DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP 2

22 de julio de 2025 Métodos Numéricos

Grupo: 11

Palabras clave:
KNN, PCA, Clasificación de imagenes, Método de la Potencia

Integrante	LU	Correo electrónico
González Correas, Álvaro	233/22	alvarogonzalezc4@gmail.com
Serna, Joaquín	361/19	joaquinserna@cryptolab.net
Guzzo, Martina	30/22	martina.f.guzzo@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Resumen

El objetivo de este informe es desarrollar una solución al problema de reconocimiento y clasificación de imágenes en categorías, en particular clasificación de clases de prendas. Para resolver este problema implementamos un clasificador basado en el algoritmo de KNN, o K Nearest Neighbours. Además, realizamos un Análisis de Componentes Principales (PCA), ya que puede haber dimensiones en las imágenes (o pixeles) que sean poco representativas a la hora de definir qué categoría le corresponde a la imagen. Realizamos validación cruzada utilizando los datos de entrenamiento para definir qué parámetros de KNN (cantidad de vecinos) y PCA (cantidad de componentes aceptadas) optimizan la performance, y finalmente utilizando los parámetros óptimos encontrados clasificamos los datos del conjunto de prueba y calculamos la exactitud de las predicciones. Para llevar a cabo esta resolución utilizamos el método de la potencia con deflación con el objetivo de obtener la transformación asociada a PCA.

2. Introducción Teórica

2.1. KNN: K Vecinos Más Cercanos

Dada una imagen x, el algoritmo de KNN (k vecinos más cercanos) la compara con una base de datos de imágenes cuyas categorías son conocidas, tomando como criterio de similitud entre dos imágenes la distancia coseno entre ellas. Una vez calculadas todas las distancias, se seleccionan las k imágenes más similares a la que se quiere clasificar, que serán aquellas cuya distancia sea menor. Finalmente, se toma la moda de las categorías de este conjunto de k vecinos. Es decir, se busca responder la pregunta de dadas las k imágenes más parecidas a x, ¿cuál es la categoría que más se repite entre ellas? Una vez obtenida una respuesta, se predice que esta será la categoría x.

2.2. Covarianza y Correlación

La covarianza entre dos vectores x e y se puede expresar mediante la siguiente ecuación:

$$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) = \frac{(\mathbf{x} - \mu_x) \cdot (\mathbf{y} - \mu_y)}{n - 1}$$

Donde

μ representa el valor medio del vector. Esta fórmula es equivalente a realizar el producto interno de los vectores centrados y normalizada por el tamaño del vector menos uno.
A partir de esta fórmula podemos definir la correlación entre x e y como la covarianza normalizada para que el resultado pertenezca al rango $[-1, 1]$.

$$\mathrm{Corr}(\mathbf{x}, \mathbf{y}) = \frac{(\mathbf{x} - \mu_x) \cdot (\mathbf{y} - \mu_y)}{\sqrt{(\mathbf{x} - \mu_x) \cdot (\mathbf{x} - \mu_x)\,(\mathbf{y} - \mu_y) \cdot (\mathbf{y} - \mu_y)}}$$

Utilizando los vectores centrados $x_i - \mu_{x_i}$ como columnas podemos crear la matriz X. Podemos calcular la matriz de covarianza de X de la siguiente forma:

$$C = \frac{X^t X}{n - 1}$$

Sobre esta matriz C, que es semidefinida positiva y simétrica por ser de la forma $X^t X$ y por lo tanto no tiene autovalores negativos, vamos a realizar el análisis de componentes principales utilizando el método de la potencia como se explica en la siguiente sección.

2.3. PCA: Análisis de Componentes Principales

Con respecto a la clasificación de imágenes, algunos pixeles de una imagen serán más representativos que otros y/o serán más significativos a la hora de comparar una imagen con otras para poder encasillarla en alguna categoría. Dada una matriz $\mathbf{X} \in \mathbb{R}^{m \times n}$ donde cada una de las m filas representa una imagen compuesta por n atributos, se utiliza el análisis de componentes principales (PCA) para encontrar un cambio de base de las filas de forma tal que las dimensiones se ordenen en componentes que explican los datos de mayor a menor. Es decir, se busca obtener una transformación de la imagen donde las columnas estén ordenadas de mayor a menor según que tan representativas son para poder clasificar una imagen.
Esto se logra realizando una descomposición en autovectores y autovalores de la matriz de covarianza de los datos.

$$C = VDV^t$$

Donde V tiene en sus columnas los autovectores de X y D tiene en su diagonal los autovalores de X. Realizando el producto XV es posible transformar los datos X en la nueva base.
Se puede reducir la dimensionalidad de los datos aceptando solo p componentes principales, es decir utilizando las primeras p columnas de V. En la sección de desarrollo de este informe exploraremos el hiperparámetro p y buscaremos sus valores óptimos.

