DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico #2 — Análisis de Sentimiento

11 de octubre de 2019 Métodos Numéricos

Integrante	LU	Correo electrónico
Catriel Omar D'Elia	964/11	catriel.delia@gmail.com
Gian Franco Lancioni	234/15	gianflancioni@gmail.com
Joaquín Perez Centeno	699/16	joaquinpcenteno@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

Índice

1. Abstracto

Distintos sitios de reseñas cinematográficas permiten a los usuarios escribir sus propias criticas y dar una puntuación a las películas. Al combinar texto con una clasificación numérica, se conforma un conjunto de datos interesante para llevar a cabo análisis supervisado de sentimiento sobre el lenguaje de los usuarios.

En este informe se evalúa el desempeño de clasificadores basados en kNN sobre *bag of words* de reseñas de películas para clasificar el sentimiento subyacente en el texto. Adicionalmente se evaluó el desempeño de utilizar PCA en el pre procesamiento del texto. Para implementar PCA tuvimos que hacer nuestro propio método de la potencia con deflación.

Todos estos métodos requieren parámetros que afectan fuertemente el resultado en términos de precisión final como de latencia. Nos interesa encontrar aquellos que minimicen sacrificios en tal *tradeoff*.

2. Introducción Teórica

2.1. Análisis de Sentimiento

2.2. Bag of Words

Bag of Words es un sistema de representación utilizado en procesamiento del lenguaje natural. Cada documento pasa a ser representado por su multiconjunto de palabras. Se pierde el orden original de las palabras en el texto. Se utiliza en problemas donde el vocabulario del documento es suficiente para estimar el sentimiento subyacente.

2.3. k Nearest Neighbors

El algoritmo *k Nearest Neighbors*, kNN es un clasificador supervisado que estima la categoría de una instancia basandose en su vecindad con instancias ya conocidas. Se desempeña mejor en problemas donde la vecindad espacial entre observaciones es un factor importante en la clasificación de instancias.

Para una instancia de entrada x, se calcula su distancia con todo el set de entrenamiento. La clasificación estimada resultante es la categoría modal (voto mayoritario) entre los k vecinos mas cercanos a x en el set de entrenamiento.

Influye en kNN la elección del k en el rango $1 \leq k \leq N$, así como la elección de la norma a utilizar para medir distancia. Para un valor de k alto, la estimación puede verse afectada por la categoría modal en el conjunto de entrenamiento. Para una eleción de k chico, la clasificación puede verse afectada por la presencia de outliers *outliers* en su cercanía.

2.4. Principal Component Analysis

Principal Component Analysis, PCA, es un algoritmo de reducción de dimensionalidad que transforma los datos de entrada en un espacio de *features* correlacionadas a un conjunto de valores cuyas *features* no guarden correlación entre sí.

Si del conjunto de vectores $\{v_1, ..v_n\} \subset \mathbb{R}^m$ que conseguimos mediante Bag of Words tomamos el vector de medias muestrales $\mu = \sum \frac{v_i}{n} \in \mathbb{R}^n$ podemos conseguir la matriz $X \in \mathbb{R}^{m \times n}$ con filas $\frac{v_i - \mu}{\sqrt{n-1}}$ podemos construir su *matriz de covarianza* $M = \frac{X^t X}{n-1}$ de cuya descomposición SVD, como para toda matriz X vale que $X^t X$ es simétrica con autovalores reales y base ortonormal de autovectores también reales, conseguimos una base de autovectores ordenada decrecientemente de acuerdo a la varianza interna de cada componente (que se asocia a qué tan grande es su autovalor) y con nula covarianza entre sí de modo que un cambio de base con los primeros autovectores pase la matriz a una base con menos ruido entre componentes.

Esto también nos permite recortar los últimos vectores de la base dado que son los que menos información proveen (además de ahorrar tiempo y espacio al considerar menos componentes espaciales en kNN), que llamermos una *reducción de dimensionalidad* a α cantidad de vectores.

2.5. Método de la potencia

Se trata de un método iterativo que permite hallar los autovalores dominantes de una matriz dada (aquellos cuya norma supera al resto) y sus autovectores asociados. O sea conseguimos un λ_1 tal que:

$$|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$$

Para todos los autovalores λ_i en una matriz de $n \times n$.

La idea es usar este método para conseguir los autovectores de las α componentes más significativas descriptas en 2.4. Al tratarse de un método iterativo que converge asintóticamente tendremos que implementar distintos criterios de parada para dejar de iterar.

Usamos la variante del método de la potencia que se presenta para matrices simétricas (como ya vimos, la matriz de covarianzas cumple tal propiedad) vista en la sección 9.3 del libro de Burden [1].

3. Experimentación

En esta sección se incluye la experimentación llevada a cabo sobre nuestro modelo de clasificador. Por restricciones de tiempo, todos los experimentos fueron corridos sobre la implementación de *Scikit Learn*.

