Métodos Numéricos

2 de junio de 2024

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico 2

Primer cuatrimestre 2024

Grupo 19

Integrante	LU	Correo electrónico
Rios, Brian Ivan	917/19	ivan.rios2010@gmail.com
Saccomano, Ignacio Manuel	222/22	nachosacco1@gmail.com
Maldonado, Juan Bautista	164/22	jbmaldonado2003@gmail.com

Resumen

Se realizó la implementación de un clasificador de imágenes usando KNN con 5-fold cross-validation. Posteriormente se evaluó la implementación de PCA (Principal Component Analysis) para simplificar el modelo y se compararon los resultados óptimos mediante la exploración de hiperparámetros.

Palabras clave: KNN, PCA, 5-fold cross-validation, machine learning.

Índice

1. Introducción

Machine Learning (o en español, Aprendizaje Automático) es una rama de la Inteligencia Artificial que persigue el desarrollo de técnicas que permitan a las máquinas aprender de manera automática y mejorar a través de la experiencia, sin haber sido explícitamente programadas para ello. Existe una fase inicial de entrenamiento en la que se nutre al software de ejemplos, experiencia directa o instrucciones. Es una fase dirigida en la que es muy importante disponer de conjuntos de datos relevantes para cumplir con los objetivos. Después del entrenamiento, se evalúa el modelo utilizando un conjunto de datos de prueba o validación con el objetivo de ajustar hiperparámetros. El objetivo final es que el modelo generalice bien y pueda reconocer datos no vistos. Esto significa que debe funcionar bien en situaciones del mundo real, no solo en los datos de entrenamiento. A lo largo de este informe desarrollaremos estas etapas a través de la implementación de un reconocedor de imágenes.

La tarea de reconocimiento de imágenes involucra crear un clasificador, el cual toma un elemento del conjunto de datos y genera una predicción. Nuestro conjunto de datos sera Fashion Mnist. Esta es una gran base de datos de imágenes de moda. Para armar este clasificador vamos a utilizar la técnica de aprendizaje automático de k-vecinos mas próximos (en ingles, k-nearest neighbors, KNN). En su versión más simple, este algoritmo considera a cada objeto del conjunto como un punto en el espacio euclıdeo m-dimensional para luego, dado un nuevo objeto, asociarle la clase del o los puntos más cercanos del conjunto de datos.

A su vez nos vamos a ver interesados en una técnica llamada PCA. El análisis de componentes principales (PCA) busca encontrar un cambio de base de los vectores, de tal manera que las dimensiones se ordenen en componentes que explican los datos de mayor a menor. De forma análoga, el cambio de base busca ordenar las dimensiones según su varianza, de mayor a menor. Al reducir la dimensionalidad, se simplifica el modelo y se mejora la eficiencia computacional.

Los algoritmos/ técnicas tienen hiperparámetros. En KNN es k la cantidad de vecinos y en PCA es p la cantidad de componentes utilizadas. Encontrar los valores de estos parámetros que maximicen la efectividad de nuestro reconocedor sobre el conjunto de validación sera lo que busque garantizar el buen funcionamiento sobre el conjunto de prueba.

Hay que tomar ciertos recaudos a la hora de realizar una experimentación y explorar estos parámetros. En caso contrario se podría llegar a obtener datos mas optimistas de lo normal. Para atacar esto utilizaremos utilizaremos validación cruzada, mas específicamente k-fold cross-validation. El objetivo principal de k-fold cross-validation es estimar cómo se desempeñará un modelo en datos no vistos.

A continuación profundizaremos en las técnicas y metodología mencionadas anteriormente acompañando con la respectiva experimentación e implementación del reconocedor de imágenes. El objetivo final es desarrollar un modelo que sea preciso y generalice bien a datos nuevos.

2. Desarrollo

2.1. KNN

El algoritmo de KNN es un clasificador de aprendizaje supervisado sobre el cual basaremos el funcionamiento del reconocedor de imágenes. Se lo clasifica como supervisado porque se le proporcionan datos etiquetados. Intuitivamente, el algoritmo sitúa un conjunto de datos de entrenamiento sobre el espacio a partir de un criterio basado en sus características. El nuevo dato entrante a ser clasificado se lo sitúa en este espacio para posteriormente encontrar los k datos mas cercanos a él, es decir, los k vecinos mas cercanos. El resultado final dependerá de si la etiqueta dominante de los k vecinos coincide con la etiqueta del dato a ser clasificado. En caso afirmativo se lo considera un acierto. Notemos que se pueden definir 2 etapas dentro del algoritmo. Una de entrenamiento donde se "memorizan"los datos para luego una etapa de clasificación de un dato ajeno al conjunto. Idealmente, los datos forman regiones bien definidas en el espacio de acuerdo a su clasificación y bastaría comparar el dato a ser clasificado con unos pocos vecinos para concluir cuál es su etiqueta. El error es asumir que las regiones están bien delimitadas y que el dato desconocido se encuentra dentro de ellas. En realidad, para la mayoría de aplicaciones donde la delimitación de grupos en el espacio no es tan clara no es trivial determinar cuántos vectores evaluar como vecinos. Más adelante se desarrollará un potencial método para encontrar un buen valor para k sin asumir condiciones de los datos.

