DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

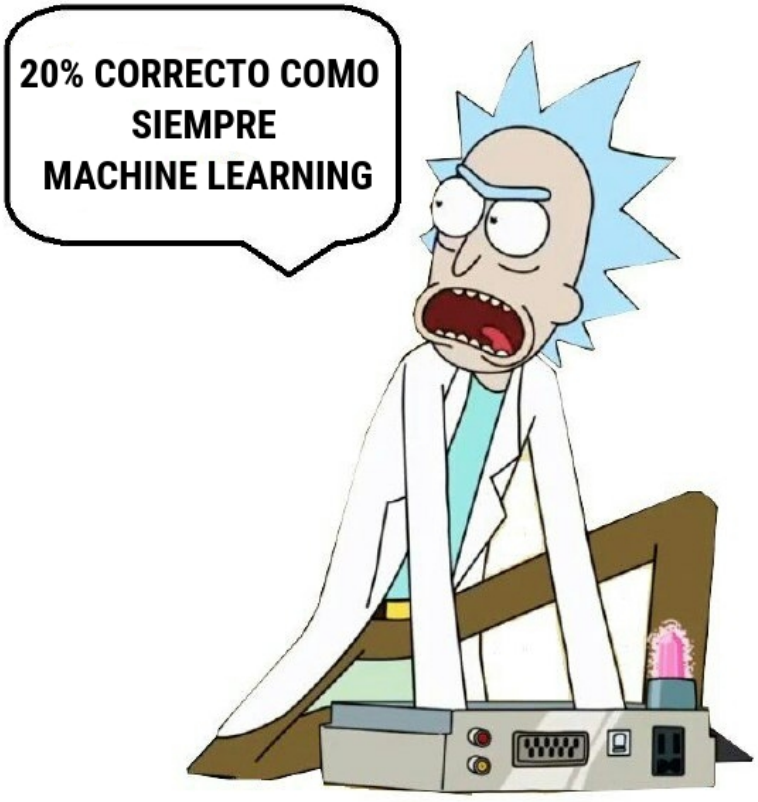
Trabajo Práctico 2

Análisis de Sentimiento

Métodos Numéricos

Primer Cuatrimestre de 2019

Integrante	LU	Correo electrónico
Alvarez Mon, Alicia	224/15	aliciaysuerte@gmail.com
Ansaldi, Nicolas	128/14	nansaldi611@gmail.com
Castro, Luis	422/14	castroluis1694@gmail.com
Suárez, Romina	182/14	romi_de_munro@hotmail.com



1. Resumen

En este trabajo práctico se nos pide entrenar un algoritmo, usando un dataset de reseñas de películas, para clasificar criticas en positivas o negativas. Para esto utilizamos el método de knn y, luego buscamos optimizarlo usando pca (análisis de componentes principales).

Luego tomamos el dataset y lo dividimos en un conjunto de entrenamiento y uno de testeo. Usando el conjunto de entrenamiento, nos quedamos con un subconjunto de palabras relevantes (Bag of Words) que usaremos para caracterizar las críticas dentro de nuestro modelo. Usando este vocabulario y knn clasificamos las críticas del conjunto de testeo. Luego tomamos esta caracterización y nos quedamos con las componentes relevantes usando pca, esto lo comparamos con la versión anterior y experimentamos para definir las ventajas y desventajas de las mismas.

Luego buscamos la mejor combinación de los parámetros para filtrar el vocabulario de knn y pca; para medir esto usamos la cantidad de aciertos sobre el total de las críticas dentro del conjunto de testeo y analizamos los errores de cada algoritmo.

2. Palabras claves

Vocabulary, Critica, KNN, PCA.

Índice

3. Introducción

En este trabajo práctico se nos pide analizar críticas a películas, las cuales fueron obtenidas de la pagina IMDB, clasificándolas en positivas o negativas. Para esto construimos nuestro modelo a partir de un conjunto de críticas etiquetadas correctamente como positivas o negativas.

IMDB o Internet Movie DataBase es una base de datos en línea que almacena información relacionada con películas, como ser los actores que participan, los directores, o las reviews de los fans, etc. En estas últimos nos enfocaremos en este trabajo. El ideal del mismo es poder clasificar una review o critica de una película en positiva o negativa, para así realizar un análisis de sentimiento de la misma.

Este último, sencillamente, es la forma de procesar computacionalmente un texto para identificar y extraer información, en este caso, si le gustó o no al usuario la película, para poder utilizar estos datos en estudios estadísticos.

Nuestro objetivo es conseguir un algoritmo que prediga la opinión de la crítica con un buen grado de precisión. Para poder construirlo, utilizamos un conjunto de entrenamiento determinado, ya etiquetado, y un conjunto de test también etiquetado para comprobar nuestra precisión. Además de esto, experimentamos para encontrar la mejor combinación de parámetros para maximizar la precisión de nuestro algoritmo, además intentaremos ver cuanto influyen el conjunto de críticas en si, ya sea el conjunto de entrenamiento, como el de testeo.

4. Desarrollo

En nuestro algoritmo nosotros queremos clasificar de manera eficaz si una critica es buena o mala. Para poder cumplir con esta función, nuestro programa tiene que pasar por una etapa de entrenamiento en donde toma una cierta cantidad de criticas y sus etiquetas. Luego de procesarlas, tomará una nueva critica e intentará diferenciar su positividad en base a las criticas que le pasamos previamente, es decir, de las cuales aprendió.

Cabe aclarar que tomamos igual cantidad de criticas positivas y negativas, ya que tener más de unas que de otras puede hacer que el sistema funcione de forma incorrecta. Esto significa que el algoritmo no va a tener las suficientes criticas como para comparar la nueva, así que va a tender a decir que pertenece a la etiqueta que predomine.

