DEPARTAMENTO DE COMPUTACION

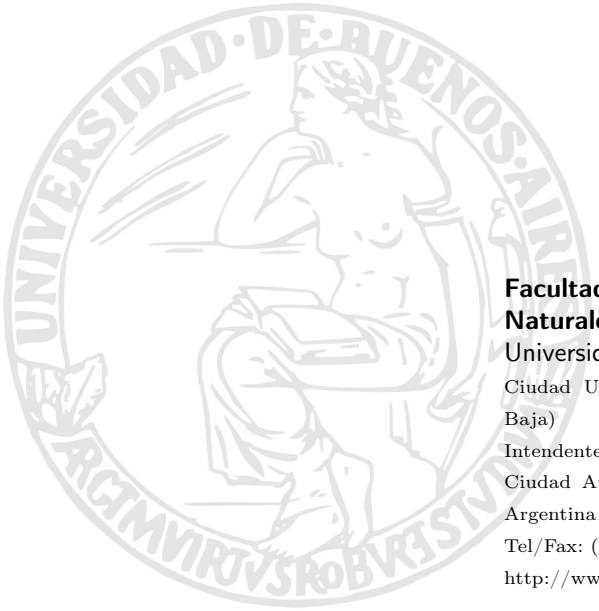
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Análisis de Sentimiento

1 de Noviembre de 2018 — Métodos Numericos

Integrante	LU	Correo electrónico
Carreira Munich, Tobías Agustín	278/17	tcarreira@dc.uba.ar
Martino, Maximiliano	123/17	maxii.martino@gmail.com
Nahmod, Santiago Javier	016/17	snahmod@dc.uba.ar
Torres, Edén	017/17	eden.etorres@gmail.com



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

http://www.exactas.uba.ar

Resumen

En este trabajo se busca obtener un primer acercamiento al área de Procesamiento del Lenguaje Natural (NLP) y al análisis de sentimiento. Además se estudia el funcionamiento y efectividad del modelo de *Bag of Words* (BoW) y las técnicas algorítmicas de k vecinos más cercanos (kNN) y el análisis de componentes principales (PCA) utilizados para la identificación de emociones en reseñas de películas.

Por medio de experimentación intentaremos obtener el mejor análisis de sentimientos posibles. La calidad de los resultados de clasificación obtenidos será analizada mediante la métrica de accuracy.

Palabras clave: bow, knn, pca, Sentiment Analysis, NLP

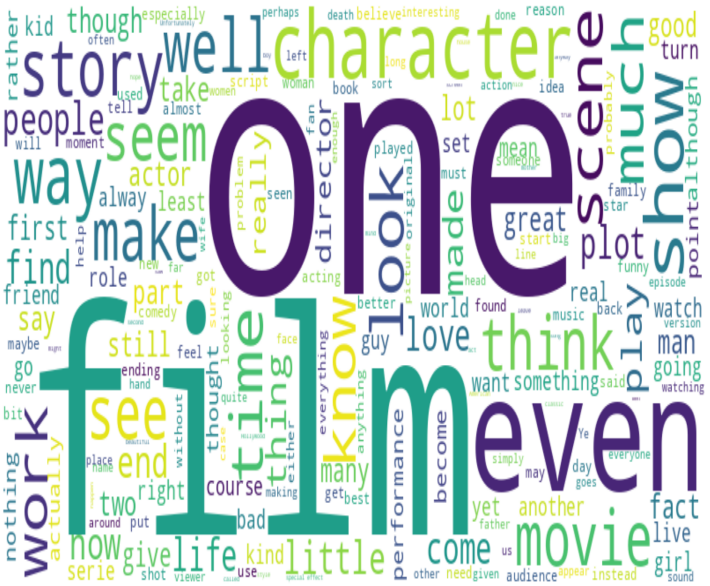


Figura 1: WordCloud armada con todas las palabras de las reviews

Índice

1. Introducción

Cada vez resulta de mayor interés el estudio de la enorme cantidad de datos generada y distribuida por Internet. Esta puede contestar preguntas que resultan de utilidad para múltiples usos, tanto comerciales como académicos.

A partir de estos intereses surgieron distintas técnicas con este objetivo. Llamamos Análisis de Sentimiento, o minería de opinión, al uso de distintas herramientas estadísticas, computacionales y lingüísticas, con el fin de extraer información de ciertos datos, sobre la emoción, opinión o interés de un grupo humano sobre un tema particular.

En el presente trabajo, nos proponemos utilizar algunas de estas herramientas en el contexto particular del estudio de polaridad de reseñas de películas en IMDb[4], es decir, para clasificar estas reseñas en "Positivas" o "Negativas".

En particular, utilizaremos:

- Modelo de Bag of Words
- Análisis de Componentes Principales (PCA)
- Clasificación mediante k vecinos más cercanos (kNN)

Las mismas serán explicadas más detalladamente en la siguiente sección.

Tenemos como objetivo estudiar la efectividad de las herramientas utilizadas, experimentando con la variación de distintos hiperparámetros y de la cantidad de datos suministrados, teniendo en cuenta el tiempo de cómputo tomado. Este último punto resulta importante dado que uno no puede esperar indefinidamente los resultados de un experimento, pero además es crucial si se desea utilizar en un sistema en tiempo real como, por ejemplo, una web que recomiende películas.

2. Desarrollo

2.1. Dataset de Reseñas

Para poder testear las herramientas que utilizaremos para el Análisis de Sentimiento, y poder detectar la polaridad dado el texto de una reseña de una película, es decir, si esta es negativa o positiva, utilizaremos un dataset de 50.000 reseñas de películas obtenidas de IMDb. Este está basado en el *Large Movie Review Dataset*, creado por Maas et al.[6]. Se encuentran clasificadas en *positivas* (aquellas que tuvieron puntaje mayor a 7 estrellas, siendo 25.000 instancias) y *negativas*(puntaje menor a 4, obteniendo otras 25.000 instancias), de las cuales la mitad de las instancias son de entrenamiento y la otra mitad de testing.

2.2. Metodología

Para la clasificación según la polaridad(*positiva* o *negativa*) utilizaremos el algoritmo de kNN. Para ello, este algoritmo considera a cada instancia como un punto en un plano euclídeo (el método es descrito en más detalle en la sección 2.2.1).

Para poder expresar a cada reseña como un vector de n coordenadas, para un n fijo, y poder usar el algoritmo de kNN, utilizaremos el modelo de Bolsa de Palabras (*Bag of Words o BoW*). Este es un método que se utiliza en el procesado del lenguaje para representar documentos ignorando el orden de las palabras. Para ello se cuenta cuantas veces aparece cada palabra en una reseña, basándose en un diccionario con todas las palabras utilizadas. De esta forma, obtenemos un vector de la misma dimensión del tamaño del vocabulario que representa esta información.

El problema que presenta este modelo a simple vista es que si tenemos un diccionario demasiado grande, generaremos vectores de igual dimensión para cada reseña. En nuestro caso, el diccionario es de aproximadamente 160.000 palabras. Para tratar de eludir este problema, experimentaremos filtrando dos tipos de palabras con distintos umbrales y veremos cuál da mejores resultados:

- **Aquellas con frecuencia alta:** casi siempre preposiciones, conjugaciones de verbos comunes.
- **Palabras con muy baja frecuencia:** no aportan información significativa para la mayoría de estos problemas, y no nos permiten encontrar un patrón común entre las diferentes clases a separar

El procedimiento de *tokenización* y construcción de vectores con *Bag of Words* fue programado y entregado por la cátedra. Por eso, sólo analizaremos los umbrales y luego nos enfocaremos en el resto de las herramientas numéricas.

También nos resulta interesante experimentar con el tamaño del dataset de entrenamiento y los hiperparámetros de los métodos numéricos empleados. Tendremos en cuenta al analizar el constante *trade-off* entre la exactitud y tiempo de ejecución de la clasificación.

2.2.1. K vecinos más cercanos

Para clasificar las reseñas, vamos a utilizar el método de los k vecinos más cercanos (kNN). Dada una reseña a clasificar, modelada como vector (como explicamos anteriormente), el método consta de buscar las k reseñas del dataset de *training* que resultan más cercanas. En particular, utilizaremos la *Norma Manhattan* para medir distancias, ya que nos interesa medir la cantidad de *tokens* que **no** tienen en común.

Luego, vemos la moda de estas k reseñas y la usamos para clasificar la nuestra. Es decir, si la mayoría de estas k reseñas son positivas, clasificaremos a la reseña como positiva. Caso contrario, como negativa. Utilizamos siempre valores impares de k, evitando empates que requerirían un criterio arbitrario de desempate.

Este algoritmo depende fuertemente del tamaño de nuestros vectores, ya que para calcular la distancia entre dos reseñas necesitamos recorrer ambos. Es por eso que surge la idea de reducir las dimensiones, intentando quedarse con la información más relevante. Para eso, se considera el **Análisis de Componentes Principales** (PCA), el cual describiremos a continuación.

