DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico #2 — Análisis de Sentimiento

30 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. Resumen

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*.

La experimentación extensa sobre α y k, no arrojó tan buenos resultados como una adecuada vectorización del vocabulario. La elección del punto de corte en palabras de más y menos frecuencia tuvo tanto o más peso que la elección de la cantidad de vecinos y componenetes principales.

2. Introducción Teórica

El análisis del lenguaje natural y machine learning son dos areas importantes en el análisis de datos que experimentaron en los últimos años un avance sinérgico. El análisis de lenguaje tiene varias aplicaciones, desde mejorar las interfaces humano-máquina hasta la detección automática de pedofilia en las redes sociales.

En el presente trabajo se efectúa un análisis de sentimiento sobre reseñas de películas usando un algoritmo de machine learning que clasifica de forma binariamente cada reseña.

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 basándose 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* 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 $\frac{v_i - \mu}{\sqrt{n-1}}$ para entonces conseguir 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 llamaremos 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.

3.1. Metodología

3.1.1. Dependencia de factores

Cuando consideramos una métrica de precisión como función objetivo de todos estos parámetros no resulta manejable optimizar en un único paso de manera multivariada (son muchos) ni tampoco analizar parámetro a parámetro dado que cada uno afecta como variable de manera interpendiente a la función objetivo no pudiendo aislar un *orden topológico* de parámetros para optimizar de manera independiente. En respuesta a esto se plantea un esquema de experimentación basado en aproximar un orden de optimización de parámetros que sea relativamente parecido a un orden topológico fijando algunos de ellos en valores intuitivos para eliminar dependencias y poder seguir con las optimizaciones de a pares o por separado según sea conveniente de manera de respetar lo máximo posible las dependencias. Aunque solamente realizamos una iteración como prueba de concepto, este esquema permite poder ir actualizando los valores que se fijan de manera iterativa incremental mejorando paso a paso la función objetivo. El esquema en cuestión es:

- buscar tolerancia de convergencia para método de la potencia (desempata candidatos con PCA con algunos parámetros fijos)
- buscar el k óptimo para kNN sin PCA en función de alguna vectorización fija
- buscar k y alfa óptimos para kNN con PCA guiando el rango de k según el paso anterior nuevamente con vectorización fija
- buscar parámetros de vectorización óptimos para kNN según los parámetros anteriores

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 u ortogonalidades entre el vector inicial escogido y el autovector asociado a λ_1) 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 criterio) no necesariamente el λ_j más cercano a $\hat{\lambda}^{(k)}$ sea el λ_1 buscado y también podría suceder que, bajo ciertas condiciones, $\hat{\lambda}^{(k)}$ esté más cerca de λ_1 que de $\hat{\lambda}^{(k-1)}$.

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 cómoda (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 la norma del mismo vector residual $||Av^{(k)} - v^{(k)}\hat{\lambda}^{(k)}||$. La inversa es creciente a mayor precisión y la norma del vector residual es una medida del error. De esta manera podemos usarlo como un indicador de la precision. Dado que mide 'qué tan buenos autovectores y autovalores' son los obtenidos por este método, 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:

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.

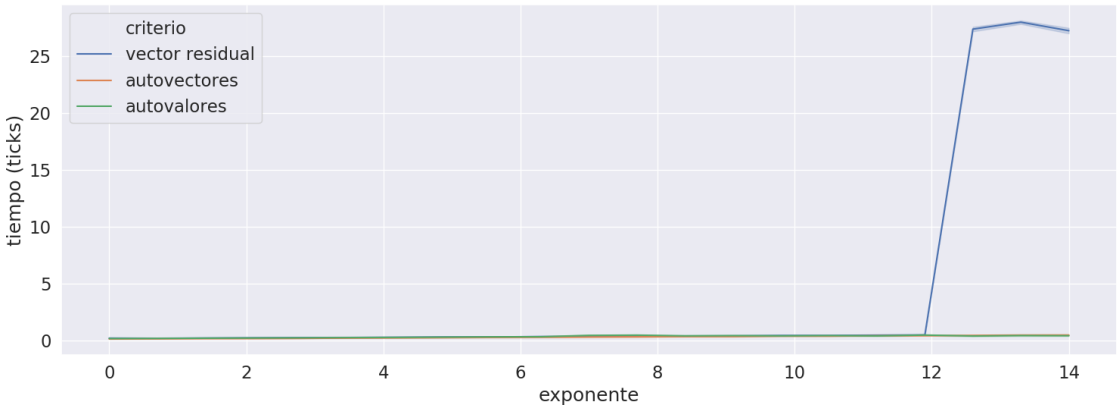


Figura 1: Progresión del tiempo medido en ticks del metodo de la potencia en función del exponente del epsilon para 2000 vectores aleatorios.