2.4. Método de la Potencia

El método de la potencia es un algoritmo iterativo utilizado en álgebra lineal para encontrar el autovalor de mayor módulo junto con su respectivo autovector de una matriz cuadrada. Sin embargo, para que el método de la potencia funcione correctamente y converja al valor propio dominante, se deben cumplir ciertas precondiciones que vamos a pedir en nuestra implementación. La primera es que los autovalores sean distintos:

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

La segunda es que la matriz cuadrada tenga una base ortonormal de autovectores. Por otro lado, al ser un algoritmo iterativo, el criterio de parada en el método de la potencia es crucial para determinar cuándo detener el proceso. El objetivo es garantizar que el algoritmo haya convergido suficientemente cerca del autovalor dominante y su correspondiente autovector. Existen muchos criterios de parada en la práctica pero en nuestra implementación usaremos (con $\epsilon = 1e^{-7}$):

$$\|v_n - v_{n-1}\|_\infty < \epsilon$$

3. Desarrollo

3.1. Algoritmos

3.1.1. KNN

Algorithm 1 DISTANCIA

procedure DISTANCIA(X,Y)
- $Y \leftarrow Y - media\_por\_fila(Y)$
- $X \leftarrow X - media\_por\_fila(X)$
- $Y \leftarrow Y / norma\_por\_fila(Y)$
- $X \leftarrow X / norma\_por\_fila(X)$
- $dist \leftarrow XY^t$
- $dist \leftarrow 1 - dist$
- **return** $dist$

Tenemos dos matrices donde cada fila es una imagen. Para calcular la distancia coseno de forma matricial entre ambas matrices, primero les restamos a ambas matrices la media de cada fila para centrar los datos. El siguiente paso es normalizar cada fila dividiéndola por su norma 2. Realizamos el producto matricial entre X e Y^t. Finalmente, a una matriz de unos le restamos *dist* para obtener la matriz de distancias. Indexando *dist* en i obtenemos las distancias entre todos los elementos de Y, que será X_{train}, y el iésimo elemento de X, que será X_{dev} . Logramos así la siguiente expresión de distancia para todo par de imágenes de X e Y:

$$D_{\text{coseno}}\left(\mathbf{x}, \mathbf{y}\right) = 1 - \text{Corr}(\mathbf{x}, \mathbf{y}) = 1 - \frac{(\mathbf{x} - \mu_x) \cdot (\mathbf{y} - \mu_y)}{\sqrt{(\mathbf{x} - \mu_x) \cdot (\mathbf{x} - \mu_x)\,(\mathbf{y} - \mu_y) \cdot (\mathbf{y} - \mu_y)}}$$

Algorithm 2 KNN

```
1: procedure KNN(k, matriz_distancias, y_train)
2:     k_vecinos ← argsort_por_fila(matriz_distancias)[:, : k]
3:     prendas_cercanas ← y_train[k_vecinos]
4:     modas ← moda_por_fila(prendas_cercanas)
5:     predictions ← array(modas)
6:     return predictions
```

El algoritmo de KNN recibe como parámetro *matriz_distancias*, que guarda para cada imagen de X_{dev}, su distancia con todas las imágenes de X_{train}. A partir de esta matriz, creamos la matriz *k_vecinos* (que tiene la misma cantidad de filas y k columnas), donde realizaremos de forma matricial la siguiente operación para cada fila: utilizando la función de *numpy* $'argsort'$, tomaremos los k índices que se correspondan a las distancias más cortas para la iésima imagen y los guardaremos en orden en la iésima fila de la matriz *k_vecinos*.
Luego, creamos la matriz *prendas_cercanas* (que tiene las mismas dimensiones que *k_vecinos*), donde reemplazamos cada índice de *k_vecinos* por el tipo de prenda que se corresponde a esa imagen. Entonces, tenemos una matriz donde la iésima fila contiene qué tipo de prenda son los k vecinos más cercanos a la iésima imagen de X_{dev}. Tomamos la moda de cada fila y la guardamos en un array, cuya iésima posición tendrá la predicción de tipo de prenda para la iésima imagen, basada en qué tipo de prenda se repitió más entre los k vecinos más cercanos.