2.2.2. Análisis de Componentes Principales

El Método de Análisis de Componentes Principales se basa en la idea de que, dado un conjunto de datos en forma de vectores (como puede ser nuestro BoW), hay dimensiones o *componentes* que dan mayor información que otras. Por eso, resulta interesante quedarse solo con estas componentes, eliminando las demás, que pueden resultar redundantes o incluso meter ruido. De esta forma se logra agilizar los cómputos posteriores y quizá incluso mejorar los resultados obtenidos, trabajando con un vector de menor tamaño pero con información relevante y menos ruido.

Para realizar este procesamientos, utilizamos la **Matriz de Covarianza**. Recordemos que la covarianza nos dice en qué medida dos variables (en nuestro caso, los *tokens*) varían de forma similar. A partir de esta matriz, podemos calcular las componentes principales quedándonos con los α autovectores asociados a los autovalores de mayor valor absoluto [1].

Con estos autovectores, definimos la *transformación característica*, con la matriz formada por ellos puestos como fila. De esta forma, dicha transformación toma vectores en $\mathbb{R}^n$ y devuelve vectores en $\mathbb{R}^\alpha$, disminuyendo dimensiones pero de forma que la información "más relevante" no se pierda.

El valor del hiperparámetro α nos da la oportunidad de experimentar, buscando el que mejor funcione para nuestro contexto, y para ver su relación con otros hiperparámetros.

2.2.3. Cálculo de autovectores

Como se mencionó en la sección anterior, el método PCA utiliza los autovectores de la matriz de covarianza. Para obtenerlos, utilizamos el Método de la Potencia[5].

[1] Pueden verse más detalles sobre los cálculos relacionados en http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf

Siendo este un método iterativo, resulta de interés experimentar con el criterio de terminación. Dado que cortamos el algoritmo luego de una cantidad determinada de iteraciones, decidimos ver como afecta este número a la exactitud del resultado.

2.2.4. Procedimiento para la clasificación con kNN y PCA

Una vez elegidos los hiperparámetros, el proceso de clasificación de una reseña nueva consta de su tokenización y pasaje al modelo de *Bag of Words*, el cálculo de su transformación característica (en caso de usar PCA), la búsqueda de los vecinos más cercanos del dataset de *training* (de acuerdo con la *Norma Manhattan*) y finalmente, la clasificación de acuerdo a la moda en estos vecinos.

Previamente, en caso de usar PCA, es necesario realizar el cálculo de las componentes principales sobre el dataset de *training*, como fue explicado en la sección 2.2.2.

2.3. Programación

Pasamos a contar las decisiones más significativas que tomamos a la hora de implementar los métodos en C++. Para más detalles, ver el código de los mismos en los apéndices o el código fuente entregado junto a este informe.

2.3.1. Estructuras elegidas

Para representar las reseñas, utilizamos *vector<double>* de la STL, dada su simplicidad y buen uso de la memoria caché y para las matrices utilizamos *vector<vector<double>>*.

También vale la pena destacar que para buscar los k vecinos más cercanos, guardamos las distancias en un vector, para luego poder construir un *MinHeap*. A partir de este, podemos conseguir el mejor vecino y quitarlo del *MinHeap* en $O(lg(n))$. Este método resulta mejor a ordenar el vector de distancias en términos de complejidad, dado que conseguir los k vecinos más cercanos toma, sin contar el cálculo de distancias, $O(k \cdot lg(n))$, contra un $O(n \cdot lg(n))$ del ordenamiento.

2.3.2. Optimizaciones

Dada la lentitud en la ejecución de los experimentos decidimos buscar alternativas. El mayor problema lo teníamos principalmente en la construcción de la matriz de covarianza necesaria para PCA, por eso decidimos realizar cálculos en paralelo utilizando *threads* para calcularla. Además, agregamos una opción a nuestro ejecutable para que utilice la matriz de covarianza y autovectores calculados en alguna ejecución anterior de existir.

2.4. Proceso de experimentación

Luego de terminado el desarrollo del programa principal, decidimos realizar los siguientes experimentos:

- Calidad de los resultados al variar:

  - El parámetro k de kNN, sin usar PCA
  - La frecuencia mínima y máxima de palabras filtradas, sin usar PCA
  - El tamaño del *dataset* de training, con y sin PCA
  - El parámetro alpha de PCA
  - La cantidad de iteraciones antes de frenar el Método de la Potencia
- Tiempos de ejecución al:
  - Realizar kNN, con PCA previo y sin PCA
  - Realizar kNN con y sin PCA previo, variando el tamaño del *dataset* de training

Explicaremos con más detalles cada uno de estos experimentos en la sección correspondiente.

3. Experimentación

En esta sección estudiaremos, mediante experimentaciones como varía la calidad de los resultados obtenidos al variar el parámetro k de kNN. También observaremos qué sucede al cambiar la frecuencia mínima y máxima de palabras a filtrar y cómo afecta el tamaño del dataset. Cuando usamos PCA, queremos ver cómo impacta modificar el valor de alpha y la cantidad de veces que ejecutamos el método de la potencia en la calidad de los resultados. En todos los casos, la calidad de los resultados de clasificación obtenidos será analizada mediante su *accuracy*.

- $Accuracy = \frac{VerdaderosPositivos + VerdaderosNegativos}{CantidadTotaldePredicciones}$

Además queremos experimentar sobre cómo afecta al tiempo de ejecución del programa usar o no PCA y cómo también lo impacta el tamaño del training data.

3.1. Experimentación sobre kNN

En esta sección desarrollaremos los distintos experimentos realizados para ver la calidad de resultados al ejecutar kNN, sin PCA previo, cambiando distintos parámetros. Además analizaremos los tiempos de ejecución de las pruebas en la sección 3.1.3.

3.1.1. Experimentación sobre k

Comenzamos experimentando con la cantidad de vecinos a considerar (parámetro k) en el algoritmo de kNN. Queremos ver cómo varía la métrica de *accuracy* a medida que crece el k usado.

Lo que esperamos ver es que a medida que aumentemos k, el *accuracy* aumente hasta un cierto punto y luego comience a descender. Creemos que usando un k demasiado chico, puede suceder que califiquemos mal por quedar demasiado cerca de algunos outsiders, pero al considerar muchos vecinos dejamos de darle importancia al valor de los más cercanos. Es decir, estaríamos calculando el promedio de cierta sección en el espacio quitándole importancia a la posición relativa en donde se encuentra el vector a predecir y asignandole esa polaridad, sesgando así el resultado del algoritmo hacia el valor mayoritario.

Para probar nuestra hipótesis corrimos el programa con 6 valores de k y con 25000 de las 50000 reseñas como *training data*. Guardamos el accuracy de cada prueba.

Queríamos hacer esta experimentación sobre k sin aplicar ningún filtro de frecuencias al dataset, pero dado que el tamaño de nuestro vocabulario es muy grande nos fue imposible. Correr kNN para un solo k habría tardado aproximadamente 16 días. Por lo tanto, decidimos aplicar el menor filtro posible que es **F**= 0,995 y **f**= 0,001 siendo **f** la frecuencia mínima a partir de la cual guardamos palabras y **F** la frecuencia máxima a partir de la cual borramos palabras.

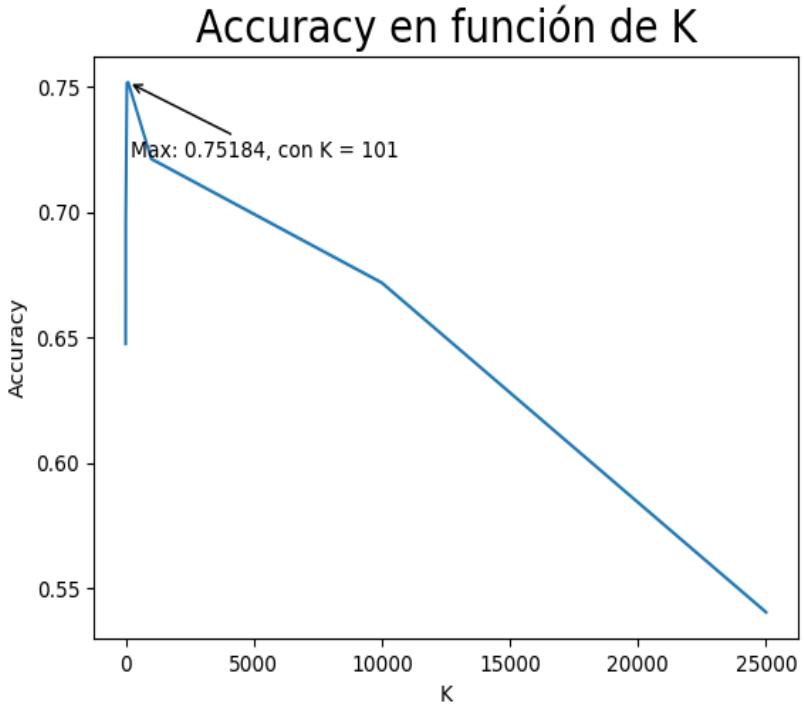


Figura 2: accuracy vs. valor de k sin PCA con F = 0.995 y f = 0.001

Luego de la experimentación, como se puede ver en la figura 2, verificamos nuestra hipótesis. Observamos cómo las métricas ascienden positivamente para los primeros 101 valores de k, pero al tomar $k = 501$ las métricas comienzan a descender.

