DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Reconocimiento de dígitos

Métodos Numéricos

Grupo: 6

Integrante	LU	Correo electrónico
Tomás Caneda	490/17	tomy.3698@gmail.com
Mallol, Martín Federico	208/20	martinmallolcc@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

Resumen

En este trabajo se abarca el estudio de un método de un área crecientemente demandante y popular en las ciencias de la computación: *el reconocimiento de patrones*. El informe se estructura en las siguientes secciones:

- **Introducción**: Se propone el estudio de una herramienta de *reconocimiento óptico de dígitos* (**OCR**) con aprendizaje supervisado mediante *k vecinos más cercanos* (**kNN**), descubriendo necesario un método de reducción de dimensión para las muestras utilizadas. Para tal propósito se escoge el *análisis de componentes principales* (**PCA**), cuyos fundamentos teóricos se discuten brevemente.
- **Desarrollo**: Se plantea una implementación de **PCA** mediante el *método de la potencia* en combinación con *el método de deflación de Hotelling* para determinar sus α componentes. La experimentación se plantea tomando las muestras de la base de datos **MNIST**.
- **Experimentación, resultados y discusión**: Utilizando las métricas *accuracy* y *F1-score* en conjunción con el método de validación cruzada *K-folds*, se presenta un abanico de experimentos cuyo propósito es determinar los valores óptimos para la cantidad k de vecinos considerados en *kNN*, la cantidad α de componentes utilizadas para **PCA** y la cantidad de folds K en que se particiona el conjunto de muestras. Se evalúa, además, el desempeño de la herramienta en términos temporales.
- **Conclusiones**: Se concluye que la herramienta de **OCR** propuesta es generalmente satisfactoria, presentando un buen desempeño y valores aceptables para las dos métricas propuestas, aunque sensiblemente inferiores para *F1-score*. Se determinan $k = 3$, $\alpha = 21$ (para *accuracy*) y $\alpha = 16$ (para *F1-score*) como valores óptimos para los métodos **kNN** y **PCA** empleados, con $K = 10$ como un valor razonable de folds para las métricas escogidas.

Palabras clave: *Optical Character Recognition*, *k-Nearest Neighbors*, *Principal Component Analysis*, *Cross-validation*.

Índice

1. Introducción

El área de *pattern recognition*, el reconocimiento automatizado de patrones, es hoy tan popular como amplia y compleja, existiendo múltiples enfoques para resolver los desafíos que se le presentan y aplicaciones en un sinnúmero de situaciones cotidianas, desde el reconocimiento de mails spam hasta la detección de síntomas de enfermedades. Se propone en este trabajo explorar este interesante campo de la computación mediante el estudio de un simple método de lo que podría considerarse una de sus ramas, el *reconocimiento óptico de caracteres* (en inglés abreviado **OCR**, *Optical Character Recognition*).

1.1. Metodología

El método particular de **OCR** que se abordará en este trabajo implementará el reconocimiento de dígitos decimales manuscritos (0 al 9), representados como imágenes de 28×28 píxeles en escala de grises, mediante aprendizaje supervisado (*supervised learning*) y el criterio de los k *vecinos más cercanos* (en inglés abreviado **kNN**, *k-Nearest Neighbors*, con k a elección). Más específicamente, el método interpretará a las imágenes como vectores $x \in \mathbb{R}^{28*28} = \mathbb{R}^{784}$ (que almacenan todos sus píxeles) y dispondrá de una base de datos de entrenamiento $\mathcal{D} = \{x_i : i = 1, ..., n\}$ de n imágenes ya etiquetadas (de acuerdo al dígito del 0 al 9 al que corresponden), que usará para clasificar a cualquier nueva imagen $y \notin \mathcal{D}$ ($y \in \mathbb{R}^{784}$) tomando la clase con mayor número de repeticiones entre los k vecinos en $\mathcal{D}$ más cercanos a y por distancia euclídea.

Aunque simple, el método planteado puede resultar altamente costoso (se deben calcular las distancias euclídeas para cada vector y a clasificar respecto a lo que usualmente serán varios miles de vectores en $\mathcal{D}$ que además serán de dimensión $\mathbb{R}^{784}$, con una cantidad considerable de elementos) y es además víctima de lo que se conoce como la "maldición de la dimensión" (*curse of dimensionality*), un término que refiere a una serie de fenómenos que surgen al operar en espacios de altas dimensiones que no se presentan en espacios de bajas dimensiones. En particular, la búsqueda de los vecinos más cercanos se ve comprometida en espacios de dimensiones altas (más detalles en [2]).

Para sortear las dificultades presentadas se recurrirá a una técnica de reducción de dimensión conocida como *análisis de componentes principales* (en inglés abreviada **PCA**, *Principal Components Analysis*), de modo que los $x \in \mathcal{D}$ se reduzcan a unas pocas componentes que concentren tanta información como sea posible de las 784 originales.

Se tiene entonces una estrategia concreta para la implementación de la herramienta de **OCR** que se desarrollará, por lo que se dará lugar a continuación a una breve discusión de los fundamentos teóricos de los métodos **kNN** y **PCA** de los que hará uso.

