DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Reconocimiento de dígitos

Métodos Numéricos

Integrante	LU	Correo electrónico
Zolezzi, Maria Victoria	222/19	zolezzivic@gmail.com
Miceli, Juan Pablo	424/19	micelijuanpablo@gmail.com
Lavalle Cobo, Ignacio	282/19	ignacio.lavalle.cobo@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

En este trabajo nos dedicaremos a analizar el método de clasificación K vecinos más cercanos (KNN) para reconocer dígitos de la base MNIST, junto con su desempeño utilizando la técnica de Análisis de Componentes Principales (PCA) para disminuir la dimensión de los datos a utilizar, ya que nuestro datos poseen 784 dimensiones.

2. Glosario

Keywords:

K Nearest Neighbors (KNN),
Principal Component Analysis (PCA),
Cross Validation
Hiperparámetros

3. Introducción

Nuestro objetivo es inicializarnos en el mundo del machine learning, para ello experimentaremos sobre algunos métodos utilizados en el reconocimiento de dígitos en imágenes.
Analizaremos el funcionamiento de un algoritmo de clasificación (KNN) y otro de reducción de dimensionalidad (PCA) sobre el dataset MNIST. Para intentar conseguir los resultados más precisos y eficientes posibles tendremos que ver y estudiar el impacto de los hiperpárametros sobre ellos y sobre el tiempo de ejecución del modelo. Para esta tarea, utilizaremos validación simple y K-fold cross validation.

A lo largo de las experimentaciones fuimos cumpliendo los objetivos nombrados anteriormente, permitiéndonos ofrecer valores óptimos de los hiperparámetros para el correcto funcionamiento de estas técnicas que estarán disponibles para quien las necesite en el futuro.

4. Dataset a analizar

En esta oportunidad utilizaremos el set de datos MNIST. El mismo está compuesto por imágenes de dígitos manuscritos de 28x28 píxeles en escala de grises. En total, contamos con 70000 imágenes, que serán divididas en dos datasets: uno para training y el otro para testing, de 42000 y 28000 imágenes respectivamente. Es importante destacar que las imágenes de training vienen provistas con sus labels.

5. KNN

K vecinos más cercanos es un método de clasificación supervisada multiclase. En este caso, nosotros tenemos diez labels (uno por cada dígito). Para predecir un nuevo dato(x), teniendo los datos de training, se debe calcular la norma dos de la resta entre x y cada uno de los elementos del conjunto anteriormente nombrado. Este procedimiento nos permite ver qué tan parecidos son los datos con los que ya contábamos a nuestro nuevo dato.

Teniendo esta información podríamos quedarnos con el dato que minimiza el valor de la norma anterior (es decir, el elemento más parecido a x) y asignarle su etiqueta.

Este algoritmo puede generalizarse para tomar más de un vecino cercano. La idea sería tomar k (elegir los k datos del set de entrenamiento que minimizan las normas de arriba). Luego, se le asigna a x la clase a la que pertenecen la mayoría de ellos a través de una votación.

6. PCA

El Análisis de Componentes Principales es una técnica de reducción de la dimensión. Su objetivo es mantener la mayor cantidad de información posible al disminuir la dimensión de los datos de entrada mediante la aplicación de un cambio de base que reduzca la redundancia.

Sabiendo que la covarianza entre dos variables mide qué tanto varían conjuntamente, buscar un cambio de base que haga que la covarianza entre todas las variables sea cero sería una buena idea para eliminar la redundancia.

Para lograrlo, diagonalizaremos la matriz de covarianza.

Sea X la matriz que contiene a las muestras centradas en cero, es decir:

$$X = \begin{pmatrix} x_{1,1} - \mu_1 & x_{1,2} - \mu_2 & \dots & x_{1,n} - \mu_n \\ x_{2,1} - \mu_1 & x_{2,2} - \mu_2 & \dots & x_{2,n} - \mu_n \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} - \mu_1 & x_{n,2} - \mu_2 & \dots & x_{n,n} - \mu_n \end{pmatrix}$$

donde $x_{i,j}$ representa el píxel j de la imagen i y μ_j el promedio respecto de todas las imágenes para el píxel j.
La matriz de covarianza M se define como:

$$m_{i,j} = \frac{1}{n-1} \sum_{k=1}^{n} (x_{k,i} - \mu_i)(x_{k,j} - \mu_j) = \frac{1}{n-1} X^T X$$

Como dijimos anteriormente, estamos buscando un cambio de base que nos permita asegurar que la matriz de covarianza del X transformado sea diagonal.

Consideremos el cambio de base $\tilde{X} = AX^T$.
Veamos cómo se escribe la matriz de covarianza de $\tilde{X}$ a partir de la matriz de covarianza de X.

$$\tilde{M} = \frac{1}{n-1} \tilde{X}^T \tilde{X}$$

$$\tilde{M} = \frac{1}{n-1} (AX^T)(XA^T)$$

$$\tilde{M} = A \frac{X^T X}{n-1} A^T$$

$$\tilde{M} = AMA^T$$

Podemos afirmar que M es una matriz simétrica. Esto implica que tiene base ortonormal de autovectores, lo cual a su vez implica que es ortogormalmente diagonalizable. Por lo tanto, sabemos que existen matrices P, D, P^{-1}, con P y P^{-1} ortogonales y D diagonal tal que:

$$M = P^{-1}\ D\ P = P^T\ D\ P$$

Donde P^T tiene como columnas a los autovectores de M, y D tiene en su diagonal a los autovalores de M.
Luego,

$$\tilde{M} = A\ P^T\ D\ P\ A^T$$

Tomando $A = P$,

$$\tilde{M} = P\ P^T\ D\ P\ P^T$$

$$\tilde{M} = D$$

Por lo que aseguramos que la matriz de covarianza de los datos transformados es diagonal.

Recapitulando, para hallar el cambio de base, debemos calcular los autovectores de la matriz M, los cuales constituyen las componentes principales de los datos. Como la matriz M es simétrica semidefinida positiva, podemos utilizar el método de la potencia con deflación para conseguirlos. Este nos permite obtener los autovalores en orden descendente por lo que las primeras componentes concentrarán la mayor cantidad de información. Una vez que conseguimos los autovectores, basta con ponerlos como columnas de una nueva matriz (P) y premultiplicarla a X^T para transformar los datos.

7. K-fold Cross Validation

Para hallar la cantidad de vecinos (k) y de componentes (α) óptimas, necesitamos testear la perfomance de nuestro clasificador para un subconjunto de estos hiperparámetros. Para lograrlo, podemos separar nuestro dataset de training en 2: training y validación.

Ahora tenemos que entrenar el modelo usando los datos de training y calcular su accuracy con los de validación. Sin embargo, esto nos reduce la cantidad de datos originales de manera significativa y aumenta la probabilidad de que se produzca overfitting si la partición de training está sesgada.

Para solucionar este problema, la técnica K-fold cross validation propone dividir el set de training en K subconjuntos, de los cuales se tomará uno para validación y el resto para entrenamiento, y se los irá variando en K ciclos como se muestra en la siguiente figura 1.

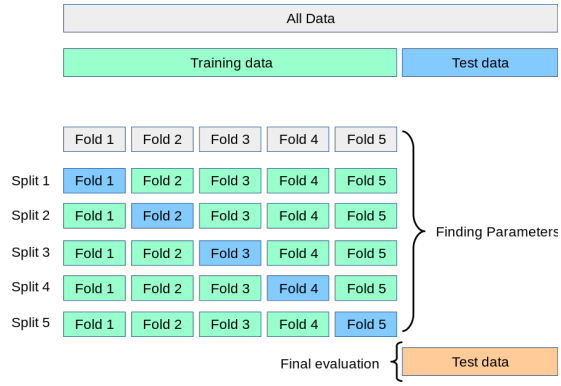


Figura 1: Procedimiento de cross validation para 5 folds.
Tomado de la documentación de scikit learn sobre cross validation.

El accuracy asignado a cada pareja se calcula tomando el promedio de las exactitudes de cada split.