Podemos concluir mediante los resultados obtenidos que los mejores 2 valores de k a considerar son $k = 51$ y $k = 101$, en donde 51 y 101 representan respectivamente el 0,204 % y 0,404 % del total de valores de *training*.

No nos quedamos conformes con el tiempo de ejecución de estas pruebas. Correr kNN para 6 valores distintos de k tardó aproximadamente 28 horas. Como queremos correr mucha más experimentación sobre distintas variables del programa queremos buscar una solución para poder acortar los tiempos. Lo que primero queremos ver es cómo cambian los resultados de la experimentación filtrando el vocabulario de palabras con otros valores que hagan los tiempos de ejecución más rápidos. Si no encontramos ningún filtro que cumpla con estas condiciones, vamos achicar el dataset de test ordenándolo de manera aleatoria antes.

Corrimos de vuelta el programa con los mismos parámetros cambiando solo los valores de **F**= 0,99 y **f**= 0,01.

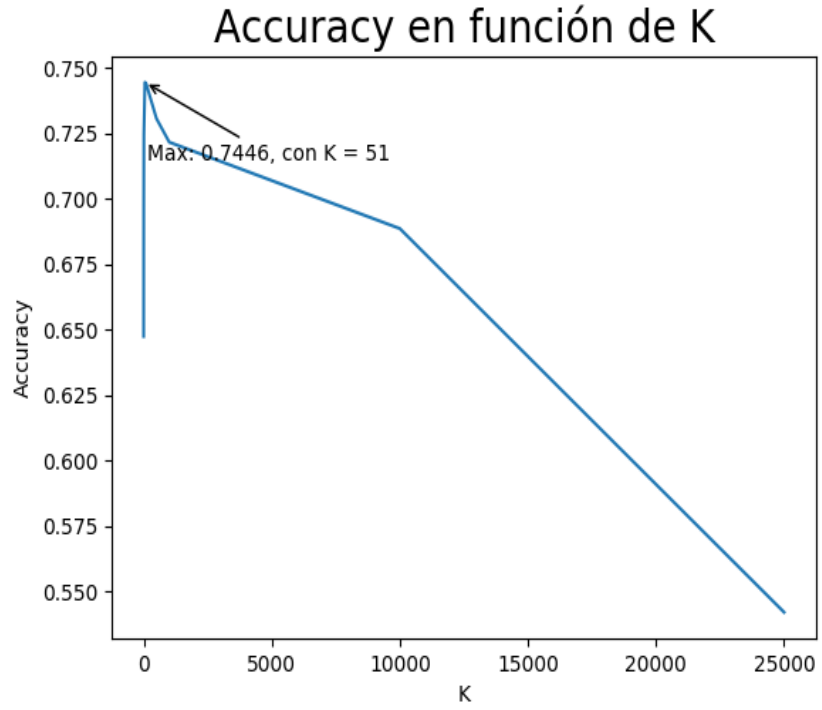


Figura 3: accuracy vs. valor de k sin PCA con F = 0.99 y f = 0.01

Con estos valores de filtrado la experimentación tardó aproximadamente 3 horas que significa un 90 % menos de tiempo que la prueba anterior.

Como podemos ver en la figura 3, los mejores valores de k siguen siendo $k = 51$ y $k = 101$. El *acurracy*, con estos valores de k, cambiando solo la frecuencia mínima y máxima de filtrado varía en 0.007. Por lo tanto para las siguientes pruebas, salvo la de la sección 3.1.2, vamos a filtrar el dataset con **F**= 0,99 y **f**= 0,01.

3.1.2. Experimentación sobre frecuencia de palabras

A continuación nos propusimos probar cómo varía la métrica de *accuracy* al cambiar la cantidad de palabras frecuentes y poco frecuentes que eliminamos de nuestros *Bag of Words*. Llamamos **f** a la frecuencia de palabras mínima a partir de la cual guardamos palabras y **F** la frecuencia máxima a partir de la cual borramos palabras. Es decir, eliminamos todas las palabras que tengan menor frecuencia que **f** y mayor frecuencia que **F**.

Suponemos que al tener una menor cantidad de palabras para cada reseña el *accuracy* al correr kNN va a ser más alto y al tener mayor cantidad de palabras va a ser peor. Esto es debido a que las palabras más y menos frecuentes suelen meter más ruido de lo que ayudan a clasificar.

Para las pruebas, elegimos 3 valores de **f** y 3 de **F** y probamos cada valor de **f** con cada valor de **F**. Corrimos las pruebas con $k = 51$, ya que en el experimento anterior vimos que es un k razonable en cuanto a calidad de resultados.

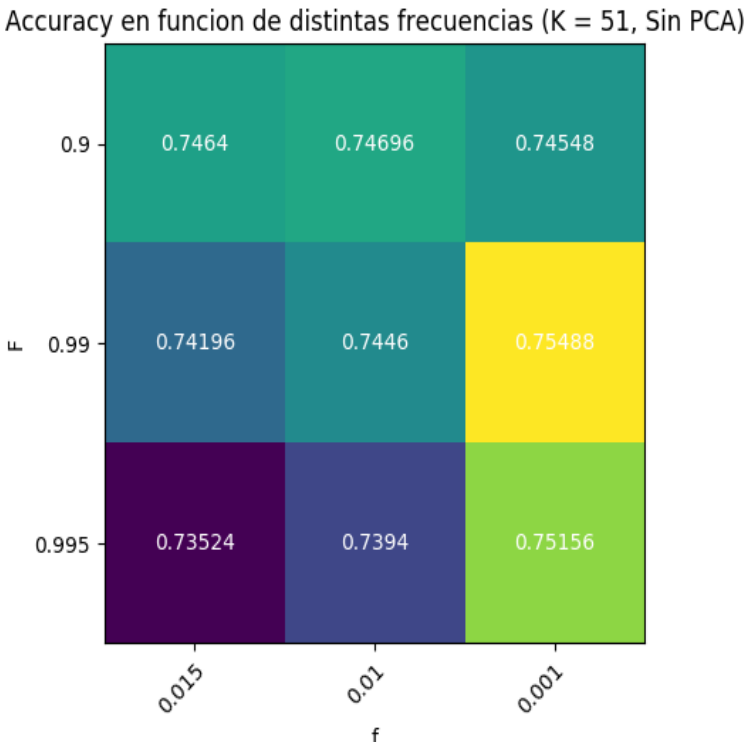


Figura 4: Accuracy vs. frecuencias con $k = 51$

Al ver que nuestra hipótesis no se cumplió en la figura 4, decidimos probar también con $k = 101$.

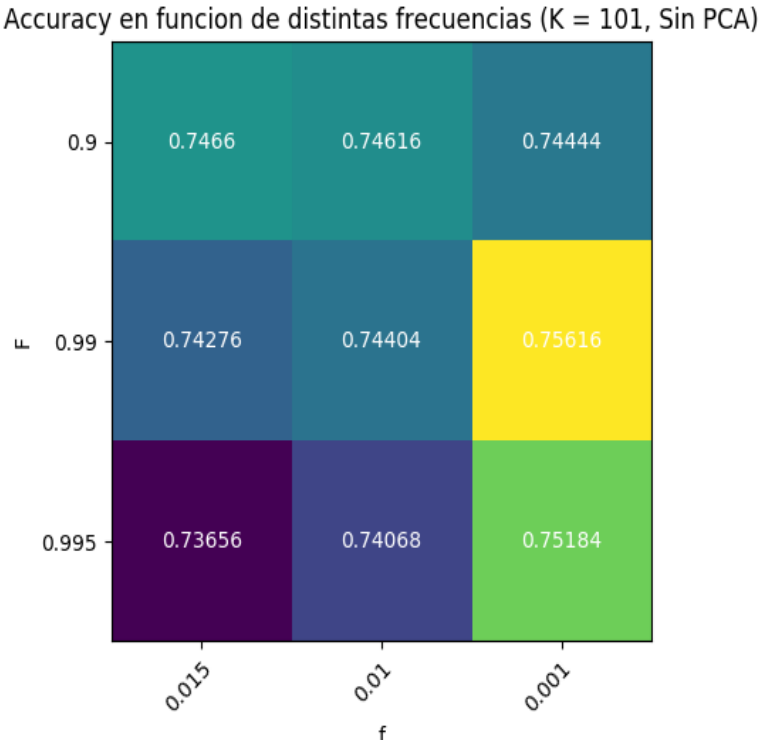


Figura 5: accuracy vs. frecuencias con $k = 101$

Tuvimos resultados parecidos para ambos k como se puede ver en la figura 4 y 5, aunque ninguno concuerda con nuestra hipótesis inicial.