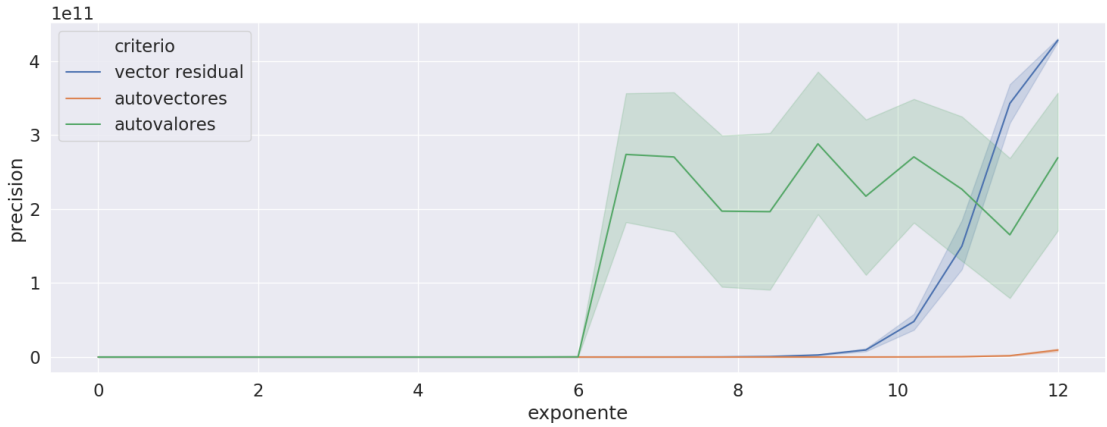


Figura 2: Progresión de precisión del metodo de la potencia en función del exponente del epsilon para los 6225 vectores del dataset.

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.

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 $\epsilon = 10^{-6}$.

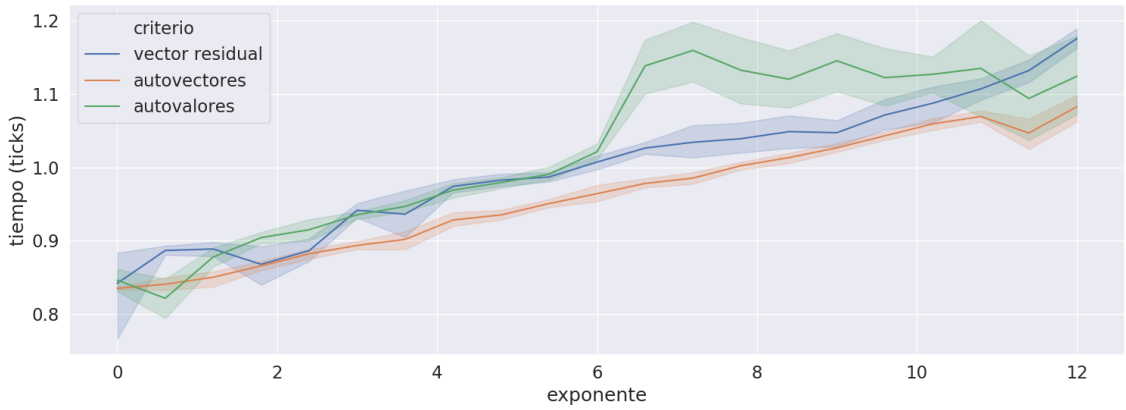


Figura 3: Progresión del tiempo medido en ticks del metodo de la potencia en función del exponente del epsilon para 6225 vectores del dataset.

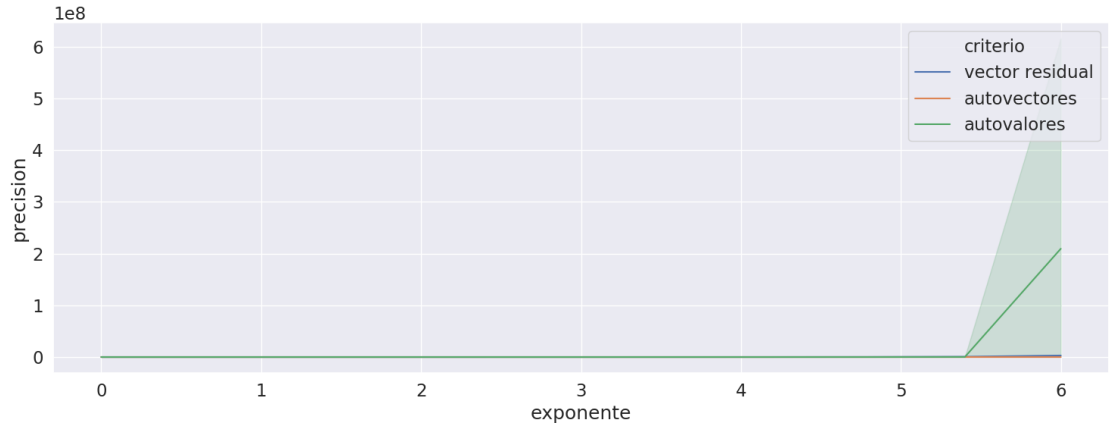


Figura 4: Progresión de precisión del metodo de la potencia en función del exponente del epsilon para los 6225 vectores del dataset.