El procedimiento ilustrado debe repetirse para cada pareja posible de hiperparámetros. Luego, se elije como mejor pareja aquella que arrojó mayor accuracy y con ella se calcula la tasa de efectividad de testing.

La cantidad de folds a considerar dependerá de cada dataset. En datasets muy pequeños, donde no podemos permitirnos perder datos de entrenamiento, se realiza un procedimiento denominado "Leave one out" el cual deja un único dato en el set de validación y todos los demás en el de entrenamiento. Por otra parte, en datasets con un número de datos generoso y no sesgados, como en nuestro caso, se puede reducir la cantidad de folds significativamente sin perder eficiencia.

Como es de imaginar, esta técnica conlleva mucho tiempo de cómputo, y el mismo va aumentando a medida que crece el subconjunto de hiperparámetros a considerar. Por esta razón, correremos nuestros experimentos con validación simple para hallar los candidatos a mejores k y α y luego utilizaremos esta técnica para quedarnos con la pareja óptima.

8. Métricas usadas

Para correr nuestros experimentos usaremos como métrica para comparar perfomances el accuracy. Este calcula la cantidad de imágenes clasificadas correctamente sobre la cantidad total de muestras.

Aparte del accuracy, existen otras métricas como precisión, recall o las F-measures.

- Precisión de la clase c: calcula la cantidad de muestras de la clase c que fueron bien clasificadas sobre la cantidad de muestras que fueron identificadas como pertenecientes a la clase c.
- Recall de la clase c: calcula la cantidad de muestras de la clase c que fueron bien clasificadas sobre la cantidad total de elementos de la clase c.
- La fórmula general de las F-measures está dada por: $F_\beta = (1+\beta^2)\dfrac{precision * recall}{\beta^2 precision + recall}$. Esta métrica se utiliza cuando las anteriores por sí solas no son suficientes, si no que se debe hacer un balance entre ambas.

Veamos cuál es la distribución en clases de nuestro dataset:

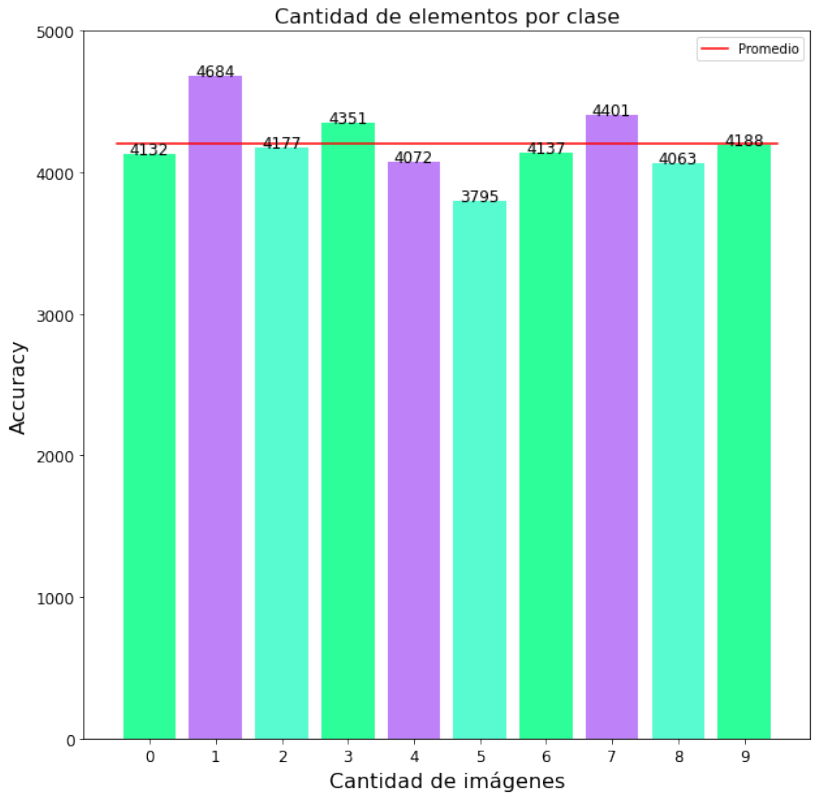


Figura 2: Cantidad de muestras por clase con la que cuenta el dataset MNIST.

Creemos que todas las métricas deberían arrojar resultados similares porque el dataset MNIST tiene todas sus clases bastante equilibradas como muestra la figura 2. Además, no estamos interesados en priorizar ninguna clase por sobre las otras. Por este motivo, a lo largo del trabajo utilizaremos como única métrica de interés el Accuracy. De todas formas, más adelante experimentaremos con algunas de las métricas nombradas anteriormente para intentar comprobar esta hipótesis.

9. Experimentación

9.1. Introducción

Para poder obtener un rendimiento óptimo de nuestro clasificador tenemos que determinar el valor del hiperparámetro k. Además, es sabido que la complejidad del algoritmo depende de la dimensión de los datos de entrada, así que también buscaremos el mejor α (hiperparámetro de PCA) para intentar disminuír en la mayor medida posible la dimensión de las imágenes sin perder eficacia.

9.2. Búsqueda del mejor k

Sabemos que utilizar un k mayor a la cantidad de elementos que posee la clase con menos imágenes no es una buena idea, ya que indefectiblemente terminaremos incluyendo entre los vecinos cercanos datos de otras clases.En pocas palabras, estaremos metiendo ruido en la votación. Así que consideraremos a este valor como el máximo valor de k que, en un principio, tendría sentido tomar. Este valor es aproximadamente el total de imágenes del dataset sobre diez.

El mínimo k a considerar será 1.

Hipótesis

- Tomar $k = 1$ debería funcionar, pero puede pasar que nuestro único vecino sea un outlier y clasifiquemos mal. No creemos que podamos obtener una accuracy muy alta.
- Creemos que tomar k igual a la mitad del k máximo es razonable ya que esperamos que la clasificación no esté sesgada por elementos de clases vecinas y que podamos evitar el problema mencionado en el ítem anterior.

- Tomar el 10 % de la cantidad de imágenes de la clase con menos imágenes sigue la línea del punto anterior, pero creemos que este valor ya será suficiente para alcanzar una buena accuracy.

9.2.1. Resultados

Como primer experimento corrimos KNN con un rango muy amplio de valores para k y graficamos el valor de el accuracy en función de los mismos como se muestra en la figura 3. Esto lo hicimos para darnos una idea de los valores de k que valían la pena.

Este experimento lo corrimos con 10000 datos (8000 de training y 2000 de testing) que extrajimos de nuestro conjunto orginal de training.

Una aclaración que consideramos adecuada es que si para este experimento recalculáramos las normas para cada valor de k, el tiempo de cómputo sería muy alto como para poder considerar su realización en este trabajo. Es por esto que el procedimiento para calcular las predicciones fue modificado resultando en el siguiente:

- Calcular la distancia un punto de testing a todos los puntos de training.
- Ordenar esas distancias en orden creciente, manteniendo un seguimiento de la etiqueta a la que corresponde el dato de training asociado a cada distancia.
- Para todo $k \in \{1, 2, ..., 7999, 8000\}$, tomar las etiquetas de los k datos más cercanos, y realizar la votación para ver cual es la etiqueta "ganadora".
- Realizar este mismo procedimiento para todas las imágenes de testing.

De esta manera, el cálculo de las normas se realizará una única vez por cada dato de testing, resultando esto en un decremento notable en el tiempo de ejecución del experimento.

