DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Trabajamos y nos divertimos

11 de julio de 2022 — Métodos Numéricos

Integrante	LU	Correo electrónico
Embon, Eitan	610/20	eembon1@gmail.com
Imperiale, Luca	436/15	luca.imperiale95@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
https://exactas.uba.ar

Índice

1. Introducción

1.1. Problema a resolver

En este trabajo práctico vamos a tratar de resolver un problema de reconocimiento de dígitos manuscritos. El objetivo de nuestro trabajo es crear un clasificador que pueda lograr resolver el problema.

Vamos a desarrollar un algoritmo de clasificación supervisada que, a través de un conjunto de datos **conocidos** deberá ser entrenado, y luego poder clasificar conjuntos de datos **desconocidos**.

El foco de este trabajo esta puesto en un método de **OCR** (Optical Character Recognition), clasificación por vecinos más cercanos con reducción de la dimensionalidad.

Algunas aplicaciones famosas de esta herramienta, son por ejemplo, el reconocimiento de patentes de autos, utilizado muy ampliamente en el ámbito de la seguridad.

1.2. Modelo

1.2.1. kNN

El algoritmo kNN es un método de clasificación supervisada que sirve para estimar la clase de un punto nuevo, respecto a a información proporcionada por el conjunto de prototipos.

En nuestro caso los puntos serán las imágenes, y las dimensiones serán la cantidad de píxeles. Una imagen, o punto, es asignada a un determinado numero, o clase, si esta última es la mas frecuente entre los k vecinos mas cercanos. Para esta distancia se toma simplemente la distancia euclídea.

Notemos la importancia del parámetro k, que al final va a ser el que decida a que dígito pertenecerá cada imagen, sobre todo en las fronteras de las clases.

1.2.2. PCA

El método PCA consiste en cambiar la base del conjunto do datos de entrada a una base ortogonal donde las dimensiones individuales de los datos no están correlacionadas. Al hacerlo se pierde información del mismo, quedándose con las componentes más importantes.

Para nuestro problema esto significará quedarnos con un subconjunto de píxeles de la imagen, y predecir el dígito en base a eso. Esto permitirá que el algoritmo performe mucho mejor, ya que no tendrá que hacer medir la distancia en tantas dimensiones.

A su vez, este método tiene un parámetro α, el cual determina con cuantas dimensiones nos estamos quedando, y estudiaremos como afecta el mismo a la calidad de los resultados y al tiempo que tarde el algoritmo.

2. Desarrollo

Los algoritmos se implementaron en C++, utilizando la biblioteca Eigen para modelar matrices, y Pybind para exportar las clases a Python para la experimentación.

2.1. Métricas

Las métricas utilizadas en el trabajo para analizar la calidad de los resultados serán la *accuracy*, la *precision*, el *recall* y el F1-*Score*.

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

$$accuracy = \frac{T}{T+F} \tag{1}$$

Antes de definir F1-*Score*, primero se definen la *precision* y el *recall*, ambas métricas usadas en clasificaciones binarias. Llamando TP(*True Positive*) a las predicciones acertadas que predicen que cierta imagen pertenece a la clase, TN (*True Negative*)a las predicciones acertadas que predicen que cierta imagen no pertenece a la clase, FP(*False Positive*) a las predicciones desacertadas que predicen que cierta imagen pertenece a la clase, y FN (*False Negative*) a las predicciones desacertadas que predicen que cierta imagen no pertenece a la clase; podemos definir,

$$recall = \frac{TP}{TP+FN} \qquad precision = \frac{TP}{TP+FP} \tag{2}$$

La *precision* es una métrica que mide la cantidad de predicciones acertadas sobre el conjunto de predicciones positivas de una clase. A su vez, el *recall* mide la cantidad de predicciones acertadas sobre el conjunto total de datos de una clase.

Dado que estas son medidas importantes, que no necesariamente tienen la misma calidad para un mismo clasificador, se define la métrica F1-*Score* como un compromiso entre ambas, de la siguiente manera,

$$F1 - Score = \frac{2 * (precision + recall)}{precision + recall} \tag{3}$$

2.2. K-Fold

Se utlizó K-Fold Cross Validation para testear los algoritmos implementados de la manera más insesgada posible.

En este método los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto (K-1) como datos de entrenamiento. Este proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado.

2.3. Dataset

Antes de proceder a la experimentación con los algoritmos implementados, exploramos un poco los datos que tenemos para entrenar.

Lo mas importante es que las etiquetas estén bien distribuidas entre los 10 dígitos, cosa que podemos ver en la figura 1.

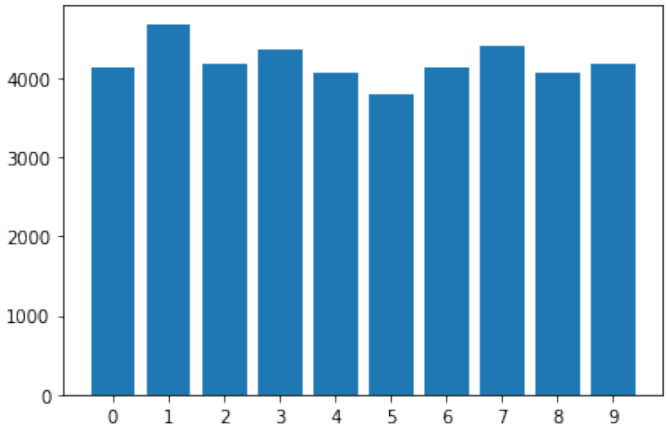
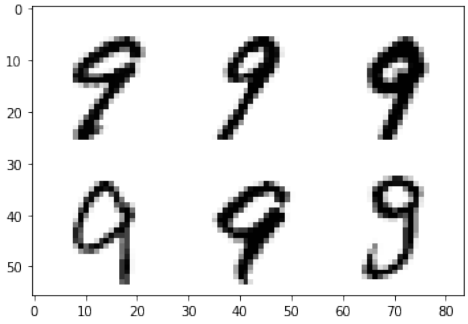


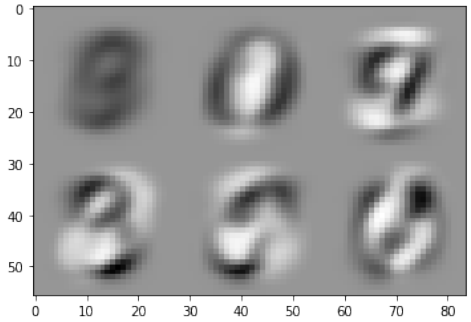
Figura 1: Distribución de dígitos

A continuación veremos algunos ejemplos de imágenes del dataset. También se aprecia una imagen con los seis autovalores obtenidos luego de aplicar PCA con $\alpha = 6$ al dataset.

