Trabajo Práctico 2
Reconocimiento de Dígitos (OCR)

Métodos Numéricos

Integrante	LU	Correo Electrónico
Teo Kohan	385/20	teo.kohan@gmail.com
Martin Santesteban	397/20	martin.p.santesteban@gmail.com

Abstract

En este informe se trata el problema del reconocimiento de dígitos numéricos arabigos. Para ello se entrenó un modelo utilizando la base de datos `MNIST`. Para clasificar las nuevas muestras se utilizó el algoritmo *kNN*. Para reducir el problema se aplicó PCA extrayendo los componentes principales o autodígitos. Adicionalmente, se estudiaron distintas formas de medir la efectividad del modelo para diversos parátros de *kNN* y PCA. Se utilizo la técnica de *k-fold cross-validation* para particionar los datos etiquetados provistos y reducir el sesgo y la varianza. Como resultado se que los algoritmos de *kNN* y PCA tienen valores predilectos independientemente de el número de muestras.

Keywords— Métodos numéricos, OCR, PCA, Eigenspaces, *kNN*, *K-fold*.

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
https://exactas.uba.ar

Índice

1 Introducción

1.1 Motivación

El reconocimiento de caracteres es un problema clásico en el campo de reconocimiento de patrones. Puntualmente el reconocimiento de dígitos, un subconjunto del problema del reconocimiento de caracteres tiene aplicaciones como la recopilación de datos para uso estadístico, reconocimiento automático en aplicaciones de *big-data*.

1.2 Datos

Se utilizó la versión de la base de datos de `MNIST`[2] ubicada en Kaggle[1], esta contiene 70.000 imágenes de los dígitos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} escritos en blanco sobre negro con una resolución de 28×28 representada en escala de grises de 8 bits de profundidad como se ve en la Figura 1. De estas, 42.000 están etiquetadas y serán utilizadas para el entrenamiento del modelo.

Figura 1: *Muestras tomadas de la base de datos `MNIST`.*

1.3 Objetivo

Se desea implementar una herramienta de reconocimiento de dígitos (OCR) utilizando PCA (Análisis de Componentes Principales) y *kNN* (K Vecinos más Cercanos). El siguiente objetivo es experimentar con los parámetros asociados a ambos métodos, midiendo tanto tasa de aciertos, Precission y Recall como tiempos de cómputo, para poder concluir con la mejor configuración que maximice la tasa de aciertos. También se experimentará con el método de la potencia para determinar cuantas iteraciones son necesarias para poder alcanzar buenas aproximaciones de autovalores. Por último, también se desea determinar cuando es necesario usar PCA y cuando no.

1.4 Desarrollo del modelo

Para la construcción de un modelo capaz de clasificar una nueva muestra $x \in \mathbb{R}^{28\times28}$, primero se lo debe entrenar, proporcionándole un conjunto base de muestras para su entrenamiento. A partir de este conjunto, el modelo contará con la información necesaria para poder hacer la predicción lo mas adecuada posible. Dado un conjunto de n muestras $m_i \in \mathbb{R}^p$ (donde en nuestro caso, p equivale a 28×28), el objetivo es poder ejecutar *kNN* (1.4.1) sobre el mismo cada vez que se quiera clasificar a una muestra nueva.

1.4.1 K Vecinos más Cercanos

K Vecinos más Cercanos o *kNN* de ahora en más, es una técnica para clasificar muestras. Dado un conjunto de muestras $x^{(i)} \in \mathbb{R}^n$ etiquetadas y una muestra entrante $x^\star \in \mathbb{R}^n$ sin etiquetar se le asigna a $x^\star$ la etiqueta correspondiente a la moda de sus k vecinos más cercanos como muestra la Figura 2.

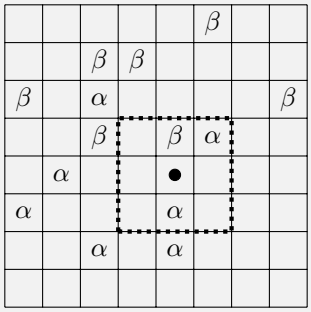


Figura 2: *kNN con $k = 3$ sobre vectores en $\mathbb{N}^2$. El punto marcado, utilizando $\|\cdot\|_1$ encuentra sus 3 vecinos más cercanos en la caja punteada. La moda de sus vecinos es α por lo tanto al vector le corresponderá la etiqueta α.*

Sin embargo, esto trae bastantes problemas. El primero es que al estar computando muestras p-dimensionales, donde en nuestro caso p es un número muy elevado, ejecutar *kNN* puede requerir mucho tiempo de ejecución. Segundo, al trabajar en dimensiones tan elevadas, aparece la maldición de la dimensión: fenómenos que surgen al analizar datos en espacios con demasiadas dimensiones. Como consecuencia, nos interesa hacer una transformación de los datos que de tal forma que al aplicarla a una muestra su imagen esté en un espacio con menos dimensiones. Además de bajar la dimensión, la transformación debe ocuparse de que mantener la mayor cantidad de información posible. Para esto, se utiliza el análisis de componentes principales, o PCA (1.4.2).

1.4.2 Análisis de Componentes Principales

Análisis de Componentes Principales o PCA , es una técnica para la reducción de la dimensión. Dado un conjunto de muestras $x^{(i)} \in \mathbb{R}^n$ expresamos estas muestras como una combinación lineal de los vectores canónicos

$$x^{(i)} = \sum_{j=1}^{n} \alpha_j^{(i)} e_j.$$

PCA consisten en encontrar una nueva base, V, tal que la varianza de sus vectores expresados en esta nueva base este ordenada de mayor a menor según sus coordenadas como muestra la Figura 3. Luego se puede reducir la dimensionalidad de las muestras descartando los componentes con menor varianza.

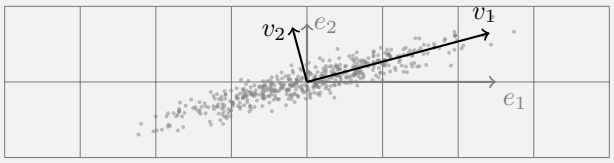


Figura 3: *Componentes principales de un conjunto de muestras en* $\mathbb{R}^2$.

Finalmente, para clasificar una muestra p-dimensional nueva, teniendo un conjunto de n datos de la misma dimensión, el modelo primero llevará a cabo un cambio de base usando PCA, para luego clasificar la muestra nueva utilizando *kNN*.

2 Implementación

Habiendo introducido los conceptos fundamentales y el desarrollo teórico del modelo, ahora se busca explicar la implementación del mismo en nuestro contexto de uso.

2.1 Vectorización

Cada imagen del conjunto de datos (1.2) es representada como una matriz en $\mathbb{N}^{28\times 28}$, sea una muestra $y^{(i)} \in \mathbb{N}^{28\times 28}$ y una función $\phi\colon \mathbb{N}^{28\times 28} \to \mathbb{N}^{784}$ tal que concatena los pixels de $y^{(i)}$ en cierto orden como muestra la Figura 4. Aplicamos ϕ a todas las muestras $y^{(i)}$ para construir la matriz $X \in \mathbb{N}^{n\times 784}$ tal que la fila i-ésima de X es igual a $\phi(y^{(i)})$.

