DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 2

Reconocimiento de dígitos

23 de marzo de 2023 — Métodos Numéricos

Integrante	LU	Correo electrónico
Fiorino Santiago	516/20	fiorinosanti@gmail.com
Salmun Daniel	108/19	salmundani@gmail.com
Valentini Nicolas	86/21	nicolasvalentini@hotmail.com

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

Palabras claves: OCR, kNN, PCA, Método de Deflación

El objetivo de este informe es trabajar con dos métodos diferentes para resolver el problema de OCR (Optical Character Recognition). Se desarrollará la implementación kNN (k-Nearest Neighbors) y una reducción de las dimensiones del mismo con PCA (Principal Component Analysis). Además de eso, se experimentará el comportamiento de los mismos a través de diversas métricas. Las más importantes entre ellas se encuentra: evaluar la accuracy con distintas combinaciones de k y α, la misma métrica pero con bases de entrenamiento de distintos tamaños, se verá también el tiempo de computo que requiere ambos métodos y se analizará cómo varía este con diferentes parámetros. Finalmente se hará un enfoque en distintas métricas, como una matriz de confusión para los dígitos, los niveles de Precision, recall y F1-score para diversos experimentos, y por ultimo, evaluar como se comporta el método de K-Fold Cross Validation cuando se le presentan diferentes k y como difieren estos resultados de los testeos realizados anteriormente.

Índice

1. Introducción

1.1. Presentación del problema

El objetivo del informe sera a desarrollar una herramienta de OCR (Optical Character Recognition) para el reconocimiento de caracteres en imágenes. En específico, buscamos reconocer dígitos manuscritos entre 0 y 9. Para esto vamos a trabajar con bases de datos que contienen imágenes de tamaño 28 x 28 píxeles en escala de grises.

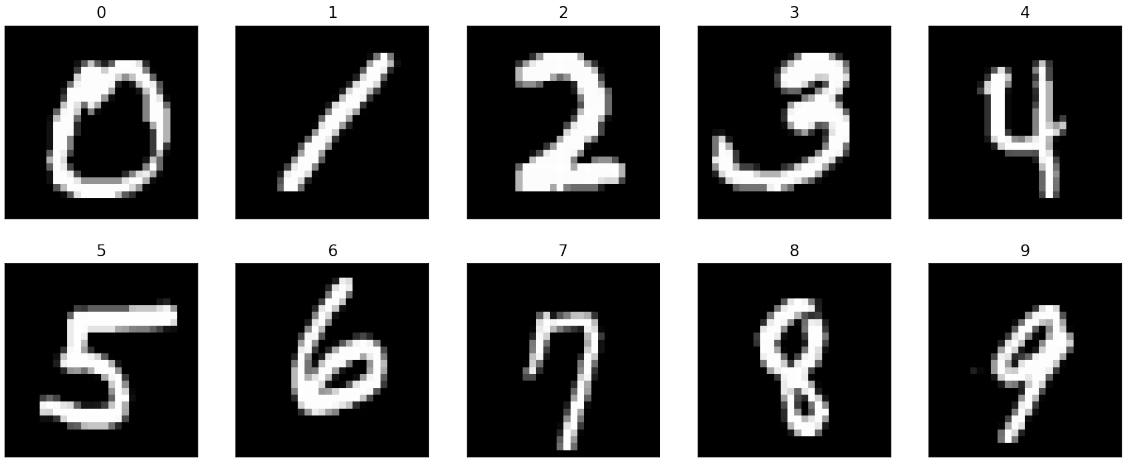


Figura 1: Ejemplos de dígitos a Clasificar

1.2. Posibles Soluciones

El método fundamental para resolver este problema es el aprendizaje supervisado. Este consiste en utilizar una base de datos la cual ya ha sido etiquetada, y usarla para entrenar nuestros modelos con el objetivo de que luego puedan generalizar, y dado una nueva imagen sin etiquetar, se pueda identificar a que dígito pertenece. Para entrenar a partir de esta base de datos de datos ya etiquetada se utilizaran dos métodos: kNN y PCA.

1.2.1. Método kNN

En primera instancia se va a considerar el método de los k vecinos más cercanos (kNN). Este metodo opera sobre vectores m-dimensionales. Para cada nueva instancia (vector a clasificar) calcula su distancia con cada uno de los vectores de entrenamiento, y separa a los k vecinos más cercanos. La clase de cada vecino ya es conocida, así que toma la moda de su clase, y esa es su predicción final. Una alternativa es tomar la moda ponderada, teniendo en cuenta también la distancia, para que un vecino que eventualmente puede estar lejos no tenga la misma importancia que uno que está muy cerca.

Si bien mencionamos que nuestras imágenes son matrices de 28x28, también es posible interpretarlas como arreglos de 784 posiciones.

Este método presenta varias desventajas. La primera es la baja eficiencia, especialmente cuando la cantidad de datos de entrenamiento es grande, ya que cada vez que se quiera clasificar una nueva imagen, se tendrá que comparar con cada una de las imágenes de entrenamiento. Otra desventaja es su sensibilidad a los atributos irrelevantes, y por último la maldición de la dimensionalidad.

1.2.2. Método PCA

Por otro lado se trabajará con el método de análisis de componentes principales (PCA). El principal fundamento de este método es reducir la cantidad de dimensiones con las que se trabaja la base de datos. La forma de hacer esto es buscando cuáles son las zonas de las imágenes que "más información proveen". Para esto se busca inicialmente cuáles son los lugares en la imagen que mayor covarianza tienen según la base de datos. El sentido de esto es que a mayor covarianza, más significativa es la diferencia entre un estado y otro, y por ende más información es proveída por la imagen. Los pasos de este método constan de crear una matriz de covarianza, diagonalizarla y luego separar los α más significativos para clasificar los elementos. Finalmente se realiza el método de kNN pero ya no con las m-dimensiones iniciales sino con tan sólo α.

2. Desarrollo

2.1. Metodología de Desarrollo

2.1.1. Implementación de kNN

La implementación de este método consta en encontrar aquellas imágenes (en este caso vectores de 784 posiciones) dentro del train, las cuales tengan la menor distancia a la imagen a clasificar. Para calcular la distancia aplicamos la norma 2 sobre la resta de los vectores. Para quedarnos con los k vecinos más cercanos usamos un arreglo de vecinos, en el que cada vecino tiene su clase y su distancia. Este arreglo lo mantenemos ordenado ascendentemente por distancia, así en cada iteración solamente debemos verificar si la distancia a la imagen en cuestión es menor a la del último vecino, en ese caso se los intercambia, y se vuelve a ordenar el arreglo.