(a) Muestras de la clase 4



(b) Muestras de la clase 9



(c) Ejemplos de autovalores de PCA para $\alpha = 6$

Figura 2: Ejemplos de imágenes del dataset.

3. Experimentación

3.1. Reproducción de experimentos

Los experimentos fueron realizados en la siguentes condiciones:

- Ubuntu 20.04.4 LTS
- Intel(R) Core(TM) i3-10100 CPU @ 3.60GHz
- 16 GB RAM
- GCC version 9.4.0 compilada con -03 flags
- Python 3.6.5

Los algoritmos del módulo *metnum*, a saber, *PCA*, *KNN*, *método de la potencia* y *deflación* se programaron en C++, en la carpeta *src*. El módulo es luego importado desde el archivo *experimentacion-checkpoint.ipynb* como un módulo de Python usando el submódulo "pybind 11 v2.2.4". Además, dentro de los algoritmos compilados con C++ se utilizó el submódulo *eigen-git-mirror v3.3*.7.

Para correr los experimentos se puede acceder a la carpeta *notebooks* y correr las celdas del archivo *experimentacion-checkpoint.ipynb*, desde la primera en adelante. En el archivo *metricas.ipynb* se pueden correr los los graficos de los experimentos que tomamos y que estan en este informe. También se puede acceder a dichos archivos, usando jupyter notebooks.

3.2. Variación en la cantidad de vecinos de KNN

En este experimento, variamos el parámetro k de **KNN** dejando los demás parámetros estáticos. El objetivo es ver como afecta al análisis cualitativo del clasificador el número de vecinos elegidos para el algoritmo de KNN. Además quisiéramos ver la relación entre calidad por aumento del tiempo. Es decir, cuanto estoy dispuesto hacer durar el experimento en pos de una mejor calidad en la clasificación.

En la siguiente figura, hicimos un experimento de una muestra de diez mil imágenes del dataset original y variamos la cantidad de vecinos desde uno hasta ocho mil. Además no usamos los algoritmos de PCA ni *cross validation*.

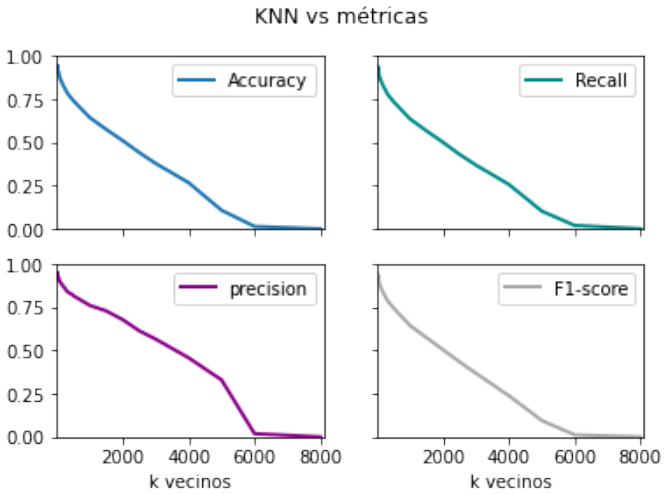


Figura 3: KNN contra métricas

Los valores de k tomados fueron: $[1, 30, 50, 100, 150, 500, 1000, 5000, 8000]$

Podemos observar que hay una tendencia a la reducción en acierto en las métricas cuando el número de vecinos es alto.

Podríamos hacer una inferencia temprana, a partir de estos datos observados, y deducir que, al ser tan "diferentes"las clases entre si, o al estar muy espaciadas en el espacio, para un número pequeño de vecinos tomados, hay mayor probabilidad de que estos sean la clase a la que pertenece la imagen clasificada. Y al agregar mas vecinos, podríamos estar agregando ruido", es decir otras imágenes que al estar "mas lejos"pertenecerían a otras clases, las cuales interferirían en la votación de una forma no deseada.

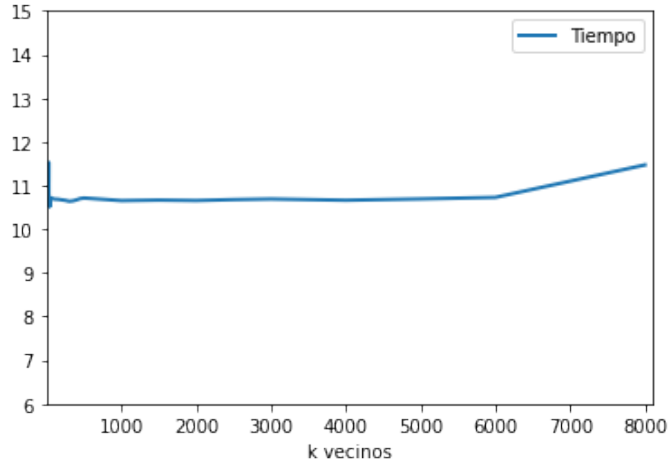


Figura 4: KNN contra tiempo (en segundos)

En la figura 4, podemos observar la relación entre el tiempo y el k de KNN. Vemos que es una función constante, con un ligero ruido en los primeros valores. Podemos decir entonces que el aumento en el número de vecinos tomados por KNN no infiere en un aumento de tiempo considerable.

3.3. Variación en la reducción de dimensiones con PCA

En esta experimentación variamos la cantidad de dimensiones con las que nos quedamos luego de reducirlas con el algoritmo de PCA. Es decir, variamos el parámetro α en del algoritmo PCA.