La combinación que menos accuracy tiene es nuestro **f** y **F** más altos, es decir, la que elimina menos palabras de las más frecuentes (principalmente artículos y palabras por el estilo) y muchas palabras poco frecuentes (palabras extrañas, con *typos* o en otros idiomas distintos al mayoritario, que es el inglés).

Por otro lado, la combinación que resultó en mayor *accuracy* fue cuando probamos el menor **f** y el **F** de tamaño medio. Es decir, tenemos más *accuracy* cuando borramos la menor cantidad de palabras poco frecuentes y cuando borramos más palabras frecuentes que en el peor caso.

Esto nos dice que hay palabras que son poco frecuentes pero igualmente son relevantes a la hora de clasificar, como pueden ser palabras que se usan en contextos muy concretos pero con una tendencia positiva o negativa marcada. Además, vemos que hay palabras con frecuencia alta que son relevantes, pero hasta cierto punto. El equilibrio entre borrar pocas o muchas palabras frecuentes parece ser cercano a $F = 0{,}99$.

3.1.3. Experimentación con distintos tamaños del training dataset

Otro experimento que nos parece interesante es cambiar el tamaño del dataset de entrenamiento. Correremos el programa sin PCA usando el mismo dataset proporcionado por la cátedra, pero con la diferencia que la cantidad de entradas de tipo training será un número variable y la cantidad de entradas de tipo test

serán los sobrantes. Para dividir esto, antes que nada, vamos a randomizar el dataset y luego vamos a buscar igual cantidad de reseñas positivas que negativas para el dataset de training.

Sostenemos la hipótesis de que al aumentar el tamaño del dataset de training subirá el *accuracy* de nuestras predicciones. Pensamos esto debido a que al aplicar el algoritmo de kNN con tantos elementos para comparar, suponemos que el vector correspondiente a una review negativa (o análogamente positiva) tendrá muchos vecinos negativos. De la misma forma, si hay pocos elementos de training, esperamos que una review negativa tenga pocos vecinos negativos, por lo tanto dando a lugar a vecinos positivos que influyen en el rating.

Además, esperamos que eventualmente el crecimiento del *accuracy* se estanque. Es decir, que en cierto tamaño de dataset de training, deje de aumentar el *accuracy* de las predicciones, debido a la cantidad de vecinos de un elemento empieza a ser redundante para un k razonable.

El siguiente gráfico muestra los resultados de correr predicciones sin PCA con diversos tamaños del training data, con los *ks* que mejores resultados dieron en experimentos anteriores. (51, 101, 501 y 1001)

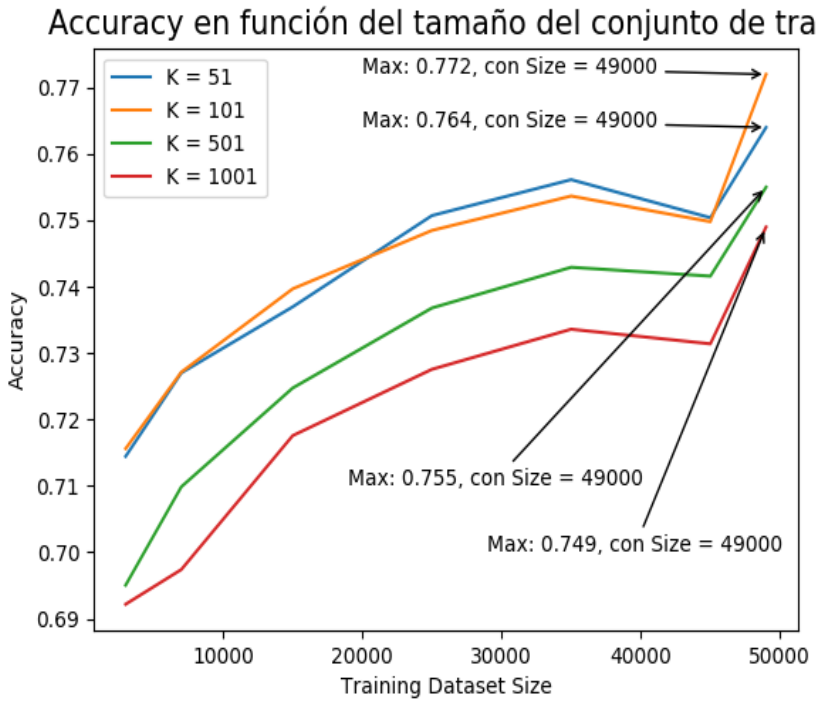


Figura 6: accuracy con distintos tamaños de training dataset

Tal como esperabamos la figura 6 muestra que a medida que incrementa el tamaño del dataset de entrenamiento, también aumenta el *accuracy*. Se puede ver que cuanto mayor sea k, los resultados de accuracy empeoran. Esto tiene sentido, dado que si hay más elementos en el algoritmo, da lugar a nuevos vecinos para una predicción en particular que sólo son abarcables por un k mayor.

No se puede observar que haya *overfitting* al aumentar el tamaño del dataset de training, es decir, se observa que el *acurracy* sigue aumentando al incrementar la cantidad de datos de entrenamiento.

Este experimento nos hizo pensar, que si bien aumentar el tamaño del conjunto de entrenamiento puede mejorar las predicciones, también puede aumentar el tiempo de ejecución. Por lo tanto, planteamos un nuevo experimento en donde ejecutamos 20 veces el algoritmo con $k = 51$ fijo, para cinco tamaños distintos del conjunto de entrenamiento (15000, 25000, 35000, 45000 y 49000) y luego medimos el tiempo de las predicciones (es decir, sin tomar en cuenta el tiempo que toma cargar los datos y tokenizar). En la figura 7, se puede ver el tiempo promedio de una única predicción.

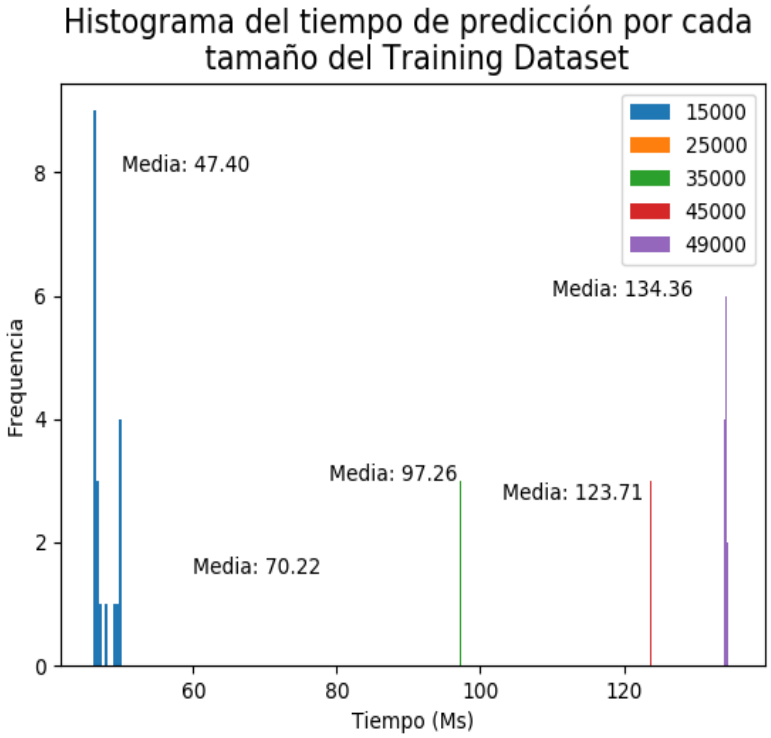


Figura 7: tiempo de ejecución promedio de una predicción, con distintos tamaños de training dataset

Como era de esperar se ve que, en la figura 7, aumentar el tamaño del dataset de training implica más elementos contra quien comparar distancias vectoriales en el algoritmo kNN, por lo tanto tiene sentido que aumente el tiempo de ejecución. Esto es algo que tener en cuenta en la práctica, dado que se podría perder *accuracy*, con tal de lograr una predicción mucho más veloz.

3.2. Experimetación sobre PCA

3.2.1. Experimetación sobre tamaño del alpha

A continuación experimentamos sobre el valor de α, siendo α la cantidad de componentes principales a considerar en el algoritmo de PCA.

Como hipótesis sostenemos que para los primeros valores de α el valor de la métrica *accuracy* va ir aumentando significativamente debido a que las primeras componentes principales describen la mayor parte de la varianza de los datos.

Luego, a partir de determinado valor de α lo suficientemente grande, conjeturamos que el valor de *accuracy* comenzará a disminuir ya que estaríamos considerando valores de la muestra de menor varianza y esto no aportaría información sino que generaría ruido afectando la predicción de polaridad del vector.

Experimentamos con distintos valores de α, con $k = 1, 51, 101, 501, 1001, 24999$, y terminando el método de la potencia luego de 20 iteraciones.