Algorithm 3 EXACTITUD

```
1: procedure EXACTITUD(k, X_train, y_train, X_dev, y_dev)
2:     matriz_distancias ← distancia(X_dev, X_train)
3:     predictions ← KNN(k, matriz_distancia, y_train)
4:     aciertos ← (predictions == y_dev)
5:     exactitud ← promedio(aciertos)
6:     return exactitud
```

La función exactitud toma un conjunto de datos de desarrollo y otros de entrenamiento junto con sus respectivas clasificaciones y calcula la tasa de aciertos obtenida comparando la clasificación de los datos (que es conocida) con las predicciones obtenidas por el clasificador de imágenes basado en KNN.
Lo primero que hacemos es calcular la matriz de distancias entre los elementos de X_{dev} y los de X_{train}. Le pasamos la matriz de distancias a KNN, y obtenemos como resultado el vector de predicciones para cada imagen de X_{dev}. Finalemente, comparamos posición a posición el arreglo *predictions* y el arreglo y_{dev}, que contiene las clasificaciones reales de las imágenes de X_{dev}. Si ambos arreglos coinciden en una posición, significa que la predicción hecha fue correcta, así que guardamos un 1 en la posición correspondiente. Caso contrario, guardamos un 0 ya que el clasificador realizó una predicción equivocada. Devolvemos el promedio del arreglo *aciertos* para obtener el porcentaje de aciertos de nuestro clasificador.

3.1.2. Cross Validation

Algorithm 4 MATRIZ DE COVARIANZA

procedure $matriz\_covarianza(X)$
$X\_centrado \leftarrow X - media\_por\_columna(X)$
$C \leftarrow \frac{X\_centrado^t X\_centrado}{len(X)-1}$ **return** C

Algorithm 5 5-FOLD CROSS VALIDATION

procedure $cross\_validation(X_{train}, y_{train})$
$max\_exactitud \leftarrow 0$
$optimos \leftarrow \{-1, -1\}$
$split_X \leftarrow dividir(X_{train}, 5)$
$split_y \leftarrow dividir(y_{train}, 5)$
for each $particion_X, particion_y$ in $split_X, split_y$ **do**
$\hat{X}_{dev} \leftarrow particion_X$
$\hat{y}_{dev} \leftarrow particion_y$
$\hat{X}_{train} \leftarrow X_{train} - \hat{X}_{dev}$
$\hat{y}_{train} \leftarrow y_{train} - \hat{y}_{dev}$
$C \leftarrow matriz\_covarianza(\hat{X}_{train})$
$V \leftarrow PCA(C)$
for $k \leftarrow 1$ **to** 15 **do**
for $p \leftarrow 1$ **to** 100 **do**
$X^*_{train} = \hat{X}_{train}(V[:, :p])$
$X^*_{dev} = \hat{X}_{dev}(V[:, :p])$
$res \leftarrow exactitud(k, X^*_{train}, \hat{y}_{train}, X^*_{dev}, \hat{y}_{dev})$
if $res > max\_exactitud$ **then**
$max\_exactitud \leftarrow res$
$optimos = \{k, p\}$
end if
end for
end for
end for
return *optimos*

En la etapa de validación cruzada, se divide X_{train} en 5 partes balanceadas y se toma cada una de ellas como conjunto de desarrollo para cada fase del entrenamiento. Una vez fijados los conjuntos de entrenamiento y de desarrollo, se calcula la matriz de covarianza y se la diagonaliza mediante el método de la potencia para relizar un análisis de componentes principales. En este paso se exploran los hiperparámetros k y p, devolviendo al final del algoritmo la tupla (k,p) que generó la mejor performance.

Algorithm 6 PIPELINE FINAL

1: **procedure** $pipeline\_final(X_{train}, y_{train}, X_{test}, y_{test})$
2: $optimos \leftarrow cross\_validation(X_{train}, y_{train})$
3: $k \leftarrow optimos.first$
4: $p \leftarrow optimos.second$
5: $C \leftarrow matriz\_covarianza(X_{train})$
6: $V \leftarrow PCA(C)$
7: $\hat{X}_{train} \leftarrow \hat{X}_{train} V[:, :p]$
8: $\hat{X}_{test} \leftarrow \hat{X}_{test} V[:, :p]$
9: **return** $exactitud(k, \hat{X}_{train}, y_{train}, \hat{X}_{test})$

En el paso final, se utiliza validación cruzada con los datos de entrenamiento para definir cuáles k (cantidad de vecinos) y p (componentes aceptadas) son óptimos para la tarea de clasificar imágenes. Una vez obtenidos los valores óptimos, estos valores son fijados y se calcula la exactitud de clasificar los datos de prueba con estos parámetros.

