DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico #1

3 de Septiembre de 2019 — Metodos Numericos

Integrante	LU	Correo electrónico
Santiago Corley	599/17	sano.corley@gmail.com
Nahuel Melis	748/18	nahuelmelis95@gmail.com
Matias Gastron	183/18	matiasgastron@hotmail.com
Nicolas Pacheco	108/18	nikopacheco22@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - Pabellón I
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Argentina
Tel/Fax: (54 11) 4576-3359
http://exactas.uba.ar

Resumen

En el siguiente informe, detallaremos nuestra implementación de un algoritmo Clasificador por KNN capaz de clasificar reseñas de peliculas como positivas o negativas; estudiaremos distintos métodos que permiten incrementar la precisión y performance de nuestro clasificador, y analizaremos el desempeño del mismo para varios parámetros relativos al clasificador (el número de vecinos de KNN) y a nuestro método de Reducción de Data (el número de autovectores principales a conservar de PCA) para encontrar la combinación de ellos que nos ofrece los mejores resultados.

Palabras claves

- KNN
- PCA
- Bag of Words

Índice

1. Introducción

El problema que se nos fue planteado fue el de lograr clasificar reseñas de películas en Positivas y Negativas, teniendo como referencia una extensa serie de reseñas previamente clasificadas. Este campo se llama análisis de sentimiento y es una de las ramas del procesamiento de lenguaje natural que se dedica a identificar,extraer, cuantificar y estudiar distintos estados o información subjetiva. Una de las tareas de este campo es el de poder clasificar la polaridad de un texto o documento dado. Si bien se pueden buscar más cosas que positivo y negativo (enojo, tristeza felicidad, entre otras) nosotros en este trabajo nos centraremos en estas dos.

Esta tarea a priori puede parecer no tener mucha complejidad ya que podría sacarse una conclusión según las palabras que usa de forma intuitiva con solo leerlo rápidamente, pero esto es erróneo ya que si una reseña dice "La mejor película del mundo.ªunque podemos estar tentados de clasificarla como positiva, está en un contexto que no podemos dejar de lado en el análisis, y este contexto puede mostrarnos que la reseña en verdad no es positiva, si no que usando la ironía resulta en una negativa. Esto para nosotros los humanos puede ser fácil de detectar, pero para una computador sin un aprendizaje previo es una tarea imposible, ya que ni el computador más avanzado es capaz de entender completamente las complejidades del Lenguaje natural humano.

Nosotros vamos a centrarnos en la resolución de este problema usando técnicas de machine learning. En este caso trabajaremos con el algoritmo de clasificación por KNN (K-nearest neighbours). Este algoritmo requiere que nuestras instancias de estudio (textos) se puedan representar como vectores en un espacio euclidiano, se basa en la búsqueda de los k vecinos mas cercanos de nuestra instancia a clasificar (en forma de vector) en dicho espacio y en base a eso define cual es la clasificación más probable. Este algoritmo de aprendizaje es del tipo supervisado ya que el data set usado para el entrenamiento contiene etiquetas que nos van a decir a que clase pertenece cada data, en nuestro caso cada reseña.

Al estar trabajando con matrices de enorme tamaño, tanto por el número de instancias a formar nuestra Base de Datos de entrenamiento como la enorme cantidad de palabras que se incluirán en nuestro diccionario, inevitablemente nos toparemos con el problema llamado . "La Maldición de la Dimensionalidad"('The Course of Dimentionality' en inglés), el cual implica que, más allá de los costos incrementados de cualquier operacion sobre nuestro Dataset, nos encontraremos con que ciertas funciones, razonamientos e implementaciones que funcionarían adecuadamente en dimensiones menores, se comportan de forma erratica, impredecible, o insatisfactoria al trabajar en mayores dimensiones; por ejemplo, el calculo de Distancia Euclidia entre dos vectores puede ser un buen indicativo de su similitud general cuando sus dimensiones son bajas, pero al aumentarlas, el mismo proceder ya no nos ofrece las mismas certezas y funcionalidades, debiendo implementar mecanismos distintos (como la Distancia Manhattan). En nuestro caso, si bien estudiaremos los efectos de la dimensionalidad en cuanto a cantidad de instancias a considerar, nos concentraremos principlamente en los efectos relativos a la cantidad de palabras que compondran nuestro diccionario (lo que determinará la cantidad de columnas de nuestro Dataset), en particular como afecta a los tiempos de cómputo, precisión en las predicciones y formas de solucionarlo o reducir su influencia.

Nuestra principal solución para este problema fue usar PCA, con el cual hacemos una proyección ortogonal a un espacio mas chico, tratando de perder la menor cantidad de información relevante posible. Evaluando la relación de Covarianza entre las distintas variables de nuestro Dataset podemos hacer justificaciones en cuanto a la conservación de información relevante y a la seguridad de que el método nunca falle (es decir, que la matriz de transformación esté construida adecuadamente, que PUEDA ser construida para cualquier Dataset,etc), de forma que, evaluando tanto tiempos de cómputo como precision de predicciones, podamos comparar los resultados con los correspondientes al clasificador pos KNN sin PCA, para poder determinar en que circunstancias, y para que parametros particulares de PCA y KNN, es conveniente usar uno por encima del otro.

Más allá de lo relativo a los Clasificadores y Análisis del Lenguaje Natural, deberemos implementar una función que nos permita generar las matrices de transformación de PCA, la cual deberá estar fuertemente basada en nuestros conocimientos de Algebra Lineal, Matrices Ortogonales, Autovalores y Autovectores y Descomposiciones Matriciales, la cual deberá no solo dar resultados adecuados para nuestra implementación, sino demostrar en su diseño nuestro entendimiento personal de los conceptos que aseguran su funcionalidad. Para eso implementaremos una funcion que aplique multiples iteraciones del Método de la Potencia, junto con el Método de Deflación asociado, de forma que nos permita obtener una cantidad deseada de los autovalores principales (con sus autovectores asociados) para una matriz dada.

2. Desarrollo

2.1. Categorización de Textos y Análisis de Componentes Principales

Co mprendido como la búsqueda y diseño de procesos automáticos capaces de organizar documentos en categorías predefinidas en un contexto de análisis particular, el rubro de la categorización de textos abarca tareas como el Análisis Temático, Análisis de Mercado, Text Datamining, y aplicaciones múltiples en ciencias tales como la Sociología y la Psicología, todas ellas de uso extenso (y en constante crecimiento) en múltiples industrias; la búsqueda de un balance entre buena performance (precisión) y adecuada complejidad temporal (ambos con demandas cada vez más estrictas por parte de los usuarios) llevaron al desarrollo de múltiples algoritmos y enfoques a través de los cuales abordar el problema; en este informe implementaremos y estudiaremos uno de ellos: el método KNN (k-nearest neighbours), junto con un estudio complementario del método de PCA.

Los distintos métodos de categorización de texto siempre trabajan con textos representados según un modelo de Bag of Words: arreglos m-dimensionales, siendo m el número de palabras definidas en nuestro vocabulario, con cada elemento del arreglo indicando el número de apariciones de cada palabra en el texto modelado. Bag of Words funciona poniendo las palabras que aparecen en las reseñas en un vector y filtraremos un porcentaje de las que tienen más y menos apariciones(variará este porcentaje en las distintas experimentaciones). Esta propiedad genera el problema de darle una enorme dimensionalidad a los Dataset; en algoritmos como el KNN, que deberán recorrer toda su Training Datasets para clasificar un nuevo texto, esto implica una complejidad de al menos O(n.m), siendo n el tamaño del Dataset.

Afortunadamente algoritmos como éste trabajan sobre Datasets en forma de matrices ralas, ya que generalmente incluso las críticas más largas y verbosas solo contendrán un pequeño porcentaje del total de palabras en nuestro diccionario, lo que nos permite aplicar ciertas factorizaciones y simplificaciones de costo reducido que, a su vez, reducirán significativamente temporal de nuestro clasificador sin sacrificar precisión, como se verá más adelante.

2.2. K-nearest Neighbours

Es uno de los representantes más simples de los métodos de Aprendizaje Basado en Instancias, los cuales almacenan sus Datasets de Entrenamiento (o representativos de los mismos) para luego estudiar la relación de nuevas instancias (las cuales, al igual que las de entrenamiento, deben poder ser representadas en forma de puntos en un Espacio Euclideo) con las mismas para poder asignarles una clasificación adecuada; los mismos son también comúnmente denominados como Lazy-Learning Methods (métodos de aprendizaje perezoso), dado que posponen la generalización del Training Dataset hasta el momento de categorizar una nueva instancia, en lugar de generar una función discriminante que generalice al mismo inicialmente para luego aplicarla a nuevas instancias.

El Clasificador KNN, en su forma más elemental, tiene un funcionamiento e implementación generalmente muy simple: Primero, un algoritmo recibe las instancias de entrenamiento y las procesa, generando su equivalente vectorial, por ejemplo:

- Si las instancias son textos, generalmente se las modela como Bags of Words: Conociéndose de antemano el Vocabulario* m-dimensional asociado a todas las posibles instancias de textos a estudiar, se crea un arreglo m-dimensional, de forma que cada espacio indica el número de apariciones de cada palabra (según su posición en nuestro Vocabulario) en el texto. Generalmente, nuestro Vocabulario no incluirá palabras que suelen aparecer muchas veces en la mayoría de textos, o muy pocas veces en muy pocos textos, ya que por lo general no darán mucha información significativa acerca del texto.

- Si las instancias son imágenes, y especialmente cuando se estudian símbolos, dígitos o formas básicas en imágenes, se suele preprocesar la imagen de forma que se elimina todos los espacios que no contengan información relevante, y se priorice el contraste entre la información a estudiar (como dígitos en una patente) y el fondo de la imagen, generalmente creando una imagen monocromática de tamaño reducido; finalmente, se genera un vector (m.n)-dimensional (siendo m y n las dimensiones de la imagen), con cada m-elementos consecutivos representando una fila de píxeles de la misma, con 0's representando el background de la imagen y 1's representando la información en la misma.

Segundo, cada una de las representaciones vectoriales del Training Dataset se almacenan en una matriz n-dimensional, tal que cada fila tiene asociado el/los valores correspondientes a la instancia original (en el

caso de nuestro estudio, el valor será si la reseña en cuestión era positiva o negativa).

