DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Análisis de componentes principales y procesamiento de lenguaje

15 de diciembre de 2024 — Métodos Numéricos

Integrante	LU	Correo electrónico
Campoverde, Axel	258/22	a.i.cpvd3000@gmail.com
Gómez, Sebastián	713/22	sebasgomezp01@gmail.com
Riera, Leandro	658/18	leandro.riera.m@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón Cero + Infinito)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Conmutador: (+54 11) 5285-9721 / 5285-7400
https://dc.uba.ar

Resumen

En este trabajo nos proponemos crear un clasificador de género de películas simplificado basándonos en una base de datos de películas con 4 opciones para el género (crime, romance, science-fiction, western) y su sinopsis ya tokenizada. Para eso vamos a implementar el algoritmo de K-Vecinos-Cercanos y utilizarlo usando la distancia coseno entre la representación vectorial de las peliculas. Adicionalmente implementamos el método de la potencia con deflación para obtener los autovalores y autovectores de una matriz y lo utilizamos para poder bajar la dimensión del problema mediante PCA. Para finalizar realizamos una exploración de los hiperparámetros usando Cross-Validation, seleccionamos los mejores y probamos el modelo usando accuracy como medición objetivo y observamos este valor.
keywords: Aprendizaje automático, KNN, componentes principales, autovalores, autovectores

1. Introducción

En la actualidad, es casi imposible no escuchar sobre el procesamiento del lenguaje, entre los diversos modelos se encuentran los modelos de clasificacion. En este trabajo practico, exploraremos la tecnica de aprendizaje automático conocido como k-vecinos mas cercanos (KNN) con el objetivo de desarrollar un sistema que sea capaz de clasificar películas a partir de la extraccion de palabras clave de la sinopsis que nos conllevara a predecir el genero de la misma , ademas como la cantidades de componentes principales puede influir en los resultados. Por ultimo, utilizaremos el proceso de validación cruzada para entrenar a nuestro modelo, con el fin adquirir un hiperparametro que haga a nuestro sistema de clasificacion generalizable y libre sesgos.

En el trabajo nos presenta cuatrocientas peliculas de cuatro posibles generos, tal que debemos crear un modelo que pueda predecir basado en su sinopsis a que genero pertence, el proceso de llevar un texto a una representacion numerica conlleva a tokenizar las palabras claves y darles un valor a cada pelicula dependiendo de la frecuencias de cada palabra, por lo que una pelicula podria estar en un espacio con altas dimensiones. Esto presenta un problema, ya que las distancias entre puntos pueden volverse poco representartivas.

Para abordar este problema, implementamos el analisis de los componentes principales (PCA), tal que busca una tranformacion lineal que proyecte nuestros datos de la muestra a un nuevo sistema de menor dimension, de forma que se mantenga la maxima varianza entre los datos, para ello partimos de una matriz de covarianza, en el que utilizaremos el metodo de la potencia con deflacion, para buscar iterativamente los autovalores y autovectores dominantes.

Por ultimo, como nos interesa que este modelo pueda predecir el genero de cualquier pelicula, buscamos la efectividad partiendo del la cantidad de vecinos nos pueda dar los mejores resultados, al contar con una cantidad fija de datos, es fundamental encontrar un el mejor valor para la cantidad de vecinos cercanos en el algoritmo KNN, por lo que la validacion de nuestro modelo dependera de cual sera el k valor que pueda darnos mejores resultados para cualquier conjunto de datos, por lo que aplicamos la validacion cruzada.

2. Desarrollo y métodos

2.1. Distancia

El desafio principal al iniciar con el modelo de clasificación radica en convertir el texto, un tipo de datos no estructurado, en una representacion numerica que nos permita distinguir entre los distintos géneros. Para lograr esto, es necesario el preprocesamiento del texto, en donde se eliminará a palabras que no aportan valor en la clasificación, como artículos y preposiciones. Ademas, se aplicara la normalización de palabras conjugadas, que se refiere a reducir una palabra a su raiz o a su forma base.

Tras el prepocesamiento, la representación de una película pasa a ser mediante las palabras que aportan la información suficiente sobre su contenido. Luego, se puede determinar que una pelicula pertenece a un género X si tiene palabras similares a películas del mismo género.

Finalmente, podemos contar la frecuencia de estas palabras de cada película, lo que nos permitiría tener una representación numérica adecuada para la clasificación. El poder llegar a una representación numérica nos ayudará a determinar que tan semenjante es una película con otra a partir de las distancias que ambas puedan tener dentro de un espacio vectorial, por lo que si una película es similar o cercana a otra, podriamos concluir que pertenecen al mismo género.

La manera para determinar la distancia entre dos vectores (películas) estará determinada por la distancia coseno, en donde nos basamos en el coseno del angulo entre dos vectores, tal que nos permita medir cuan cercanos se encuentra en términos de dirección por lo que para dos vectores x y y en $\mathbb{R}^n$, donde n seria la cantidad de apariciones de las principales palabras de una película, por lo que tenemos la siguiente formula:

$$D_{coseno}(x, y) = 1 - \frac{x \cdot y}{\|x\|\|y\|}$$

Tenemos la situación de que buscamos la distancias entre distintas películas por lo que para calcular las distancias entre todas las películas procesamos en forma matricial tal que aplicamos el siguiente código:

DISTANCIA-COSENO(A, B) :
1 $norm_A \leftarrow [\|fila_i(A)\|_2 \ for \ i \ in \ \{1, ..n\}]$
2 $norm_B \leftarrow [\|fila_i(B)\|_2 \ for \ i \ in \ \{1, ..n\}]$
3 $NORM\_inv_A, NORM\_inv_B \leftarrow I$
4 **for** $i \leftarrow 0$ **to** $length(A[0])$:
5 **do** $NORM\_inv_A[i][i] \leftarrow (1/max(1e^{-24}, norm_A[i]))$
6 **for** $i \leftarrow 0$ **to** $length(B[0])$:
7 **do** $NORM\_inv_B[i][i] \leftarrow (1/max(1e^{-24}, norm_B[i]))$
8 return $I - (NORM\_inv_A \cdot A \cdot B^t \cdot NORM\_inv_B)$

Cabe aclarar que la utilización de $max(1e^{-24}, norm_X[i]))$ es para evitar la indeterminación en el caso que norma de algún vector sea cero.

