DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Reconocimiento de dígitos

Métodos Numéricos

Integrante	LU	Correo electrónico
Gómez, Bruno Agustín	428/18	bgomez@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Resumen

El reconocimiento ópticos de caracteres (OCR) permite la automatización de la digitalización de documentos físicos o detección de patentes, entre otros. En el presente informe se propone analizar y comparar el comportamiento de dos métodos de clasificación de caracteres a partir de una base de entrenamiento, kNN con y sin PCA (análisis de componentes principales). Mediante experimentos con validación cruzada se pretende establecer cuál es la mejor combinación de hiperparámetros para los datos. El método kNN consiste en determinar la identidad de un elemento a partir de los elementos de entrenamiento más cercanos. PCA intenta reducir la redundancia del sistema y así utilizar un menor número de variables lo suficientemente informativas. Se utilizó un conjunto de imágenes etiquetadas perteneciente a la base de datos MNIST. Se observ ó que el método con PCA es más rápido y otorga mejores clasificaciones. Se observó que la mejor cantidad de vecinos que puede utilizarse en kNN es $k = 5$ y la mejor cantidad de componentes principales (α) para el análisis PCA es $\alpha = 40$. Se observó que para realizar validación cruzada era suficiente con realizar una partición en $K = 5$ para obtener resultados representativos en el menor tiempo posible.

Palabras claves: *reconocimiento de caracteres, OCR, experimentación, métodos numéricos, kNN, PCA, K-Fold*

2. Introducción

El reconocimiento óptico de caracteres (OCR, por sus siglas en inglés), generalmente conocido como reconocimiento de caracteres consiste en la conversión mecánica o electrónica de imágenes de información escrita, tipográfica o manuscrita en texto digitalizado a partir de un documento escaneado, una foto de un documento, una fotografía (que permita reconocer texto en señales o carteles), etc. El OCR es ampliamente utilizado como mecanismo de *data entry* a partir de documentos impresos (como pueden ser pasaportes, informes bancarios, etc.). Además, es utilizado en la digitalización de documentos, lo que permite la educación electrónica, búsqueda, almacenamiento compacto y permite que sean utilizados en procesos de computación cognitiva, traducción automática, *text-to-speech*, *text mining*, etc.

En el caso concreto de los textos, existen y se generan continuamente grandes cantidades de información escrita, tipográfica o manuscrita. Es por ello que poder automatizar la introducción de caracteres evitando la entrada por teclado implica un importante ahorro de recursos humanos y un aumento de la productividad, al mismo tiempo que se mantiene, o hasta se mejora, la calidad de muchos servicios.

Los primeros sistemas OCR eran entrenados con imágenes de cada carácter y trabajaban con una tipografía a la vez. Actualmente, los sistemas más avanzados permiten un alto grado de exactitud al reconocer todo tipo de tipografías y soportan una amplia variedad de formatos de entrada.

El proceso de OCR consiste en dos etapas principales. Por un lado, la selección de una base de datos que servirá para entrenar al sistema, permitiendo calibrar los criterios que utilice para clasificar los distintos caracteres. Luego, la utilización del sistema generado para reconocer dentro de una imagen o conjunto de imágenes el o los caracteres presentes.

En particular, en este trabajo, utilizaremos imágenes de la base de datos MNIST, en la versión disponible en *kaggle* para la competencia *Digit Recognizer*. Para el proceso de entrenamiento utilizaremos los métodos *kNN* y *PCA*.

2.1. *k* vecinos más cercanos (kNN)

El método de los *k* vecinos más cercanos es un método de clasificación supervisado. En este trabajo se utiliza para reconocer dígitos que vienen representados por imágenes a escalas de grises. En la base de entrenamiento, las imágenes van a tener las etiquetas correspondientes al dígito que corresponden.

Dada una nueva imagen de un dígito *d*, con el mismo formato, vamos a recorrer toda la base de

entrenamiento buscando los x_i elementos que minimicen $min_{i=1,\dots,n}\|d - x_i\|_2$. Se consideran los k vecinos más cercanos al elemento que se quiere clasificar. Luego de encontrar los vecinos más cercanos, se realiza una “votación". En este trabajo, la votación consiste en elegir la moda del vecindario y se asigna ese valor al dígito d a evaluar.

Una gran desventaja de este método de clasificación es que sufre de “la maldición de la dimensión". Este fenómeno ocurre al organizar datos en espacios de múltiples dimensiones, que ocasiona que al aumentar la dimensionalidad, el volumen del espacio aumenta exponencialmente, haciendo que los datos disponibles se vuelvan muy dispersos. En otras palabras, si se quiere mejorar este método considerando este contexto se requiere de una técnica que nos permita reducir la dimensión de la matriz del conjunto de datos.

2.2. Análisis de componentes principales (PCA)

El análisis de componentes principales es utilizado para transformar el espacio de variables que presentan cierto nivel de correlación entre ellas a un espacio donde se definen la misma cantidad de variables pero completamente independientes entre sí. A estas variables se las conoce como “componentes". Estas variables generadas se ordenan según el grado de varianza del sistema que explican. La traza de la matriz generada, que sería la sumatoria de las varianzas de cada componente, representa la varianza total del sistema, debido a que se tratan de variables independientes.

Teniendo n muestras de m variables, se construye la matriz $X \in \mathbb{R}^{n\times m}$, donde cada muestra corresponde a una fila y tiene media cero $x_i = \frac{(x_i-\mu)}{\sqrt{n-1}}$, donde μ es el vector que contiene la media de cada una de las variables (columnas).

A partir de esta matriz, se construye la matriz de covarianzas como $\frac{X^tX}{n-1}$ y, además, la matriz V que tiene como columnas los autovectores de la matriz de covarianzas. Se realiza un cambio de base definido como V^tX^t, que reduce la redundancia entre las variables y asigna a cada muestra un nuevo nombre mediante un cambio de coordenadas. Luego del cambio de base, es posible seleccionar un subconjunto de estos componentes permitiendo simplificar los cálculos utilizando un menor número de variables que expliquen gran parte de la variabilidad del sistema. Esto permite, sobre todo para casos con m grandes, reducir el *overhead* de los cálculos de predicción.

3. Desarrollo

3.1. Estructura de datos

La base de datos utilizada para el entrenamiento de los modelos cuenta con 42000 entradas de imágenes cuadradas de caracteres individuales de 28×28 píxeles (Fig. 1). Cada imagen se condensa para formar una fila dentro de la base de datos con 28×28 columnas. Adicionalmente, hay una que identifica la etiqueta asignada a la imagen (en la Fig. 1 sería "7"). De esta manera, cada columna de la matriz de entrada representa cada píxel de la imagen original representado por un valor equivalente a una graduación de gris en el rango $[0,\ 255]$.

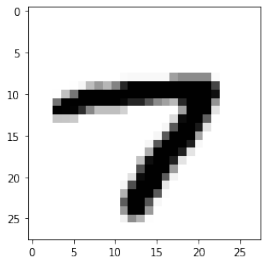

Figura 1: Imagen de la base de datos Kaggle de dimensiones 28x28 píxeles correspondiente a un dígito 7

Para modelar los datos de entrada en nuestro código se utilizó el paquete *Eigen*, utilizando su representación de *Matrix* y *Vector*. Para realizar los cálculos de distancias, autovalores y autovectores se utilizó una representación *double*.