El criterio utilizado será la **distancia coseno centrada.** Esta se define para 2 vectores x e y como:

$$D(x, y) = 1 - \frac{(x - \mu_x) \cdot (y - \mu_y)}{\|x - \mu_x\|_2 \|y - \mu_y\|_2}$$

Donde μ es la media de los vectores. La similitud coseno mide la similitud direccional entre dos vectores en un espacio n-dimensional, es decir, la clasificación se dará según que tan parecidos son los vectores en orientación. La normalización de los vectores es necesaria para garantizar que los resultados no se vean afectados por la magnitud de los vectores.

La implementación va a consistir de calcular esta distancia. En un primer momento se define el conjunto de entrenamiento y de prueba. Para cada dato del conjunto de prueba se le calcula la distancia coseno centrada con cada dato de entrenamiento. Luego se ordenan las distancias obtenidas de menor a mayor y se calcula la moda de las etiquetas de los k vecinos mas cercanos. Por último, se calcula la exactitud del algoritmo midiendo la cantidad de aciertos en relación a la cantidad de datos evaluados.

Algorithm 1 KNN con distancia coseno

$X_e \leftarrow$ conjunto de datos de entrenamiento
$X_p \leftarrow$ conjunto de datos de prueba
$res \leftarrow [\ ]$
$D \leftarrow [\ [\ ]\ ]$ ▷ En la matriz D se almacenarán las distancias
$d_{i,j} \leftarrow 0,\ i = 1, 2, ..., \text{length}(X_p),\ j = 1, 2, ..., \text{length}(X_e)$
for $i \leftarrow 1$ hasta $\text{length}(X_p)$ **do**
    for $j \leftarrow 1$ hasta $\text{length}(X_e)$ **do**
        $d_{i,j} \leftarrow \text{D}(x_e^i, x_p^j)$
    end for
end for
for $i \leftarrow 1$ hasta $\text{length}(X_p)$ **do**
    $d_i \leftarrow \text{sort}(d_i)$ ▷ Ordenar las filas
end for
for $i \leftarrow 1$ hasta $\text{length}(X_p)$ **do**
    $res_i = \text{moda}(d_i, k, etiquetas)$ ▷ En res_i se aloja la etiqueta dominante entre las primeras k de d_i
end for
Devolver res

La complejidad computacional del algoritmo depende mayoritariamente de el tamaño de los conjuntos de datos y de los datos en sí. Estamos hablando de una complejidad $\mathcal{O}(\text{length}(X_e) * \text{length}(X_p) + \text{length}(X_p) * \Phi)$ donde Φ es la complejidad de ordenar los datos de d. Teniendo en cuenta que el tamaño de los conjuntos de datos suele ser masivo y el tamaño de los datos aún más (por ejemplo, una imagen Full HD esta compuesta por 2.073.600 píxeles) resulta necesario disminuir estos costes. Para esto veamos

que el producto escalar de los vectores con media cero puede hacerse de forma matricial. Poder expresar este cálculo como una multiplicación de matrices permite reducir tiempos de cómputo en la práctica.

Se define el cálculo de la distancia coseno entre 2 conjuntos de vectores en forma matricial de la siguiente manera. Dada una matriz A tal que cada filas corresponde a un vector normalizado de uno de los conjuntos, dada una matriz B tal que cada fila corresponde a un vector normalizado del otro conjunto y siendo C una matriz llena de unos de tamaño apropiado, se entiende el cálculo como:

$$DMatricial(A, B) = C - A \cdot B^t$$

Con esta fórmula se puede re-definir el algoritmo de KNN entre 2 conjuntos como:

Algorithm 2 KNN con distancia coseno Matricial

$X_e \leftarrow$ conjunto de datos de entrenamiento
$X_p \leftarrow$ conjunto de datos de prueba
$X_e \leftarrow \text{normalize}(X_e)$ ▷ normalize() normaliza las filas, es decir, los datos
$X_p \leftarrow \text{normalize}(X_p)$
$res \leftarrow [\,]$
$D \leftarrow [\,[\,]\,]$ ▷ En la matriz D se almacenarán las distancias
$C \leftarrow [\,[\,]\,]$
$c_{i,j} \leftarrow 1,\ i = 1, 2, ..., \text{length}(X_p),\ j = 1, 2, ..., \text{length}(X_e)$
$D \leftarrow DMatricial(X_p, X_e)$
$D \leftarrow D^t$
for $i \leftarrow 1$ hasta $\text{length}(X_p)$ **do**
    $d_i \leftarrow \text{sort}(d_i)$ ▷ Ordenar las filas
end for
for $i \leftarrow 1$ hasta $\text{length}(X_p)$ **do**
    $res_i = \text{moda}(d_i, k, etiquetas)$ ▷ En res_i se aloja la etiqueta dominante entre las primeras k de d_i
end for
Devolver res

Aunque a nivel practico esta implementación ahorra tiempo de computo, la complejidad teórica sigue siendo la misma. Si se busca utilizar KNN en un modelo de computo actual será prioridad reducir la complejidad, o lo que es lo mismo, reducir el tamaño de los datos. Esto mismo se vera a continuación.

2.2. PCA

El Análisis de Componentes Principales (PCA) es una técnica de reducción de dimensionalidad, en este caso de la dimensión de los datos, tal que los componentes se ordenan por la cantidad de varianza original que describen. En muchos conjuntos de datos, las características están altamente correlacionadas entre sí. Esto significa que hay redundancia en la información. Intuitivamente se busca reducir la información a aquellas características que tienen mayor varianza, las que se pueden interpretar como características que hacen únicos y distintos a los datos.