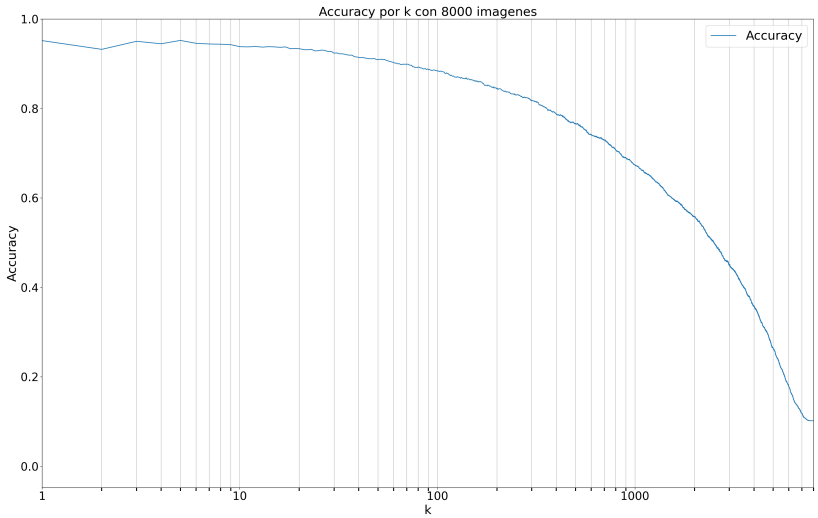


Figura 3: Accuracy obtenido para 8000 datos de entramiento y 2000 de validación según distintos k.

En este gráfico podemos observar como el accuracy es muy baja cuando k supera el valor de la cantidad de datos de la clase con menos imágenes (725), confirmando nuestra hipótesis sobre este valor. Esto tiene sentido, ya que como dijimos anteriormente, vamos a estar considerando necesariamente datos de distintas clases.

Además podemos ver que los mejores valores de accuracy se encuentran entre los primeros 100 valores de k. Es por esto, que procederemos a evaluar como se comporta el tiempo de ejecución de este modelo, con estos distintos valores de k.

Para esto vamos a realizar *KNN* con 10000 datos, utilizando 8000 datos para training y clasificando los 2000 restantes. En este caso no nos interesa el accuracy resultante, sólo queremos ver cuánto tiempo tarda el programa en ejecutarse. Los resultados de este proceso se pueden ver en la figura 4.

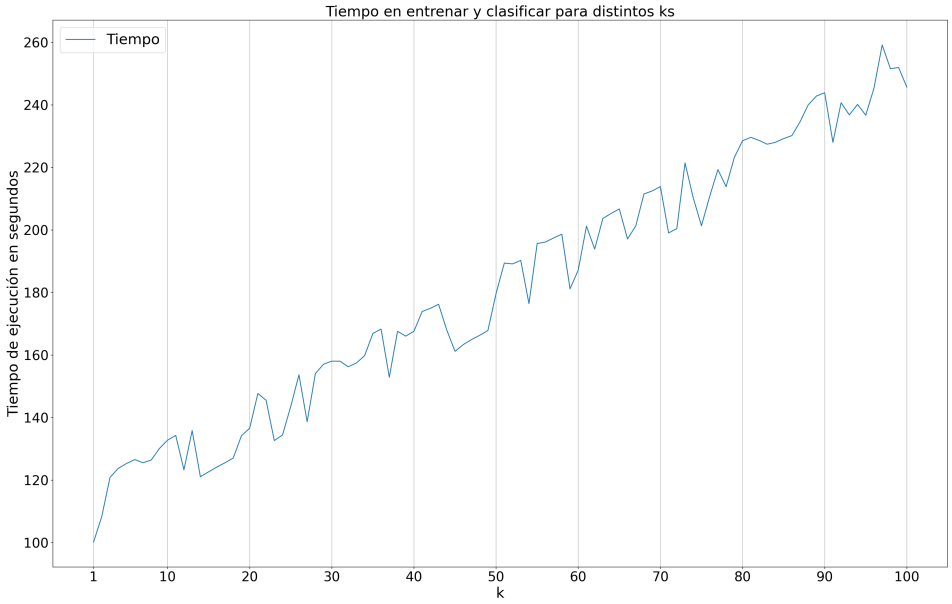


Figura 4: Tiempo de ejecución para distintos valores de k con 8000 imágenes.

Observando la figura 4, podemos ver que el tiempo fluctúa alrededor de 20 segundos de forma ruidosa.

Podemos intentar aproximar el gráfico anterior por una curva lineal con pendiente 1,4 y ordenada 110 como se puede apreciar en la figura 5. A primera vista, esta recta pareciera aproximar de manera bastante correcta las mediciones originales, así que podemos decir con relativa seguridad que el tiempo de cómputo crece de manera lineal respecto al valor de k elegido. Consideramos que un análisis más profundo de esta aproximación no entra en el alcance del trabájo práctico.

Por otra parte, creemos que el ruido en las mediciones se debe a procesos del sistema operativo a lo largo de la ejecución del programa, cuya manipulación no está a nuestro alcance.

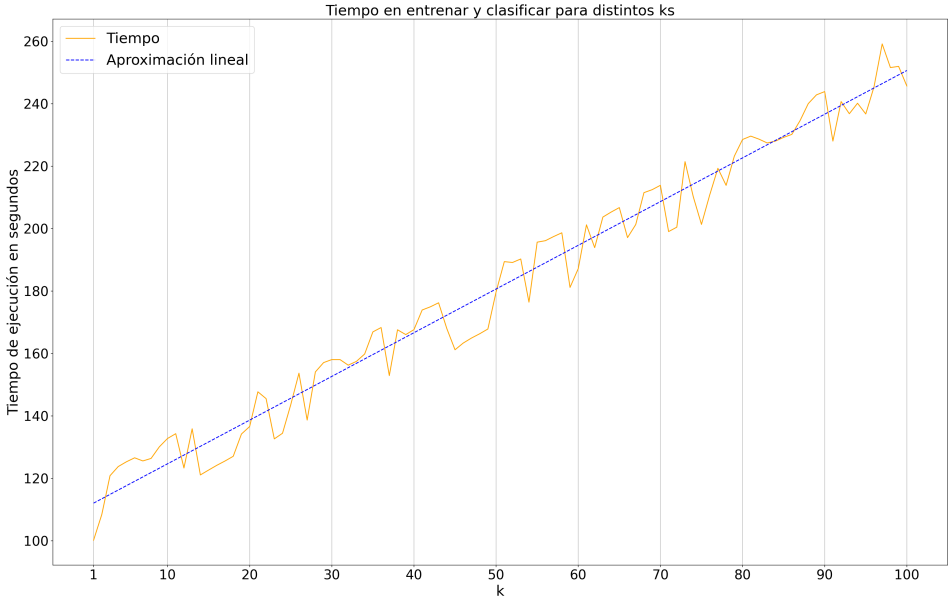
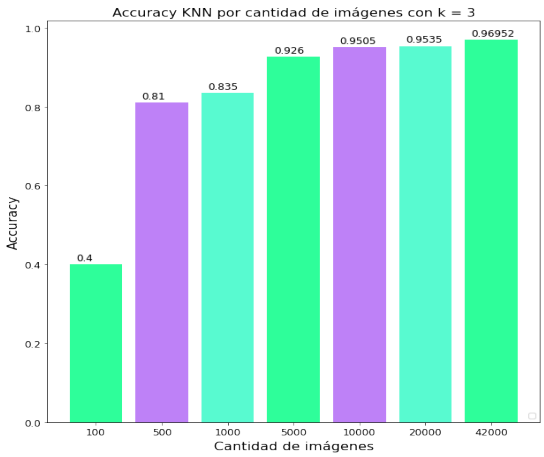
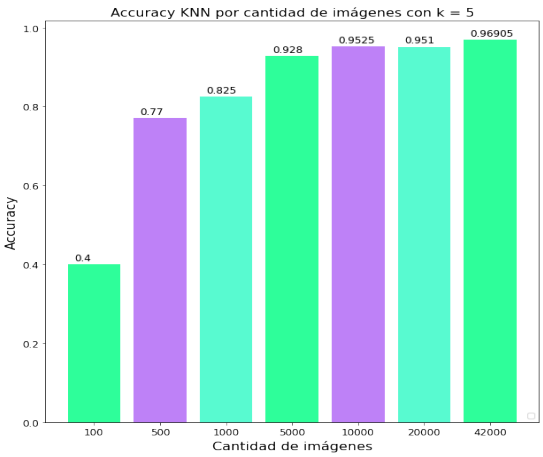


Figura 5: Tiempo de ejecución para distintos valores de k con 8000 imágenes junto con su aproximación lineal.