2.2. K-vecinos más cercanos (KNN)

Una vez definida y calculada la distancia entre dos películas podemos encontrar las k más cercanas. El resultado que retorna DISTANCIA-COSENO(A, B) es una matriz en $\mathbb{R}^{n \times m}$, que en este caso n es

la cantidad de películas de A y m las películas de B, la forma para determinar los k mas cercanos va a estar dado por los k valores mas bajos de cada fila correspondiente a un película i. Esto nos permite encontrar las k películas más similares, por lo que podríamos deducir a partir de estas el género de la misma, asumiendo que mientras mas frecuente sea el género dado por las películas similares, podríamos suponer que nuestra película a predecir será mas probable que sea del mismo género que se repite.

Para esto primero preprocesamos la información de las películas de forma tal que su representación sea un vector donde en cada posición tiene la cantidad de apariciones de una palabra en su sinopsis, utilizando las primeras n palabras con mayor cantidad de apariciones en total. Teniendo esto en cuenta realizamos la siguiente implementación del algoritmo conocido como KNN

KNN($test, train, k, n$) :

1 $X\_test \leftarrow$ PREPROCESAMIENTO($test, n$)
2 $X\_train \leftarrow$ PREPROCESAMIENTO($train, n$)
3 $distancias \leftarrow$ DISTANCIA-COSENO(X_test, X_train)
4 return $argsort(distancias)[: k]$

Donde los parámetros *test* y *train* tienen la información de las películas que se usan para entrenar y testear al modelo. Adicionalmente la función toma el parámetro k que indica cuántos vecinos vamos a considerar a la hora de predecir el género de la película y n que indica cuántas palabras tenemos en cuenta a la hora del preprocesado. Sobre cómo elegir el parámetro k se desarrolla en la siguiente sección.

2.3. Cross-Validation

Decidir cuántos vecinos considerar para *knn* no es trivial. Si elegimos demasiados pocos estos pueden no ser representativos del vecindario mientras que si elegimos demasiados podriamos estar introduciendo ruido que empeoraría nuestra predicción. Podríamos evaluar cada valor de k con todos nuestros datos de entrenamiento y test pero esto podría generar que nuestro valor de k esté 'sobreajustado' a nuestros datos, es decir, que sea óptimo sólo para estos datos y si cambiamos los datos de entrenamiento o de test el desempeño podría caer drásticamente. Para solucionar esto y elegir el valor óptimo realizamos una implementación de $Cross - Validation$. Este algoritmo consiste en dividir nuestros datos de entrenamiento en una cantidad de partes (o folds), tomar un fold y usarlo para testear un valor de k fijo entrenando al modelo con los folds restantes. Este proceso se repite para todas las partes y tomamos el promedio para cada valor de k a analizar. De esta forma nos aseguramos que el valor de k elegido es el óptimo independientemente de con qué datos lo entrenemos.

La implementación resultante se da en 2 partes, una primera parte donde dividimos los datos de forma tal de asegurarnos que se mantenga la proporción de géneros en cada fold y una segunda parte donde evaluamos los distintos valores de k para cada fold.

Para separar los datos usamos fuertemenete que para todos los experimentos libamos a usar 4 folds y que nuestros datos sólo tenía películas de 4 géneros.

```
CROSS-VALIDATION(train_data, k_values)
folds ← SPLIT-DATA(train_data)
accuracies ← {}
for k in k_values
    do accuracies[k] ← []
       for j ← 0 to 4
           do test_fold ← folds[j]
              train_fold ← folds[(j + 1) %4] + folds[(j + 2) %4] + folds[(j + 3) %4
              accuracies[k].append(ACCURACY(PREDICCION(test_fold, train_fold)))
return accuracies
```

2.4. Método de la potencia con deflación

Para poder calcular autovalores y autovectores de forma aproximada realizamos una implementación del Método de la potencia con deflación en C++. Luego creamos una interfaz entre Python y C++ con Pybind11, que nos permite compilar el programa escrito en C++ y convertirlo en un módulo de Python, de manera tal que podemos utilizar en Python las operaciones definidas en el módulo implementadas originalmente en C++. Para la manipulación de matrices y vectores en C++ utilizamos la librería Eige.

Bajo la hipótesis de que una matriz A es diagonalizable, y por ende tiene una base de autovectores, y tiene un autovalor dominante, es decir, mayor en módulo al resto, el método de la potencia es un método iterativo que toma esta matriz A y devuelve el autovalor dominante y su autovector asociado, que denominaremos autovector dominante. Como hipótesis adicional necesitamos un vector v que, al ser escrito como combinación lineal de los autovectores de A, el factor que multiplica al autovector dominante no sea 0, algo que se consigue con probabilidad 1 usando un vector aleatorio. En cada iteración del método se multiplica la matriz A a derecha por el vector v, y luego se normaliza el vector resultante. Si tanto la matriz A como el vector v cumplen con las condiciones iniciales, luego de cierta cantidad de iteraciones esto va a converger al autovector dominante de A, asociado al autovalor de mayor módulo.

En caso de que la matriz A no tenga un único autovector asociado al autovalor dominante, el método de la potencia igualmente converge hacia un v que pertenece al subespacio generado por los autovectores asociados a éste autovalor.