Esta técnica también posee un hiperparámetro a estudiar. Este es p, el cuál indica la cantidad de dimensiones finales de la reducción y varia dependiendo el caso. El valor adecuado de p para el reconocedor de imágenes se analizará más adelante a la par que el valor adecuado del hiperparámetro k.

Para llevar a cabo este cambio de base se utiliza la matriz de covarianza. Esta se define para una matriz X donde cada columna es un dato/variable normalizado como:

$$C_x = \frac{X^t \cdot X}{\text{tam}(X) - 1}$$

Esta matriz tiene la varianza de cada variable en la diagonal, y la covarianza en las restantes posiciones. La matriz de covarianza captura las relaciones lineales entre las características originales de los datos. Se puede entender como que esta nueva matriz no contiene a los datos sino a la relación que hay entre ellos, mas específicamente que tan distintos son.

El cambio de dimensión se lleva a cambo diagonalizando esta matriz, mas específicamente obteniendo los autovectores. Se podría decir que los autovectores y autovalores son la "esencia" de la matriz en el sentido de que capturan sus propiedades fundamentales y permiten una comprensión más profunda de su estructura. Al calcular los autovectores de la matriz de covarianza estos indican las direcciones de

máxima varianza en los datos, y como se explico anteriormente estas direcciones son las mas importantes, las que caracterizan mejor a los datos. Siendo C_x la matriz de covarianza de X y V la matriz que posee en sus columnas los autovectores de C_x, entonces la transformación de los datos se da por:

$$Z = X \cdot V$$

Donde Z contiene los "nuevos"datos. Con este en mente la técnica de PCA se reduce a calcular C_x y V, siendo el calculo de V el subproblema no trivial.

V contiene los autovectores de C_x en sus columnas. Calcular autovectores de una matriz no es tarea fácil. El costo de calcularlos de la manera tradicional es de $\mathcal{O}(n^3)$ [2]. Sin embargo notar que la matriz en cuestión es simétrica semidefinida positiva, lo que habilita a usar el método de la potencia.

Este tiene como requisito que la matriz sea cuadrada, lo cual se cumple puesto que es simétrica, y que el autovalor dominante sea distinto del resto. Afortunadamente, este ultimo requisito no es necesario cuando uno implementa PCA. La idea detrás de este método es hacer converger un vector cualquiera a el autovector asociado al autovalor dominante en módulo aplicando la transformación lineal asociada a la matriz, es decir, multiplicando la matriz y el vector. Esto se realiza muchísimas veces para que en cada iteración el vector "absorba la esencia" de la matriz. De esta manera se obtiene el autovector asociado el autovalor dominante. Para obtener el autovalor basta con calcular el Cociente de Rayleigh. Algorítmicamente, dada una matriz cuadrada con autovalor dominante distinto en modulo del resto B, un vector inicial v distinto de 0 y un limite de iteraciones tenemos:

Algorithm 3 Método de la Potencia

1: **Entrada:** B (matriz), x_0 (vector inicial), niter (número de iteraciones)
2: $v \leftarrow x_0$
3: **for** $i = 1, \ldots,$ niter **do**
4: $\quad v \leftarrow \frac{Bv}{\|Bv\|}$
5: **end for**
6: $\lambda \leftarrow \frac{v^T B v}{v^T v}$
7: **Devolver** λ, v

En la teoría este algoritmo con $niter = \infty$ devuelve exactamente el autovalor y autovector en cuestión. En la practica este debe ser finito, por lo que mientras más grande sea el limite de iteraciones mejor será la aproximación. Se debe elegir un valor lo suficientemente grande para que la respuesta sea adecuada a la vez que lo suficientemente chico para que el tiempo de cómputo no sea demasiado.

Para simplificar este aspecto se puede definir un criterio de parada. Este se debe encargar de interrumpir el algoritmo y notificar que no es necesario seguir con el proceso de convergencia porque se considera que se llegó a una solución válida de acuerdo a un criterio que se pase como parámetro. En el presente trabajo evaluamos la norma infinito de la diferencia entre el vector obtenido en la iteración anterior y la actual. De esta manera se consigue comparar que tan parecidos son los vectores y en el caso que el resultado de calcular la norma sea ϵ (con ϵ cercano a 0) se puede concluir que el método convergió por lo que no es necesario continuar. La utilización de un máximo de iteraciones lo suficientemente alto y un criterio de parada como el plantado resulta lo adecuado para implementar el método de la potencia lo mejor posible.

Para obtener los autovalores y autovectores restantes construiremos una matriz a partir de la original cuyos autovectores serán los mismos y autovalores también a excepción del ya calculado, que se sustituirá por 0. A esta técnica se le conoce como deflación, en particular se utilizará la deflación de *Hotelling*. Sea B una matriz cuadrada tal que sus autovalores son distintos entre sí y tiene una base ortonormal de autovectores (la matriz de covarianza lo cumple debida a que es semi definida positiva [1]), entonces $(B - \lambda_1 v_1 v_1^t)$ cumple lo mencionado anteriormente. Ver que:

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

Volver a aplicar el método de la potencia sobre esta nueva matriz dará como resultado el segundo autovalor dominante y su respectivo autovector. Este procedimiento de puede realizar para encontrar todo

los autovalores y autovectores de una matriz, siempre y cuando se cumplan los requisitos. El algoritmo del método de la potencia + deflación se define como:

Algorithm 4 Método de la Potencia + Deflación de Hotelling

Entrada: B (matriz), x_0 (vector inicial), niter (número de iteraciones)
$avalores \leftarrow []$
$avectores \leftarrow []$
for $i = 1, \ldots, \text{length}(B)$ **do**
    $\lambda, v \leftarrow \text{MétodoDeLaPotencia}(B, x_0, niter)$
    $avalores \leftarrow avalores + \lambda$
    $avectores \leftarrow avectores + v$
    $B \leftarrow B - \lambda * v * v^t$
end for
Devolver $avalores, avectores$

De esta manera se tienen 2 técnicas que permitirán construir a V a partir de la matriz C_x y llevar adelante el algoritmo de PCA. Este combinado con KNN serán los engranajes principales del reconocedor de imágenes. Sin embargo, durante el desarrollo de ambas técnicas se omitió el peso de elegir los hiperparámetros adecuados.

