DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Reconocimiento de Dígitos

Trabajo Práctico 2

3 de agosto, 2020 — Métodos Numéricos

Grupo 4

Integrante	LU	Correo electrónico
López Menardi, Justo	374/17	juslopezm@gmail.com
Strobl, Matías	645/18	matias.strobl@gmail.com
Yulita, Federico	351/17	fyulita@dc.uba.ar

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

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

Resumen

Optical Character Recognition es la automática conversión de imágenes o texto impreso o escrito a formato digital. Es un área de investigación importante en la computación con mucho interés para implementar en nuevas tecnologías. El objetivo de este trabajo es desarrollar y evaluar una herramienta de OCR para la clasificación de dígitos manuscritos en imágenes. Se cuenta con la base de dígitos manuscritos provista por MNIST que será usada para entrenar al algoritmo *k Nearest Neighbors*. Este algoritmo consiste en hallar las k imágenes más parecidas a una nueva imagen fuera del conjunto de entrenamiento y predecir el valor predominante entre estos k "vecinos". Además, se usó la técnica *Principal Component Analysis* para reducir la dimensionalidad del conjunto de imágenes y minimizar el tiempo de cómputo. Se obtuvieron entonces parámetros óptimos para la utilización de estos algoritmos sobre el conjunto de datos y se logró demostrar la efectividad de ambos algoritmos, con resultados precisos y tiempos de cómputo optimizados.

Palabras Clave: Optical Character Recognition, Principal Component Analysis, k Nearest Neighbors Algorithm, Multiclass Classification

1. Introducción

Optical Character Recognition (OCR) es la automática conversión de imágenes o texto impreso o escrito a formato digital.[1] Es un área de investigación importante en la computación con mucho interés para implementar en nuevas tecnologías. El objetivo de este trabajo es desarrollar y evaluar una herramienta de OCR para el reconocimiento de dígitos manuscritos en imágenes. Se cuenta con la base de dígitos manuscritos provista por MNIST [2] que será usada para entrenar al algoritmo de clasificación desarrollado.

Este algoritmo que se usará para clasificar es kNN (*k Nearest Neighbors*), que considera a cada objeto de la base de entrenamiento como un punto en el espacio euclídeo m-dimensional para el cual se conoce a qué dígito corresponde para luego, dado un nuevo dígito, asociarle la clase de los puntos más cercanos de la base de datos. El problema de este algoritmo es que su costo temporal aumenta a medida que aumenta la cantidad de dimensiones. Por ello, se requiere un método para reducir la cantidad de las muestras y entonces trabajar con una cantidad de variables acotada, buscando que las nuevas variables tengan información representativa para clasificar los dígitos de la base.

Por ello, se implementará el método de reducción de dimensionalidad PCA (*Principal Component Analysis*), que reducirá la cantidad de dimensiones de las instancias, quedándose así con las que más información aporten y reduciendo el tiempo de cómputo de kNN.

1.1. kNN

El algoritmo kNN analiza un conjunto de puntos del espacio y determina a qué dígito corresponde.[3] Cada imagen está representada como un vector donde cada elemento es un pixel. El conjunto de puntos elegidos son aquellos que estén más cerca de la imagen a clasificar.

Un dígito es asignado a una determinada clase si esta última es la más frecuente entre los k ejemplos de entrenamiento más cercanos. Para eso usamos la distancia euclídea

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{p} (x_{ri} - x_{rj})^2}.$$

kNN entonces calcula la distancia entre el ítem a clasificar y el resto de los ítems del dataset de entrenamiento. Luego, selecciona los k elementos más cercanos, es decir, con aquellos en los que tenga menor distacia. Finalmente, realiza una votación entre los k puntos y aquellos que predominen serán los que determinen la clasificación.

Notemos la importancia del valor de k elegido, que será al final el que decida a qué clase pertenecerá el dígito, sobre todo en las fronteras entre grupos. Este algoritmo es muy costoso en tiempo debido a que utiliza todo el dataset para entrenar cada punto. Por eso se empleará el método PCA para trabajar con muestras de menos dimensiones (pixeles).

1.2. PCA

El método PCA consiste en cambiar la base del conjunto de datos de entrada a una base ortogonal donde las dimensiones individuales de los datos están descorrelacionadas.[4] Al hacerlo se pierde información del mismo, quedándose con las componentes más importantes.

