DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

2do cuatrimestre 2024 — Metodos Numericos

Integrante	LU
Francisco Fazzari	900/21
Felipe Miriuka	1693/21
Ignacio Fernández Oromendia	29/21

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

En este informe trataremos el problema de clasificación de películas en géneros usando el algoritmo KNN. Adicionalmente, con PCA y *k-fold-cross-validation* buscaremos los mejores parámetros para optimizar la clasificación. En PCA debemos obtener los autovalores y autovectores, para ello implementamos el método de la potencia con deflación.

Palabras clave:
Metodo de la Potencia, vectores, autovectores, autovalores, k-vecinos, set de entrenamiento, set de testeo, performance.

1. Introducción Teórica

En este informe evaluaremos nuestra implementación de algoritmos para clasificar textos que describen peliculas y lograr identificar el genero de las peliculas. Para esto vamos a usar vectores de atributos que indican las apariciones de palabras claves en los textos, y utilizando estos vector calculamos las distancias vectoriales entre las distintas peliculas para poder clasificarlas. Para las clasificaciones utilizamos una clasificacion por k-vecinos mas cercanos utilizando las distancias coseno de los vectores que representan los textos de las peliculas y luego el objetivo es poder comparar distintos conjuntos de datos de entrenamiento y de testeo para ver cuales resultan en los mejores resultados.

2. Desarrollo

2.1. K vecinos mas cercanos (KNN)

El método de K vecinos mas cercanos o KNN sirve para clasificar vectores según los k vectores que se encuentren mas cerca. El objetivo es poder aproximar a que grupo pertenece un vector utilizando el grupo con mayor cantidad de elementos entre los K vectores vecinos mas cercanos. Para lograr esto con nuestros vectores de películas primero contamos las apariciones de los tokens de las películas en vectores y luego utilizando las distancias coseno entre los distintos vectores podemos encontrar a sus K vecinos mas cercanos. La distancia coseno la calculamos haciendo

$$D_{coseno}(X, Y) = 1 - \frac{XY}{\|X\|\|Y\|}$$

Para encontrar a los vecinos más cercanos debemos comparar las distancias coseno y ordenarlas utilizando por las más cercanas y finalmente encontrando la moda de las películas es decir cual es el genero de película más presente.

Algorithm 1 KNN $(k, X\_train, X\_test, generos\_train)$

D = distCoseno(X_train, X_test)
cercanos = indiceKDistanciasCercanas(D)
generos = generos_train[cercanos]
return moda(generos)

Luego, queremos medir la performance de nuestro algoritmo, para ello vamos a generar un data frame de prueba diferente al data frame de entrenamiento. Luego, vamos a tomar el promedio de todas las películas de testeo sumando 1 si nuestro algoritmo acertó en la categorización de género y si no 0. En otras palabras la performance la medimos con la siguiente fórmula. Sean P las predicciones y *acertado* una función que devuelve 1 si y sólo si la predicción p_i corresponde con el género de la película i.

$$\text{performcane} = \frac{\sum_{i=1}^{|P|} \text{acertado}(p_i)}{|P|}$$

2.2. Método de la Potencia + Deflación

Para encontrar los autovalores y autovectores de una Matriz A *simétrica definida positiva* usaremos el **Método de la Potencia** junto con **Deflación** [2]. Para esta matriz podemos asegurar que $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ ya que para v_i, autovector asociado a λ_i:

$$v_i^t A v_i > 0 \rightarrow v_i^t \lambda_i v_i > 0 \rightarrow \lambda_i v_i^t v_i > 0 \rightarrow \lambda_i \|v_i\|_2^2$$

Luego como $\|v_i\|_2^2 > 0 \rightarrow \lambda_i > 0$. Por lo tanto, como A es *simétrica*, $\lambda_i \in \mathbb{R}$ y como también es *definida positiva*, $\lambda_i \in \mathbb{R}_{>0}$. Además le podemos dar un orden tal que $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$.

Ahora si, veamos como encontrar los autovalores y autovectores, para el primer autovalor vamos a generar un v aleatorio y luego iteraremos una cantidad necesaria de iteraciones hasta que v ya no sufra cambios significativos, aplicándole $Av/\|Av\|_2$ en cada iteración. Luego para encontrar el λ asociado a v debemos hacer el siguiente calculo v^tAv/v^tv.

Por otro lado, utilizamos un estructura *AutoData* para almacenar información interesante, aprovechando así el algoritmo para calcular el *error* y la cantidad de *iteraciones* necesarias para converger en λ para luego realizar un análisis del método utilizado.