2.3. Validación cruzada

Los hiperparámetros influyen directamente en el rendimiento del modelo. Un ajuste adecuado según las características de los datos puede mejorar significativamente la precisión y generalización. Poder experimentar para conseguir los valores adecuados de k (cantidad de vecinos cercanos) y p (cantidad de componentes principales) es una tarea extremadamente importante, tanto que es necesario tomar ciertos recaudos y emplear una técnica de validación cruzada. Supongamos que para elegir un valor de k adecuado se experimenta de la siguiente manera: Se dividen los datos en un conjunto de entrenamiento y otro de prueba, y se ajustan los hiperparámetros para maximizar el rendimiento en el conjunto de prueba, podemos obtener resultados engañosos. Esto porque se usan el conjunto de prueba para elegir los hiperparámetros, lo cual introduce un sesgo. Como resultado, la configuración seleccionada puede parecer muy buena para los datos de prueba, pero no necesariamente generalizará bien a datos nuevos no vistos, es decir, cuando el modelo sea "expuesto al mundo real" puede no funcionar como se esperaba. Se estaría entrenando al modelo con datos sobre los cuales se va a evaluar. Esto puede llevarnos a creer que nuestro modelo es mejor de lo que realmente es.

Para atacar este problema se cuenta con la técnica de verificación cruzada k-fold cross-validation. La verificación cruzada implica dividir los datos en múltiples subconjuntos y luego realizar múltiples rondas de entrenamiento y evaluación. Estos subconjuntos provienen del conjunto de entrenamiento pero serán utilizados para entrenar al modelo.

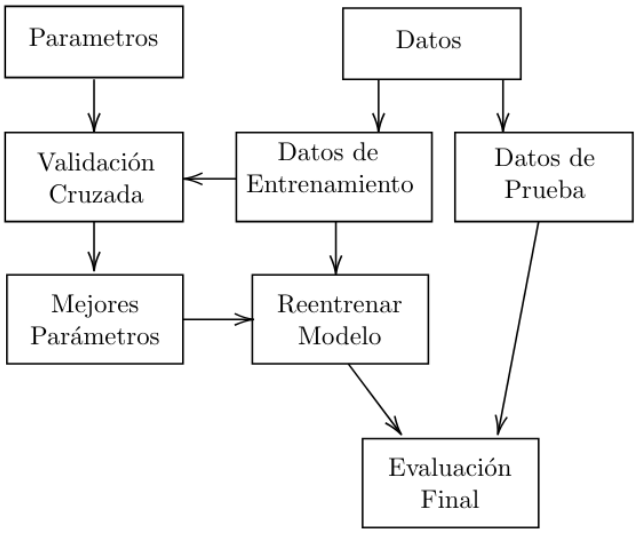


Figura 1: Flujo de desarrollo.

El flujo de desarrollo llevado adelante es el de la **figura 2.3**. En el inicio de la experimentación se cuenta con el conjunto de hiperparámetros a optimizar y el conjunto de datos sobre el cual entrena y se pone a prueba el proceso. Este segundo no se utiliza durante la fase de entrenamiento, en caso contrario podría ocurrir lo comentado anteriormente.

Para la optimización entra en juego k-fold cross-validation. El conjunto de entrenamiento se subdivide en 2: un conjunto de datos llamado de desarrollo y otro conjunto llamado nuevo entrenamiento. El primer conjunto es de menor tamaño que el segundo y ambos contienen una cantidad similar de elementos de un mismo tipo. Esto para asegurar que el modelo se entrena para todos los tipos de datos por igual. Este mismo proceso se realiza k cantidad de veces (no confundir con el hiperparámetro de KNN) tomando distintas rebanadas del conjunto de entrenamiento. Idealmente todo dato de alguna repetición fue parte del conjunto de desarrollo y en las otras repeticiones fue parte del conjunto de nuevo entrenamiento.

Todo este proceso se lleva adelante analizando que tan exitosa fue la combinación de hiperparámetros elegida, por lo que para poder mejora la elección de estos es necesario analizar una cantidad considerable de combinaciones. Una vez elegida queda el paso final: poner a prueba el modelo entrenando sobre todos los datos de entrenamiento y evaluando sobre los datos de prueba, aquellos que nunca se utilizaron.