1.1.1. k-Nearest Neighbors (kNN)

La idea tras el método **kNN** (ver [4], **1.4.2 A simple non-parametric classifier: K-nearest neighbors**) es simple: clasifica a un vector (imagen) $y \in \mathbb{R}^{784}$ con la moda (la más repetida) de las clases de los k vectores $x_{i_1}, ..., x_{i_k} \in \mathcal{D}$ que minimizan la distancia euclídea respecto a dicho y, donde la distancia euclídea entre dos vectores $p = (p^{(1)}, ..., p^{(n)}), q = (q^{(1)}, ..., q^{(n)}) \in \mathbb{R}^n$ se define como $d(p, q) = \sqrt{\sum_{i=1}^{n} (p^{(i)} - q^{(i)})^2}$. Notar que $\#\mathcal{D} = n$ (cantidad de elementos en $\mathcal{D}$) será por lo general un valor alto, y que los vectores en dicho conjunto ya se encuentran en una dimensión alta relativa a las usuales (con $n = 784$), de modo que este método, por sí mismo, resultará costoso si no se toman medidas adicionales.

1.1.2. Principal Components Analysis (PCA)

PCA (ver [4], **12.2 Principal components analysis (PCA)**) permite "resumir" en unas pocas componentes (no sin cierta pérdida) la información almacenada en todos los valores de los $x_i \in \mathcal{D}$. Esto, como ya fue mencionado, permite lidiar con las dificultades asociadas al método **kNN**, tanto desde una perspectiva cualitativa (evitando la "maldición de la dimensión") como cuantitativa (reduciendo el tamaño de su entrada y así su tiempo de cómputo).

Intuitivamente, el método **PCA** busca una transformación definida sobre una base de n muestras $x \in \mathbb{R}^m$, que se denominará *transformación característica* y se notará como tc, tal que $tc : \mathbb{R}^m \to \mathbb{R}^\alpha$ con $\alpha \ll m$, intentando maximizar la dispersión de los datos en las nuevas α componentes de modo que se minimice la pérdida de información respecto a los m datos originales (al priorizar entre ellos a aquellos que ofrecen más diversidad, reduciendo la impronta de los más similares). Los pasos para lograr este cometido pueden resumirse como sigue:

1. Se define una matriz $X \in \mathbb{R}^{n \times m}$ ($n = \#\mathcal{D}, m = 784$) tal que $fila_i(X) = X_i^t = \frac{1}{\sqrt{n-1}}(x_i - \hat{\mu})^t$ $\forall i \leq n$, considerando que $\hat{\mu} = (\sum_{i=1}^n x_i)/n$ es la media empírica de las n muestras en la base de entrenamiento $\mathcal{D}$ $(x_1, ..., x_n)$.

2. Se calcula luego lo que se denominará la matriz de covarianzas empírica $M = X^t X \in \mathbb{R}^{m \times m}$ para las $m = 784$ componentes de los n vectores, de modo tal que $M_{jk} = \hat{\sigma}_{x^{(j)}x^{(k)}} = \frac{1}{n-1}\sum_{i=1}^n (x_i^{(j)} - \hat{\mu}^{(j)})(x_i^{(k)} - \hat{\mu}^{(k)}) = \hat{\sigma}_{x^{(k)}x^{(j)}} = M_{kj}$ $\forall j, k \leq m$, resultando entonces M simétrica. Como consecuencia de la simetría de M, esta tiene una base ortonormal de m autovectores, cada uno asociado a uno de los m autovalores (consecuencia del *Teorema de descomposición espectral* o *Spectral Theorem*[5]).

3. Ocurre luego, para todo vector $v \in \mathbb{R}^m$ de norma 1, que $v^t M v$ corresponde a la varianza empírica de los $v^t x_i$ $\forall i \leq n$, donde $v^t x_i$ son las proyecciones de los x_i en la dirección (recta) de v. Se busca entonces los vectores v (de norma 1) cuyas direcciones maximicen dicha varianza, para luego usar las proyecciones de los x_i hacia las direcciones de tales vectores como nuevas componentes para cada x_i, que resumirán todos sus m datos proyectándolos sobre una dirección en la que tienen alta varianza (aparecen dispersos, y por ende resultan poco redundantes). Dichos vectores resultan ser los autovectores (de norma 1) asociados a los m autovalores de M, de modo tal que los autovectores asociados a los autovalores de mayor magnitud ofrecen las direcciones de mayor varianza para las proyecciones de los x_i. Se calculan entonces $\alpha < m$ autovectores $v \in \mathbb{R}^m$ con $\| v \|_2 = 1$ (tantos como la cantidad α de componentes a las que se desee reducir las muestras) ordenadamente de acuerdo a la magnitud del autovalor asociado, y se los usa como filas para armar una matriz de cambio de base $W \in \mathbb{R}^{\alpha \times m}$ ($fila_i(W) = v_i^t$ $\forall i \leq \alpha$, donde v_i es el autovector de norma 1 asociado al i-ésimo autovalor de mayor magnitud).

4. Con W construida, basta con aplicarla como transformación lineal a cualquier vector $x \in \mathbb{R}^m$ para conseguir su *transformación característica* de acuerdo a las α componentes escogidas, $Wx = tc(x) = (v_1^t x, v_2^t x, ..., v_\alpha^t x) \in \mathbb{R}^\alpha$. Notar que W también puede construirse usando a los autovectores como columnas, en cual caso $tc(x) = x^t W$.