Ahora bien, esto solo nos sirve para calcular el autovalor/autovector dominante de la matriz. En el caso de que nos interese calcular todos los autovectores, necesitamos realizar una deflación a la matriz cada vez que calculamos un autovector. ¿Qué es una deflación? Si llamamos λ_1 al autovalor dominante y v_1 a su autovector asociado, para calcular el siguiente autovalor de la matriz, queremos construir una matriz en la que λ_1 ya no sea un autovalor dominante. Si la matriz tiene base ortogonal de autovectores, este proceso de 'remover' el autovalor dominante se llama deflación, y se puede realizar mediante el método de Hotelling con la siguiente operación:
$A^{k+1} = A^k$ - $\lambda_1 v_1 v_1^t$ donde A^k representa la matriz de la cual ya obtuvimos el autovalor y autovector dominante, y A^{k+1} es la deflación de esta matriz que tiene como autovalores $\{0, \lambda_2, \dots \lambda_n\}$ siendo λ_1 el autovalor dominante de y $\{\lambda_2, \dots, \lambda_n\}$ los autovalores restantes de la A^k original. Para ver que esto funciona podemos plantear qué pasa con los autovectores v_i. Si $i = 1$ entonces $(A^k - \lambda_1 v_1 v_1^t)v_1 = A^k v_1 - \lambda_1 v_1 v_1^t v_1$ y como podemos tomar v_1 tal que $\|v_1\|_2 = 1$ el resultado es $A^k v_1 - \lambda_1 v_1 1 = \lambda_1 v_1 - \lambda_1 v_1 = 0 v_1$. Si $i \neq 1$, al ser vectores ortogonales entre si, $v_1^t v_i = 0$. Luego $(A^k - \lambda_1 v_1 v_1^t)v_i = A^k v_i - \lambda_1 v_1 0 = A^k v_i = \lambda_i v_i$.

Repitiendo este proceso podemos obtener todos los autovalores y autovectores de la matriz en orden del mayor autovalor en módulo al menor. Para la implementación realizamos dos funciones, una que se encarga de realizar el método de la potencia como tal, devolviendo el autovalor y el

autovector dominante de una matriz cuadrada A y la cantidad de iteraciones que tomó en converger, dato que nos puede ser útil para analizar el comportamiento y la efIciencia del método. Esta función se llama *powerIteration* y recibe como parámetros, además de la matriz A, un *int* que indica un número máximo de iteraciones y un *epsilon* que utilizaremos para el criterio de parada cuyo valor por defecto es 10^-20. Para decidir cuándo parar, en cada iteración chequeamos que $\|vAnterior - v\|_\infty$ sea menor a *epsilon*, cuando esto se cumple salimos del bucle y devolvemos los valores calculados y la cantidad de pasos en una tupla de 3 elementos. El pseudocódigo es el siguiente:

```
POWERITERATION(A, niter, épsilon) :
 1  n ← A.rows()
 2  v ← randomNumbersVector(n)
 3  pasos ← 0
 4  for i ← 0 to (niter − 1) :
 5        do pasos ← pasos + 1
 6           vAnterior ← v
 7           v ← A ∗ v
 8           v ← v/‖v‖_2
 9           normaInfinito ← ‖vAnterior − v‖_∞
10           if normaInfinito < epsilon
11              then break;
12  eigenvalue ← v^t ∗ (A ∗ v)
13  eigenvalue ← eigenvalue/v^t ∗ v
14  result ← tuple(eigenvalue, v, pasos)
15  return result
```

La otra función necesaria para calcular todos los autovalores y autovectores de la matriz es la que se encarga de ejecutar *powerIteration* sobre la matriz y luego realizar la deflación, repitiendo esto n veces podemos obtener los n autovalores y autovectores de la matriz. Esta función es *eigen*, que recibe como parámetros la matriz A, un *num* de tipo *int* que nos indica cuántos autovalores y autovectores de la matriz queremos calcular, en el caso de querer todos este parámetro debe ser la cantidad de filas/columas de A, un tercer parámetro *niter* para la cantidad de iteraciones y un *épsilón* para el criterio de parada:

```
EIGEN(A, num, niter, epsilon) :
 1  copyA ← A
 2  eigenvalues ← vector(num)
 3  eigenvectors ← matrix(A.rows(), num)
 4  for i ← 0 to (num − 1) :
 5        do eigens ← powerIteration(A, niter, epsilon)
 6           eigenvalues[i] ← eigens.get(o)
 7           eigenvectors.col[i] ← eigens.get(1)
 8           copyA ← copyA − ((eigenvalues[i] ∗ eigenvectors.col[i]) ∗ eigenvectors.col[i]^t)
 9  result ← tuple(eigenvalue, eigenvectors)
10  return result
```

2.5. PCA

Si bien la sinopsis de la película consiste en una gran variedad de palabras, a la hora de predecir el género de la misma no todas aportan la misma información. Díficil sería adivinar el género de una película por la presencia de la palabra 'protagonista' en su sinopsis mientras que si vemos frases como 'tecnología alienigena', 'viajes temporales' o 'Civilizaciones intergalácticas en guerra' podríamos inclinarnos por la ciencia ficción. Para incluir esto en nuestro modelo realizamos un análisis de las componentes principales (PCA por sus siglas en inglés). Construimos la matriz de covarianza que indica, para dos palabras, 'qué tanto aparecen juntas' y con esta información podemos quedarnos con las combinaciones de palabras que más varianza aportan a la muestra. Más formalmente, si X es la matriz que tiene en cada fila la representación vectorial de cada película y en cada columna la cantidad de apariciones de una palabra, centramos los datos realizando

$$X' = X - media(X)$$

donde *media* calcula la media por columna, de esta forma nos aseguramos de que X' represente la cantidad de apariciones de cada token en relación al promedio de apariciones de dicho token. Y con esta nueva matriz calculamos la matriz de covarianza C

$$C = \frac{(X')^t X'}{n-1}$$

Como esta matriz C es una matriz simétrica, podemos asegurar que tiene base ortonormal de autovectores y, si bien no podemos asegurar que todos sus autovalores sean distintos, esto es muy poco probable e igualmente solo reduciría la velocidad de convergencia. Por lo tanto podemos afirmar que cumple con la hipótesis del método de la potencia y deflación. A esta matriz le aplicamos el método de la potencia con deflación y calculamos los autovalores y autovectores. Estos últimos nos dan una base en la cual podemos proyectar nuestros datos ordenada según cuánta información aporta cada componente principal. Una vez hecho esto podemos quedarnos con un subconjunto de estas componentes reduciendo la dimensionalidad del problema ahorrando tiempo de cómputo y eliminando ruido.