El método consiste en diagonalizar la matriz de covarianza de los datos. Sea $X \in \mathbb{R}^{n \times m}$ la matriz con los datos entonces representamos a cada componente como $x_j^{(i)}$, donde j es la fila e i la columna. Sea $\mu \in \mathbb{R}^m$ un vector con el promedio de cada columna de X entonces definimos una matriz $M \in \mathbb{R}^{n \times m}$ con filas

$$m_i = \frac{(x_i - \mu)^\dagger}{\sqrt{n-1}}.$$

Entonces, la matriz de covarianza $M \in \mathbb{R}^{m \times m}$ se define como

$$C = M^\dagger M.$$

Esta matriz cumple que cada uno de sus elementos es

$$c_j^{(i)} = \frac{1}{n-1} \sum_{k=1}^{n} (x_i^{(k)} - \mu_i)(x_j^{(k)} - \mu_j),$$

que son las covarianzas de cada columna, por eso la matriz es de $m \times m$. Entonces, el método PCA busca reducir la redundancia de los datos y como la covarianza es una medida de la correlación entre los datos entonces se busca minimizar la covarianza entre los datos. Por lo tanto, vamos a buscar una base que diagonalice la covarianza. Para eso vamos a buscar los autovalores y autovectores de la matriz ya que, como C es simétrica, sabemos que podemos armar una matriz ortogonal $P \in \mathbb{R}^{m \times m}$ con los autovectores de la matriz y una matriz diagonal $\hat{C} \in \mathbb{R}^{m \times m}$ con los autovalores de la matriz que cumplan que $C = P\hat{C}P^\dagger$. A los autovalores los vamos a ordenar de mayor a menor en módulo en la matriz $\hat{C}$ y vamos a llamarlos *componentes principales*. Una vez obtenida la matriz P de cambio de base (con los autovectores ordenados por las componentes principales) vamos a trabajar con la matriz $\hat{X} = XP$ ya que estos son los datos en la base que minimiza su covarianza.

Ya que las componentes principales están ordenadas entonces la matriz $\hat{X}$ va a estar ordenada por las columnas que maximizan la varianza del conjunto de datos. Por lo tanto, no hace falta conservar todas las columnas, sólo las primeras columnas son relevantes. Esto nos permite reducir la dimensionalidad del conjunto de datos preservando las primeras α columnas. Este factor α es el que determina la calidad y la complejidad de este análisis, así que va a ser relevante para el estudio determinar un valor adecuado.

2. Desarrollo

Los algoritmos utilizados se implementaron en C++ usando la biblioteca Eigen para facilitar las operaciones con matrices y vectores. Para estudiar el conjunto de datos se utilizaron *notebooks* de Jupyter Python en el que se corría el código de C++ y se analizaban los datos con las bibliotecas NumPy, Matplotlib, Pandas y SciKit Learn. Se hicieron tres notebooks, uno para testear la implementación de los algoritmos, otro para crear un conjunto de datos para ejecutar por consola y otro con la experimentación descrita en este informe.

En la experimentación, se hallaron los valores óptimos de k y α para usar en cada uno de los algoritmos implementados: kNN y PCA respectivamente. Se utilizó K-Fold Cross Validation con el objetivo de comprobar los algoritmos implementados de una manera lo más insesgada posible. Para obtener una bondad de ajuste de los algoritmos se utilizaron cuatro distintas métricas: Exactitud, Kappa de Cohen, Recall y Puntaje F1.

2.1. K-Fold Cross Validation

A la hora de utilizar los datos para entrenar y experimentar, primero se definió qué cantidad de datos se iban a usar del total de la base y luego se implementó el método de K-Fold Cross Validation.

Es importante remarcar que si uno simplemente entrena a la máquina con un conjunto de datos y luego testea con otro, puede que llegue a desarrollar un clasificador sesgado. Debido a que el clasificador kNN no devuelve error o accuracy del entrenamiento, no se puede utilizar K-Fold para detectar si el algoritmo está haciendo overfit de forma tan directa. Cuando uno se encuentra con una base de datos acotada, hay

muchas chances de que la máquina se entrene con un sesgo muy alto si simplemente particionamos los datos en entrenamiento y testeo. Puede haber información clave que no aparezca en el entrenamiento y que la máquina luego no logre clasificar de forma adecuada. Por esto mismo se utilizara el metodo K-Fold para re-samplear los datos y de esta forma buscar que el clasificador evite este problema.