Para clasificar una nueva instancia, se le aplica la transformación a vector a la misma y se la compara iterativamente con TODAS las instancias del Dataset, midiéndose la distancia euclidiana entre cada uno de ellos y nuestra instancia a clasificar; Así, se obtienen las k instancias en el Dataset cuya distancia a la misma es menor que todas las demás. Finalmente, se le aplica a este conjunto de k instancias una "votación" que determinará la clasificación de la nueva instancia. Esta generalización usualmente caerá en uno de los siguientes métodos:

- **Por Moda:** Simplemente se busca el valor que aparece la mayor cantidad de veces en nuestro conjunto de instancias (su moda) y se le asigna a la instancia a evaluar.
- **Distance-Weighted:** Similar a la votación por moda, con la diferencia de que cada vector del conjunto, en lugar de aumentar en 1 la cantidad de apariciones de su clase asociada, lo aumenta en una cantidad positiva inversamente proporcional a su distancia con respecto a la instancia a clasificar:

$$Clase(x) =_{v \in V} \sum_{i=0}^{k} \frac{\delta(v, Clase(x_i))}{d(x, x_i)^2}. \tag{1}$$

Tal que:

$$\delta(v, Clase(x_i)) = \begin{cases} 1 & si \quad v{=}Clase(x_i) \\ \\ 0 & si \quad v{\neq}Clase(x_i) \end{cases} \tag{2}$$

Siendo V el conjunto de clases/valores asociados al conjunto de k vectores, y d(x,y) la distancia euclidiana entre los puntos x e y. Si llegase a ocurrir que uno de los vectores del conjunto coincide PERFECTAMENTE con la posición del vector a clasificar, se le asigna automáticamente su clase asociada.

Cabe señalar que, a diferencia de lo que ocurre con la votación por Moda, al reducirse el impacto a la hora de determinar la clase de una instancia nueva de vectores más lejanos, se reduce el impacto negativo de elegir un k demasiado grande. Inclusive, es posible elegir un k tan grande como el tamaño del Training Dataset sin romper el funcionamiento del clasificador, aunque la complejidad temporal sigue siendo muy alta para que resulte práctico.

La precisión del algoritmo, mas allá de la calidad de nuestro Training Dataset y su cantidad de elementos, estará determinada por el valor que elijamos para k. Sin embargo, el valor mas apropiado para k dependerá completamente de las propiedades del Dataset y por lo tanto, como consecuencia de la enorme dimensionalidad requerida para el buen funcionamiento del algoritmo, sera imposible determinarlo de antemano aplicando algún razonamiento o función matemática de bajo costo temporal; en la mayoria de los casos, la forma mas confiable de determinar un k apropiado será reservar una parte de nuestro Dataset para usar, crear nuestra matriz con el resto y estudiar la precision del algoritmo para múltiples valores de k, pudiendo repetirse el proceso escogiendo diferentes elementos del Dataset para usar como control de precisión. Por lo general, un k muy bajo acarreará alta inestabilidad y sensibilidad a los outliers del Dataset, mientras que un k demasiado alto incrementara el sesgo, y en casos de estudio que involucren múltiples clases, aquellas que tengan significativamente menos elementos que las demás pueden resultar ignoradas por el algoritmo si el k esta cerca de su cantidad total de instancias asociadas.

2.3. Principal Component Analysis

Aún después de aplicar criterios de discriminación de palabras muy usadas y muy poco usadas en las instancias de nuestro Dataset, inevitablemente deberemos trabajar con vocabularios que involucren mas de 100000 palabras, lo que puede ser mucho mayor al número de instancias en la Database en cuestión, lo que como se discutió antes, se traduce en un significativo costo temporal a la hora de clasificar nuevas instancias.

Además, nada asegura que todas las palabras restantes transmitan la misma cantidad de información; aquellas que aparecen equitativamente en reseñas positivas y negativas no ofrecerán información que permita distinguir la clase de una nueva reseña, pero aun así afectarán las mediciones de distancias y, por lo tanto influirán en la elección de los k-nearest neighbours; lo mismo ocurrirá con palabras utilizadas en contextos erróneos, que vendrían a representar *ruido* en nuestras muestras. Por lo tanto, si encontráramos una reducción de la dimensión de nuestro Dataset que ignore ese tipo de palabras no solo se conseguirá un clasificador más

rápido, sino que creeremos que generalmente se obtendrá una precisión mayor que conservando la dimensión original.

El método de **Análisis de Componentes Principales** permite realizar esto empleando los conceptos de Covarianza y *Descomposición en Valores Singulares*, y trabaja generando una base de vectores ortogonales m-esima alternativa a la canónica, de forma que cada vector tiene asociado un valor real que indica su "peso.en cuanto a cuanta información transmite, de forma que siendo M una matriz simétrica derivada de la Database, tal que $m_{i,j}$ indica la covarianza entre las palabras i'esima y j'esima del Vocabulario, podemos diagonalizarla como $M = V.D.V^t$, con D una matriz Diagonal conteniendo los autovalores de M (ordenados por magnitud decreciente) y V tal que su columna i'esima contiene el autovector asociado al autovalor $d_{i,i}$.

Asi, V puede interpretarse como una transformación lineal que proyecta los vectores m-dimensionales que componen nuestras instancias, de la Base Canónica a la Base definida por los autovectores ortonormales de M. Y si defino una matriz $V_r \in R^{n.r}$ con $r < m$, tal que la columna i de V_r es igual a la columna i de V, estaríamos definiendo una transformación lineal que proyecta nuestros BoW m-dimensionales en el Espacio r-dimensional cuya base son los primeros r autovectores ortonormales de M; dado que priorizamos los autovectores asociados a los mayores autovalores de M, esta transformación $R^m \to R^r$ sera la que conservara la mayor cantidad de información relevante de entre todas las posibles; inclusive, si se elije un valor adecuado para r, es posible que la transformación disminuya el ruido de nuestra Database, generando una matriz que represente mejor el comportamiento real de las entidades estudiadas.

El procedimiento que elegimos para generar nuestra matriz de transformacion V_r consiste en:

- A partir de nuestra Database $X \in R^{n.m}$ generamos una matriz $M \in R^{m.m}$ a través de la multiplicación matricial $M = \frac{1}{\sqrt{n+1}} X^t.X$, tal que $m_{i,j}$ indica la covarianza entre las palabras i'esima y j'esima del Vocabulario en nuestra Database.

- Buscamos la diagonalización $M = S.D.S^{-1}$ (debe existir, ya que M es simétrica). Para encontrar los autovalores y autovectores que formarán D y S, aplicamos múltiples iteraciones del Método de la Potencia y un método de deflación a nuestra matriz M.

- Elegimos un número r de autovectores de M que conservaremos para definir V_r; ya que es casi imposible conocer de antemano qué valor exacto será el que nos dará el mejor balance precisión/performance, la forma más practica para hallarlo en nuestra implementación es efectuar los procesos de entrenamiento, PCA y testeo de precisión para múltiples combinaciones de valores para k (número de vecinos en KNN) y r para encontrar cuales nos dan los mejores resultados.

- Proyectamos cada BoW en nuestra Database según la transformación definida por V_r, a través de la operación $X_r = X.V_r$. Este $X_r \in R^{n.r}$ será nuestra nueva Database de entrenamiento para usar en KNN.

- Para clasificar una nueva instancia, primero le aplicamos a la BoW x la transformación $x_r = x.V_r$, luego buscamos sus k vecinos mas cercanos en la Database reducida X_r.

2.4. Método de la potencia y Deflación

Como mencionamos en 2.3, vamos a necesitar eliminar la redundancia en los datos para conseguir mejores resultados. Para esto, vamos a necesitar encontrar los primeros n autovalores y autovectores de la matriz de covarianza. El método de la potencia es una forma sencilla de lograr esto. Para poder utilizar el método se necesita pedir de antemano que la matriz tenga un autovalor mayor (en modulo) que el resto de los autovalores.

Este método se basa en construir una función que, dado un vector elegido arbitrariamente (en nuestro aleatoriamente) lo haga tender hacia un vector que sea el autovector asociado al mayor autovalor. Algo importante a tener en cuenta es que este algoritmo logra que un vector cualquiera tienda al autovector asociado al mayor autovalor en iteraciones infinitas, algo obviamente imposible computacionalmente. Por esta razón, es importante considerar un buen **criterio de parada**, que nos permita saber si nos estamos acercando suficientemente a la solución buscada.

Una vez encontrado este autovector, encontrar el autovalor al que esta asociado es sencillo, simplemente se puede despejarlo de la ecuación de autovalores conocida. El algoritmo del Método de la potencia es el siguiente.

El criterio de la parada lo explicaremos en detalle en la experimentación.

Algorithm 1 Metodo de la potencia

```
1: procedure metPot(conj S, int W)
2:     vector v ← vectoraleatorio
3:     for i from 0 to last iteration do
4:         v ← Mv / ||Mv||
5:         If (Mi Criterio De De paradas se cumple) Break
6:     λ ← v.tXv / v.tv                          ▷ Aqui v.t es el transpuesto de v
7:     Return λ y v
```

Habiendo definido este algoritmo, tenemos una forma de saber el autovalor mas grande (en valor absoluto). Pero como mencionamos anteriormente, no solo necesitamos un autovalor de nuestra matriz de covarianza, podemos llegar necesitar muchos. Para esto vamos a utilizar la técnica de deflación, que consiste en dado un autovalor λ, de una Matriz M conseguir otra matriz N que tenga como autovalores (y autovectores asociados) a todos los de M, sin contar a λ, además de agregarle el autovalor 0. Esto quiere decir que, de utilizar el método de la potencia para conseguir el autovalor mas grande, podemos utilizar el método de deflación para conseguir una matriz que no tenga este autovalor, así permitiéndonos volver a utilizar el método de la potencia iterativamente.

Algo importante a mencionar es que para poder realizar deflación es necesario saber de antemano que tenemos una base ortonormal de autovectores. Es importante también saber que los autovalores de la matriz sobre la cual estamos aplicando el método son diferentes a 0. Estas dos cosas se cumplen al trabajar con una matriz simétrica, y nosotros siempre lo haremos.