Una vez obtenidos los K vecinos más cercanos se aplica una moda basada en la distancia de cada uno. Para ello creamos un arreglo de 10 posiciones (una por cada dígito). Luego recorremos cada vecino, y le sumamos a la clase a la cual pertenece (1 / distancia). Usamos el inverso de la distancia ya que buscamos que mientras más cerca esté el elemento, mayor peso tenga su voto. Una vez sumadas todas las distancias, solamente queda quedarnos con el elemento máximo del array, y su índice será la predicción.

Algorithm 1 Implementación de kNN

```
for i = 1 to Test.size() do
    for j = 1 to Train.size() do
        if (Test[i] - Train[j]).norm2() < kNN.[k-1].dist then
            kNN[K-1] = Train[j]
            kNN.sort()
        end if
    end for
    Vector classesCount (10, 0)
    for j = 0 to K do
        classesCount[kNN[j].nro] += 1/kNN[j].dist
    end for
    maxCount = 0
    for int j = 1 to 10 do
        if classesCount[j] > classesCount[maxCount] then
            maxCount = j
        end if
    end for
end for
```

Un detalle a destacar es que no es necesario hacer este proceso múltiples veces en base al k que se quiere evaluar. Lo que quiere decir esto es que una vez sacado el k = 20, ya se sabe también los vecinos de todos los k menores que ese.

2.1.2. Implementación de PCA

Por lo tratado en la introducción, la función principal de este método es reducir la cantidad de dimensiones con las que se va a ejecutar el kNN, para eso lo que se busca hacer es transformar las matrices del train en algo de menor cantidad de dimensiones. Para que la reducción de dimensiones sea eficiente se busca analizar aquellas partes de una imagen que sean las que más información proveen. En este caso definimos esto como las partes que contienen mayor cantidad de covarianza. Para hacer un simple ejemplo de esto, los bordes dentro de una imagen tienden a ser un lugar que proporcionan poca información, y por ende varían poco entre las diferentes instancias, por otro lado el centro suele ser un lugar de mucha importancia y tiende a variar mucho entre las distintas imágenes. Para representar bien esto lo primero que se realiza es crear una matriz de covarianzas. Esta matriz contiene en cada punto las covarianzas basadas en las coordenadas a analizar, por ende esta matriz existiría en MxM siendo M la dimensión de una imagen. La covarianza de un punto se la define como:

$$\sigma_{x_j x_k} = \frac{1}{n-1}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k) = \frac{1}{n-1}(x_k - \mu_k v)^t(x_j - \mu_j v) \tag{1}$$

Debido a esto, una idea para la implementación de la matriz podría ser :

Algorithm 2 Matriz de Covarianza

for i = 1 to 784 **do**
  for j = 1 to Train.size() **do**
    $X_{ij} = (X_{ij} - \mu_j)/(\sqrt{n-1})$
  end for
end for
$M_x = X^t X$

Una vez obtenida la matriz de covarianza, queda por ver cuales son aquellos lugares más "importantes". Se ha de notar que esta matriz por como fue creada (multiplicación de una por su traspuesta) es simétrica. Al ser una matriz simétrica se puede asegurar que se puede realizar una diagonalización en matrices ortogonales de la misma, en donde en la diagonal quedarían los autovalores de forma ordenada y en las columnas sus autovectores asociados. Al estar ordenados en base a sus autovalores, estos autovectores son aquellos que mayor covarianza tienen respecto al resto. Para realizar esta diagonalización, fue implementado el método de deflación, el cual requiere de la obtención de autovalores, estos autovalores son obtenidos a través del método de la potencia. La implementación de ambos es la siguiente:

Algorithm 3 Método de la potencia

for i = 1 to nIter **do**
  $v = Bv$
  $v = \frac{v}{\|v\|_2}$
end for
Return $\frac{v^t B v}{v^t v}$

Algorithm 4 Metodo de deflacion

for i = 1 to B.size() **do**
  $\lambda = metodoDeLaPotencia(B, v)$
  $B = B - \lambda v v^t$
end for

Con estos autovectores queda transformar tanto el train como el test. La transformación consta de multiplicar a las imágenes por los α autovectores más significativos, reduciendo así el tamaño de lo que se va a evaluar de M a α. Finalmente se utiliza nuevamente el método de kNN, pero ya no con M dimensiones sino con α ya transformadas.

3. Experimentación

3.1. Accuracy de PCA dependiendo de k y α

El objetivo de este experimento es ver cómo varía el accuracy del algoritmo PCA con diferentes combinaciones de parámetros k y α. Para este experimento usaremos la base de entrenamiento y de test de Kaggle completas, y obtendremos el accuracy desde la misma plataforma.

Esperamos observar que para alphas chicos, el resultado no sea muy bueno, ya que estaría dejando información importante de lado. Sin embargo, para alphas muy grandes esperamos ver muy poca mejora, ya que estaríamos teniendo en cuenta información no importante, o ruido.