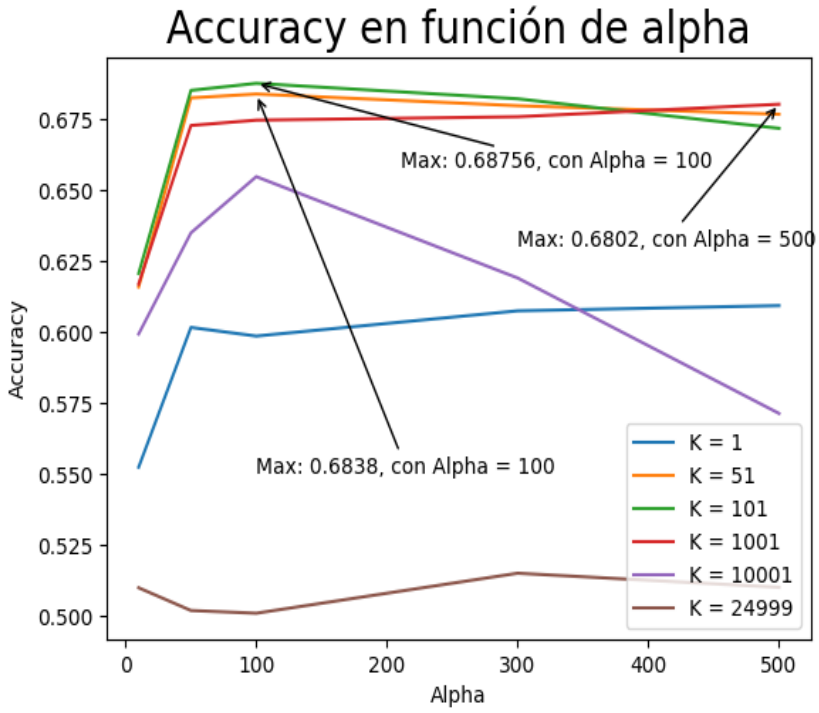


Figura 8: accuracy vs. valor de alpha

Concluimos luego de la experimentación, viendo la figura 8, que el algoritmo de PCA es particularmente útil para reducir la dimensionalidad de un grupo de datos. Observamos que para valores bajos de α la accuracy aumenta debido a que las primeras componentes principales describen la mayor parte de la varianza de los datos (más, cuanto más correlacionadas estuvieran las variables originales). Estos componentes de bajo orden a veces contienen el aspecto "más importante"de la información, y las demás componentes se pueden ignorar.

3.2.2. Experimentación sobre cantidad de iteraciones del Método de la Potencia

En esta sección mostraremos la experimentación realizada sobre distintos valores de número de iteraciones antes de detener el Método de la Potencia (En experimentos anteriores de PCA, este número era 20). El mismo, como ya contamos anteriormente, es usado para calcular los autovectores necesarios para PCA, por lo que resulta relevante detener el método en una cantidad de iteraciones suficiente como para que el método converja, pero no demasiado alto, ya que aumenta el tiempo de cómputo.

Nuestra hipótesis es que el *accuracy* aumentará a medida que aumente la cantidad de iteraciones, primero con una gran pendiente, pero luego de forma más moderada hasta llegar a un techo. Justificamos esto en que luego de cierta cantidad de iteraciones, el método converge lo suficiente y no puede mejorar más de lo que la memoria finita de los puntos flotantes se lo permite.

A continuación, mostramos la experimentación realizada con $k = 51$ y $k = 101$, y $\alpha = 100$ (el cual resultó ser de los mejores en el experimento anterior):

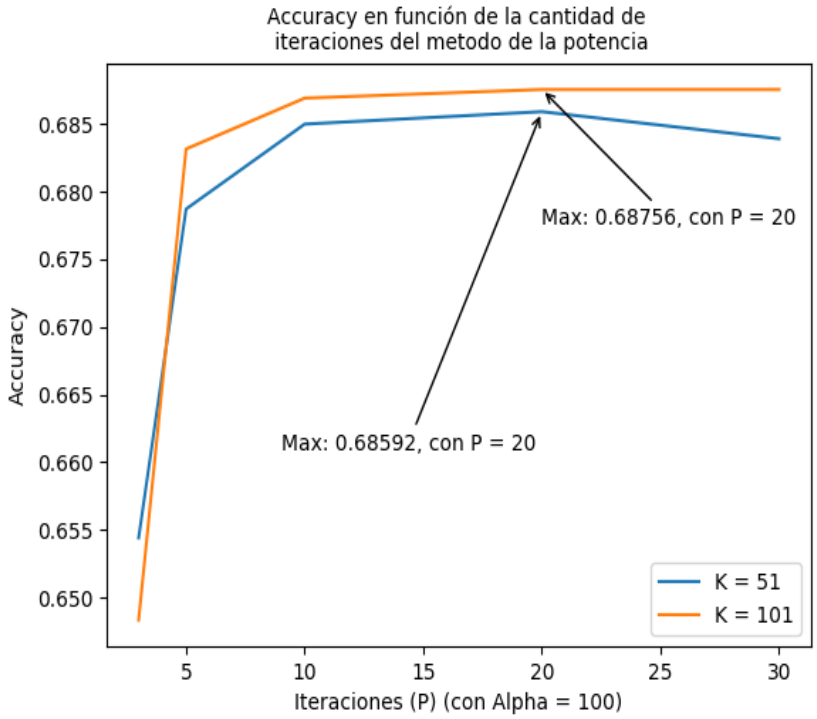


Figura 9: accuracy vs. cantidad de iteraciones del método de la potencia

Podemos observar que los resultados de la figura 9 son los esperados. En las pruebas con menor número de iteraciones se ve como el método mejora, hasta que llega a 10 y comienza a aplanarse, sin demasiada diferencia en la calidad de los resultados con un número de iteraciones superior.

3.2.3. Experimentación sobre tamaño de training data

Realizaremos distintas pruebas sobre el tamaño del dataset, descripto anteriormente en la sección de desarrollo, que utilizaremos como Training Data para nuestro algoritmo. Este experimento es similar a uno realizado anteriormente, pero ahora lo realizaremos con PCA.

Para este experimento, luego de randomizar el dataset proporcionado por la cátedra, vamos a buscar igual cantidad de reseñas positivas que negativas para distintos tamaños del dataset de training.

Lo que esperamos observar es, al igual que en el experimento anterior sin PCA, que a medida que aumentemos el tamaño del Training Data, eligiendolo de igual manera que en la sección 3.1.3, el algoritmo prediga cada vez mejor, resultando esto en un aumento en el valor de *accuracy*. Sin embargo, consideramos que este valor vaya creciendo cada vez más lento a medida que el tamaño del Training Data es suficientemente grande, debido a que ya no hay mucho más para "aprender", es decir, que la información comienza a ser redundante y no aporta nuevo "conocimiento" de manera significativa.

Además, cuanto más grande sea el dataset de entrenamiento, hay que calcular más distancias para realizar una predicción, por lo que esperamos que cada una tome mayor tiempo.

En los experimentos, utilizamos 5 tamaños distintos, que son los vimos como más relevantes en el experimento análogo a este sin PCA, con $k = 51$, $k = 101$, $k = 501$ Y $k = 1001$. Al método de la potencia, lo terminamos luego de 20 iteraciones.

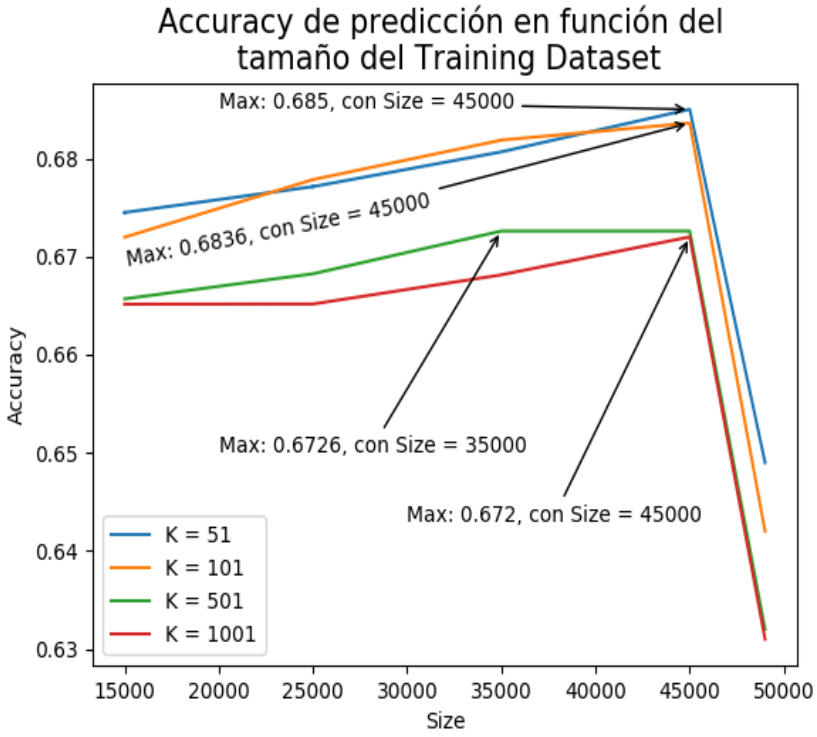


Figura 10: Acurracy de PCA con distintos tamaños de training dataset

Para medir tiempos, ejecutamos las predicciones 20 veces, tomando el tiempo de clasificar con kNN una vez que ya se redujeron las dimensiones con PCA.