2. Desarrollo

La herramienta de **OCR** propuesta y todos los métodos necesarios para realizarla fueron implementados en `C++` utilizando la librería *Eigen*[1] (versión `3.3.7`), exportando mediante *Pybind*[2] (versión `2.2.4`) los métodos (como clases con funciones) a notebooks *Jupyter* (`jupyter-notebook`

[1] `https://eigen.tuxfamily.org/index.php?title=Main_Page`, repositorio en `https://gitlab.com/libeigen/eigen`.

[2] `https://github.com/pybind/pybind11`.

`6.0.3` y `jupyter core 4.6.3`), en las cuales se realizó la experimentación y se generaron los gráficos. Para más detalles, consultar el archivo `README.md` y `requirements.txt` asociados al trabajo y los directorios `src` y `notebooks` con el código fuente y las notebooks para experimentaciones, respectivamente.

Los métodos **kNN** y **PCA** fueron implementados mediante clases de `C++` con, en el caso de la primera, funciones públicas `fit` y `predict` (código en `src/knn.cpp`), y, en el caso de la segunda, funciones públicas `fit` y `transform` (código en `src/pca.cpp`). Dada la simpleza algorítmica de los métodos no se ofrecerá pseudocódigo ilustrativo de las funciones mencionadas, pero se insta a los lectores a consultar el código fuente, que fue comentado concienzudamente para facilitar su lectura y comprensión.

La implementación de **PCA** precisó de un método de cálculo de α autovectores para la matriz de covarianzas M. Dicho cálculo consistió en una combinación del *método de la potencia* (*Power Method*, consultar [1] **7.3.1 The Power Method**) y el *método de deflación de Hotelling* (*Hotelling's deflation* [3]). La implementación de estos métodos puede econtrarse en `src/eigen.cpp`.

Por último, las muestras usadas a lo largo de la experimentación fueron tomadas de la base de datos **MNIST** en la versión disponible en *kaggle* para la competencia *Digit Recognizer*[3].

3. Experimentación, resultados y discusión

3.1. Métricas utilizadas

Para la experimentación de este trabajo, nos centramos en dos métricas en particular. La primera, y más importante, es la tasa de efectividad lograda, también llamada accuracy. Mientras que la segunda que tomamos en consideración fue la métrica F1. Las pusimos a prueba en todos los experimentos que realizamos para ver cuál era su performance. Quisimos constatar si había tests donde una sería mucho mas eficaz que la otra, pero a su vez habría casos donde esta misma se desempeñaría mucho peor que su contraparte. ¿O acaso siempre una de las dos métricas dominaría categóricamente? A medida que fuimos probando escenarios, nos fuimos dando cuenta que la primer métrica, accuracy, fue constantemente la que mejor valor devolvió. A continuación explicaremos brevemente en qué consiste cada métrica.

3.1.1. Accuracy

Con esta métrica se busca encontrar la cantidad de aciertos totales por sobre la cantidad total de resultados obtenidos. Básicamente se calcula la media de todos los resultados obtenidos.

Si quisieramos escribirlo en formato de fórmula, se tienen en consideración cuatro componentes para integrarla. Estos son: los verdaderos positivos (tp), los verdaderos negativos (tn), los falsos positivos (fp) y por último, los falsos negativos (fn). Por lo tanto, la fórmula para esta métrica quedaría: $\frac{tp+tn}{tp+fn+fp+tn}$

3.1.2. F1-Score

F1 se centra en dos métricas que no usamos en la experimentación pero eran elegibles. Una es "Recall" y la otra "Precission". La primera, dada una clase "i" calcula la cantidad de predicciones exitosas sobre la cantidad de elementos "i". La segunda calcula (también dentro de una clase) la cantidad de aciertos positivos sobre la cantidad total de elementos positivos.

[3] `www.kaggle.com/c/digit-recognizer`

Por lo tanto, F1-Score busca combinar la efectividad de ambas métricas y se centra en medir la relación entre las dos. Técnicamente calcula la media armónica entre "Precission" y la "Recall". Por lo que su fórmula está dada por lo siguiente: $\frac{2*Precission*Recall}{Precission+Recall}$

3.2. Evaluación de los métodos a utilizar

3.2.1. K-fold cross validation

Para evaluar los distintos algoritmos que especificamos en C++ usamos como procedimiento el de "K-fold cross validation". Si probasemos nuestors algoritmos con una muestra x de la base de datos disponibles, es posible que los resultados arrojados no sean representativos, ya que se puede contar con datos que no representen a la totalidad de los mismos. Por lo tanto, utilizamos este método que se centra en particionar los datos que tenemos disponibles en K-1 grupos y se entrenan mientras que un integrante de dicho grupo se utiliza como validación.

3.3. Estudio de kNN sin PCA

3.3.1. Estudio de k

Primero comenzamos estudiando cómo la variación de la cantidad de vecinos ('k') afecta a la exactictud del algoritmo **k-Nearest Neighbors**. Para realizar dicho estudio, primero lo hicimos sin tener en cuenta la redimensión de la imagen con **PCA**. Luego, lo volvimos a realizar con $\alpha =$ **10 y 50**. En todos los casos, se usó como fold a $\mathbf{K = 10}$, que es un número estándar para este tipo de investigaciones en la rama del machine learning. Elegimos como valores de **'k'vecinos**, los que van **del 1 hasta el 50**.