3.1.1. Resultados

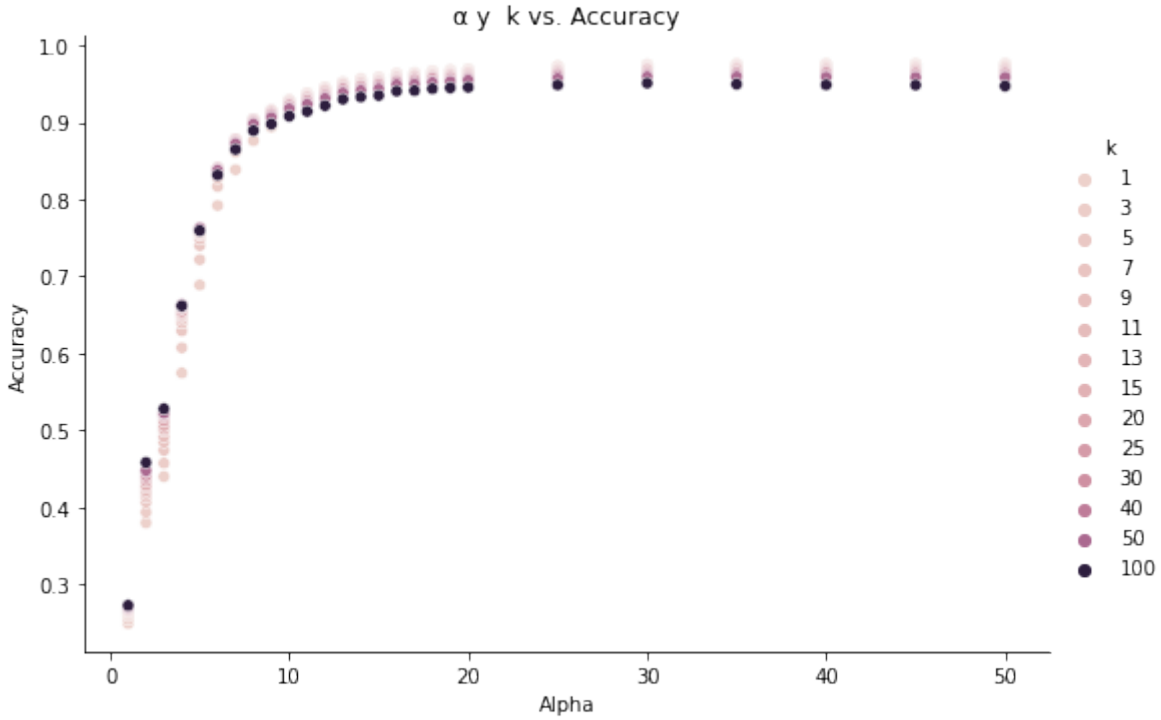


Figura 2: Accuracy de PCA y kNN variando la cantidad de imágenes

En la Figura 2 podemos observar que el resultado fue el esperado. Para alphas muy chicos el accuracy es muy pobre. También notamos que los primeros 10 autovectores influyen mucho, haciendo crecer el accuracy de 0.25 a 0.90, sin embargo a partir de $\alpha = 20$ el crecimiento es muy poco.

Otra observación interesante es que cuando el α es pequeño (menor a 10), es más conveniente tomar una mayor cantidad de vecinos, es decir aumentar el k de kNN. Sin embargo, pasada estas instancias, a mayor α pasa a ser más conveniente tomar una menor cantidad de vecinos. Una posible explicación para esto puede ser que cuando el α aumenta, la categorización que se realiza es más completa, por ende es más probable que con un k más pequeño queden aquellos resultados más representativos, mientras que en un k más grande pueden influenciar más resultados fuera del real.

3.2. Comparación de Accuracy entre kNN y PCA

En este experimento buscamos observar la diferencia de accuracy entre kNN y PCA para distintos parámetros k y α. Para este experimento usaremos la base de entrenamiento y de test de Kaggle completas, y obtendremos el accuracy desde la misma plataforma.

Como observamos en el experimento anterior (3.1), para alphas chicos el accuracy es muy bajo, por lo tanto esperamos que el desempeño de PCA sea peor que el de kNN, sin embargo, también esperamos que para ciertos alphas el accuracy de PCA sea superior al de kNN, ya que sólo tendrá en cuenta las partes más importantes de las imágenes, ignorando el ruido.

Con respecto al método de kNN, esperamos que exista cierto k para el cual el accuracy sea óptimo. Usando k = 1 existe la posibilidad de que el más cercano sea un outlier de otra clase, por lo tanto con un k más alto estaríamos evitando ese problema. Por otra parte, usar un k muy alto podría tener efectos negativos, sobre todo si las nubes de las clases se encuentran cerca, ya que estaríamos tomando vecinos de otra clase. Por lo tanto, buscamos observar dónde se encuentra este balance de vecinos.

