DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

"Si nos organizamos aprobamos todos..."

Metodos numericos
Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Gastón Zanitti	058/10	gzanitti@gmail.com
Ricardo Colombo	156/08	ricardogcolombo@gmail.com
Dan Zajdband	144/10	Dan.zajdband@gmail.com
Franco Negri	893/13	franconegri200@gmail.com
Alejandro Albertini	924/12	ale.dc@hotmail.com

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. Introduccion

El objetivo de este trabajo es la realización y el análisis de algoritmos eficientes para el reconocimiento óptico de caracteres (OCR), particularmente de dígitos, a través de la utilización de técnicas simples de Machine learning.
El trabajo consiste en una serie de experimentaciones. El desarrollo de estas encuentra un hilo conductor en las mejoras aplicadas a un algoritmo basadas en problemas particulares que se pueden encontrar en la resolución del problema:

- Se parte de una base de datos de imágenes ya etiquetadas y otra con imágenes sin etiquetar. Usando la base de datos etiquetada como información de entrenamiento del algoritmo, se intenta etiquetar de modo correcto los dígitos de la base de datos sin etiquetas.
- La primera aproximación a la resolución del problema utiliza el método más intuitivo encontrado: Por cada imagen de la base de datos sin etiquetas, se busca la que más se le parece en la base de datos etiquetada y se marca a la imagen sin etiqueta con la etiqueta de aquella que denominamos como la más parecida. Por supuesto, todavía queda determinar cual es el criterio para decir que dos imágenes se "parecen". Esta definición está dada con profundidad en la sección de desarrollo.
- Surge entonces la pregunta acerca de que pasa si, por una particularidad de la imagen, la etiqueta más parecida no es la correcta para el dígito a averiguar. Para mitigar este problema parcialmente se pueden tomar las k imágenes más parecidas (que a partir de ahora llamaremos vecinos) y elegir como etiqueta aquella que se repita más entre los k vecinos. Detrás de esta idea se encuentra el algoritmo KNN, que se utiliza para mejorar el comportamiento en estos casos donde el vecino más cercano no pertenece necesariamente a la misma clase que la imagen a etiquetar.
- Por último, a esta idea se le puede aplicar una mejora sustancial utilizando un método probabilístico conocido como PCA. Este consiste en aplicar una transformación a las imágenes, de tal manera de solo tener en cuenta aquellas de mayor variabilidad y desechar aquella información que pueda estar introduciendo ruido.

Para entender las diferencias y similitudes entre los métodos y sus variantes, se realizan los experimentos con variaciones en los parámetros. En el caso de KNN se varía la cantidad de vecinos, esto ayuda a entender que valores ayudan a la optimización del algoritmo.
Para el caso de la mejora utilizando el algoritmo de PCA también hay que tener en cuenta el α utilizado. Vamos a ver como modificar este valor conlleva diferentes tiempos de ejecución y pérdida o ganancia de precisión.

2. Desarrollo

2.1. Algortimo de kNN

Como primera aproximación para la resolución del problema de OCR, implementamos el algoritmo de K-vecinos más cercanos (o kNN por sus siglas en inglés). Este método de clasificación consiste básicamente en, dado un dato del que no conocemos a que clase pertenece, buscar entre las imágenes del dataset etiquetado las k más parecidos, llamados también como sus "vecinos"(habiendo que definir que es ser "parecido"), y luego de estos k vecinos, determinar cual es la moda.

2.1.1. Similitud entre imágenes

Para este trabajo en particular, tomamos las imágenes como vectores numéricos y definimos que dos imágenes son "parecidas"si la norma dos entre ellas es pequeña. Luego la idea del knn será tomar todas las imágenes etiquetadas, compararlas contra la nueva imagen a reconocer, ver cuales son las k imágenes cuya norma es la menor posible y, entre esos k vecinos, ver a que clase pertenecen. La etiqueta para esta imagen vendrá dada por la moda.

Para los siguientes pseudocódigos será necesario asumir que todas las estructuras utilizadas almacenan datos enteros a menos que se indique lo contrario, esto se indica agregando entre paréntesis el tipo de dato que almacena.

TP1 1 Vector KNN(matriz etiquetados, matriz sinEtiquietar,int cantidadVecinos)

```
1: vector etiquetas = vector(cant_filas(sinEtiquetar))
2: for 1 to cant_filas(sinEtiquetar) do
3:    etiquetas_i = encontrarEtiquetas(etiquetados,sinEtiquetar_i,cantidadVecinos)
4: end for
5: return etiquetas
```

TP1 2 int encontrarEtiquetas(matriz etiquetados, vector incognito,int cantidadVecinos)

```
1: colaPrioridad(norma,etiqueta,vectorResultado) resultados
2: for 1 to size(incognito) do
3:    resParcial = restaVectores(etiquetados_i,incognita)
4:    colaPrioridad.push((norma(resParcial),etiqueta(etiquetados_i))
5: end for
6: vector numeros = vector(10)
7: while cantidadVecinos>0 & noesVacia(resultados) do
8:    int elemento =primero(resultados.etiqueta)
9:    numeros_elemento ++
10: end while
11: return maximo(numeros)
```

Al comienzo del desarrollo de los experimentos pensamos en diferentes maneras de mejorar el procesamiento de las imágenes, ya sea pasandolas a blanco y negro para no tener que lidiar con escala de grises o recortar los bordes de las imágenes, ya que en ellos no hay demasiada información útil (en todas las imágenes vale 0).