El método en si, dada una matriz M la transforma de la siguiente manera:

$M = M - \lambda * v_1 * v_1.t$

Esta matriz, cumple con lo estipulado mas arriba.

Dado este método de deflación sumado al método de la potencia podemos definir el siguiente algoritmo que nos encuentra los primeros a autovalores de una matriz con sus autovectores asociados.

Algorithm 2 Conseguir Primeros a autovalores

```
1: procedure Conseguir Primeros a autovalores(Matriz M, int A)
2:     vector autoValores ← vectorvacio
3:     Matriz autoVectores ← matrizvacia
4:     for i from 0 to a do
5:         v[i] ← Obtener mayor autovalor usando metPot
6:         m[i] ← Obtener su autovector asociado
7:         M ← M = M − λ * v1 * v1.t
8:     Return v y M
```

3. Experimentacion

3.1. La precisión del método de la potencia

El principal problema que nos trae el método de la potencia es el siguiente: Al ser un método no exacto, que converge al autovalor de mayor modulo de una matriz en infinitas iteraciones, necesitamos saber de forma confiable con cuantas iteraciones

3.2. PCA y el criterio de la parada

Al implementar el método de la potencia, es normal preguntarse cuando consideramos que los autovectores que estamos construyendo se acercan lo suficientemente a los autovectores reales de la matriz.

Habiendo hecho esto, notando que las iteraciones que se realizan al final del algoritmo producían cambios muy menores decidimos definir un criterio de la parada que nos permita saber cuando la multiplicación A.b da resultado un vector de ángulo muy similar al de b.

Para nuestra implementación elegimos estudiar el ángulo entre el vector antes y después de multiplicarlo por A, ya que el cambio de norma del vector (el que corresponder a al autovalor asociado) no ofrece mucha

información para saber que tan cerca estamos de alcanzar los valores reales de autovalor y autovector; si la modificación del ángulo es menor a 0,0001, consideramos que nuestro autovector ya esta lo bastante cerca del autovector real.

Existe, sin embargo, un posible escenario en que este sistema de corte se rompa: si nuestro vector aleatorio esta MUY cerca de ser perpendicular al autovector que buscamos, es posible que la primer multiplicación modifique su ángulo en una magnitud menor a 0,0001, en cuyo caso el método de la potencia terminaría luego de 1 iteración sin estar ni remotamente cerca de haber encontrado el autovector. Dado que las posibilidades de que se presente un caso así son extremadamente bajas, tanto desde un punto de vista teórico como demostrado por múltiples iteraciones del método en nuestras Experimentaciones (en las cuales nunca se presento una situación similar), consideramos que tal sistema de corte era adecuado para nuestro algoritmo.

Para estudiar el error que nuestro algoritmo produce, decidimos plantear los siguientes experimentos. El primero consiste en, teniendo una matriz sobre la cuál sabemos a priori cuales van a ser sus autovalores, calcular con nuestro algoritmo los mismos, analizando la diferencia en módulo.

Teniendo en cuenta que sabemos los autovalores de cualquier matriz diagonal, decidimos generar una matriz poniendo en su diagonal valores elegidos aleatoriamente. En particular, para los valores en la diagonal usamos una distribución uniforme entre 0 y 100. Al ser los valores de la diagonal los autovalores de la matriz, resulta sencillo comparar la diferencia entre los autovalores reales y los aproximados. Algo importante a mencionar es que en PCA solamente utilizamos verdaderamente los autovectores de la matriz de covarianza, no los autovalores, pero como nuestro algoritmo despeja sencillamente los autovalores a partir de los autovectores, al analizar el error entre los autovalores podemos tener una muy buena idea de que tan importante es el error del algoritmo.

Es importante tener en cuenta además que, al utilizar deflación es posible que el error se vaya acumulando iteración a iteración. Por esto consideramos correcto correr el algoritmo con matrices de diferentes tamaños, analizando el crecimiento. El siguiente gráfico muestra el error promedio entre los autovalores reales de una matriz generada aleatoriamente y entre los autovalores resultantes de utilizar el método de la potencia. Otra aclaración relevante es que, además del criterio de la parada, pusimos un tope de 1000 iteraciones en el algoritmo.

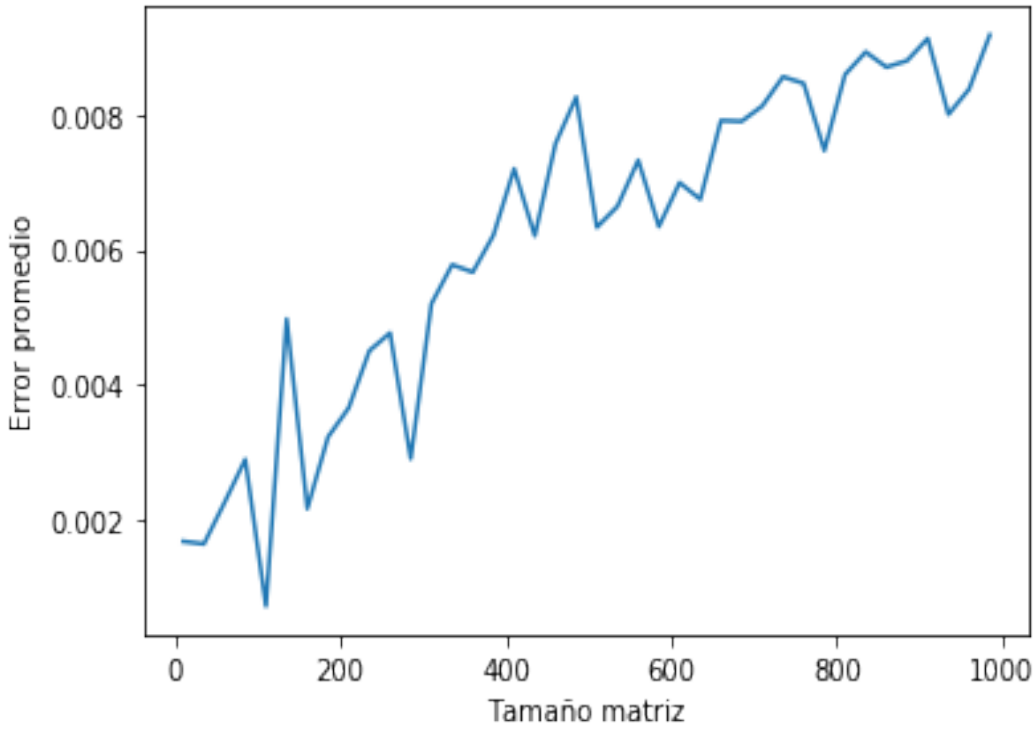


Figura 1: *Error promedio en el calculo de los autovalores*

Como se puede observar en 1, el error no parece ser muy estable. Sin embargo, es evidente que el error es pequeño relativamente, nunca sobrepasando 0,01 en matrices de hasta 1000 por 1000.

Es claro sin embargo que el error crece a medida que las matrices lo hacen, habiendo una tendencia evidente. Viendo que este error no es despreciable, y peor aun, es tan inestable, corrimos el algoritmo

de nuevo, pero esta vez con diferentes parámetros. Al ver que el error es inestable, decidimos en primer instancia aumentar el numero de iteraciones máximas del algoritmo de 1000 a 3000. Nuestras hipótesis en este experimento es que el error promedio va a disminuir considerablemente. A continuación mostramos los resultados obtenidos.

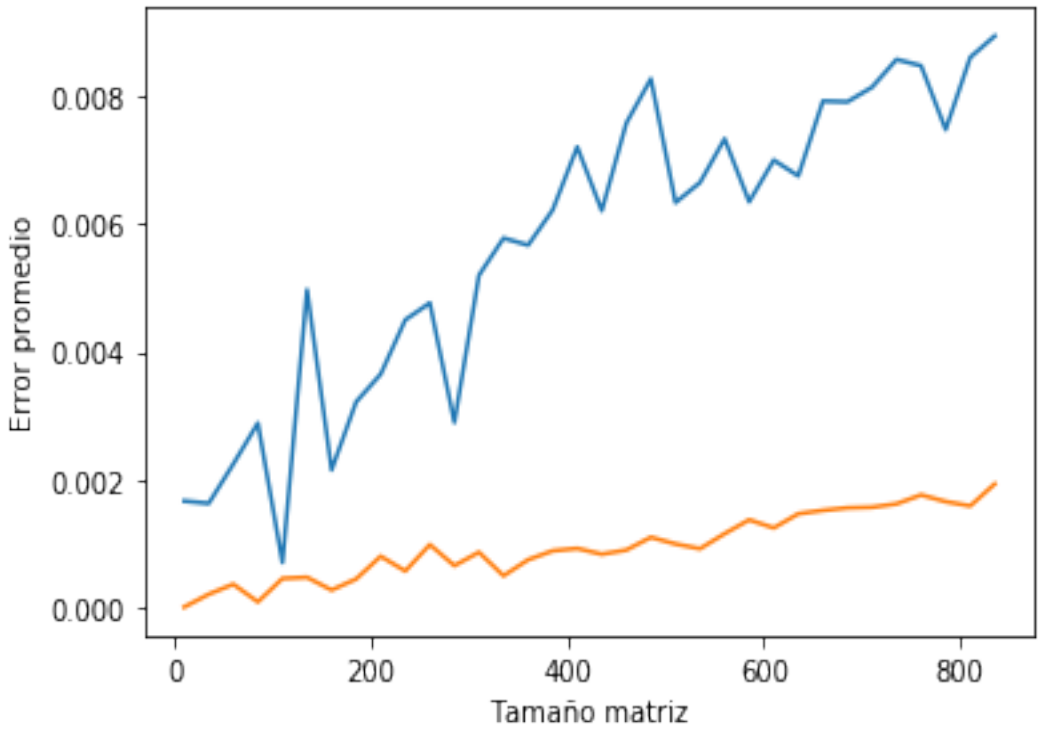


Figura 2: *Error promedio en el calculo de los autovalores*

Como se puede ver, el error es muchísimo mas bajo, nunca superando la barrera de 0,002. Esto significa en parte que el error antes visto estaba en gran parte producido por la cantidad de iteraciones máximas elegidas. Algo que es importante mencionar es que en 2 trabajamos con matrices cuadradas de hasta 850 filas, simplemente por notar que el experimento estaba tardando demasiado y que con los resultados obtenidos la tendencia era clara.