3.2.1. Resultados

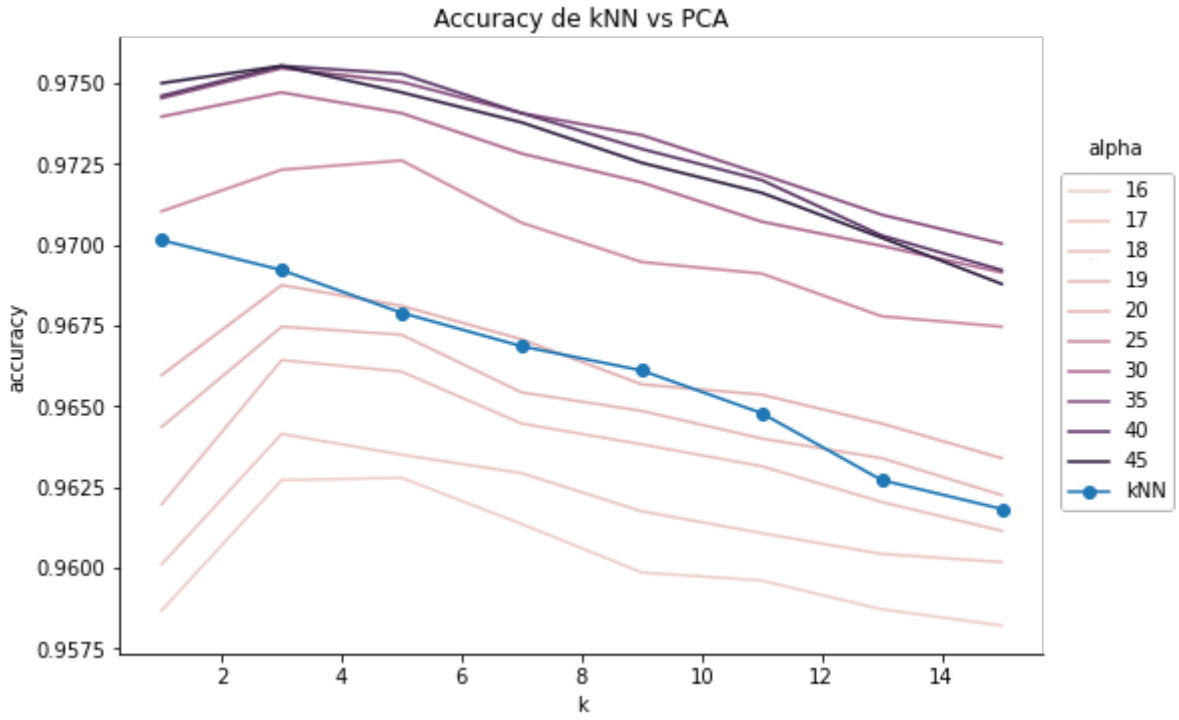


Figura 3: Comparación de Accuracy entre kNN y PCA para distintos k y alphas

En la Figura 3 se puede ver que recién con un $\alpha = 25$ el método PCA dio resultados consistentemente mayores a los de kNN. Nuestra interpretación es que hasta el caso de tomar un $\alpha = 25$ sigue habiendo información importante para una correcta clasificación. En el caso de tomar alphas menores, estaríamos dejando de lado información importante, que todavía tiene una varianza significativa para cada clase. En ese sentido también se puede ver que la diferencia de los resultados entre $\alpha = 20$ y $\alpha = 25$ es mucho mayor que entre $\alpha = 40$ y $\alpha = 45$. La explicación atrás de esto sería que ya para la instancia de 40 y 45, cada α ya provee muy poca información significativa.

La curva de kNN no resultó la que esperábamos, ya que obtuvimos el mejor accuracy con k = 1, y a medida de que aumentamos el k, el accuracy empeoraba. Esto creemos que se debe a la proximidad de las nubes de las clases, que provocan que al tomar más vecinos, varios de ellos sean de clases cercanas (pero diferentes), obteniendo clasificaciones erróneas.

3.3. Accuracy dependiendo de las imágenes de entrenamiento

El objetivo de este experimento es ver como varía el accuracy dependiendo de la cantidad de imágenes de entrenamiento. Para ello creamos 6 subsets del dataset original, con tamaños 1000, 2500, 5000, 10.000, 25.000 y 40.000. Estos subsets están balanceados, conteniendo exactamente la misma calidad de imágenes de cada dígito. Luego, correremos kNN con y sin PCA, y usaremos el test de Kaggle para verificar el accuracy. Los parámetros del algoritmo PCA serán $\alpha = 50$ y k = 3, ya que estos fueron los que mejor resultado dieron en el experimento 3.1. Para kNN sin PCA también usaremos k = 3, para que la comparación sea justa.

Esperamos que mientras mayor sea la cantidad de imágenes de entrenamiento, mayor sea el accuracy de ambos métodos, ya que habrá más vecinos con los que comparar la imagen a clasificar.

3.3.1. Resultados

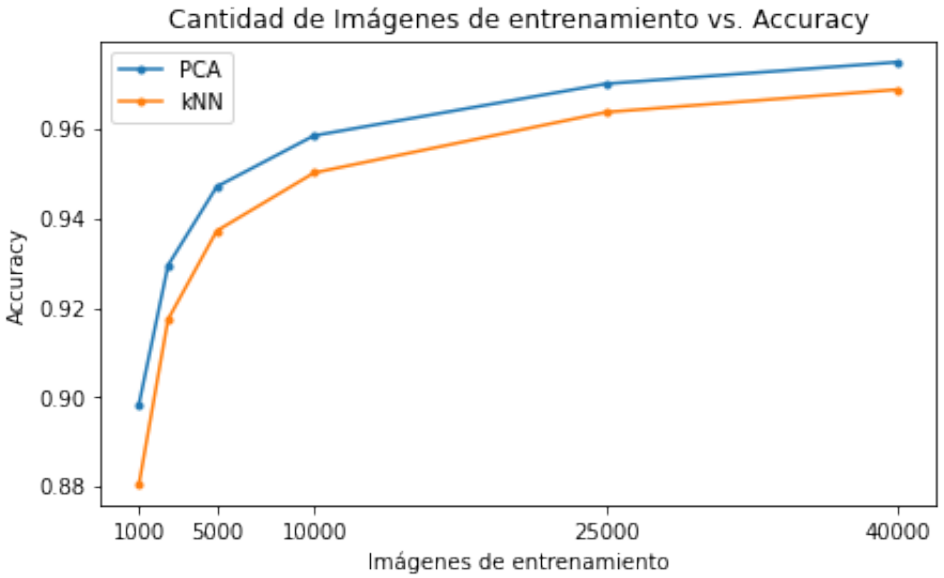


Figura 4: Accuracy de PCA y kNN variando la cantidad de imágenes

Por lo trabajado anteriormente, la mejora del PCA era esperable debido al α alto y al k bajo. Con respecto a la diferencia de resultados también se podría decir que era esperable. Se supone que cuando se cuenta con una base de aprendizaje grande, los resultados de accuracy deberían ser mejores. En ese sentido es importante remarcar que al principio hay una diferencia grande debido a que en esas instancias no hay tantas imágenes de las cuales puede aprender el train. Por otro lado, ya llegado a las 25.000, el salto a las 40.000 presenta menos de un 1 % de mejora, debido a que la diferencia del aprendizaje no es tan grande.

3.4. Eficiencia temporal

Para los siguientes experimentos usaremos un subset del dataset de entrenamiento de kaggle con 5000 imágenes, ya que en el experimento 3.3 pudimos ver que eran suficientes para obtener una accuracy muy buena. También usaremos un subset del dataset de test de kaggle, en este caso con 3000 imágenes, así mantenemos el porcentaje train/test del set completo. Cada experimento será ejecutado 10 veces y nos quedaremos con el tiempo promedio de ejecución.

Esperamos que a mayor k ambos algoritmos tengan un peor desempeño, ya que par cada imagen a clasificar habrá que quedarnos con los k vecinos más cercanos, y calcular la moda entre más elementos. Además, en el caso de PCA esperamos que para mayor alpha, el rendimiento también empeore, debido a que estaríamos trabajando sobre dimensiones más grandes.

