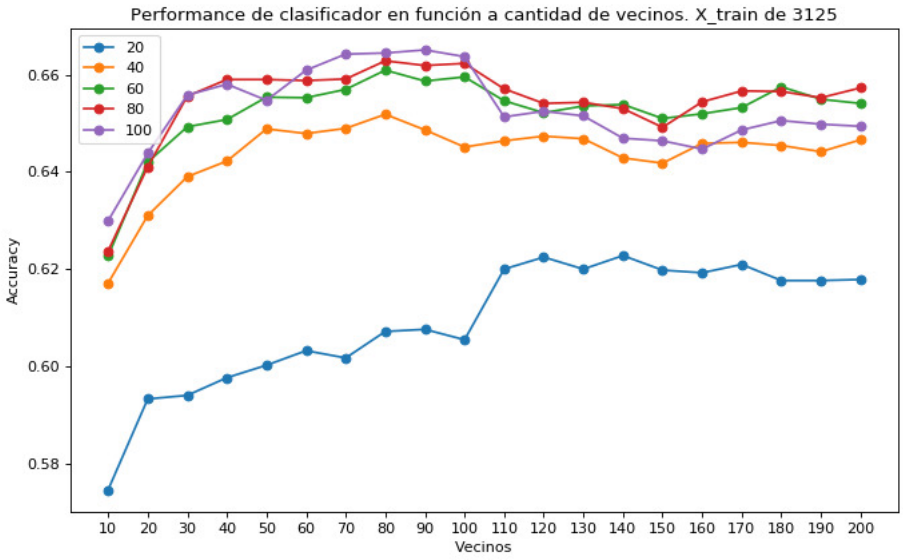

Figura 8: Resultados de la precisión tomando un conjunto de entrenamiento que representa 1/4 del total de las instancias en función de distintos α y k vecinos. Cada una de las líneas del gráfico representa el α que muestra el recuadro de la esquina.

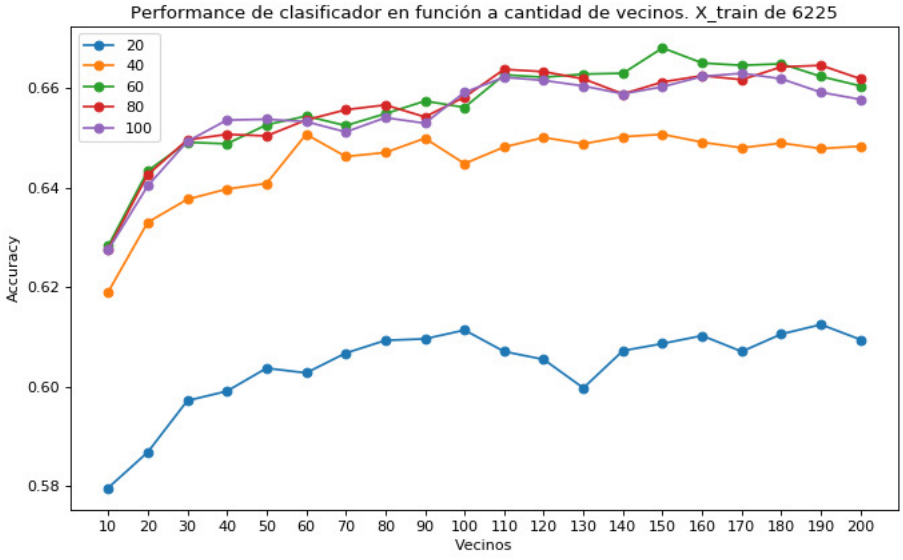

Figura 9: Resultados de la precisión tomando un conjunto de entrenamiento que representa 1/2 del total de las instancias en función de distintos α y k vecinos. Cada una de las líneas del gráfico representa el α que muestra el recuadro de la esquina.

Resultados y conclusiones

Podemos observar comparando los tres gráficos que nuestra hipótesis parece confirmarse. Podemos observar en el gráfico 8 que con la base de menor tamaño, el α que mejor precisión alcanza es el mayor (100), pero en la base que representa 1/2 del conjunto de todas las insrancias, este máximo se obtiene con $\alpha = 60$. Además, como se observa en el gráfico de la figura 9, las precisiones obtenidas con $\alpha = 60$, 80 y 100 es muy parecida. Finalmente, con la mayor base, podemos ver en la figura 2 que se obtuvieron mejores resultados con $\alpha = 60$ y 80 que con 100. Es decir, que con un clasificador mejor entrenado, podemos permitirnos una menor cantidad de componentes principales considerados y seguir obteniendo una buena accuracy.