3.3. Encontrando la cantidad de vecinos óptima

Sabiendo que el algoritmo de KNN depende muchísimo de la cantidad de vecinos con la que realicemos la votación, el primer parámetro que vamos a analizar es este. Decidimos plantear el siguiente experimento variado solamente la cantidad de vecinos como parámetro, considerando desde la cantidad mínima posible de vecinos hasta la máxima (en nuestro caso particular, al tener 6275 criticas en nuestras instancias de Training, tomamos los vecinos en el rango [1, 6275]).[1]

Es sencillo notar que el porcentaje de aciertos sera bajo tanto considerando pocos vecinos como considerando demasiados. De tomar la cantidad máxima posible de vecinos, por ejemplo, el algoritmo simplemente devolvería positivo si la mayor parte de nuestros elementos pertenecientes al conjunto de Training son positivos y devolvería negativo en caso contrario. De forma similar, al tener en cuenta muy pocos vecinos, sospechamos que el porcentaje de aciertos sera bajo, ya es evidente que cualquier critica que pueda ser considerada un 'outlier', es decir, que sea negativa o positiva pero muy similar a otras criticas opuestas podría llegar a afectar los resultados de KNN considerablemente.

[1] Algo importante a tener en cuenta es que realizamos los primeros experimentos usando los tamaños de Train de exactamente la mitad de las criticas. Mas adelante en 3.4 experimentamos cambiando estos tamaños

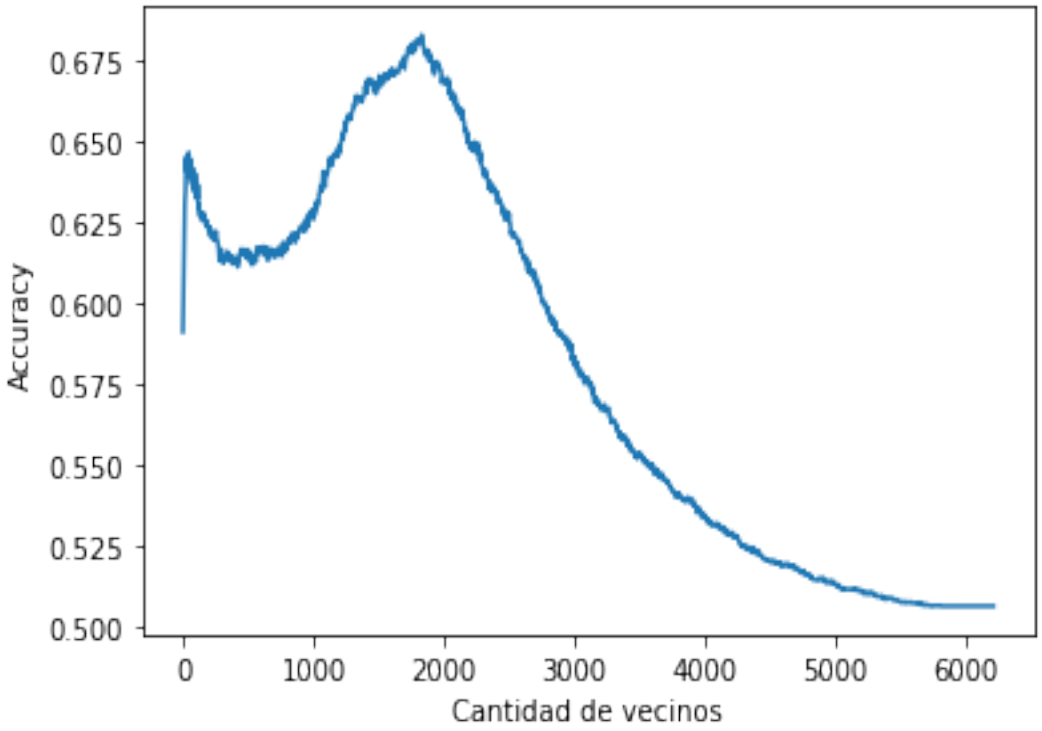


Figura 3: *Accuracy* de Knn variando la cantidad de vecinos considerados entre 1 y 6275

Como se puede ver en 7 nuestras hipótesis iniciales se cumplieron. Es sencillo ver que con un K muy grande el porcentaje de acierto disminuye considerablemente al acercarnos a el máximo posible de vecinos, estancándose en 0.506454. Este numero es exactamente el porcentaje de criticas negativas en nuestro conjunto de Test, algo que, teniendo en cuenta que hay mas criticas negativas que positivas en nuestro conjunto de Training, es mas que lógico.

Es importante mencionar que al correr el algoritmo, realizamos saltos de a 5 vecinos a la vez.

Es interesante ver como se alcanzan dos máximos locales. Uno se alcanza al correr el algoritmo considerando los 1830 vecinos mas cercanos (alcanzando una precisión de 0.6830278884462151) y el otro se alcanza considerando exactamente los primeros 100 vecinos (alcanza una precisión 0.61434262948).

3.3.1. Encontrando el alfa óptimo

Como desarrollamos anteriormente, nos propusimos implementar una transformación a nuestra base de datos a través del método de PCA, de forma de reducir los tiempos de cómputo e, idealmente, aumentar la precisión de nuestro clasificador al priorizar representaciones de nuestra Database que conserven la mayor cantidad posible de la información más relevante. Para ello, queremos encontrar una cantidad de autovectores de la matriz de covarianza a buscar (en adelante llamaremos a esta magnitud α), la transformación resultante del Database, para algún valor o rango de valores de k, nos da una precisión mayor que para cualquier otra combinación de α y k. Como no podemos asegurar que un pico de precisión para un valor de α (o para el clasificador sin PCA) aparezca también para otro valor de α distinto, debemos testear nuestro calificador en un rango extenso de k's para cada alfa, para comparar valores y posiciones de los picos de precisión, y luego estudiar mas detenidamente los rangos de k en los que uno o mas alfas alcanzan las mayores precisiones posibles. Realizamos dicho estudio para dos clasificadores: El clasificador con votación por Moda (KNN Simple) y el clasificador con votación Distance-Weighted (KNN por peso):

- KNN por moda:

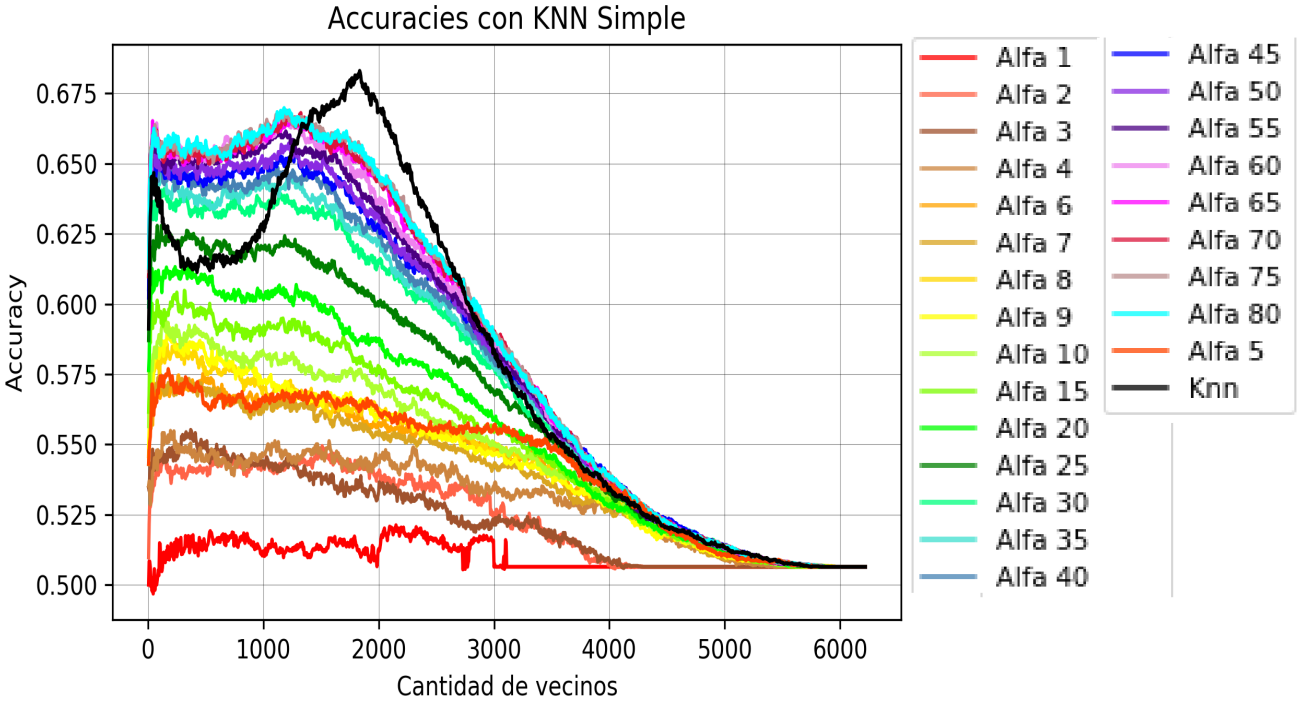


Figura 4: *Accuracy* de Knn con votacion por moda, con y sin PCA

Tal y como esperábamos, el accuracy del clasificador sin PCA cae de forma MUY rápida para valores altos de k en votación por moda, ya que al incluir porciones muy grandes del Dataset a la hora de votar, estamos acumulando información que no es relevante para la clasificación: mientras mas se acerca k al total de elementos del Dataset, más se desvía la votación hacia la clase con la mayor cantidad de instancias en el mismo (en este caso, la clase de instancias positivas). Al acercarse a los 6000 vecinos, esta situación alcanza su punto limite, en el cual tantas instancias son incluidas en la votación que TODAS las reviews son clasificadas como positivas, resultando en una precisión de 0.52 que indica la total disfuncionalidad del clasificador.