3.4.1. Resultados

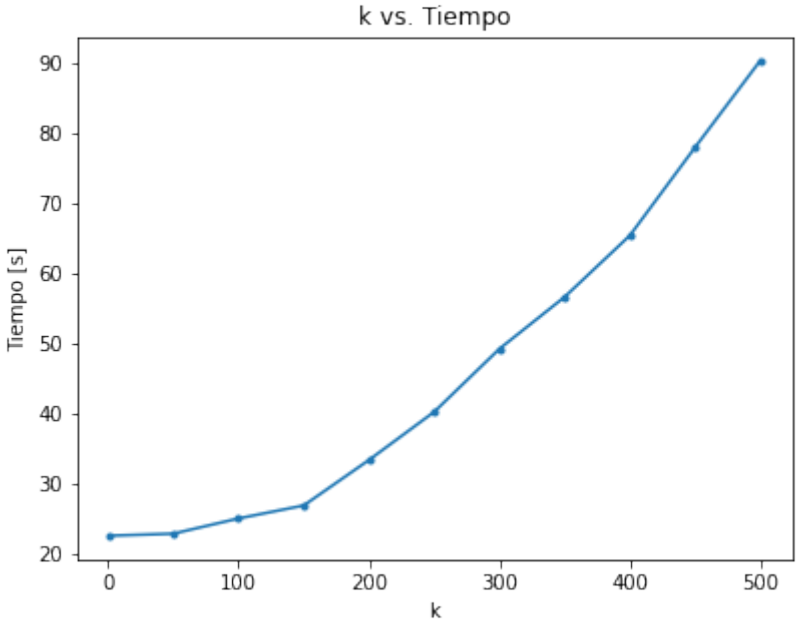


Figura 5: Tiempo promedio tomado por kNN en función del k

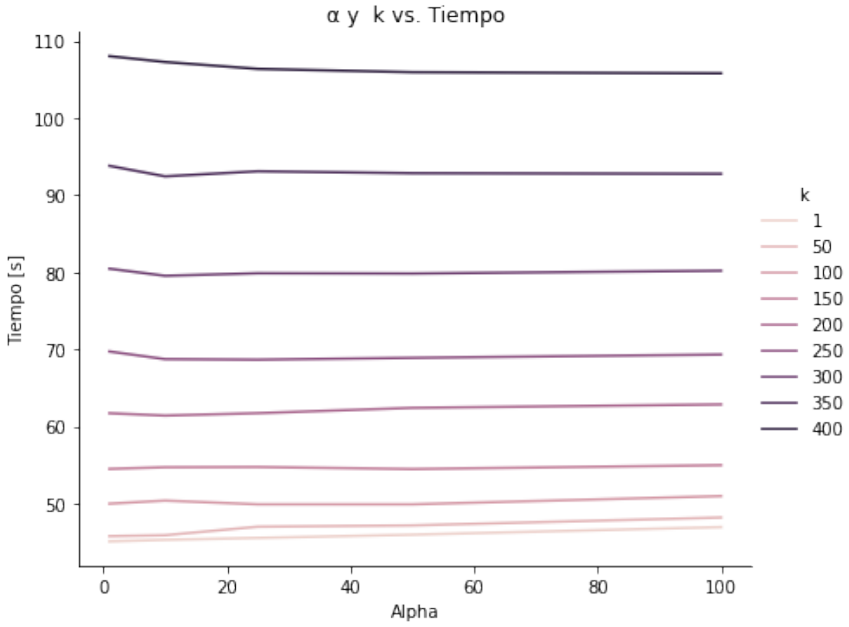


Figura 6: Tiempo tomado por PCA para diferentes parámetros

En la Figura 5 queda claro que el parámetro k del método kNN tiene una gran influencia en el tiempo de computo. Sin embargo, en la Figura 6 podemos observar que el parámetro α de PCA prácticamente no afecta en el desempeño. Esto creemos que se debe a que la biblioteca de C++ que usamos (Eigen) está muy bien optimizada, y el tiempo tomado por las operaciones entre matrices y vectores aumenta muy poco al aumentar las dimensiones de los mismos. Sin embargo se nota que el k si está influyendo, por lo tanto para observar mejor esto, en el siguiente experimento fijaremos el α en 50, y variaremos el k.

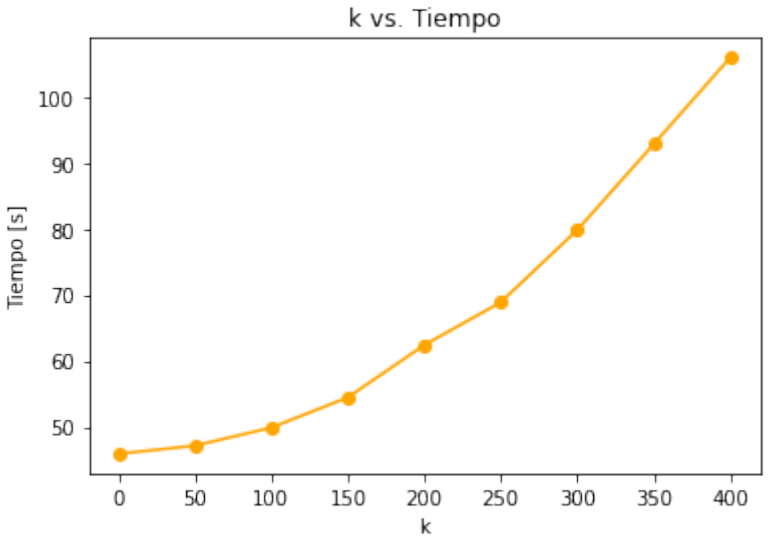


Figura 7: Tiempo promedio tomado por PCA en función del k

Como se puede ver en la Figura 7, el comportamiento de la función es básicamente el mismo con y sin PCA. Aumentar el parámetro k produce un aumento de tiempo exponencial. En este caso, observando la Figura 5 y la 7 podemos observar que el tiempo del método de PCA resultó mayor al de kNN. Este resultado se debe a que el tamaño del test es pequeño. Cuando no hay muchas imágenes a clasificar, con una misma base de entrenamiento el kNN será más rápido, debido a que el proceso de calcular la matriz de covarianza y diagonalizarla lleva tiempo. Mientras el método de PCA se dedica a calcular esas dos matrices, el algoritmo de kNN ya está clasificando imágenes. Sin embargo, a mayor cantidad de imágenes a clasificar el método de kNN es mucho más lento que el método de PCA.

3.5. Dígitos confusos

El objetivo de este experimento es determinar cuáles son los dígitos que más confunde el algoritmo de PCA, y observar como disminuye la confusión al incrementar el parámetro alpha. Los gráficos a continuación son el resultado de PCA con k = 3, entrenado con la técnica de K-Fold Cross Validation, con 10 folds.