Para poder comparar las críticas, estas tuvieron que sufrir transformaciones que nos permitan codificar la entrada en una estructura más fácil de comparar algoritmicamente. Para ello, utilizamos las funciones que nos provee la librería sklearn, y con esta, armamos un bow(bag of words). Este esta encargado de tomar todas las palabras de las criticas y filtrarlas bajo ciertos criterios elegidos por nosotros (una palabra es relevante dependiendo de su porcentaje de apariciones con respecto a otras criticas), dándonos así una matriz donde en la fila j, posición i están las veces que apareció la palabra i en la critica j.

Luego, aplicamos el algoritmo knn para resolver la problemática de como clasificar una critica. Para ello, buscamos las k criticas más parecidas en nuestro sistema con respecto a la critica que estamos analizando actualmente y dependiendo de la proporción de negativas y positivas de las k vecinas, tomamos el valor de la mayoría o "la moda"(positivo en caso de empate).

4.1. Técnicas a utilizar

4.1.1. K-Nearest Neighbors o KNN

Para comparar las muestras, primero convertimos cada crítica en un vector de largo fijo. Para hacer eso utilizamos el modelo bag of words: Construimos un vocabulario con las palabras de cada crítica y luego creamos un vector con las apariciones de las mismas. Como el vocabulario de todas las criticas es muy extenso se filtran aquellas que son más usadas o extremadamente raras, ya que estas no aportan información relevante para determinar el sentimiento de la crítica.

Luego de que tenemos nuestros datos tanto de entrenamiento como de testing organizados en vectores, vamos a utilizar la técnica de Knn para determinar la etiqueta de la nueva crítica. En este punto, nosotros consideramos 3 métodos posibles para comparar una critica con otra:

- Método 0: En este método utilizamos la lógica matemática: 2 vectores v1,v2 son parecidos mientras más chica sea la cuenta $||v1 - v2||_2$.
- Método 1: Planteamos una lógica más lingüística. Dos críticas se parecen si utilizan las mismas palabras, para este método elegimos determinar al más cercano como al que tenga más palabras en común(contadas una única vez).
- Método 2: Siguiendo con la misma idea del método anterior, 2 críticas se parecen si comparten las mismas palabras, pero ¿y si contamos la cantidad de veces que usan las mismas palabras?. Esto capaz no tiene un razonamiento tan intuitivo pero en un caso general puede mejorar nuestro método anterior, cosa que vamos a ver en la experimentación.

Para resolver este problema nosotros implementamos la clase KKNNClassifier, que posee los métodos fit y predict. Fit toma la matriz de entrenamiento como base para predecir las entradas, y predict, dada una matriz donde cada fila es una instancia de testeo, devuelve un vector con las predicciones de cada instancia.

En el algoritmo de predict acumulamos en el vector el resultado de hacer predict_row a cada fila de la matriz de entrada. Este método privado es el que compara la distancia euclídea(método 0) entre el vector entrada y los de la base de entrenamiento y hace la votación entre los k más cercanos.

En un principio habíamos hecho el algoritmo con matrices ralas, pero las operaciones de resta directa, mientras iteramos en los elementos, son mucho más costosas en matrices ralas, por lo que usar las matrices densas nos redujo mucho el tiempo de ejecución del cálculo de distancias euclídeas y del algoritmo en sí. Además, dado que sólo nos interesan los primeros k valores más cercanos, usamos un partial sort para ordenar solo lo que requerimos y así mejoramos un poco los tiempos de computo.

4.1.2. PCA o Análisis de componentes principales

Nuestro algoritmo de knn, en uno de los metodos, mide la distancia euclídea entre los vectores de testeo y los de nuestra base de entrenamiento. Dado que nuestros vocabularios son extensos, la complejidad de realizar este proceso puede ser muy costosa.

Para optimizar esto, utilizamos PCA o el análisis de componentes principales. Este es un método de reducción de dimensiones que busca transformar y reducir la matriz de críticas para solo mantener y representar los valores significativos de la misma.

En el procedimiento PCA lo que hacemos es trabajar con la matriz de covarianza de la matriz original. La covarianza mide la relación entre las varianzas (la dispersión) de las variables y mayor covarianza suele implicar mayor relación.

Esta matriz de covarianza M se genera como:

$$M = \frac{A^T * A}{n-1} \tag{1}$$

Donde A es una matriz generada restándole μ a cada elemento de la matriz de entrada $X \in R^{nxm}$, con n la cantidad de críticas, m la cantidad de palabras y μ la media de la suma de las filas de la matriz X.

Esta matriz de covarianza M tiene en la diagonal a las varianzas de las instancias y en el resto de los elementos a la covarianza entre el elemento fila y columna de la matriz X. Por lo tanto, es una matriz simétrica (y semi-definida positiva), con base de autovectores orto-normales (li).

Luego como tenemos una base de autovectores podemos diagonalizar la matriz, entonces nos quedará que $M = V \cdot M \cdot V^T$, donde V tiene en sus columnas los autovectores. Luego, hacemos el cambio de base haciendo $X' = X \cdot V$. Como V representa los valores principales, nos quedamos con un subconjunto de ellos que le dan más peso al algoritmo (Estos son los primeros, porque están asociados a los autovalores de mayor peso/más dominantes).

En nuestro algoritmo lo que hacemos es agregar el cambio de base a las matrices de entrenamiento y de testeo antes de pasarlas a knn. La matriz cambio de base utilizada es conseguida al hacer el fit de la matriz de entrenamiento. Para encontrar los autovalores y sus respectivos autovectores de la matriz, utilizamos el método de la potencia. Este método nos devuelve el autovalor más grande (con su respectivo autovector), luego usamos deflación para conseguir una matriz semejante, a la cuál podemos obtenerle el siguiente autovalor, y así sucesivamente. Como conseguir todos es mucho trabajo computacional, tenemos un parámetro alfa de todos los autovalores que queremos conseguir. La matriz resultante pertenece a R^{nxalfa}

Por último, obtenida esta matriz cambio de base, usamos transform para cambiar las bases de nuestras matrices y finalmente las usamos en knn.