3. Experimentación

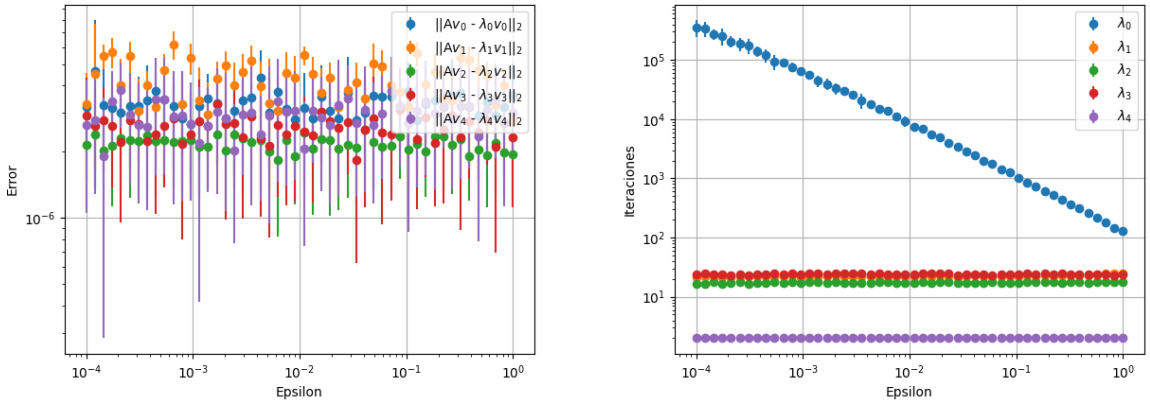
Los siguientes experimentos se llevaron a cabo en los archivos adjuntos. Entre ellos se cuenta con una implementación en C++ del método de la potencia sumado al algoritmo de deflación que se ejemplificó. Esta hace uso de la biblioteca Eigen, la cual es un referente de operaciones de álgebra lineal, matrices y vectores. También se implementó el algoritmo de KNN pero esta vez en Python utilizando la conocida biblioteca Numpy. Los gráficos y el flujo se trabajo también se llevaron a cabo desde Python llamando a funciones de C++ a través de la biblioteca Ctypes. Esta permite la interfaz entre ambos lenguajes de programación.

Como se ha mencionado la experimentación divide en 2 fases: una primera de optimización de hiperparámetros y una segunda etapa de evaluación del modelo.

3.1. Método de la potencia

Para estudiar la convergencia del método de la potencia se medirá la cantidad de pasos que tarda en converger y una medida del error del método dada por $||A_{v_i} - \lambda_i v_i||$ siendo v_i , λ_i la aproximación al autovector y autovalor i-ésimo. Para simplificar se toma una matriz con los siguientes autovalores $10, 10 - \epsilon, 5, 2, 1$ y se harán varias mediciones con matrices de *Householder* aleatorias. Esto último se usa para crear una matriz a partir de su diagonalización, ya que los autovalores y los autovectores son

conocidos y podemos compararlos con el resultado del algoritmo. Se procede a realizar un gráfico del error y otro para las iteraciones que tomó el algoritmo en converger, ambos en función de épsilon (tomamos 50 valores desde 10^{-4} hasta 10^{0}).



(a) **(b)**

Figura 2: En ambos gráficos los ejes están en escala logarítmica. El máximo de iteraciones tolerado fue de 500,000 y el criterio de parada fue $\delta = 10^{-7}$. Los puntos en el gráfico representan el promedio de 20 mediciones con vectores aleatorios para la matriz de Householder y las barras representan la varianza de los resultados.

En la figura **2b**, se puede observar cómo la cantidad de iteraciones que toma la convergencia de los autovalores se mantiene constante a excepción del más grande. Cuando ϵ es muy pequeño, el autovalor $10 - \epsilon$ es extremadamente cercano al autovalor dominante que es la constante 10. Por lo tanto, sabiendo que la **velocidad de convergencia** depende estrictamente de qué tan distantes son los autovalores dominantes [2] es esperado que el método tarde demasiado en converger, incluso tardando más que el tope establecido de iteraciones.

Sin embargo, independientemente de los valores de ϵ el error expresado en la **figura 2a** se mantiene relativamente constante para todos los autovalores (a pesar de las fluctuaciones producto de la medición aleatoria). Esto sucede porque, al haber establecido un tope de iteraciones considerablemente elevado, el método encontró los autovalores y autovectores adecuados o tolerablemente cercano a los valores reales. Es decir, sin importar lo que tarde el método, si se le da una cantidad de iteraciones suficientes converge a los valores correctos (o muy cercanos a los reales) sin importar qué tanta diferencia haya entre el autovalor dominante y el segundo (siempre y cuando no sean exactamente iguales). Lo único que puede variar es la velocidad de convergencia, que como fue probado teoricamente y verificado en la experimentación expresada en la figura 2b, sí que depende fuertemente de la diferencia en módulo del autovalor dominante y el segundo más grande.

3.2. Optimización de hiperparámetros

Inicialmente, se desconoce el valor del hiperparámetro k para el algoritmo KNN que sea efectivo en términos de aciertos sobre datos desconocidos a partir de un conjunto de ajuste dado. Por ejemplo, si probamos con $k = 5$ usando los datos de ajuste y probamos sobre el conjunto de testing obtenemos un acierto del 85 %. Sin embargo, al no haber hecho ningún análisis sobre el hiperparámetro y haberlo probado directamente en el conjunto de testing este porcentaje puede no solo no ser el óptimo alcanzable sino que además puede performar con una eficiencia mucho menor a 85 % con otros datos. Para realizar una exploración de hiperparámetros eficiente y más rigurosa, como fue mencionado, se emplea la técnica de 5-fold cross-validation.