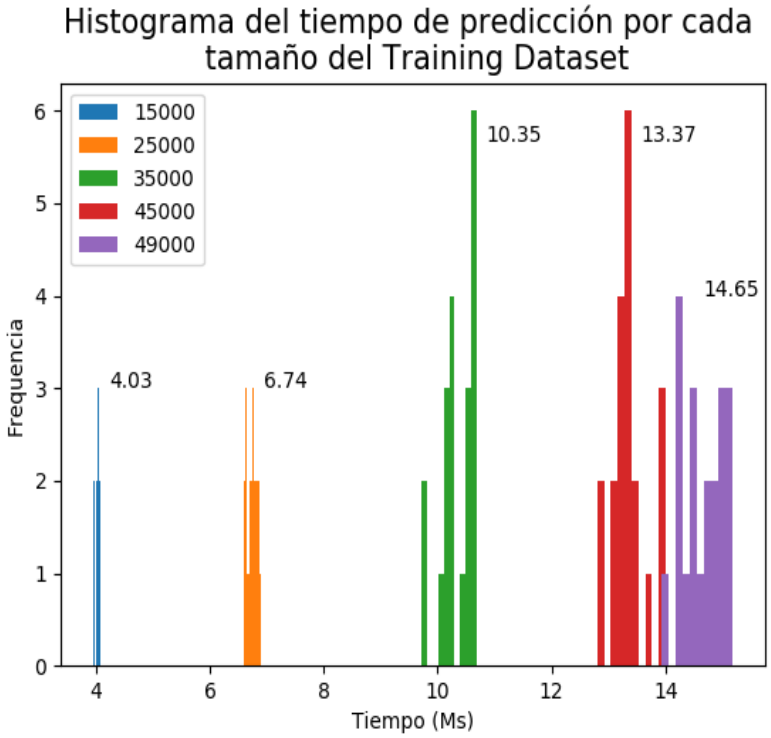


Figura 11: tiempo de ejecución promedio de una predicción, con distintos tamaños de training dataset

Posterior a la experimentación, viendo la figura 10, podemos verificar nuestra hipótesis ya que el valor de *accuracy* aumenta a medida que aumentamos el tamaño del Training Dataset, pero de forma gradual hasta que llega un punto donde el *acurracy* empieza a descender. Esto nos indica que se produce *overfitting*, es decir, llega un punto donde el dataset es tan grande que sobreentrenamos el sistema para poder predecir reseñas muy específicas que tienen poco relación con las reseñas que se testean. Comparando con la Figura 6, podemos ver que el salto entre el *accuracy* con el dataset de 15000 y el de 25000 es más pronunciado al usar kNN sin PCA que al agregarlo. Esto puede resultar de que, al usar PCA, perdemos información que puede ser valiosa, incluyendo lo que ganamos al aumentar el tamaño del Training Dataset.

Nuevamente, repitiendo los resultados del test sin PCA, observamos que hay un *trade-off*: con menos datos de entrenamiento, las estimaciones resultan menos exactas, pero toman menos tiempo. De todos modos al usar PCA el *accuracy* creció relativamente poco con respecto a lo que crecieron los tiempos de ejecución, como se ve en la figura 11, por lo que es más factible usar un dataset "pequeño" como el de 15000.

3.3. Experimetación sobre kNN y PCA

Para finalizar, nuestro último experimento consistirá en comparar el tiempo de computo al ejecutar kNN con y sin PCA previo. Nuestra hipótesis es que con

PCA corra más rápido que sin, dado que disminuye el tamaño de los vectores.

Utilizamos los valores $k = 51$ y $\alpha = 100$, que en experimentos anteriores han mostrado ser de los mejores. Computamos 18 veces los resultados, midiendo los tiempos exclusivamente de kNN (es decir, sin contar el tiempo de cargar los datos, tokenizar, o reducir dimensiones con PCA en los casos que aplique).

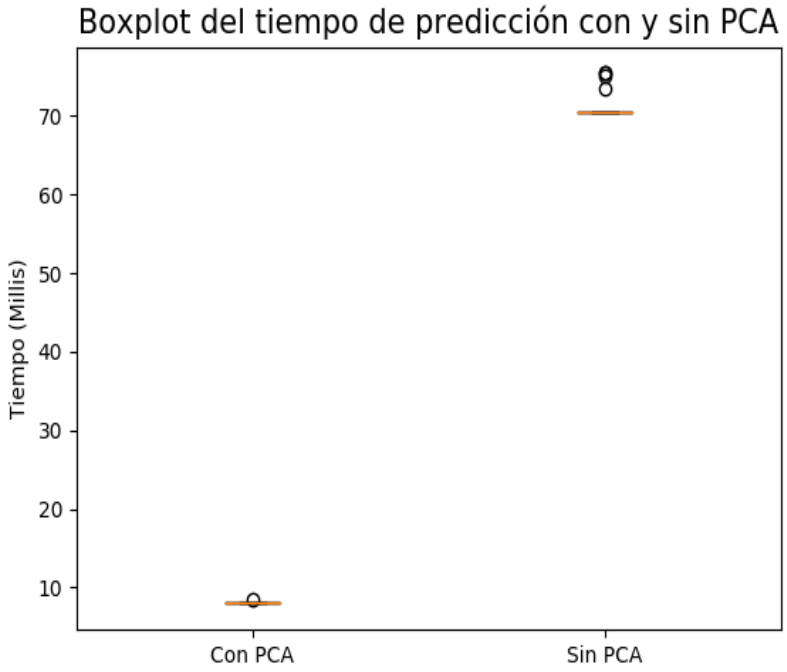


Figura 12: tiempo de ejecución promedio de una predicción, con y sin PCA previo

Como esperábamos, se ve en la figura 12 que, los tiempos de ejecución bajan al ejecutar PCA antes de hacer kNN. La contraparte es que, como se puede ver en las figuras 2 y 8, el *accuracy* con PCA no supera el 70 %, mientras que sin PCA llega a superar el 74 %. Esto nos muestra que, al usar PCA, nos encontramos con un *trade-off* entre velocidad y exactitud de resultados. Esto puede resultar del hecho de que PCA descarta información que puede resultar valiosa para la clasificación.

4. Conclusiones

En este trabajo realizamos un análisis del funcionamiento de kNN y PCA para un caso muy particular (análisis de polaridad de comentarios de películas), pero de gran interés para poder realizar en el futuro un análisis social o de mercado, además de por su posible uso en otros tipos de textos.

Con los distintos experimentos que realizamos, pudimos ver que estos métodos tiene el potencial de predecir sentimientos con una exactitud de hasta el 75 % aproximadamente, que no es para nada despreciable.

Sin embargo, estos resultados los conseguimos al utilizar kNN **sin PCA**. Como vimos en la sección anterior, usar PCA nos presenta un *trade-off* entre velocidad y exactitud.

Algo interesante visto en la experimentación fue la relación entre las palabras filtradas por frecuencia y la calidad de los resultados. En estos, contrario a nuestras expectativas previas, pudimos ver que hay palabras poco frecuentes que resultan realmente relevantes. Consideramos que un posible estudio a realizar en el futuro podría venir por el lado de ver qué palabras concretas son, para tener una mejor comprensión de lo que sucede, y probar con una mayor variedad de porcentajes para filtrar palabras.

Finalizando este trabajo, concluimos que kNN es un método simple pero efectivo para clasificar, al menos en el contexto particular de este trabajo, y que PCA es una buena herramienta para acompañar a kNN cuando necesitamos acelerar el mismo, y podemos permitirnos una reducción en la exactitud del análisis.

5. Referencias

Referencias

[1] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval, 2(1-2):1-135, 2008.

[2] Liu, B. (2007). Web Data Mining. Exploring Hyperlinks, Contents, and Usage Data. Alemania: Springer. p. 412.

[3] Weiss, S. (2005). Text Mining. Predictive Methods for Analyzing Unstructured information. (en inglés). EUA: Springer. p. 6.

[4] Internet Movie DataBase, https://www.imdb.com/

[5] R. L. Burden y J.D.Faires, Numerical Analysis, Ninth Edition. p. 576 (9.3 The Power Method)

[6] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 142–150, Portland, Oregon, USA, June 2011. Association for Compu- tational Linguistics.

A. Enunciado

Métodos Numéricos
2º Cuatrimestre 2018
Trabajo Práctico 2

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Análisis de Sentimiento

Introducción

Poder extraer información de lo que dicen los usuarios en Internet es un tema que está muy en boga hace ya unos cuantos años y que ha tenido su boom con la eclosión de las Redes Sociales[5]. Un uso potencial de estas opiniones se encuentra en lo comercial: ¿qué opinan los televidentes de la nueva serie de Adrián Suar? ¿hablan en Twitter positivamente de la nueva Coca Cola? ¿gustó la nueva película de Nicholas Cage?.