5. Experimentación

Primero, recordemos las métricas vistas en clase: Siendo tp: true positives, tn: true negatives, fp: false positives y fn: false negatives. Llamaremos:

$$\textbf{Accuracy} = \frac{tp+tn}{tp+tn+fp+fn} \qquad \textbf{Precision} = \frac{tp}{tp+fp} \qquad \textbf{Recall} = \frac{tp}{tp+fn} \qquad \textbf{F1-Score} = \frac{2*Precision*Recall}{Precision+Recall}$$

De esto podemos decir que accuracy es un promedio de la muestra y nos enseña la cantidad de correctos sobre totales, sin embargo, esta métrica puede ser engañosa ya que no revela cuantos falsos negativos o positivos se tuvieron. Además, nuestro accuracy suma tanto true positives como true negatives, ya que acertar criticas positivas como negativas es, en definitiva, acertar, para nuestro algoritmo.

La métrica precisión nos muestra la cantidad de elementos recuperados que son efectivamente relevantes, en otras palabras, la cantidad de falsos positivos que hubo en proporción a la cantidad de true positive.

La métrica de Recall nos habla de los elementos relevantes que si fueron recuperados, es decir, la cantidad de falsos negativos que hubo, esto, como antes, en proporción a la cantidad de true positive.

Por último, tenemos el F1-Score como la media armónica entre precision y recall, donde si la primera es 1, las otras 2 son buenas, y si es 0, ocurre lo contrario.

5.1. Knn: Variando K y sus métodos de comparación

Para realizar este experimento, los parámetros a tomar en cuenta fueron los siguientes: Las criticas para entrenar el algoritmo son las totales, 6225. Las criticas a testear fueron 6275. Variamos k entre 1 y 3000, y tomamos valores en múltiplos de 15. Los limites para el vocabulario fueron: 0.6 máximo, y 0.01 el mínimo y se utilizaron los 3 métodos de comparación dentro de Knn.

La **hipótesis** de este experimento es que a mayor cantidad de k, de ks muy pequeños a ks más grandes, se deberá tomar en cuenta una mayor cantidad de críticas para hacer el consenso, y por lo tanto el algoritmo sera más preciso. Sin embargo, a muy altos valores de K, pasará todo lo contrario, el consenso será entre más cantidad de criticas, que pueden ser ya muy diferentes a la que estábamos analizando, provocando que se erre en el acierto. Además, como segunda **hipótesis** tenemos que el método de comparación de vecinos afecta los resultados obtenidos, es decir, el análisis del algoritmo es gravemente afectado por la forma de comparar y tomar vecinos. Veamos la siguiente figura:

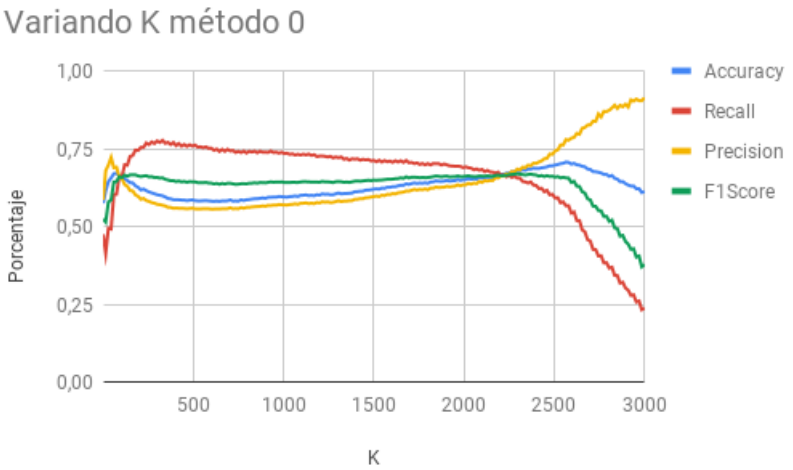


Figura 1: Variaciones de K en algoritmo de Knn sin PCA y método de comparación 0

Podemos ver que para valores pequeños de k, antes de 250, el recall es más bajo, es decir, hay mayor cantidad de false positives, comparando con k mayores. Pero luego, podemos ver que para mayores k, alrededor del 2500, aumenta el precisión y decae el recall. Esto significa que para estos valores de k, aumentan los false negative, pero decaen los false positives, traduciéndose en un mejor diagnostico de las criticas positivas, pero no de las negativas. Por lo tanto a valores muy bajos y muy altos de k, el algoritmo no da un buen análisis, como también se puede ver en el F1-score.

¿Cómo influye el método de comparación en este caso? Recordemos que para este gráfico estamos analizando el método 0, el cual compara vecinos por resta entre vectores, y mientras esta resta, en modulo tienda a 0, más se parecerán 2 vectores. Sin embargo, este método da peores resultados pues la norma de la resta prioriza las diferencias entre 2 vectores, y no la intersección entre ellos, como veremos en los siguientes gráficos.