3.3.2. Estudio de k sin PCA

Tuvimos como hipótesis que si el valor de k era muy bajo, la "accuracy" no sería tan eficiente como si se tomara un valor de mayor magnitud. Para nuestra sorpresa, aunque era esperable que en los primeros valores de k hubiese mucha variación en los resultados de su predicción, fueron mucho más altos de lo que esperamos. Hasta llegar el punto en el que tomar un k demasiado alto era mucho más ineficiente que tomar un "k" minúsculo. Esto nos demostró que lo más viable es tomar un "k" no muy grande para hacer una ronda de predicciones.

La caída de exactitud a medida que los vecinos considerados aumentan, decrece de manera lineal. Tanto para la métrica "Accuracy" como para la métrica "F1". Las comparamos una con la otra para ver qué método es preferible utilizar. Ambos se comportan muy parecido, pero en todo momento el primero nos da una mayor exactitud que el segundo.

Como primer caso, intentamos ver cómo era el comportamiento del algoritmo cuando los vecinos a tomar son muy pccos. La diferencia entre las dos métricas es siempre casi la misma, con "Accuracy" siendo la vencedora. Los valores oscilan bastante entre 0.9 y 0.87 para los k más pequeños. Luego muestra una clara merma que es la tendencia que seguirá a medida que incremente k (como se podrá ver en los siguientes gráficos).

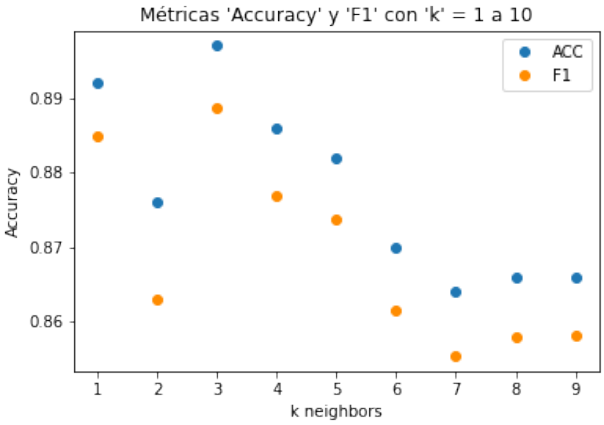

Figura 1: Gráfico de la performance de $'Accuracy'$ y $'F1'$ con k en sus valores más pequeños

Como tomar sólo una muestra chica de "k" puede llegar a no ser representativo, procedimos a realizar el experimento pero esta vez hasta llegar a un valor de 100 vecinos. Con esto nos quedó clara la tendencia que tímidamente mostró el otro gráfico. El método empieza a arrojar menos exactitud a medida que los vecinos a elegir se acercan al tamaño de la muestra.

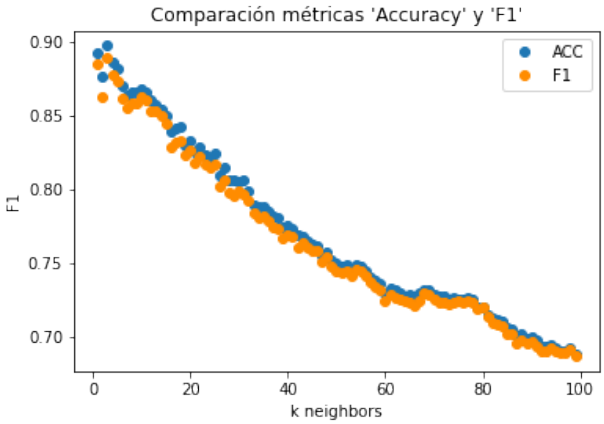

Figura 2: Gráfico de la performance de las métricas elegidas con k de 1 hasta 100.

Viendo el gráfico de arriba y el de abajo, se observa que la métrica accuracy siempre prevalece por sobre F1, aunque cabe destacar que sus curva son muy parecidas y los resultados arrojados también lo son. Tanto el point plot como el histograma demuestran su semejanza.

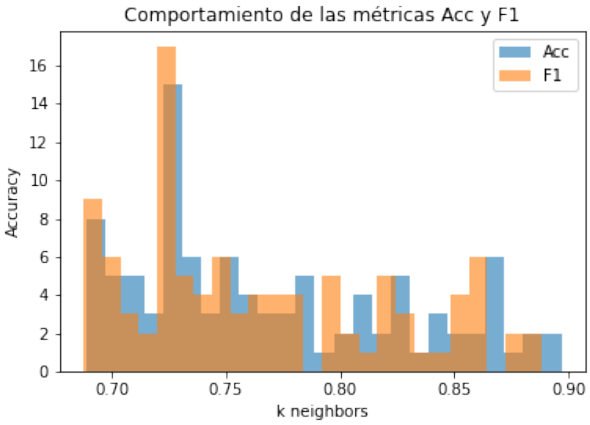

Figura 3: Histograma de la cantidad de valores de exactitud de cada métrica.

3.3.3. Estudio de α

Para estudiar el valor óptimo del α del método PCA, se trabajó con 'k'desde 1 hasta 100 y 'K'= 10. Para la elección de los α, hicimos su seguimiento desde el valor 1 hasta el valor 50.

Como hipótesis tomamos la idea de que, si contamos con un α muy pequeño, habrá una gran inexactitud en el valor final, mientras que al aumentar la cantidad de autovectores que se toman en consideraciíon, la exactitud aumentaría.

Lo que notamos al realizar el experimento, es que al tener un α demasiado grande, el nivel de exactitud comienza a bajar y a parecerse al valor del método kNN sin redimensión de su muestra. Por lo tanto no tiene sentido utilizar un valor cercano a la máxima cantidad de autovectores posibles. El sentido del método es poder reducir considerablemente la dimensión de las muestras con tal de poder tener valores representativos.