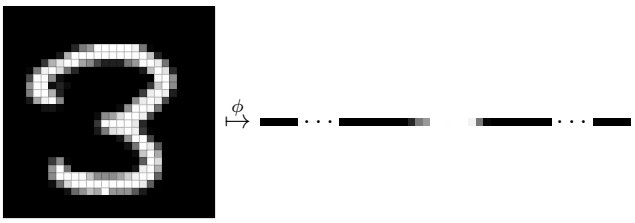


Figura 4: *Transformación de la muestra de* $\mathbb{N}^{28\times 28}$ *a* $\mathbb{N}^{728}$.

2.2 Implementando PCA

Para implementar PCA, primero vamos a centrar todas las muestras del conjunto base. Para eso calculamos la esperanza de cada componente, y sea x_i la columna i-ésima de X,

$$\mu = \begin{bmatrix} \overline{x_1} \\ \overline{x_2} \\ \vdots \\ \overline{x_n} \end{bmatrix}.$$

Ahora centramos las muestras restando la esperanza,

$$\tilde{X} = \frac{X - (\mathbf{1} \cdot \mu^t)}{\sqrt{n-1}}.$$

Donde $\mathbf{1} \in \mathbb{R}^n$ y $\mathbf{1}_i = 1$ para todo $1 \leq i \leq n$.

Una vez centradas las muestras se calcula la matriz de covarianza $M_X = \tilde{X}^t \cdot \tilde{X}$ que contiene la varianza en su diagonal y la covarianza entre la muestra i y j en $(M_X)_{ij}$. Notemos que la matriz de covarianza es simétrica, entonces cuenta con una base de autovectores y p autovalores distintos. Se diagonaliza la matriz utilizando el método de la potencia 2.2.1.

2.2.1 Método de la potencia

Sea $A \in \mathbb{R}^{n \times n}$ una matriz con base de autovectores y autovalores tales que $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ y dado un x_0 tal que $x_0 = \beta \cdot v_1 + c$, donde $\beta \neq 0$ y v_1 es el autovector asociado al autovalor dominante, el Método de la potencia aproxima el autovector dominante con la sucesión q, para luego calcular el autovalor dominante utilizando el coeficiente de Rayleigh.

$$q^k = \frac{Aq^{k-1}}{\|Aq^{k-1}\|_2}, \text{donde } q^0 = x_0 \text{ y el coeficiente de Rayleigh es } \frac{(q^k)^t \cdot A \cdot q^k}{(q^k)^t \cdot q^k}.$$

Se lo implementa de forma iterativa, donde se discutirán distintas formas de implementación en la sección de experimentación posterior.

Para reducir la dimensión ejecutaremos el método de la potencia α veces (parámetro de PCA), obteniendo α componentes principales. Estos componentes forman la matriz para el cambio de base. El autovalor dominante de la matriz de covarianza hace referencia a la magnitud de dispersión de la componente mas independiente del resto, mientras que su autovalor asociado hará referencia a su dirección.

La matriz de cambio de base será la asociada a la transformación característica a aplicar a cada muestra nueva que se desee etiquetar.

$$\text{tc}(x^{(i)}) = \frac{x^{(i)} - \mu}{\sqrt{n-1}}$$

Una vez transformada, se llevará a cabo la clasificación utilizando *kNN* sobre el conjunto de muestras de entrenamiento ya transformado.

Notemos lo siguiente: una vez aplicada la transformación característica, todas las imágenes de las muestras son una combinación lineal de los componentes principales, los autovectores de la matriz de covarianza. Luego, se puede pensar que cada muestra está compuesta por los siguientes *autodígitos* (producto de visualizar los autovectores). En la figura 5 exponemos la visualización de los autovectores del cambio de base, para $\alpha = 5$.

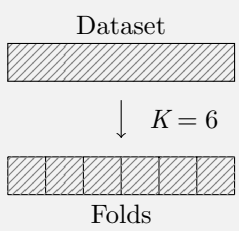
Figura 5: *Visualizacion de los pirmeros 5 autovectores de la matriz de covarianza de la base de datos.*

3 Experimentación

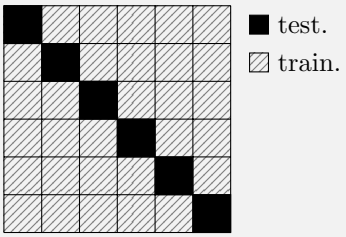
Se introducirán las clases de predicción y la matriz de confusión y en función de estas las métricas a utilizar. La experimentación buscará ver como varía la eficiencia del modelo en función a los parámetros más importantes de los métodos utilizados, como cantidad de vecinos a tener en cuenta en kNN (k), la cantidad de componentes principales a calcular (α) o simplemente la cantidad de muestras tomadas de la base de datos (n). También habra una sección dedicada a la experimentación asociada al método de la potencia. Por último, hay que mencionar que para la experimentación siempre fue de uso el K-Fold cross validation, explicada en 3.0.1.

3.0.1 *K-fold cross-validation*

El *K-fold* es una técnica de validación cruzada, en la que el conjunto de datos se divide en k subconjuntos (pliegues). El objetivo es utilizar $k-1$ pliegues en el entrenamiento del modelo, y el restante para la comprobación de eficacia. Este análisis se hace con todas las configuraciones de pliegues posibles, eligiendo uno para comprobar y entrenando en el resto como se ve en la Figura (6).
Esta metodología permite variar el *dataset* sobre el que se hace la comprobación, ya que de tenerse un único conjunto de muestras para el testeo no es posible llevar a cabo una buena medición de precisión con respecto a las clases que no se vieron (a menos que el conjunto sea demasiado extenso).



(A) *Partición en 6* pliegues.

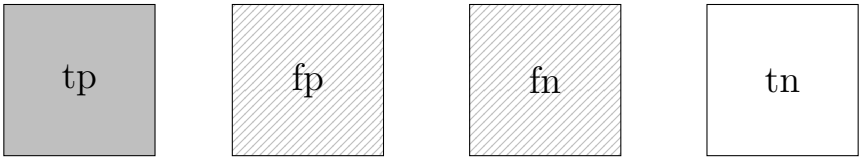


(B) *Visualizacion de los distintos modelos.*

Figura 6: *Primero se particiona el dataset en* pliegues **(A)** *luego se realiza la* cross-validation *utilizando una vez cada* pliegue *como el conjunto de* test *y el resto de los pliegues como* train **(B)**.

3.1 Clases de predicción.

Al etiquetar una muestra nueva, la predicción pertenece a una de cuatro clases, cuando se ve desde el punto de vista de una clase $\overline{k}$. Las clases son las exhibidas en la Figura 7



(A) *True Positive.* (B) *False Positive.* (C) *False Negative.* (D) *True Negative.*

Figura 7: *Posibles clasificaciones de una predicción.*

Sea x una muestra que pertenece a la clase $\overline{i}$, tal que se predice que pertenece a la clase $\overline{j}$ entonces si estoy analizando la clase $\overline{k}$, la predicción de x es

$$\text{predicción}(\overline{k}, \overline{i}, \overline{j}) = \begin{cases} \text{tp} & \text{si } \overline{k} = \overline{i} = \overline{j} \\ \text{fp} & \text{si } \overline{k} = \overline{j} \wedge \overline{j} \neq \overline{i} \\ \text{fn} & \text{si } \overline{k} = \overline{i} \wedge \overline{j} \neq \overline{i} \\ \text{tn} & \text{si } \overline{k} \neq \overline{j} \wedge \overline{k} \neq \overline{i} \end{cases}$$

3.2 Matriz de Confusión

Sea $\mathcal{L}$ una familia de m clases y $x^{(i)}$ un conjunto de muestras pertenecientes a alguna clase de $\mathcal{L}$, sea la matriz de confusión $C \in \mathbb{N}^{m \times m}$ tal que

$$C_{jk} = \sum_{i=1}^{n} \begin{cases} 1 & \text{si } x^{(i)} \in \overline{j} \text{ y el modelo predice que } x^{(i)} \in \overline{k}. \\ 0 & \text{en otro caso.} \end{cases}$$

Con $\overline{j}$, $\overline{k}$ clases de la familia $\mathcal{L}$, dicho de otra forma

$$C_{jk} = \text{Cantidad de veces que se predice } k \text{ cuando la clase verdadera era } j.$$

Notar que la j-ésima fila hace referencia a los casos en los que se sabe que la clase es $\overline{j}$, mientras que la k-ésima columna refiere a los casos en que se predice $\overline{k}$.