3. Resultados y experimentación

Nuestra base de datos consiste en un conjunto de 300 películas, las cuales están divididas equitativamente entre los 4 géneros que consideramos (crime, romance, science fiction, western). Luego de realizar una tokenización de la sinopsis de cada película para extraer las palabras claves, obtuvimos una cantidad de 9581 palabras distintas de las sinopsis de todas las películas. El gráfico 1 muestra la frecuencia de las 50 más comunes.

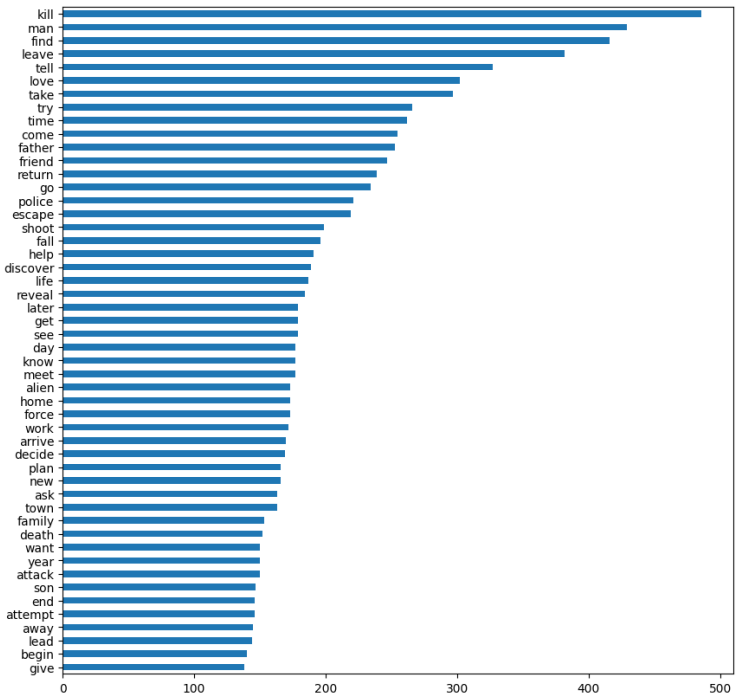

Figura 1: distribución de los 50 tokens más repetidos

Tomamos las 1000 palabras más frecuentes asumiendo que serian suficientes para poder definir las distancia para distintos géneros. Y, con el objetivo de ver que tan marcadas van a estar las distancias entre cada genero, calculamos la distancia entre todas las películas, usando solo de las 1000 palabras mas frecuentes.

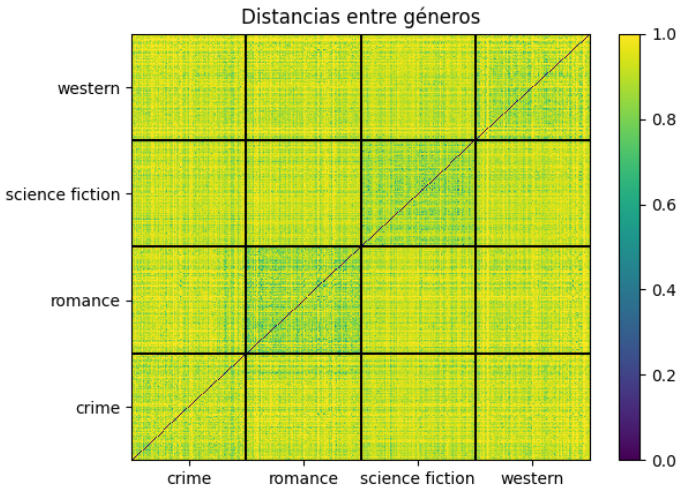

Figura 2: distancia normalizada entre cada par de películas según su género

A partir del gráfico 2, podemos observar que las películas que pertenecen a los géneros de romance

y ciencia ficción presentaran menor dispersión en términos de la distancia. Esto sugiere que es más probable obtener una mejor predicción para estos dos géneros. Esta hipótesis la pondremos a prueba una vez ajustados los hiperparametros del modelo.

3.1. Tests y resultados del Método de la potencia con deflación

Para probar la implementación del método de la potencia con deflación en el cálculo de autovectores y autovalores de una matriz cuadrada, realizamos diferentes tests utilizando una matriz cuyos autovalores conocemos de antemano, esto lo logramos construyendo una matriz a partir de una matriz diagonal, cuyos elementos son elegidos en forma aleatoria en el rango $[0, 100]$, luego nos construimos una matriz de Houselder Q utiliazando la siguiente fórmula: $Q = I - 2vv^T$, con $||v||_2 = 1$. Al ser Q una matriz de Houselder sabemos que es ortogonal, y de este modo podemos construirnos una matriz A de la siguiente forma:

$$A = Q \begin{pmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & d_n \end{pmatrix} Q^T$$