La figura de abajo arroja como revelación que a partir de un α con valor de 15, la accuracy se mantiene no sólo en niveles altos sino con (aproximadamente) el mismo valor. El pico de rendimiento del test "Accuracy" se evidencia con $\alpha = 20$ (para ambos "k"), mientras que para la métrica "F1" obtiene en $\alpha = 16$ (también para ambos k). Teniendo en cuenta el tiempo que ahorra tener un α bajo (y encima eficiente), consideramos a $\alpha = 20$ como el ideal para mayor efectividad.

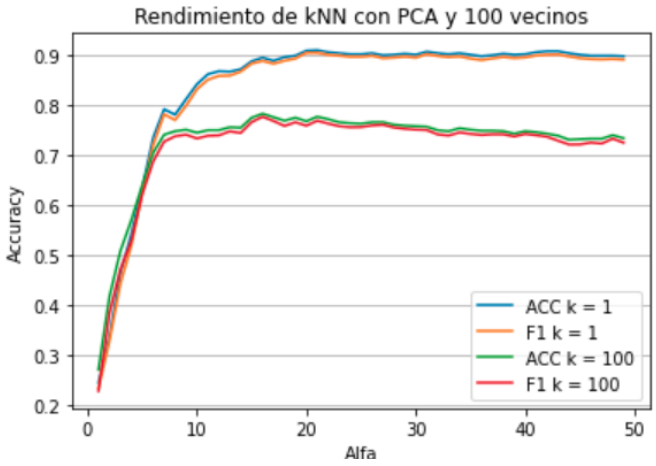

Figura 4: Estudio del coeficiente α del algoritmo PCA. Puesto a prueba con 1 vecino y con 100 vecinos. Ambos tests hechos con la métrica "Accuracy" y "F1".

3.4. Diferencia de kNN con y sin PCA

Al tener ambas experimentaciones hechas, las pusimos en comparación una con la otra. Comparamos el metodo kNN con k = 50 y con $\alpha = 20$ y 50. Concluímos que al utilizar un α ideal, el rendimiento del método kNN aumenta. No solo aumenta con un α ideal (20), sino que al utilizar un α por encima del ideal, obtendremos mejores resultados que si no usaramos uno. Por lo tanto, siempre que usemos una cantidad prudente de autovectores, la redimensión con PCA provocará una mejoría en el rendimiento del algoritmo kNN.

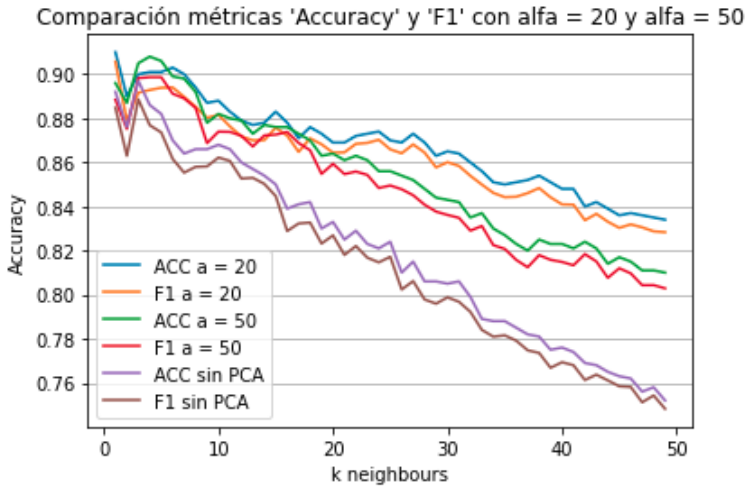

Figura 5: Gráfico de la performance de $'Accuracy'$ y $'F1'$ con PCA (distintas reducciones de dimensión) y sin PCA.

Como ejemplo, acá tenemos el caso en que un α mal elegido puede perjudicar enormemente la performance de nuestro algoritmo. Con $\alpha = 4$ (muy pequeño) no tiene sentido realizar la disminución de la dimensión porque nos quedaríamos con datos muy poco representativos a la muestra que teníamos originalmente. Por lo tanto, en este caso, no es recomendable aplicar PCA antes de realizar kNN.

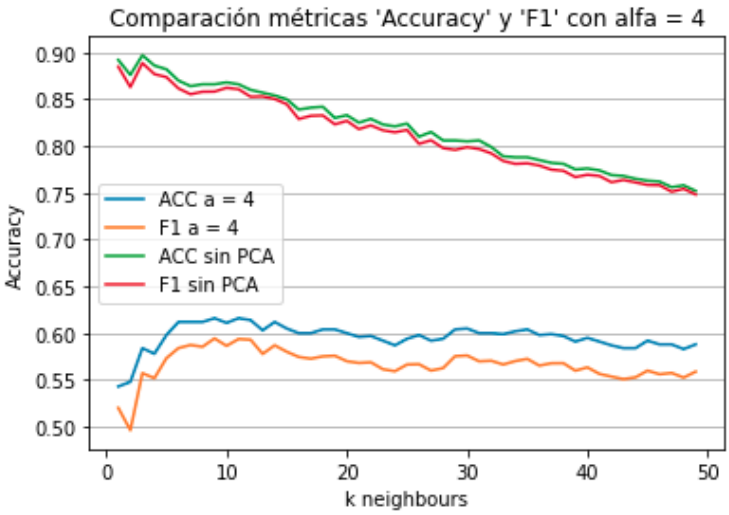

Figura 6: Gráfico de la performance de $'Accuracy'$ y $'F1'$ con PCA ($\alpha = 4$) y sin PCA.