3.1. Metodología

3.2. Convergencia de método de la potencia

Como en cada k-ésima iteración del método de la potencia estamos consiguiendo un $v^{(k)}$ autovector tentativo asociado a un $\hat{\lambda}^{(k)}$ autovalor también tentativo podemos analizar su evolución para determinar si están convergiendo y cortar la iteración cuando ya lo consideremos suficiente. Dado un $\epsilon \in \mathbb{R}$ tal que $\epsilon > 0$, la matriz cuadrada $A \in \mathbb{R}^{n \times n}$ sobre la cual buscamos su autovalores, y los autovalores y autovectores tentativos de la k-ésima iteración $\hat{\lambda}^{(k)} \in \mathbb{R}$ y $v^{(k)} \in \mathbb{R}^n$ planteamos los siguientes criterios de corte:

- Criterio de *vector residual*: cortar si $||Av^{(k)} - v^{(k)}\hat{\lambda}^{(k)}|| < \epsilon$
- Criterio de diferencia de autovectores: cortar si $||v^{(k)} - v^{(k-1)}|| < \epsilon$
- Criterio de diferencia de autovalores: cortar si $||\hat{\lambda}^{(k)} - \hat{\lambda}^{(k-1)}|| < \epsilon$

Como bajo ciertas condiciones (como $|\lambda_1| \approx |\lambda_i|$ para λ_1 autovalor dominante y λ_i otro autovalor de la matriz A) el método de la potencia puede no converger, al margen de los criterios se corta la iteración tras una cantidad fija y lo suficientemente grande (de modo que si converge no interrumpa a los criterios anteriores) de veces.

En cuanto a precisión especulamos originalmente con que el mejor fuera el criterio de vector residual dado que mide qué tan bien aproximan $\hat{\lambda}^{(k)}$ y $v^{(k)}$ a un autovalor y autovector asociado, seguido del criterio de autovectores bajo la presunción de que funciona mejor al tener que aproximar varias componentes del autovector para terminar versus aproximar únicamente el autovalor. Bajo las mismas suposiciones especulamos livianamente que la precisión sería inversamente proporcional a la cantidad de iteraciones necesarias siendo el criterio de diferencia de autovalores el que antes se cumpliría, seguido del de autovectores y por último el de vector residual.

Sin embargo, investigando más profundamente nos encontramos con el resultado del Teorema 9.19 del libro de Burden [1] que indica que si vale:

$$||Av^{(k)} - \hat{\lambda}^{(k)}v^{(k)}|| < \epsilon \tag{1}$$

entonces tenemos que, con λ_j autovalores de A:

$$\min_j |\lambda_j - \hat{\lambda}^{(k)}| < \epsilon \tag{2}$$

Si bien da una buena idea de que el criterio de vector residual contiene en sí información sobre la convergencia de los autovalores (tentando la idea de que es más restrictivo que este) no necesariamente el λ_j más cercano a $\hat{\lambda}^{(k)}$ sea el λ_1 buscado y también podría suceder que, bajo ciertas condiciones, $v^{(k)}$ esté más cerca de λ_1 que de $v^{(k-1)}$.

El criterio de diferencia de autovectores nos resultaba interesante porque como no considera autovalores para cortar de ser lo suficientemente bueno significaría solo computarlos al final del método y no en cada

iteración, además de que se condice en espiritu con el uso que le vamos a dar al resultado (solamente usar los autovectores para cambiar de base la matriz de covarianzas, sin importar los autovalores)

Otro resultado interesante en la misma sección del mismo libro es que, como indican al hablar de aceleración de convergencia, la secuencia $\{\hat{\lambda}^{(k)}\}$ converge, cuando $\lambda_1 > \lambda_2$ con λ_2 el más grande (potencialmente no único) de los autovalores no dominantes, linealmente (más específicamente en tasa de convergencia $\mathcal{O}(\lambda_2/\lambda_1)$). Inicialmente entendimos mal que esto significaría que la cantidad de iteraciones fuera lineal sobre ϵ sino que, por el contrario, que la tasa de convergencia sea lineal significa justamente que la sucesión se aproxima de manera lineal a λ_1 (ver definición en sección del libro [1]), lo que implica una convergencia lenta con potencialmente muchas más iteraciones que una secuencia de convergencia, por ejemplo, cuadrática. De hecho, como se puede ver en la tabla 2.7 [1], una secuencia de tales características es potencialmente exponencial incluso.

Para experimentar usamos método de la potencia con los 3 criterios de manera separada moviendo con 20 iteraciones (sobre las que se considera promedio y desvio estandar) por valor de ϵ en un rango de 1 a 10^{-14} (iteramos el exponente i tal que $10^{-i} = \epsilon$ entre 0 y 14), el primer número resultado de una corrida en la que verificamos que el error era muy grosero como para aproximar bien autovectores y el último número resultando de probar a mano y ver que el tiempo que tardaba no lo considerábamos admisible para una ejecución cómdo (teniendo en cuenta que luego en vecinos más cercanos y PCA ibamos a iterar continuas veces el método), buscando un punto intermedio de precisión y latencia aceptables. Para medir el error usamos la inversa (de modo que sea creciente conforme mejor precisión) de la norma del mismo vector residual $||Av^{(k)} - v^{(k)}\hat{\lambda}^{(k)}||$ dado que por su naturaleza de medir 'qué tan buenos autovectores y autovalores' son nos resultó más atractiva que la idea de comparar contra librerias como *NumPy*.