La matriz original y cada una de las submatrices tomadas en los experimentos fueron abstraídas en las clases *KNNClassifier* y *PCA*, que se encargan de manejar la interacción con los archivos de entrada y salida, así como la división entre conjuntos de *training* y *testing* y la predicción de las etiquetas de los datos de prueba.

3.2. Método de la potencia y método de deflación

Para el utilizar el método PCA es necesario conocer los autovalores y autovectores de la matriz de entrenamiento. El método de la potencia permite determinar el autovector asociado al autovalor dominante. La matriz de entrenamiento utilizada en este trabajo es semi definida positiva (sDP), esto asegura que el método pueda converger al autovalor dominante. Además, se requieren de otras consideraciones particulares como la cantidad de iteraciones utilizadas en la aproximación o el criterio de corte (*epsilon*), que evalúa la diferencia entre el valor obtenido para una iteración y la siguiente. Esto permite reducir el costo computacional permitiendo cortar en un menor número de iteraciones sin perdedir precisión.

Listing 1: Metodo de la potencia

```
MetodoPotencia(B, vector_inicial, n_iter) {
    v = vector_inicial

    for (i = 1, ... , n_iter):
        v = Bv / ||Bv||

    a = v.T * B * v /(v.T * v)

    return (a, v)
}
```

Se puede combinar este procedimiento con el método de deflación, que permite obtener así todos los autovectores y autovalores asociados.

Se puede observar que dada una matriz B de dimensión $n \times n$, la matriz B - $\lambda_1 v_1 (v_1^t v_1)$ tiene autovalores 0,λ_2,...,λ_n con autovectores asociados v_1,..., v_n

A esto se le llama deflación. De esta manera, se pueden combinar ambos métodos realizando el cálculo del autovalor y autovector dominantes, actualizando la matriz utilizando la deflación y realizando una nueva iteración para calcular el siguiente autovalor. Luego se puede iterar hasta conseguir todos los autovalores y autovectores asociados.

3.3. Análisis de componentes principales (PCA)

La implementación de PCA cuenta con un valor α que representa el número de componentes al que se reduce la matriz original. El método *fit* recibe la matriz de entrenamiento con la que realiza la clasificación y calcula los autovectores de la matriz para luego utilizarlos para transformar las matrices. El método *transform* es el que recibe una matriz y la transforma al nuevo espacio de variables (componentes principales). A partir del conjunto de entrenamiento, se genera la matriz de covarianzas y se utiliza el método de la potencia junto con el de la deflación para obtener los α autovectores asociados a la matriz de covarianzas en orden decreciente a su varianza. A continuación se describien los pasos del algoritmo:

- A las muestras del conjunto de datos recibido se le resta la media de cada una de las variables (columnas)
- Luego, a partir de la matriz obtenida se genera la matriz de covarianzas como $M_x = \frac{X^t X}{n-1}$
- Se aplica el método de la potencia junto con el de deflación para obtener los primeros α autovectores de M_x, es decir los componentes principales.
- Se devuelve el producto de X.V, siendo V la matriz que tiene por columnas los α autovectores de M_x.

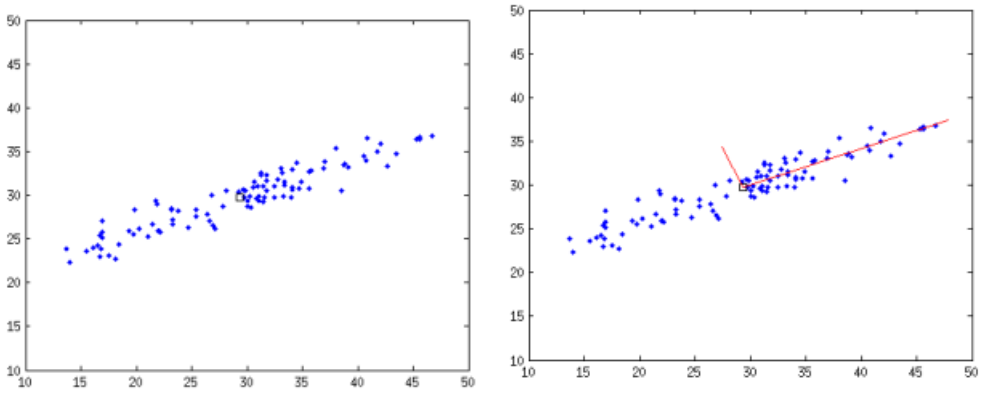

Figura 2: Figuras que muestran cómo actúa PCA en dispersión de datos en R^2. A la izquierda los datos antes de aplicar PCA y a la derecha los componentes que son tomados luego de la aplicación del método.

3.4. K vecinos más próximos (*kNN*)

La implementación de *kNN* cuenta con un valor k que representa la cantidad de vecinos a considerar en los cálculos de distancias. Luego, el método *fit* sólo se encarga de recibir y guardar la matriz de entrenamiento. El método *predict* es el encargado de calcular las distancias de cada elemento a predecir con todos los elementos de entrenamiento para poder clasificar el dígito, *a priori* desconocido. El algoritmo de *kNN* utilizado es bastante directo:

- Al inicializar la clase *KNNClassifier* se le asigna un tamaño de vecindario o *k*.

- Recibe una matriz de entrada sin procesar o que puede encontrarse previamente procesada por PCA para reducir la dimensionalidad a α variables o componentes principales y, además, recibe la instancia a clasificar en la misma base.
- Genera un vector (de tamaño k) de tuplas que contiene los elementos más cercanos a la instancia a clasificar. La tupla contiene primero la distancia (*double*) entre la instancia que se quiere clasificar y cada dato de la matriz de entrenamiento, seguido de la etiqueta (*int*) correspondiente a la clase del elemento de entrenamiento.
- Se inicializa el vector con las primeras k entradas de la matriz de entrenamiento. Se ordena el vector.
- Luego, se recorre el resto de la matriz y para cada elemento se calcula la distancia. Si esa distancia es mayor que la del más lejano de los k más cercanos hasta el momento, se continúa por el siguiente elemento. En caso de tener menor distancia, se lo reemplaza y luego se realiza *bubble sort* hasta encontrar la posición correcta del nuevo vecino más cercano.
- Finalmente se realiza la votación entre los k vecinos más cercanos para clasificar la instancia en cuestión.

3.5. Método de K-Fold cross validation

Cross-validation es un procedimiento de remuestreo utilizado para evaluar modelos de aprendizaje automático en una muestra de datos limitada.

El procedimiento tiene un único parámetro llamado K que refiere al número de grupos en los que se dividirá la muestra de datos. El procedimiento a menudo se llama *K-Fold cross-validation*. Cuando se fija el valor de K en 10, por ejemplo, el método se determina como *10-folds cross-validation*.

Cross-validation se utiliza principalmente en aprendizaje automático aplicado para entrenar un modelo en datos no vistos. Es decir, usar una muestra limitada para estimar cómo se espera que el modelo funcione en general, realizando las predicciones sobre datos que no se usaron durante el entrenamiento del modelo.

Es un método popular debido a su simpleza y porque generalmente da como resultado una estimación menos sesgada de la calidad del modelo que otros métodos que podrían llegar a generar *overfitting*.