Usamos un dataset de diez mil imágenes, se usa el algoritmo de KNN con $k = 10$. Éste es un k genérico, ya que todavía no hablamos de optimizar ningún parámetro. Tampoco se usa cross validation.

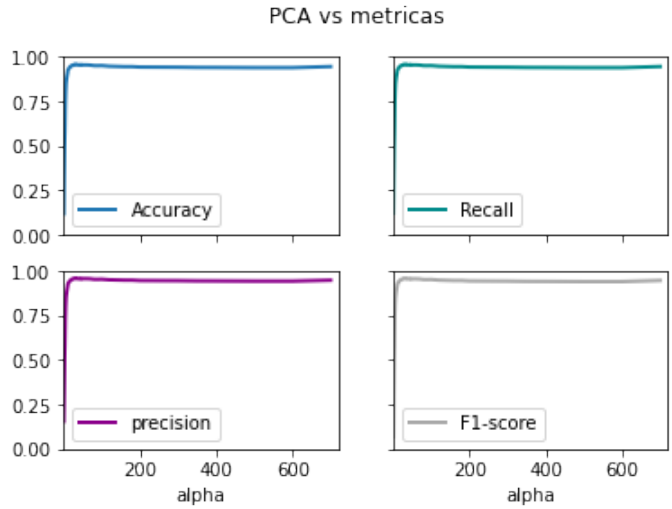


Figura 5: PCA contra métricas

Los valores que se tomaron para α son: $[1, ..., 60, 75, 90, 100, 200, 300, 400, 500, 600, 700]$

Podemos observar en la figura 5 que el valor de las métricas parece permanecer estable con α mayores a 30, pon lo cual centraremos nuestro estudio en esa región. En la figura 6 se ven dichos valores.

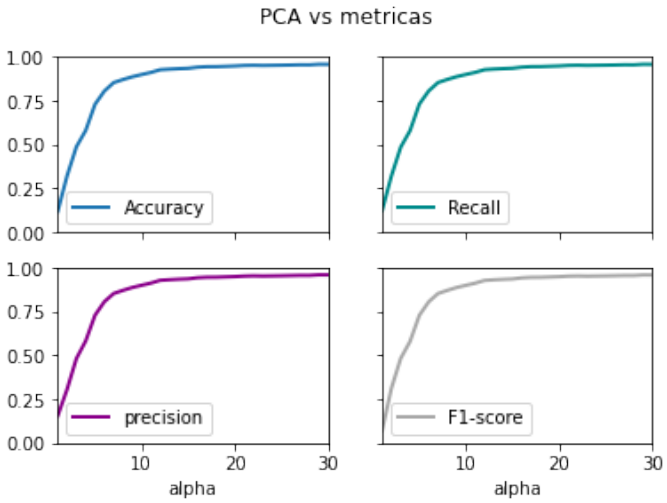


Figura 6: PCA contra métricas, con α tomados menores a 30

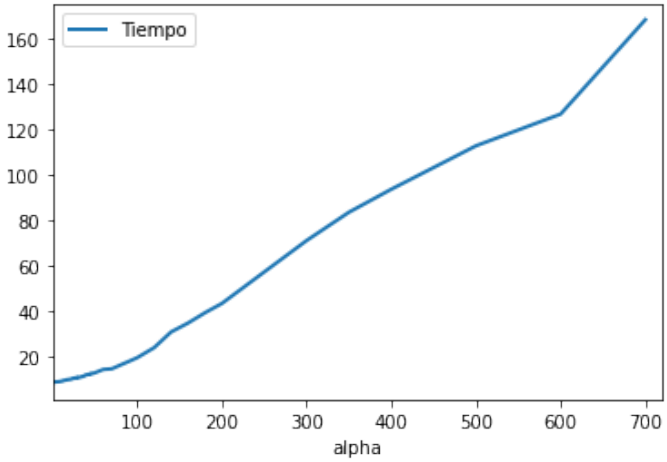


Figura 7: Tiempo (en segundos) de algorimo PCA

Finalmente, vemos en la figura 7, el tiempo empleado en función de éste parámetro. Vemos que aumenta linealmente con α.

Esto inferimos que sucede porque, dentro del algoritmo de PCA, a medida que aumentamos el alpha, las componentes principales que se calculan con el método de la potencia, tienden a estar mas cerca del cero. Con lo que el método de la potencia tarda mas iteraciones en converger, y por ende, aumenta el tiempo total del algoritmo.

3.4. Optimización de k

Una vez ya explorado como varían las métricas y el tiempo con respecto al parámetro k, nos propusimos optimizarlo.

Para ello hicimos un experimento con valores de k mas granulares que en el anterior, centrados en la región con las métricas mas altas. Los valores fueron de $[1, ..., 30]$, sin usar ni PCA ni $KFold$. También se utilizó el dataset completo, a diferencia de los experimentos anteriores.

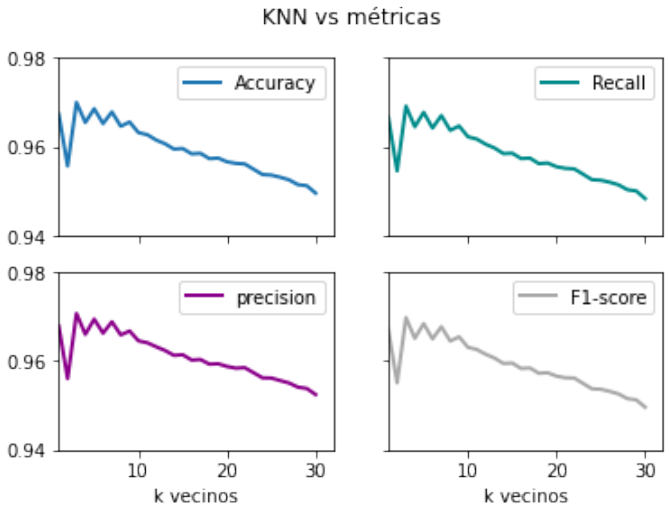


Figura 8: Métricas KNN

Los resultados del rendimiento máximo para cada métrica y un k determinado fueron:

- $Accuracy = 0{,}9746$, para $k = 3$
- $Presicion = 0{,}9743$, para $k = 3$
- $Recall = 0{,}9746$, para $k = 3$
- $F1 - score = 0{,}9746$, para $k = 3$

Podemos concluir entonces que el k óptimo es 3.