Inicialmente probamos con un set reducido de 2000 vectores (elegidos aleatoriamente sobre el total) provenientes del dataset *imdb_small.csv* sobre los que se arma la matriz de covarianzas para poder determinar si podíamos recortar un poco más el intervalo antes de pasar al dataset entero. Consiguiendo los siguientes resultados:

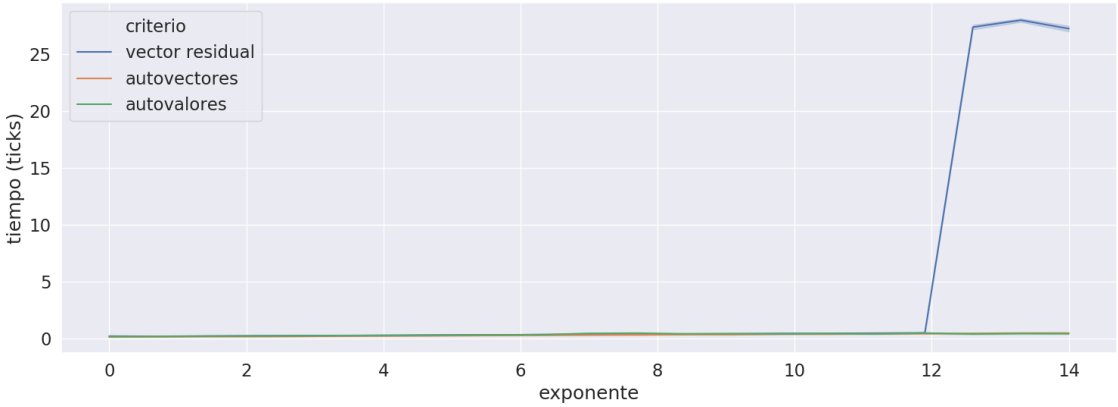

Figura 1: Progresión del tiempo medido en ticks del metodo de la potencia en función del exponente del epsilon para 2000 vectores aleatorios.

Como se puede ver en la figura 1 la latencia del criterio vector residual explota desmedidamente a partir del exponente 12, como el tiempo era demasiado decidimos recortar el rango y mover el exponente solo hasta 12 antes de pasar a medir con el dataset completo.

Vemos en la figura 2 que el criterio de autovalores alcanza precisiones considerables con menos exponentes que el resto pero se mantiene inestable en variaciones y con media constante antes de que el criterio de vector residual empiece a subir de manera mucho más consistente. El criterio de autovalores recien sobre los últimos exponentes empieza a mostrar una mínima mejora.

El problema de la explosión rápida en precisión del criterio de autovalores es que viene acompañado de una explosión en latencia como indica la figura 3 y en el caso del criterio de vector residual para el momento en que alcanza una precisión similar al criterio de autovalores la latencia ya es considerable (tener en cuenta que la linealidad en función del exponente significa exponencialidad respecto de epsilon para la latencia). Decidimos entonces, antes de analizar los exponentes mayores a 6 tras la explosión del criterio de autovalores y en la subida del criterio de autovectores, analizar qué sucede antes del 6.

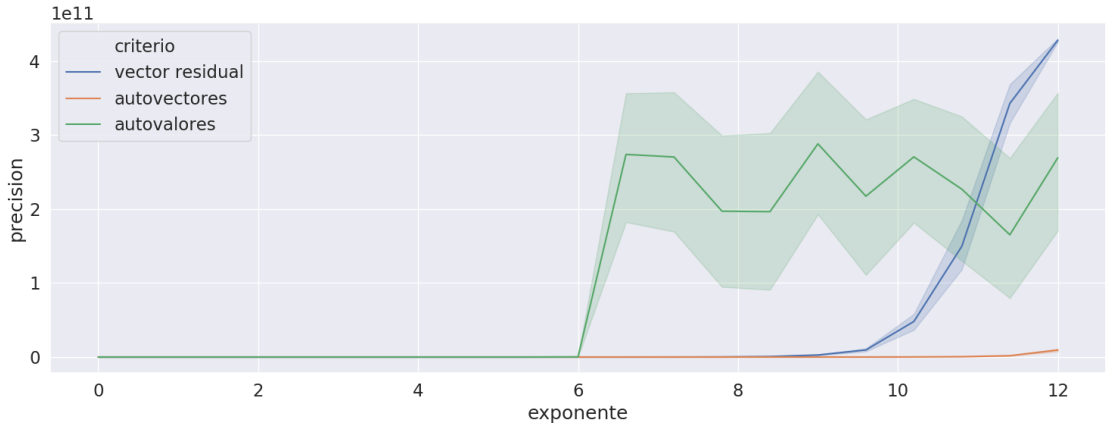

Figura 2: Progresión de precisión del metodo de la potencia en función del exponente del epsilon para los 6225 vectores del dataset.

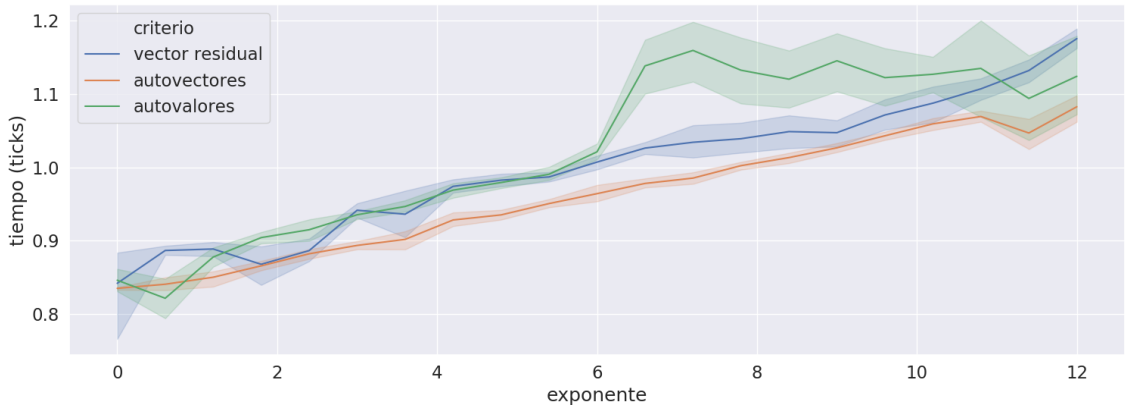

Figura 3: Progresión del tiempo medido en ticks del metodo de la potencia en función del exponente del epsilon para 6225 vectores del dataset.