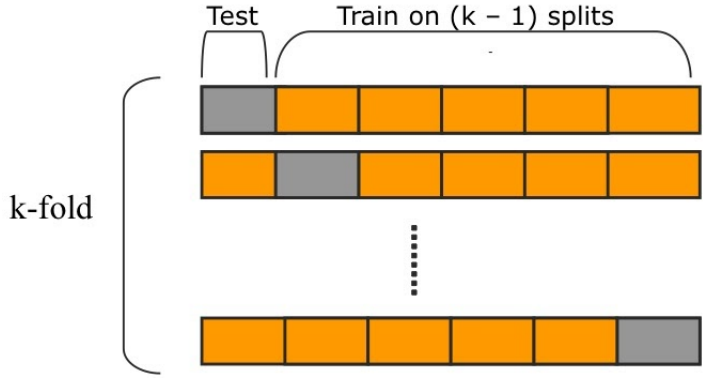

Figura 3: Esquema explicativo del método de *K-Fold cross-validation*.

El procedimiento general es el siguiente:

- Se divide el conjunto de datos en k grupos
- Se elige el grupo para el conjunto de datos de prueba
- A los grupos restantes se los utiliza como un conjunto de datos de entrenamiento.

- Se ajusta el modelo al conjunto de entrenamiento y se lo evalúa en el conjunto de prueba
- Para cada grupo se guarda el puntaje de evaluación y luego se descarta el modelo
- Se obtiene una medida de calidad del modelo utilizando la muestra de puntajes de evaluación.

Es importante destacar que cada observación en la muestra de datos se asigna a un grupo individual y permanece en ese grupo durante la ejecución del procedimiento. Esto significa que cada muestra va a ser utilizada en el conjunto de prueba una vez y se usa para entrenar el modelo k-1 veces.

3.6. Métricas de evaluación de desempeño

Existen diferentes medidas para evaluar el rendimiento de los clasificadores. En este Trabajo Práctico fueron utilizadas tres medidas, *Accuracy*, *F1-Score* y *Kappa de Cohen*.

- **Accuracy**: Los aciertos totales sobre los casos totales. En términos de la matriz de confusión (Fig. 4) corresponde a sumar la diagonal dividido la suma de todas las celdas.
- **F_1-Score**: Dado que accuracy y recall son dos medidas importantes que no necesariamente tienen la misma calidad para un mismo clasificador, se define la métrica F_1 para medir un compromiso entre el recall y la accuracy. La métrica F_1 se define de la siguiente manera.

$$\frac{2 \times accuracy \times recall}{accuracy + recall}$$

- **Kappa de Cohen:** Es una medida para indicar cuánto concuerdan dos clasificadores sobre un mismo conjunto de datos. Esta métrica puede utilizarse para determinar si el problema contiene ejemplos particularmente complicados dado que, por ejemplo, ningún clasificador lo reconoce correctamente. Dicha medida se define de la siguiente manera.

$$\frac{p_o - p_a}{1 - p_a}$$

Donde p_o es la probabilidad observada de que los dos clasificadores concuerden y p_a es la probabilidad aleatoria de que lo hagan.

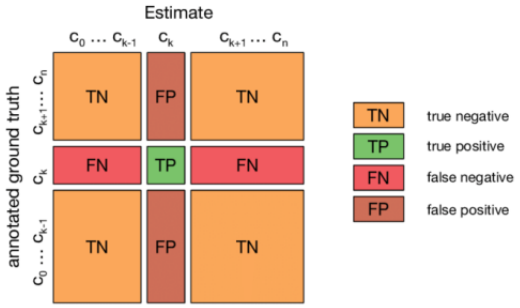

Figura 4: Matriz de confusión utilizada para la construccion de las distintas métricas.

- *Verdadero positivo (**TP**): el número de ejemplos correctamente clasificados de una clase específica (a medida que se calculan estas medidas para cada clase).*
- *Verdadero negativo (**TN**): el número de ejemplos correctamente clasificados que no pertenecían a la clase específica.*
- *Falso positivo (**FP**): el número de ejemplos que se asignaron incorrectamente a la clase espe-*

cífica.

- *Falso negativo (**FN**): el número de ejemplos que se asignaron incorrectamente a otra clase.*

3.7. Análisis exploratorios

Se realizó un análisis previo con el objetivo de acotar el rango de búsqueda de los hiperparámetros que presentaran los mejores resultados para poder reducir el tiempo de cómputo necesario para la ejecución de los experimentos. En el caso general, se buscaría tomar todos los valores posibles del α y así elegir el que genera un mejor resultado. Sin embargo, en este caso particular eso requiere variar el α en todo su rango (desde 2 hasta n) y, además, variar el tamaño de la matriz de entrenamiento sobre la que se realiza la ejecución. Esto no parece algo plausible, dado que el tamaño del conjunto de datos es 42.000 y cada muestra es de dimensión 28×28.

Para acotar el rango de variación del α se aprovecharon ciertas características de los datos de trabajo. Considerando que el objetivo del método PCA es generar nuevas variables para el conjunto de datos que tengan la mayor varianza posible y reduzcan la covarianza entre variables al mínimo, la matriz resultante tendrá en su diagonal las varianzas de cada componente ordenadas según su magnitud y, además, los elementos fuera de la diagonal, que representan la covarianza, serán nulos. Gracias a esto se puede saber, para cierta muestra, cuán representativa es la matriz resultante de la siguiente manera:

> *Sea λ_{ii} uno de los elementos de la diagonal de la matriz resultante del método PCA, el porcentaje de representación de la componente asociada a λ_{ii} se define como $\frac{\lambda_{ii}}{\sum_{j=1}^{n} \lambda_{jj}}$.*

Por lo tanto, tomando los primeros α componentes principales se puede medir cuanta varianza se puede explicar con respecto a todo el conjunto y así decidir, de manera sencilla, una cota para α. En este trabajo se consideró suficiente obtener aproximadamente el 95 % de la varianza total del conjunto obteniendo los α primeros componentes principales.

> *Se mide cuanto representa en términos de porcentaje de la varianza total los primeros α componentes principales y se evalúa que llegue al porcentaje determinado como umbral realizando el siguiente cálculo:*

$$95\,\% \leq \frac{\sum_{j=1}^{\alpha} \lambda_{jj}}{\sum_{j=1}^{n} \lambda_{jj}}$$

De esta manera se puede acotar el rango de α para realizar la experimentación sin riesgo de pérdida sustancial de información.

3.8. Efectos de la elección de la medida de distancia en KNN

En el clasificador kNN, las distancias entre la muestra de prueba y las muestras de datos de entrenamiento se identifican por diferentes medidas. Por lo tanto, estas medidas juegan un papel vital en la determinación del resultado final de la clasificación. La distancia Euclídea es la métrica de distancia más utilizada en las clasificaciones kNN. En este trabajo se intentará hacer un breve análisis comparativo utilizando una familia particular de medidas de distancia.

Se desea elegir la “mejor" métrica de distancia, es decir, aquella que permite realizar las mejores estimaciones mediante kNN.

Medidas de distancia de L_p *Minkowski*: esta familia incluye tres métricas de distancia que son casos especiales de distancia de Minkowski, correspondientes a diferentes valores de p para esta distancia de potencia. Este tipo de distancias tiene una métrica generalizada que se define de la siguiente manera:

$$D_{Mink}(x,y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

donde p es un valor positivo. Cuando $p = 2$, la distancia se convierte en la distancia euclidiana. Cuando $p = 1$ se convierte en la Distancia de Manhattan. Donde x_i y y_i representan al i-ésimo valor en el vector x e y.