3.4.1. Tiempo de ejecucion

Estudiamos cómo cambia el tiempo de ejecución del algoritmo kNN con y sin PCA. Es decir, con su muestra de datos original y con esa misma muestra pero con su dimensión reducida. Los resultados fueron muy alentadores, demostrando que, efectivamente, el método PCA no sólo ayuda con los tiempos de ejecución (de manera notable), sino que también ayuda con la tasa de efectividad en sus resultados.

Hicimos la experimentación sobre una base k vecinos que va de 1 hasta 50, y con $\alpha = 20$ y 50. Comparamos sus resultados con el método kNN sin PCA y obtuvimos esas concluciones.

Tiempo de ejecución de kNN con k de 1 a 50

Sin PCA	1.98 segundos
Con alfa = 4	0.93 segundos
Con alfa = 20	4.55 segundos
Con alfa = 50	37.4 segundos

Figura 7: Tabla de la performance del tiempo de ejecución cuando se utiliza el algoritmo kNN con y sin PCA.

3.5. Estudio de muestra

Para hacer a mayoría de nuestros experimentos utilizamos una muestra de 1000 datos de la base de train que se nos fue provista. Ahora, cuando tuvimos que experimentar sobre la muestra, aumentamos considerablemente la cantidad de data a experimentar para poder ver qué tan favorecidos salían los métodos kNN y PCA al recibir un mayor caudal de datos. Elegimos muestras de 500, 1000, 2500 y 5000 elementso. Utiliazamos 10 folds como siempre (que sería la medida estándar), y estudiamos el algoritmo kNN con vecinos de 1 a 50. Como el estudio de muestra fue realizado utilizando PCA para redimensionar el espacio, elegimos el α que mejores resultados nos dio, o sea, $\alpha = 20$.

El test de muestra se realizó sobre la métrica “Accuracy”, que fe la que arrojó resultados más estables hasta ahora.

Intuímos que al aumentar el tamaño de la cantidad de datos disponibles, los algoritmos funcionarían mejor ya que tendrían la capacidad de trabajar con una muestra mucho mas representativa. Por lo que sus resultados serían más certeros. El experimento nos dio la razón y no sólo nos arrojó mejor exactitud la mayor muestra de todas, sino que a medida que se acrecientan los datos, más estable se vuelve la solucion. No varía tanto la taza de efectividad. Aún así, mientras más vecinos se tienen en cuenta, menos accuracy obtenemos. Esto respeta lo que experimentamos al estudiar el efecto que puede producir el valor de “k”. El α que escogimos nos permitió obtener los resultados en muy poco tiempo, ya que fue el α ideal.

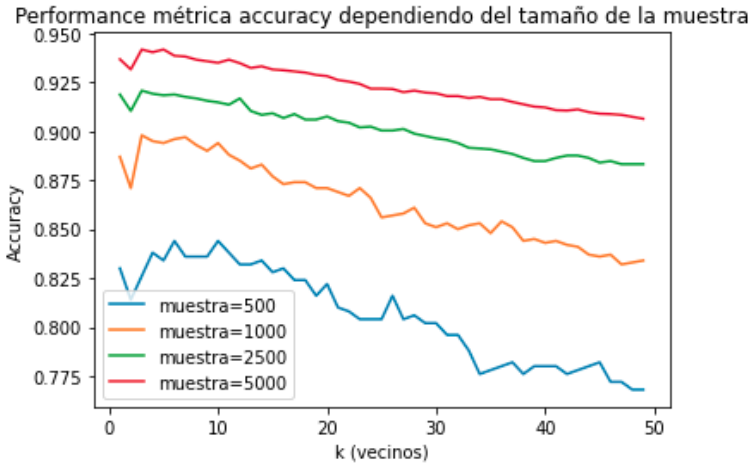

Figura 8: Gráfico de la performance de $'Accuracy'$ con distintos tipos de tamño de muestra, redimensión mediante PCA ($\alpha = 20$) y distintos vecinos en consideración.

Como era de esperarse, la exactitud del metodo KNN subió al poder trabajar con más data. Nos demuestra que es un método fiable cuando la muestra que se tiene es lo suficientemente representativa. También pudimos notar que el algoritmo es mucho más estable con una gran muestra. Es decir, al aumentar los ’k’vecinos considerados, la performance no decae como lo hace en muestras más pequeñas.

3.5.1. Tiempo de ejecución

Tiempo de ejecución de kNN con PCA y con distintas magnitudes en su muestra

Muestra de 500 elementos	1.3 segundos
Muestra de 1000 elementos	1.78 segundos
Muestra de 2500 elementos	6.09 segundos
Muestra de 5000 elementos	19.8 segundos

Figura 9: Tabla de la performance del tiempo de ejecución cuando se utilizan varias muestras para el algoritmo kNN con PCA.