3.4. Umbrales del modelo de bolsa de palabras

Para resolver el problema del amplio tamaño del vocabulario son necesarios umbrales para filtrar las palabras de éste en el modelo de bolsa de palabras. Buscamos determinar los umbrales que optimizan la predicción de una reseña. CountVectorizer representa la implementación del modelo de bolsa de palabras, toma tres parámetros diferentes. El primer parámetro es max_df es un float en el rango entre 0.0 y 1.0, luego al construir el vocabulario el modelo ignora las palabras cuya frecuencia en la base de reseñas es mayor al porcentaje pasado. El segundo parámetro es min_df, funciona con la misma lógica que max_df pero ignorando las palabras cuya frecuencia es menor al parámetro. Por último, max_features determina cuáles palabras serán representadas en el modelo, tomando únicamente la cantidad de palabras pasadas por parámetro eligiendo aquellas cuya frecuencia es mayor. Haremos variar los dos primeros parámetros, min_df y max_df, y analizaremos cómo afecta esto a la accuracy de las predicciones.

Hipótesis

Suponemos que al excluir las palabras cuya frecuencia es o muy alta o muy baja la predicción se vuelve más acertada ya que estas palabras no aportan información relevante para clasificar una reseña. Sin embargo, si recortamos demasiado las palabras que son representadas en el modelo, es decir si los umbrales dejan afuera una gran proporción de las palabras por sus frecuencias, se corre el riesgo de perder información útil al clasificar. Entonces, para optimizar la accuracy es necesario encontrar un balance de los umbrales para filtrar las palabras; éstos deben ser lo suficientemente altos como para filtrar las palabras que no aportan información a la hora de clasificar pero sin excluir palabras de más.

Experimento

Para estudiar esto planteamos el siguiente experimento tomando un k del método de k vecinos más cercanos y un α de PCA arbitrarios e hicimos variar min_df y max_df de forma simultéanea. En el gráfico de la figura 10 las tuplas presentes en el eje x representan estos valores: tienen la forma (min_df, max_df). El min_df va aumentando dejando de lado las palabras cuya frecuencia es menor al valor de la primera componente de la tupla y max_df va disminuyendo excluyendo así las palabras cuya frecuencia es mayor al valor de la segunda componente. En el eje y se refleja la accuracy de la predicción del método de k vecinos más cercanos luego de aplicar PCA con los distintos umbrales. De esta forma la cantidad de palabras excluídas por el modelo de bolsa de palabras crece en cada iteración del experimento planteado. Tomando $k = 100$ y $\alpha = 90$, aleatorizando la base de instancias, asignando una proporción de 3/4 de ésta para entrenamiento y promediando cinco corridas del experimento obtuvimos los siguientes resultados:

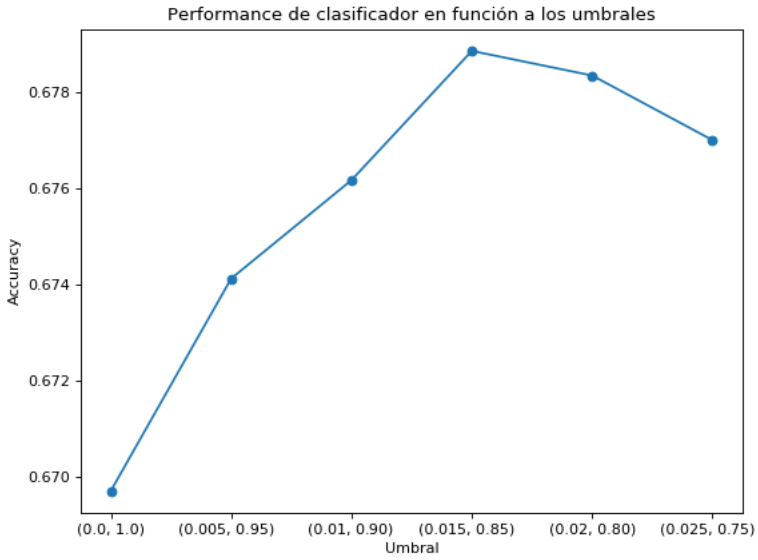

Figura 10: Resultados del experimento sobre los umbrales del filtro de palabras con k=100 y $\alpha = 90$. Las tuplas presentes en el eje x representan los valores de min_df y max_df respectivamente.