3.5.1. Resultados

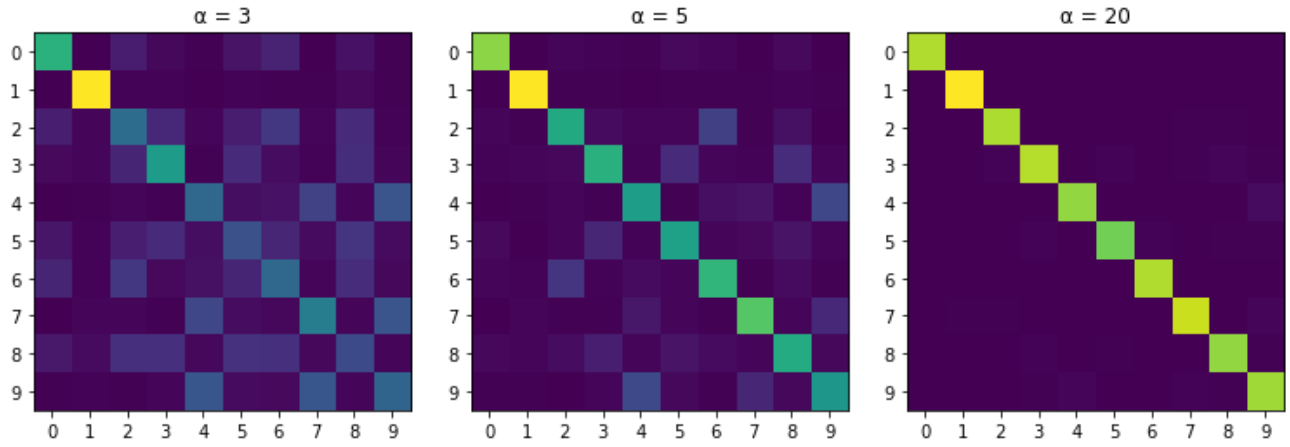


Figura 8: Matrices de confusión de los dígitos en función de α

alpha	0	1	2	3	4	5	6	7	8	9
3	0.489	0.841	0.225	0.378	0.218	0.170	0.211	0.266	0.140	0.201
5	0.809	0.891	0.509	0.513	0.436	0.510	0.528	0.608	0.496	0.398
20	0.978	0.976	0.953	0.914	0.931	0.928	0.965	0.943	0.921	0.888

Cuadro 1: Precisión de cada dígito en base a distintos α

Tras lo visto en la Figura 8 una primera conclusión que se puede realizar es que a mayor α menos confusión hay entre los dígitos. Esto era muy esperable en base a lo que se venia trabajando anteriormente. Hay algo muy destacable de este experimento y son los resultados muy altos del 1 en todas las pruebas. Esto conceptualmente tiene mucho sentido debido a que hay pocas instancias necesarias para poder definir la imagen del 1, por ende resulta que con un α muy chico como 3, ya puede ser suficiente para identificarlo consistentemente bien. En su extremo opuesto se presenta el 9, como el dígito más problemático para identificar. Este fenómeno se puede entender pensando en la imagen del 9, y como puede haber no solo muchas formas de escribirlo, sino también muchos dígitos con rasgos muy similares, como el 4 o el 7.

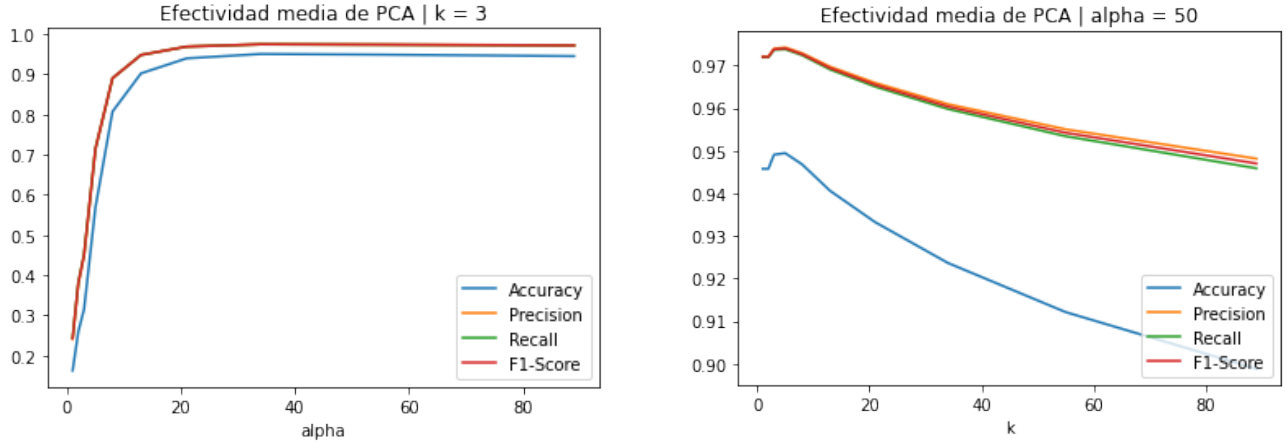
3.6. Precision, recall y F1-Score

En este experimento la idea es evaluar cómo va variando el accuracy, precisión, recall y F1-Score de PCA con distintos parámetros. Para este experimento se uso K-Fold con 5 folds en vez de 10 para reducir el tiempo de computo. Como anteriormente los mejores parámetros de PCA dieron $\alpha = 50$ y $k = 3$ se corrió PCA primero fijando k en 3 y variando el alpha y luego se corrió el experimento nuevamente fijando alpha en 50 y variando el k. Graficaremos las métricas dadas tanto para cada uno de los dígitos como la media de estas.

3.6.1. Resultados

Al ser demasiados gráficos los generados solo presentaremos en este informe las que nos resultaron interesantes. Si se quieren ver los otros gráficos se puede ejecutar el código adjuntado con el informe.

Primero analizaremos las medias de estas métricas para cada uno de los dígitos. Notese que como el falso positivo de uno es el falso negativo de otro entonces el precisión, recall y F1-Score deberían ser aproximadamente equivalentes en estos casos. En la Figura 9a podemos observar que todas las métricas aumentan rápidamente a medida que se aumenta el alpha hasta llegar a aproximadamente $\alpha = 30$ donde a partir de ahí ya la varianza es casi nula. En cambio en la Figura 9b se puede observar como van disminuyendo las métricas luego de los primeros valores de k, demostrando nuevamente que un k bajo es más óptimo para el dataset dado.