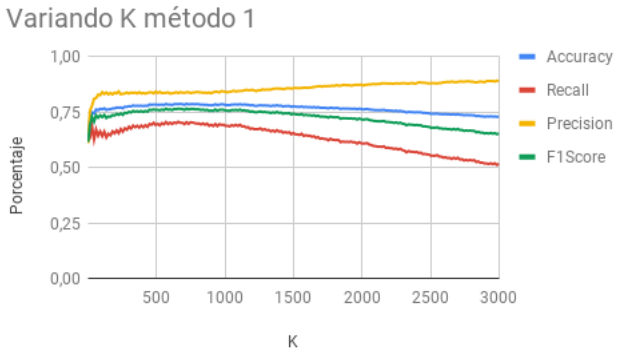


Figura 2: Variaciones de K con Método 1

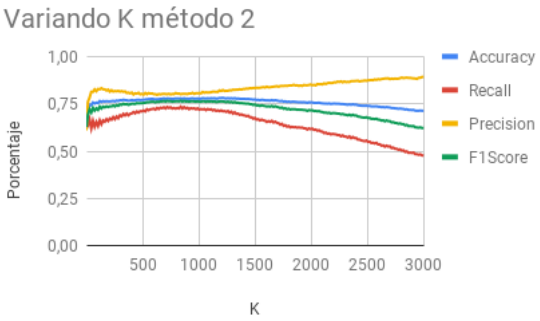


Figura 3: Variaciones de K con Método 2

El método de comparación 1 y 2 arrojó resultados casi idénticos y podemos ver que se mantiene la hipótesis de que a k muy bajos, las métricas no son las mejores y luego, para valores muy altos, las 3 empeoran. ¿Cómo afectan los métodos de comparación? ¿ Por qué son parecidos? Recordemos que el método 1 toma vecinos según los k vectores que comparten la mayor cantidad de palabras, y el 2do método, además de hacer esto, ordena estos k vecinos, por quien tenga mayor cantidad de repeticiones de estas palabras ya compartidas. De ahí que sus conjuntos de k vecinos sean parecidos pero no iguales. Sin embargo, vemos que para los dos métodos, los valores de k les afecta de la misma manera.

5.2. Knn: Variando la cantidad de criticas de entrenamiento

Para realizar este experimento, los parámetros a tomar en cuenta fueron los siguientes: Las criticas para entrenar el algoritmo variaron de 30 a 6225. Las criticas a testear fueron 6275. Mantuvimos k fijo en 6, un número arbitrario pero pequeño, para poder analizar las bases de entrenamiento incuso si eran pequeñas. Los limites para el vocabulario fueron: 0.1 máximo, y 0.01 el mínimo y se utilizo el método número 2 de comparación dentro de Knn.

La **hipótesis** de este experimento es que a mayor cantidades de criticas de entrenamiento el algoritmo sabrá analizar mejor nuevas criticas, y a la vez, tardará más tiempo en hacerlo:

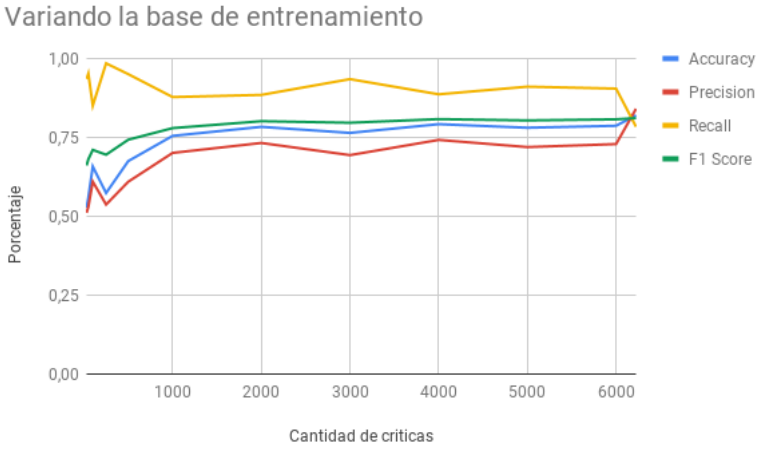


Figura 4: Variaciones de la base de entrenamiento

Como podemos ver, a medida que aumentan las criticas de 0 a 1000, el algoritmo tiene más vectores con los cuales comparar y acertar mejor en su diagnostico y para el resto de valores se mantiene casi constante. Esto ocurre mayormente así, porque k se mantiene fijo. Como vimos en el experimento anterior, si k varia, también varían las 3 métricas.

Figura 5: Variaciones de la base de entrenamiento en tiempo

Finalmente, al tener mayor cantidad de criticas para analizar y comparar, el algoritmo tarda más en dar un resultado, pues debe analizar todos los vectores que componen la matriz de entrenamiento para obtener los k vecinos.

5.3. Knn: Variando el umbral

Para realizar este experimento, los parámetros a tomar en cuenta fueron los siguientes: Las criticas para entrenar el algoritmo fueron 6225. Las criticas a testear fueron 6275. Mantuvimos k fijo en 100. Los limites para el vocabulario fueron variando de 0.000001 a 0.6 para el mínimo y de 0.001 a 0.999999 para el máximo, y se utilizo el método número 2 de comparación dentro de Knn.

Las **hipótesis** de este experimento son: Variar el umbral, varía el accuracy porque influye en el vocabulario y por lo tanto en la matriz de entrenamiento. Y tener el mayor umbral no se traduce en los mejores resultados, necesariamente, pero si en los mayores tiempos de corrida. Veamos:

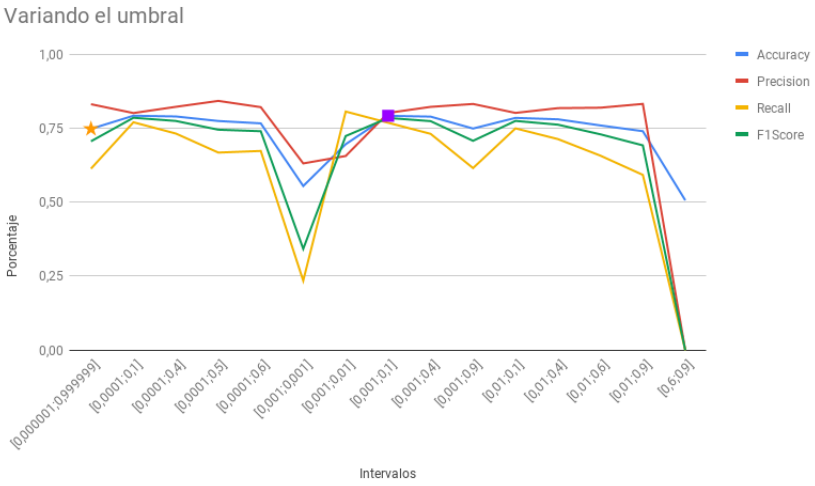


Figura 6: Variaciones del umbral

En la primera entrada de este gráfico (Estrella naranja) vemos los valores del mayor umbral tomado, con [0,000001;0,999999], obteniendo un accuracy de 0,7474, un precision de 0,8307 y un recall de 0,6131. Sin embargo, para el intervalo de [0,001;0,1] obtuvimos un mayor accuracy de 0,7915 y recall de 0,7681. Por lo tanto, podemos ver 2 cosas: modificar el umbral afecta la cantidad de palabras que forman parte del vocabulario y entonces, se modifica la base de entrenamiento con la cual Knn comparara los k vecinos.