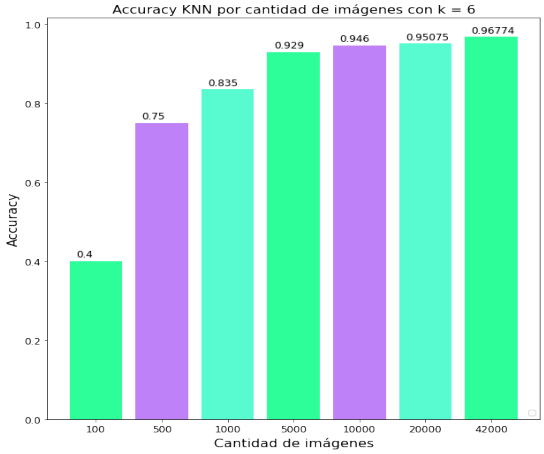
Luego de este paréntesis para analizar el rendimiento temporal de *KNN*, veamos qué accuracy obtenemos con valores pequeños de k, los cuales obtuvieron muy buenos resultados en el primer experimento (figura 3), tomando un número variado de imágenes. Esto se puede ver en la figura 6.



(a) Accuracy obtenido para cantidades distintas de imágenes con $k = 3$.



(b) Accuracy obtenido para cantidades distintas de imágenes con $k = 5$.

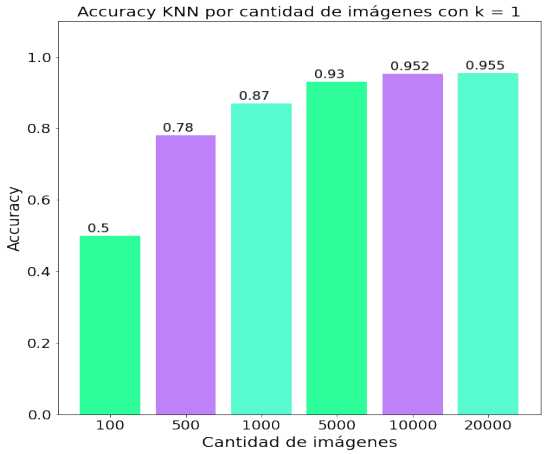


(c) Accuracy obtenido para cantidades distintas de imágenes con $k = 6$.

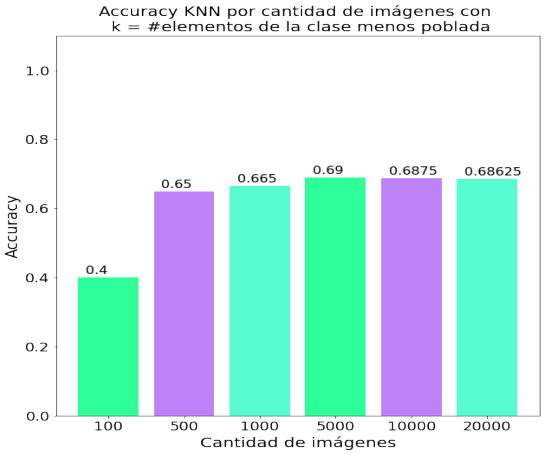
Figura 6: Accuracy para distintos valores de k pequeños utilizando datasets de tamaños distintos (las cantidades en los ejes x corresponden al tamaño antes de tomar un 20 % para validación).

Como era de esperar, a mayor número de imágenes, mayor accuracy. Igualmente, a medida que el número de imágenes se hace mayor, el porcentaje del incremento en el accuracy se reduce. Por ejemplo, en la figura 6c podemos ver que el incremento en el accuracy al pasar de 500 a 1000 imágenes es similar al que se produce al pasar de 1000 a 5000. Sabiendo esto, al analizar el rendimiento del clasificador entrenado con más de 10000 imágenes notamos que la mejora en el accuracy es muy pequeña como para que justifique realizar todos los experimentos sobre el dataset completo teniendo en cuenta el excesivo tiempo de ejecución que implica.

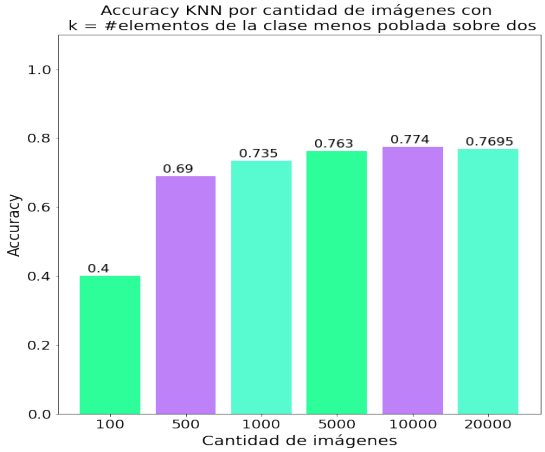
Veamos qué pasa con los valores de k que consideramos en nuestras hipótesis:



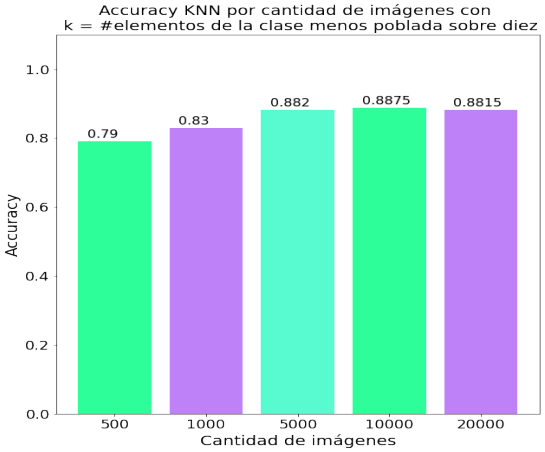
(a) Accuracy obtenido para cantidades distintas de imágenes con $k = 1$



(b) Accuracy obtenido para cantidades distintas de imágenes con el máximo valor de k que consideramos (cantidad de elementos de la clase 5).



(c) Accuracy obtenido para cantidades distintas de imágenes con el máximo valor de k sobre dos.



(d) Accuracy obtenido para cantidades distintas de imágenes con el máximo valor de k sobre diez.

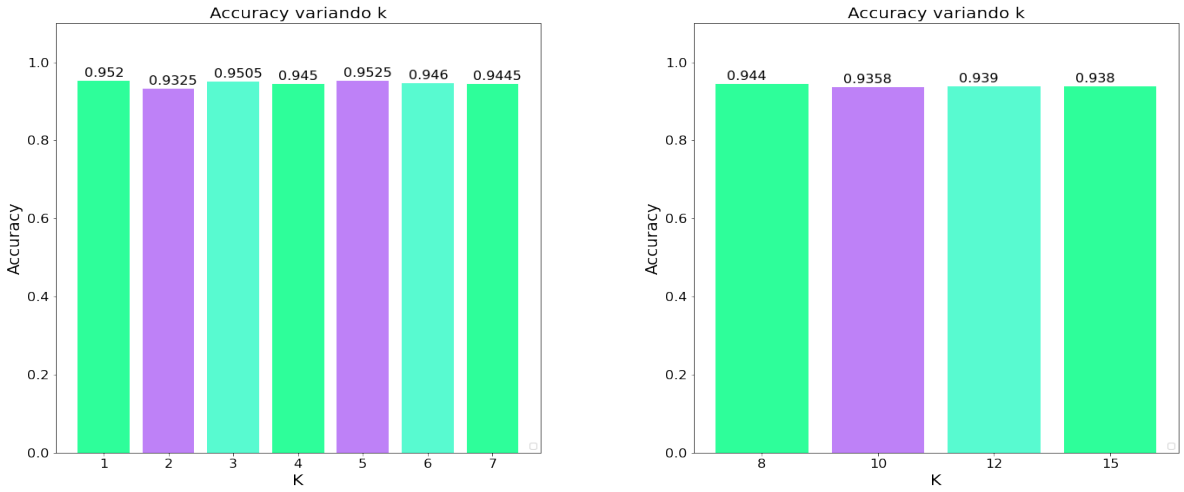
Figura 7: Accuracy para distintos valores de k utilizando datasets de tamaños distintos (las cantidades en los ejes x corresponden al tamaño antes de tomar un 20 % para validación).