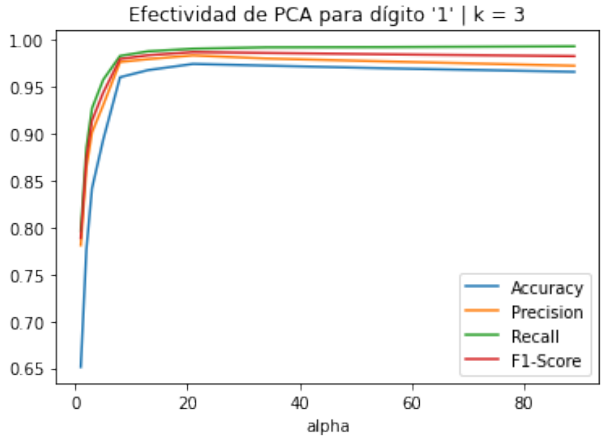


(a) En función de α

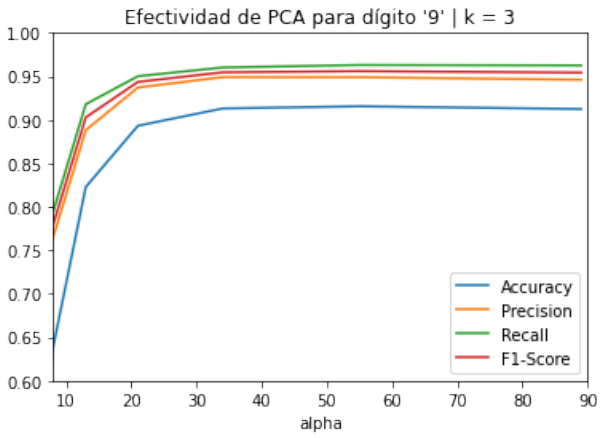
(b) En función de k

Figura 9: Media de las métricas en función de distintos parámetros de PCA

Analizando las métricas dígito a dígito destacamos dos resultados que consideramos interesantes. En primer lugar, en la sección *Dígitos confusos* vimos que el dígito con peor accuracy para alphas bajos era el 9 y el que más accuracy tenía es el 1. En la Figura 10 esto se ve aún más claro ya que vemos que para el accuracy para el dígito 1 con $\alpha = 1$ es aproximadamente equivalente al accuracy del dígito 9 con $\alpha = 8$. Algo que notamos tanto en estos gráficos como en otros en función del alpha, es que el precision, recall y F1 score tiene una correlación bastante clara con el accuracy y que no varía de manera considerable para distintos alphas.



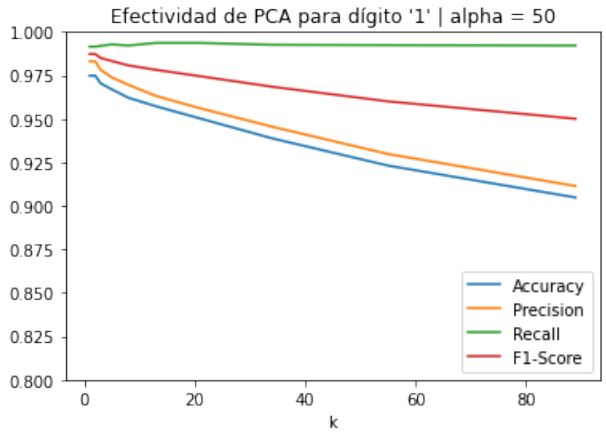
(a) Métricas para el dígito 1



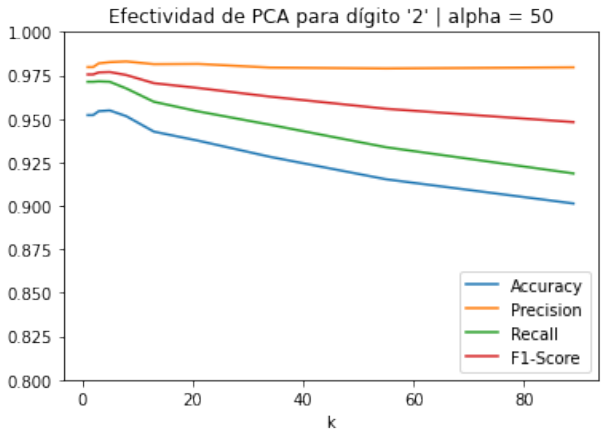
(b) Métricas para el dígito 9 - Zoomeado

Figura 10: Métricas para ciertos dígitos ejecutando con PCA con $k = 3$ y variando el α

En segundo lugar, cuando hicimos los gráficos en función de k, sí hubo varianzas en las métricas independientes del accuracy. En particular descubrimos que a medida que se aumenta el k, hay mayores probabilidades de que PCA clasifique a un dígito como un 1. Esto causa que aumente el recall del dígito 1 a medida que se aumente el k pero causa que decrezcan sus otras métricas considerablemente, como se puede ver en la Figura 11a. Sin embargo, en los otros dígitos no encontramos que aumente la precision de manera significativa a causa de esto, así que no encontramos ningún beneficio en aumentar el k. El único dígito que aumenta ligeramente su precisión es el 2 a coste de un decrecimiento considerable de su recall como se puede ver en la Figura 11b



(a) Métricas para el dígito 1



(b) Métricas para el dígito 2

Figura 11: Métricas para ciertos dígitos ejecutando PCA con $\alpha = 50$ y variando el k

3.7. K-Fold Cross Validation

En este experimento observaremos como se comporta el Precision, Recall y F1-Score cuando variamos el parámetro K de la técnica K-Fold Cross Validation. Para este experimento usaremos el método de PCA, con $\alpha = 50$ y k = 3, y la base de train será la de Kaggle completa. Además, compararemos los resultados con el Accuracy obtenido usando el test de Kaggle.