Además, tomar el mayor umbral no nos da los mejores resultados necesariamente, ya que, al no filtrar las palabras más o menos usadas, 2 vectores pueden ser parecidos cuando lo que puede ser parecidos entre ellos, mayormente, sean estas palabras indeseadas que no aportan al concepto de crítica positiva o negativa.

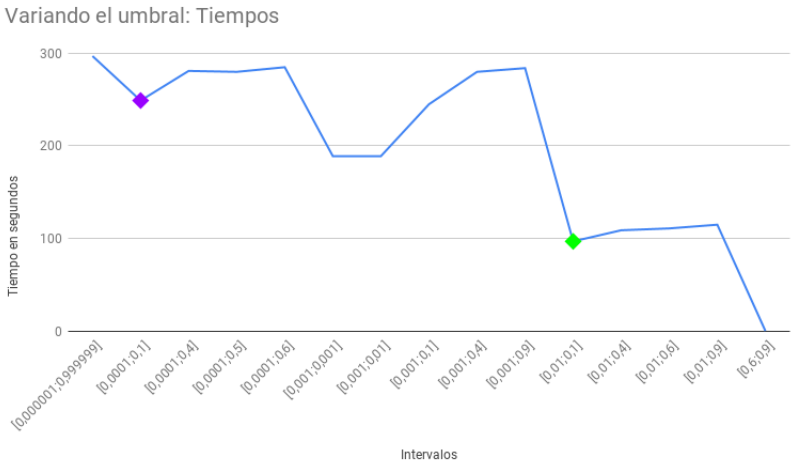


Figura 7: Variaciones del umbral en tiempo

Con el gráfico de tiempos podemos ver la primera hipótesis: al tener mayor umbral, el algoritmo tendrá un mayor vocabulario para generar los vectores de criticas de entrenamiento, y por lo tanto la matriz sera más grande, tomándole más tiempo a knn para comparar.

Podríamos discutir aquí el valor costo-beneficio entre tiempo de corrida y accuracy obtenida. Veamos por ejemplo, comparando el valor del intervalo [0,0001;0,1](diamante violeta), tenemos un accuracy de 0,7918, tomándonos un tiempo de 249 segundos, mientras que para el intervalo [0,01;0,1](diamante verde), tenemos un accuracy de 0,7845, tomándose un tiempo de 97 segundos. Es decir que en un poco más del doble del tiempo, conseguimos tan solo 0,0073 de mejoría de accuracy.

5.4. PCA + KNN: Variando los vecinos con un alpha fijo

Hasta ahora estuvimos hablando únicamente del KNN. Ahora vamos a incorporar PCA a nuestro cálculo y veremos cómo se comporta el algoritmo.

Al principio, nos propusimos ver cómo se comporta knn variando la cantidad de k vecinos y dado un alpha fijo, para poder ayudarnos a determinar mejor cómo se comportan los parámetros.

PCA realiza un cambio de base utilizando los componentes principales. Esto lo realiza para mejorar la complejidad con la escala de dimensionalidad le agrega al KNN. No sabíamos de ante mano si este cambio aumentaría la precisión además de disminuir el tiempo de calculo, pero supusimos que las métricas no deberían disminuir, ya que estamos haciendo un cambio de base.

Vemos el comportamiento en los siguientes gráficos utilizando los distintos métodos de KNN. Como el cómputo de los autovalores es lo que más tiempo le agrega al algoritmo, decidimos empezar con un número de componentes chico (10). La cantidad de k vecinos va de 1 a 3000, porque en los 3 métodos vimos que la accuracy caía más allá de este punto (si bien vuelve a subir en 4000, no vuelve a alcanzar la misma cantidad). Lo cual es consistente, ya que es un valor muy cercano a la mitad de las críticas de la base de entrenamiento (si tomamos más de la mitad de la información, dado que es una base con críticas equilibradas, estaremos siempre tomando candidatos falsos automáticamente).

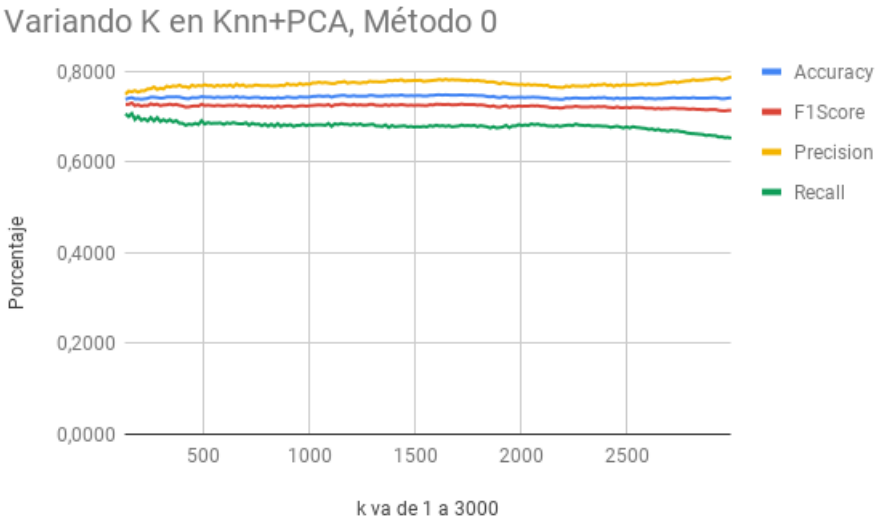


Figura 8: Variación de k en PCA+KNN método 0

Como podemos ver, 2 cosas interesantes pasaron al aplicar PCA con este método. La primera es que nuestras métricas dan mejor en general, con 0.7 siendo la base de las tres cuando antes, salvo en algunos valores, rondaba el 0.6.(La excepción es el recall, pero baja donde antes subía).