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 práctico 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.

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 100 al principio y luego analizamos la región al rededor del máximo que encontramos.

Originalmente pensábamos arrancar en un rango mas adelantado dado que para k pequeños sobre mues-

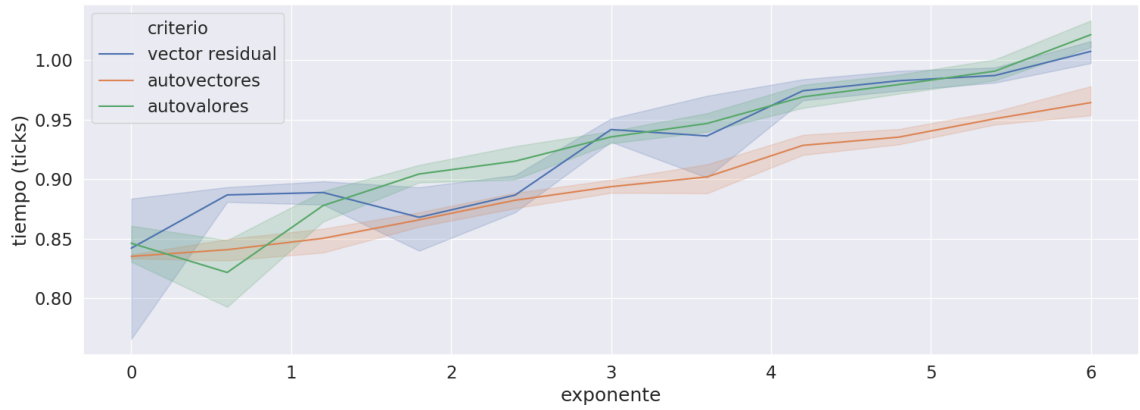


Figura 5: Progresión del tiempo medido en ticks del metodo de la potencia en función del exponente del epsilon para 6225 vectores del dataset.

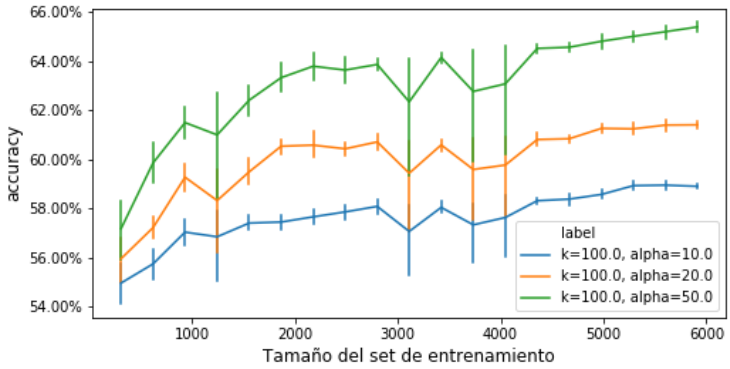


Figura 6: *Accuracy* obtenido al reducir el conjunto de entrenamiento. Para valor fijo de $k = 100$.

tras 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.

Como podemos apreciar en la figura 8, se cumple nuestra predicción de que el accuracy score empeora conforme la cantidad de vecinos se vuelve muy grande o muy pequeña. Focalizando en el pico del máximo local más grande en la figura 9 se ve que alcanzamos el máximo accuracy score en 0,685 para $k = 1826$ que se sitúa muy cerca de la mitad del intervalo que elegimos, interesantemente superando el 0,681116 para $k = 1801$ que nos había arrojado Scikit-learn para una experimentación similar en primera instancia. Nos resulta complicado comprender por qué desciende el accuracy score 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.

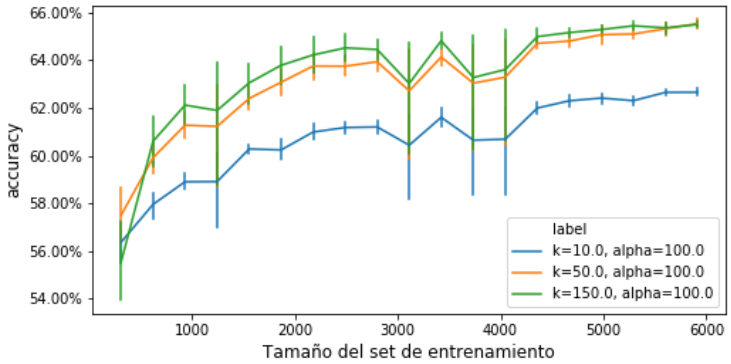


Figura 7: *Accuracy* obtenido al reducir el conjunto de entrenamiento. Para valor fijo de $\alpha = 100$.