Viendo las figuras 5 y 4 consideramos que el exponente 6 era una interesante alternativa dado que tiene en autovalores latencia mucho menor a los exponentes mayores y aun así mostraba una mejora importante de precisión respecto de exponentes menores. Decidimos entonces, antes de pasar a comparar los valores mayores a 6 la posibilidad de comparar el criterio de autovalores para exponente 6 contra estos (particularmente 7, primer valor tras la explosión) en una instancia de PCA con k y α fijos arbitrarios, dado que solamente nos interesa medir qué tan bien funcionan en un caso práctico de PCA (que es la finalidad del método) y ninguno de los dos parámetros favorece a ninguno de los dos exponentes (sí afecta α en cómo se arrastra errores de precisión en el método por cuestiones de deflación, pero penaliza justamente al menos preciso de modo que no nos afecta). Al iterar corridas con tales parámetros y notar que la medida de accuracy era exactamente la misma en ambos casos con tiempos favorables (cerca de la mitad en algunos casos) para el menor exponente decidimos entonces fijar $\epsilon = 10^{-6}$.

3.3. Impacto del tamaño del set de entrenamiento

Fueron tomadas mediciones sobre el comportamiento del *accuracy* al reducir el tamaño del set de entrenamiento para kNN con PCA. Para ello se redujo el tamaño muestral, tomando distintas sub-muestras al azar, respetando una proporción pareja entre reseñas con calificación positiva y negativa. Se mantuvieron constantes k y α. El experimento se repitió para distintas configuraciones de los mismos.

Puede observarse en las figuras 6 y 7 que el *accuracy* incrementa al incrementar el tamaño muestral. Por otro lado, la calidad del modelo crece cada vez en menor medida, por lo que se cree que existe un punto a partir del cual carece de sentido incrementar el set de entrenamiento. Como el rendimiento del modelo sigue siendo bajo, creemos que la máxima cantidad de instancias de entrenamiento disponible, $N = 6225$, esta por debajo del tamaño óptimo.

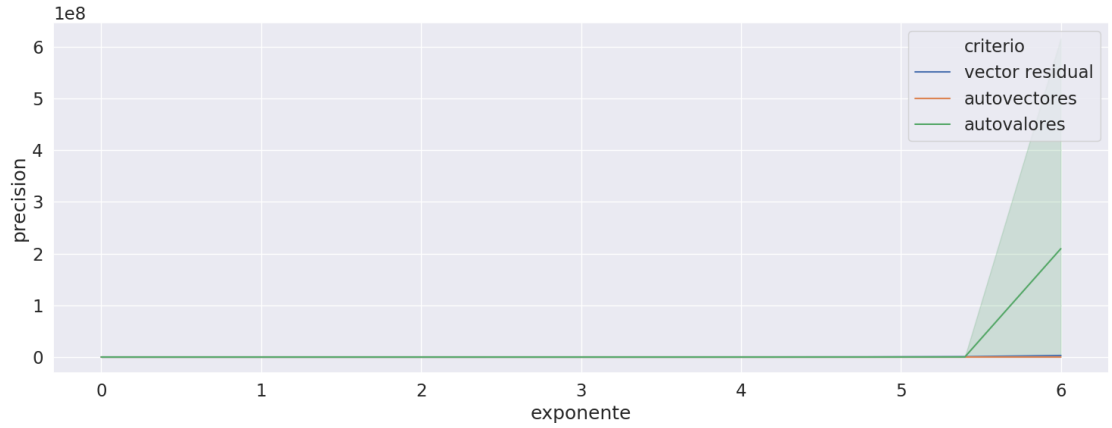

Figura 4: Progresión de precisión del metodo de la potencia en función del exponente del epsilon para los 6225 vectores del dataset.

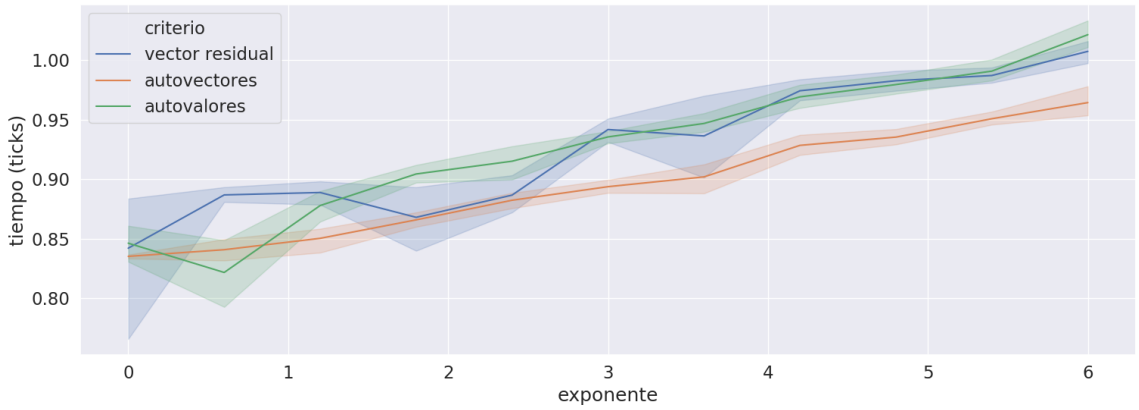

Figura 5: Progresión del tiempo medido en ticks del metodo de la potencia en función del exponente del epsilon para 6225 vectores del dataset.