3.1.3. PCA y Método de la Potencia

Algorithm 7 MetPotencia(B,niter,ϵ)

procedure
$v \leftarrow v.setRandom()$
$vant \leftarrow v$
$i \leftarrow 0$
while $i < niter$ **do**
$v \leftarrow \frac{Bv}{\|Bv\|}$
if $\|v - vant\|_\infty < \epsilon$ **then**
$break$
end if
$vant \leftarrow v$
$i++$
end while
$\lambda \leftarrow \frac{v^t B v}{v^t v}$ =0

Nuestro algoritmo tomaría como entrada la matriz B, el x_0 inicial (el cual debe tener norma 2 igual a 1), la cantidad de iteraciones máxima que hace (en nuestra implementación son 10000) y el ϵ para el criterio de parada. Una vez que tenemos el método de la potencia implementado el próximo paso es hacerlo con deflación, con el objetivo de encontrar todos los autovalores y sus respecivos autovectores de la matriz. Nosotros implementamos el método de Hotelling que consiste en: Una vez que tenemos el primer autovalor λ_1 con su respectivo autovector v_1 calculamos una nueva matriz:

$$B - \lambda_1 v_1 v_1^t$$

Esta nueva matriz tendrá como autovalores $0, \lambda_2, ..., \lambda_n$ con autovectores asociados $v_1, ..., v_n$

$$(B - \lambda_1 v_1 v_1^t) v_1 = B v_1 - \lambda_1 v_1 (v_1^t v_1) = \lambda_1 v_1 - \lambda_1 v_1 = 0 v_1$$

$$(B - \lambda_1 v_1 v_1^t) v_i = B v_i - \lambda_1 v_1 (v_1^t v_i) = \lambda_i v_i$$

Acá utilizamos que tenemos una base ortonormal de autovectores. Sin esta precondición esta demostración sería incorrecta.

Algorithm 8 Método de la Potencia

procedure METPOTENCIA($B, niter, \epsilon$)
  $res \leftarrow []$
  $C \leftarrow B$
  $i \leftarrow 0$
  while $i < C.size()$ **do**
    $res.append(MetPotencia(C, niter, \epsilon))$
    $v \leftarrow res[i].first()$
    $vtrans \leftarrow v.T$
    $vgrande \leftarrow v * vtrans$
    $vgrande \leftarrow res[i].second() * vgrande$
    $C \leftarrow C - vgrande$
    $i++$
  end while
  return res

4. Resultados y Discusión

4.1. Optimización de Hiperparámetros

Esta sección del trabajo práctico trata de la exploración conjunta de los hiperparámetros en la etapa de validación cruzada. Sea x una imagen que queremos clasificar, describimos a continuación las hipótesis hechas y los resultados obtenidos mediante la experimentación.

4.1.1. k

El hiperparámetro k hace referencia a la cantidad de vecinos serán que considerados en el algortimo KNN. Es decir, se quiere definir con cuántas imágenes "similares"será comparada x para tomar la moda entre ellas y hacer una predicción sobre a qué categoría pertenecerá x.
Exploramos el rango $(1, ..., 15)$ para k. Antes de hallar el valor óptimo de k, predijimos que el mejor valor posible de k sería relativamente bajo. Luego de ejecutar el algoritmo de *cross validation* verificamos que efectivamente se obtiene la mejor performance cuando $k = 4$. Suponemos que esto se debe a que al incrementar la cantidad de vecinos considerados, estamos permitiendo comparar con distancias cada vez más grande, o con imágenes cada vez menos parecidas a x. Si tenemos en cuenta muchos vecinos, corremos un mayor riesgo de que se incluyan en los k vecinos seleccionados imágenes que no sean de la misma clase que x.

4.1.2. p

El hiperparámetro p representa la cantidad de componentes principales consideradas para la clasificación de imágenes. Sabemos que hay atributos de las imágenes que tienen más significancia que otros a la hora de clasificar, entonces nos interesa ordenar los atributos de mayor a menor importancia y limitar el problema a una cantidad determinada de componentes aceptadas para incrementar la probabilidad de acierto al categorizar una imagen.
Durante esta etapa del trabajo práctico, decidimos limitar el rango a explorar para p a 100 componentes como máximo. En la sección 4.3 describimos la motivación de esta decisión. Luego de ejecutar el programa de *cross validation*, obtuvimos que el valor de p que genera mejor performance es $p = 58$.