Pudimos ver que, efectivamente, la aplicación del método de PCA incrementa el accuracy del clasificador en varios rangos de k (aunque no tantos, ni en tal magnitud, como esperábamos inicialmente); Asimismo, vimos un incremento significativo en la velocidad de ejecución para todos los k, la cual pudimos incrementar aun mas implementando una versión alternativa de nuestro KNN que explicamos mas adelante. Podemos ver que el desempeño general es marcadamente inferior al del clasificador sin PCA para todo valor de $\alpha < 20$, lo que es razonable ya que las primeras columnas de la transformación de PCA corresponden a los autovalores mas grandes de su matriz de covarianza, por lo que ignorarlos implica perder muchísima información relevante para la clasificación.

También, tal como esperábamos, se puede ver que la inclusión de columnas correspondientes a autovalores mas chicos afecta cada vez menos la precisión del clasificador (si bien nunca llegan a afectarlo negativamente), hasta el punto en el que, para α's superiores a 60, la diferencia es prácticamente inexistente (aunque el tiempo de computo seguirá en aumento, por incrementar la cantidad de columnas de la Dataset transformada).

En cuanto al hecho que el clasificador sin PCA alcanza un pico de accuracy mayor que el de todos los PCA, lo justificamos de la siguiente manera: La transformación de PCA inevitablemente hará que se pierda cierta cantidad de información por cada review transformada; nosotros asumimos que esa ignorar información, por ser poco relevante o incluso perjudicial, mejoraría la precisión del clasificador, pero es razonable pensar que esa información, también, aporte datos relevantes a la hora de clasificar, la cual posiblemente solo influya significativamente cuando se busca un gran número de vecinos. Si bien usar α's alfas mas y mas grandes implicaría perder menos información y llevaría la curva de precisión m'as cerca a la de no-PCA (como podemos ver experimentalmente), ya que los autovalores correspondientes a alfas altos están muy cerca de 0, podemos esperar que, por la naturaleza de la representación de floats en la maquina y la implementación de nuestro método de la potencia, los errores de cada autovalor

calculado (y por consiguiente, su autovector que usaremos en la transformación) sea mas significativo, por lo que, por más que elijamos alfas muy altos, la perdida de información hará que el clasificador con PCA nunca llegue a igualar el desempeño del clasificador sin PCA.

- KNN por peso:

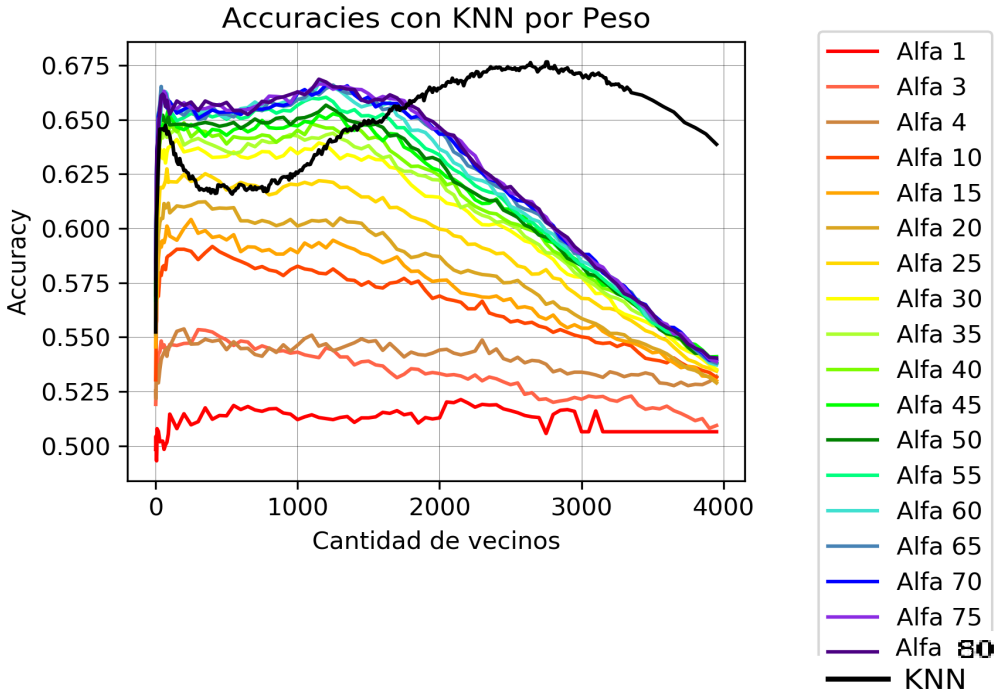


Figura 5: *Accuracy* de Knn con votación por peso, con y sin PCA

Como habíamos esperado, la votación por peso afecta la precisión del clasificador de forma mas notable para valores más altos de k; Ya que se disminuye el impacto a la hora de votar mientras mas lejano está una fila del Dataset con respecto a la review a clasificar, el clasificador ya no se rompe por .[en]globarüna porción demasiado alta del Dataset como ocurrió con la votación por Moda. Esta propiedad (en la que todos los k-vecinos dan información relevante a la hora de clasificar una review, con un impacto proporcional a su similitud a la misma) es la que, asumimos, explica que el clasificador (sin PCA) vea su máxima precisión en un rango MUY elevado de k, entre 2000 y 3500.

Por otro lado, el comportamiento del clasificador con PCA parece haberse modificado MUY poco con respecto a su comportamiento con votación por moda. Suponemos que, como resultado de la transformación de PCA (que debería .[a]grupar"los vectores que son nuestras instancias acorde a su clase), el efecto de realizar la votación por peso (que es especialmente útil cuando se tienen pocas instancias muy cercanas con la misma clase que queremos clasificar,y muchas mas lejanas con la clase opuesta) deja de ser tan influyente y modifica en poco la precisión del clasificador.

La relación entre el clasificador con y sin PCA sigue siendo muy similar al que se presentó en los casos de votación por moda. Las únicas diferencias notables son que se redujo la diferencia de magnitud entre los mayores picos de accuracy con PCA y el de no-PCA, y que el pico de no-PCA , si bien es mas ancho que antes (lo que implica que tenemos un mayor rango de K's para elegir tal que nuestro accuracy sea óptimo) también se movió para valores mas altos de k, lo que implicaría que obtener buenos resultados solo sería posible incrementando los tiempos de cómputo (ya que la diferencia de tiempos para el clasificador con votación por moda y por peso son insignificantes).

Concluimos que, si bien existen rangos de k en los cuales el Clasificador sin PCA supera cualquier nivel de precisión alcanzado por el Clasificador con PCA, estos acarrean también un tiempo de cómputo mucho mayor que los mismos (no solo por el alto valor de k que requieren, sino por el costo temporal inherente al Clasificador por KNN sin PCA); Como discutimos mas adelante en la sección 3.4, el costo temporal relacionado con generar la matriz de transformación de PCA hace que usar la misma solo ofrezca una ventaja

temporal cuando la cantidad de reseñas a evaluar es MUY alta. Y dado que las diferencias de accuracies máximos entre los clasificadores con PCA (con $\alpha > 60$) y el clasificador sin PCA son bastante bajas, consideramos que el Clasificador con PCA es una mejor elección en la mayoría de los casos (particularmente si usaremos siempre la misma Dataset de Entrenamiento, de forma que solo haga falta buscar y aplicar su Transformación una vez).

En situaciones distintas a las descritas anteriormente, consideramos que ambos KNN sin PCA (con votación por Moda y por Peso) son m'as adecuados para distintos casos: Cuando la prioridad son los tiempos de cómputo o se requiere una precisión lo mas alta posible, elegiríamos el KNN por Moda, ya que sus pico requiere un k menor y alcanza un valor mas alto; cuando se quiere asegurar que el accuracy sea, en el peor de los casos, lo mas alto posible, elegiríamos el KNN por peso, ya que sus accuracies mínimos con mas altos y muestra variaciones de accuracies según k mucho menos marcadas.

3.4. Variación del Accuracy para distintas cantidades de Instancias de Entrenamiento

A continuacion, habiendo encontrado rangos de k y valores de α que nos permiten alcanzar nuestros máximos valores de Accuracy, quisimos estudiar como se ven modificados dichos valores si la matriz de entrenamiento se ve reducida. Para ello, tomamos la implementación del Clasificador que consideramos mas predecible y con los mejores resultados (valores de Max Def = 0.9 y Min Def = 0.01 para el Vectorizador, $\alpha = 80$ para PCA y un k entre 0 y 6000) cuyo desempeño sin modificar la cantidad de instancias de entrenamiento estudiamos en la sección 3.3.1 . Así, estudiamos los valores de accuracy en todo el rango de valores de k para distintas reducciones de la Matriz de instancias de entrenamiento (la cual redujimos tomando las primeras i filas, asumiendo que las clases positivas y negativas estaban distribuidas de forma equitativa), obteniendo los siguientes resultados:

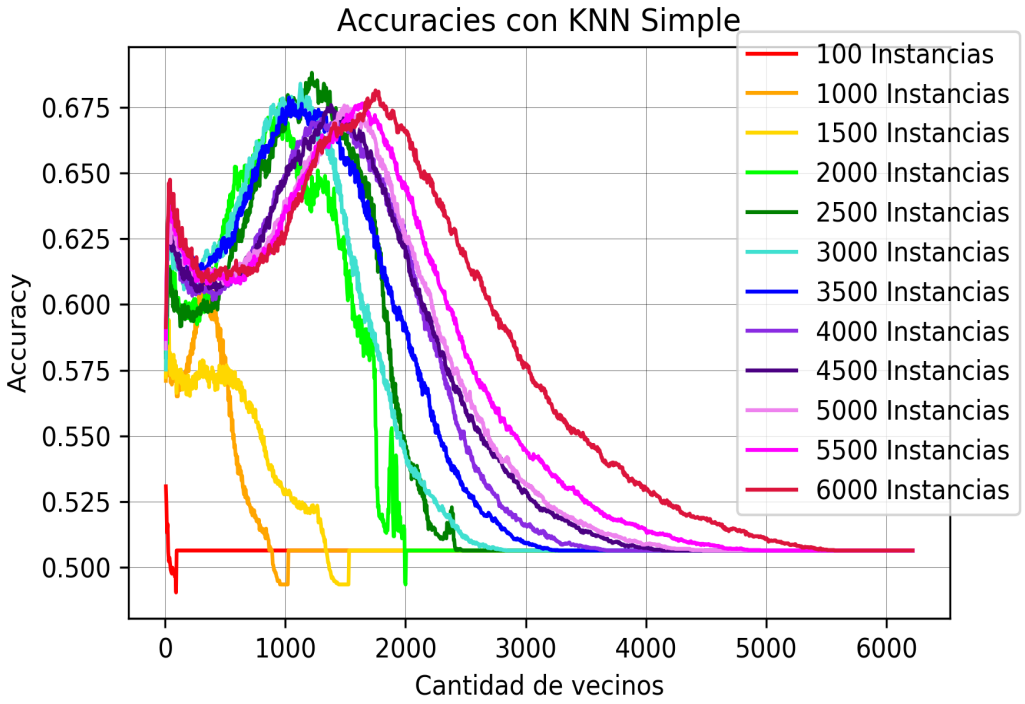


Figura 6: *Accuracy* de Knn sin PCA variando la cantidad de instancias de entrenamiento usadas.