Se observó el mismo comportamiento de la curva para todas las configuraciones de k y α, por lo que se cree, el tamaño muestral óptimo es independiente de los hiperparámetros a escoger.

A su vez se observa que la varianza en el *accuracy* es mayor para tamaños muestrales chicos. Esto es porque las reseñas conocidas por el modelo pueden ser, con mayor probabilidad, poco representativas la totalidad de las reseñas.

3.4. Optimización de k para kNN sin PCA

Para buscar una k cantidad de vecinos que optimice kNN usamos nuevamente el dataset de *imdb_small*, iteramos un k de 1 a 3000 con saltos de a 20.

Originalmente pensábamos arrancar en un rango mas adelantado dado que para k pequeños sobre muestras grandes se puede correr riesgos de overfitting por puntos ruidosos"que estén demasiado pegados ganandole en la votación a la verdadera clase, pero como k pequeños no son caros de computar los consideramos en la experimentación de todos modos.

Elegir k grande tiene el problema de que los elementos de la clase de un punto quizás están todos en su vecindad inmediata y a medida que vamos buscando vecinos en zonas más alejadas vamos tendiendo a clasificar peor el punto: una clase muy densa pero rodeada de otra mas dispersa y sobrerrepresentada será peor catalogada cuanto más grande sea k. Si además k es lo suficientemente grande (aproximadamente superando la mitad de la población total) la proporción de vecinos más cercanos se parece cada vez más y más a la proporción muestral de cada clase sobre el total, lo cual no aporta nada de información. Por lo tanto decidimos iterar solamente hasta la mitad.

Tendría sentido con este análisis que acabamos de hacer encontrar los mejores resultados en un punto intermedio en la magnitud de k.

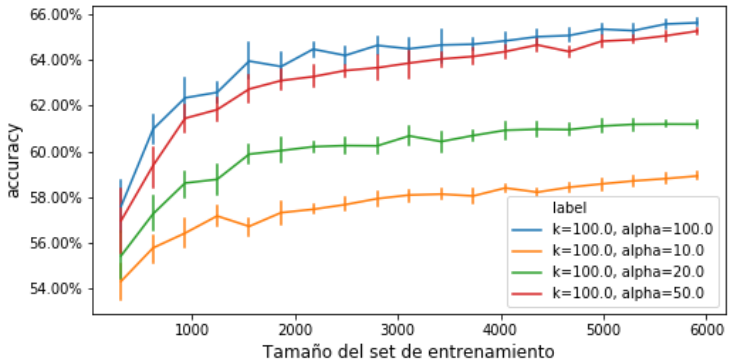

Figura 6: *Accuracy* obtenido al reducir el conjunto de entrenamiento. Para valor fijo de $k = 100$.

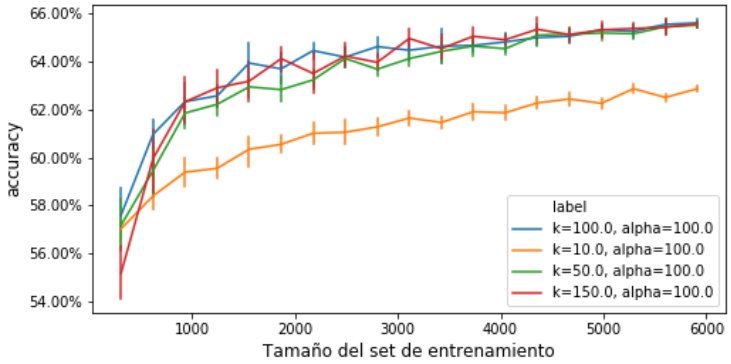

Figura 7: *Accuracy* obtenido al reducir el conjunto de entrenamiento. Para valor fijo de $\alpha = 100$.

Como podemos apreciar en la figura 8, se cumple nuestra predicción de que la precisión empeora conforme la cantidad de vecinos se vuelve muy grande o muy pequeña. Alcanzamos el máximo accuracy score en 0.681116 para $k = 1801$ que se sitúa muy cerca de la mitad del intervalo que elegimos. Nos resulta complicado comprender por qué desciende la precisión en los primeros valores, especulamos con que los problemas de ruido y sesgo requieran valores de k bajos pero mayores a los mínimos para que empiecen a aparecer. Otro motivo sea que si el espacio de los vectores no esté separado de manera clara existan zonas con más ruido que en el resto y que se vayan acumulando outliers de clases difusas hasta que se sale de los mismos.

3.5. Optimización de k y α para kNN con PCA

En esta sección se buscan los hiperaparámetros de KNN y PCA que mejor precisión[1] nos brinden, sin perder de vista, pero dejando en segundo plano, el tiempo de cómputo asociado. Además respetaremos los parámetros de tolerancia y vocabulario encontrados en las secciones anteriores.

3.5.1. Performance

El primer problema con el que nos encontramos es la amplia variación que pueden tomar los parámetros; así como los tiempos largos que incurre el cómputo de este método.

En particular la obtención de valores singulares para lograr el análisis de componententes principales, que ya fué mencionada en el apartado de *power method*.