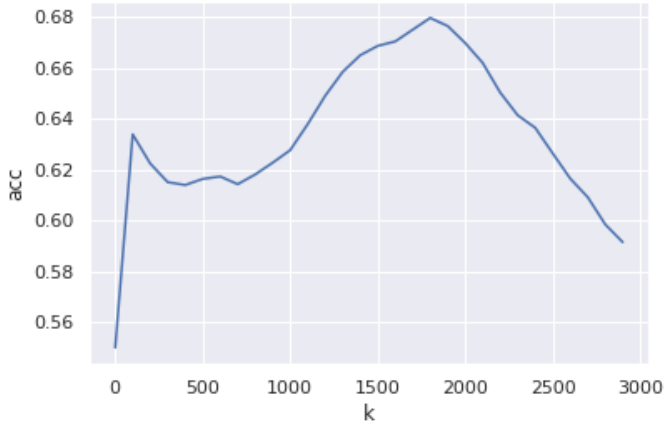


Figura 8: Progresión de accuracy score de KNN en función de la cantidad de vecinos.

3.5. Optimización de k y α para kNN con PCA

En esta sección se buscan los hiperaparámetros de KNN y PCA que mejor *accuracy* score[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. Más aún, este tipo de estructuras permite paralelismo en las queries, lo que nos permitió pasar del orden de minutos a segundos para buscar vecinos más cercanos.

Para experimentar de manera más veloz sobre PCA también decidimos computar una única vez los cambios de variables de la matriz sobre el mayor alfa requerido y recortar luego componentes de esa matriz ya almacenada.

[1] Porcentaje de predicciones correctas

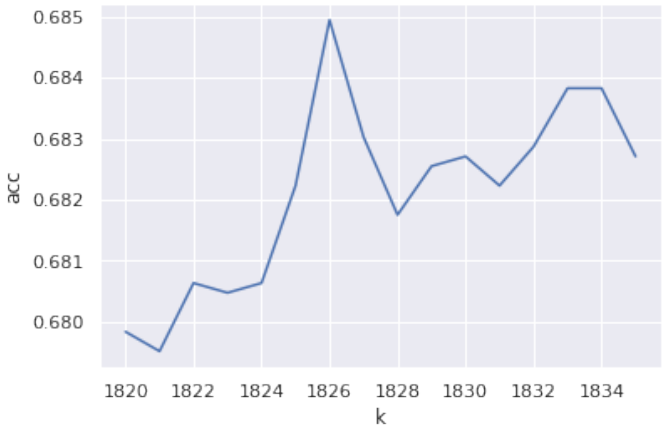


Figura 9: Progresión de accuracy score de KNN en función de la cantidad de vecinos en la zona del máximo.

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][2]

Antes de seguir comentaremos rápidamente en que consistió esta metodología

Hillclimbing engrillado Por cada grilla, 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 figura 10a muestra la idea de *hillclimbing* en una búsqueda de una dimensión.

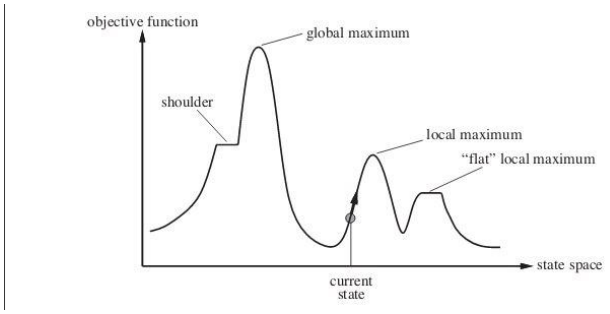
La definición de estos pequeños intervalos, está relacionada con el tamaño de cada celda por un lado y por otro de la cantidad de iteraciones que hará la búsqueda local hasta alcanzar un máximo.

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). En las figuras 10b 10c 10d mostramos como se generan los heatmaps, tomando el mayor de los valores en cada celda.

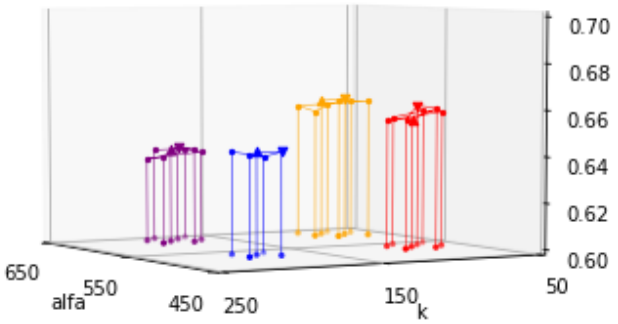
Esta metodología nos permite reducir mucho el espacio de búsqueda. Por poner un ejemplo de una grilla de 100x100, donde se efectúa una búsqueda local considerando 200 elementos, pagamos un costo $\frac{200}{100^2} = 0,002$ menor al de explorar toda la grilla y es sensiblemente mejor que simplemente tomar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

Busqueda iterativa engrillada de escaladores: Para la definición de los parámetros de la búsqueda local, tuvimos en cuenta las consideraciones anteriores y generamos varias grillas. Primero una de gran tamaño (componentes principales entre 100 y 1000 con cantidad de vecinos de hasta 2000, guiandonos por el pico de la curva de la experimentación de kNN sin PCA) que nos permitiera analizar el relieve general