3.6. Estudio de KFold

Por último nos centramos en estudiar para qué valores de K obtenemos una mayor efectividad en nuestras predicciones. El 'K'vendría a ser la cantidad de folds en las que se entrena a nuestro conjunto de datos. Para luego calcular la media de los resultados de todos ellos, y asi aproximar nuestra respuesta. Pudimos notar algo curioso, que es que a un K mayor, el método F1 rinde mucho peor que el método accuracy. Se ve mucho más perjudicado que su competidor. No sólo baja su exactitud sino que sus resultados son más inestables. Esto pudimos notarlo al, no sólo calcular el resultado medio de todos nuestros datos, sino tambien el resultado de los 'folds´ máximos y mínimos de cada método. Esto nos dio una pista sobre cómo varía la performance de cada método al darle una muestra reducida de datos. Mirando los graficos claramente se puede ver como la métrica accuracy arroja datos muchos mas parejos y cercanos a la media. Aún así, cuando la cantidad de folds a tomar es muy grande, inevitablemente la diferencia entre el valor medio y el valor del fold maximo y minimo se acrecienta considerablemente.

En los primeros 10 folds, cuando tenemos muestras más grandes, ambos metodos estan muy parejos. De hecho, el gráfico nos da la revelación de que "F1" funciona mejor cuando tenemos pocos folds, como 2 o 4. Por lo que, al contar con pocas particiones, esta se alza como la métrica más fiable. Ahora, de 5 folds en adelante, se ve como la tendencia cambia y "Accuracy" revierte la situación anterior. La diferencia entre la media y el fold máximo y mínimo se empieza a acrecentar a medida que se agrandan las particiones.

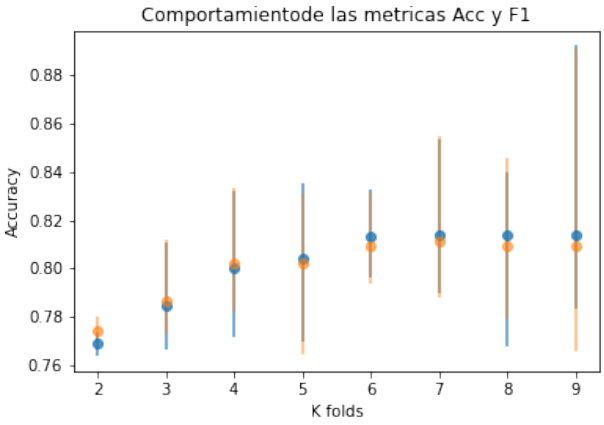

Figura 10: Estudio del coeficiente "K" del algoritmo KFold. Puesto a prueba con 25 vecinos. Ambos tests hechos con las métricas "Accuracy" y "F1".

Luego extendimos el estudio a 50 folds. Aca la tendencia es clara, "Accuracy" mantiene el valor de exactitud mientras que "F1' despues de 10 folds empieza a percibir una gran caìda en su fiabilidad, que no se revierte. Usar este tipo de gr+aficos nos permitió mirar la diferencia que existe entre el valor medio arrojado por los folds, y el valor máximo y mínimo. Quisimos no solo quedarnos con la media, sino con qué tanto se pueden a llegar a diferenciar los resultados de cada fold entre ellos. Esto nos demuestra que "Accuracy" mantiene niveles de exactitud más parejos en sus folds que "F1". Aunque cabe destacar que, como era esperable, a mayor cantidad de particiones, mayor diferencia habrá entre las predicciones de cada fold. Ambos gráficos nos demuestran este suceso.

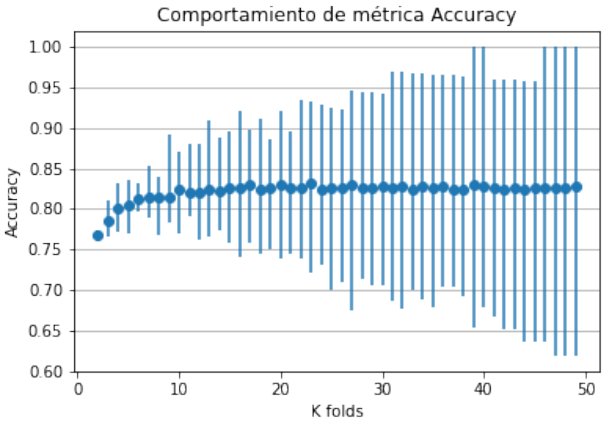

Figura 11: Estudio del coeficiente "K" del algoritmo KFold y performance de la métrica "Accuracy". Se asigna con un punto al valor medio de exactitud de todos los folds. Las lineas de la parte superior son la diferencia entre el valor medio de los folds y el valor máximo de efectividad alcanzado por uno. Análogamente, las lineas de la parte inferior, son la diferencia del fold con el valor de exactitud mínimo, y la media.

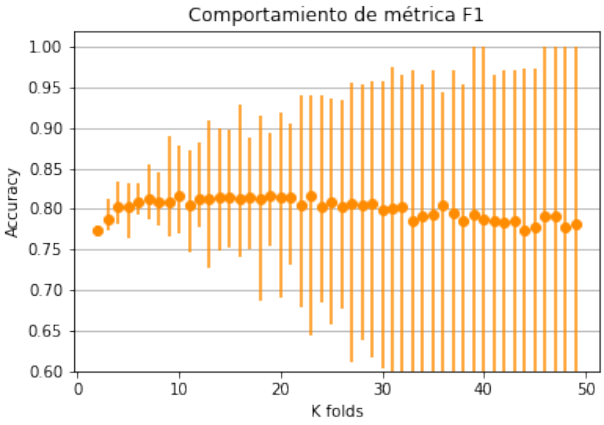

Figura 12: Estudio del coeficiente "K" del algoritmo KFold y performance de la métrica "F1". Mismo tipo de gráfico que el anterior.