Resultados y conclusiones

Podemos observar en el gráfico de la figura 10 que la precisión de los resultados obtenidos aumenta a medida que se filtra una mayor cantidad de palabras tanto por su alta o baja frecuencia. El crecimiento de la accuracy se produce hasta alcanzar su máximo con los umbrales min_df = 0.015 y max_df = 0.85. Luego, al excluir más palabras del modelo, se comienza a perder precisión en los resultados. Esto coincide con nuestra hipótesis de obtener un balance al fijar los umbrales con el objetivo de optimizar la precisión de las clasificaciones. Este balance parece obtenerse en los valores mencionados, por lo que filtrar una mayor cantidad de palabras supone una pérdida de información que impacta negativamente en la accuracy de la predicción.

3.5. Método de la potencia: criterio de parada

Como explicamos anteriormente, el método de análisis de componentes principales utiliza el método de la potencia. El mismo es un procedimiento iterativo que converge al autovalor principal y su autovector asociado, por lo que cuenta con un criterio de parada. Es decir, que para saber en qué momento ya no tiene sentido seguir iterando sobre el vector y el autovalor que se están calculando, según la perspectiva que se desea tomar. En nuestra implementación decidimos frenar el algoritmo una vez que los valores que se obtienen en una iteración son muy similares a los que ya se tenían. Para ello comparamos el vector calculado con el anterior en norma, saliendo del ciclo cuando la norma de la diferencia es menor a un ϵ. Esto lleva a la duda de cuál es la norma vectorial que optimiza este método, que es lo que estudiaremos en este experimento. Analizaremos cómo se modifica el desempeño temporal del método de la potencia así como la precisión obtenida por el método de predicción que lo utiliza al variar el criterio de parada.

Hipótesis

Como un primer paso hacia pensar qué norma vectorial será más eficiente, pensamos en la interpretación de aplicar cada norma a la resta de dos vectores. La norma 1 realiza la suma de los módulos

coordenada a coordenada, por lo que al realizar la norma 1 a una resta entre vectores, el resultado es la suma de la resta de cada coordenada en módulo. Esta norma se minimizará entonces cuando los dos vectores se parezcan coordenada a coordenada. La norma ∞ es la máxima coordenada en módulo, por lo que se minimizará cuando la diferencia entre las coordenadas que menos se parecen, esto es, la diferencia máxima entre dos coordenadas, sea mínima. Observando con detenimiento las interpretaciones podemos concluir que ambas normas se minimizan en las mismas condiciones. Por otro lado, la norma 2, que es la raíz de la suma de los cuadrados de cada coordenada, también se minimizará cuando la diferencia coordenada a coordenada sea menor. La diferencia entre ésta y las otras mencionadas es que es siempre menor a la norma 1.[3]

Dicho esto, respecto del tiempo que tardará el algoritmo en ejecutarse creemos que la norma 2, al ser siempre menor a la norma 1, va a llegar antes al ϵ, por lo que el algoritmo va a tardar menos. Un criterio de parada más laxo implica que se realizan una menor cantidad de ciclos, por lo tanto el algoritmo tarda menos, pero también se obtiene un vector menos cercano al vector al cual converge el método. Luego, la matriz que se utiliza para aplicar el cambio de base en el método de PCA va a ir arrastrando un error lo que va a conducir a una menor precisión al clasificar una reseña utilizando este método. Por esta razón creemos que va a ser mejor que la norma 2, en cuanto a la accuracy de la predicción, la norma 1. Suponemos que la norma infinito se va a comportar como la norma 1 y que por lo tanto obtendremos resultados similares en cuanto al tiempo de ejecución y a la precisión de los resultados.