3.3. 5-fold cross-validation

Para mejorar la rigurosidad del proceso de exploración de hiperparámetros se presenta la experimentación de 5-fold cross-validation. La etapa de cross-validation consiste en separar los datos de ajuste en una

parte de desarrollo y otra de entrenamiento. Lo que haremos será, para distintos valores de k, ejecutar el algoritmo de KNN con la partición de entrenamiento y trataremos de predecir las etiquetas de los datos de desarrollo. Este procedimiento lo haremos con 5 particiones (o folds) distintas y consideraremos el promedio de efectividad de los folds como la efectividad dada para el valor particular del hiperparámetro probado.

A continuación se presenta el experimento mencionado para 300 valores de k:

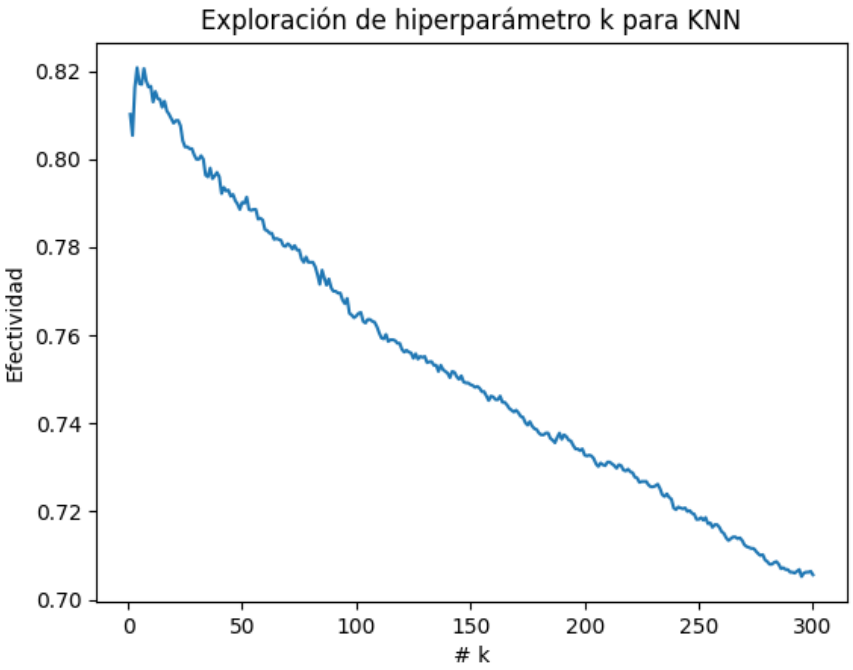


Figura 3: Exactitud del modelo en función de parámetro k

Como se puede ver en la **figura 3**, a excepción de valores de $k \leq 50$ la efectividad decrece linealmente. Esto sucede porque al comparar el dato desconocido con tantos vecinos puede ocurrir que haya un subgrupo lejano de datos que sea mayor en cantidad en comparación a los que de verdad asemejan el dato desconocido. Entonces, al ver la moda se descarta la etiqueta correcta por tener una cantidad menor de ocurrencias a pesar de que sean más similares. Sin embargo, lo opuesto también ocurre como se ve al comienzo de la curva. En el caso donde el dato desconocido cae 'en el borde' del subgrupo que compone la etiqueta correspondiente a este, puede ocurrir que los datos de entrenamiento más cercanos sean los bordes de otro grupo. De esta manera no se le da información suficiente al algoritmo y concluye erróneamente la etiqueta desconocida. En síntesis, se evidenciaron 2 problemas clásicos de machine learning: la sobreinformación y la falta de ella.

3.4. PCA

Para el apartado de PCA primero verificamos qué tan relevantes son las componentes principales. Recordemos que la matriz de covarianza de $\mathbf{X}$ se diagonaliza mediante el método de la potencia con deflación de Hotelling [1].

$$C_x = VDV^T \tag{1}$$

Por lo tanto, al obtener la diagonalizacion de C_x se obtiene una factorizacion donde los elementos de la diagonal de D son las covarianzas de la componente principal *iesima* de X consigo misma (pues son los autovalores de X^TX). Esto denota qué tanto representa a los datos de X la componente principal compuesta por el vector *iesimo* de V. Al ser calculados mediante el método de la potencia, los autovalores de mayor módulo están en las primeras componentes diagonales de D y los autovectores asociados están en la misma columna que D en V. De este modo, viendo las varianzas que se acumulan al ir incrementando

la cantidad de componentes principales se puede observar qué tanto aporta cada una para representar a X en menos dimensiones. Esto se puede observar mejor si se divide a cada D_{ii} (i.e: autovalor de Cx y covarianza de los atributos de X consigo mismos) por la suma de todos los D_{ii}. Esto hará que la suma de todos valga 1, es decir, que entre todos acumulan toda la varianza de los datos.

En resumen, al reducir dimensiones mediante PCA queremos observar cuán representativos son los autovectores de V que junto con los otros calculados generan a todas las muestras de X. Esta información la proporcionan las varianzas de la columna *iesima* de D. Si se divide a estos coeficientes por la suma de toda la diagonal se obtiene un porcentaje que indica qué tan relevante es para componer a cualquier X. Si se usan 784 componentes, el resultado es 1. Esto quiere decir que se puede construir a cualquier vector de X con exactitud lo que tiene sentido, pues los vectores de V son ortogonales por lo que forman una base y fueron construidos en base a los vectores que componen X.