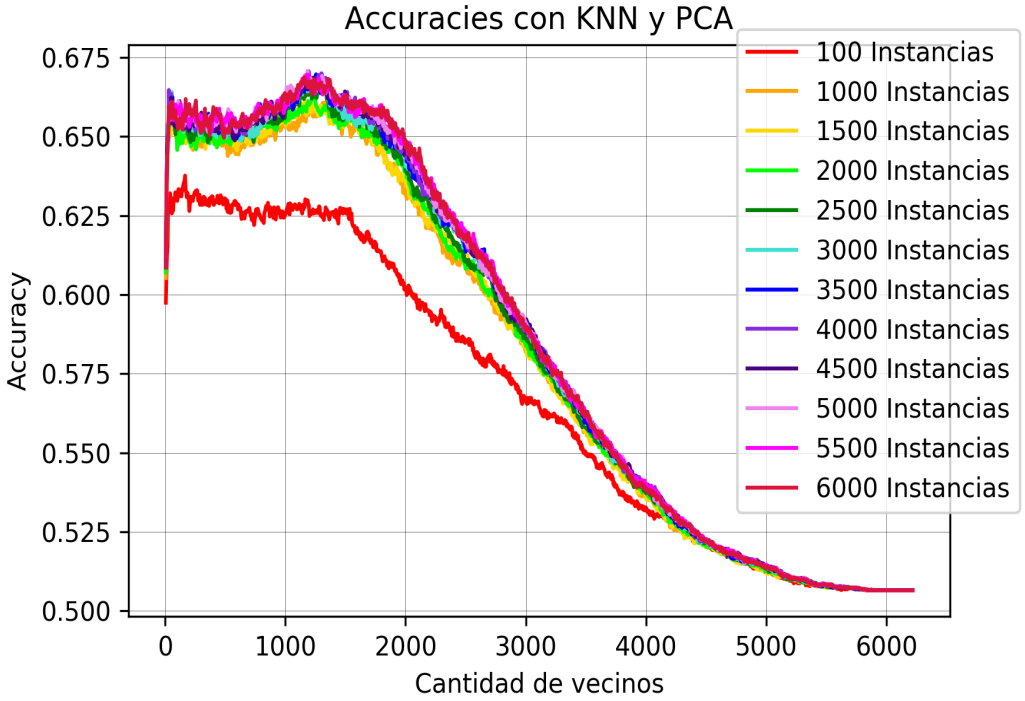


Figura 7: *Accuracy* de Knn con PCA variando la cantidad de instancias de entrenamiento usadas.

Para el Clasificador sin PCA,vemos que el Clasificador solo empieza a comportarse de forma predecible y dar buenos resultados para cantidades de instancias mayores a 2000. A partir de las 3500, los clasificadores parecen demostrar comportamientos muy similares, pero con reducciones de accuracy ocurriendo para k mas bajos mientras menos instancias se toman en cuenta (lo que es razonable, ya que a mayor cantidad de instancias equitativamente distribuidas, mayor es el k que puedo elegir antes de que el fenómeno de .[en]globamiento de clases.[a]fecte negativamente la precisión del mismo). Si bien el clasificador con 2500 instancias parece ser el que ofrece el mayor pico de precisión, elegimos trabajar con 6000 instancias en los estudios posteriores basados en que el área entre las curvas correspondientes a sus accuracies tal que la de 2500 instancias es mayor o igual a la de 6000 instancias, es menor que el área entre las curvas cuando la de 6000 instancias es mayor o igual a la de 2500. Matemáticamente, interpretamos esto como que, si bien **existen** rangos de k para los cuales elegir 2500 instancias daría mejores resultados que elegir 6000, inclusive alcanzando picos más altos, el desempeño del Clasificador con 6000 instancias será superior (para cualquier número de instancias menos a 6000) en un rango mayor de k, y en una magnitud mas significativa. Dicho de otra manera, el Clasificador con 6000 instancias sera "generalmente mejor"que todos los demás, lo que nos da mayor seguridad de sus resultados a la hora de evaluar otros parámetros.

En cuanto al Clasificador con PCA, se puede ver que, mas allá del caso de solamente 100 instancias de entrenamiento, todos los Clasificadores con distintas cantidades de instancias tienen un comportamiento casi idéntico, lo cual justificamos con la hipótesis que, dado que la transformación de PCA, bien implementada, proyectaría a las mismas en un Espacio de menor tamaño, en el cual instancias similares y que comparten clases estarán agrupadas en çlusters", el aumentar la cantidad de instancias de entrenamiento solo aumentaría el tamaño de dichos clusters sin cambiar sus posiciones, o distancias, uno respecto al otro, tal que el comportamiento de los clasificadores para las mismas reviews diferirá muy poco.

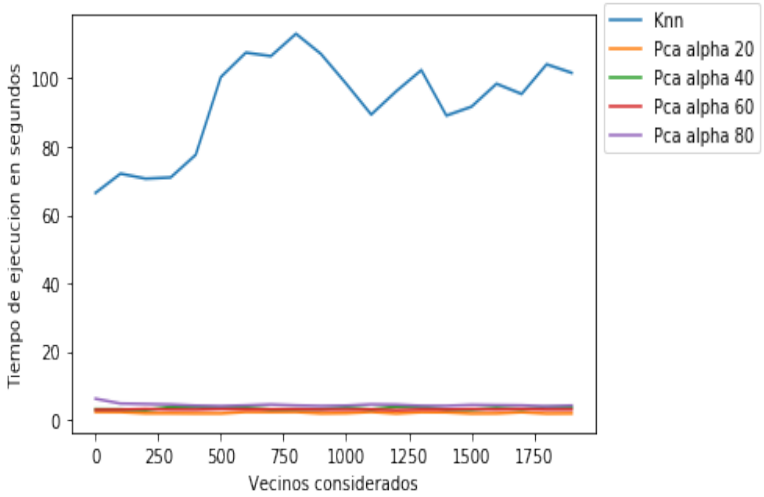
Ya que el Clasificador con 6000 instancias de entrenamiento, nuevamente, esta ligeramente por encima de todos los demás en términos de accuracy, decidimos usarlo para nuestros posteriores estudios sobre el Clasificador con PCA.

3.5. Tiempos de ejecución, PCA vs KNN simple

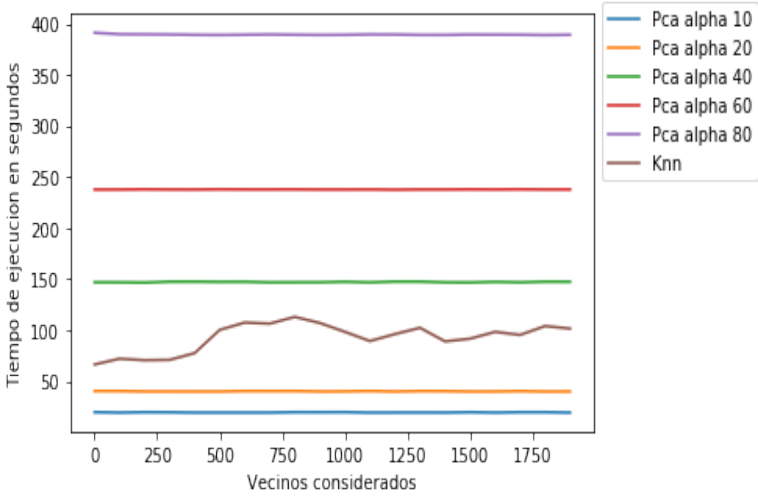
Habiendo analizado las diferencias entre la eficiencia de KNN optimizado con y sin PCA, resta analizar la diferencia en los tiempos de ejecución entre ambos algoritmos. Es sencillo ver que KNN, una vez que se elimina la redundancia de sus conjuntos de Test y Training, es mucho mas rápida en su ejecución, al menos en teoría, ya que se disminuye el tamaño de la base de datos en la cual buscar los vecinos mas cercanos. Esto,

evidentemente, va a tener mayor impacto utilizando alphas menores (con alpha nos referimos a parámetro de PCA). Sin embargo, es importante tener en cuenta que realizar proyección ortogonal de PCA es un algoritmo a priori costoso. Nuestra hipótesis inicial es que al comparar los tiempos de ejecución sin tener en cuenta el costo temporal de PCA el rendimiento del segundo va a ser mucho mejor.

Teniendo en cuenta que el tiempo de ejecución de KNN debería crecer a medida que el parámetro de K (cantidad de vecinos) crezca [2], decidimos analizar los tiempos de ejecución variando este parámetro. Aclaramos que no consideramos los tiempos de realizar la proyección de PCA, si no simplemente los tiempos de ejecución de KNN una vez hecha la proyección. A continuación mostramos los resultados de realizar las corridas, primero teniendo en cuenta el tiempo de realizar la proyección ortogonal de PCA y luego sin tenerlo en cuenta.



(a) Tiempo de ejecución de los algoritmos,sin considerar el tiempo de preproceso de PCA



(b) Tiempo de ejecucion de los clasificadores con PCA, teniendo en cuenta los tiempo de preprocesamiento del Dataset

Figura 8: Tiempos de ejecución de KNN corrido con diferentes Alphas

En 8 podemos ver como, primero, el tiempo de ejecución de KNN sin utilizar PCA es ampliamente superior al de utizando PCA, esto es sin tener en cuenta el tiempo de preproceso. La razón por la que sucede esto es sencilla, al haber achicado las instancias de Training y test con PCA, el algoritmo es notoriamente mas rápido. Al tener en cuenta sin embargo el tiempo que tarda en realizarse el preprocesamiento de PCA, se puede ver fácilmente que de querer realizar PCA para solamente una instancia de K sin necesitar otras, utilizando PCA con alphas de 60, 80 y 40 ya no seria productivo en absoluto. Es claro en estos gráficos que el tiempo constante que lleva realizar la transformación de PCA es mucho mas grande de lo que de tarda, en una vez hecho esto, simplemente correr KNN.