La matriz de confusión permite obtener fácilmente datos respecto a cada clase de equivalencia analizando sus filas y columnas como muestra la Figura 8.

tn	tn	fp	tn	tn
tn	tn	fp	tn	tn
fn	fn	tp	fn	fn
tn	tn	fp	tn	tn
tn	tn	fp	tn	tn

Figura 8: *Análisis de la clase* 3 *en un conjunto de* 5 *clases,* C_{33} *contiene los aciertos,* C_{i3}, $i \neq 3$ *las predicciones de* 3 *que eran* i *y* C_{3i}, $i \neq 3$ *las predicciones de* i *que eran* 3.

3.3 Métricas

En función a las clases de predicción y la matriz de confusión se puede derivar el cálculo de distintas métricas para poder estudiar el rendimiento del modelo. Entre estas se encuentra la exactitud (*accuracy*), la precisión (*precision*) y la exhaustividad (*recall*). A partir de estas medidas se diferencian los modelos exhaustivos de los precisos, donde la eficiencia de cada uno de ellos depende del contexto de uso.

3.3.1 *Accuracy*

Accuracy o exactitud, es una medida de la efectividad general del modelo, definida como

$$accuracy = \frac{\#\text{aciertos}}{\#\text{muestras}}.$$

En función de la matriz de confusión tenemos

$$accuracy = \frac{\text{tr}(C)}{\sum_i \sum_j C_{ij}}.$$

3.3.2 *Precision*

Precision es la cantidad de elementos correctamente clasificados de la clase observada, sobre la cantidad total de muestras que fueron clasificadas como la clase observada. Sea esta la clase i

$$precision = \frac{\#tp}{\#tp + \#fp}.$$

En función a la matriz de confusión, sea la clase observada la clase i

$$precision = \frac{C_{ii}}{\sum_j C_{ij}}.$$

3.3.3 *Recall*

Recall hace referencia a cuantas clasificaciones fueron acertadas de la clase observada, sobre el total de elementos que son verdaderamente de esa clase independientemente de la clasificación.

$$recall = \frac{\#tp}{\#tp + \#fn}.$$

Y con la matriz de confusión, para la clase i se calcula

$$recall = \frac{C_{ii}}{\sum_j C_{ji}}.$$

3.4 Experimentos

Se utilizaron dos computadoras para ejecutar los experimentos. La primera constituida por un Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz, memoria 16GB RAM @ 1600mhz, y un disco Samsung 960 evo SSD 1TB en un sistema operativo Manjaro Linux. La segunda cumplía las siguientes espcificaciones: AMD Ryzen 7 2700 @ 4.3GHz, 16GB de memoria RAM, y un disco rígido de 1TB usando también Manjaro Linux.

3.4.1 k y α

Objetivo

Obervar para cada α, el k que obtiene mejor *accuracy*.

Para los modelos definidos con $K = 5$ pliegues, se ejecutó PCA con
α y $k \in \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 15,\ 20,\ 30,\ 40,\ 50,\ 75,\ 100,\ 150,\ 200\}$ el número de muestras fue 5000.

Hipótesis

Lo esperado de este experimento es contemplar como el *accuracy* siempre mejora a medida que crece α, ya que cuantos más componentes principales, más información del modelo se tiene en cuenta.

Sin embargo, progresivamente también se empezará a tener en cuenta información poco aportativa. Siendo los autovalores de la matriz de covarianza una medida de dispersión para los componentes independientemente del resto, llegará un punto en el que para un valor de α muy grande, el método de la potencia (junto con la deflación) comenzará a devolver a los autovalores menos dominantes, haciendo que se tenga en cuenta componentes con poca varianza, o sea, componentes que aportan poca información del modelo. Por lo tanto, a medida que crece α, el *accuracy* siempre mejorará, pero esta mejoría decrecerá rápidamente.

Por otro lado, esperamos que a medida que se incrementa k, el *accuracy* también suba hasta que se alcance un pico para algún k especifico, para luego comenzar a decrecer. Esto es porque, si k es un número significativo, se comienza a tener en cuenta el valor de las etiquetas de vecinos demasiado lejanos, de clases muy distintas. Adicionalmente, tener un k demasiado bajo no puede dar un buen rendimiento. En el caso en que $k = 1$ solo vamos a estar clasificando a una muestra nueva en función a la primera mas cercana. Puede ocurrir que la muestra en verdad es (por ejemplo) una imagen de un uno, que está demasiado cerca a una muestra del 7.

Resultado

En la figura 9 se puede ver como a medida que se incrementa α, el *accuracy* incrementa para todos los valores de k. Para $\alpha \approx 40$ la tasa de aciertos alcanza valores sumamente altos. Sin embargo, a medida que α tomó valores más grandes, para $k = 5$ no se sostuvo el accuracy, ni tampoco tuvo leves mejorías, solo comenzó a decrecer. A medida que sube α, se toman en cuenta nuevos autovectores que aportan poca información, presentan "ruido" en la clasificación, bajando el rendimiento general. Por ejemplo con $\alpha = 200$ el k "óptimo" presenta peores resultados. Con respecto a k, para todo valor de α se alcanzó un pico para valores bajos y el rendimiento comenzó a bajar drásticamente a medida que se incrementaba k.

Se concluye que el *accuracy* comenzó a decrecer a partir de $\alpha \approx 40$ debido a que se incorporaron componentes con el mismo peso que las anteriores pero menos descriptivas de los datos.

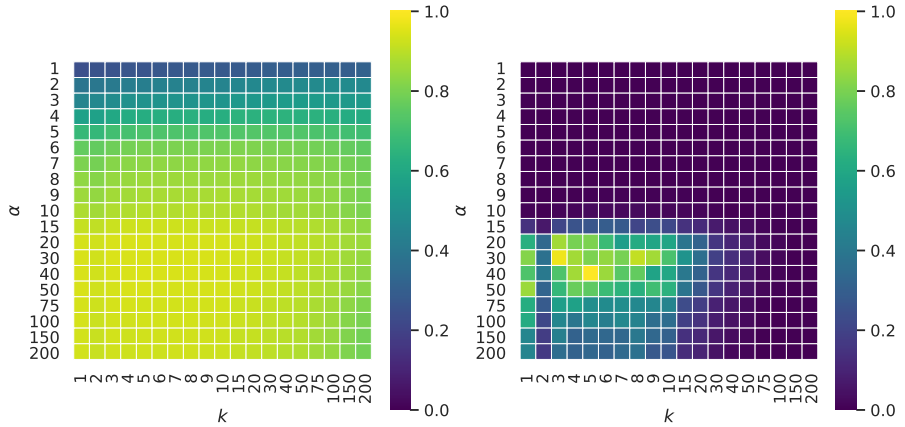


Figura 9: *A la izquierda heatmap de accuracy para cada par* (α, k)*; a la derecha el mismo heatmap con cada valor elevado a la* 64*, normalizado pico a pico.*

3.4.2 *kNN*. ¿Con o sin PCA?