K-Fold Cross Validation es un técnica empleada para analizar resultados estadísticos sobre un conjunto de datos conocidos y ver cómo se generalizarán a un conjunto de datos desconocido.[5] Esta técnica consiste en separar un conjunto de datos X y su conjunto asociado de valores a predecir y en cuatro subconjuntos X_{train}, y_{train}, X_{test} y y_{exp}. Los conjuntos X_{train} y y_{train} se utilizan para entrenar al algoritmo y el conjunto X_{test} se utiliza para formar predicciones sobre el conjunto. Luego, estas predicciones se validan bajo cierta métrica usando el conjunto y_{exp} que tiene los valores asociados a X_{test}. En el caso de K-Fold lo que se hace es separar el conjunto entero de datos en K subconjuntos de igual tamaño, $K-1$ para entrenamiento y uno para testeo. Esto se hace K veces para que cada uno de los K subconjuntos se use $K-1$ veces para entrenamiento y una vez para testeo.[6] Este proceso permite obtener un resultado más robusto y menos sesgado.

2.2. Métricas

Consideremos tres conjuntos ordenados, uno $\mathcal{D}$ con las imágenes, uno $\mathcal{L}$ con los dígitos asociados a cada una de las imágenes (o *labels*) y otro $\mathcal{P}$ con las predicciones sobre las imágenes hechas mediante alguno de los algoritmos descritos. Si la predicción fuera completamente correcta entonces $\mathcal{L} = \mathcal{P}$. Entonces, a través de las métricas mencionadas se obtienen distintas medidas de la calidad de la predicción (o *bondad de ajuste*).

La exactitud, o *accuracy*, es una medida del porcentaje de aciertos de un conjunto de predicciones.[7] Sean T la cantidad de predicciones acertadas y F la cantidad de predicciones desacertadas entonces la exactitud es

$$Accuracy = \frac{T}{T+F}. \tag{1}$$

Notemos que debido a su definición la exactitud es una cantidad entre 0 y 1, donde mientras más alta es mayor es la proporción de predicciones acertadas. A la exactitud también se la llama probabilidad empírica, ya que es un estimador de la probabilidad de que una predicción sea correcta.

La Kappa de Cohen es una métrica que calcula qué tan de acuerdo están dos conjuntos de datos en la medición.[8] Sea P_o la probabilidad empírica de que la predicción sea correcta y sea P_c la probabilidad de que una predicción al azar sea correcta entonces se define la Kappa de Cohen como

$$\kappa = \frac{P_o - P_c}{1 - P_c}. \tag{2}$$

Notemos que, al igual que la exactitud, esta métrica es una cantidad entre 0 y 1, donde mientras más alta es mejor es la predicción. La probabilidad P_c se estima usando

$$P_c = \frac{1}{N^2} \sum_{k=1}^{C} n_{1k} n_{2k},$$

donde N es la cantidad total de predicciones, C es la cantidad de categorías distintas (en este caso son 10, una para cada dígito) y n_{ik} es la cantidad de veces que la categoría k figura en el conjunto i. Debido a que no todos los dígitos están uniformemente representados en el conjunto de imágenes esta cantidad es ligeramente distinta a 0,1.

El Recall es una métrica que se usa en clasificaciones binarias y mide la cantidad de predicciones acertadas sobre el conjunto de datos de cierta clase.[9] Consideremos una clase del conjunto y consideremos todas las predicciones relevantes a esta clase. Llamamos TP (*True Positive*) a las predicciones acertadas sobre esta clase que predicen que cierta imagen pertenece a la clase (por ejemplo, para la clase 7 si el algoritmo predice que un número 7 pertenece a la clase entonces es un true positive); TN (*True Negative*) a las predicciones acertadas sobre esta clase que predicen que cierta imagen no pertenece a la clase (por ejemplo, si el algoritmo predice que un número 8 no pertenece a la clase entonces es un true negative); FP (*False Positive*) a las predicciones desacertadas sobre esta clase que predicen que cierta imagen pertenece a la clase (por ejemplo, si el algoritmo predice que un número 9 pertenece a la clase es un false positive); y FN (*False Negative*) a las predicciones desacertadas sobre esta clase que predicen que cierta imagen no pertenece a la clase (por ejemplo, si el algoritmo predice que un número 7 no pertenece a la clase es un false negative). Entonces, el Recall se define como

$$Recall = \frac{TP}{TP+FN}. \tag{3}$$

Notemos que el recall sólo se fija en el conjunto de datos de condición positiva. En los ejemplos mencionados, estas serían las imágenes que representen el dígito 7. En este trabajo, ya que hay múltiples clases se calculó el recall para cada métrica y luego se promedió. Al igual que las otras métricas, el recall es una cantidad entre 0 y 1 que mientras mayor es mejor es la predicción.