Experimento

Para estudiar la norma óptima para el clasificador, tomamos el código implementado del método de la potencia y utilizamos las normas ya implementadas que presenta Eigen para decidir cuándo salir del ciclo. Es decir, cambiamos de norma, recompilamos, y corrimos el algoritmo con un k y un α arbitrarios ($k = 130$ y $\alpha = 140$). Los resultados obtenidos fueron:

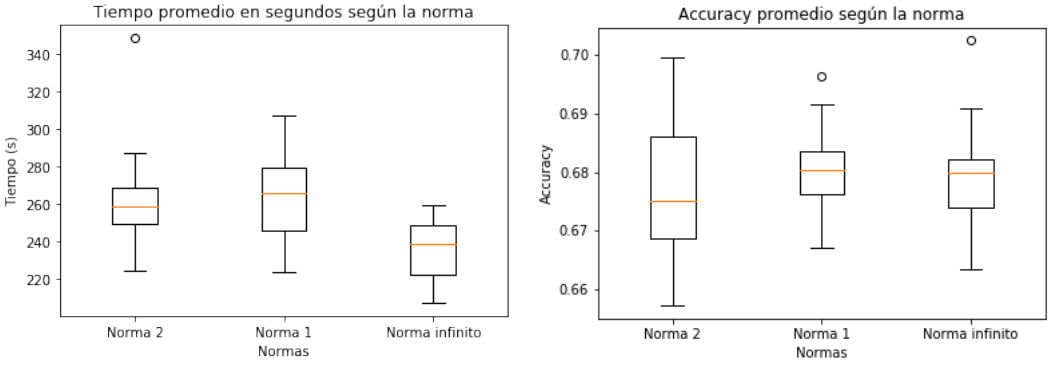

Figura 11: Resultados del tiempo de ejecución medido en segundos (izquierda) y la precisión (derecha) de la clasificación con las distintas normas.

Resultados y conclusiones

Podemos observar en la figura 11 que el método de clasificación utilizando como criterio de parada del método de la potencia la norma ∞ toma en promedio un tiempo de ejecución menor al que utiliza la norma 1 o la norma 2. Esto se contradice con nuestra hipótesis; habíamos propuesto a la norma 2 como el criterio que optimizaría el tiempo de ejecución. Una causa de esto puede ser el considerable costo temporal que implica aplicar norma 2 en cada iteración del método de la potencia ya que esta

[3] http://www.ehu.eus/izaballa/Ana_Matr/Apuntes/lec2.pdf

norma realiza operaciones costosas entre floats. Luego, si bien comparar los vectores para verificar si el método convergió con norma 2 implicaría salir antes del ciclo realizando así menos iteraciones, cada iteración tiene un mayor costo temporal. La norma 1, a diferencia de la norma ∞, realiza sumas entre los módulos de las coordenadas del vector al cual se la aplica. Esto podría explicar entonces la inesperada diferencia en el desempeño temporal entre la norma 1 y la norma ∞. Sin embargo, esta deducción nos genera dudas ya que resulta sorprendente que tanta diferencia en el tiempo de ejecución se deba únicamente al hecho de realizar sumas en lugar de comparaciones simples en cada iteración. Podríamos como trabajo a futuro estudiar más en detalle las diferencias de aplicación de las dos normas con el objetivo de encontrar más causas de esta disparidad.

En cuanto a la precisión de los resultados, la norma 1 y la norma ∞ presentan resultados similares. Esto coincide con nuestra hipótesis acerca de la semejanza de las condiciones para las cuales se minimizan ambas normas y por lo tanto, la homogeneidad de sus resultados en cuanto a accuracy. Sin embargo podemos ver que la mayor precisión se obtuvo con la norma ∞, aunque en promedio la norma 1 tuvo una mejor accuracy. En contraste con esta similitud, la norma 2 exhibe resultados menos precisos, lo que también se corresponde con nuestra suposición inicial; al ser menor que la norma 1 y en consecuencia cumplir con el criterio de parada con una menor cantidad de iteraciones, el método de la potencia que utiliza la norma 2 obtiene valores menos precisos. En conclusión, el criterio de parada del método de la potencia que optimiza la clasificación en términos de precisión y de tiempo de ejecución es la norma ∞.