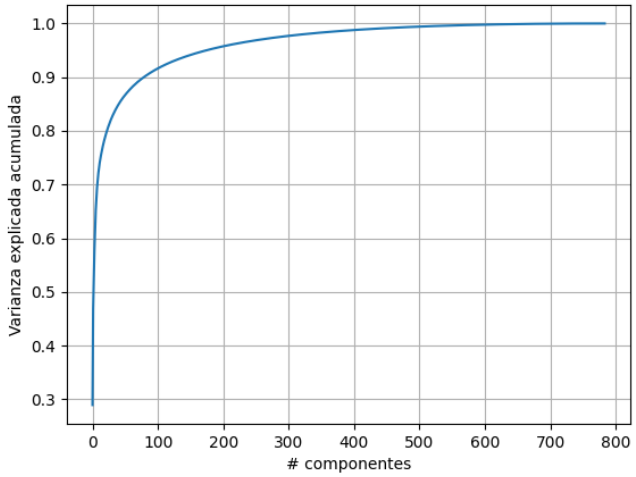
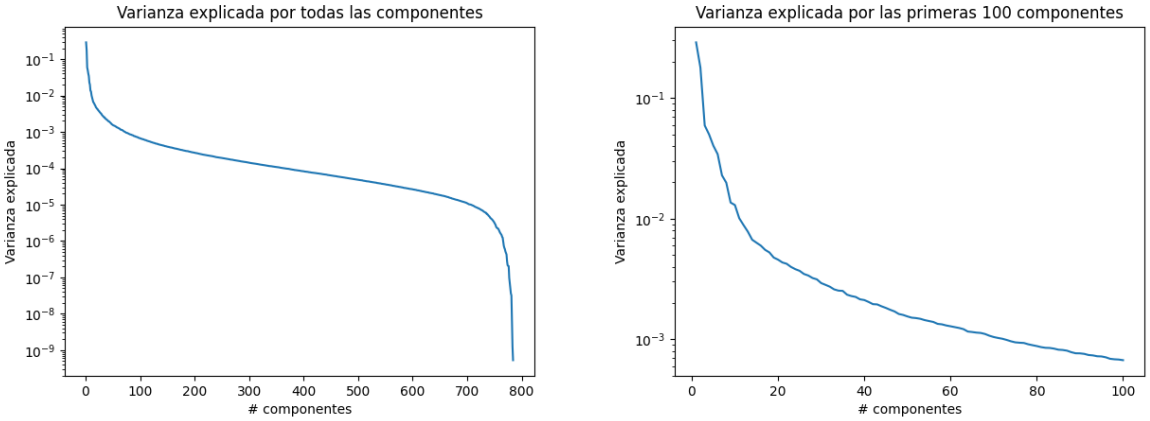


Figura 4: Representatividad de las primeras p componentes principales acumuladas, ambos ejes en escala logarítmica.

Como se puede observar en la **figura 4**, la varianza crece de forma logarítmica porque las primeras componentes explican mucha varianza y a medida que se van agregando más se observa un rendimiento decreciente. Es por ello que esta técnica es tan útil. En la mayoría de aplicaciones, y particularmente con el reconocimiento de imagenes, los datos contienen atributos redundantes. En el caso del trabajo presente , por ejemplo, al estar las imágenes centradas los bordes siempre van a ser oscuros. Por lo tanto, es intuitiva la idea de transformar los datos de modo tal que la información se mantenga pero eliminando atributos redundantes reduciendo así la complejidad del problema. Para evidenciar aún más la cantidad de componentes redundantes a continuación enseñaremos las varianzas explicadas por cada componente por separada.



(a) **(b)**

Figura 5: En la figura 5a las ordenadas están representadas en escala logarítmica.

Como se puede observar en la figura 5a, la varianza decrece a medida que aumenta p y cae abruptamente para los valores superiores a 700. Haciendo un acercamiento todavía mayor sobre las primeras 100 componentes se evidencia cómo las primeras abarcan mayor varianza y como disminuye rápidamente a medida que se van agregando más componentes.

En conclusión, si se quitan dimensiones al conjunto de datos original (i.e: Se hace el cambio de base $\hat{X} = XV$) se pierde exactitud, pero analizando las varianzas acumuladas se puede concluir que eligiendo una cantidad de componentes principales mucho menor que las que componen a los datos de X se puede representar con un porcentaje alto de exactitud debido a la varianza contemplada. Por ejemplo, en la figura 4 se observa que disminuyendo la imagen de 784 píxeles a solo 100 se acumula un 90 % varianza. Esto quiere decir que al calcular $\hat{X} = XV$ podemos reconstruir los datos originales con un 90 % de exactitud pero reducimos las componentes de X a solo 100. Por lo tanto, dado que reduce los tiempos de cálculo en gran medida y representa con bastante exactitud a los datos de entrada, a continuación aplicaremos KNN con 5-fold cross-validation para ver si es posible aprovechar esta reducción de dimensionalidad sin perder exactitud.

3.5. Optimización de hiperparámetros con PCA

A continuación, exploraremos cual es el mejor k de una forma similar a lo que fue realizado previamente pero con la salvedad de que esta vez exploraremos con menos valores de k por el fenómeno observado en la figura 3. Además, empleando la técnica de PCA recientemente explorada introduciremos un nuevo hiperparámetro p que permita reducir a X en p componentes con la técnica previamnte discutida. Con esto buscaremos no solo optimizar la cantidad de vecinos sino también ver si es posible obtener una tasa de aciertos máxima con menos componentes que las que contienen los datos. Una vez encontrados los hiperparámetros óptimos haremos la prueba final sobre los datos de testing.