Otra métrica relevante que no se usó directamente en este trabajo es la precisión. La precisión es una métrica que se usa en clasificaciones binarias y mide la cantidad de predicciones acertadas sobre el conjunto de predicciones positivas.[9] Usando las definiciones anteriores, la precisión se define como

$$Precision = \frac{TP}{TP + FP}$$

Esta métrica no se usó directamente en el trabajo pero se usó otra que depende de esta. Esta otra métrica es el Puntaje F1.

El Puntaje F1 es una métrica que actúa sobre clasificaciones binarias y mide el promedio harmónico entre el recall y la precisión.[10] Se define como

$$F1 = 2\frac{Recall \cdot Precision}{Recall + Precision} \tag{4}$$

Al igual que las otras métricas esta es una cantidad entre 0 y 1 que mientras mayor sea mejor es la predicción. Además, como esta métrica se usa sobre clasificaciones binarias se calculó este puntaje para cada clase y luego se promedió.

3. Resultados y Discusión

En esta sección se observarán los resultados y análisis de la experimentación que se llevó a cabo con ambos métodos: kNN sin PCA y kNN con PCA. Sin embargo, es necesario dejar en claro el procedimiento y los métodos que se utilizaron para poder experimentar. Se estudiaron los tiempos de cómputo de cada algoritmo y los resultados de las métricas descriptas mediante las ecuaciones (1) para exactitud, (2) para la kappa de Cohen, (4) para el puntaje F1 y (3) para el recall.

La experimentación se separó en dos grandes ramas, una para cada método. Por un lado, se experimentó con el algoritmo de kNN sin PCA, que a primera instancia debería tener un procedimiento más costoso en términos de memoria y tiempo, y además tiene un parámetro menos para variar en comparación al otro método. Por otro lado, tenemos kNN con PCA, que se trata de la misma forma que el anterior solo que además se le puede modificar un parámetro α pertinente a PCA cuando se experimenta con él.

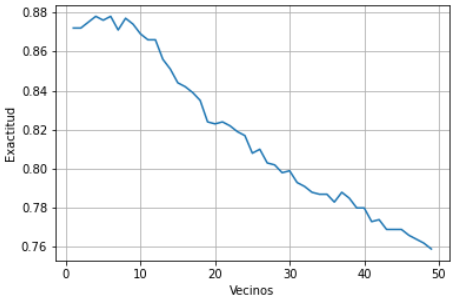
3.1. Estudio de k

La primera experimentación que se decidió llevar adelante es aquella que se centraba en analizar como el coeficiente k afecta los resultados. Se tomó $K = 10$ y se llevó a cabo sin transformar los datos con el método PCA. Se tenía la hipótesis de antemano que este valor no podría ser demasiado grande debido a que se estarían agregando instancias alejadas del conjunto de vecinos, pero tampoco debía contener un valor muy bajo ya que podría suceder que el conjunto de vecinos esté compuesto por aquellos que no sean ciertamente representativos.

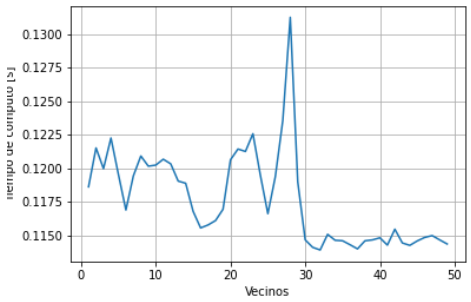
En la **Figura 1a** se puede ver un gráfico de la exactitud en función de los valores de k estudiados. Notemos que al principio la exactitud incrementa hasta un máximo entre $k = 1$ y $k = 10$ y luego decrece a medida que k incrementa. Se corrieron estos valores varias veces y se definió que el máximo de esta función está en $k = 3$. El resto de las métricas estudiadas se comportaban similarmente.