También es interesante ver que no hubo diferencias significativas en el tiempo que tomó correr el algoritmo, al igual que en 3.2. Se puede ver en la figura 9, valores similares de tiempos con el k óptimo que con los otros k probados.

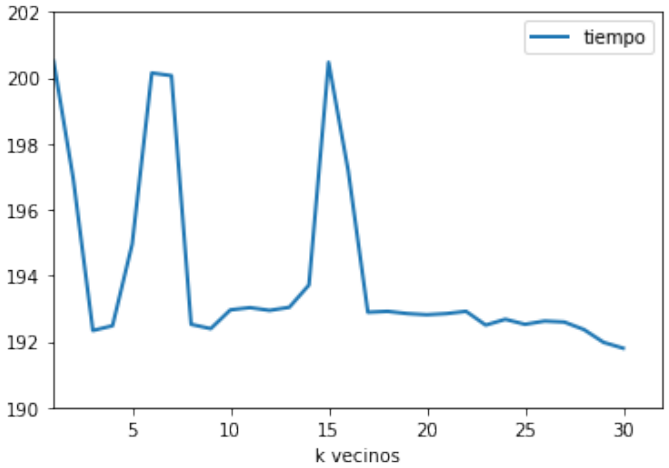


Figura 9: Tiempos KNN

3.5. Optimización de α

Ahora nos propusimos optimizar el siguiente parámetro, α.

Para esto, volvimos a correr un experimento con el dataset entero, pero esta vez usando PCA, y con el k óptimo ya obtenido. El rango de α's probado fue de $[1, ..., 90]$, por lo visto anteriormente en el experimento 3.3.

Los valores que arrojó fueron los presentados en la figura 10.

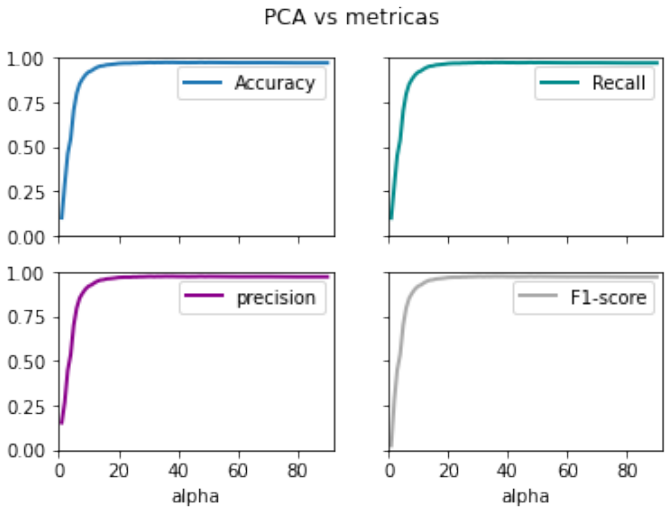


Figura 10: Métricas KNN + PCA, con $k = 3$

Los resultados del rendimiento máximo para cada métrica y un α determinado fueron:

- $Accuracy = 0{,}9746$, para $\alpha = 36$
- $Presicion = 0{,}9743$, para $\alpha = 36$
- $Recall = 0{,}9746$, para $\alpha = 36$
- $F1 - score = 0{,}9746$, para $\alpha = 36$

3.6. PCA vs KNN

En esta sección de los experimentos, realizamos un entrenamiento de los datos con los mismos parámetros que en el experimento 3.4, pero seteando el parámetro alpha en 36, el mejor obtenido. Luego vamos a comparar los dos experimentos en las métricas y el tiempo que tardaron para poder inferir algunas conclusiones a partir de lo observable.

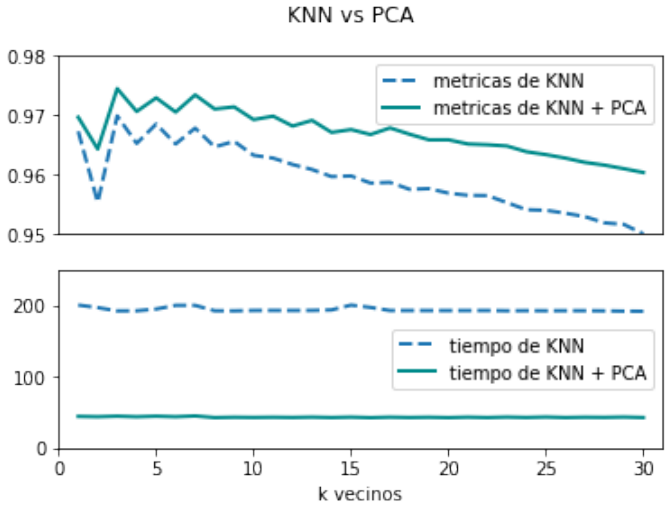


Figura 11: KNN vs PCA.
Parametros: $k = [1..,30]$, $K = 1$ y $alpha = 36$ solo en el caso de PCA.
La figura de arriba grafica el parámetro k contra el promedio de las las métricas (accuracy, F1-score, presicion, recall).
La segunda figura grafica el parámetro k contra el tiempo.

En el caso donde se tomó KNN mas PCA, el mejor rendimiento para cada métrica fue en los siguientes valores de k.

- $Accuracy = 0{,}9700$, para $k = 3$
- $Presicion = 0{,}9706$, para $k = 3$

- $Recall = 0{,}9691$, para $k = 3$
- $F1 - score = 0{,}9697$, para $k = 3$

Queda esclarecido que incluir *PCA* en el entrenamiento para unos mismos parámetros (en este caso los ya mencionados en la figura 11) mejora las métricas, y el tiempo es drásticamente menor.