Los gráficos de la figura 7 confirman que utilizar un k grande no funciona nada bien. Además, muestran claramente que cuantos más datos tengamos, mejor es nuestra clasificación, lo cual es esperable debido a que hay más información. Por otro lado, teniendo en cuenta la figura 3, podemos afirmar con bastante seguridad que el valor óptimo de k es chico.

Notemos que en la figura 7d la cantidad de imágenes comienza desde 500. Esto es así ya que para 100 imágenes, el k a utilizar era 0 (la clase con menos datos tiene 6 elementos, y dividido diez queda en 0).

Otro dato a destacar es el que el tiempo de ejecución aumentó considerablemente al trabajar con valores de k mayores. Por ejemplo, correr KNN con $k = 1$ nos llevó 26 minutos, mientras que con el mayor k (cantidad de datos de la clase con menos imágenes) 4 horas. De todas formas, esto no fue una sorpresa, sabíamos que el tiempo de ejecución aumentaba al aumentar k por el experimento de la figura 5.

Luego de haber obtenido estos resultados, procederemos a estudiar con más profundidad el comportamiento de nuestro clasificador con valores más pequeños de k.



(a) Accuracy obtenido para distintos valores de k con 8000 datos de training y 2000 de validación.

(b) Accuracy obtenido para distintos valores de k con 8000 datos de training y 2000 de validación.

Figura 8

Como vemos en la imagen 8, el accuracy obtenida es muy similar y alcanza valores bastante altos. Creemos que $k = 1$ obtuvo tan buena accuracy por la poca presencia de outliers que posee el dataset. Sin embargo, sabiendo que tomar un k muy pequeño podría generar overfitting y viendo la información que aporta la figura 4, creemos correcto utilizar como valor óptimo para este parámetro $k \in \{3, 5, 6\}$.

Estos valores de k serán los que tendremos en cuenta cuando corramos Cross Validation.

9.2.2. Conclusiones

- Tomar $k = 1$ da una buena accuracy, pero creemos que este es un caso muy particular ya que el dataset MNIST cuenta con un número muy elevado de muestras de buena calidad y clases equilibradas. Consideramos que si estuviéramos trabajando con un set de datos más general que no contara con esas características, tendríamos más overfitting y sería necesario tomar un k mayor para obtener una mejor estabilidad.
- Los valores de k genéricos que utilizamos no funcionaron bien porque resultaron ser muy altos. Creemos que la mejor manera de hallar un valor óptimo para este hiperparámetro es mediante un gráfico similar al de la figura 3 para un rango amplio de k_s. De igual manera, el rango a tener en cuenta dependerá del dataset con el que se esté trabajando.
- Comprobamos que cualquiera de los valores de k del gráfico 8a funciona bien. Sin embargo, consideramos que tomar los valores más pequeños no es una buena idea, ya que incrementaría la probabilidad de que se genere overfitting.

9.3. Búsqueda de α

Ahora vamos a analizar el rendimiento de KNN utilizando PCA para reducir la dimensionalidad de los datos. Usaremos un k fijo e iremos variando el hiperprámetro α para hallar su valor óptimo.

Creemos que un α muy chico no será de utilidad ya que comprimiria demasiado los datos dejando afuera información relevante. Además, creemos que mientras más grande sea el α, mejor será el accuracy, a causa de que cada vez estaremos perdiendo menos información relevante. Sin embargo, sospechamos que llegará un punto en el que el aumento del accuracy en proporción al aumento de α será lo suficientemente menor como que no valga la pena seguir incrementando el tiempo de cómputo que requiere PCA.

Por otra parte, como el objetivo final es utilizar KNN, el cual se ve afectado por la *maldición de la dimensionalidad*, consideramos que este algoritmo debería funcionar mejor luego de reducir la dimensión de los datos. Por esta misma razón, buscaremos usar un α chico que nos permita obtener un buen accuracy.

Hipótesis

- El valor del accuracy en función del α debería en un principio subir abruptamente, para luego estancarse.
- El accuracy luego de combinar *PCA* con *KNN* será mayor al que se obtiene utilizando solo *KNN*.

9.3.1. Resultados

En este caso, repetiremos el primer experimento de la sección anterior, pero ahora en función del hiperparámetro α, manteniendo fijo el valor k en 5, ya que este mismo obtuvo muy buenos resultados. Con esto esperamos ver cuáles son los mejores α_s.

Como dijimos en la hipótesis, esperamos que en algún α determinado el crecimiento del accuracy se estanque. Comenzaremos tomando un valor arbitrario de α, en este caso 100, y analizaremos cuánta información aportan estas componentes. Si logramos ver que el aporte de las mismas es muy bajo, entonces podemos decir que es correcto hacer el análisis posterior sobre estas componentes, ya que el tomar más de las mismas no nos va a dar una mejora significativa en el rendimiento.

Como calcular autovalores es un procedimiento computacionalmente costoso, decidimos que en los experimentos que requieran correr *PCA* con valores distintos de α no buscaremos los autovalores en cada pasada, si no que calcularemos tantos autovalores como el máximo α necesario para el experimento, y luego haremos slicing sobre las columnas de la matriz resultante. De esta forma estaremos computando los autovalores una única vez.

Para empezar a buscar los α_s relevantes, correremos *PCA* con valores de α entre 1 y 100, ya que creemos será una reducción de datos que no pierda mucha información relevante de las imágenes originales. Con esto, podremos graficar el porcentaje de varianza aportado por cada componente para las primeras 100 componentes.

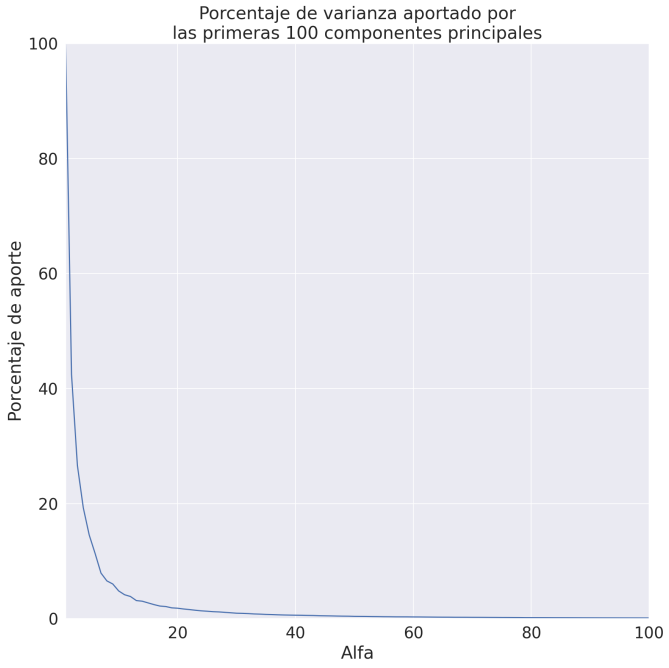


Figura 9: Porcentaje de varianza aportado por las primeros 100 componentes principales.

Podemos notar en la figura 9 que el porcentaje de varianza aportado por las últimas 50 componentes es prácticamente cero. Confirmemos este resultado graficando el accuracy arrojada por KNN luego de reducir los datos con PCA para $k = 5$ con los mismos valores de α. Esto se puede ver en la figura 10.