En la **Figura 1b** se puede ver un gráfico del tiempo de cómputo de kNN en función de los valores de k estudiados. Se puede ver también que el tiempo de cómputo se mantiene ruidoso pero constante, entre 0.13 s y 0.115 s. Esto es esperable debido a que cuando la cantidad de vecinos aumenta, solo se debe estar tomando la distancia entre vecinos, que no es de alta complejidad de cómputo. Es posible que si el gráfico se hiciera para un mayor rango de valores de k entonces la complejidad de cómputo de la distancia euclidiana entre varios vecinos se haga evidente.

Con este análisis se puede concluir que el valor óptimo es $k = 3$, ya que maximiza las métricas sin sacrificar tiempo de cómputo.

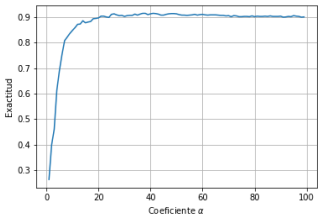


(a) Exactitud del ajuste en función de k.

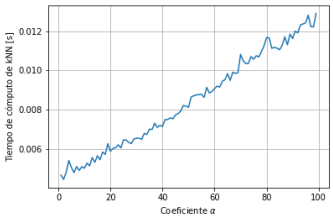


(b) Tiempo de cómputo de kNN en función de k.

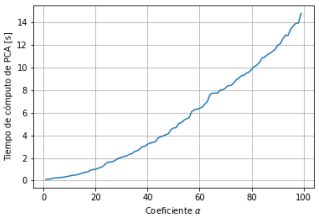
Figura 1: Gráficos de exactitud y tiempo de cómputo de kNN sin PCA en función del factor k.



(a) Exactitud del ajuste en función de α.



(b) Tiempo de cómputo de kNN en función de α.



(c) Tiempo de cómputo de PCA en función de α.

Figura 2: Gráficos de exactitud y tiempo de cómputo de kNN y PCA en función del factor α.

3.2. Estudio de α

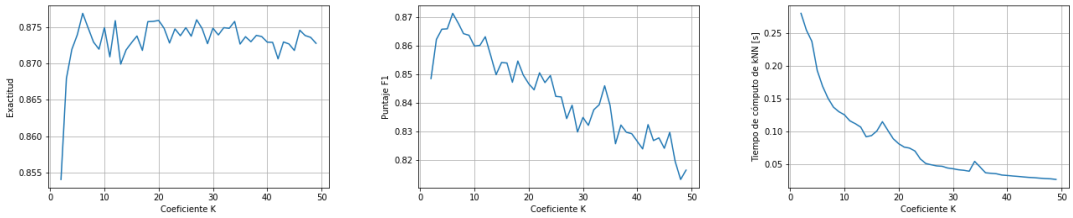
Esta vez utilizando el algortimo kNN con PCA, se analizó cómo los resultados se ven afectados por el coeficiente α. Los valores iniciales que se tomaron fueron $k = 3$ y $K = 10$. Se tiene la hipótesis de que α no podría tratarse de un valor menor a 10 debido a que de esa manera se estaría desertando información relevante.

En la **Figura 2a** puede verse un gráfico de la exactitud del ajuste en función de los valores de α estudiados. Se puede ver claramente que a partir de $\alpha = 20$, la exactitud se mantiene considerablemente constante. Se puede ver también, que el coeficiente alcanza un punto maximo entre los valores de 30 y 40 aproximadamente pero también se ve que ese punto máximo no dista demasiado de los valores que se encuentran entre 20 y 80. Las otras métricas estudiadas presentaban gráficos similares y por eso no nos parece relevante mostrarlos.

En la **Figura 2b** puede verse un gráfico del tiempo de cómputo de kNN en función de los valores de α estudiados. Nótese cómo el tiempo de cómputo incrementa linealmente a medida que α incrementa. Esto se debe a que cuando α es mayor la matriz que se usa en K-Fold es más grande y entonces kNN tiene que ajustar más dimensiones.

En la **Figura 2c** puede verse un gráfico del tiempo de cómputo de PCA en función de los valores de α estudiados. También podemos notar cómo claramente a medida que se incrementa el valor de α, el tiempo de cómputo de PCA incrementa. Esto es algo esperable debido a que se calculan más autovalores. Además, como el tiempo de cómputo de los primeros autovalores es menor que el tiempo de cómputo de los últimos, el gráfico incrementa más rápidamente.