3.8.1. Distancia Manhattan

La distancia de Manhattan, también conocida como norma L_1, norma uno, distancia rectilínea o distancia de la cuadra de la ciudad, que formuló Hermann Minkowski en la Alemania del siglo XIX. Esta distancia representa la suma de las diferencias absolutas entre los valores opuestos en los vectores.

$$MD(x,y) = \sum_{i=n}^{n} |x_i - y_i|$$

3.8.2. Distancia de Chebyshev

La distancia de Chebyshev también se conoce como distancia de valor máximo o “distancia del tablero de ajedrez". Esta distancia es apropiada en los casos en que dos objetos deben definirse como diferentes si son diferentes en cualquier dimensión. Es una métrica definida en un espacio vectorial donde la distancia entre dos vectores es la mayor de sus diferencias a lo largo de cualquier dimensión de coordenadas.

$$CD(x,y) = max_i|x_i - y_i|$$

3.8.3. Distancia Euclidea

También conocida como la norma 2 o distancia de la regla, que es una extensión del Teorema de Pitágoras. Esta distancia representa la raíz de la suma del cuadrado de las diferencias entre cada coeficiente de los vectores.

$$ED(x,y) = \sqrt[2]{\sum_{i=1}^{n} |x_i - y_i|^2}$$

3.9. Hipótesis

3.9.1. Relación entre *k* y el tamaño del conjunto de entrenamiento

> ***Hipótesis 1.*** *A partir de cierto punto, seguir incrementando el k con respecto a cierto tamaño del conjunto de entrenamiento va a resultar en una perdida de precisión y un aumento del tiempo de ejecución en la clasificación de elementos.*

Utilizando el análisis realizado para el método PCA, se fijó el parámetro α para reducir la dimensionalidad y así aumentar la velocidad de cómputo. Considerando esto, interesa analizar cómo va a variar el tiempo de ejecución y distintas métricas para medir la calidad de los resultados con respecto a la variación del *k* y el tamaño del conjunto de datos con los que se trabaja.

Se espera observar que para valores de *k* pequeños en relación a la cantidad de elementos en la base de entrenamiento se presenten mejores resultados en términos de calidad y tiempo, puesto que

la elección de k bajos implica que se observen unos pocos elementos de los más cercanos al elemento que se quiere clasificar. Si bien podría suceder que para k muy bajos justo esos elementos cercanos sean *outliers* y la clasificación que se realice no sea buena, se cree que ese no sería el caso general dado que al estar trabajando con un conjunto de datos uniformemente distribuidos estos casos suelen ser poco probables. Luego, se puede asumir que estos casos “patológicos"para k muy bajos no suelen representar gran parte de los resultados estadísticamente. Por otro lado, si se considera tomar el valor máximo de k, es decir toda la base de entrenamiento, se estarán considerando todos los elementos del conjunto como vecinos cercanos y, por lo tanto, los elementos siempre van a ser clasificados en la clase que representa la moda del conjunto de entrenamiento, es decir el elemento que más repetido del conjunto de datos. Es entonces razonable asumir que tomar conjuntos de vecinos cada vez mas grandes puede traer como consecuencia una gran perdida de precisión en los resultados, pues se incrementa la cantidad de casos donde los elementos de otras clases que caigan dentro del radio de vecinos cercanos superen en número a los elementos de la clase a la que verdaderamente pertenece.

3.9.2. Análisis comparativo de kNN con y sin PCA

> ***Hipótesis 2.*** *El método de kNN con PCA va a presentar una mejor perfomance, en términos de calidad y tiempo, que aquel realizado sin realizar PCA.*

Si bien el método PCA debe realizar la reducción de la dimensionalidad de la matriz y esto tiene un cierto costo computacional, lo que se consigue a partir de ello es una reducción en el costo de realizar la clasificación de los elementos. Es decir que a pesar del costo computacional que se paga para ejecutar el método de PCA, este ayuda a mejorar el costo del kNN puesto que se harán las mismas comparaciones que en el método que no utiliza PCA pero el tamaño de la matriz se verá reducido significativamente.

A su vez, utilizar el método PCA junto con kNN mejora uno de los problemas mas significativos del kNN que es la “la maldición de la dimensión". Solucionar este problema en términos de la varianza de los elementos de la matriz de entrenamiento permite acentuar las diferencias entre ellos y así obtener una mejor precisión.

3.9.3. Variación del tamaño del conjunto de entrenamiento para kNN con PCA

> ***Hipótesis 3.*** *A medida que crece el conjunto de entrenamiento van a crecer proporcionalmente el tiempo de ejecución y la calidad de los resultados.*

En líneas generales se puede observar que existe una relación entre el conjunto de datos con los que se trabaja y ciertas métricas a tener en cuenta sobre los métodos utilizados. Analizando principalmente la calidad de los resultados y el tiempo de ejecución, es razonable asumir que el tiempo crecerá proporcionalmente junto con el tamaño del conjunto de datos con el que se trabaja. A su vez, se espera ver una mejor calidad de de predicción al tener más muestras y, por lo tanto, un espacio de vecinos más denso.

En el método PCA, como se menciona anteriormente, se puede obtener una cota para el rango del mismo analizando la cantidad de varianza que acumulan las α componentes principales. Por otro lado, el método kNN selecciona las k muestras mas cercanas al elemento que se intenta clasificar, por lo que cuanto mayor es el número de elementos de la matriz de entrenamiento, mayor es la densidad de elementos. Esto, en combinación con la aplicación del método PCA, afecta directamente en la calidad de “consenso"que se lleva a cabo en el método kNN para decidir la clasificación.

Todo esto lleva a suponer que si bien procesar una mayor cantidad de datos requiere un mayor tiempo de procesamiento, la cantidad de datos disponibles para los métodos mencionados podría ayudar a obtener una mayor precisión en la clasificación.

3.9.4. Análisis de los experimentos con el método de *K-fold*

> ***Hipótesis 4.*** *A mayor cantidad de Folds se espera una disminución en el tiempo de ejecución. A su vez, se espera ver un crecimiento en las métricas de calidad.*

Al aumentar el valor de K se genera un mayor número de subdivisiones sobre la matriz de entrenamiento, por lo que se tendrán matrices de testeo cada vez más pequeñas y, a su vez, se tendrán que realizar un mayor número de repeticiones de la validación cruzada, debido a que se van rotando las submatrices que se utilizan para testear hasta que cada una de esas K subdivisiones haya sido utilizada para testear. De esta manera, para valores grandes de K se realizarán muchos repeticiones pero el costo computacional individual será menor dado que los conjuntos de test serán más pequeños y, por lo tanto, la cantidad de predicciones a realizar será menor.

Considerando todo esto, se espera que la calidad de las estimaciones aumente con el K debido a que la cantidad de elementos contenidos en las correspondientes matrices de entrenamiento será mayor. En contraposición, se espera que los tiempos de ejecución totales aumenten debido al aumento de las repeticiones (tantas repeticiones como el valor de K seleccionado).

3.10. Posible mejora para kNN

Una posible modificación para el método kNN sería realizar una validación previa de la matriz de entrenamiento, donde a cada elemento se le calcula su *validez* observando los n vecinos más cercanos y observando la proporción que pertenece a la misma clase o etiqueta. De esta manera, lo que se logra es clasificar a los elementos de la matriz por su nivel de representatividad de la clase a la que pertenecen. Si imaginamos a los puntos de una clase como una nube de puntos en el espacio, podría ocurrir que dos o más nubes se superpongan. En ese caso, aquellos puntos de una nube que no estén rodeados por puntos de la otra clase representarán mejor a la clase que aquellos que estén rodeados de puntos de diferentes clases. ¿Esto cómo se interpreta? Si yo estoy cerca de un punto menos conflictivo, propbablemente se me asigne mejor la clase.