Sumar al método de entrenamiento *PCA* siempre va a conllevar que el tiempo que tarda en realizarse sea reducido ya que estamos quitando dimensiones al problema y por ende bajando la cantidad de cálculos que tiene que realizar el algoritmo. Pero no necesariamente podría ser el mismo caso en el rendimiento de la predicción de la base de datos de test. En nuestro caso podría deberse a que el dataset esta muy equilibrado en las clases y dentro de las mismas no hay demasiada diferencia una de las otras, con lo cual reducir dimensiones no es muy problemático.

3.7. Validación a través de Cross Validation (K-fold)

En este apartado vamos a tratar de validar los datos anteriores a través del método conocido como *cross validation*. Para esto vamos usar los mejores parámetros hallados anteriormente y correr el experimento para distintos valores de K del método *K-fold*, y medimos el promedio y el desvío estandar de las métricas de todas las intancias de *cross validation* para cada K especifico.

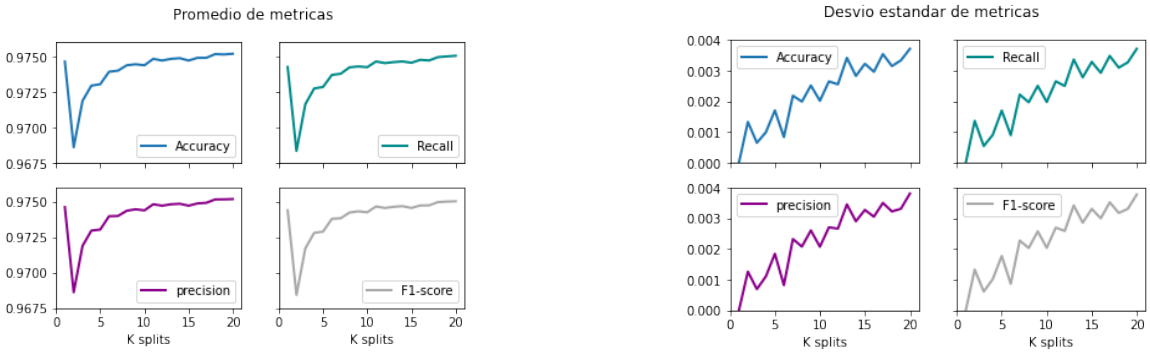


Figura 12: Parámetros: Se tomó el dataset de 42000 imágenes, $k = 3$, $alpha = 36$, $K = [1..,20]$.
La figura de arriba representa el premedio y el desvío estandar de cada métrica en las diferentes instancias de *cross validation* en función de la variación de K.

Creemos que tomar un rango de valores del dos al veinte para el parámetro K es suficiente para ver el comportamiento al variar dicho valor. Cuando aumenta el valor de K aumenta el tamaño de la porción de datos usada para el entrenamiento en cada instancia de *cross validation* y disminuye la porción de la base de datos usada para el testeo, por lo que si variamos a K entre dos y veinte, el datatest va desde el 50 % al 5 % y el datatrain va desde el 50 % al 95 %, con lo que creemos que cubrimos gran parte de los casos interesantes a observar. Notar que el valor de $K = 1$ implica que la cantidad de splits es única, es decir que no hay *cross validation*.

Los resultados del rendimiento promedio máximo para cada métrica y un $K \geq 2$ [1] fueron.

- $Accuracy = 0{,}97519$, para $K = 20$
- $Presicion = 0{,}97517$, para $K = 20$
- $Recall = 0{,}97504$, para $K = 20$
- $F1 - score = 0{,}97501$, para $K = 20$

En la figura 12, notar que en el gráfico de los promedios hay una primera bajada abrupta de las métricas, esto se produce del pasaje de $K = 1$ a $K = 2$, en el primer caso, no hay *K-fold* y la proporción de imágenes para entrenamiento-testeo es del 80 %-20 % respectivamente, mientras que para la siguiente iteración, donde comienza realmente el *K-fold*, la proporción es 50 %-50 % y de ahí va variando, 66, 6 %-33, 3 % en la tercera, 75 %-25 % en la cuarta, y así sucesivamente.

La diferencia en el rendimiento entre la primera iteración y la segunda, podría estar fundamentada en que reducimos el conjunto de entrenamiento y aumentamos el de testeo [2] y que quedan con la misma proporción del dataset original. Es intuitivo pensar que se necesitan mas muestras para entrenar que para testear, ya que en caso contrario aumentamos

[1] es decir con *cross validation*
[2] Pasamos de una proporción 80 %-20 % a una de 50 %-50 %

la posibilidad de que haya casos en el set de datos de testeo que en el set de datos de entrenamiento no estén cubiertas y por ende falle o se dificulte su predicción.

A medida que seguimos iterando, la proporción de entrenamiento aumenta y por ende, es mas probable que los casos en el dataset de testeo queden cubiertos y por lo tanto, es entendible un aumento en el rendimiento de las predicciones. Podría deberse a esta causa que hay una tendencia de aumento en el promedio de las métricas a medida que aumentamos el K, y que su valor preferible requiera ser alto.

Por otro lado la varianza de las metricas parecen tener también una correlación positiva con respecto a K. Esto podría deberse a que en cada iteración de K la cantidad de splits es mas grande y el espacio de testeo se reduce, y por ende los casos mas específicos aparecen, por lo que es mas probable que en algunas instancias de validación de *cross validation* hayan casos de test específicos y mas difíciles de predecir, mientras que en otras instancias no sea este el caso. Aún así notamos que el desvío estandar no es muy grande y no llega al 0,005, lo que significa que, estos casos específicos no aparecen con mucha frecuencia y el dataset cuente con una buena proporción de cada clase, o que nuestros parámetros elegidos, justifiquen los casos que sean específicos y puedan ser predecirlos, ambos casos se produce una validación correcta, tanto para los parámetros como del dataset de entrenamiento original.

A su vez el que la pendiente de aumento de la varianza sea chica, nos permitiría tomar un K mucho mas grande y realizar una validación aún mas exhaustiva, por desgracia el tiempo es considerablemente un factor importante en el aumento del K como veremos a continuación.