Los intereses sobre este campo trascienden lo comercial, y también la posibilidad de analizar cuantitativamente las opiniones de los usuarios (y votantes) despierta intereses políticos: ¿qué candidato tiene mejor opinión de los votantes? ¿a qué referente político atacan más en las redes sociales? son preguntas que atacan muchas consultoras. Hace ya más de 10 años se estudia la incidencia de las campañas en Internet y de las opiniones de los usuarios [3, 6] y recientemente el escándalo de *Cambridge Analytica* en las elecciones presidenciales del 2016 en EEUU han dado cuenta del uso de estas tecnologías con fines electorales.

El Análisis de Sentimiento (también conocido como *Opinion Mining*) se refiere al conjunto de técnicas de Procesamiento de Lenguaje Natural (NLP), Lingüística Computacional e Inteligencia Artificial utilizadas para poder extraer, identificar y cuantificar información acerca del estado de ánimo de un sujeto. En particular, nos interesa analizar la polaridad de un texto; es decir, si un texto tiene una emoción "positiva" o "negativa" en líneas generales.

En este trabajo práctico construiremos un analizador de polaridad para las opiniones sobre películas de usuarios de IMDB[1]. Para ello, construiremos un clasificador basado en la técnica de vecinos más cercanos (*K-Nearest Neighbors*) y reducción de la dimensionalidad con análisis de componentes principales (*Principal Component Analysis*).

Metodología

Se requiere desarrollar un algoritmo de detección de polaridad para el dataset de reseñas de IMDB; esto es: dado el texto de una reseña de una película, queremos poder decir si esta es negativa o positiva.

Para ello, contaremos con un dataset de reseñas etiquetadas en *positivas* y *negativas*. El algoritmo se *entrenará* sobre un subconjunto de esas reseñas(*conjunto de entrenamiento*). Luego, para verificar su efectividad, corroboraremos con otro subconjunto disjunto (*conjunto de testing*) los resultados que nos dé sobre cada una de las reseñas.

Como algoritmo de clasificación, utilizaremos *k vecinos más cercanos* (kNN, k-Nearest Neighbors) [1]. En su versión más simple, este algoritmo considera a cada instancia de entrenamiento (en nuestro caso, cada reseña) como un punto en el espacio euclídeo m-dimensional. Cuando querramos clasificar una instancia como positiva o negativa, buscaremos las k instancias de entrenamiento más cercanas, y le daremos la etiqueta mayoritaria entre esos vecinos.

[1] Internet Movie DataBase, `https://www.imdb.com/`

Modelo de Bolsa de palabras

La entrada de *kNN* deben ser vectores de $\mathbb{R}^n$, para un n fijo, uno por cada instancia. A cada una de estas coordenadas se les llama *features* o características en la jerga de Aprendizaje Automático. Para convertir cada reseña en un vector de longitud fija utilizamos el modelo de Bolsa de Palabras (*Bag of Words* o BoW). Dado un vocabulario precalculado (y ordenado) de palabras, ignoramos el orden de las palabras en un texto y sólo contamos cuántas veces apareció en éste. Luego, en la coordenada i-ésima tendremos cuántas veces apareció la palabra número i en ese texto.

Por ejemplo, si mi vocabulario ordenado es:

$$V = (chicos, chicas, y, quieren, rock, falta, te) \tag{1}$$

El texto "chicos y chicas quieren rock quieren rock" la codificaríamos como el vector $(1, 1, 1, 2, 2, 0, 0)$. El texto "te falta rock" lo codificaríamos como el vector $(0, 0, 0, 0, 1, 1, 1)$.

El problema que tiene este método es que si tenemos un vocabulario demasiado grande, genera vectores de igual dimensión. En nuestro caso, el vocabulario es de 160.000 palabras. Para sortear este problema, se suelen filtrar dos tipos de palabras:

- Aquellas con frecuencia demasiado alta: casi siempre preposiciones, conjugaciones de verbos comunes.
- Palabras con muy baja frecuencia

Esas palabras no aportan información significativa para la mayoría de estos problemas, y no nos permiten encontrar un patrón común entre las diferentes clases a separar (en nuestro caso, reseñas positivas y negativas). Parte de la experimentación consistirá en poner umbrales para filtrar las palabras y ver cuál da mejores resultados.

Procedimiento de k vecinos más cercanos

- Se define una base de datos de entrenamiento como el conjunto $\mathcal{D} = \{x_i \ : \ i = 1, \ldots, n\}$.
- Luego, se define m como el número total de dimensiones de la i-ésima instancia almacenada por filas y representada como un vector $x_i \in \mathbb{R}^m$.
- De esta forma, dada una instancia $x \in \mathbb{R}^m$, talque $x \notin \mathcal{D}$, para clasificarla simplemente se busca el subconjunto de los k vectores $\{x_i\} \subseteq \mathcal{D}$ más cercanos a x, y se le asigna la clase que posea el mayor número de repeticiones dentro de ese subconjunto, es decir, la moda.

El algoritmo del vecino más cercano es muy sensible a la dimensión de los objetos y a la variación de las componentes del vector instancia.

Es por eso, que las instancias dentro de la base de datos $\mathcal{D}$ se suelen *preprocesar* para lidiar con estos problemas.

Estas instancia se pueden considerar como vectores que se encuentran en un espacio de dimensión alta (igual a la cantidad de elementos), lo cual suele traer dificultades para realizar

cálculos y para diseñar algoritmos de reconocimiento que puedan utilizar la información que representan.

Teniendo en cuenta esto, una alternativa interesante de preprocesamiento es buscar reducir la cantidad de dimensiones de las muestras para trabajar con una cantidad de variables más acotada buscando que las nuevas variables tengan información representativa para clasificar los objetos de la base de entrada.

En esta dirección, consideraremos el método de reducción de dimensionalidad *análisis de componentes principales* o PCA (por su sigla en inglés) dejando de la lado los procesamientos de datos que se puedan realizar previamente o alternativamente a aplicar PCA.

Análisis de componentes principales

El método de análisis de componentes principales o PCA consiste en lo siguiente.

Sea $\mu = (x_1 + \ldots + x_n)/n$ el promedio coordenada a coordenada de los datos $\mathcal{D} = \{x_i : i = 1, \ldots, n\}$ tal que $x_i \in \mathbb{R}^m$. Definimos $X \in \mathbb{R}^{n \times m}$ como la matriz que contiene en la i-ésima fila al vector $(x_i - \mu)^t/\sqrt{n-1}$. La matriz de covarianza de la muestra X se define como $M = X^t X$.

Siendo v_j el autovector de M asociado al j-ésimo autovalor, al ser ordenados por su valor absoluto, definimos para $i = 1, \ldots, n$ la *transformación característica* de x_i como el vector $\mathbf{tc}(x_i) = (v_1\, x_i, v_2\, x_i, \ldots, v_\alpha\, x_i)^t \in \mathbb{R}^\alpha$, donde $\alpha \in \{1, \ldots, m\}$ es un parámetro de la implementación. Este proceso corresponde a extraer las α primeras *componentes principales* de cada muestra. La idea es que $\mathbf{tc}(x_i)$ resuma la información más relevante de la muestra, descartando los dimensiones menos significativas.

El método PCA previamente presentado sirve para realizar una transformación de los datos de entrada a otra base y así trabajar en otro espacio con mejores propiedades que el original.

Procedimiento para la clasificación con kNN y PCA

Una vez elegidos los parámetros k del kNN y α de PCA, el proceso completo de clasificación de reseñas se puede resumir como:

1. Ingresa una nueva reseña x no presente en el conjunto de entrenamiento.
2. Se computa el $x_b = BoW(x)$
3. Se calcula $\mathbf{tc}(x_b)$ la transformación característica
4. Se compara con cada $\mathbf{tc}(x_{bi})$, $\forall x_{bi} \in \mathcal{D}_b$ donde D_b es el dataset de training procesado con BoW
5. Se elige la moda entre los k vecinos más cercanos
6. Se devuelve la clase de la moda ("pos" o "neg") como el resultado de la clasificación.

Enunciado

Se pide implementar un programa en `C` o `C++` que lea desde archivos las instancias de entrenamiento correspondientes y que, utilizando los métodos descriptos en la sección anterior, dada una nueva instancia determine a qué clase pertenece.

Para ello, el programa **deberá** implementar el algoritmo de kNN así como también la reducción de dimensión utilizando PCA.

Con el objetivo de obtener la transformaciones características de cada método, **se deberá** implementar el método de la potencia con deflación para la estimación de autovalores/autovectores de la matriz de covarianza en el caso de PCA. Además, se **deberá** analizar el criterio de convergencia en función de la precisión obtenida y tiempo de cómputo.

Se recomienda realizar tests para verificar la implementación del método de la potencia en casos donde los autovalores y autovectores sean conocidos de antemano. También, puede resultar de utilidad emplear Python, Matlab/Octave o alguna librería de cálculo numérico para verificar los resultados.

Es **requisito** trabajar con el modelo de bolsa de palabras descripto anteriormente y estudiar los mejores umbrales para filtrar las palabras que reporten mejores resultados.