El otro suceso interesante es que donde antes la precisión subía y el recall disminuía, ahora se presentan continuamente el recall por debajo y la precision arriba, y las tres métricas se mantienen en el mismo umbral (especialmente la accuracy).

El f1-score que muestra el promedio entre el recall y la precisión está sólo un poco abajo de la accuracy, lo cual muestra que hubo esa proporción de las críticas que debieron ser positivas que son positivas, y las que debieron ser negativas y son positivas, por lo que ambas métricas se desempeñan aceptablemente.

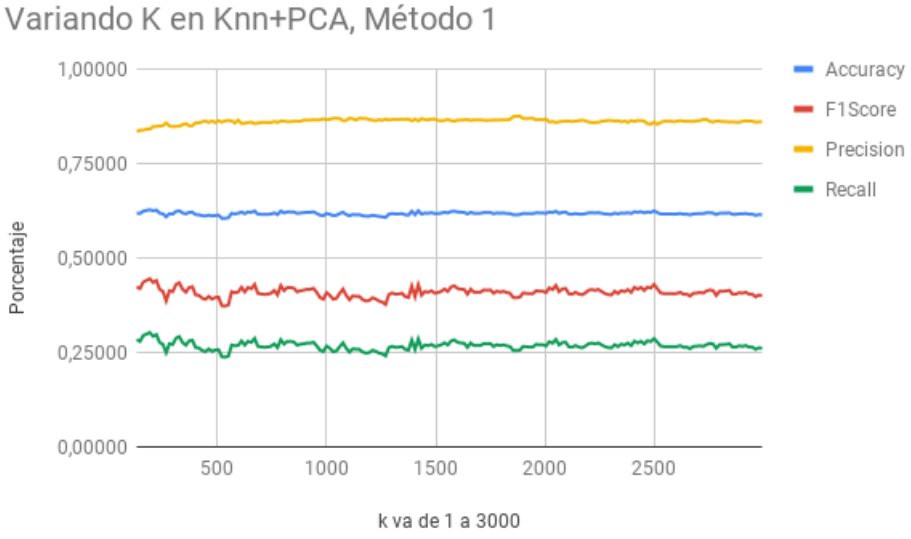


Figura 9: Variación de k en PCA+KNN método 1

Este es el resultado de correr el PCA con el método 1. Como podemos ver, el efecto de PCA es aumentar la precisión y disminuir el recall, mientras que la accuracy se mantiene en los mismos niveles que antes. El efecto del PCA parece ser aumentar la precisión dada por el KNN, a costa de reducir el recall. Aquí el F1-Score es menor a 0.5, mostrando que en nuestro resultado la mayoría de los errores de etiquetado son de falsos negativos.

Esto se debe a que en la medición, utilizando el método 1, no vemos la intersección de palabras ni cuantas de estas ultimas se repiten, por lo que puede ser que el control para ser positivo sea más estricto. Las criticas tienen que coincidir en tener las palabras de las críticas positivas, y cómo no miramos la cantidad de palabras, hay críticas negativas que coinciden en palabras que en esta medición son más cercanas. Por eso también la precisión aumenta.

5.5. Variación del alpha con k fijo

Ahora que ya tenemos una idea de cómo se comporta knn + alpha al variar la cantidad de vecinos, nos enfocamos en algunos que dan bien en terminos de suma (precision/accurancy/recall), y nos concentramos en ellos. Nuestra hipótesis es que al aumentar el alpha van a aumentar las métricas, porque estamos considerando más valores principales. Variamos los alpha de 5 en 5 desde 5 a 600. Para el método 0, que es donde nuestras métricas mejoran, los gráficos obtenidos fueron :

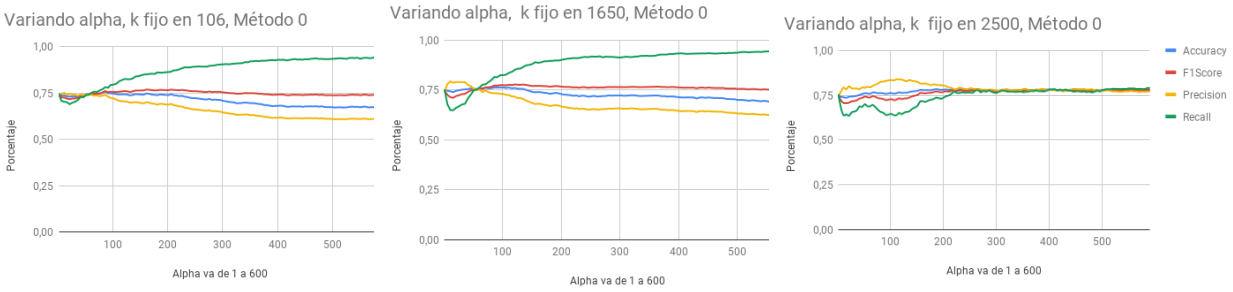


Figura 10: Variación de alpha en PCA+KNN método 0

Podemos ver que en los tres gráficos se observa el mismo comportamiento, el recall y la precisión tienen valores muy similares en un intervalo, junto con la accuracy, luego del cual la precision y la accuracy descienden y el recall aumenta. También podemos ver como al aumentar la cantidad de vecinos, el alpha requerido para llegar a este punto aumenta, con k = 106 y k = 1650 este punto se empieza con alpha = 86, mientras que cuando k = 2500 se desarrolla con alpha = 280 y el intervalo de intersección es más largo también (en el último gráfico podemos ver como las métricas se empiezan a separar con alpha = 500).

En los tres casos, utilizar un alpha mayor a esos parámetros es contraproducente, pues la accurancy se reduce y lo único que aumenta es el recall (se reducen la cantidad de falsos negativos, pero en la clasificación de críticas, nos es más importante tener mejor accurancy general). La causa de esto es porque los primeros componentes principales son más grandes y más representativos de las instancias, por lo que llega un punto que agregar autovectores no le afecta a las decisiones de etiquetado y sólo nos aumenta el tiempo de cómputo.