Por lo que se puede utilizar este coeficiente de validez para hacer una votación pesada (*weighted*) al momento de realizar un *kNN* normal.

$$\mathcal{V}(x) = \frac{1}{k} \sum_{i=1}^{k} Eq(label(x), label(K_i(x)))$$

Donde k es la cantidad de vecinos a considerar, *Eq* es la función que mide la similitud entre las etiquetas, *label(x)* es la etiqueta del elemento analizado y *label(K_i(x))* es el *i*-ésimo vecino del elemento.

4. Resultados y discusión

En la sección anterior se analizaron algunas preguntas que podrían surgir en el ambito teórico. En lo que resta de este trabajo se presentarán algunas simulaciones para los escenarios considerados, métricas que se tuvieron en cuenta y experimentos concretos analizando sus resultados y relacionándolos con las hipótesis ya enunciadas.

4.1. Especificaciones generales

Para contextualizar las métricas, se adjuntan a continuación las especificaciones del equipo en donde se corrieron los experimentos.

- **CPU:** IntelCore i5-6300U CPU @ 2.40GHz
- **Memoria Cache:**
  - L1I 64 KiB 2x32 KiB 8-way set associative
  - L1D 64 KiB 2x32 KiB 8-way set associative write-back
  - L2 512 KiB 2x256 KiB 4-way set associative write-back
  - L3 3 MiB 2x1.5 MiB write-back
- **Memoria Principal:** 8GB RAM DIMM DDR4 Synchronous @ 2400 MHz
- **Memoria Secundaria:** 256GB SATA SC308 Sk Hynix @

Todos los experimentos fueron corridos en *Jupyter Notebook*. Se adjunta el *script* correspondiente para que puedan replicarse fácilmente los análisis realizados, donde a su vez puede variarse los parámetros y datos de entrada fácilmente.

Siguiendo el formato de la cátedra, para correr el programa se debe ejecutar el siguiente comando.

Listing 2: ¿Cómo correr el programa?

```
1 python3 tp2.py -m <method> -i <train_set> -q <test_set> -o <classif>
```

Donde:

- <method>el método a ejecutar (0: kNN , 1: PCA + kNN)
- <train_set>será el nombre del archivo de entrada con los datos de entrenamiento.
- <test_set>será el nombre del archivo con los datos de test a clasificar
- <classif>el nombre del archivo de salida con la clasificación de los datos de test de <test_set>

Nota: los archivos de entrada y salida deberán ser rutas a un archivo en formato csv.

4.2. Análisis exploratorios

Para comenzar la experimentación se decidió hacer un análisis exploratorio de los datos con el fin de conocer las particularidades de la base de entrenamiento a utilizar. En primer lugar, se observó que los datos estaban uniformemente distribuidos, lo que fue tenido en cuenta a la hora de hacer *cross-validation* para evitar *overfitting*.

Dígito	Proporción
1	0.111524
7	0.104786
3	0.103595
9	0.099714
2	0.099452
6	0.098500
0	0.098381
4	0.096952
8	0.096738
5	0.090357

Para aplicar el método PCA resultó interesante analizar con qué valores de α se obtendrían las suficientes componentes principales que acumularan una varianza $\geq 95\,\%$.

En primer lugar se analizó cuánta varianza se obtenía sumando cada componente principal para una matriz de entrenamiento de tamaño 42000. Allí se observó que se presentaba un crecimiento *logarítmico*.

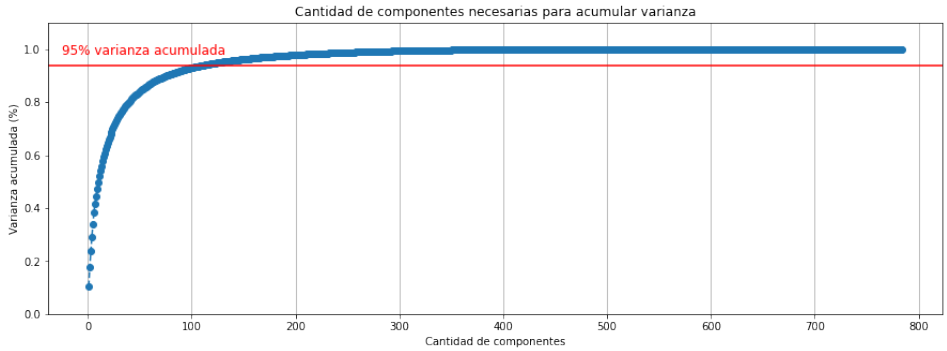

Figura 5: Crecimiento de varianza acumulada según cantidad de componentes principales tomadas para una matriz de entrenamiento de tamaño 42000.

Luego, se decidió analizar qué sucedería para distintos tamaños de matriz de entrenamiento. Para ello, se seleccionaron una serie de tamaños que resultaran representativos y se analizaron los α resultantes:

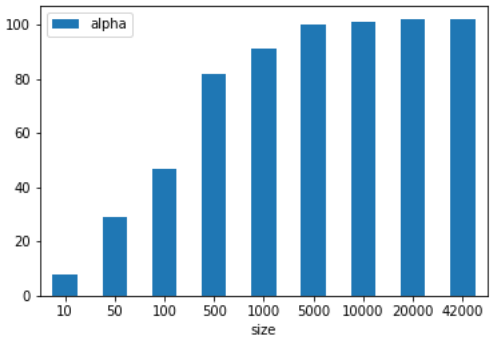

Figura 6: Valores de mínimos de α que permiten representar un 95 % de la varianza obtenidos para distintos tamaños de matriz de entrenamiento.

4.2.1. Efectos de la elección de diferentes medidas de distancia en el método kNN

A pesar de que las medidas utilizadas pertenecen a la misma familia de L_p Minkowski, se puede apreciar como ocurren ciertas variaciones, no muy significativas, dada la similitud entre las medidas utilizadas. Una comparación extra que se realizó fue con la distancia Euclidea, utilizando la provista por la biblioteca de Eigen y una implementada en C++ por nosotros.

Distancia	Time (s)	Accuracy	Kappa	F1 Score
Euclidea (Eigen)	8.843	0.973	0.970	0.973
Euclidea(ED)	17.605	0.973	0.970	0.973
Manhattan(MD)	17.005	0.971	0.968	0.971
Chebyshev(CD)	17.437	0.961	0.956	0.961

En la tabla anterior se puede apreciar que para la comparación de la ED provista por Eigen y la implementada por nosotros existe una reducción de casi la mitad del tiempo de ejecución del método de kNN. Demostrando así como a pesar de tener los mismos valores para las métricas de evaluación del desempeño, tiene un gran impacto en el tiempo de ejecución.

Por otro lado, observando las metricas que fueron implementadas por nosotros se puede ver una variación de entre 0.12 y 0.14 entre las distintas métricas para la calidad de los resultados, obteniendo el peor resultado para CD y el mejor para ED en terminos generales. A pesar de que son variaciones menores, es un dato significativo considerando que son medidas de distancia muy similares y que pertenecen a la misma familia. Esto muestra como podría impactar la utilización de las medidas de distancia en un método como kNN, siendo estas un factor muy pocas veces considerado para la perfomance de este tipo de métodos.

4.3. Experimentos

4.3.1. Relación entre *k* y el tamaño del conjunto de entrenamiento

Para analizar el rol del hiperparámetro *k* respecto al tamaño del conjunto de entrenamiento se dividió en 2 *folds* para hacer *cross-validation* y se utilizó un rango con crecimiento exponencial para *k* definido de la siguiente manera:

$$k = \lfloor i + 1{,}2^i \rfloor$$

Donde i es la iteración actual. El ciclo se corta cuando se deja de cumplir que $k \leq size$.