Teniendo en cuenta ambos el tiempo de cómputo y la exactitud, hay que encontrar un balance óptimo entre los dos para poder experimentar de forma eficiente. Es por eso que se decidió tomar $\alpha = 20$ como el valor óptimo.



(a) Exactitud del ajuste en función de K.

(b) Puntaje F1 del ajuste en función de K.

(c) Tiempo de cómputo de kNN en función de K.

Figura 3: Gráficos de exactitud, puntaje F1 y tiempo de cómputo de kNN sin PCA en función del coeficiente K.

3.3. Estudio de K

El párametro K fue estudiado para ver cómo la cantidad de particiones consideradas para el cross-validation afecta los resultados. La hipótesis inicial que se tiene es que el valor óptimo no debe ser tan grande como el tamaño mismo del dataset, pero tampoco demasiado pequeño. Si fuera muy grande (que se acerque al tamañ o del dataset) entonces nos encontraríamos con un conjunto de entrenamiento muy pequeño, y si fuera muy chico entonces no estariamos logrando variar lo suficiente los datos de entrenamiento y testeo con tan pocas iteraciones. La realidad es que no hay un valor estándar de K para cualquier base de datos, este valor esta fuertemente relacionado con el tamaño de los datos y con el estudio que se esté llevando a cabo. Por eso mismo decidimos probar con distintos valores de K y analizar la situación en la que nos encontremos. Se tomó $k = 3$.

En la **Figura 3a** se puede ver un gráfico de la exactitud del ajuste en función de los valores de K estudiados y en la **Figura 3b** puede verse un gráfico del puntaje F1 en función de los valores de K estudiados. Nótese que la exactitud llega a un punto máximo cerca de $K = 10$ y luego se mantiene constante para mayores valores. Sin embargo, el puntaje F1 también alcanza un máximo cerca de $K = 10$ pero a diferencia de la exactitud luego decrece a medida que los valores de K incrementan. Los gráficos de el coeficiente de Cohen y de el recall presentaban formas similares.

En la **Figura 3c** se puede ver un gráfico del tiempo de cómputo de kNN en función de los valores de K estudiados. Nótese que el tiempo de cómputo decrece a mayores valores de K. Esto se debe a que a medida que hay más particiones habrá que realizar menos ajustes. Sin embargo, es evidente que el tiempo de cómputo total incrementará cuando K sea mayor debido a que habrá más rondas.

Con este análisis se puede concluir que el valor óptimo es $K = 10$, ya que maximiza las métricas sin sacrificar tiempo de cómputo.

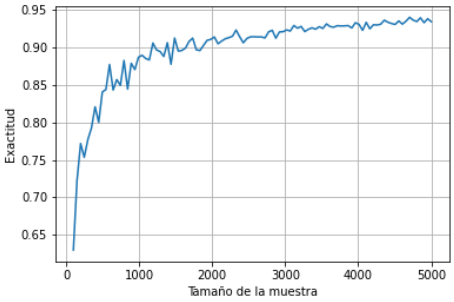
3.4. Estudio del tamaño de la muestra

Uno de los objetivos del trabajo consiste en analizar cómo afecta la cantidad de muestras que se tiene para identificar en qué situación se comporta mejor cada uno de los métodos. Para poder obtener observaciones y resultados más precisos y rigurosos, se decidió partir la experimentación, de modo que primero se evaluará el comportamiento de el método kNN sin PCA y luego incorporando este último algoritmo. Los valores que se tomaron fueron $k = 3$, $K = 10$, y en el caso que se incora al metodo PCA, $\alpha = 20$.

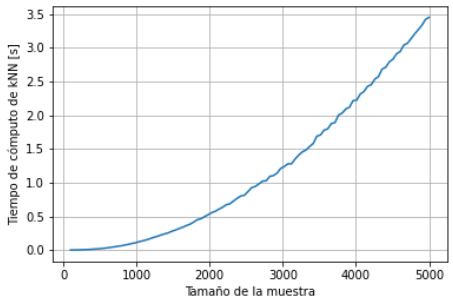
3.4.1. Estudio del tamaño de la muestra sin PCA

Lo primero y lo más importante que se quiere analizar es la exactitud. Se tiene como hipótesis que la exactitud incrementará a medida que aumente la cantidad de muestras. Teniendo un método que pueda ajustar datos y lograr generalizar lo suficiente a la hora de predecir, siempre conviene tener mayor cantidad de datos ya que se podrá entrenar mejor al clasificador.