Algorithm 2 Método de la Potencia $(A, niter, eps)$

$v = (-1, \ldots, -1)$
$w = v$

for $i \in [0, niter]$ **do**
  $v = Av/\|Av\|_2$
  if $\|v - w\|_\infty < eps$ **then** ▷ Criterio de parada
    break
  end if
  $w = v$
end for

$\lambda = v^tAv/v^tv$
$err = \|Av - \lambda v\|_2$

return AutoData(λ, v, i, err)

Una vez calculado el λ_1 y v_1 queremos los siguientes valores, para ello, haremos la siguiente operación sobre nuestra matriz A, $A' = A - \lambda_1 v_1 v_1^t$ y usaremos A' en el método de la potencia. Esta técnica es conocida como **deflación**, donde para cada λ_i y v_i generamos un A' para calcular λ_{i+1} y v_{i+1}.

Algorithm 3 Obtener_Autovalores $(A, niter, eps)$

$B = A$
$reuslt = \emptyset$

for $i \in [0, filas(B)]$ **do**
  AutoData res = Metodo_de_la_Potencia$(B, niter, eps)$
  result.add(res)
  $B = B - (res.\lambda \times res.v \times res.v^t)$ ▷ Deflación
end for

return *result*

2.3. KNN Cross Validation

Antes de evaluar nuestro algoritmo de KNN con los datos de prueba, vamos a realizar un *k-fold cross validation* con los datos de entrenamiento. Es decir, vamos a separar los datos de entrenamiento en k particiones (*folds*) con los géneros balanceados y del mismo tamaño. Para realizar la partición de datos, agregamos una columna *"Partition"* al data frame donde le asignamos a cada fila el número de partición al que pertenece, es decir, un número entre 0 y k-1. Luego vamos a elegir k-1 folds como datos de entrenamiento y 1 como dato de prueba, para eso simplemente filtramos el data frame para tener la partición i en la iteración i como prueba y las demás como entrenamiento. Realizamos todas las combinaciones posibles, con lo cual, cada partición k será al menos una vez dato de prueba. Con esta técnica podemos obtener el mejor K para nuestro algoritmo KNN para luego utilizarlo contra los datos de prueba. El algoritmo se ve así:

Algorithm 4 KNN_cross_validation $(k, Q, folds, dataFrame, X)$

```
particionar(dataFrame)
performances = Ø

for i ∈ [0, folds] do
    test_set = pariticiónIgualA(dataFrame, i)
    train_set = pariticiónDiferenteA(dataFrame, i)

    performances.add(clasificador_de_genero(k, X, train_set, test_set))
end for

return promedio(performances)
```

2.4. Análisis de componentes principales

Para poder mejor evaluar nuestros vectores queremos poder evaluarlos de manera mas rápida manteniendo los datos mas relevantes presentes, para hacer esto utilizamos el método de Análisis de componentes principales o PCA [1]. PCA es un algoritmo que permite encontrar cuales son los componentes que mejor explican los datos es decir los componentes con mayor covarianza para los datos, esto nos permite poder recortar la cantidad de componentes que tenemos que analizar sin perder demasiada información. Para realizar esto se hace una descomposición en auto-vectores y auto-valores de la matriz de varianza de los datos. Para hacer esto en código nosotros primero centramos todos los valores es decir les restamos la media y luego calculamos la matriz de covarianza de estos X, si hace falta calcular los auto-valores los calculamos y los guardamos y luego utilizando los auto-valores como sabemos que mientras mas grande el auto-valor mas grande es la covarianza, con ordenar los auto-valores obtenemos los de mayor covarianza primero luego hacemos la descomposición y retornamos la varianza y la matriz V que podemos utilizar para transformar los datos.

$$C = VDV^T$$

Algorithm 5 PCA(X)

```
C_train = matriz_covarianza(X)

w, V = separaAutoData(C_train)
indices = ordenarIndices(w)

w = w[indices]
V = V[:,indices]

varianza = cumsum(w)/sum(w)

return varianza, V
```

2.5. KNN y PCA *k-fold-cross-validation*

Ahora al algoritmo anterior vamos a sumarle PCA, así usamos cross validation para encontrar no solo el mejor k, sino que también vamos poder encontrar mejor p. En este algoritmo, encontraremos la combinación de p y k que maximice el promedio de aplicar la evaluación en cada uno de los fold. De esta manera vamos a encontrar los k y p mejor optimizados con cross validation sobre la información de entrenamiento para calcular clasificar sobre la información de testeo. Sin embargo, en la practica utilizamos un algoritmo que usa multiprocesamiento para cada fold para optimizar los tiempos de ejecución ya que calcular los folds son contextos disjuntos entre ellos.