Igualmente, lo más interesante es ver que el alpha requerido es mayor si se utiliza una cantidad de vecinos mayor y que es en proporción mucho menor que el k, pero la ganancia en las métricas es la misma. Por lo cual, dado que el tiempo de ejecución aumenta con el k, y que el alpha requerido para conseguir las mejores métricas aumenta con el k, es preferible usar el menor k posible en el knn.

Veamos ahora que ocurre si utilizamos el método 1:

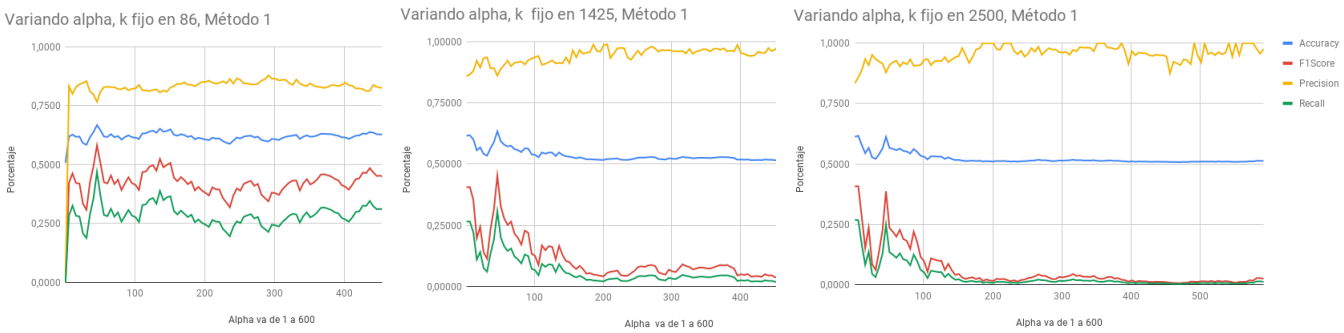


Figura 11: Variación de alpha en PCA+KNN método 1

En este método nos encontramos por el contrario, que utilizando el método 1 y variando el alpha no genera diferencias significativas en las métricas, en los tres casos hay un pico con alpha = 80 por los que nos conviene dejar ese alpha.

El que usar distintos k no nos modificara las mediciones, nos implica que nos conviene usar el k más chico de estos (86), y también que al utilizar el método 1 para calcular la distancia el pca no mejora las métricas. La causa presumiblemente es que solo lo que modificamos entre ambos es la manera de medir las distancias. En el método 1 se mide la distancia viendo las palabras que comparte y descartando las apariciones de estas, así que sólo contamos la ocurrencia o ausencia de las palabras. Sin embargo, nuestro pca genera los valores principales con la matriz de las críticas vectorizadas, donde se cuentan estas apariciones, por lo que las componentes principales y el valor del autovector no le agregan precisión a la métrica.

Una alternativa que sería interesante ver es la de cambiar la vectorización para que solo ponga 0 y 1 si aparecen las palabras, y luego realizar el proceso pca+knn con el método 0. Haciendo esto, estaríamos calculando las distancias que vemos del método 1, pero como los vectores que utilizamos son a los cuales le aplicamos pca y a esto le tomamos directamente la norma, la descomposición en elementos principales puede que nos mejore las métricas.

5.6. Comparación de tiempos KNN contra PCA + KNN

La utilización del método de PCA nos permite reducir la dimensión de nuestras matrices, por lo que hicimos la **hipótesis** de que nuestros tiempos de ejecución se verían reducidos, al utilizar PCA + KNN. Al experimentar, comprobamos que si bien esto es cierto, una vez calculada la matriz de cambio de base en el pca, el cálculo de la misma puede tardar más que el cómputo del algoritmo KNN completo sin esta.

Por eso, para nuestras mediciones decidimos observar el overhead de construir la matriz cambio de base aparte y nos fijamos solamente en la diferencia de tiempos en correr knn con las matrices normales y las cambiadas de base. Comparamos como se comportan con tamaño de base de entrenamiento diferentes, utilizando el método 0 solamente porque no hubo diferencia significante de tiempo en las distintas mediciones de knn. Y dejamos k fijo en 100.

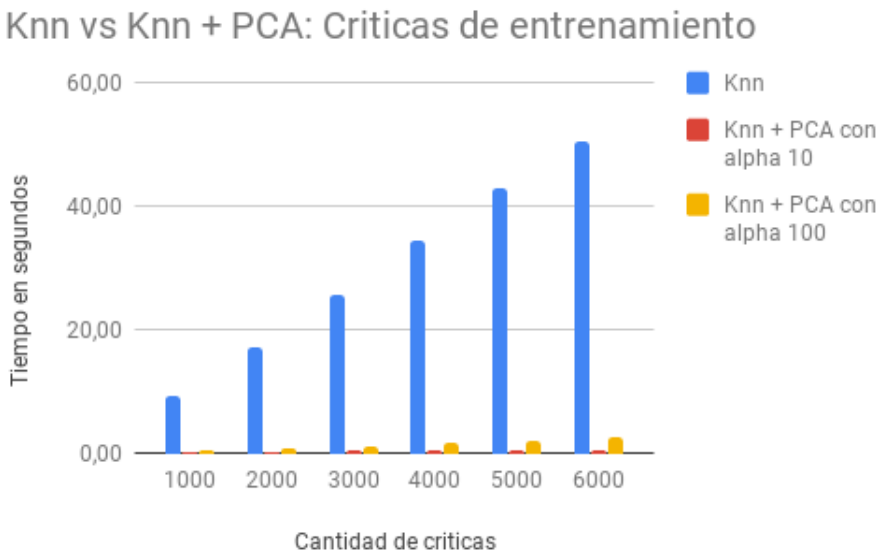


Figura 12: Comparación de tiempos variando k método 0

Como es lógico, aumentar la cantidad de críticas de la base de entrenamiento, que genera un vocabulario más grande, produce que la dimensión de la matriz de entrenamiento sea mayor y como habíamos visto antes, esto produce que los tiempo de cómputo de knn aumenten. Al compararlo con pca + knn, vemos no sólo como los tiempos de correr disminuyen considerablemente (ya que disminuimos las dimensiones), sino que el aumentar la dimensión de entrada no causa un gran aumento en el cómputo (porque la única dimensión que se incrementa para el knn es la cantidad de críticas). La diferencia entre pca = 10 y pca = 100 es proporcional a cómo aumentan los vectores de cada crítica de 10 a 100.