4. Conclusiones

A lo largo del trabajo utilizamos distintos métodos con el objetivo de estudiar la connotación positiva o negativa de las reseñas de la base de datos de IMDB. Para poder comparar distintas instancias, en nuestro caso reseñas, utilizamos el modelo de bolsa de palabras. Luego implementamos el procedimiento de los k vecinos más cercanos. Este método predice dada una instancia la clase de ésta en función de las clases de instancias pertenecientes al conjunto de entrenamiento cercanas. Las instancias de los conjuntos de entrenamiento suelen tener un tamaño considerable y contener información poco significativa o hasta contraproducente de tener en cuenta a la hora de efectuar una predicción. Por este motivo implementamos y aplicamos al procedimiento anterior el método de análisis de componentes principales. En la sección de experimentación, estudiamos distintos parámetros que inciden en la precisión de nuestra implementación para la clasificación de reseñas como positivas o negativas. Es decir, vimos de qué manera influyen diferentes variaciones de éstos en la calidad de los resultados obtenidos.

Constatamos que la precisión de los resultados utilizando el procedimiento de k vecinos está fuertemente relacionada con el k elegido. La cantidad de vecinos que optimiza la exactitud de las predicciones debe ser lo suficientemente grande como para considerar una cantidad de instancias cercanas que resulten significativas. Sin embargo, si la cantidad de vecinos es demasiado grande en relación al tamaño del conjunto de entrenamiento, podemos estar tomando instancias muy alejadas a la que queremos clasificar lo que podría conducir a un error en la predicción.

Con PCA, no solo confirmamos en los experimentos que este método reduce ampliamente la duración del entrenamiento del calificador (KNN), sino que también incrementó la precisión de este último. Además, los costos en tiempo de utilizar PCA para transformar el conjunto de entrenamiento, son despreciables comparados con todo lo ahorrado en la parte de analizar cada instancia de entrenamiento. Sin embargo, lo afirmado previamente no sería cierto si la cantidad de componentes principales no fuera la correcta. Si no tomamos los suficientes, estaríamos perdiendo información importante de cada instancia. Y tomar de más, implicaría un mayor tiempo dedicado a transformar el conjunto de entrenamiento, como vimos en la experimentación del trabajo.

Otro factor importante analizado en este informe fue el tamaño del conjunto de entrenamiento. Pudimos concluir que influye notablemente en la precisión del calificador. Encontramos un balance aproximadamente en una proporción de 3/4 de las instancias totales para entrenar y el otro 1/4 para testing. Si pasábamos a darle una proporción mayor al conjunto de entrenamiento, consideramos que el de testing perdía representatividad. Por otro lado, el conjunto de testing no puede perder tanto tamaño ya que podrían no tener sentido los resultados.

Analizamos también qué combinación de los parámetros involucrados por PCA, KNN y el tamaño del conjunto de entrenamiento generan una mayor precisión en los resultados en nuestro caso particular de estudio. Luego de experimentar con esta cuestión, constatamos que los mejores resultados se obtienen al tomar una proporción del conjunto de entrenamiento que represente aproximadamente 3/4 de la base de datos, tomando entre las primeras 60 y 80 componentes principales y considerando entre los primeros 80 y 190 vecinos, que son aproximadamente entre un 1 y un 2 % de la base de entrenamiento total.

Los umbrales que filtran palabras en el modelo de bolsa de palabras juegan igualmente un rol importante; al filtrar más palabras el tiempo de cómputo de la clasificación mejora y hasta un cierto punto también lo hace la precisión de los resultados. Luego de experimentar concluimos que los valores de los umbrales que resultan óptimos para nuestro conjunto de instancias son min_df = 0.015 y max_df = 0.985.

Por último, estudiamos el impacto en los tiempos de ejecución y la precisión de la clasificación al elegir diferentes criterios de parada para la iteración del método de la potencia. Comparando los resultados utilizando como criterio de parada la minimización de la diferencia entre vectores en norma 2, norma 1 y norma ∞ inferimos que la norma ∞ resulta óptima en términos temporales como en calidad de resultados obtenidos.

Un criterio que nos pareció de interés y sería interesante analizar en un trabajo posterior es utilizar un procedimiento que sirva para predecir la próxima palabra que aparecerá en un texto teniendo en cuenta lo que ya se leyó. Esto podría servir para "arreglar" los posibles errores que surgen de utilizar la bolsa de palabras al no considerar el orden de las palabras de un texto. Con esto nos referimos a que dos reseñas con palabras similares pueden tener connotaciones diferentes al utilizarlas en un orden particular, pero el algoritmo no ve esto por lo que las tomaría como parte de la misma clase. Un ejemplo de esto serían dos reseñas como: "Me gustó la película porque no tenía..." y "No me gustó la película porque tenía...".