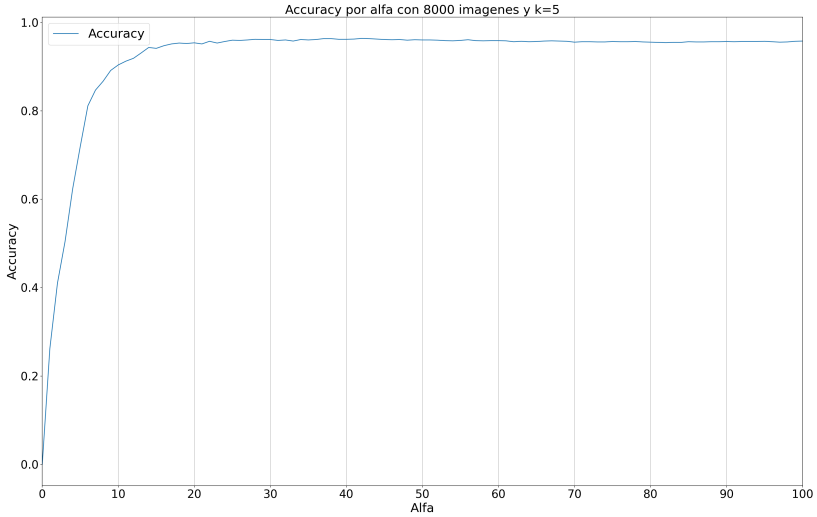


Figura 10: Accuracy con $k = 4$ segun el valor de α con 8000 imágenes de training y 2000 de validación.

Podemos ver, que sucede algo que se condice mucho con lo dicho en la hipótesis, ya que podemos ver un gran crecimiento de el accuracy para los primeros 15 valores de α. Pero a partir de allí, este valor se estabiliza.

Como las accuracy's obtenidas para los distintos α_s se estabilizan rápidamente, veremos qué pasa con los tiempos de ejecución para elegir el valor óptimo de este hiperparámetro.

En esta ocasión, correremos $PCA+KNN$ con distintos valores de α en un rango entre 13 y 25 fijando el valor de k en 5. Lo hacemos para este reducido rango ya que podemos ver que el α óptimo (teniendo el cuenta el trade-off entre tiempo de cómputo y tasa de efectividad obtenida) se encuentra allí y además estos experimentos son computacionalmente muy costosos con lo que no consideramos que valga la pena extender el rango. Nuevamente lo haremos usando 8000 imágenes de entrenamiento y clasificaremos otras 2000. En cada ejecución registraremos el tiempo que toma el programa en terminar. Los resultados de esto, se pueden ver en la figura 11.

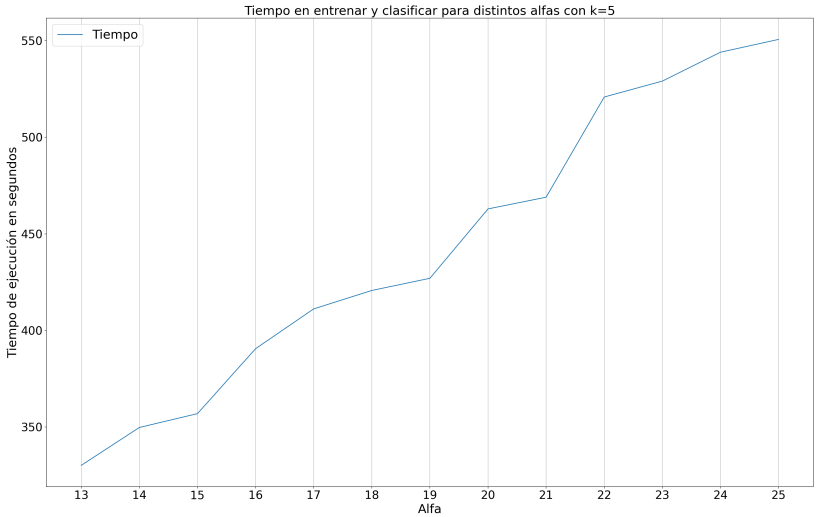


Figura 11: Tiempo de ejecución para distintos valores de α con $k = 5$ usando 8000 datos de training y 2000 de validación.

Volveremos a aproximar estos datos por una curva lineal con pendiente 19,4 y ordenada 74,6 como se puede ver en la figura 12.

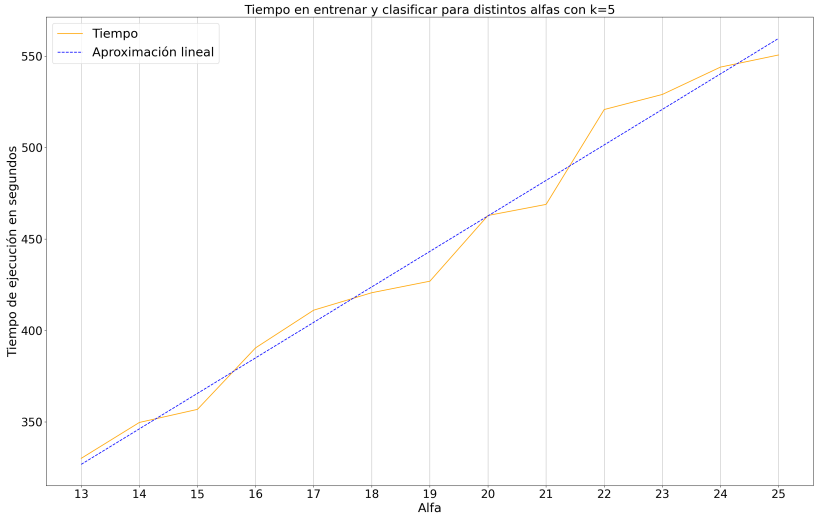


Figura 12: Tiempo de ejecución para distintos valores de α con $k = 5$ usando 8000 imágenes junto con su aproximación lineal.

Dado que la aproximación resultante parece ser bastante precisa y que ya cubrimos los α_s de interés para experimentos posteriores, reafirmamos que no será necesario realizar este experimento para valores mayores dado que el tiempo de cálculo de los mismos es muy elevado y no esperamos obtener resultados anómalos.

Mirando las figuras 10 y 11 nos damos cuenta de que el accuracy no presenta mejoras relevantes con $\alpha > 30$. Sin embargo, el tiempo de ejecución empeora considerablemente al tomar valores más grandes. Por estas razones, para los próximos experimentos consideraremos α_s que estén entre 13 y 25.

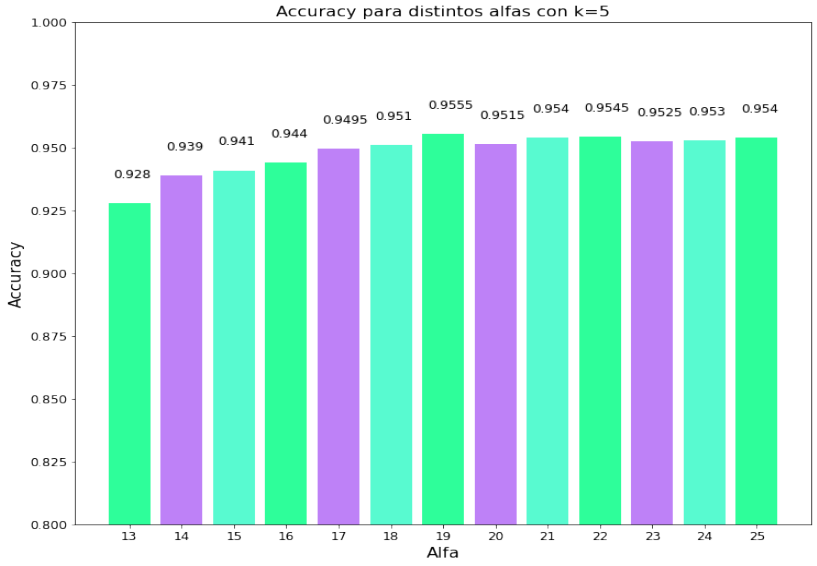


Figura 13: Accuracy para distintos valores de α con $k = 5$ usando 8000 imágenes de training y 2000 de testing

.

En la figura 13 vemos que el accuracy varía relativamente poco. Los mejores valores de α parecerían ser 19, 25 y 22. Sin embargo, como vimos anteriormente, usar un α grande implica tener un tiempo de cómputo más alto que no consideramos sea necesario frente al pequeño incremento observado en el accuracy.