Algorithm 6 mejores_parámetros($dataFrame, Q, folds$)

```
train_df = datosTrain(dataFrame)
particionar(train_df)

resultados = zeros(Q, Q)

for i ∈ [0, folds] do
    desarrollo_set = particiónIgualA(train_df, i)
    train_set = particiónDiferenteA(train_df, i)

    generos_train = generos(train_set)
    generos_desarrollo = generos(desarrollo_set)

    X_train = reducirMatriz(X, indices(train_set))
    X_desarrollo = reducirMatriz(X, indices(desarrollo_set))

    centrar(X_train)
    centrar(X_desarrollo)

    w, V = pca(X_train, i)

    for p ∈ [0, |filas(X_train)|] do
        X̂_train = X_train V[:, : p]
        X̂_desarrollo = X_desarrollo V[:, : p]

        for k ∈ |filas(train_df)| do
            predicciones = clasificar(k, X̂_train, X̂_test, generos_train)
            resultados[p, k]+ = performance(predicciones, generos_desarrollo)
        end for
    end for
end for

resultados = resultados / folds
p, k = max(resultados)
return p, k
```

Algorithm 7 pipline_final($dataFrame, Q, folds$)

X = matriz_tokens($Q, dataFrame$)
p, k = mejores_parametros($dataFrame, Q, X, folds$)

test_set = datosTest($dataFrame$)
train_set = datosTrain($dataFrame$)
generos_train = generos(train_set)
generos_test = generos(test_set)
X_train = reducirMatriz(X, indices(train_set))
X_test = reducirMatriz(X, indices(test_set))
centrar(X_train)
centrar(X_test)

$var, V = pca(X\_train)$

$\hat{X}\_train = X\_train\ V[:, :p]$
$\hat{X}\_test = X\_test\ V[:, :p]$

predicciones = clasificar($k, \hat{X}\_train, \hat{X}\_test$, generos_train)

return performance(predicciones, generos_test)

3. Resultados

3.1. Resultados con auto-valores cercanos

Un resultado importante para tener en cuenta es como se comporta el método de la potencia si los auto-valores son muy cercanos es decir son muy parecidos, para ver el efecto de esto podemos probar utilizando auto-valores fijos y luego agregando auto-valores que sean λ_1 - ϵ donde ϵ es la distancia entre los auto-valores. Probando para distintos ϵ podemos observar el efecto sobre el método de la potencia de tener auto-valores muy parecidos.

Utilizando los auto-valores $\{10, 10 - \epsilon, 5, 2, 1\}$ y calculando siempre con la misma matriz de auto-valores solo modificando el valor del ϵ podemos observar los siguientes efectos sobre los resultados del método de la potencia. La diagonal de la matriz tiene los auto-valores y luego multiplicamos la matriz.