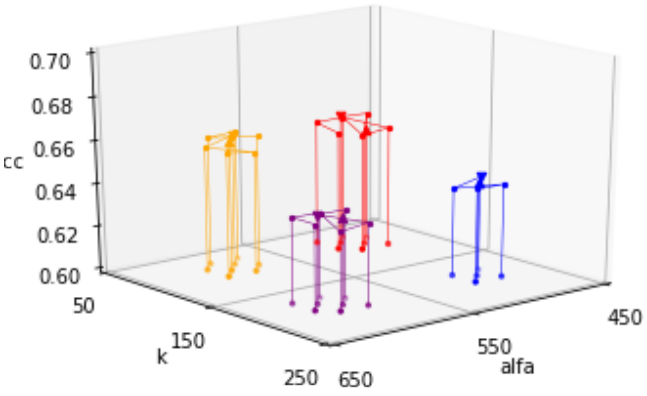
[2] o búsqueda local, intenta en cada paso mejorar la solución obtenida, considerando una frontera de vecinos.



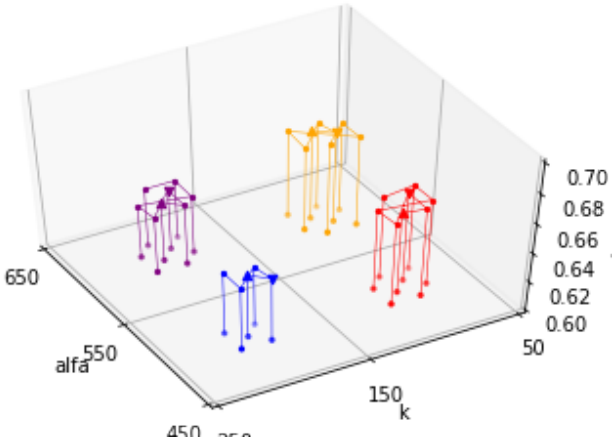
(a) Búsqueda local o hillclimbing en un caso hipotético sobre una dimensión. La heurística permite reducir el espacio de búsqueda, pero puede caer en extremos locales sub óptimos.



(b) Hill climbing engrillado. Perspectiva 1.



(c) Hill climbing engrillado. Perspectiva 2.



(d) Hill climbing engrillado. Perspectiva 3.

Figura 10: De (b) a (d) se muestran distintas perspectivas de la partición del espacio en 4 celdas de 100x10 (k x α). En distintos colores cada búsqueda local. Las lineas verticales proyectan sobre el plano para dar una idea de donde están. Las lineas en el espacio conectan en orden los puntos explorados. Los triángulos que apuntan hacía arriba indican de donde comienza la búsqueda. Los que apuntan hacia abajo, el máximo valor encontrado. Estos últimos suelen no ser el último valor de la búsqueda, porque para saber que es un máximo el algoritmo tiene que ver todos los vecinos. Esto explica que la linea que une el orden de visita, no sea una curva ascendente monotónica si no un zig-zag de ascenso y descenso.

y buscar mesetas sobre las cuales hacer distintos "zooms" con mayor granularidad enfocados en regiones específicas de interés, luego una más pequeña con valores chicos de alfa y k cercanos a una meseta en particular inspirados por las grillas anteriores.

En el panorama mascroscópico de la figura 11 se puede notar una característica positiva respecto de nuestro método de experimentación: la imagen de la función a maximizar es muy suave, lo que es favorable para el hillclimbing frente a superficies más "cerruchadas" y que nos permite tener más confianza en los máximos locales obtenidos como posibles máximos globales de cada celda (lo que significa necesitar menos iteraciones de "zooms" para análisis exhaustivos). Otra particularidad interesante es que la función se comporta muy parecido al caso que vimos optimizando sin PCA: alcanza buenos valores con pocos (< 150) vecinos, luego empeora y cerca de los 2000 vecinos se vuelve a comportar de manera precisa. Si bien analizaremos con más detalle ambas mesetas, asumiendo que la suavidad de la función nos indica que los máximos encontrados son esos y que ninguna supera a la otra, decidimos descartar aquellas combinaciones con valores grandes de k o alfa (en el caso de que no haya picos dominantes en tal región) por cuestiones de tiempo de cómputos dado que, sobre esa misma frontera de nuestra región total, se vuelven muy poco prácticas tales latencias.

En el zoom de la región de parámetros grandes visto en la figura 12 se valida nuestra suposición de que la suavidad de la imagen de la función objetivo implicaría buena representatividad de los máximos locales hallados con granularidades más altas: muy pocas celdas superan el 0,66 de accuracy score correspondiente

a las respectivas zonas del muestreo panorámico anterior. Si la imagen de la función diera saltos crecerían nuestras chances de encontrar picos puntuales en análisis más microscópicos. Como dijimos, esto nos permite descartar la región en favor de valores más pequeños con menos latencia asociada y misma precisión.