Por último, comparamos en una misma figura la exactitud de ambos métodos. Como tal vez en los anteriores dos gráficos no se notaba tal diferencia entre una métrica y la otra, las pusimos en el mismo para reflejar lo distinto que fue su performance. La figura nos demuestra que haber tomado un K de valor 10 para las anteriores experimentaciones fue prudente.

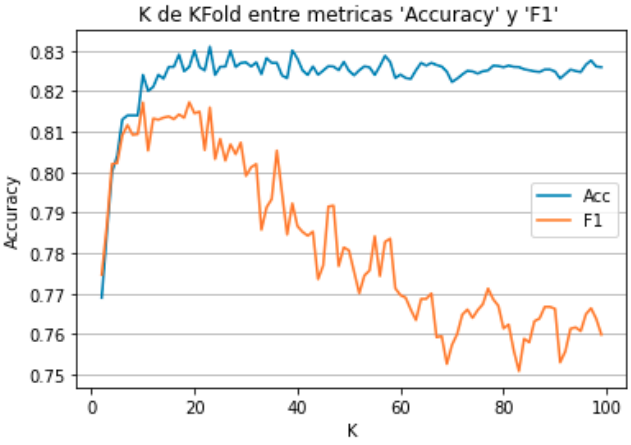

Figura 13: Estudio del coeficiente "K" del algoritmo KFold y performance ambas métricas.

3.6.1. Tiempo de ejecucion

Tiempo de ejecución de kNN con Folds

Folds de 1 a 50	42.3 segundos
Folds de 1 a 100	1 minuto 39 segundos

Figura 14: Tabla de la performance del tiempo de ejecución cuando se utilizan folds de 1 a 50 y de 1 a 100.

4. Conclusiones

Concluímos que tanto el algoritmo k-Nearest Neighbors (kNN) como el algoritmo Principal Components Analysis (PCA) son lo suficientemente eficientes para ser utilizados en el ámbito del Reconocimiento de Dígitos. Arrojaron exactitudes por arriba del 0.950 en una muestra lo suficientemente considerable. El PCA no solo ayudó con la eficiencia del resultado sino que también permitió que los tiempos de ejecución se reduzcan drásticamente. Siempre y cuando se elijan parámentros adecuados, por supuesto. El elegir parámetros que no responden a las necesidades de la experimentación puede arrojar resultados mediocres (como un α demasiado pequeño o una cantidad de vecinos demasiado grande).

El método k-Fold Cross Validation también demostró ser un procedimiento más que fiable. Contando con una cantidad de folds adecuada, provocó que la muestra a testeaer resultara más representativa, se disminuyó mucho el sesgo que podría llegar a haber si se usara la muestra completa sin un entrenamiento previo. Aunque cabe destacar que la métrica "F1'Score" vio un decrecimiento en su efectividad a medida que los folds aumentaban a valores muy grandes.

Por lo tanto, teniendo en cuenta las dos métricas que fueron consideradas, concluímos que "Accuracy" nos muestra resultados muchísimo más precisos y, además, estables. Ya que sus predicciones eran muchísimo más cercanas a la media que la de su contra parte. No fue afectada por el aumento de los folds y casi en la totalidad de las pruebas dio una exactitud mayor a "F1" (salvo cuando se contaba con muy pocos folds, como 2 o 3).

En cuanto al tamaño de la muestra, se probó que, a mayor cantidad de datos disponibles, menos sesgada va a ser la respuesta que nos den nuestras métricas. No solo la exactitud fue mayor, sino que la estabilidad de las respuestas también aumentó.

Como consideraciones finales, sobre cuál sería el escenario ideal para predecir un dígito aleatorio, la experimentación nos reveló que el número de vecinos óptimo a elegir ("k") es 3. La cantidad de autovectores a elegir para realizar la re-dimensión ("α") es 21 para la métrica Accuracy y 16 para la métrica F1-Score. Por el lado de la cantidad de folds a elegir, K = 10 probó ser una medida más que razonable para ambas métricas. Esto muestra por qué es uno de los valores estándares dentro de la especialidad del "machine learning". Finalmente, el tamaño total de la muestra a considerar debería ser el mayor posible siempre y cuando no entorpezca el tiempo de ejecución del programa.

Referencias

[1] Gene H. Golub y Charles F. Van Loan. *Matrix Computations, Fourth Edition*. Johns Hopkins University Press, 2013. ISBN: 978-1421407944.

[2] Beyer K. y col. *When Is Nearest Neighbor Meaningful?* Inf. téc. TR1377. University of Wisconsin-Madison Department of Computer Sciences, jun. de 1998. URL: `https://minds.wisconsin.edu/bitstream/handle/1793/60174/TR1377.pdf`.

[3] Lester W. Mackey. "Deflation Methods for Sparse PCA". En: *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*. Ed. por Daphne Koller y col. Curran Associates, Inc., 2008, págs. 1017-1024. URL: `https://proceedings.neurips.cc/paper/2008/hash/85d8ce590ad8981ca2c8286f79f59954-Abstract.html`.

[4] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012. ISBN: 978-0262018029.

[5] Walter Rudin. *Functional Analysis, Second Edition*. McGraw-Hill, 1991. ISBN: 978-0070542365.