En la **Figura 4a** se puede ver un gráfico de la exactitud del ajuste en función del tamaño de la muestra. Nótese que a medida que el tamaño de la muestra incrementa la exactitud también incrementa. Esto es un resultado positivo, ya que como el objetivo del trabajo es ajustar estos algoritmos al conjunto de muestras entero (con 42000 imágenes) y como nosotros estamos buscando los parámetros óptimos usando



(a) Exactitud del ajuste en función del tamaño de la muestra.



(b) Tiempo de cómputo de kNN en función del tamaño de la muestra.

Figura 4: Gráficos de exactitud y tiempo de cómputo de kNN sin PCA en función del tamaño de la muestra.

una muestra de 1000 imágenes este resultado nos asegura que los parámetros que encontremos no van a empeorar la calidad del ajuste cuando tomemos el conjunto entero de imágenes. Sin embargo, es importante mencionar que este resultado no nos asegura que los parámetros óptimos no cambien con el tamaño de la muestra. También es importante remarcar que el rango de tamaños llega hasta 5000 imágenes, número que sigue estando lejos de ser 42000. Esto lo hicimos porque si tomáramos el rango entero de muestras el programa tardaría demasiado tiempo en terminar. Esto es algo que se puede mejorar en trabajos futuros con más tiempo. El resto de las métricas presentaban gráficos similares.

En la **Figura 4b** se puede ver un gráfico de el tiempo de cómputo de kNN en función del tamaño de la muestra. Nótese que, como es de esperarse, el tiempo de cómputo incrementa con el tamaño de la muestra. Esto ya lo habíamos visto en la sección 3.2, donde el tiempo de cómputo de kNN incrementaba con el valor de α porque incrementaba el tamaño de la matriz. Sin embargo, a diferencia de ese gráfico este no es lineal. Esto se debe a que en este caso lo que incrementa es la cantidad de imágenes en el conjunto de datos y no la cantidad de dimensiones.

3.4.2. Estudio del tamaño de la muestra con PCA

Nuevamente, una de las cuestiones más interesantes de analizar gira en torno de la exactitud que se tendrá a medida que se aumenten las muestras. Se cree que no debería variar rotundamente con respecto a la exactitud con la misma cantidad de muestras sin contemplar el algoritmo de PCA. Con respecto al tiempo de cómputo, se tiene la hipótesis de que este debería mejorar, ya que esa es la razón principal para incorporar este método.

El gráfico de exactitud del método con PCA con respecto al tamaño de la muestra es muy similar al del primer método. La única gran diferencia es que kNN con PCA logra alcanzar mejores resultados con muestras menores que 1000, luego de esto convergen.

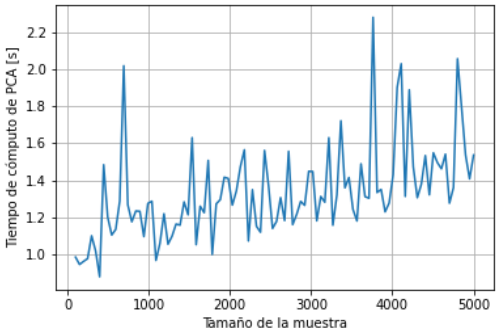


Figura 5: Tiempo de cómputo de PCA en función del tamaño de la muestra.

En la **Figura 5** puede verse un gráfico de el tiempo de cómputo de PCA en función del tamaño de la muestra. Nótese que el tiempo de cómputo es muy ruidoso pero que crece a medida que el tamaño de la muestra incrementa. Este incremento ruidoso es debido a que en lo único que el tamaño de la muestra afecta al algoritmo PCA es en el cálculo de la matriz de covarianza y no en el cálculo de autovalores. Ya que la matriz de covarianza es relativamente rápida de computar, el efecto del incremento del tamaño no es muy aparente. Luego, tal como se predijo, se puede ver una notable mejora con respecto al tiempo de cómputo en comparación al metodo kNN sin PCA.