En la figura 13 el zoom de la región de parámetros pequeños nos muestra que los valores cercanos al 0.68 de la función objetivo están pegados a la frontera contraria del zoom anterior: son valores mucho más chicos de lo que esperábamos dado nuestro rango original. Si bien esto significaría buenos resultados con poco costo de performance, como ya analizamos, valores pequeños de estos parámetros podrían significar menor tolerancia a ruidos muestrales (por ejemplo mayor variación entre predicciones de distintos datasets)[3].

Habiendo visto que valores particularmente grandes de nuestros parámetros no mejoraban la función objetivo tanto como esperábamos y que el rango macroscópico original que planteamos no parecía del todo concluso decidimos extender el espacio de búsqueda buscando valores aún más pequeños con los resultados vistos en la figura 14 donde la franja 40-52 para k y 150-350 para alfa supera los demás resultados vistos anteriormente.

De este modo no pudimos superar en una primera iteración de grillas y parámetros iniciales los resultados de kNN sin PCA agregando tal método. Esto no significa que tras ajustes en otros parámetros que afectan al método (como aquellos de la vectorización o el dataset de entrenamiento mismo) no podamos reiterar las experimentaciones mejorando los resultados y progresando sobre estos, además de que la cualidad de seleccionar componentes con mucha cohesión interna respecto de la vectorización original es muy útil para eliminar ruidos muestrales significando mayor estabilidad a cambios en datasets de entrenamiento y testeo que sin PCA.

Cabe destacar que según se acostumbra en procesamiento de lenguaje[3] natural[4] la cantidad de vocabulario se suele tomar como función creciente de la raiz cuadrada de los tokens[5], y que según nuestros datos, la raiz es un valor que ronda los 300[6]; bien dentro de la franja de componentes principales, que mencionamos antes.

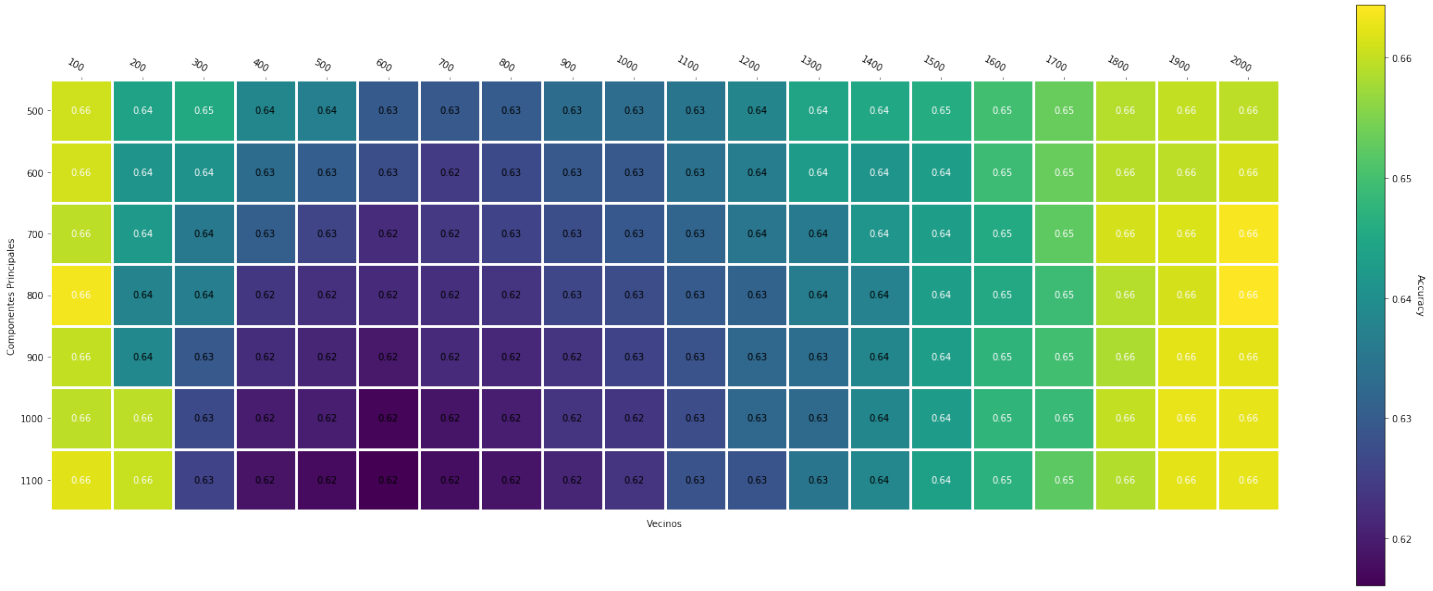


Figura 11: Resultado de la busqueda local sobre cada región **grande** de k100xα100. En cada una de las celdas se corrió una búsqueda local con "steps"de 10 unidades sobre ambos parámetros comenzando en el valor mostrado en los ejes hasta alcanzar un máximo local. Se muestran sobre el color, el accuracy logrado por el máximo local.