Esta forma de definir a *k* permitió obtener muchos valores de *k* bajos y un crecimiento exponencial que permitiera analizar a grandes rasgos lo que sucedería para valores grandes sin necesidad de sufrir grandes tiempos de ejecución.

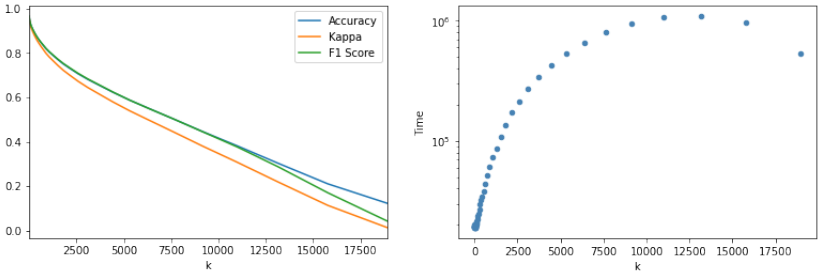

Figura 7: Análisis de las métricas de calidad y tiempo del método de kNN con PCA para diferentes valores de *k* tomando K-Fold = 2.

Los resultados fueron contundentes y esperables. Para grandes valores de k la calidad de los datos disminuyó significativamente, llegando al punto de ser opciones completamente descartables. A su vez el tiempo de cómputo se vió gravemente afectado puesto que la cantidad de comparaciones necesarias entre un elemento a predecir y la base de entrenamiento crecía conforme al k.

4.3.2. Análisis comparativo de kNN con y sin PCA

Gracias al análisis preliminar realizado sobre el α, se pudo acotar el rango sobre el que varía el mismo para la ejecución de PCA. A su vez, considerando los resultados del experimento anterior se pudieron establecer rangos bajos de k que permitieran reducir los tiempos de cómputo de los experimentos y que fueran realmente significativos en términos a lo que se intenta observar.

Para este experimento se utilizaron 5 Folds. Esto permitió tener conjuntos de entrenamiento grandes (y similares la base original) y una buena cantidad de grupos que permitieran evitar sesgos. A su vez, se tomaron rangos para k y α de la siguiente manera:

$$k = [1, 5, 10, ..., 45, 50] \; y \; \alpha = [10, 15, ..., 95, 100].$$

Para el caso de PCA no se computó el tiempo que tardaba en hacer la transformación puesto que los autovectores se calculaban 1 sola vez por Fold, ya que se podían utilizar submatrices correspondientes a cada α.

A continuación se presentan los resultados para el método kNN con PCA.

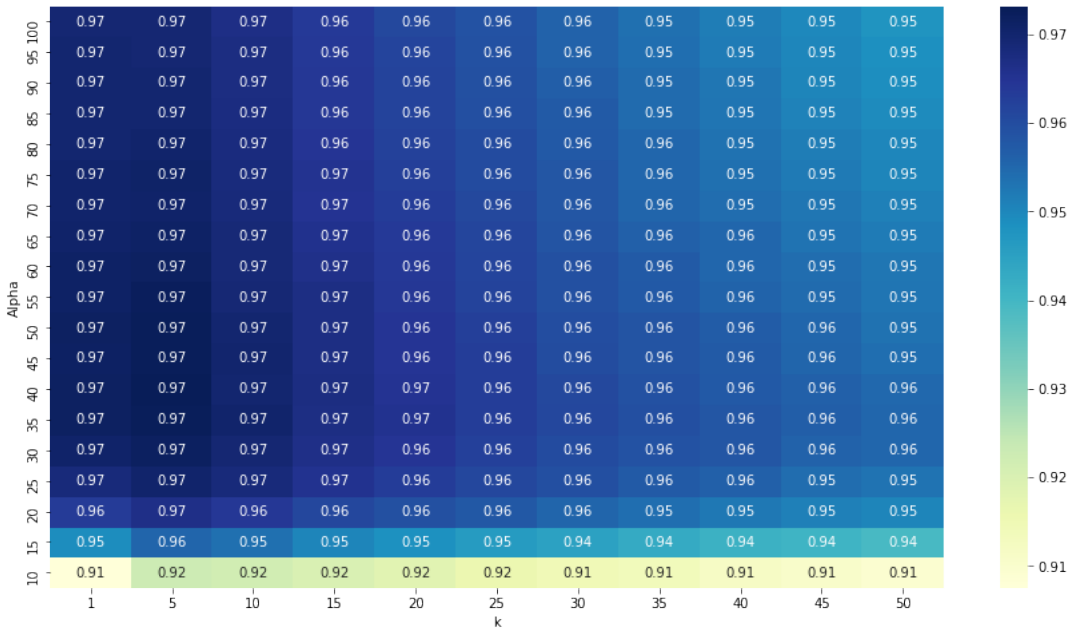

Figura 8: Accuracy del método kNN con PCA para distintos valores de k y α

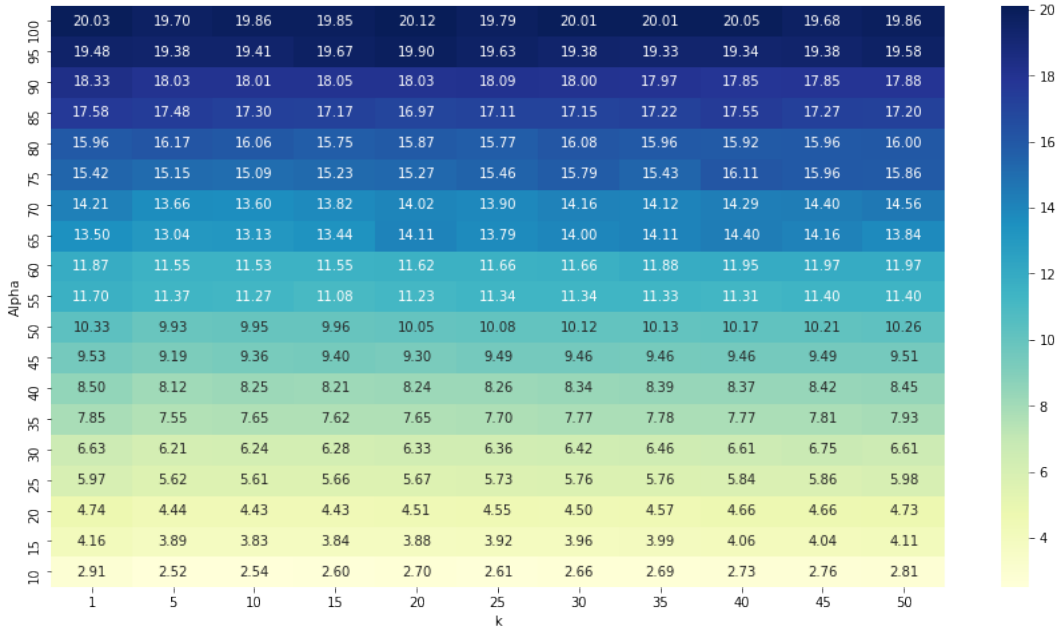

Figura 9: Costo temporal (segundos) del método kNN con PCA para distintos valores de k y α