Intentaremos ver si podemos ahorrar cómputo tomando un α menor a 19 sin perder un número signifitivo de accuracy. Para esto, correremos cross validation con $\alpha \in \{17, 18, 19\}$ para confirmar estas hipótesis.

9.3.2. Conclusiones

- Las primeras componentes principales concentran la mayor cantidad de varianza. En nuestro caso, alcanza con tomar un α chico con respecto a la cantidad de pixeles de los datos originales (estamos hablando de una dimensión

final menor a 30 frente a las imágenes de 784 dimensiones que teníamos en un principio).

- No encontramos mayores diferencias entre el accuracy calculado con KNN con PCA y sin PCA, sin embargo no estamos seguros de por qué se da este suceso ya que esperabamos un incremento en el accuracy al usar PCA debido a que estaríamos reduciendo los efectos de la *maldición de la dimensionalidad*.

9.4. Búsqueda de la mejor pareja

Para conseguir una mayor robustez en nuestros resultados, correremos K-fold Cross Validation con los mejores valores de los hiperparámetros conseguidos hasta el momento, los cuales son: $k \in \{3, 5, 6\}$ y $\alpha \in \{17, 18, 19\}$.

Hipótesis

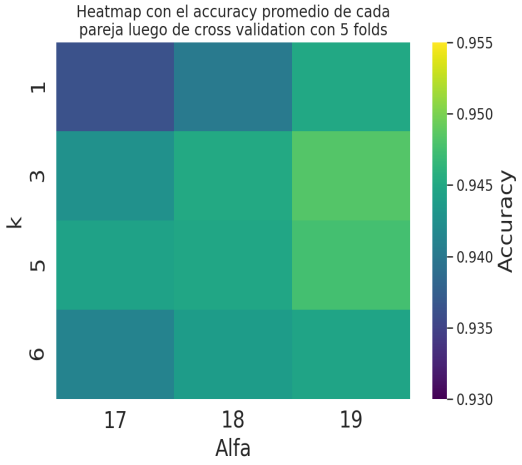
- Como MNIST es un set de datos insesgado y muy poblado, creemos que no habrá diferencia al tomar una cantidad grande de folds.
- Creemos que el tiempo de cómputo que requiere realizar el procedimiento con una cantidad amplia de folds será muy elevado.

9.4.1. Resultados

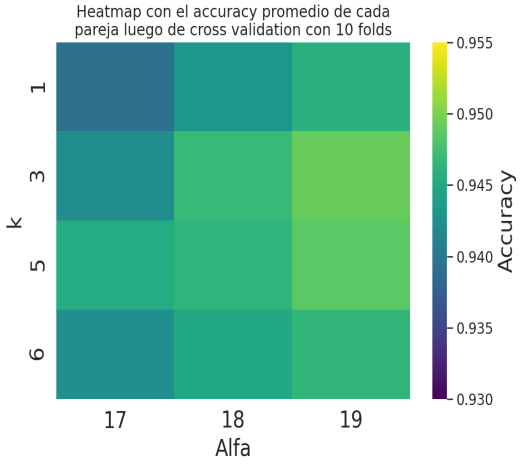
Para comprobar nuestra primer hipótesis, corrimos K-fold con $K \in \{5, 10, 25\}$ con los valores de k y α nombrados anteriomente, y agregaremos el 1 entre los k para ver si se cumple la hipótesis que tenemos acerca de que en el caso general este valor no es bueno.

La pareja ganadora para cada K se mantuvo constante en $k = 3$ y $\alpha = 19$. Las accuracy's promedio obtenidas fueron 0.9483, 0.9492, 0.9503 para los valores de K elegidos en orden creciente. El tiempo de corrida para cada K fue de 1 hora 25 minutos, 2 horas 19 minutos y 4 horas 53 minutos respectivamente.

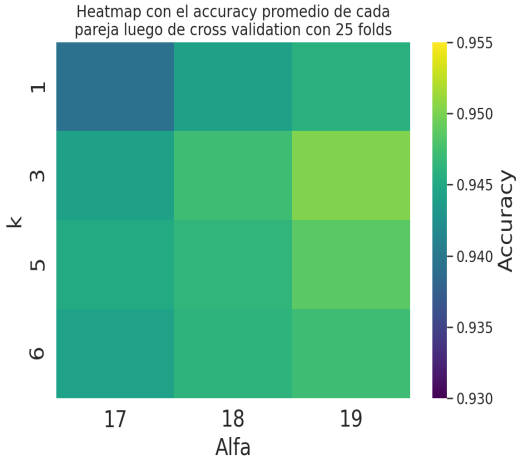
Grafiquemos las accuracy's promedio de validación para cada pareja con los tres valores de K para confirmar que no hubo mayores diferencias.



(a) Accuracy's promedio obtenidas luego de correr Cross Validation con 5 folds para 42000 datos.



(b) Accuracy's promedio obtenidas luego de correr Cross Validation con 10 folds para 42000 datos.



(c) Accuracy's promedio obtenidas luego de correr Cross Validation con 25 folds para 42000 datos.

Figura 14

Los heatmaps de la figura 14 muestran que las accuracy's promedio son casi iguales, lo que termina de confirmar que no hay diferencia entre tomar un valor de K chico frente a valores más grandes que son computacionalmente muy caros. Sospechamos que la razón de este resultado es que MNIST es insesgado y sus elementos están desordenados (en caso contrario podría pasar que, por ejemplo, en el batch de training no hayan datos de etiqueta 9, ya que se encuentran todos en el grupo de testing, lo cual sería contraproducente). Es probable que si queremos clasificar otro set de datos con más variabilidad este experimento arroje resultados muy distintos. De todas formas, esta hipótesis está por fuera de los objetivos de este trabajo.

Consideramos que tomar $K = 5$ es una buena idea ya que permite asegurar que no estamos overfitteando el modelo y no conlleva tanto tiempo de cómputo.

9.5. Resultados finales

Ahora que ya sabemos que la mejor pareja es $k = 3$ y $\alpha = 19$, entrenaremos el clasificador con 33600 imágenes y confeccionaremos la matriz de confusión del mismo al momento de clasificar las 8400 imágenes separadas para validación. La matriz de confusión vista en la figura 15 nos provee de un diagrama, en donde en muy poco espacio se condensa mucha información. Esto sumado a su fácil armado lo convierten en un gráfico ideal para evaluar como se está comportando nuestro clasificador, por ejemplo para ver qué conjunto de datos está clasificando sin problemas, o por otro lado, cuáles son los datos que más dificultad tienen para ser clasificados.

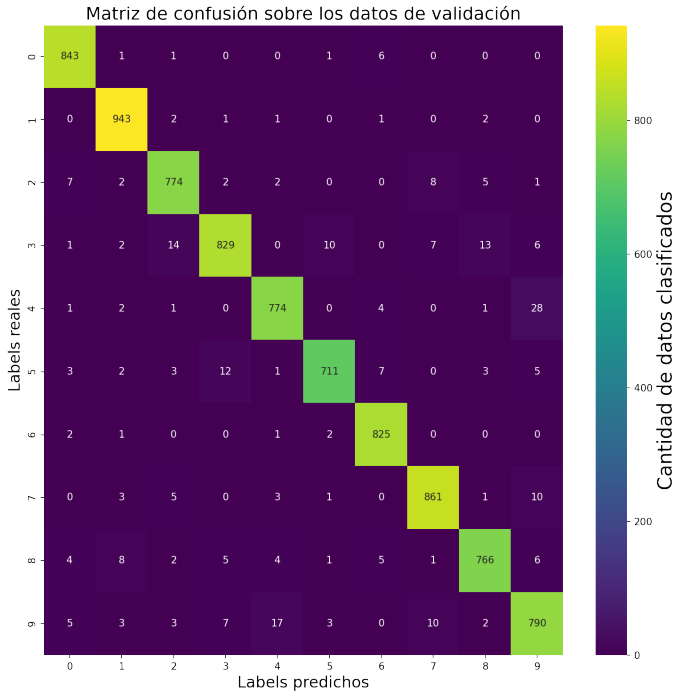
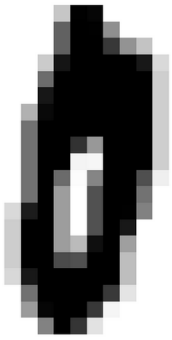


Figura 15: Matriz de confusión para el clasificador con $\alpha = 19$ y $k = 3$ armada sobre 8400 datos.