Objetivo

Observar la diferencia de *accuracy* entre *kNN* cuando se preprocesan los vectores con PCA y cuando utilizamos los vectores como vienen.

Para los modelos definidos con $K = 10$ pliegues, se ejecutó *kNN* combinado con PCA, $\alpha = 20$ y también sin el uso de PCA, simplemente clasificando en función a las muestras vecinas sin disminuir la dimensión para $k \in \{1, 2, \ldots, 25\} \cup \{25, 30, \ldots, 50\} \cup \{50, 60, \ldots, 100\}$ y 5000 muestras.

Hipótesis

Lo esperado es contemplar como el uso de PCA presenta mejor *accuracy*. Esto se debe a que PCA es una herramienta para disminuir la dimensión de las muestras de la forma mas descriptiva posible, sino que además los componentes principales ayudan a predecir en función a los rasgos mas importantes de cada dígito. Se espera una mejor *accuracy* al utilizar PCA.

Resultados

Se puede observar en la figura 10 como la tasa de aciertos al utilizar PCA es superior. Ambas curvas decrecen a medida que aumenta la cantidad de vecinos que se tienen en cuenta, sin embargo el uso de PCA siempre presenta mejor **accuracy**. También se puede obervar que para k = 5 aproximadamente, se alcanza un pico muy elevado cerca del 0.95, utilizando PCA o no. Las bandas indican el intervalo de confianza 0.95.

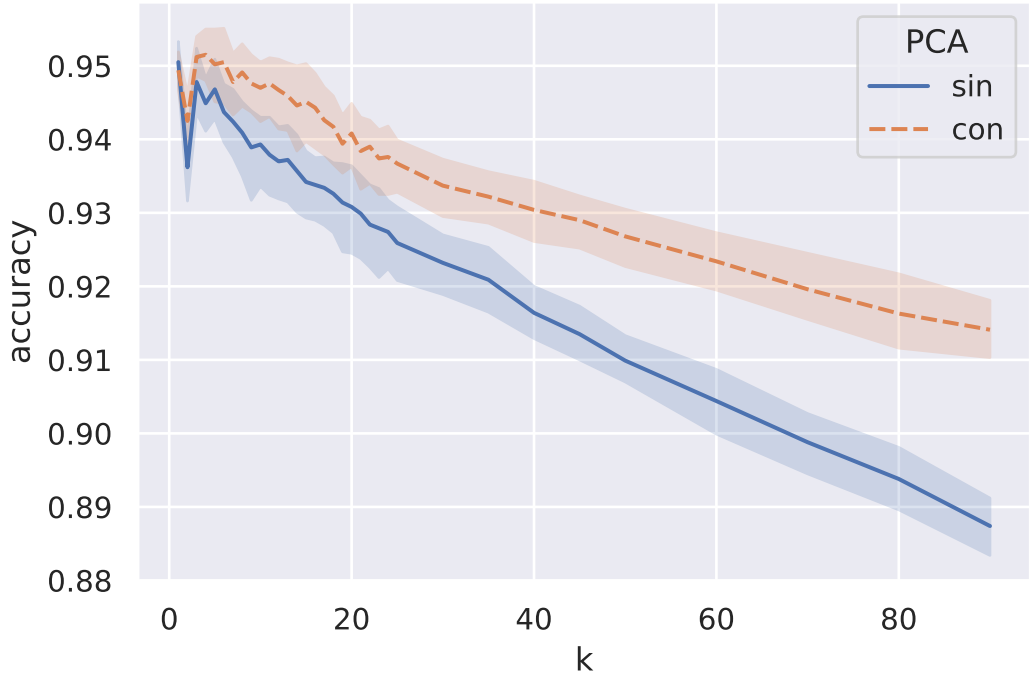


Figura 10: *Accuracy en función de k para kNN con PCA y sin.*

3.4.3 El k indicado para cada n

Objetivo

Buscar el valor k que tenga mejor *accuracy* para cada n muestras.

Para los modelos definidos con $K = 5$ se ejecutó *kNN* con $k \in \{1, \ldots, 100\}$, PCA con $\alpha = 20$ para distintas cantidades de muestras, $n \in \{200, 225, \ldots, 5000\}$.

Hipótesis

Fue observado en el experimento 3.4.1 que a medida que aumenta k más allá de 15 $\sim$ el *accuracy* decrece, lo supuesto a medida que aumenta n, el *accuracy* para todo k aumente también. Esto es producto de pensar que cuantas más muestras, más alta es la probabilidad de encontrar una muestra similar a la que quiera clasificar. Si tuvieramos las 256^{784} posibles muestras etiquetadas la distancia entre el primer vecino más cercano y un vector a etiquetar es cero..

Resultados

En la figura 11 se puede observar el *accuracy* para cada n normalizado pico a pico (con el mínimo representado por 0 y el máximo por 1). Para todo n el k óptimo es pequeño, independientemente de n. El resultado se condice con el experimento (3.4.1), donde los mejores valores para k se encuentran en $\{5,\ 6,\ \ldots,\ 10\}$.

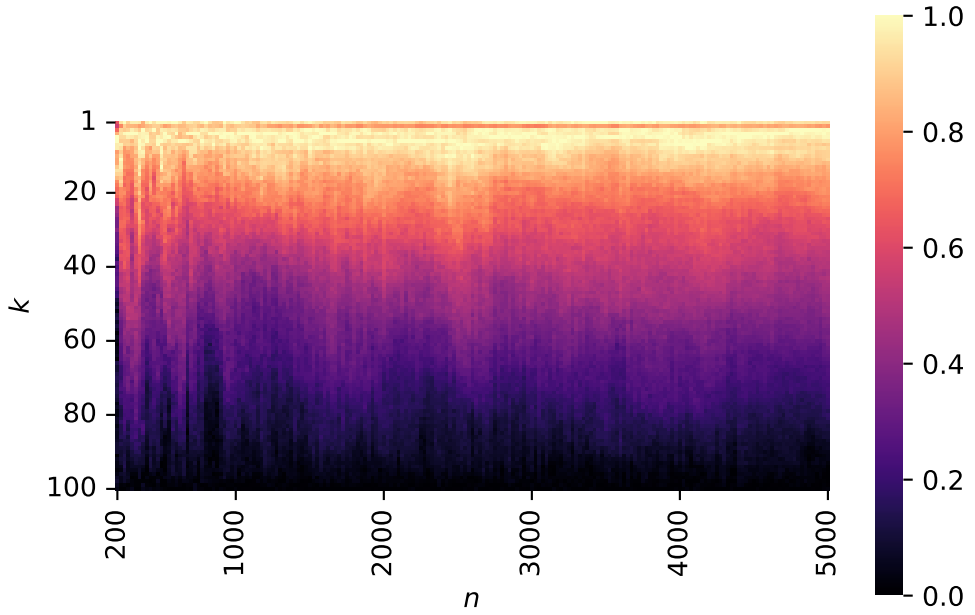


Figura 11: *Accuracy en función de k y n.*

3.4.4 Medicion de tiempos con y sin PCA

Objetivo

El objetivo del experimento es medir los tiempos de cómputo para dos instancias: Una que utilice PCA con $\alpha = 21$ y $k = 5$, mientras que la segunda no utilizará PCA. La idea es visualizar como los tiempos de cómputo crecen en función de la cantidad de muestras de las base de datos.