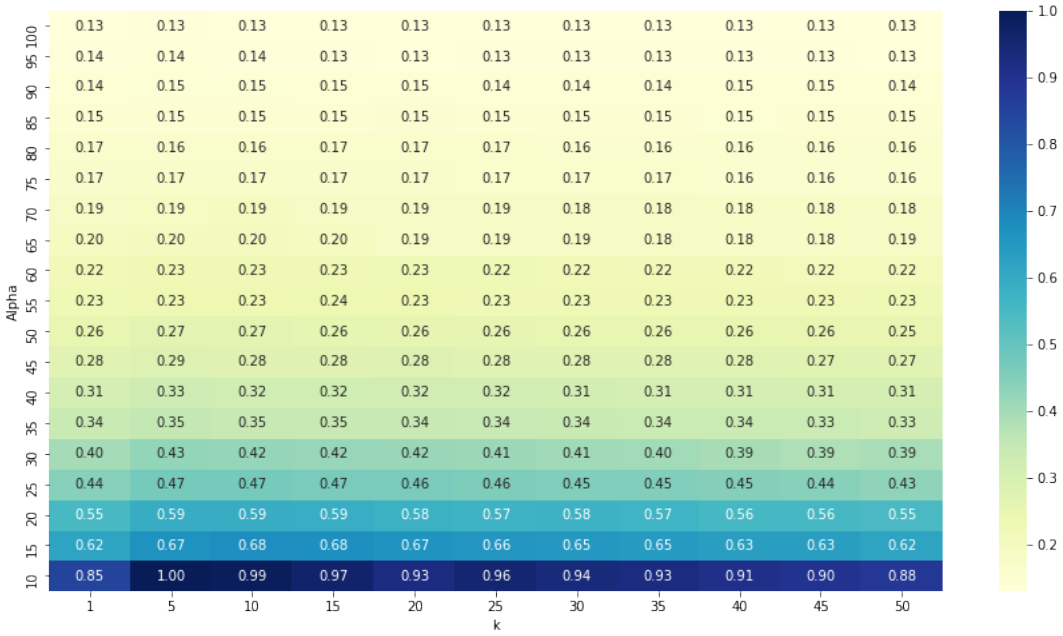

Figura 10: Relación accuracy-tiempo del método kNN con PCA para distintos valores de k y α

Gracias a un detallado análisis se pudo observar que la mejor combinación de parámetros para obtener el mayor accuracy fue $\alpha = 40$ y k $= 5$. Resultó particularmente interesante que a pesar de que según los resultados preliminares se necesitaría $\alpha \approx 100$ para acumular la varianza necesaria para obtener resultados precisos, los mejores resultados se obtuvieron con un α menor. De lo que se desprende la posibilidad de que no sea necesario acumular tanta varianza para obtener resultados óptimos.

Para comparar ambos métodos se requirió seleccionar un α particular y contrastar los resultados. Luego, y considerando el análisis previo, se tomaron los datos correspondientes a $\alpha = 40$ y se obtuvieron las siguientes mediciones que se observan en la Fig. 11.

Los resultados fueron sin duda contundentes. El método de kNN con PCA resultó significativamente mejor en términos de calidad y tiempo. Incluso la relación entre ambas variables a considerar (calidad y

tiempo) fue significativamente mayor, lo que da a entender que el método de kNN con PCA resulta mejor no solo en términos de calidad y tiempo sino también en su relación costo-beneficio.

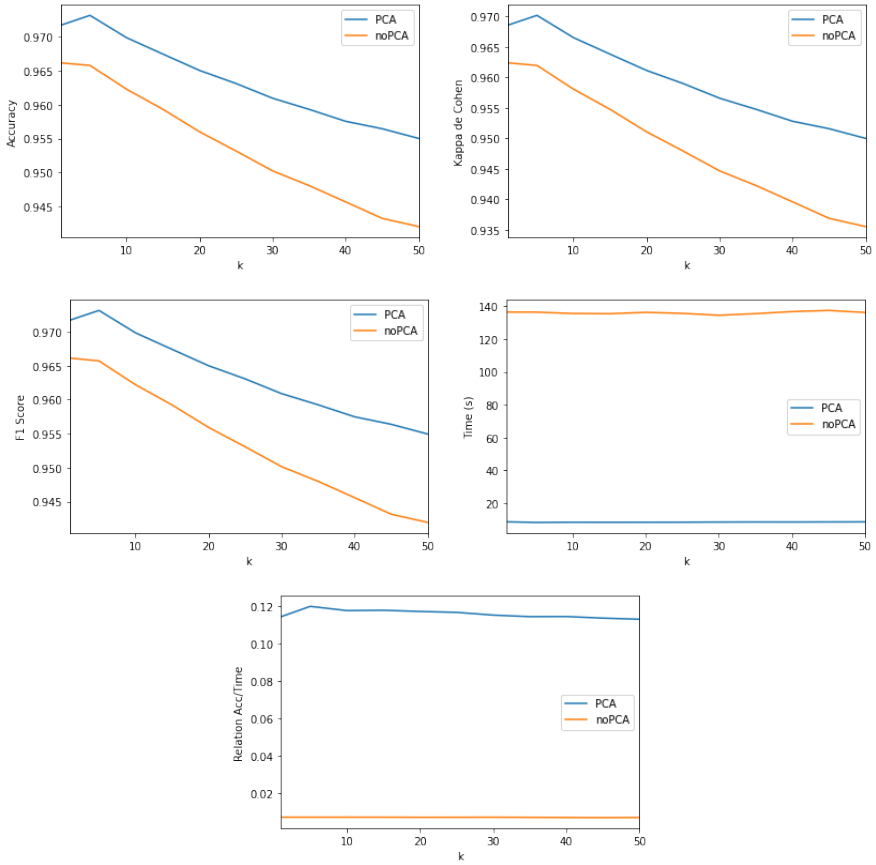

Figura 11: Análisis de las métricas (Accuracy, Kappa de Cohen y F1-Score) de calidad de resultados y tiempo de ejecución para el algoritmo kNN con y sin PCA. Además se representa la relación entre la calidad y el tiempo obtenidos para ambos casos. Se toman los resultados del método con PCA con $\alpha = 40$ y se grafican valores de k entre 1 50 para K-Fold $= 5$

4.3.3. Variación del tamaño del conjunto de entrenamiento para kNN con PCA

Para analizar el desempeño de la implementación en términos de tiempos y calidad frente al crecimiento del tamaño de la matriz de entrenamiento, se utilizaron los hiperparámetros obtenidos del análisis exploratorio y del experimento anterior para intentar obtener óptimos resultados.

Se seleccionó un rango de tamaños, se fijaron hiperparámetros y se pudo observar un fuerte crecimiento en los valores de ambos aspectos.

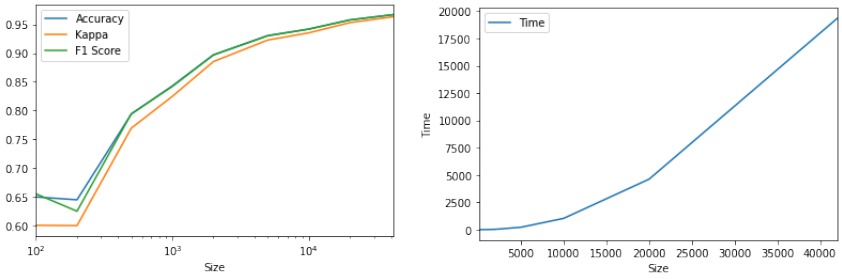

Figura 12: Análisis de las métricas (Accuracy, Kappa de Cohen y F1-Score) de calidad de resultados y tiempo de ejecución al variar el tamaño del conjunto de entrenamiento con PCA. K-fold: 5

A su vez, resulta interesante analizar hasta qué punto el crecimiento de la matriz de entrenamiento permite una buena relación entre la calidad de los datos obtenidos y el tiempo de ejecución.