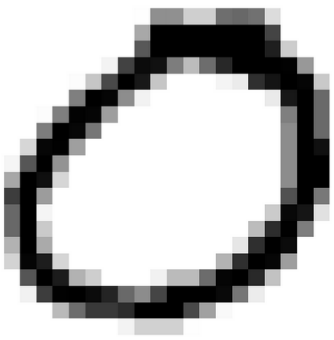
En la figura, podemos ver como la diagonal es donde se concentra la mayor cantidad de predicciones de nuestro clasificador. Esto es algo muy bueno, ya que quiere decir que el label predicho por nuestro modelo se condice con el label real de la imagen de entrada. Igualmente esto no nos aporta tanta información ya que vimos a lo largo de todo este trabajo que el accuracy de los clasificadores usados siempre se mantenian por encima del 92 %, y sumado a que las clases estaban equilibradas, era de esperarse encontrar ese porcentaje de imágenes distribuido uniformemente a lo largo de la diagonal.

Donde sí consideramos hay información útil es en donde encontramos los números más altos fuera de la diagonal. Ya que aquí es donde vemos exactamente dónde está fallando nuestro modelo.

En este caso, podemos ver que los 2 errores más comunes fueron clasificar un 4 cuando la imagen orginal era un 9 y viceversa. Esto se debe a que son números con formas muy similares. En contraposición a esto, podemos ver que las clases 0 y 1, muy diferentes en forma, fueron confundidas una única vez en todo la ejecución. Asumimos que este error se debe a que el número que fue mal clasificado no fue escrito de manera clara, como para que sea fácilmente distinguible. Igualmente podemos verificar esto viendo la imagen que fue mal clasificada. Esta, junto con una imagen de etiqueta 0 elegida al azar se encuentran en la figura 16 a modo de comparación.



(a) Imagen clasificada como un 1 cuando la etiqueta marcaba que era un 0.

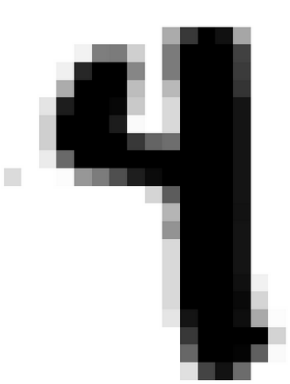


(b) Imagen tomada al azar del set cuyas etiquetas eran 0.

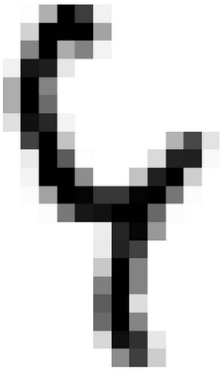
Figura 16: imágenes con etiqueta 0 utilizadas para analizar resultados.

Efectivamente, podemos notar que el 0 de la figura 16a, el mal clasificado, podría llegar a asemejarse a un 1 si no se mira con detenimiento. Mientras que el 0 tomado al azar (16b) es claramente diferenciable de un 1, por lo que sería raro que estuviera mal clasificado.

Ahora, en la figura 17 haremos un análisis similar, tomando una foto de etiqueta 9 que fue mal clasificada como un 4. Y una foto de etiqueta 4, mal clasificada como un 9.



(a) Imagen clasificada como un 9 cuando la etiqueta marcaba que era un 4.



(b) Imagen clasificada como un 4 cuando la etiqueta marcaba que era un 9.

Figura 17: Imágenes mal clasificadas utilizadas para analizar resultados.

Viendo las dos imágenes podemos darnos una idea de por qué fueron mal clasificadas. Ambas son fotos poco claras, consideramos que hacen dudar incluso a humanos sobre qué etiqueta les corresponde, incluso resultan ser más parecidas a las clases arrojadas por el clasificador que a su etiqueta original. Es por esto que no consideramos que este error sea grave.

Por último, utilizando las mismas predicciones con las que generamos la matriz de confusión de la figura 15 calcularemos tres métricas para evaluar numéricamente el rendimiento del clasificador, las mismas son:

- Precision score
- Recall score
- Accuracy score

Como dijimos previamente en la sección 8, al tratarse de clases balanceadas, esperamos ver que el resultado arrojado por las 3 métricas sea muy similar.

Comenzaremos calculando la precisión y el recall para cada clase y volcaremos los resultados en la tabla de la figura 18.

Clase	Precision Score	Recall Score
0	0.98943662	0.97344111
1	0.99263158	0.97518097
2	0.96629213	0.96149068
3	0.9399093	0.96845794
4	0.95437731	0.96388543
5	0.95180723	0.97530864
6	0.99277978	0.97287736
7	0.9739819	0.97068771
8	0.95511222	0.96595208
9	0.94047619	0.93380615

Figura 18: Precision y Recall por clase para KNN con $k = 5$ y $\alpha = 19$ con 42000 datos.

Como era de esperarse podemos ver que ambos valores son muy consistentes de a pares, teniendo una diferencia promedio de valor 0,014. Luego, podemos tomar el promedio no ponderado de las clases para estas dos métricas y compararlas con el valor del accuracy obtenido. Esto se hace en la tabla de la figura 19.

Precision Score	Recall Score	Accuracy Score
0.965680	0.966109	0.96619

Figura 19: Precision, Recall y Accuracy score para KNN con $k = 5$ y $\alpha = 19$ con 42000 datos.

Podemos ver tres valores muy similares, los cuales consideramos cumplen los requisitos de un clasificador de muy buen nivel siendo que los valores obtenidos son bastante cercanos al 100 %.

Como último paso, para terminar de verificar lo dicho en el párrafo anterior, subimos el csv con las predicciones obtenidas al entrenar nuestro clasificador con los 42000 datos que tenemos en el set de training utilizando los valores óptimos hallados para los hiperparámetros en la sección 9.4 a Kaggle. El accuracy de testing arrojado resultó ser 0,96696, lo cual se condice con todo lo expresado previamente.

10. Conclusiones generales

- Si bien los resultados obtenidos fueron muy buenos, no creemos posible extrapolar los hiperparámetos conseguidos a otros modelos de clasificación. Consideramos que estos son muy particulares debido a las características del dataset utilizado, ya que el mismo estaba muy poblado y contaba con clases equilibradas.
- Estamos al tanto de que probablemente consigamos aumentar el accuracy obtenido aumentando el α, sin embargo creemos que la ganancia obtenida en tiempo de ejecución supera a la que podríamos haber logrado en la tasa de efectividad incrementando este hiper parámetro. Igualmente, esta fue una elección y dependerá de los objetivos que se quieran alcanzar con el clasificador.
- Concluímos que si bien los resultados de los experimentos no demostraron que combinar *PCA* con *KNN* genere un mayor accuracy, seguimos creyendo que utilizar menos dimensiones será siempre preferible, ya que lo más probable es que encontremos en ellas información redundante que puede ser eliminada. Además, estaríamos obteniendo así tiempos considerablemente más bajos.

11. Aclaraciones

11.1. Por qué no usamos todo el rango de imágenes

A lo largo del trabajo hicimos una serie de experimentos, los cuales requirieron el uso de datasets de distintos tamaños. Para la mayoría de los experimentos usamos 10000 imágenes, es decir, aproximadamente un cuarto de las que teníamos disponibles. Tomamos esta decisión debido a que experimentar con tantas imágenes requería tiempo de cómputo que no podíamos costear y que tampoco consideramos necesario, como vimos en los primeros experimentos.

El por qué de la elección de este número determinado de imágenes puede verse en los gráficos de la figura 6, que muestran que para valores fijos de k (notar que graficamos aquellos que parecían más prometedores viendo la figura 3) el accuracy obtenido para datasets con más de 10000 imágenes es muy similar.

12. Fuentes

Teoría y gráfico de cross validation