Hipótesis del experimento

Lo esperado es observar como PCA reduce los tiempos drásticamente, mientras que las demás métricas no cambiarán demasiado en comparación a los experimentos anteriores. Esto se debe a que la búsqueda de kNN al utilizar PCA será con vectores en $\mathbb{R}^{21}$ mientras que al no utilizarlo será en $\mathbb{R}^{784}$.

Resultados

Se puede observar en la figura 10 como los tiempos de ejecución de ambas instancias crecen de forma monótona. Ocurrió lo esperado, el tiempo al utilizar PCA se reduce considerablemente en comparación a la instancia que no la utilizaba. Sin embargo, también se puede observar como inicialmente los tiempos de la primera son mayores: se puedo apreciar el costo inicial de computar la transformación característica. Consecuentemente, por mas que su cómputo implique cierto costo de *overhead*, realizar *kNN* con PCA se vuelve casi despreciable a medida que aumentan las muestras.

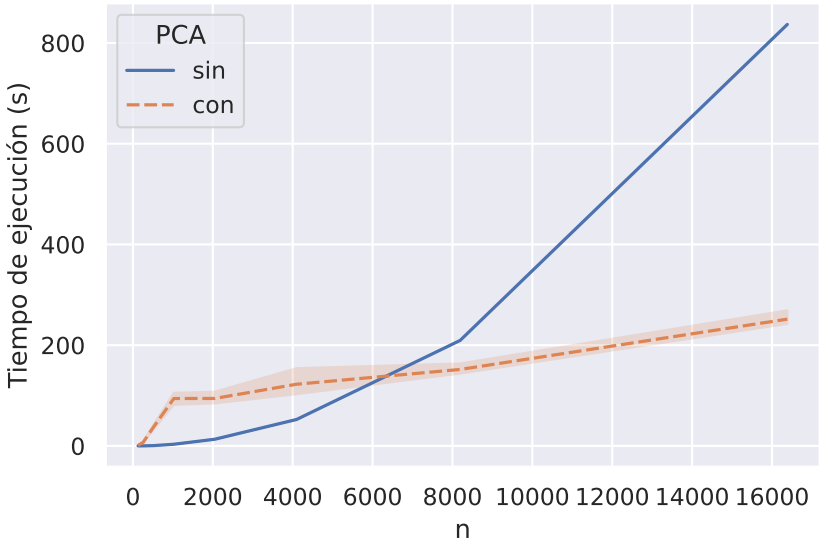


Figura 12: *Tiempos de ejecución en función al con y sin PCA.*

3.4.5 Grado de estrictez

Objetivo

El objetivo del experimento es ver como, agregando un "grado de estrictez" cambian las métricas de *precision* y *recall* en función del mismo. El grado de estrictez es un porcentaje, y cambia la forma en la que se clasifica a una muestra nueva al ejecutar *kNN*. Nosotros sabemos que *kNN* se encarga de clasificar la muestra en función a la etiqueta de los k vecinos mas cercanos. Con el grado de estrictez (g), le pedimos al modelo que solo etiquete la muestra nueva si de los k vecinos, $g \cdot k$ o más son de la misma clase (la clase con más vecinos cercanos). Por ejemplo, en el caso en que estemos usando *kNN* con $k = 4$, y los vecinos de nuestra muestra x corresponden a las clases $votos = (1,\ 2,\ 2,\ 6)$, con un grado igual a cero (caso estándar) se clasificaría a x como una imagen que corresponde al 2, con $g = 0.5$, también se lo clasificaría como un 2 ya que $0.5 \cdot 4 \leq \#votos_A(2)$. Sin embargo, para los casos en los que $0.5 < g$, se introdujo una nueva clase para la predicción, el (-1). Cuando $g \cdot k > \#votos(\text{claseConMasVotos})$, el modelo clasificará a x como un (-1), indicando que los vecinos no cumplen con la estrictez requerida para la clasificación. El experimento se ejecutó

para $n = 20000$, usando K-Fold con $K = 5$, kNN con $k = 20$ y PCA con $\alpha = 13$.

Recall vs Precision

Preferir un módelo con mejor Precision o Recall depende completamente del contexto en el que se ejecutará el modelo, junto con el tipo de información que se maneja. En un modelo donde se tiene que asegurar que todos los casos positivos (teniendo únicamente dos clases, positivo y negativo) sean predecidos, se necesita un mejor puntaje de Recall que de Precision. Esto se debe a que no importa cuantas de las muestras se clasificaron correctamente de todas las clasificadas, sino que importa más haber clasificado bien a las que si eran. Un contexto típico donde este escenario se presenta es al diagnosticar enfermedades virales: es preferible que la mayor parte de la población tome las precauciones indicadas, antes de que solo la parte de la población que estoy seguro que deben tomarlas efectivamente lo hagan. Un escenario donde es preferible un mejor grado de Precision es al buscar texto dentro de un conjunto de documentos: es preferible tener pocas instancias del texto encotnradas a muchas erroneas.

Hipótesis

Lo esperado, es ver que a medida que aumenta el grado de estrictez, la *precision* promedio del modelo suba mientras que el *recall* y el *accuracy* bajan. Esto se debe a que a medida que sube el grado, es más difícil clasificar una muestra (ya que cada vez se piden más vecinos de la misma clase), pero sin embargo, los pocos que sean clasificados con una clase distinta al (-1) serán aquellos que pertenecen a la clase predicha con mucha seguridad. Luego, la precisión debería subir ya que las muestras fueron clasificadas con mucha "seguridad". Consecuentemente el *recall* debería bajar, ya que a medida que sube la cantidad de muestras clasificadas como (-1), sube la cantidad de muestras que eran de cierta clase, pero se las clasifico como (-1) al tener un grado alto de estrictez. Luego, el porcentaje de muestras que eran de cierta clase, y se las clasificó como la misma bajo.

Resultados

En la figura 13 se puede observar como a medida que sube el grado de estrictez, sube la cantidad de muestras clasificadas como (-1), y consecuentemente sube proporcionalmente la *precision* y baja el *recall*.