[3] Situación que pudimos comprobar al pasar del set de testing de 6000 casos de imdb_small al set de 100 de test_sample de la cátedra, donde el comportamiendo se volvía más errático seguramente por cuestiones de tamaño muestral.

[4] 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. $V > O(\sqrt{N})$.

[5] 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.

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

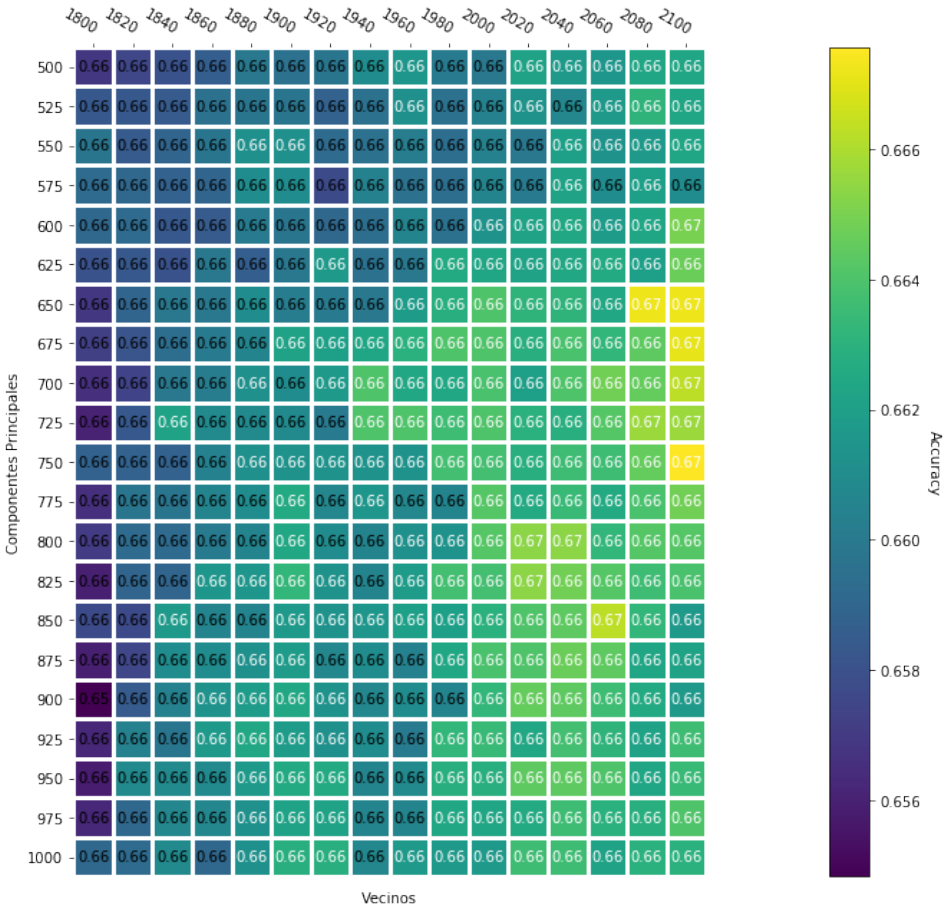


Figura 12: Resultado de la busqueda local sobre cada región de k20xα25 del **zoom sobre la meseta de valores altos de cantidad de vecinos**. En cada una de las celdas se corrió una búsqueda local con "steps"de 5 unidades para ambos parámetros comenzando en el valor mostrado en los ejes hasta alcanzar un **máximo local**. Se muestran sobre el color, el accuracy logrado por el máximo local.

3.6. Optimización de `min_df` y `max_df`

Se realizaron mediciones sobre el *accuracy* al variar los valores de `min_df` y `max_df` para la mejor configuración de parámetros conocida hasta el momento de realizar este experimento ($k = 1800$, sin PCA).

El hiperparámetro `max_df` permite ignorar los términos que aparecen con mucha frecuencia en los documentos, del set de datos. Por ejemplo, un valor de `max_df` = 0,7 indica al vectorizador que debe ignorar las palabras que aparecen en el 70 % o mas de las reseñas. Se espera que los términos de alta frecuencia sean inútiles para distinguir los documentos entre calificaciones positivas y negativas. Por otro lado, el hiperparámetro `min_df` indica al vectorizador que debe ignorar las palabras que se presentan en una porción baja de los documentos. Por ejemplo, un valor de `min_df` = 0,05 configura al vectorizador para ignorar los términos que aparecen solamente en el 5 % o menos de los documentos. Se conoce poco sobre los términos de baja frecuencia, por lo que incluirlos puede provocar problemas de *overfitting* en el modelo.

Como resultado, se obtuvo para `min_df` = 0,001 y `max_df` = 0,3 un *accuracy* de 0,77, el mejor valor alcanzado hasta el momento.