5.6.1. Tiempo de cálculo de la matriz cambio de base

Debido a que no lo vimos en el cálculo anterior, en esta sección mostramos el costo de calcular el overhead de la matriz de cambio de base. En nuestro algoritmo, para ayudarnos con la experimentación, nuestro PCA guarda la matriz cambio de base de mayor alfa, y si se requiere un alfa menor, utiliza la submatriz de las las primeras columnas de la matriz obtenida. Esto nos reduce en gran medida el coste de recomputo. Podemos hacer esto mientras la base de entrenamiento sea la misma porque los autovectores no cambian.

En la siguiente tabla se muestra cómo escala el coste del cómputo de la matriz de cambio de base, usando la misma cantidad de iteraciones con K = 1000 para cada autovalor, comparando la diferencia entre calcular cada vez la matriz de cero o reutilizar los datos de una matriz más grande:

Nro de componentes alpha\Tiempo(seg)	**Matriz entera**	**Usando submatrices**
10	53.04	0.274
100	472.502	0.561
200	907.860	0.87
300	1405.455	1441.988

Cuadro 1: Tiempos del calculo de la matriz cambio de base

Como podemos ver en la tabla, el tiempo de calculo de las matrices de cambio de base aumenta con cada autovalor pedido. Vemos que si duplicamos la cantidad de componentes, el tiempo requerido también se duplica, porque el mayor tiempo del cálculo se debe al cálculo del autovalor i-ésimo, y cada calculo al hacerse en matrices semejantes de igual dimensiones tiende a tender el mismo tiempo. Comparándolo además con nuestra optimización podemos ver que el tiempo sólo se gasta la primera vez, en el cálculo de la matriz de 300 que es la más grande, y en el resto utilizar las submatrices nos reduce el tiempo de cómputo

Cuando estamos utilizando KNN conjuntamente con PCA, la cantidad de componentes calculadas puede aumentarnos el tiempo de cómputo. Si estamos utilizando matrices de dimensión baja, el PCA puede desmejorar la precisión y el tiempo de ejecución. Por otra parte, si nuestro vocabulario es amplio, el tiempo perdido en calcular el cambio de base una vez es compensado con la disminución del tiempo de ejecución de KNN. Recordemos además que el alpha con mayor precisión incrementa al aumentar la cantidad de k vecinos, por lo que nos es conveniente agarrar el menor de los k con mejor resultado.

6. Buscando los mejores parámetros

En esta sección vamos a hablar sobre la metodología que seguimos para encontrar la solución óptima. Para esto, reutilizamos las soluciones formadas de las experimentaciones pasadas e intentamos determinar que método (con su respectiva configuración) es el que tiene más "potencial". En el experimento de Knn donde se varia la cantidad de vecinos y se puede apreciar el comportamiento de los 3 métodos dependiendo el valor de k (y un umbral fijo elegido arbitrariamente), los resultados recolectados arrojaron que el que llega a tener mejor accuracy es el método uno. Esto nos pareció suficiente para hacer que nos concentremos principalmente en él (aunque no solo nos enfocamos en este).

Una vez determinado el método que íbamos a tratar de optimizar nos propusimos encontrar la configuración que maximice el accuracy. Como primer idea fijamos la cantidad de vecinos y vimos como nuestro algoritmo se comportaba a medida que íbamos modificando el umbral mínimo y máximo de manera iterativa. Uno de los principales problemas con esta forma de análisis es que aunque utilicemos números relativamente chicos para iterar hay infinitos números encerrados en el medio y a partir de ciertos valores nuestro algoritmo empeoraba más y más. Siguiendo esta estrategia estábamos bastante limitados a mejorar nuestro programa pero nos sirvió para saber en que intervalo numéricos este probablemente la mejor solución.

A partir de acá pasamos a lo que vamos a hacer que es buscar de manera aleatoria una configuración de nuestro algoritmo que haga que éste supere al accuracy actual. Esta idea seria aplicada sobre el área ya estudiada con anterioridad y siendo usada una cantidad de veces indeterminada. Con este criterio de búsqueda pudimos encontrar varias configuraciones distintas que mejoraban nuestro programa pero el mayor problema es lo complicado que es encontrar un patrón de mejora entre distintos settings y también estábamos limitando las posibilidades a un área reducida. Cabe aclarar que esta búsqueda de parámetros fue hecha de forma aleatoria y se aplico sobre toda la entrada del programa Knn.

Acá podemos ver una tabla con los resultados obtenidos con Knn método 1:

Vecinos	**Umbral Máximo**	**Umbral Mínimo**	**Accuracy**	**Precision**	**Recall**
449	0.095	0.0025	0.8178	0.8307	0.7923

Cuadro 2: Mejor configuración y resultados sobre toda la muestra de tests.

Se puede apreciar en los resultados como las métricas están relativamente bien balanceadas y esto sabiendo que nuestra entrada tiene casi la misma proporción entre positivos y negativos es un buen indicativo para poder decir que nuestro algoritmo funciona de buena forma en el caso general.

7. Conclusión

Pudimos ver que estas herramientas (Knn y Pca) son una buena forma de discretizar y resolver un problema que no es simple a priori. Como técnicas a implementar nos parecieron buenos métodos y aplicables matemáticamente. Pudimos ver como el algoritmo aprendía dependiendo de los parámetros involucrados y tuvimos que analizar las métricas en consecuencia. Si bien los experimentos coincidieron con las hipótesis, las métricas fueron difíciles de entender visual y directamente.