Pero también la comparación de cada nuevo punto en el predict, es altamente dependiente de la cantidad de puntos que tiene el dataset de entrenamientos. Como se mostró en la sección de *tamaño de muestra*, el accuracy es sensible a la cantidad de datos de entreamiento. Por esto es que buscamos una manera de reducir la cantidad de comparaciones que se efectúan, sin reducir la cantidad de datos.

Para esto usamos una estructura "arboles kd" o "kd-trees" que permiten particionar el espacio de forma binaria por dimensión y de esta manera permiten implementaciones mucho más eficientes de KNN.

[1] En el informe usaremos indistintamente accuracy o precisión

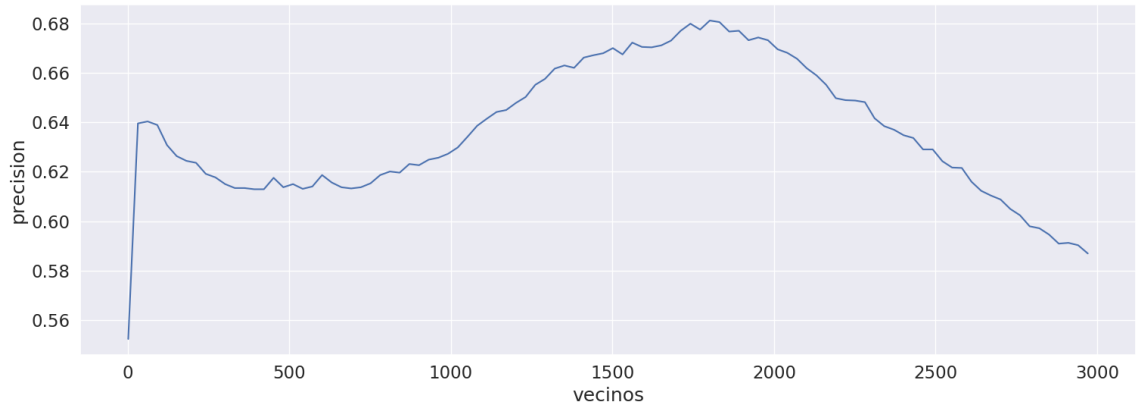

Figura 8: Progresión de la precisión de KNN en función de la cantidad de vecinos.

3.5.2. Busqueda de Parámetros

A pesar de las optimizaciones hechas, la búsqueda es intensa en tiempo y extensa en los valores que pueden tomar los parámetros.

Valores grandes de PCA se descartan, pues uno de las características del método es reducir la cantidad de dimensiones de los puntos muestrales. Es decir, en nuestro caso, elegir aquellas palabras que tienen un mayor peso en la predicción de la clasificación.

Por otro lado valores muy pequeños pueden generar pérdida de información.

Algo similar sucede con el parámetro de los vecinos, valores muy pequeños hacen demasiado sensible al punto a ser predecido, de su vecindad inmediata, volviendo el método sensible a las particularidades de los datos y por lo tanto poco fiable.

Valores muy grandes de vecindad, son computacionalmente costosos y además se pierde el sentido de vecindad que es necesario para una clasificación exitosa. En el caso extremo, tomar como vecinos toda la población de entrenamiento, le asignaría a cada nuevo punto el mismo valor: aquel mayoritario en el universo.

Los valores buscados entonces, están en algún punto intermedio. Sin embargo, este sigue siedo un intervalo bastante grande.

Con un vocabulario del orden de las 5 mil palábras y una cantidad de datos de entrenamiento del orden de los 15 mil, tenemos un espacio enorme para la búsqueda.

Nuestra aproximación al problema, fue el de hacer una grilla, donde se generan particiones del plano vecinos-componentes de forma regular. Luego usar heurísticas de búsqueda en cada una de las celdas de la misma. En particular se usó *hill climbing*[**?**][2] Definiendo un tamaño de grilla y un punto inicial en la misma, la heurística se mueve a pequeños intervalos -discretos- en las cercanías de la solución obtenida, hasta llegar a un máximo local que no puede ser mejorado. La definición de estos pequeños intervalos, está relacionada con el tamaño de cada celda por un lado y por otro de la cantidad de iteraciones que hará la búsqueda local hasta alcanzar un máximo. Esto último se debe a que los intervalos son discretos, entonces cuanto más grandes son menos soluciones estamos considerando y es más corto el camino hacía un máximo que no se pueda mejorar (lo que no significa que esa un máximo real).

Esta metodología nos permite reducir mucho el espacio de búsqueda. Por poner un ejemplo de una grilla de 100x100, donde se efectúa una búsqueda local considerando 200 elementos, pagamos un costo $\frac{200}{100^2} = 0,002$ menor al de explorar toda la grilla y es sensiblemente mejor que simplemente tomar un punto arbitrario de la misma.

Para la definición de los parámetros de la búsqueda local, tuvimos en cuenta las consideraciones anteriores y generamos dos grillas. Una de tamaño pequeño, donde las componentes principales se mueven entre 10 y 90 mientras que los vecinos entre 100 y 1000 9 y otra con una grilla de un orden de maginitud más grande 10 que sigue de donde dejó la anterior en los componentes principales, pero los incrementa a un ritmo mayor, pero mantiene las divisiones de los vecinos.