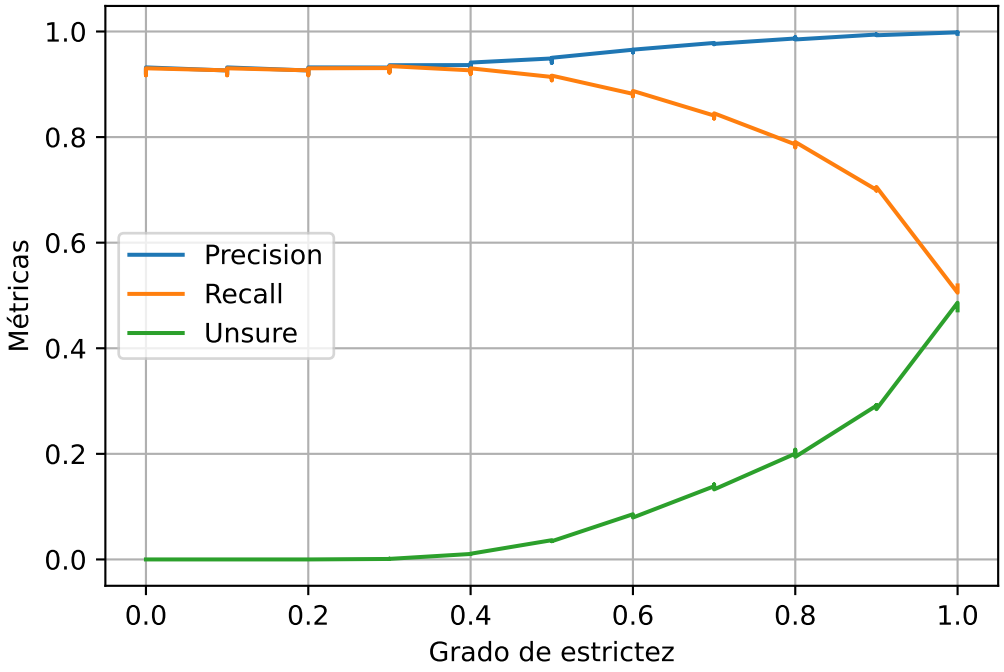


Figura 13: *Métricas en funcion al grado de estrictez.*

3.4.6 Grado de recall

Objetivo

El objetivo es agregar un "grado de recall" y observar cambios en las métricas de *precision* y *recall*. El grado de *recall* es un vector r, de longitud igual a la cantidad de etiquetas. Dentro de la clasificación, al momento de hacer el recuento de votos, a cada etiqueta t_i se le suma una cantidad de votos equivalente a $r_i \cdot \max v$, esto es, cuando $r_i = 1$ a la etiqueta i se le suman $\max v$ votos. De esta forma a medida que aumenta r_i aumenta el *bias* de la etiqueta i. Al aumentar el *bias* su grado de *recall* aumenta, ya que tiene un grado de preferencia sobre las demás clases. Además esto baja el *accuracy* y la precisión general ya que más y más muestras son incorrectamente catalogadas como la clase favorecida. El experimento se ejecutó para $n = 20000$, usando K-Fold con $K = 5$, kNN con $k = 20$ y PCA con $\alpha = 13$.

Hipotesis

Ya que se beneficia a una clase es razonable esperar que esta clase aumente su recall, por otro lado como este beneficio es independiente de cualquier indicio de que la instancia pertenezca a esta clase entonces las demás clases perderán *recall*. Por otro lado el *accuracy* y *precision* global disminuirá también.

Resultados

Se puede *apreciar* en la figura 14 como a medida que aumenta el grado de recall para el número 3 aumenta su recall, pero imultaneamente cae la *precisión* y el *accuracy* global. El 3 fue elegido por tener un *recall* menor a la mayoría de las clases.

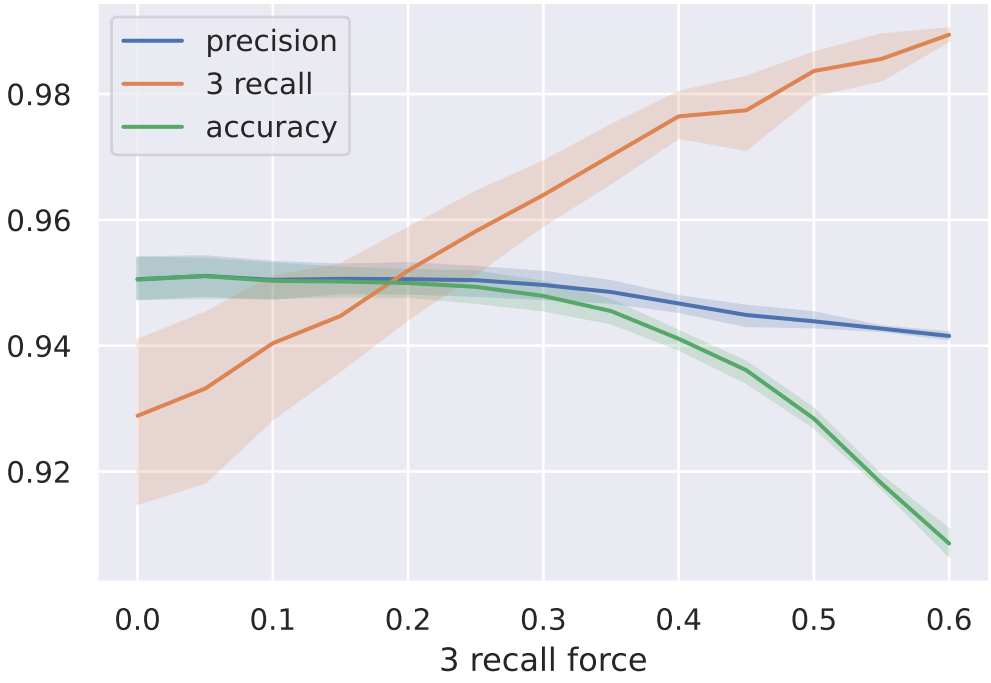


Figura 14: *Métricas en función al grado de recall.*

Distintas implementaciones del método de la potencia

El método de la potencia es un proceso iterativo, que utilizando una muestra x_0 cualquiera logra converger a un autovector del que se puede calcular el autovalor asociado utilizando el coeficiente de Rayleigh. Sin embargo, al implementarlo se necesitan hacer un par de ajustes: La computadora al no poder representar todos los números del mundo continuo, puede suceder que trate de converger a un autovalor con números no representables al usar aritmética de punto flotante. Sin embargo, es suficiente con acercarse lo suficiente a la solución, fijando cierto épsilon (ϵ de ahora en mas). Sin embargo, para seguir iterando al utilizar el método de la potencia, podemos ver cuanto se acerca Ax_i a $\lambda \cdot x_i$, y parar de iterar cuando el módulo de la resta es menor a cierto épsilon. Ó también se puede evaluar si el vector de la iteración es muy distinto al anterior, y detener la iteración cuando ya son suficientemente parecidos. Esta sección es dedicada a experimentar con los guardianes del

método de la potencia, y para ello definimos los siguientes predicados:

$$B_1(v_i) \equiv |A \cdot v_i - \lambda \cdot v_i| < \epsilon$$
$$B_2(v_i) \equiv |v_i - v_{i-1}| \quad < \epsilon$$

donde v_i es el autovector aproximado en la i-ésima iteración y λ es el autovalor asociado (coeficiente de Rayleigh).

3.5 Cantidad de iteraciones

Objetivo

En este experimento se busca estudiar cuantas iteraciones del método de la potencia son necesarias para poder converger a buenos autovalores de la matriz. Por lo tanto, se creará una matriz simétrica (tal que tenga base de autovectores) y en función a la cantidad de iteraciones se medirá la calidad de los autovectores aproximados comparándolos con los calculados por el método `eig`, de la biblioteca `numpy.linalg`. Para la ejecución del experimento, se tomó un promedio de las mediciones en función a 100 instancias distintas, donde se le calcularon a una matriz (distinta para cada instancia) los primeros 50 autovalores. Luego, se observó la diferencia de los autovalores estimados con el método de la potencia con los autovalores calculados con Numpy. La diferencia es

$$\frac{\sum_{i=1}^{50} |\texttt{estimado}(A)_i - \texttt{numpy.eig}(A)_i|}{50}.$$

Hipótesis

Intuimos que la calidad de los autovalores sube exponencialmente luego de cierta cantidad de iteraciones, pero a partir de cierto punto también se espera que no hayan más mejorías. Consecuentemente nuestras mediciones de diferencia deberían comenzar en valores relativamente altos y luego converger a cero. Esto se debe a que teóricamente se converge a un autovector, y luego de cierta cantidad de iteraciones la diferencia debería ser despreciable, y por lo tanto la diferencia entre autovalores también deberá serlo.