Resultados	kNN sin PCA	kNN con PCA
Accuracy	$(0{,}9695 \pm 0{,}0028)$	$(0{,}9694 \pm 0{,}0031)$
κ	$(0{,}9661 \pm 0{,}0031)$	$(0{,}9660 \pm 0{,}0035)$
$F1$	$(0{,}9693 \pm 0{,}0027)$	$(0{,}9692 \pm 0{,}0030)$
Recall	$(0{,}9690 \pm 0{,}0027)$	$(0{,}9692 \pm 0{,}0029)$
Tiempo de kNN [s]	(880 ± 30)	$(16{,}36 \pm 0{,}04)$
Tiempo de PCA [s]	–	5,565
Tiempo total [s]	8820	169

Cuadro 1: Resultados de correr kNN con y sin PCA en el conjunto entero de 42000 imágenes.

3.5. Resultados de todo el conjunto

Gracias a la experimentación anteriormente descrita se logró obtener los parámetros óptimos para el conjunto, que son $k = 3$, $\alpha = 20$ y $K = 10$. Entonces, se usaron estos parámetros sobre el conjunto entero de 42000 imágenes y se obtuvieron los resultados que se muestran en la **Tabla 1**. Los errores de los valores mostrados se tomaron como las desviaciones estándar provenientes de las rondas de K-Fold Cross Validation. Es por eso que algunas cantidades como el tiempo total de ejecución y el tiempo de cómputo de PCA no tienen error registrado. Se muestran los resultados a dos cifras significativas del error para que no sean todos iguales y puedan apreciarse sus diferencias, aunque sean muy pequeñas.

Nótese que en casi todas las métricas el resultado de hacer kNN sin PCA es mejor que el de hacer kNN con PCA, siendo el recall la única excepción. Esto es esperable, ya que PCA es un algoritmo que reduce la dimensionalidad del conjunto de datos, peridiendo información pero ahorrando tiempo de cómputo. Nótese que esta diferencia es evidente, ya que el tiempo de cómputo total y de kNN es casi dos órdenes de magnitud menor para kNN con PCA que kNN sin PCA. De esta manera, se logró demostrar la utilidad y efectividad de la reducción de dimensionalidad de PCA ya que las diferencias en la bondad de ajuste es casi despreciable mientras que la diferencia en tiempo de cómputo es más que notable.

4. Conclusiones

Se estudió un conjunto de datos de 42.000 imágenes manuscritas. Se lograron hallar parámetros óptimos para realizar un ajuste sobre el conjunto usando los algoritmos kNN con y sin PCA. Se logró demostrar la efectividad de los algoritmos, tanto de kNN en su simplicidad y en sus resultados precisos como en PCA en su efectividad reduciendo el tiempo de cómputo sin afectar significativamente el resultado. Se estudiaron cuatro métricas distintas (exactitud, kappa de Cohen, puntaje F1 y recall) y las cuatro presentaron resultados considerablemente satisfactorios de aproximadamente $(0{,}970 \pm 0{,}003)$. El tiempo de cómputo de kNN con PCA resultó ser casi dos órdenes de magnitud menor que sin PCA. Se discutieron también posibles mejoras en el procedimiento para estudios futuros, relacionados con el tiempo de cómputo y optimización del mismo.

Referencias

[1] H. F. Schantz, *The History of OCR, Optical Character Recognition*. Manchester Center, 1982.

[2] Y. LeCun, C. Cortes, and C. J. C. Burgess, "The mnist database of handwritten digits."

[3] N. S. Altman, "An introduction to kernel and nearest neighbor nonparametric regression," *The American Statistician*, vol. 46, pp. 175–185, 1992.

[4] I. T. Jolliffe, *Principal Component Analysis*. Springer, 2002.

[5] D. M. Allen, "The relationship between variable selection and data agumentation and a method for prediction," *Technometrics*, vol. 16, pp. 125–127, 1974.

[6] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[7] C. E. Metz, "Basic principles of roc analysis," *Seminars in Nuclear Medicine*, vol. 8, pp. 283–298, 1978.

[8] J. Sim and C. C. Wright, "The kappa statistic in reliability studies: Use, interpretation, and sample size requirements," *Physical Therapy*, vol. 85, pp. 257–268, 2005.

[9] D. L. Olson and D. Delen, *Advanced Data Mining Techniques*. Springer, 2008.

[10] Y. Sasaki, "The truth of the f-measure," 2007.