4.2. Pipeline Final

Una vez calculados los hiperparámetros que optimizan la performance en la etapa de cross validation (como vimos en la sección anterior), fijamos estos valores y ejecutamos la función de

exactitud sobre el conjunto de datos de validación (X_{test}). Dados k y p, óptimos, queremos calcular la tasa de aciertos que es posible obtener sobre el conjunto de prueba, la cual esperamos que sea máxima.
Con $k = 4$ y $p = 58$, se obtiene una exactitud de 0,83. Es decir, el 83 % de las imágenes del conjunto de prueba fueron clasificadas correctamente, la cual es una proporción bastante alta. Concluimos que el experimento fue satisfactorio.

4.3. Varianza en Función de Componentes Principales

Preprocesamos los datos de entrenamiento utilizando el método de la potencia para lograr hallar los autovalores de la matriz de covarianza. Nos interesa realizar un cambio de base de la matriz para obtener una nueva representación de la imagen, reducir la dimensionalidad del problema y eliminar redundancia (los atributos que están muy correlacionados pueden resultar redundantes). Queremos seleccionar la cantidad de componentes principales que produzca la máxima eficiencia. Con este objetivo, generamos un gráfico para evaluar la varianza explicada en función de la cantidad de componentes p:

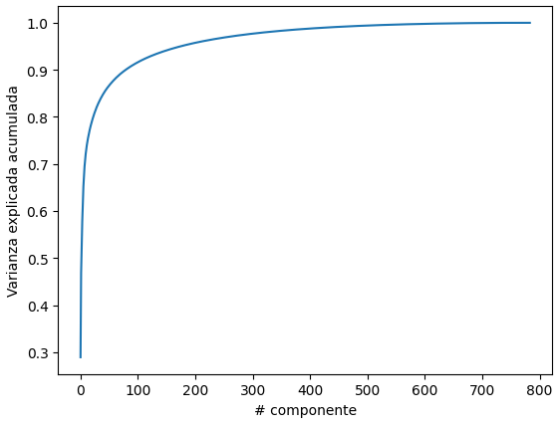

Figura 1: Visualización de la cantidad de varianza explicada en función de la cantidad de componentes p

Analizando el gráfico, podemos ver que para $p = 100$, la varianza acumulada es mayor al 90 % y para valores entre 100 y 784 se mantiene casi constante, por lo cual concluimos que no sería eficiente explorar valores mucho mayores a 100 para p y será suficiente limitarnos al rango $(1, ..., 100)$ durante la optimización de parámetros.

4.4. Estudio de Convergencia

En esta parte del informe estudiaremos la convergencia del método de la potencia. Para esto mediremos la cantidad de pasos que tarda en converger y una medida del error del método dada por $\|Av_i - \lambda_i v_i\|_2$ siendo v_i, λ_i la aproximación al autovector y autovalor i.
Para este estudio utilizamos una matriz con los autovalores $\{10,\ 10 - \epsilon,\ 5,\ 2,\ 1\}$ e hicimos varias mediciones con matrices de Householder aleatorias.
El método para generar matrices de Householder que utilizamos fue: primero creamos una matriz diagonal D la cual tiene en la diagonal los autovalores anteriormente mencionados. Segundo, creamos un vector aleatorio v normalizado. Tercero generamos una matriz de Householder B de la siguiente forma:

$$I - 2(vv^t)$$

Por último nuestra matriz a estudiar H va a estar dada por $B^t D B$

Para el primer experimento estudiamos el error e hicimos el promedio y el desvío estándar de 100 repeticiones para 10 epsilons diferentes y nos dimos cuenta que el único caso en el que el error variaba significativamente ante el cambio del epsilon era el autovector asociado al primer autovalor. Para calcular el desvío estándar utilizamos la función $np.std()$ la cual usa la siguiente expresión de desvío estándar:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Con N la cantidad de datos, x_i los valores y μ la media.