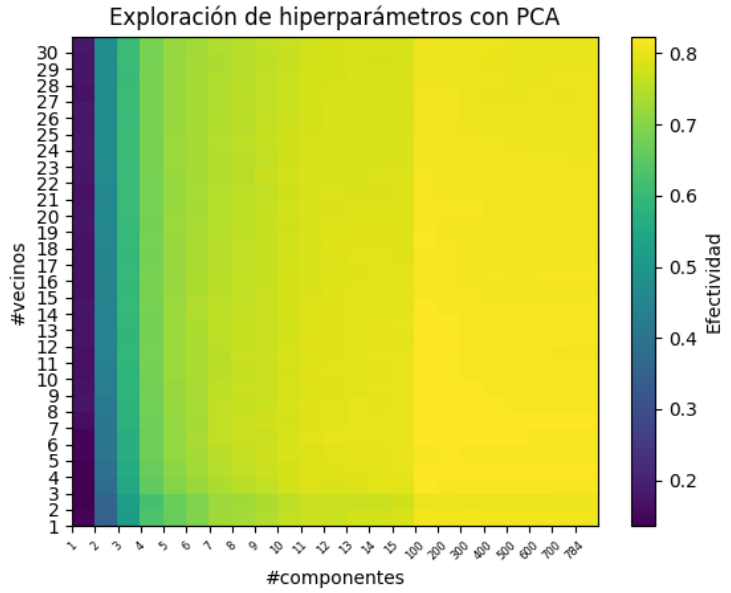


Figura 6: El calor representa la efectividad del modelo. Para el rango de (k, p) elegido. En un principio se analizan los primeros 15 valores de p y luego se aumenta de a 100 para mostrar el retorno decremental de exactitud de las últimas componentes.

El mapa de calor de la figura 6 expresa la relación entre los hiperparámetros y se observa cómo el porcentaje de efectividad para cada valor de k aumenta logarítmicamente a medida que incrementa el valor de p. Esto es congruente con los resultados previamente expuestos, pues la varianza acumulada por las primeras componentes es mucho mayor que la acumulada por las últimas. Por lo tanto, con un número más bajo de componentes se puede aproximar muy bien el porcentaje de exactitud total (i.e: El alcanzado por todas las 764 componentes) porque la matriz reducida logra preservar gran parte de la información. De hecho como se explicó previamente con tan solo 15 componentes se explica un 80 % de varianza y como se puede ver la diferencia en exactitud de usar 15 componentes no cambia demasiado con respecto a ir agregando de a 100 más. De esta manera se puede concluir que la reducción de dimensionalidad no solo agiliza la solución al problema sino que además tiene un costo despreciable en términos de exactitud perdida, pues como fue analizado previamente las últimas centenas no difieren demasiado en términos de varianza acumulada con respecto a la primera.

Luego de la expermentación concluimos que los hiperparámetros óptimos (notados como $k*$ para la cantidad de vecinos y $p*$ para la cantidad de componentes principales) fueron $k* = 10$ y $p* = 100$. Notar cómo a pesar de que la eficiencia de las últimas componentes es alta, con tomar en cuenta las primeras 100 se logra el mismo resultado. Esto tiene sentido pues la varianza acumulada es más de 90 %

Con $k*$ y $p*$ encontrados, ahora ejecutamos el algoritmo de KNN con el conjunto de ajuste y de test y obtenemos una tasa de aciertos final de 85 %. Este resultado es igual al que obtuvimos previamente probando con un k al azar pero logramos llegar al mismo resultado con menos atributos de los datos, lo que quiere decir que no sacrificamos eficiencia por reducción en complejidad.

4. Conclusiones

Se propuso el algoritmo de KNN para el problema de clasificación de imágenes. Para mayor rigurosidad en el ajuste de hiper parámetros aplicamos 5-fold cross-validation en el subconjunto de datos de entrenamiento.

Adicionalmente, se propuso la técnica de PCA para reducir el tamaño de los datos significativamente y así agilizar el algoritmo de clasificación de imágenes, demostrando que no hay una pérdida en exactitud para el caso presente.

En vista de los resultados presentados, quedo evidenciado como para algunas aplicaciones donde existe mucha información redundante (en este caso los bordes de las imágenes por ejemplo) el uso de PCA mejora significativamente la performance del modelo en términos de tiempo de acuerdo a la cantidad de componentes principales que se elija. Además, vimos que para la solución óptima no hubo una pérdida de performance lo que quiere decir que el ahorro en dimensiones de los datos no tiene por qué traducirse en una pérdida de generalidad del modelo.

Sin embargo, para los casos donde no sea clara la presencia de atributos redundantes en los datos la implementación de PCA puede no valer la pena. Esto porque el costo de diagonalizar la matriz de covarianza es elevado y además, en caso de que no haya atributos redundantes, la experimentación concluya en un óptimo de p muy cercano a la cantidad de atributos original. Por lo tanto, lo más probable es que no solo no se ahorre mucho sino que además se pierde tiempo de cómputo en la conversión $\hat{X} = XV$ y en el cálculo de la diagonalización.

5. Bibliografía

Referencias

[1] Diapositivas del laboratorio

[2] Clases teóricas

6. Apéndice

Para el desarrollo de los experimentos se utilizó el lenguaje **Python** junto con las bibliotecas de **Numpy** para los cálculos matriciales y **Pyplot** para los gráficos.

Adicionalmente, empleamos la biblioteca **Eigen** de C++ para agilizar el algoritmo del método de la potencia.