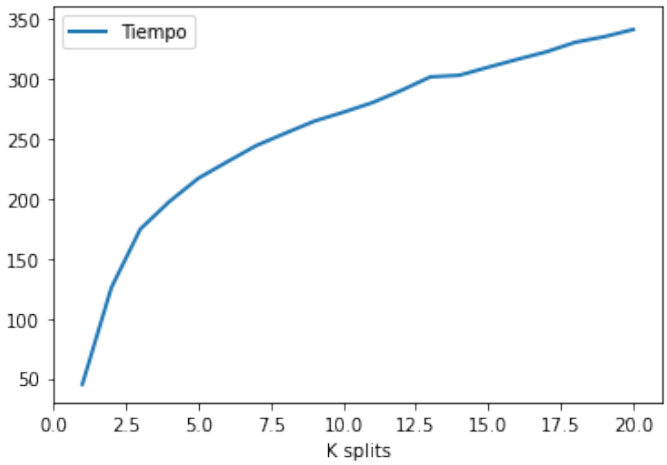
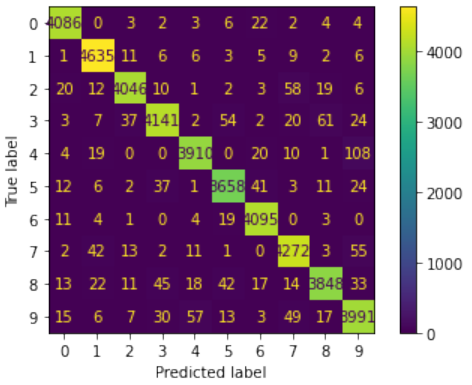
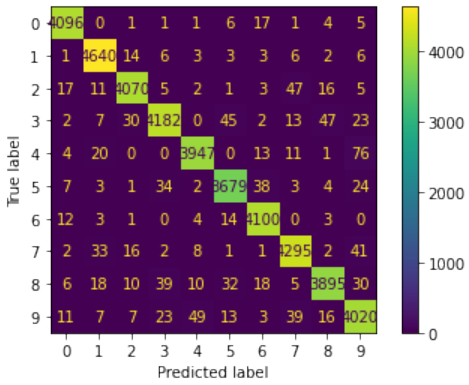


Figura 13: *K-fold* contra el tiempo (segundos)

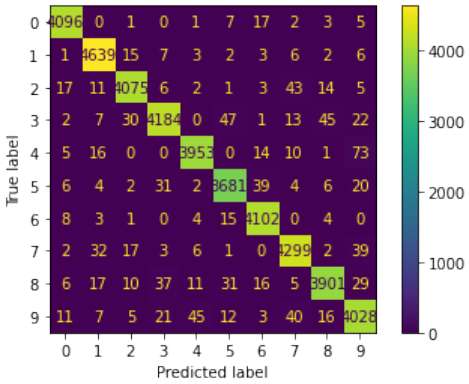
En la figura 14 observamos tres matrices de confusión para tres diferentes K. Podemos apreciar como en las matrices mas altas hay menos *False Positive* que en la iteración mas baja, y que por ende los *True Positive* aumentan. Esto es también lo que se ve reflejado en la figura 12. Notamos también que el algoritmo tiene problemas para predecir las imágenes dentro de la clase 9, en especial se lo confunde con el 4, donde hay 108 *False Positive* para el peor de los casos ($K = 2$), lo cual favorablemente, tan solo representa aproximadamente el $2,65\,\%$ de las imágenes de la clase 4.



(a) $K = 2$



(b) $K = 10$



(c) $K = 20$

Figura 14: Matrices de confusión tomados para diferentes K con los parámetros especificados en la figura 12

Las matrices de confusión en cada iteración de K que se observan resultan de sumar todas las matrices de confusión obtenidas en cada instancia de validación. Cuando testeamos en una instancia, lo estamos haciendo por $\frac{1}{K}$ del dataset original, es decir sobre un split de los K, y si sumamos las matrices de confusión de las K instancias en la iteración k-ésima resulta una suma de los resultados de los testeos y por ende en lo que resultó el rendimiento general de las instancias en la iteración k-ésima.

3.8. Variación en el tamaño del dataset

En este apartado, vamos a experimentar variando el tamaño del dataset y viendo como influye este en los resultados. Lo vamos a hacer usando los parámetros de k y *alpha* mejor encontrados y le aplicaremos *cross validation* tomando $K = 20$ a cada uno de los datasets.

Los datasets usados en este experimento se formaron a partir de tomar distintas cantidades de muestras aleatorias del dataset original. En especial las cantidades son $[100, 500, 1000, 5000, 10000, 20000, 30000, 42000]$

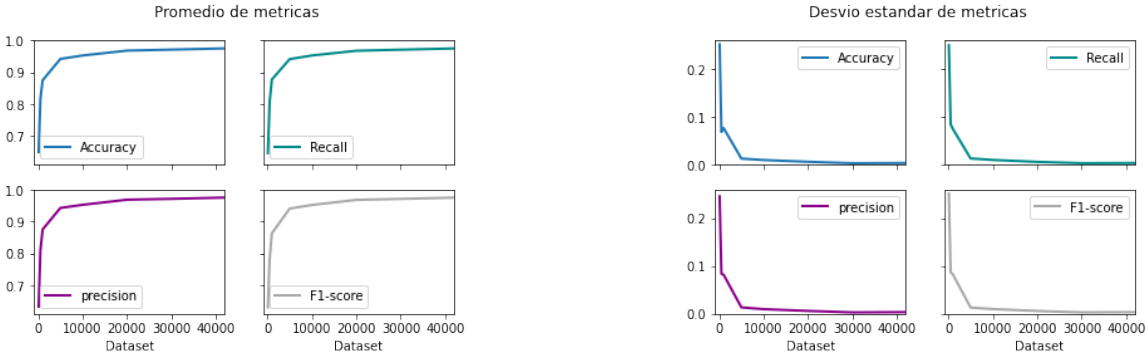


Figura 15: Parámetros: Se tomó $k = 3$, $alpha = 36$, $K = 20$ y dastasets de tamaños [100..,42000].
La figura de arriba representa el promedio y el desvío estándar de cada métrica en función del tamaño del dataset.