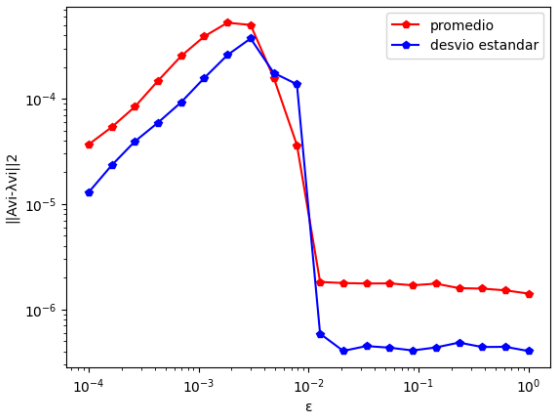

Figura 2: promedio de error sobre ϵ con desvío estándar

En este gráfico podemos ver que el error es alto al principio pero, a medida que se va notando más la diferencia entre el primer autovalor y el segundo ($\epsilon \approx 10^{-2}$), vemos que cae bastante rápido. Si entendemos cómo funciona el método de la potencia podemos ver que esto tiene sentido ya que: como sabemos que el x_0 se puede escribir como una combinación lineal de los autovectores de nuestra matriz de Householder y como en cada paso del algoritmo tenemos que:

$$x_{k+1} = \frac{Ax_k}{\|Ax_k\|}$$

Podemos ver que esta sucesión converge al autovector v_1 ya que es el autovector asociado al autovalor mas grande (λ_1). Pero podemos observar que si λ_1 y λ_2 son suficientemente cercanos entonces la sucesión no aproxima solo a v_1 sino a una mezcla de v_1 y v_2 lo cual resulta en un error en la estimación de v_1.

Para el segundo experimento estudiamos la cantidad de iteraciones utilizando el mismo metodo anterior.

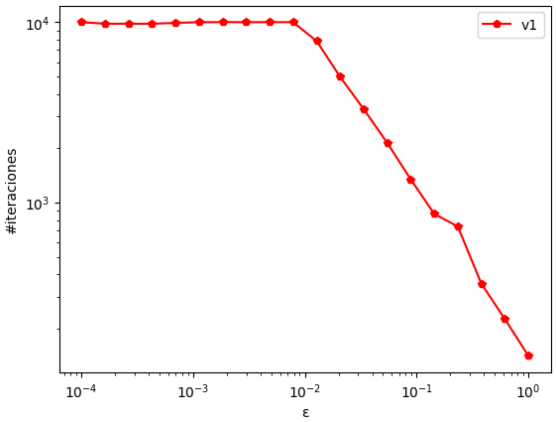

Figura 3: Promedio de iteraciones sobre ϵ

En este segundo gráfico podemos ver, al igual que el anterior, que la cantidad de iteraciones para convergencia son altas al principio pero, a medida que se va notando más la diferencia entre el primer autovalor y el segundo ($\epsilon \approx 10^{-2}$), vemos que decrece. Esto una vez mas es algo lógico ya que, por lo mencionado anteriormente, el método de potencia converge a una velocidad que está determinada por la relación $|\frac{\lambda_2}{\lambda_1}|$. O sea, cuanto más chica sea esta división, más rápido converge. Pero si λ_1 y λ_2 son cercanos esto es aproximadamente 1, lo que implica que la convergencia será lenta, o sea, muchas iteraciones.

5. Conclusión

Luego de finalizar el trabajo, llegamos a las siguientes conclusiones: Sobre el método de la potencia y la técnica de deflación, vimos que permiten descomponer una matriz (con ciertas características) en autovalores y autovectores. Este algoritmo es particularmente eficiente cuando se tienen autovalores no nulos y no repetidos. En caso contrario, puede traer problemas no solo al estimar autovectores sino que tambien a la velocidad de convergencia del método. Pudimos experimentar con el algoritmo de KNN y vimos que fijando una cantidad relativamente chica de vecinos considerados, podemos obtener una tasa de aciertos bastante alta, por lo cual podemos concluir que este algortimo genera buenos resultados y es una buena opción para utilizar en clasificación de datos.
Finalmente, experimentando con PCA pudimos ver que la dimensionalidad del problema puede reducirse en gran medida y aún así conseguir resultados satisfactorios. Aunque las imágenes estén compuestas por 784 atributos, vimos que podemos reducir ese número a 100 atributos o incluso menos y no perder información importante para una categorización correcta de las imágenes. De esta forma, eliminamos información redundante, evitamos cálculos innecesarios, obtenemos una mejor interpretación de los datos/datos de mejor calidad y logramos mejorar el rendimiento del modelo. En conclusión, mediante la implementación de PCA logramos simplificar el proceso de clasificación e incluso podemos obtener resultados maś acertados ya que se elimina en gran parte el ruido"de las imágenes.