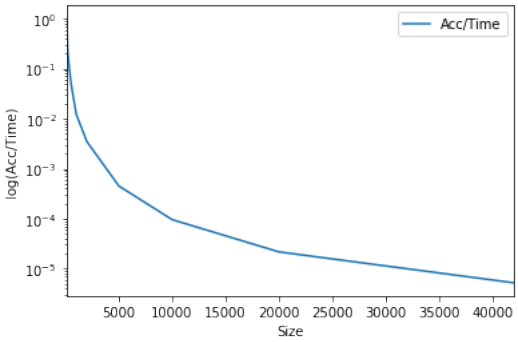

Figura 13: Relación Accuracy-Tiempo del método de kNN con PCA según tamaño de matriz de entrenamiento.

Se puede observar en la figura 13 que la relación entre ambas variables cae exponencialmente, de lo que se puede deducir que ésta relación es buena solo al principio. Sin embargo, al ser tan mala la calidad de los datos obtenidos para tamaños pequeños permite establecer un criterio de corte claro, dado que una vez obtenida una calidad aceptable (en términos de accuracy, por ejemplo) para un tamaño en particular, agregar más elementos a la matriz de entrenamiento produciría un mayor tiempo de cómputo en comparación a la mejora de calidad obtenida, lo que puede ser de gran interés en algunas aplicaciones de kNN.

4.3.4. Análisis de los experimentos con el método de *K-fold*

Para analizar lo que sucedía al variar la cantidad de Folds utilizados para hacer *cross-validation* se utilizaron los hiperparámetros óptimos y un rango de tamaños de Fold para una matriz de entrenamiento de tamaño 42000. Aquí se pudo evidenciar que al utilizar un conjunto de tests cada vez más pequeño por la mayor cantidad de Folds, la cantidad de predicciones necesaria fue menor y, por lo tanto, los tiempos de ejecución se redujeron significativamente. A su vez, se puede observar un crecimiento en la calidad de los datos, que resultaba esperable considerando que al tener una matriz de entrenamiento más grande se esperarían resultados similares a los del experimento anterior.

A continuación se presentan los tamaños de Fold usados y los resultados del experimento:

Fold	Train Size	Test Size
2	21000	21000
5	33600	8400
10	37800	4200
15	39200	2800
20	39900	2100

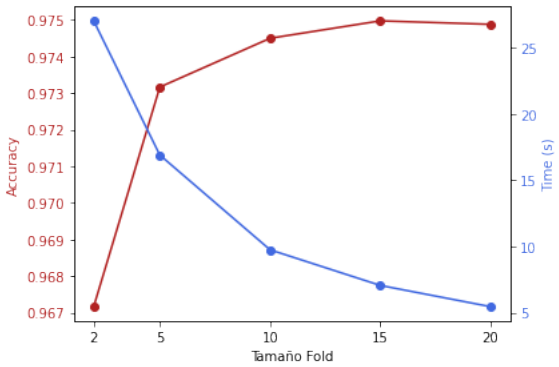

Figura 14: Análisis de Accuracy y tiempo del método kNN con PCA con hiperparámetros $k = 5$ y $\alpha = 40$ para distintos tamaños de Fold.

Si bien el valor máximo de *Folds* que puede tomarse es *N* (*N-1* elementos en la matriz de entrenamiento y 1 elemento de testeo), vemos que esto no tiene mucho sentido puesto que el *accuracy* crece logarítmicamente, por lo que lo que se gana en términos de precisión puede resultar, en algunos casos, despreciable.

Resulta relevante preguntarse cuándo es conveniente utilizar el método de *K-Fold* con respecto a no utilizarlo. En este sentido, y en base a las observaciones realizadas, resulta menos conviente utilizar este método cuando nuestra base de datos de entrenamiento es reducida, puesto que quitar elementos de ella para testearla podría resultar dramático. A su vez, se podría decir que *K-Fold* solo sirve para hacer un *testing* general y no para uno específico. Para ello existen otras herramientas (no contrapuestas a *K-Fold* sino complementarias) que permiten realizar este tipo de testing de manera eficiente.

En términos generales se seleccionó a 5 como cantidad de *Folds* para las experimentación. Esto se debe a que en este punto se obtiene una muy buena calidad de datos sin que esto conlleve un gran tiempo de cómputo (notar que para *cross-validation* se suman los tiempos de ejecución de cada *Fold* pues no se analiza aisladamente).

4.4. Análisis global y elección de hiper parámetros óptimos

A partir de los resultados de la experimentación se pudieron obtener los mejores hiperparámetros para esta base de datos. Se comparó cada una de las métricas utilizadas y fue clara la superioridad del método de kNN con PCA por sobre el método sin PCA.

Metodo	Time (s)	Accuracy	Kappa	F1 Score
Sin PCA	136.545	0.966	0.962	0.966
Con PCA	8.119	0.973	0.970	0.973

5. Conclusiones y trabajo futuro

A partir del análisis exploratorio fue posible determinar un número máximo óptimo de (α) para el análisis de componentes principales (PCA) dependiendo el tamaño de la matriz de entrenamiento utilizando un criterio de corte del 95 % de la varianza acumulada. Así pudo restringirse el rango de búsqueda.

Se obtuvieron mejores clasificaciones al observar las tres métricas de calidad de resultados para el método kNN con PCA, aunque se observa que las diferencias entre ambos métodos son relativamente bajas. Sin embargo, al analizar el tiempo que lleva correr cada método, se observa una diferencia sustancial, siendo el método sin PCA 7 veces más lento (20 a 140 segundos). Al realizar un análisis de la relación entre estas medidas (calidad y tiempo), se observó que el método con PCA representa la mejor decisión al obtener mejores resultados en ambos tipos de métricas.

Se pudo observar que cuando se fijan los valores de los hiperparámetros (k y α) aumenta la calidad de los resultados y el tiempo requerido en la corridas al aumentar el tamaño de la matriz de entrenamiento. Al realizar una estimación de calidad ganada en relación al tiempo, se observa que cae abruptamente, por lo que se recomienda establecer un umbral de calidad y buscar el tamaño mínimo de matriz que lo cumple para evitar aumentar innecesariamente el tiempo de corrida.

Al utilizar la técnica de *cross validation* se observa que tomar valores de K mayores lleva a mejores estimaciones y consume menos tiempo por cada corrida, aunque aumenta la cantidad de iteraciones. A partir de $\mathrm{K} = 5$ se observa que no se modifica significativamente la calidad de los resultados, por lo que se decide realizar la experimentación con dicho valor.

La elección de una medida de distancia entre elementos tienen un impacto directo en las métricas de desempeño y en los tiempos de ejecución. Es por ello que se considera importante elegir aquella que mejor se adapte a cada trabajo. En este trabajo se obtuvo que la distancia euclídea implementada por el paquete *Eigen* es la que presenta mejores resultados. A futuro, podría considerarse la utilización y comparación de un mayor número de medidas distintas, contenidas dentro de distintas familias. Por ejemplo, *squared chord distance* utilizada por paleontólogos, *chord distance*, distancias medidas por productos internos, *Shannon entropy distance* y otras.

En conclusión, se observa que la mejor combinación de hiperparámetros para el conjunto de datos estudiado es $\mathrm{k} = 5$ y $\alpha = 40$.