cuyos autovalores sabemos que son $d_1, d_2, ..., d_n$, y cuyos autovectores asociados van a ser las columnas de Q, a las cuales podemos llamar $q_1, q_2, ..., q_n$. De este modo obtenemos una matriz A con autovalores y autovectores conocidos ideal para probar nuestra implementación del método de la potencia con deflación. Consideramos diferentes casos: el primero con autovalores de tipo enteros sin repetidos en el rango $[0, 100]$, el segundo con autovalores de tipo float sin repetidos en el rando $[0, 100]$ y el tercero con autovalores de tipo entero en el mismo rando pero asegurándonos de que exista al menos un autovalor repetido. Para cada caso construimos matrices desde $2x2$ a $100x100$, a cada una le calculamos sus autovalores y autovectores utilizando el método *eigen* y luego los comparamos utilizando la función de $Numpyallclose$, que verifica si todos los elementos de dos vectores son aproximadamente iguales dentro de una tolerancia especificada, utilizamos la tolerancia por defecto. Otro detalle importante es que comparamos los números en valor absoluto ya que el método de la potencia nos puede devolver un múltiplo de los autovectores en ciertos casos, pero lo que nos interesa de un autovector es su dirección, Para el caso de la matriz con autovalores enteros no repetidos y con autovalores de tipo floats no repetidos, obtuvimos una prueba exitosa, pero para el caso en que tenemos autovalores enteros repetidos, luego de varias repeticiones los assert nunca nos pasaron de matrices de 24x24, para chequear que esto no se deba a la limitación en la cantidad de iteraciones o en la tolerancia aumentamos los valores de las iteraciones hasta 1.000.000.000 y los de épsilon hasta 1e-60 pero no obtuvimos mejores resultados.

3.2. Análisis de la cantidad pasos para converger y el error

Tomando una matriz con los autovalores $\{10, 10 - \epsilon, 5, 2, 1\}$ con ϵ tomando valores log-espaciados en el rango $[10^6, 10^0]$, para cada valor de épsilón creamos 100 matrices con dichos autovalores utilizando nuevamente el truco de las matrices de Householder mencionado anteriormente. Luego calculamos el error de la siguiente forma: $\|Av_i - \lambda_i v_i\|_2$, siendo λ_i y v_i el íésimo autovalor y autovector respectivamente. Luego tomamos el promedio y el desvío estándar de los de los 100 errores obtenidos. Luego graficamos el promedio y utilizamos el desvió estándar como barra de error, obteniendo en el caso del autovalor y autovector dominantes λ_1 y v_1 el gráfico 3:

Podemos ver como, al menos para el primer autovalor, para valores bajos de epsilon, como la diferencia relativa entre el autovalor principal y el siguiente es muy chica, tanto el error como la cantidad de pasos hasta la convergencia es muy grande, siendo estos últimos limitados por la

máxima cantidad de pasos (50000). A medida que el valor de epsilon aumentaba y aumentaba esta diferencia relativa, el error disminuyó hasta el threshold y la cantidad de pasos hasta la convergencia bajo hasta una media de casi 0.

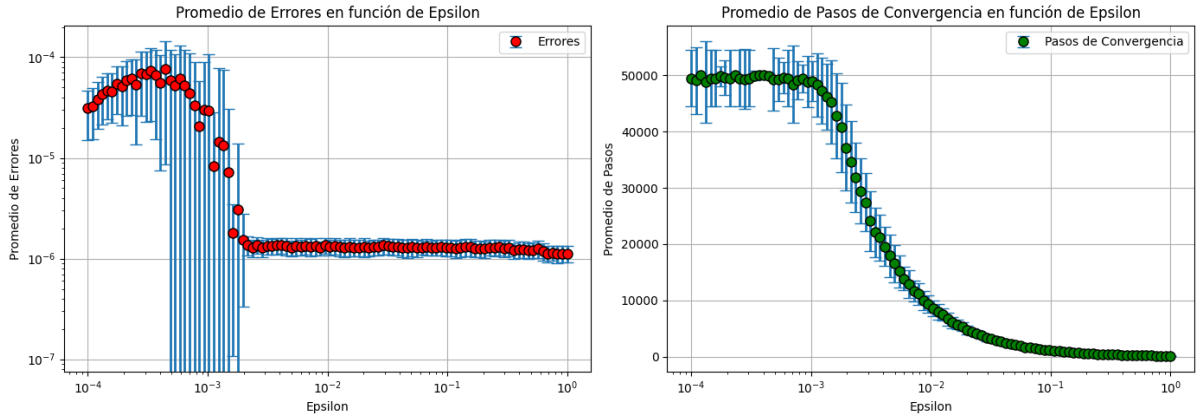

Figura 3: resultados para el primer autovalor

Luego nos propusimos ver qué pasaba con el resto de autovalores y repetimos el experimento pero analizando los valores para todos los autovalores y no sólo para el primero.