Los resultados del rendimiento máximo para el promedio de cada métrica y un tamaño determinado fueron:

- $Accuracy = 0{,}97519$, tamaño= 42000
- $Presicion = 0{,}97517$, tamaño= 42000
- $Recall = 0{,}97504$, tamaño= 42000
- $F1 - score = 0{,}97501$, tamaño= 42000

Podemos observar que, como podría ser esperado, a mas tamaño de muestra mejor calidad de resultado, por lo que parece razonable que el mejor resultado haya sido para el tamaño del dataset original. Esto podríamos explicarlo del siguiente modo, a mayor numero de imágenes de muestra tenemos una mayor cantidad de datos con la que entrenar y por lo tanto, nuestro algoritmo es mas capaz de captar la esencia general que distinguen a las clases, o de otro modo, es capaz de guiarse por el rasgo general y no el especifico.

Con la varianza observamos que hay una correlación negativa con respecto al tamaño del dataset, es decir que a mas tamaño menos varianza van a tener las diferentes instancias de *cross validation* entre si. Podríamos explicar esto de la misma manera que en el párrafo anterior, si tenemos mas datos sobre los que entrenar y testear, y los splits son tomados de forma aleatoria claro, hay menos posibilidad de que los casos de test caigan en algo muy específico.

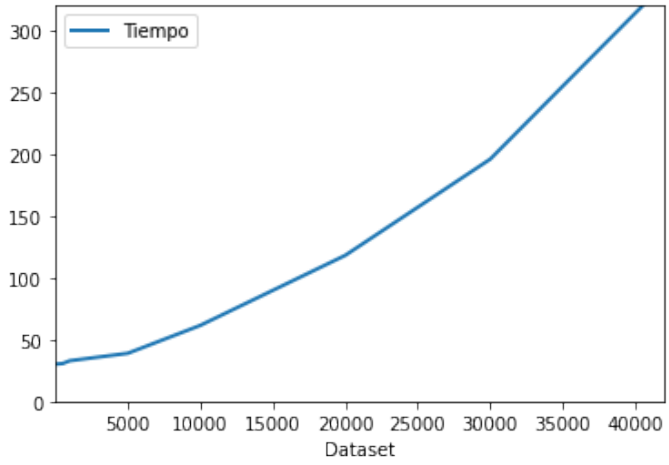


Figura 16: Tamaño de Dataset contra tiempo

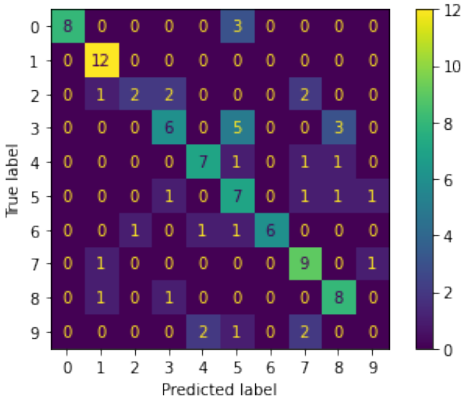
Podemos observar que a medida que aumentamos el tamaño del dataset el tiempo también aumenta.

Para el primer tamaño de dataset medido (100), nos dio un tiempo de $\approx 30{,}2510seg$, y para el ultimo tamaño medido, el tiempo dió $\approx 337{,}4957seg$. Si aplicamos un modelo de un predictor lineal naive, definiendo primero: $(x_0, y_0) = (100, 30{,}2510), (x_1, y_1) = (42000, 337{,}4957)$

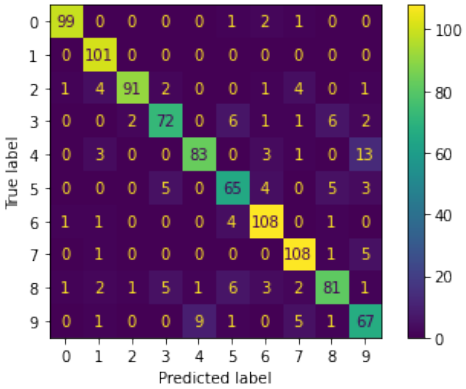
- $m = \frac{y_1 - y_0}{x_1 - x_0} = 0{,}007332$
- $b = y_1 - mx_1 = 29{,}5177$

Podemos predecir aproximadamente para un tamaño más grande, por ejemplo para un tamaño de doscientas mil imágenes: $m(200{,}000) + b \approx 1496{,}07926seg$, es decir aproximadamente 25 minutos, con lo que estimamos que si conseguimos un dataset mas grande se podría aumentar mas la calidad del resultado sin afectar mucho al tiempo.

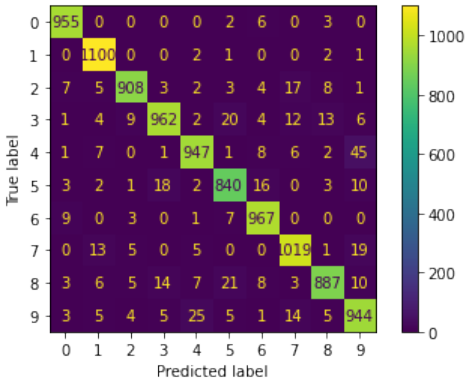
El problema, al aumentar el tamaño de un dataset, radicaría en encontrar las imágenes y etiquetarlas correctamente. Esto necesariamente debería hacerlo una persona, con lo que aumentaría el costo del proceso.