En ambos experimentos, la definición del tamaño de los pasos que hace la búsqueda local, tuvimos en cuenta la diferente relación de aspecto de las grillas. En el primer caso usamos un paso de 20 para los vecinos y de 2 para las componentes principales, que guarda la relación entre el tamaño de las aristas de

[2] o búsqueda local, intenta en cada paso mejorar la solución obtenida, considerando una frontera de vecinos.

cada celda. Para el segundo experimento usamos para ambas dimensiones un paso de 10, que guarda la relación cuadrada de la celda.

En lafiguras 9 se ve una buena zona para elegir parámetros en toda la franja 70-90 con algunos picos con pocos vecinos (200) rodeados de algunos valles, mientras que al aumentar los vecinos (800) parecíera tener menos picos pero una mejor base.

Vemos en la figura 10 que se conecta adecuadamente, la primera fila, con la última del gráfico anterior; sugiriendo cierta suavidad en los superficie i.e en la relación de los parámetros con la precisión. El descenso más abrupto entre 500 y 600 para PCA que el gráfico anterior se debe al cambio de escala. El descenso de la precisión se puede deber a que la franja 60-600 presenta las componentes con mayor peso y todo lo que se agregue luego es ruido.

Cabe destacar que según se acostumbra en procesamiento de lenguaje[3] natural[3] la cantidad de vocabulario se suele tomar como función creciente de la raiz cuadrada de los tokens[4], y que según nuestros datos, la raiz es un valor que ronda los 300[5]; bien dentro de la franja de componentes principales, que mencionamos antes.

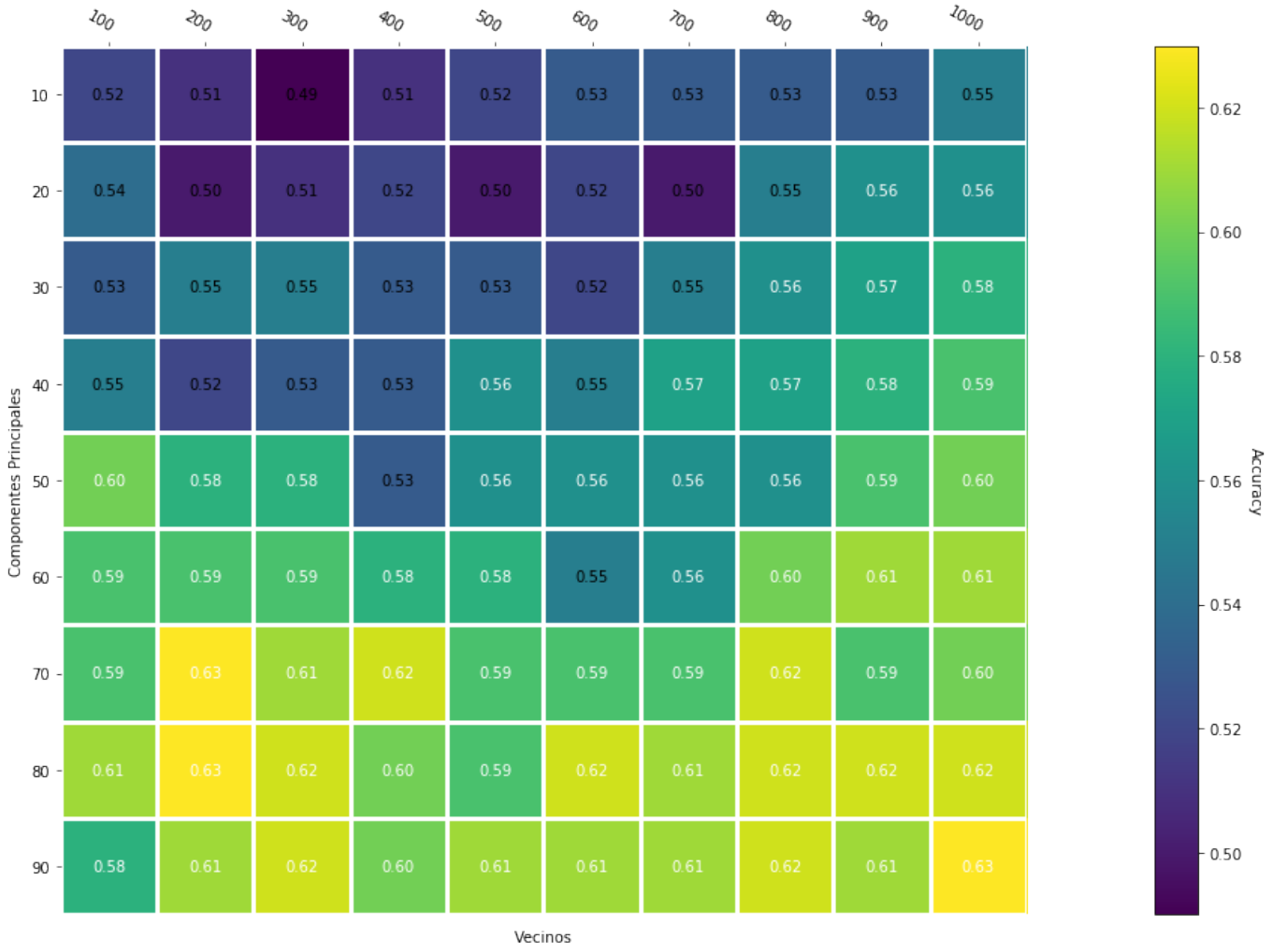

Figura 9: Resultado de la busqueda local sobre cada región **pequeña** de k100xα10. Usando la implementación de sklearn. En cada una de las celdas, se corrió una búsqueda local, comenzando en el valor mostrado en los ejes hasta alcanzar un máximo local. El promedio de corridas es aproximadamente de 8, versus una cantidad de puntos de **1000**. Se muestran sobre el color, el accuracy logrado por el máximo local.