Puede verse que con un ligero aumento de `min_df`, el modelo pierde eficacia. Por otro lado, los mejores resultados fueron obtenidos para valores de `max_df` en la franja $0{,}2 \leq$ `max_df` $\leq 0{,}5$. Estos valores distan mucho de los valores originales que fijamos al hacer la optimización de hiperparámetros. Nuestra hipótesis al respecto es que las palabras de mayor frecuencia no sirven para distinguir los documentos entre sí, mientras que las palabras de menor frecuencia son las que mejor caracterizan la orientación de cada reseña.

Queda como experimento a seguir, repetir la optimización de hiperparámetros con distintos valores mas bajos de `min_df` y `max_df`. Esto tiene el beneficio añadido de reducir el tamaño de los datos, reduciendo el costo en tiempo de llevar a cabo futuros experimentos.

Figura 13: Resultado de la busqueda local sobre cada región de k20xα100 del **zoom sobre la meseta de valores chicos de cantidad de vecinos** . En cada una de las celdas, se corrió una búsqueda local con "steps"de 10 para la cantidad de vecinos y 50 para la cantidad de componentes principales comenzando en el valor mostrado en los ejes hasta alcanzar un **máximo local**. 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 con buen rendimiento basado en diferencias residuales e incrementales con una tolerancia $\epsilon = 10^{-6}$ que no costara mucho en términos de tiempo, sobre todo considerando que al aplicar deflación este método se llama continuamente para usar PCA.

Analizamos la precisión respecto del tamaño del set de entrenamiento al que se somete kNN con PCA notando que progresan de manera proporcional.

Vimos que, antes de optimizar la vectorización, sobre el set de entrenamiento y testeo de imdb_small provisto por la cátedra no fue posible conseguir accuracy scores por encima de 0,68 para kNN en una sola iteración de optimización de parámetros independientemente del agregado de PCA. El k que optimizaba tal situación era $k = 1826$ obteniendo accuracy 0,685.

La experimentación sobre PCA no fue enteramente inconclusiva: pudimos ver que la función objetivo en función de k acompaña (con una dimensión más) a aquella de la experimentación de kNN sin PCA sugiriendo que las mejoras que se puedan hacer en términos de vectorización sobre una alternativa impactarían de manera similar en la otra.

Vimos que finalmente el salto más notable en resultados no vino de los métodos de clasificación en sí sino de la vectorización.

Los métodos de experimentación y desarrollo planteados son la base sobre la cual se podría reiterar la optimización de parámetros de modo de conseguir progresivamente mejores resultados. 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. Se podría extender el análisis de recorte según frecuencias de la vectorización a kNN con PCA (a pesar de que nuestro mejor kNN con PCA no resultó ser más preciso que nuestro mejor kNN sin PCA) para buscar mejoras también ahí.

La opinión subyacente en los documentos queda caracterizada por las palabras que aparecen con menor frecuencia. Queda pendiente repetir la optimización de hiperparámetros para un valor de `max_df` en el rango $[0{,}2, 0{,}5]$.

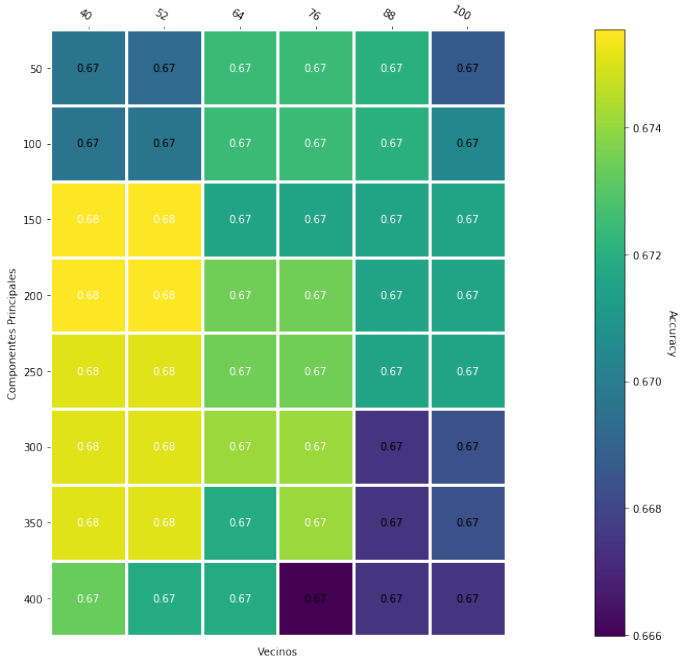


Figura 14: Resultado de la busqueda local sobre cada región **chica** de k12xα50. En cada una de las celdas, se corrió una búsqueda local con "steps"de 6 para la cantidad de vecinos y 25 para la cantidad de componentes principales comenzando en el valor mostrado en los ejes hasta alcanzar un **máximo local**. Se muestran sobre el color, el accuracy logrado por el máximo local.

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.

Figura 15: Valores de *accuracy* para un *grid search* sobre `min_df` y `max_df` para el modelo de KNN sin PCA y $k = 1800$. Se obtuvo un mejor rendimiento al probar nuevas configuraciones del vectorizador.