Resultados

Se puede observar como constantemente la diferencia entre los autovalores calculados es cada vez menor, convergiendo a cero. Luego de 30000 iteraciones, la aproximación del método de la potencia es lo suficientemente buena como para utilizar los autovalores en el modelo. Hay que notar que son 30000 iteraciones totales, por lo tanto el cálculo de cada autovector tomó (en promedio) 600 iteraciones para alcanzar una diferencia baja. Sin embargo,luego de las 40000 iteraciones la diferencia ya es extremadamente chica. En caso de que se pida un error menor a 0.05, debería ejecutarse el método minimo 400 veces por autovalor.

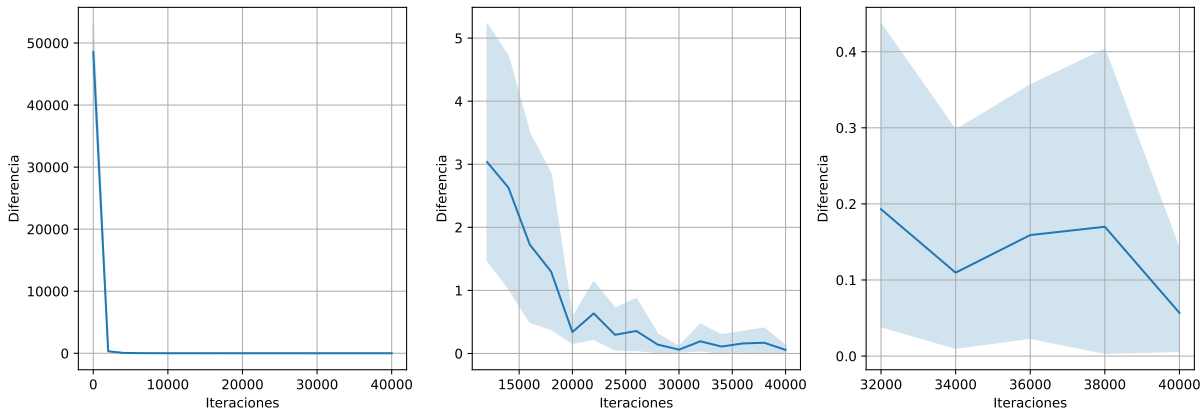


Figura 15: *Diferencia de los autovalores en función a la cantidad de iteraciones.*

Además, como era esperado, el tiempo requerido por número de iteraciones creció linealmente, como puede apreciarse en la figura 16.

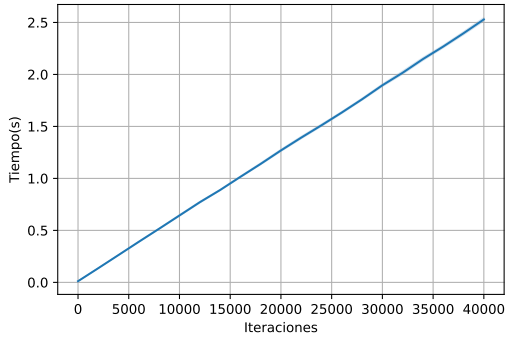


Figura 16: *Mediciones de tiempo con respecto a la cantidad de iteraciones.*

3.6 Comparación de guardianes

Objetivo

El objetivo es comparar la eficiencia del método de la potencia usando tanto a B_1 y B_2 como guardas. Se comparará la calidad de los resultados en función del épsilon, junto con los tiempos de cómputo asociados. Para esto se graficará la diferencia entre los autovalores aproximados y los de Numpy. Luego se determinará cual de los dos métodos es mas eficiente en relación a la calidad y tiempo. Notar que se conseguirá una diferencia muy baja, cuando se llegue a la cantidad de iteraciones óptima para la aproximación de los autovalores, vista en el experimento anterior. Este experimento, al igual que el anterior, se ejecutará 100 veces y se graficará el promedio de las mediciones. Para todo épsilon sé calcularon los primeros 50 autovalores de un matriz distinta entre

instancias. Los épsilon para los que se corrió el experimento son 0.1, 0.01, ..., $1 \cdot 10^{-8}$. Se utilizan estos epsilons, ya que Numpy tiene un error de $+-1 \cdot 10^{-9}$, por lo tanto solo tiene sentido comparar con numpy hasta esa diferencia.

Hipótesis

Siendo que B_1 debe ejecutar una multiplicación matricial además de una resta entre vectores, esperamos ver que los tiempos de ejecución son mucho mayores, más teniendo en cuenta el tamaño de la matriz a la que se le calcularan los autovectores. Sin embargo, con respecto a la calidad de los datos en función al ϵ, no esperamos ver demasiada diferencia. Esto se debe a que la diferencia entre un vector antes y luego de aplicarle la transformación lineal, debería ser muy parecida a la diferencia que tiene con el autovector aproximado.

Resultados

Se observa en la figura 17, que hay una diferencia sumamente grande al utilizar B_2 como guarda para epsilons muy altos. Sin embargo esto es comprensible, ya que no se puede esperar mucha exactitud con esos épsilon. Sorprendentemente, B_1 tiene un muy buen rendimiento para epsilons altos. Sin embargo a partir de $\epsilon = 1 \cdot 10^{-4}$ se ve como, independientemente de la guarda, las diferencias en promedio menores a 0.02, implicando que independientemente de la guarda en uso se puede conseguir un rendimiento aceptable. Sin embargo, para épsilon muy bajos, el mejor rendimiento en general lo tiene B_2, ya que B_1 presenta resultados inconsistentes: tiene la menor diferencia en $1 \cdot 10^{-7}$ y a la vez la mas alta en $1 \cdot 10^{-8}$. Se puede concluir que ambas guardas son sumamente eficientes, pero la menos volátil es B_2 (en promedio). Aun así, ambos presentan buenos resultados y se tiene poca diferencia con respecto a los autovalores calculados por Numpy.

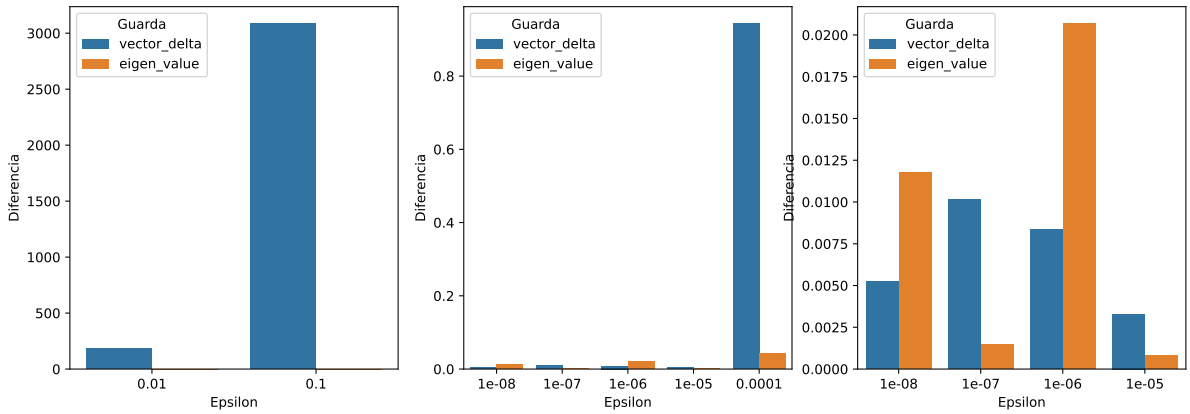


Figura 17: *Diferencia en función al épsilon para los guardianes B_1 y B_2.*

Sin embargo, con respecto a los tiempos, la figura 18 muestra, como fue esperado, que B_1 tiene tiempos de cómputo mucho mayores en comparación a los de B_2. Esto se debe a que para evaluar

la guarda se requiere efectuar una multiplicación matricial.