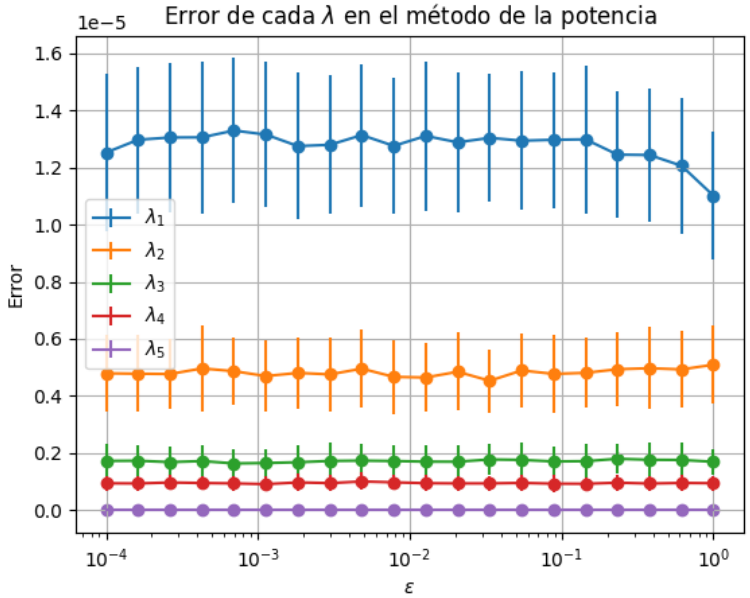

Figura 1: Error de método de la potencia para ϵ distintos

El primer auto-valor se ve afectado porque es muy parecido al segundo cuando el ϵ es chico y luego el motivo por el cual los otros auto-valores a los cuales no se les resta ϵ se ven afectados es porque al utilizar un auto-valores equivocado por el ϵ al momento de la deflación el resto de los auto-valores quedan afectados porque la deflación esta incorrectamente hecha porque no se encontró el auto-valor correcto en el paso anterior y por ende los demás auto-valores más chicos quedan afectos por el valor de ϵ.

Podemos observar en el gráfico que a medida que el ϵ se vuelve más pequeño es decir los auto-valores se acercan y son más parecidos el error al intentar calcularlos se vuelve más grande y viceversa cuando los valores de ϵ son muy grandes acercando se a 1 entonces los auto-valores están alejados y el error disminuye drásticamente hasta desaparecer.

3.2. Convergencia del Método de la Potencia

En el siguiente gráfico analizaremos en cuantas iteraciones converge cada autovalor para cada ϵ. En otras palabras, cuantas iteraciones le toma al método de la potencia para llegar al *i-ésimo* autovalor con una cantidad máxima de iteraciones de 10^8 y una tolerancia de error de 10^-7.

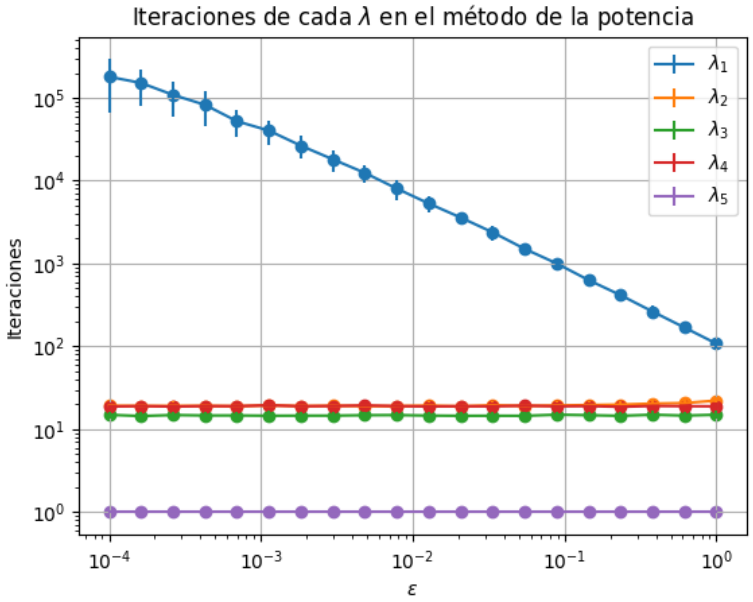

Figura 2: Performance de distintos k para distintos Q

Se puede observar en el gráfico que para el ϵ más chico el algoritmo en cada autovalor tomó muchos más pasos en converger que para los demás ϵ. Esto se debe a que en el ϵ más chico la diferencia entre el primer y el segundo autovalor es mucho menor que en los ϵ, por lo que al estar tan cerca ambos autovalores la convergencia tarda ya que la dirección de ambos autovectores es muy parecida y el método de la potencia tarda más iteraciones en converger en la dirección del primer autovalor.

3.3. Performance KNN

Una vez desarrollado el algoritmo de KNN queremos explorar los distintos valores que puede tomar K. Para ello, vamos a realizar un experimento donde tomamos $Q \in \{500, 1000, 5000\}$ tokens, conjuntos de prueba y entrenamiento y $k \in [1, |C\_prueba|]$, los cuales representan un 80 % y 20 % de los datos totales respectivamente para calcular la performance de KNN y poder distinguir los k más eficientes. Los resultados fueron estos:

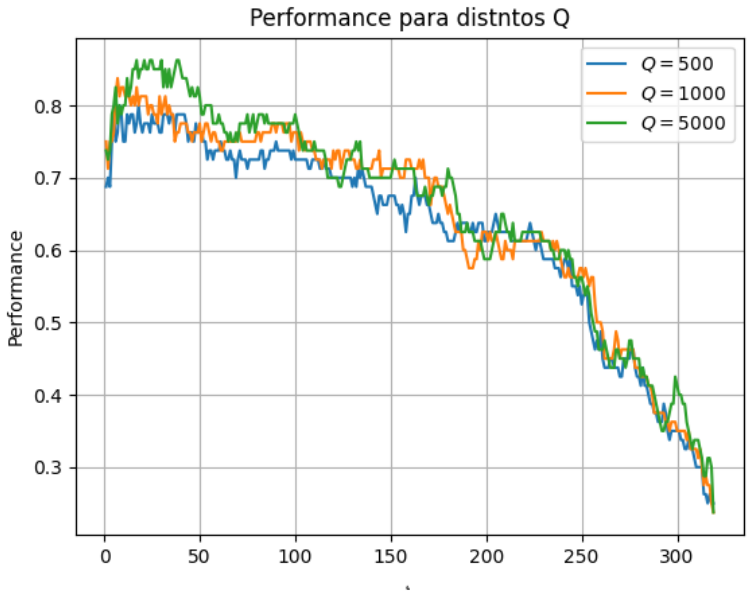

Figura 3: Performance de distintos k para distintos Q

Las observaciones indican que los K óptimos se encuentra entre 10 y 40 aproximadamente, es decir, si K es muy chico no estamos teniendo en cuenta muestras que nos ayudarían a la predicción. Por otro lado, a medida que tomamos más vecinos en consideración, la performance comienza a descreer cada vez más hasta el punto en el que K tiene el valor del total de los datos de entrenamiento. Es interesante ver que en para este último valor de K, la performance es de 0.25 lo que significa que clasifica aleatoriamente las películas dado que se está basando en todas las muestras.

3.4. Explorando el hiper-parámetro K

Para Q buscamos el mejor K. Quisiéramos saber como afecta el valor de K dependiendo de la cantidad de tokens que se utilizan para calificar los textos de las películas. En este experimento probamos buscar el mejor K es decir cual es la cantidad de vecinos optima a elegir para cada cantidad de tokens y luego intentamos buscar una explicación para los fenómenos que observamos.

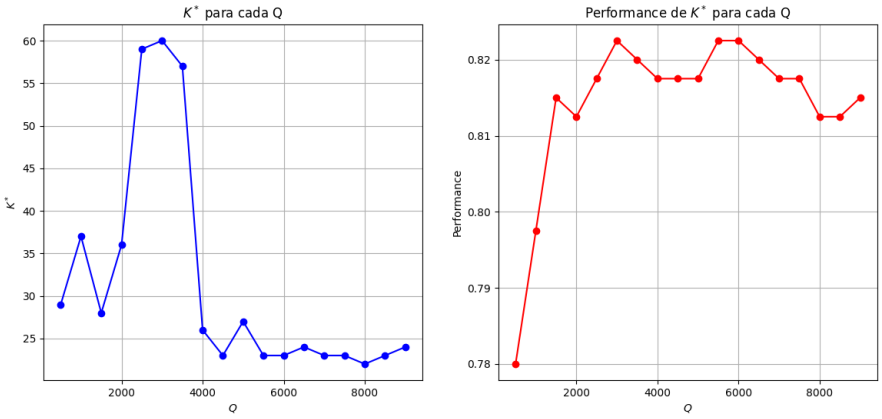

Figura 4:

Las observaciones indican que a medida que se eligen más tokens en medida general el K baja pues al tener todos los tokens disponibles se vuelve más difícil elegir entre los vecinos y por ende hay que reducir el numero de vecinos K que se elige para la predicción, tomando esto en cuenta también disminuye la performance.

3.5. Varianza en función de la cantidad de componentes p

Sabemos que podemos utilizar las p componentes principales para reducir la dimensionalidad de los datos, reduciendo el ruido y optimizando el análisis de los datos o la clasificación de los mismos. Sin embargo, para saber cuál es el mejor p a tener en cuenta vamos a analizar la varianza acumulada de las primeras p componentes eligiendo como criterio un 95 % de varianza, ya que consideramos que es lo suficientemente alto para tener información valiosa pero no tan alto como para tener mucho ruido. Lo que esperamos de este experimento es que la varianza vaya creciendo a medida que p crece pero no de forma lineal, ya que si fuera así entonces nunca convendría tener un p que no sean todas las componentes.