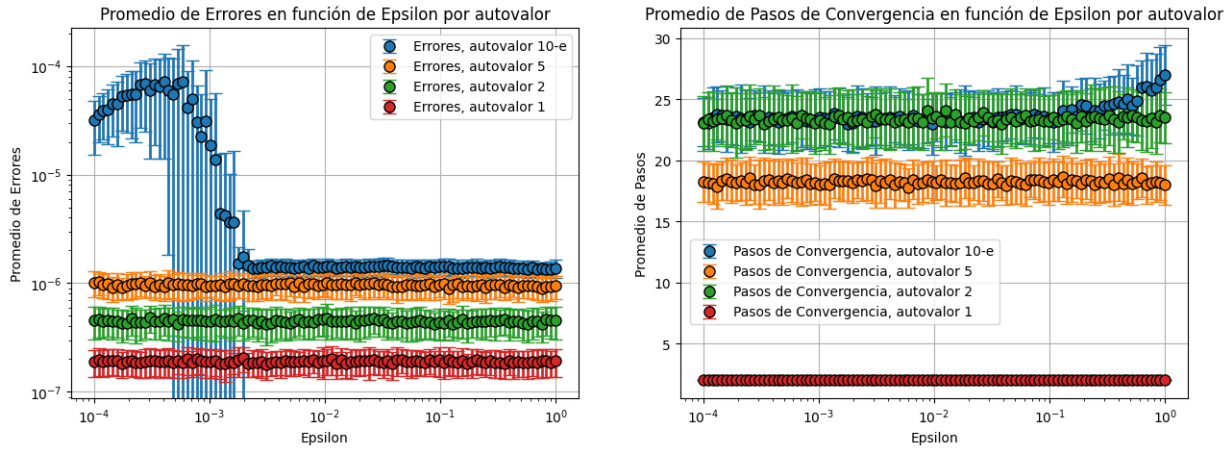

Figura 4: resultados para el resto de autovalores

Como la velocidad de convergencia del método esta dada por el cociente entre el autovalor principal y el siguiente, en el gráfico 4, podemos ver que para el autovalor $10 - \epsilon$, a diferencia del primer autovalor, la cantidad de pasos es constante y mucho menor casi independientemente del valor de ϵ (aumenta un poco para valores altos de ϵ ya que disminuye la diferencia con el siguiente autovalor). Esta cantidad de pasos es similar a la observada para el autovalor 2 ya que poseen la misma diferencia relativa (1/2). Para el autovalor 5 la diferencia es mayor (5/2) y por eso la cantidad de pasos es menor. Para el autovalor 1, debido a la deflación, la matriz que usamos para calcularlo tiene como autovalores el 1 y 0 (o un valor cercano a 0, debido al error numérico), es por esto que la diferncia relativa es tan grande que la cantidad de pasos media hasta la convergencia es cercana a 0.

Si miramos el error, observamos que para los primeros valores de ϵ, el error obtenido para el autovalor $10 - \epsilon$ es mayor ya que se arrastra el error obtenido al intentar aproximar el primer

autovalor (10). Una vez que ϵ es suficientemete grande, y el error de aproximacion para el primer autovalor se estabiliza, lo mismo pasa con el error para el segundo. Esto sucede para valores de ϵ del orden de 10^{-3}. Para el resto de autovalores vemos que el error es constante sin importar el valor de ϵ dado que el error inicial solo afecta a los primeros autovalores. Este error es mayor para los autovalores mas grande. Esto puede explicarse ya que, al tener magnitud, magnifican los errores de aproximacion del metodo, sin embargo, como la diferencia relativa es poca (menor o igual a 1/2) esta diferencia en el error no es tan grande.

3.3. Clasificacion de géneros

En primer lugar probamos cómo sería el porcentaje de aciertos del modelo usando distintas cantidades de palabras usando sólo *knn* con $k = 5$. Para eso separamos un 80 % de los datos para entrenamiento dejando un 20 % para testeo y obtuvimos los resultados del grafico 5.

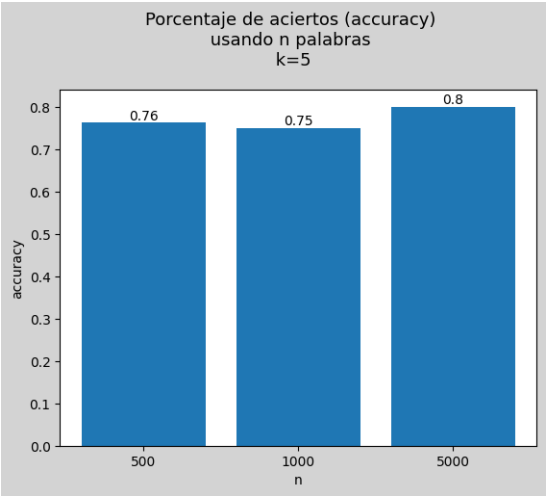

Figura 5: Accuracy para knn usando distintos valores de k

Sin embargo para mejorar estos valores decidimos mejorar la muestra buscando un valor óptimo para k. Para eso, usando $Cross - Validation$ probamos con distintos valores de k y obtuvimos el gráfico 6. Donde se ve que tener en cuenta 500 palabras da resultados significativamente peores a tener en cuenta 1000 palabras pero esa mejoría no continua si aumentamos a 5000. Es decir, 1000 palabras son suficientes como para tener buenos resultados y no representa un costo computacional extremadamente alto, es por esto que a partir de ahora utilizaremos esta cantidad para el resto de experimentos.

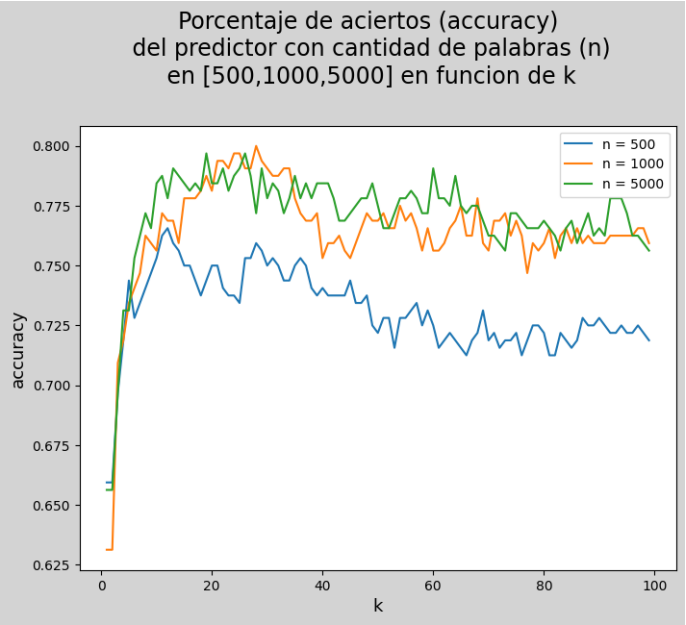

Figura 6: Accuracy en función de k para distintos valores de n

Finalmente utilizamos PCA para intentar mejorar estos valores. Primero analizamos la cantidad de varianza explicada por la cantidad de componentes principales tenidas en cuenta.