En todos los casos, se **deberá** trabajar al menos la base de datos provista por la cátedra descripta en el documento adjunto.

Experimentación

Para guiar la experimentación, se detallan los siguientes lineamientos y preguntas:

- Analizar la calidad de los resultados obtenidos al combinar kNN con y sin PCA, para un rango amplio de combinaciones de valores de k y α. Llamamos k a la cantidad de vecinos a considerar en el algoritmo kNN y α a la cantidad de componentes principales a tomar.
- Analizar la calidad de los resultados obtenidos al combinar kNN con PCA, para un rango amplio de instancias de entrenamiento. Utilizar desde muy pocas hasta todas las disponibles para identificar en que situación se comporta mejor cada uno de los métodos.
- ¿Cómo se relaciona k con el tamaño del conjunto de entrenamiento? Pensar el valor máximo y mínimo que puede tomar k y qué sentido tendrían los valores.
- En base a los resultados obtenidos para ambos métodos, seleccionar aquella combinación de parámetros que se considere la mejor alternativa, con su correspondiente justificación, compararlas entre sí y sugerir un método para su utilización en la práctica.

También, **se debe** considerar en los análisis anteriores el tiempo de ejecución.

La calidad de los resultados de clasificación obtenidos será analizada mediante diferentes métricas:

1. Accuracy
2. Precision/recall
3. F1-Score

En particular, la métrica más importante que **debe** reportarse en los experimentos es la tasa de efectividad lograda o *accuracy*. También, se **debe** utilizar al menos otra de las métricas mencionadas, aunque no necesariamente para todos los experimentos realizados.

En todos los casos es **obligatorio** fundamentar los experimentos planteados, proveer los archivos e información necesaria para replicarlos, presentar los resultados de forma conveniente y clara, y analizar los mismos con el nivel de detalle apropiado. En caso de ser necesario, es posible también generar instancias artificiales con el fin de ejemplificar y mostrar un comportamiento determinado.

Puntos opcionales (no obligatorios)

- Proponer y/o implementar alguna mejora al algoritmo de kNN.
- Aplicar n-gramas (ver [2] para una explicación).
- Realizar un estudio experimental de los métodos propuestos sobre una base de entrenamiento utilizando la técnica *K-fold cross validation* mencionada anteriormente, con el objetivo de analizar el poder de clasificación y encontrar los mejores parámetros de los métodos.
- Realizar los experimentos de los ítems anteriores para valores distintos de K del método K-fold[2], donde K es la cantidad de particiones consideradas para el cross-validation.
  - Justificar el por qué de la elección de los mismos. Tener en cuenta que probar todos los valores posibles puede ser muy costoso.
  - ¿En qué situaciones es más conveniente utilizar K-fold con respecto a no utilizarlo?
  - ¿Cómo afecta el tamaño del conjunto de entrenamiento?

Formato de entrada/salida

El ejecutable producido por el código fuente entregado deberá contar con las funcionalidades pedidas en este apartado. El mismo deberá tomar al menos tres parámetros por línea de comando con la siguiente convención:

```
$ ./tp2 -m <method> -d <dataset_path> -o <classif>
```

donde:

- `<method>` el método a ejecutar con posibilidad de extensión (0: kNN, 1: PCA + kNN, ... etc)

[2]Para esta tarea en particular, se recomienda leer la rutina `cvpartition` provista por Octave/MATLAB.

- `<dataset_path>` será el nombre del archivo de entrada con los datos de entrenamiento y los datos de testeo a clasificar. Este debe contener múltiples líneas con el siguiente formato:

  `<id>,<dataset>,<etiqueta>,<tokens>`

  donde

  - `<id>` es el id de la review
  - `<dataset>` es `test` o `train` según corresponda
  - `<etiqueta>` es `neg` o `pos` según corresponda
  - `<tokens>` es una lista de tokens separados por coma

  Utlizar el archivo `imbd_tokenized.csv` a modo de ejemplo.

- `<classif>` el nombre del archivo de salida. Este archivo deberá tener tantas líneas como entradas de testing hayan en el dataset que se utilice. Cada una deberá tener el id de la review que haya así clasificada, así como la clase (positiva o negativa) a la cual fue asignada.

Un ejemplo de invocación sería el siguiente:

```
$ ./tp2 -m 1 -d datos/imbd_tokenized.csv -o result.csv
```

Además, el programa deberá imprimir por consola un archivo, cuyo formato queda a criterio del grupo, indicando la tasa de reconocimiento obtenida para cada conjunto de test y los parámetros utilizados para los métodos.

Nota: cada grupo tendrá la libertad de extender las funcionalidades provistas por su ejecutable. En particular, puede ser de utilidad alguna variante de toma de parámetros que permita entrenar con un porcentaje de la base de datos de entrenamiento y testear con el resto (ver archivos provistos por la cátedra). Además, puede ser conveniente separar la fase de entrenamiento de la de testeo/consulta para agilizar los cálculos.

Fecha de entrega

- *Formato Electrónico:* jueves 1º de noviembre hasta las 23.59 hs, enviando el trabajo (informe + código) a la dirección `metnum.lab@gmail.com`.

  - El subject del email debe comenzar con el texto `[TP2]` seguido de la lista de apellidos de los integrantes del grupo separados por punto y coma `;`.
    Ejemplo: `[TP1] Lennon; McCartney; Starr; Harrison`

  - Se ruega no sobrepasar el máximo permitido de archivos adjuntos de 20MB. Tener en cuenta al realizar la entrega de no ajuntar bases de datos disponibles en la web, resultados duplicados o archivos de backup.

- *Formato físico:* viernes 2 de noviembre en la clase de laboratorio.

  Se debe entregar solamente la impresión del pdf del informe que se entregó por mail. Ambas versiones **deben** coincidir.

- *Recuperatorio:* jueves 22 de noviembre hasta las 23.59 hs, enviando el trabajo corregido a la dirección `metnum.lab@gmail.com`

- Pautas de laboratorio:
  `https://campus.exactas.uba.ar/pluginfile.php/109008/course/section/16502/pautas.pdf`

Importante: El horario es estricto. Los correos recibidos después de la hora indicada serán considerados re-entrega.

Referencias

[1] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[2] Daniel Jurafsky. Speech and language processing: An introduction to natural language processing. *Computational linguistics, and speech recognition*, 2000.

[3] Matthew James Kushin and Masahiro Yamamoto. Did social media really matter? college studentsúse of online media and political decision making in the 2008 election. *Mass Communication and Society*, 13(5):608–630, 2010.

[4] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[5] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[6] Lee Rainie and John Horrigan. Election 2006 online. *Pew Internet & American Life Project Report*, 2007.

B. Código fuente relevante numéricamente

Código de búsqueda de k vecinos más cercanos (kNN):

```
// Utiliza dist, que simplemente calcula la distancia Manhattan dados dos vectores
bool knn(const VectorizedEntriesMap &entries, const std::vector<double> &words, int k) {
    std::vector<std::pair<double, bool> > distancias;

    for (auto entry:entries) {
        distancias.push_back({
            dist(words, entry.second.bag_of_words),
            entry.second.is_positive});
    }

    MinHeap heapDistancias(distancias.begin(), distancias.end());

    int cant_pos = 0;
    for (int i = 0; i < k; i++) {
        std::pair<double, bool> vecino = heapDistancias.top();
        heapDistancias.pop();

        if (vecino.second) {
            cant_pos++;
        }
    }

    return cant_pos >= (k - cant_pos);
}
```

Código de PCA:

```
// Este codigo es parte del metodo pca, escrito en pca.cpp
// Solo ponemos este segmento ya que es el realmente relevante numericamente
// El resto es preparacion de datos y manejo de datos precalculados

// Vector para calcular autovector
std::vector<double> v(dimReviews);
// Autovectores calculados
std::vector<std::vector<double>> autovectores;
for (unsigned int i = 0; i < alpha; i++) {
    std::cerr << "                              " << "\r";
    std::cerr << "Buscando autovector " << i << "\r";
    double autovalor = metodoPotencia(covarianza, v, p);
    autovectores.push_back(v);
    reducirAutovalores(covarianza, v,
                       autovalor);
}
salidas = autovectores;
```

Código de reducirAutovalores, que dado una matriz y un autovalor de la misma, cambia la matriz de forma que ese autovalor cambie por 0:

```
void reducirAutovalores(
    std::vector<std::vector<double>> &B,
    std::vector<double> &v,
    double autovalor) {
    for (unsigned int i = 0; i < B.size(); i++) {
        for (unsigned int j = 0; j < B.size(); j++) {
            B[i][j] -= autovalor * v[i] * v[j];
        }
    }
}
```

Código del Método de la Potencia:

```
double metodoPotencia(
    const std::vector<std::vector<double>> &A,
    std::vector<double> &v,
    int numeroIteraciones) {
    obtenerVectorRandom(v);

    for (int iteracion = 0; iteracion < numeroIteraciones; iteracion++) {
        v = normalizar(A * v);
    }
    return (v * (A * v)) / (v * v);
}
```