[3]In general (Gale and Church, 1990) suggest that the vocabulary size (the number of types) grows with at least the square root of the number of tokens (i.e. $V > O(\sqrt{N})$.

[4]Tomamos vocabulario, como types. Los tokens son todas las entidades identificables en un corpus (una colección de textos) Puede incluir hasta signos de puntación. Los types/tipos, son las palabras utiles.

[5]El valor no es exacto porque depende de como se consideren los tokens

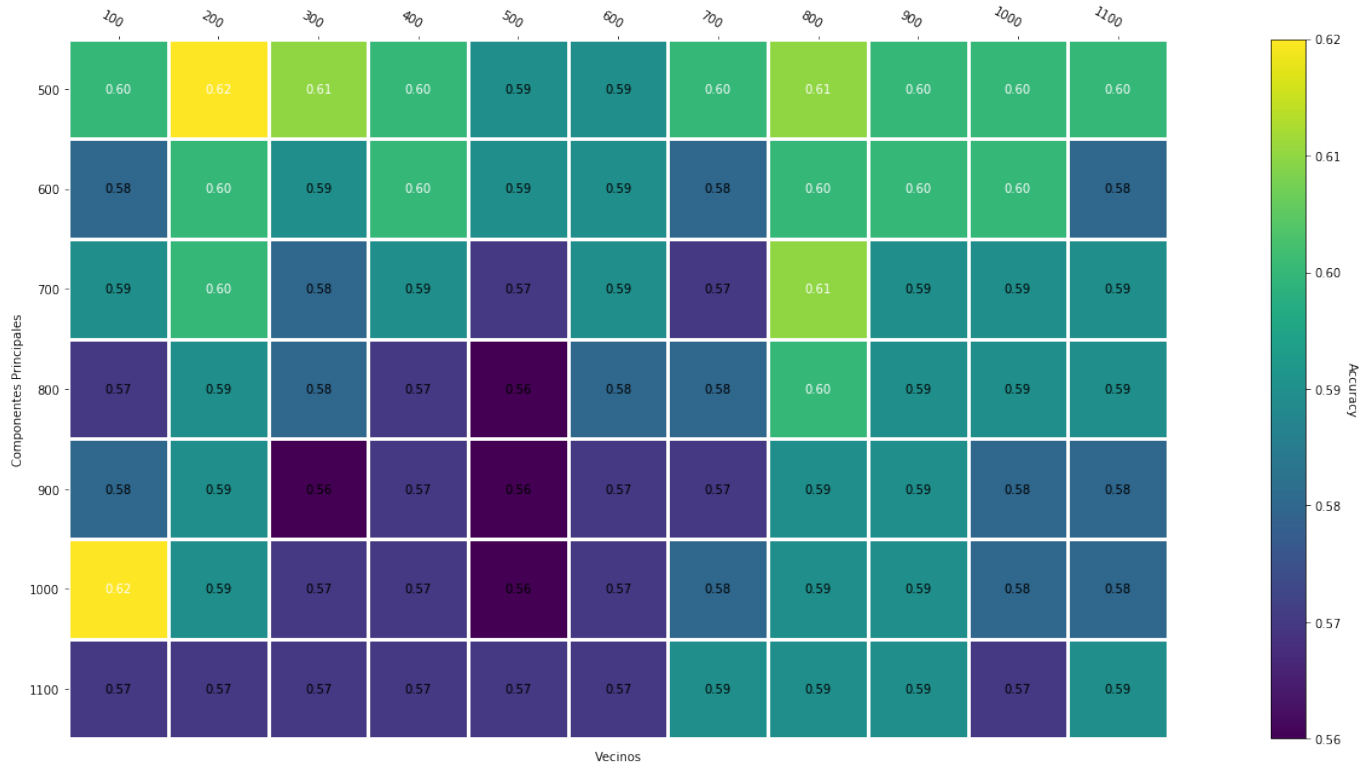

Figura 10: Resultado de la busqueda local sobre cada región **grande** de k100xα100. Usando la implementación de sklearn. En cada una de las celdas, se corrió una búsqueda local, comenzando en el valor mostrado en los ejes hasta alcanzar un **máximo local**. El promedio de corridas es aproximadamente de 12, versus una cantidad de puntos de **10000**. Se muestran sobre el color, el accuracy logrado por el máximo local.

4. Conclusiones

Para el método de la potencia determinamos un criterio de corte buen rendimiento asociado a un ϵ que no costara mucho en términos de tiempo, sobre todo considerando que al aplicar deflación este método se llama continuamente.

De mismo modo usamos una estructura de datos alternativa para que kNN fuera mas perfomante y optimizamos parámetros de k y α para tal método y PCA.

Potencial trabajo futuro sería experimentar sobre las normas que se usan para buscar vecinos cercanos y modos de vectorización que optimicen la información contenida en los vectores.

5. Referencias

Referencias

[1] Burden, Richard L. and J. Douglas Faires. *Numerical analysis. Fifth Edition.* Boston: PWS Publishing Company, 1993.

[2] Stuart Russell, Peter Norvig *Artificial Intelligence: A Modern Approach* (3rd Edition) 2009.

[3] Daniel Jurafsky, James H. Martin *Speech and Language Processing:* An introduction to natural language processing, computational linguistics, and speech recognition. 2006.