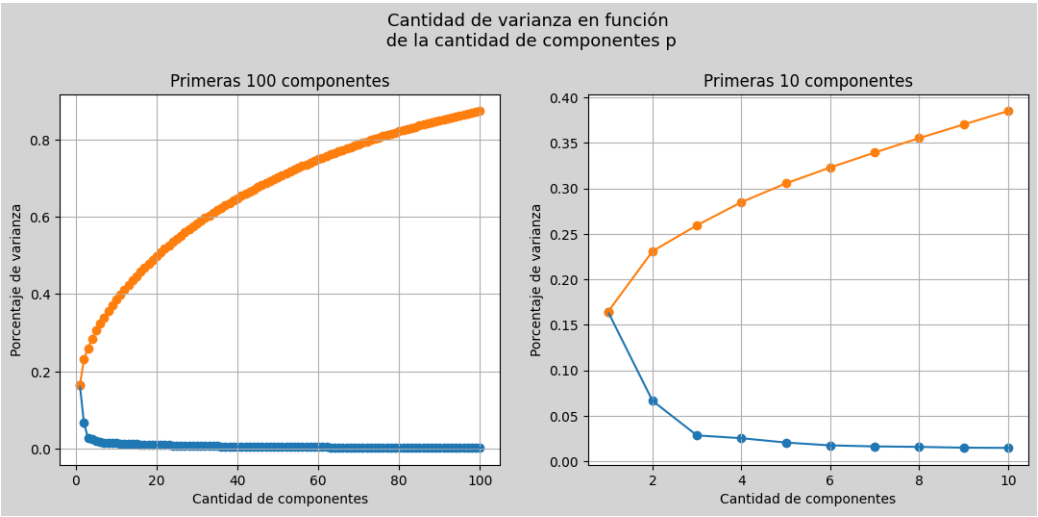

Figura 7: Cantidad de varianza explicada por componentes principales

En el gráfico 7 podemos ver que las primeras componentes principales son las que mayor cantidad de varianza explican siendo las primeras 80 (sobre 1000) las que explican más del 80 % del total. Sin embargo para un mejor análisis decidimos hacer una exploración en conjunto de los hiperparámetros k y p.

Probamos para valores de p (cantidad de componentes principales a tomar en cuenta) entre 2 y 200 y valores de k en $[5, 15, 25, 35, 45, 55, 65, 75, 85, 95]$. El valor de accuracy promedio entre las 4 particiones del $Cross - Validation$ para cada par de valores de n y k se puede ver en el gráfico 8.

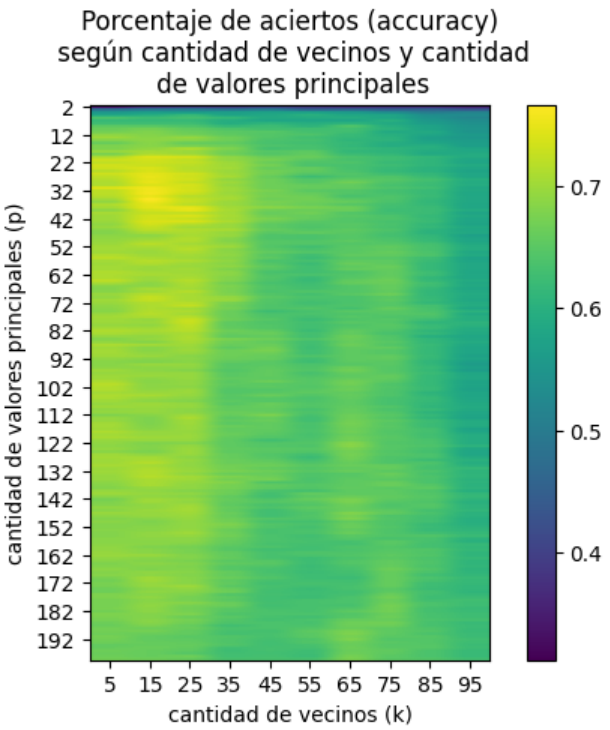

Figura 8: Exploracion de accuracy para valores de n y k

Al probar estos valores en los datos de entrenamiento vimos que la combinación óptima de parámetros era de $k = 15$ y $p = 35$ por lo que probamos entrenar al modelo con todos los datos de entrenamiento con esos parámetros y ver los aciertos en todos los datos de test. En este caso el valor de accuracy fue de 0.825.

4. Análisis por genero

Al comienzo de la sección planteamos una hipótesis sobre el desempeño del modelo para cada género. En el gráfico 2 observamos, a simple vista, que los géneros de *romance* y $science fiction$ parecían tener menos distancia entre las películas de su mismo género en comparación a las películas de los géneros *crime* y *western*. Para formalizar un poco más esto calculamos el promedio de las distancias entre películas que comparten genero y obtuvimos los resultados del gráfico 9. Donde podemos ver que, efectivamente, estos géneros presentaban menor distancia promedio.

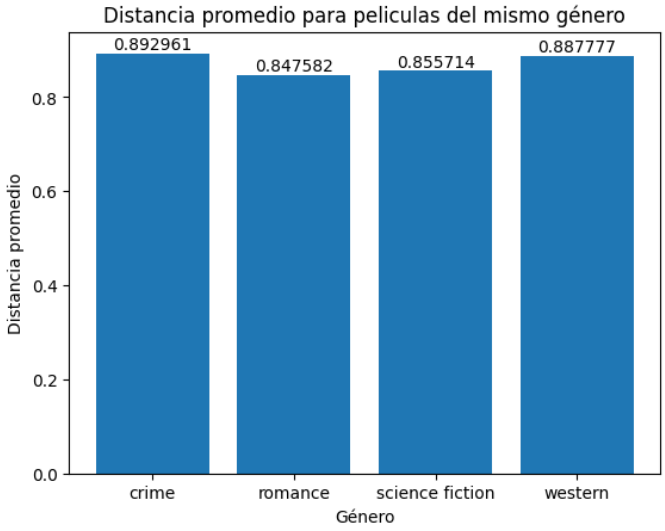

Figura 9: Distancia promedio entre películas del mismo género

Para ver cuánto impacto tiene esta diferencia en el modelo analizamos el porcentaje de aciertos por género usando los hiperparámetros conseguidos en el análisis anterior. El resultado se puede ver en el gráfico 10 donde cada fila muestra que porcentaje de las películas de ese género fueron clasificadas en los distintos géneros (por ejemplo, de las películas de genero *crime*, un 70 % fue clasificada como *crime*, un 10 % como *sciencefiction* y un 20 % como *western*)