3.7.1. Resultados

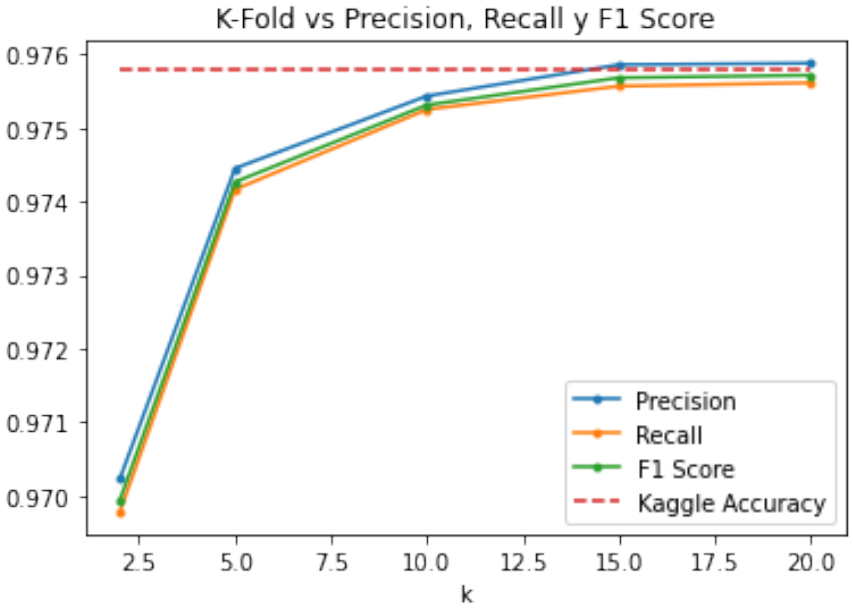


Figura 12: Metricas en función del parámetro 'K' de K-Fold

Como podemos observar en la Figura 12, para K bajos la predicción de la efectividad del método da más baja. Esto se debe a que en el caso de K = 2 por ejemplo, estaríamos partiendo la base de entrenamiento en dos. Al partir la base en dos, tenemos una base de entrenamiento mucho más chica, y como vimos en el experimento 3.3, esto afecta negativamente en el resultado. Al aumentar el K, cada entrenamiento será con una mayor cantidad de imágenes, y por lo tanto la predicción del desempeño del método estará más cerca de la real. Observar como se acerca a la recta de Kaggle Accuracy, siendo esta el Accuracy "real"del modelo. Sin embargo, aumentar el K tiene una gran desventaja, que es el tiempo de computo. Aumentar el K implica tener que entrenar, testear el modelo y comparar con el resultado esperado K veces, lo cual puede ser muy costoso.

4. Conclusiones

Tras lo trabajado en las diversas áreas del informe se pueden ir formando muchas conclusiones. La primera es que tras experimentar con diversos k en kNN y distintos α cuando se aplica PCA, la combinación que mejor resultado de accuracy tuvo fue con k = 3 y $\alpha = 50$ cuando se trata de tanto el train como el test completo provisto por Kaggle. En estos experimentos se pudo observar también que cuando los α son menores, los resultados de accuracy tienden a mejorar cuando los combinamos con k mayores, sin embargo, a medida que se incrementa α, se obtienen mejores resultados con k menores. La explicación propuesta fue que cuando se tenían α bajos los vecinos ayudan más a determinar un dígito especifico, sin embargo, en los α más grandes, con los cuales se tiene más precisión, más vecinos puede implicar un voto más contaminado en kNN.

Sobre el comportamiento de kNN cuando se aplica o no PCA, se pudieron observar varios detalles también. A nivel accuracy, se pudo ver que PCA empezaba a presentar una mejora por sobre kNN básico recién cuando se trabajaba un α igual a 25. La razón encontrada fue que antes de esta instancia, cada α proveía información muy significativa y por ende se reducía la calidad de la clasificación. También se compararon ambos métodos a nivel tiempo de computo y se llegaron a unos resultados no tan esperados en un principio. Por un lado que cuando se aplica PCA, practicante no hay una diferencia de computo cuando se varían los α, sí cuando se varía el k. Por otro lado también se vio que con el data set experimentado de 5000 imágenes, kNN se ejecutaba de forma más rápida sin PCA. Una razón propuesta podría ser que cuando se aplica PCA, lo que más tiempo consume es crear la matriz de covarianza y diagonalizarla, mientras que en el caso de kNN básico el procedimiento simplemente recorre todo el train. Como en este caso el train no era tan grande, recorrer el train lo vencía en tiempo de computo.

Otro tema que también fue trabajado fue el accuracy de kNN con y sin PCA cuando se tienen bases de distintos tamaños. De lo observado se pudo ver que cuando se trabaja con una base de 5000 imágenes, los resultados ya son muy buenos y cuando se incrementa, la mejora en la métrica trabajada no es muy significativa.

Por otro lado se realizaron experimentos midiendo aspectos distintos al de accuracy. Uno de ellos fue la realización de una matriz de confusión de los dígitos para diferentes α en PCA. Aquí se entendió un fenómeno muy interesante y es que PCA necesita de pocos α para poder tener un buen grado de precisión del dígito 1. Esto tiene mucho sentido para nosotros debido a que tiende a ser el dígito menos ambiguo y que en menos zonas varia entre sus diferentes imágenes. Por su contrapuesto, el 9 pareció ser el dígito más complejo, llegando al extremo de que con un $\alpha = 20$ este numero tiene un grado de precisión menor que el 1 con $\alpha = 5$. Una posible explicación para la deficiencia del 9 puede ser la similaridad que comparte con el 4 y el 7.

Siguiendo con el análisis de métricas distintas a las del accuracy se lograron ver varios fenómeno interesantes. El más destacable es que cuanto más crece el k, aumenta significativamente el recall del dígito 1. Lo que quiere decir esto es que el método tiende a ser más laxo a la hora de definir que dígitos son 1. Esto esta relacionado con lo mencionado anteriormente sobre los k elevados. Al tomar mayor cantidad de k, y ser analizado con un α elevado, muchos números corren el riesgo de ser catalogados en una forma más cercana a los 1, por ende cuando el k es bajo, esto no tiene demasiado peso, sin embargo cuando se aumenta, se tiende a tomar con mayor peso el voto de los 1, y de ahí el recall significativamente más alto.

Finalmente, sobre los experimentos de la K-Fold Cross Validation, los resultados fueron los esperados, debido a que con los k bajos, el train se corta en un mayor tamaño, los resultados son un poco peores. Sin embargo, con el aumento del K, los resultados mejoraron, llegando a los niveles del test de con la base de Kaggle. La gran contra parte de este método y sobretodo de los K altos, son definitivamente los tiempos de computo.