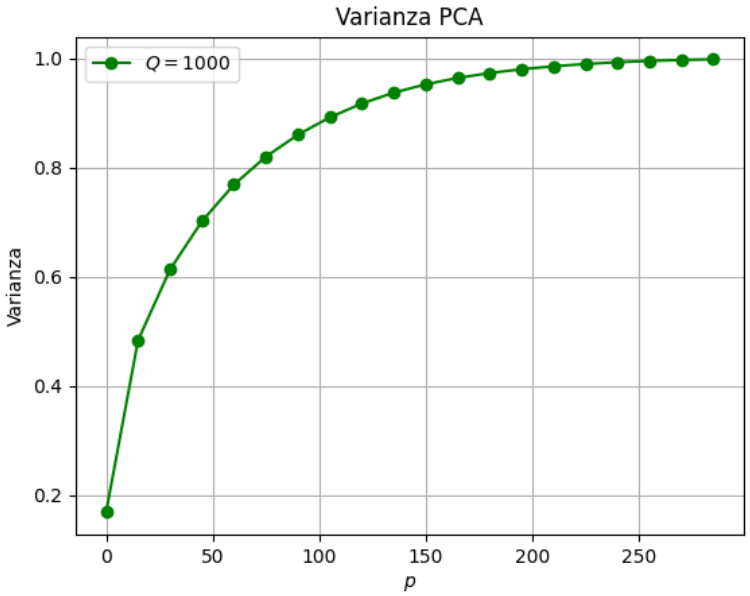

Figura 5: Varianza de las p componentes principales para $Q = 1000$

El resultado fue el esperado, podemos observar cómo converge p a 1 de forma logarítmica lo que tiene sentido, ya que luego de cierto punto solamente estamos agregando ruido al conjunto de datos. Por lo tanto tenemos una idea de como se comporta la varianza según el p que elijamos.

3.6. Pipeline Final

En este experimento realizaremos el estudio de los hiper-parámetros k y p en conjunto, donde haremos una *4-fold-cross-validation* de ambos a la vez con un $Q = 1000$, es decir los 1000 *tokens* más importantes. Luego de calcular los mejores k y p vamos a querer evaluar estos parámetros con los datos de prueba no tocados.

El resultado obtenido fue $p = 179$ y un $k = 12$, luego para la evaluación final obtuvimos una performance de 0,7 lo cuál nos sorprende porque en otras evaluaciones sin ningún p y obtuvimos mejor performance aunque sospechamos que esta discrepancia pueda deberse al conjunto de datos elegido. Para una mejor evaluación deberíamos conseguir más conjuntos de datos para realmente estar seguros de la implementación del algoritmo. Pudimos observar que es posible que exista una falla en alguna parte del algoritmo ya que sin tener en cuenta la p obtuvimos una performance de 0,83 con $k = 7$.

4. Conclusiones

En conclusión pudimos evaluar de varias formas los hiperparametros de los algoritmos de KNN y de PCA siempre utilizando el método de la potencia para encontrar los auto-valores asociados a las matrices, lo que pudimos observar es que al usar cross-validation para obtener mejores resultados con los mismos datos de entrenamiento y de prueba alternando los no siempre es fácil decidir cual es la cantidad de vecinos optima para el algoritmo de KNN y tampoco es fácil decir cual es el numero optimo de componentes principales. Cada uno de estos algoritmos tiene que ser testeado con distintos valores para poder obtener el numero optimo necesario, en el caso de KNN este numero depende fuertemente de la cantidad de tokens que son utilizados es decir de la cantidad de componentes dentro de los vectores de las distancias y para el caso de PCA la cantidad de componentes principales también depende ya que al utilizar demasiados estos terminan generando ruido y terminan empeorando la performance de nuestras predicciones. En general al utilizar demasiados tokens o demasiados componentes principales o demasiados vecinos las predicciones casi siempre terminan empeorando ya que se introduce demasiado ruido en las muestras y eso afecta los resultados. Algo que podemos concluir es que cuando utilizamos el algoritmo de KNN cuantos mas tokens hay peor es la inferencia y para PCA mientras mas componentes hay también. Por otro lado para el método de la potencia hay que tener en cuenta los auto-valores que queremos estimar ya que si estos son muy parecidos las estimaciones del método de la potencia empiezan a fallar y peor todavía el auto-valor erróneo termina pasando su error al resto de los auto-valores que se calculan después al utilizar la deflación.

Referencias

[1] David G. Stork Richard O. Duda, Peter E. Hart. *Pattern classification.* Wiley, 2nd edition, 2001. Chapter 3.8.1.

[2] David S. Watkins. *Fundamentals of Matrix Computations.* JOHN WILEY SONS, INC, 2rd edition, 2002. Chapter 5.3.