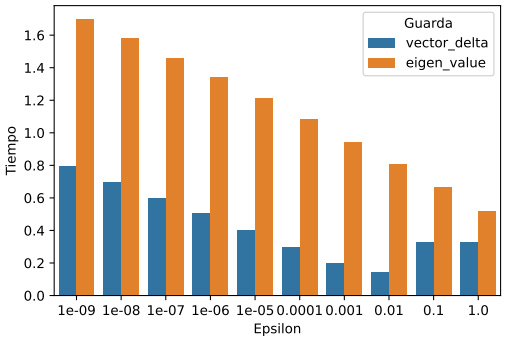


Figura 18: *Mediciones de tiempos (en segundos) en función de épsilon para los guardianes B_1 y B_2.*

3.7 Experimentación con respecto a la cantidad de Folds al utilizar K-Fold cross validation.

Objetivo

El objetivo de esta sección del trabajo es ver como mejora el accuracy general del sistema al utilizar distintas cantidades de Folds, para parámetros fijos ($\alpha = 13$ y $k = 5$), para cantidades altas y bajas de muestras de entrenamiento. Además, nos interesa ver los tiempos de cómputo, para encontrar el K óptimo. Se corrió el experimento con 500 muestras y con 10000. Además, los valores de K usados fueron 2,5,10,25 y 50.

Hipótesis

Es intuitivo pensar, que al tener muchas muestras (n alto), con una cantidad alta de Folds el modelo debería tener una mejor tasa de accuracy. Esto se debe a que se entranará con una buena parte del conjunto de muestras. Sin embargo,el accuracy para un K bajo debería seguir siendo relativamente bueno, aunque peor en comparación a los K mayores. Además, se espera ver que a medida que K crece se converge a un porcentaje de accuracy, ya que a partir de cierto punto tener Folds cada vez más chicos no debería generar grandes cambios. En el caso en que se tengan pocas muestras, el nivel de accuracy debería ser muy bajo en comparación al caso anterior, sin embargo la diferencia entre un K muy bajo y uno alto debería ser muy significativa. Con respecto a los tiempos de cómputo, estos deberían aumentar a medida que crece K, debido a que cada vez son más la cantidad de veces que se debe entrenar y testear el modelo.

Resultados

La figura 19 expone como efectivamente, independientemente de la cantidad de muestras de entrenamiento, siempre se converge a un valor de accuracy a medida que K crece: a partir de $K = 10$ la calidad del modelo es siempre la misma. El rendimiento general al tener pocas muestras es menor, como era de esperar, pero el accuracy se comporta igual en función de K. Sin embargo, cuando se tienen pocas muestras, un K menor a 10 implica una gran diferencia en rendimiento, fenómeno que no pasa para cantidades de muestras mayores.

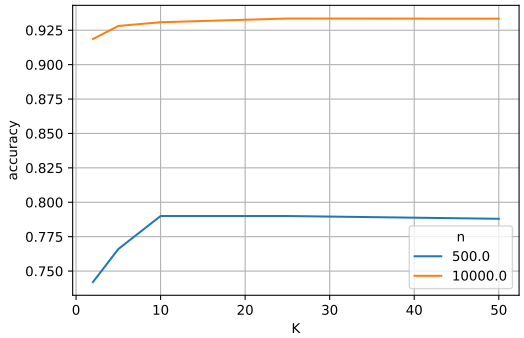


Figura 19: *Mediciones de acciracy en función a la cantidad de folds.*

Sin embargo, si vemos los tiempos de cómputo en la figura 20, estos crecen linealmente. Por lo tanto, se puede concluir que verdaderamente no tiene sentido usar una cantidad de Folds mayor a 10 para entrenar el modelo, ya que en caso contrario se tardará más y se conseguirá el mismo rendimiento.

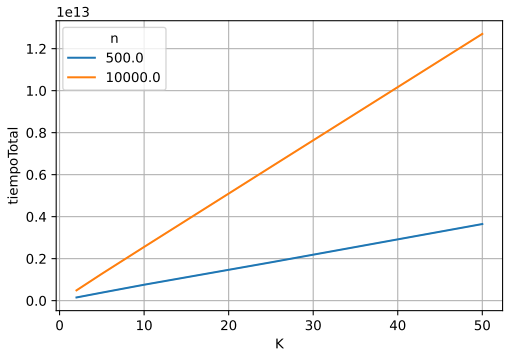


Figura 20: *Tiempos de cómputo en función a la cantidad de pliegues.*

4 Conclusiones

El trabajo presente abordó el reconocimiento de dígitos numéricos arábigos. Para la resolución del mismo se evaluaron los parámetros de *kNN* y PCA. Mediante experimentación se pudo concluir que los parámetros óptimos para *kNN* y PCA en término del *accuracy* rondan $k = 5$ y $\alpha = 20$ siendo k la cantidad de vecinos en *kNN* y α la cantidad de componentes en PCA. Además se corroboró que a medida que crece n el modelo tiene mejores resultados sin importar los parámetros elegidos. Adicionalmente se puedo comprobar que PCA presenta un tiempo de cómputo adicional al calcular la transformación característica, mientras que a medida que aumenta las cantidad de muestras, la ejecución de *kNN* con PCA se vuelve casi despreciable. Sin embargo, antes de las 6000 muestras el tiempo de calcular la transformación característica es mucho mayor, y se obtiene un rendimiento alto y sin mucha diferencia con o sin PCA. Por lo tanto, cuando se tienen menos de 6000 muestras, conviene no utilizar PCA, siempre que los parámetros α y k sean óptimos. También se comprobó que al darle la opción al modelo de clasificar a una muestra nueva como (-1) cuando no tiene una clasificación que cumple con las condiciones necesarias, sube significativamente la precisión del modelo, mientras que el *recall* baja. Analógamente, se pudo observar que agregando un grado de Recall para una clase en especial implica que aumente el Recall pero baje la precisión y accuracy. Se pudo experimentar con distintos métodos de corte para el método de la potencia, y se pudo determinar cuantas iteraciones son necesarias para que su implementación converja a autovectores aceptables, con (casi) la misma precisión que la biblioteca de python Numpy. Finalmente, también se pudo observar que el valor óptimo para la cantidad de folds es siempre 10, independientemente de la cantidad de muestras.

Referencias

[1] *Kaggle*. URL: `https://www.kaggle.com`.

[2] *Mnist database*. URL: `http://yann.lecun.com/exdb/mnist/`.