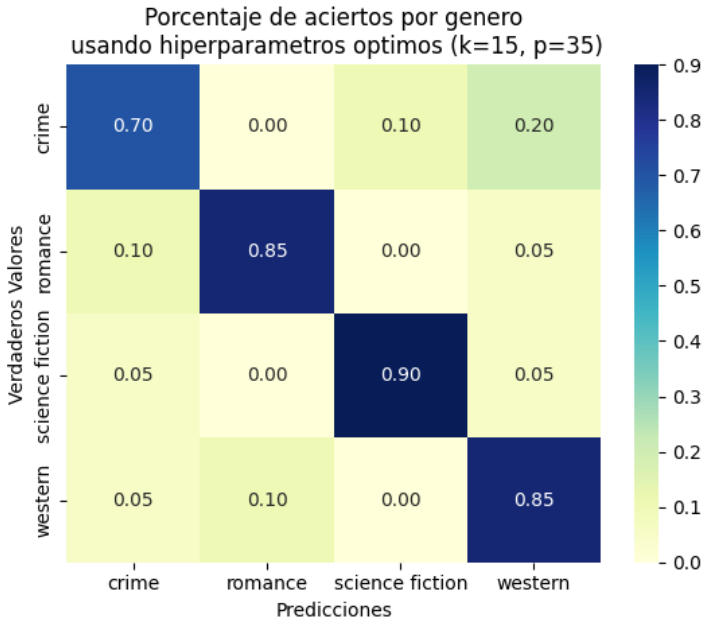

Figura 10: Confusion matrix del modelo optimizado

Si bien el genero *sciencefiction* fue el que obtuvo mayor porcentaje de aciertos, la diferencia no fue tanta como la esperada respecto al género *western* e incluso nula entre este último y el genero *romance*. Pero como las distancias originales fueron calculadas sin PCA, quizás fue esto lo que

generó que la diferencia fuera menor.

Para comprobar esto realizamos el mismo análisis para el mismo valor de k pero sin realizar PCA.

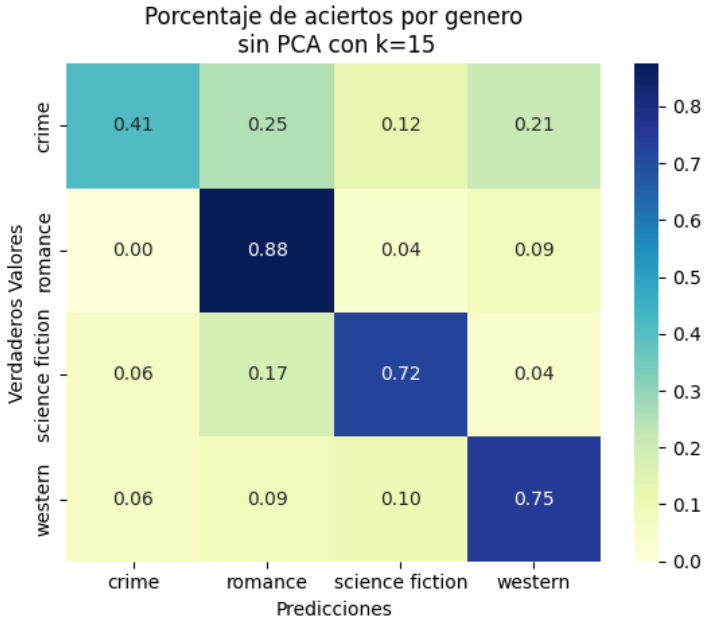

Figura 11: Confusion matrix del modelo sin PCA

Como vemos en el gráfico 11. Si bien el desempeño del modelo para las películas del género *crime* se desplomó obteniendo un porcentaje de aciertos de solo 41 %, para las películas de $science fiction$ el porcentaje de aciertos también decayó incluso por debajo del género *western*. Esto nos indica que, si bien la distancia promedio entre películas del mismo genero puede tener impacto en el desempeño del modelo final, esto no es seguro ni definitivo, mucho menos si realizamos una proyección en los componentes principales.

5. Conclusión

En este trabajo se pudo desarrollar un modelo para la clasificación por género de películas mediante técnicas de procesamiento de texto y el entrenamiento de un modelo para optimizar resultados. Por medio de la implementación del algoritmo KNN y el análisis de componentes principales, se pudo reducir la dimensionalidad de los datos con el fin de mejorar el rendimiento del modelo. Así mismo, el uso de la técnica de validación cruzada permitió determinar el valor óptimo para el número de vecinos y la cantidad de valores principales a considerar, mejorando así la precisión de las predicciones y evitando el sobreajuste.

Pudimos ver que, la cantidad de palabras (o tokens) a considerar afecta la performance del modelo pero hasta cierto punto. También observamos que la cantidad de vecinos, es decir, el parámetro k del modelo, tiene un gran impacto en qué tan bueno es el modelo a la hora de predecir, si el valor es muy bajo la cantidad no llega a ser suficiente como para ser representativa del vecindario pero si es muy alto podríamos estar agregando 'vecinos' más alejados y que solo aportan ruido a la predicción. Lo mismo podemos decir de la cantidad de valores principales tenidos en cuenta. Como la mayor cantidad de varianza se explica con las primeras componentes principales, podemos

disminuir la dimensionalidad del problema mejorando la performance, teniendo en cuenta que si este valor, el parámetro p del modelo, es muy bajo perdemos información pero si es muy grande consideramos dimensiones que no agregan información e incluso podrían introducir ruido.

Si bien el método de predicción basado en sus vecindario (knn) es simple, obtuvimos un porcentaje de acierto mayor al 80 %, esto gracias al análisis de componentes principales y una buena elección de hiperparámetros que pudimos obtener gracias a la validación cruzada.