Es evidente además que el tiempo de preproceso de PCA aumenta rápidamente al aumentar el alpha, probablemente debido a que requiere del calculo de muchos mas autovectores. A continuación mostramos los tiempos de ejecución del gráfico de arriba a a la izquierda, eliminando KNN sin del mismo para lograr un analisis mas fino de los tiempos de ejecución de PCA variando los alpha.

[2]Realizamos nosotros, a propósito de acelerar la corrida de KNN, un algoritmo que realiza rápidamente la corrida de varios K, simplemente aprovechandose del hecho de que habiendo calculado los k vecinos mas cercanos, para calcular los k+1 mas cercanos simplemente falta buscar el k+1 esimo mas cercano. Para esta experimentacion sin embargo, realizamos corridas individuaes de KNN para todo K, ya que lo consideramos mas pertinente al experimento

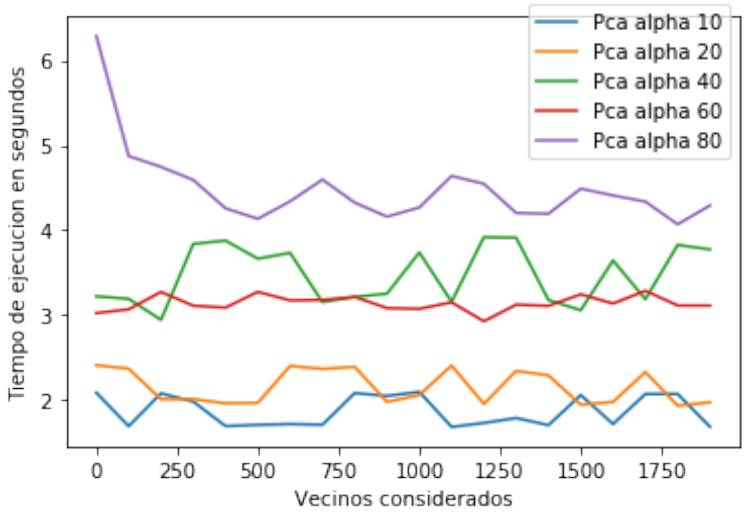


Figura 9: *Accuracy* de Knn variando el porcentaje de rechazo de palabras muy raras entre 0,00 y 1

De este ultimo gráfico se desprende que la variación del tiempo de ejecución entre un mismo alpha es muy baja a medida que aumentamos la cantidad de vecinos considerada, a diferencia que KNN simple donde vimos que claramente había un crecimiento en tiempos de ejecución considerando mas vecinos.

3.6. Cambiando los parámetros de BOW

Como mencionamos en 2.1, es importante filtrar algunas de las palabras que procesamos en cada critica, ya que tanto las mas comunes como las mas extrañas pueden a llegar a tener una influencia negativa en el porcentaje de acierto. Un ejemplo de esto es como la palabra 'a' puede ser tan común que decir que dos criticas se parecen mucho por usarla es algo que no nos serviría. De la misma forma, de tener en cuenta palabras muy raras puede llegar a ser contra productivo, ya que no nos daría una buena base sobre la cual poder comparar las criticas. Nuestra hipótesis inicial es que vamos a obtener los mejores resultados filtrando considerablemente las palabras mas comunes utilizadas, y filtrando también las palabras raras aunque quizás no en la misma medida (sospechamos que posiblemente muchos adjetivos como 'sorprendente', 'extraordinaria' o 'espantosa' serán algo raros, y no querríamos en principio eliminarlos)

Llamaremos al parámetro **maxDef** al máximo porcentaje de apariciones en diferentes criticas que cada palabra debe tener para ser rechazada. Es decir, de estar utilizando maxDef = 0,6 estaríamos rechazando todas las palabras que aparezcan en mas del 60 % de las criticas. Análogamente llamaremos **minDef** al parámetro del mínimo porcentaje de criticas en las que una palabra tiene que estar para ser rechazada. De estar considerando un minDef de 0,2 estaríamos ignorando aquellas palabras que aparecen en menos de 20 % de las criticas.

Decidimos experimentar variando independientemente el maxDef y el minDef, primero buscando maximizar el porcentaje de acierto variando un solo parámetro, y sin considerar el otro. Nuestro primer experimento entonces, consiste en simplemente variar este parámetro MaxDef entre 0 y 1, corriendo KNN en cada caso.

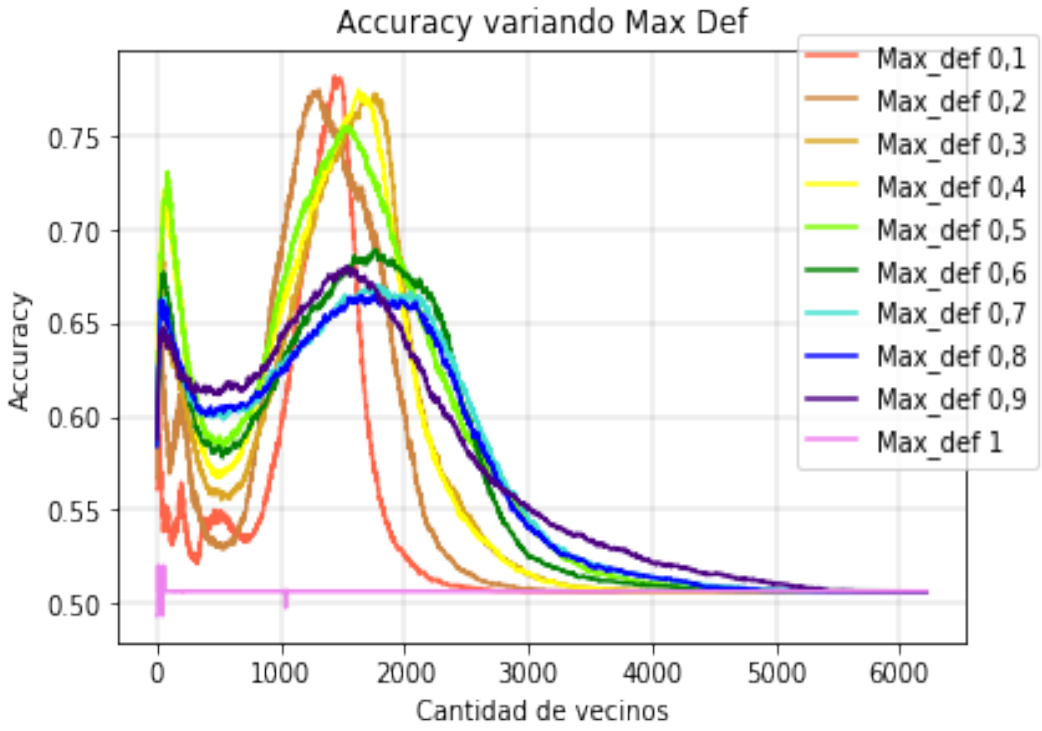


Figura 10: *Accuracy* de Knn variando el porcentaje de rechazo de palabras muy comunes entre 0,1 y 1

Los resultados fueron muy diferentes a los esperados. Primero es importante mencionar como los el porcentaje de acierto fue muy bajo ignorando las palabras que aparecen en mas de 100 % de las criticas. Esto es razonable, ya que no estamos rechazando ninguna palabra, y al tomar todas las palabras de uso frecuente no deberíamos obtener buenos resultados, como explicamos en la hipótesis.

Dejando de lado el caso de MaxDef = 1, vemos que el resto de los resultados pueden dividirse a simple vista en dos bloques, aquellos cuyo MaxDef varia entre 0,9 y 0,6 y aquellos que varían entre 0.6 y 0.1. Los primeros dieron resultados similares a los obtenidos hasta ahora, todos alcanzando picos de accuracy entre 1500 y 2000 palabras, rondando en estos picos una precisión entre 0,65 y 0,68, siendo el mas eficaz el caso de tener maxDef = 0,6 (que llega a un máximo de 0.6894). Lo mas notorio es, sin embargo, la enorme mejora que obtuvimos una vez que el maxDef supero la barrera de 0,6. La mejora fue increíblemente notoria, con cada corrida superando l barrera de 0,7 de eficacia con facilidad (cosa que hasta ahora no habíamos hecho). El mejor resultado lo obtuvimos utilizando un maxDef de 0,1 es decir, rechazando casi todas las palabras comunes, quedándonos con las que aparecen solamente en menos del 10 % de las criticas (el valor alcanzado fue un inesperado 0,7823) . Teniendo en cuenta como la tendencia de ir disminuyendo el MaxDef parecía estar dando los mejores resultados, corrimos disminuyendo este parámetro aun TILDE mas, sin obtener mejores resultados que los aquí mostrados.

A pesar de parecer estos resultados extraños a simple vista, consideramos que los resultados fueron tan buenos tomando solamente palabras poco usadas porque muchas palabras raras del nicho cinematográfico (por ejemplo, 'Neorealismo', 'expresionismo', 'surrealista') pueden darnos una buena forma de comparar criticas entre si, además de considerar que los adjetivos tanto positivos como negativos que una película puede tener pueden llegar a no ser palabras muy frecuentes.

Habiendo realizado este experimento, corrimos el análogo, esto es, considerando MinDef entre 0 y 1 y sin utilizar MaxDef en lo mas mínimo.Los resultados fueron los siguientes.