(a) Dataset de 100 imágenes



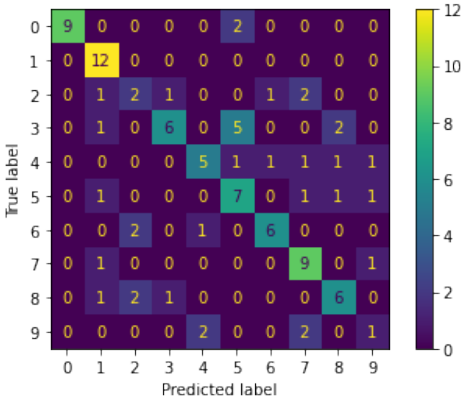
(b) Dataset de 1000 imágenes



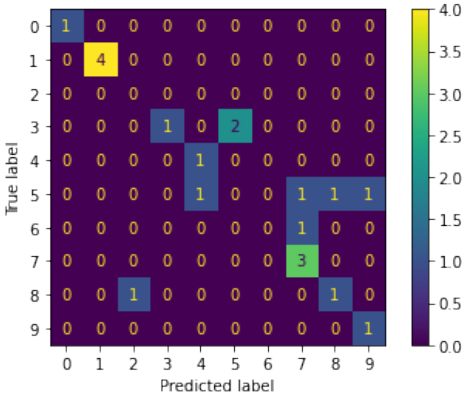
(c) Dataset de 10000 imágenes

Figura 17: Matrices de confusión tomados para diferentes proporciones del set de datos de entrenamiento original con los parámetros especificados en la figura 15

Con estas matrices de confusión junto a las de la figura 14 (en especial para $K = 20$), tenemos cuatro matrices de confusión con los mismos parámetros de k, *alpha* y K pero distintos tamaños de datasets. En las matrices de confusión en función del dataset podemos observar que hay poco acierto en las predicciones, en especial para la clase 9, esto se ve fuertemente reflejado en el dataset de 100 imágenes. Esto podría deberse a la utilización de un K relativamente alto lo cual significa un 5 % del dataset para testeo en cada instancia de validación, con lo que nos estamos quedando con 5 imágenes para testear en este caso para cada instancia. Pero aún así, si no utilizáramos *cross validation* nuestro set de datos de test estaría constituido por tan solo 20 imágenes, si tomamos una proporción 80 %-20 %, y por ende en el mejor de los casos resultaría en dos imágenes para cada clase, por lo que resultaría insuficiente. En cambio cuando usamos *cross validation* estamos, en el fondo, testeando sobre todo el dataset. Para dataset chicos habría que colocar un valor de K mas acorde. Probamos para $K = 5$ y los resultados mostraron una mejora para algunas métricas.



(a) Con $K = 5$



(b) Con $K = 1$

Figura 18: Matrices de confusión donde se usó un dataset de 100 imágenes del set de datos de entrenamiento original, $K = 5$ y $K = 1$, $k = 3$ y $alpha = 36$

Si comparamos los resultados de usar $K = 20$, 5 o 1 en el dataset de 100 imágenes encontramos,

1. Sin *cross validation*:
    - $Accuracy = 0{,}600000$
    - $Presicion = 0{,}566667$
    - $Recall = 0{,}648148$
    - $F1 - score = 0{,}564815$
    - Promedio: 0.5949
2. $K = 5$
    - $Accuracy = 0{,}630000$
    - $Presicion = 0{,}678528$
    - $Recall = 0{,}631065$
    - $F1 - score = 0{,}610676$
    - Promedio: 0.63756
3. $K = 20$
    - $accuracy = 0{,}65$
    - $Presicion = 0{,}632778$
    - $Recall = 0{,}646667$
    - $F1 - score = 0{,}630833$
    - Promedio: 0.64006

Si tomamos el promedio sin considerar el *F1-score*, que vendría a ser un promedio harmónico equilibrado [3]entre la precisión y el recall, para $K = 5$ es 0,6465 y para $K = 20$ es 0,6431.

[3]en nuestro caso, porque no favorecemos a Precisión por sobre Recall y viceversa

4. Conclusiones

La idea en este apartado es resumir algunas conclusiones que llegamos de los anteriores experimentos, también comentar resultados inesperados, así como propuestas para experimentos futuros.

En *KNN* el aumento de cantidad de vecinos tomados como parámetros no afecta al tiempo, creemos que esto es bastante interesante, ya que podríamos aumentar la cantidad de vecinos tanto como queramos. El mejor k obtenido es bajo, lo cual tiene sentido cuando trabajamos con clases muy definidas y diferenciadas entre si, como son los dígitos de cero a nueve. Una propuesta podría ser comparar un clasificador armado con solo dos clases muy parecidas y con otro clasificador armado con clases muy distintas, por ejemplo, comparar las clases uno y siete, con las clases uno y ocho.

En *PCA* observamos que para los mejores parámetros obtenidos en *KNN* se puede reducir hasta 4 veces menos el tiempo [4]. Consideramos que el algoritmo de *PCA* siempre va a reducir el tiempo ya que, poniendo una cota adecuada para el *alpha*, siempre se van a reducir las dimensiones, y los cálculos van a tener una constante mucho mas chica acompañándolos. Algo interesante que vimos, fue que aplicar el algoritmo de *PCA* no solo acorto el tiempo si no que también mejoro la calidad de las métricas. Se propone rehacer el experimento donde se varia el k de *KNN* pero con *PCA* aplicado, y observar cual es el mejor k obtenido.

En el experimento de *K-fold*, hemos concluido que el uso de *cross validation* debe usarse de diferente manera para situaciones diversas. Por ejemplo, para un dataset muy grande, es conveniente utilizar un K relativamente alto, para que podamos ver mas profundo en los datos y confirmar si es que la predicción funciona en diferentes proporciones del dataset. En cambio para un dataset mas pequeño como era el caso de la figura 18, es probable que tengamos que ajustar nuestro K con valores mas pequeños que dejen proporciones de set de datos que representen relativamente parecida y completa a todas las clases. Nos parecería conveniente también bajar el valor de K cuando tenemos muchas clases, ya que de lo contrario a cada instancia de validación se le sería difícil representar cada clase en el la base de datos de testeo. Por otro lado, un factor determinante para considerar el uso de *cross validation* es el tiempo que pesa aumentar K, cuando se requiera velocidad se deberá considerar seriamente el uso de este método.

Para concluir, en el experimento donde variamos el tamaño del dataset resultó ser que el mejor tamaño para la calidad del experimento era el mas grande. Creemos también que es mejor elegir un tamaño de dataset mas grande a costa de que aumente el tiempo, ya que en realidad la constante del tiempo no es tan costosa. Pero hay que decir que el costo de trabajar con un dataset mas grande, consideramos, radica en el armado del mismo, a saber, la búsqueda de figuras que cumplan los requisitos y la clasificación de las mismas.

[4] Observar la figura 11