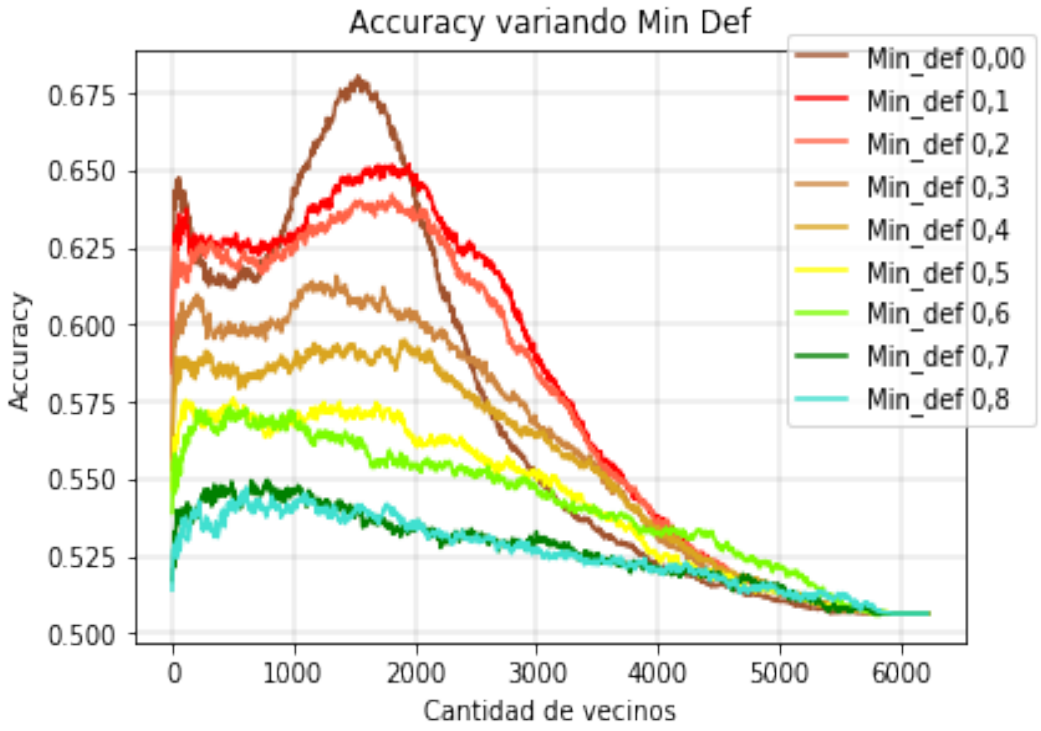


Figura 11: *Accuracy* de Knn variando el porcentaje de rechazo de palabras muy raras entre 0,00 y 1

Como se puede ver en 11 los resultados no fueron tan positivos como los anteriores. El máximo entre todas las variaciones de Min Def se alcanzo poniéndolo en 0,00, es decir, tomando absolutamente todas las palabras menos comunes (alcanzo aquí una precisión de 0.6804780876494024) . Esto de nuevo nos confirma lo sospechado anteriormente, considerar las palabras poco frecuentes parece solamente incrementar la precisión, no decrementarla. Se puede observar también en el gráfico como a medida que el parámetro Min Def se acerca a 1, se pierde la precisión considerablemente. Esto creemos que es porque al estar teniendo en cuenta solamente las palabras mas comunes al buscar las criticas mas "similares"verdaderamente no hay un buen parámetro de búsqueda, estaríamos quedándonos con palabras que posiblemente en su mayoría sean pronombres, preposiciones, o similares.

Habiendo encontrado los parámetros que mejores resultados fijando uno de los parámetros, nuestro siguiente paso fue ver si modificando el otro parámetro, dados los máximos encontrados, ayudaba a mejorar la eficacia.

Siendo mas especifico, para el parámetro de Max Def encontrado (0,1), vamos a variar el Min Def en el intervalo [0.00, 0.09]. Es importante notar que no podemos, dado el Max Def encontrado, variar el Min Def por fuera de ese intervalo (de tener un valor superior al de Max Def estaría ignorando todas las palabras).

Algo interesante es que, de intentar hacer eso pero fijando el Min Def en el máximo encontrado hasta el momento, (0), estariamos repitiendo el mismo experimento realizado en 10, ya que en ese momento fijamos el Min Def en 0 simplemente para no tenerlo en cuenta.

Habiendo dicho eso, a continuación mostramos los resultados de correr KNN fijando el Max Def en 0,1, y variando el Min Def en el intervalo mencionado. Los resultados mostrados en 12 muestran que, primero, aquellos Min Def que se acercan mucho al valor de Max Def que fijamos dan los peores resultados. Esto es razonable ya que estaríamos en estos casos quedándonos con demasiadas pocas palabras, aparentemente fuertemente afectando la eficacia. A medida que nos vamos acercando a 0 sin embargo, van mejorando considerablemente los resultados, hasta llegar a un máximo (con Min Def igual a 0,01) que mejora en apenas con respecto al máximo encontrado anteriormente. El valor alcanzado aquí es de 0,7858.

4. Conclusión

Más allá de las conclusiones y justificaciones particulares de cada ítem de la Experimentación, concluimos que, contra lo que esperábamos inicialmente, el método de PCA no incrementa de forma tan marcada la precisión del Clasificador, si bien sí mejora su desempeño general y definitivamente mejora su Costo

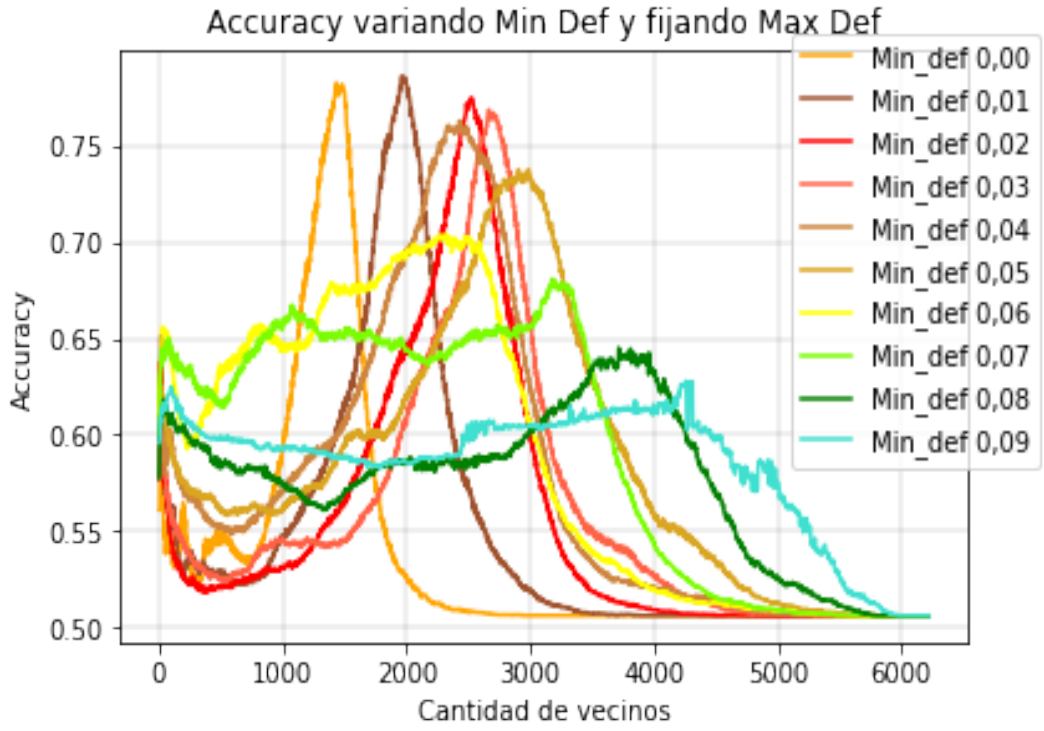


Figura 12: *Accuracy* de Knn variando el porcentaje de rechazo de palabras muy raras entre 0,00 y 1

Temporal, siempre y cuanto la cantidad de instancias a clasificar sea lo bastante grande como para compensar el costo adicional relativo a construir la matriz de covarianza y buscar sus autovalores.

Concluimos también que, si bien el funcionamiento del Clasificador esta fuertemente ligado a los conceptos de Álgebra Lineal que vimos durante el cuatrimestre, los aspectos que determinan que tan preciso resultará están mas determinados por aspectos mas sutiles (combinaciones de parámetros de alfa, k, min df y max df, cantidad de instancias de entrenamiento) y conceptos no tan relacionados con dichos temas (como por ejemplo, el desempeño de distintas métricas de distancia en dimensiones altas) las cuales, mas allá del conocimiento dado por la experiencia y el uso extenso de Clasificadores de este tipo, solo pueden conocerse con claridad a través de la experimentación sobre nuestro caso de estudio en cuestión.

Por último, hacemos notar la enorme 'maleabilidad' de este Clasificador, tanto por su capacidad de ver modificadas sus funciones internas (cálculos de distancia, búsqueda de autovalores, métricas de votación) y funcionar con parámetros (y combinaciones de ellos) en muy diferentes magnitudes de forma que nos permiten, según las necesidades lo requieran, priorizar tiempos de cómputo, desempeño general, precisión máxima y memoria usada en una gran variedad de combinaciones.

Referencias

[1] David Hand, Heikki Mannila, Padhraic Smyth. *Principles of Data Mining* MIT Press, 2001.

[2] Tom M. Mitchell. *Machine Learning* McGraw-Hill Science/Engineering/Math, 1997.

[3] Yun-lei Cai, Duo Ji ,Dong-feng Cai. *A KNN Research Paper Classification Method Based on Shared Nearest Neighbor* Natural Language Processing Research Laboratory, Shenyang Institute of Aeronautical Engineering, Shenyang, China, June 2010.

[4] Benjamin Fayyazuddin Ljungberg. *Dimensionality reduction for bag-of-words models: PCA vs LSA* Stanford, 2017.

[5] Peter Grant. *k-Nearest Neighbors and the Curse of Dimensionality* July 2019.

[6] Natasha Sharma. *Importance of Distance Metrics in Machine Learning Modelling* January 2019.

[7] Charu C. Aggarwal1, Alexander Hinneburg, and Daniel A. Keim. *On the Surprising Behavior of Distance Metrics in High Dimensional Space* Institute of Computer Science, University of Halle, Germany, October 2001.

[8] Pádraig Cunningham, Sarah Jane Delany. *k-Nearest Neighbour Classifiers* University College Dublin,Ireland, March 2007.

[9] Jianping Goua;, Lan Dub, Yuhong Zhanga, Taisong Xionga *A New Distance-weightedk-nearest Neighbor Classi er* School of Computer Science and Engineering, University of Electronic Science and Technology ofChina, Chengdu 611731, China.