Sin embargo, y mas allá de las mejoras que puedan realizarse sobre los datos en crudo, este algoritmo es muy sensible a la variabilidad de los datos. Un conjunto de datos con un cierto grado de dispersión entre las distintas clases de clasificación hace empeorar rápidamente los resultados.
En el siguiente apartado pasaremos a describir una metodología más sofisticada para resolver este problema que mejora tanto los tiempos de ejecución como la tasa de reconocimiento con respecto al método descripto anteriormente.

2.2. Optimización mediante Análisis de componentes principales

El Análisis de Componentes Principales o PCA es un procedimiento probabilístico que utiliza una transformación ortogonal para convertir un conjunto de variables, posiblemente correlacionadas, en un conjunto de variables linealmente independientes llamadas componentes principales.
Esta transformación está definida de tal manera que la primera componente principal tenga la varianza más grande posible, la segunda componente tenga la segunda varianza más grande posible y así sucesivamente hasta encontrarse con la componente de menor varianza en la última posición.
De esta manera será fácil quedarnos con los λ componentes principales que concentren la mayor varianza y quitar el resto. En la sección de experimentación, uno de los objetivos principales será buscar cual es el λ que concentra la mayor varianza de manera tal de optimizar el número de predicciones.
A fines prácticos, lo que hacemos es, a partir de nuestra base de datos de elementos etiquetados, construir la matriz de covarianza M de tal manera que en la coordenada M_ij obtenga el valor de la covarianza del pixel i contra el pixel j.
Luego, utilizando el método de la potencia, procedemos a calcular los primeros λ autovectores de esta matriz. Una vez obtenidos los autovectores multiplicamos cada elemento por los λ autovectores y asi obtenemos un nuevo set de datos.
Sobre este set de datos, ahora aplicamos el algoritmo KNN nuevamente y lo que esperamos ver es un mayor número de aciertos, ya que hemos quitado ruido del set de datos, sumado a mejores tiempos de ejecución, ya que hemos reducido la dimensionalidad del problema.

2.3. Cross-validation

Para medir la precisión de nuestros resultados utilizamos la metodología de cross-validation. Esta consiste en tomar nuestra base de datos de entrenamiento y dividirla en k bloques. Primero se toma el primer bloque para testear y los bloques restantes para entrenar a nuestro modelo, observando los resultados obtenidos. Luego se toma el segundo bloque para testear y los restantes como dataset de entrenamiento. Esto se realiza sucesivamente siempre y cuando queden datasets sin ser testeados.
De esta manera evitamos testear contra datos propios del modelo, lo que podría resultar en que el modelo solo reconozca sus propias imágenes de entrenamiento pero no imágenes fuera de él, que es justamente el propósito de este trabajo.

2.4. Algoritmo PCA

TP1 3 void PCA(matriz etiquetados, matriz sinetiquetar,int cantidadAutovectores)

```
1: matriz covarianza = obtenerCovarianza(etiquetados)
2: vector(vector) autovectores
3: for 1 to cantidadAutovectores do
4:    vector autovector=metodoDeLasPotencias(covarianza)
5:    agregar(autovectores,autovector)
6:    double lamda = encontrarAutovalor(auovector,covarianza)
7:    multiplicarXEscalar(auovector,lamda)
8:    restaMatrizVector(covarianza,auovector,lamda)
9: end for
```

TP1 4 matriz obtenerCovarianza(matriz entrada,vector medias)

1: matriz covarianza, vector nuevo
2: **for** i=1 **to** size(medias) **do**
3: **for** j=1 **to** cant_filas(entrada) **do**
4: $nuevoVector_j = entrada_{(j,i)} - medias_i$
5: **end for**
6: agregar(covarianza,nuevoVector)
7: **end for**
8: **for** i=1 **to** cant_filas(entrada) **do**
9: **for** k=1 **to** cant_filas(entrada) **do**
10: $covarianza_i$ = multiplicarVectorEscalar($covarianza_k$, cantidad_filas(entrada)
11: **end for**
12: **end for**
13: **return** covarianza

TP1 5 Vector metodoDeLasPotencias(matriz covarianza,cantIteraciones)

```
1: vector vectorInicial= vector(cant_filas(covarianza))
2: for 1 to cantIteraciones do
3:    vector nuevo = multiplicar(covarianza,vectorInicial)
4:    multiplicarEscalar(nuevo,1/norma(nuevo))
5:    vectorInicial = nuevo
6: end for
7: return vectorInicial
```

TP1 6 Vector medias(matriz entrada)

```
1: vector medias=vector(cant_columnas(entrada))
2: for i=1 to cant_columnas(entrada) do
3:    suma = 0
4:    for j=1 to cant_columnas(entrada) do
5:          suma + = entrada_{i,j}
6:    end for
7:    medias_i = suma/cant_filas(entrada)
8: end for
9: return vectorInicial
```

3. Análisis

3.1. KNN

En esta sección definimos:

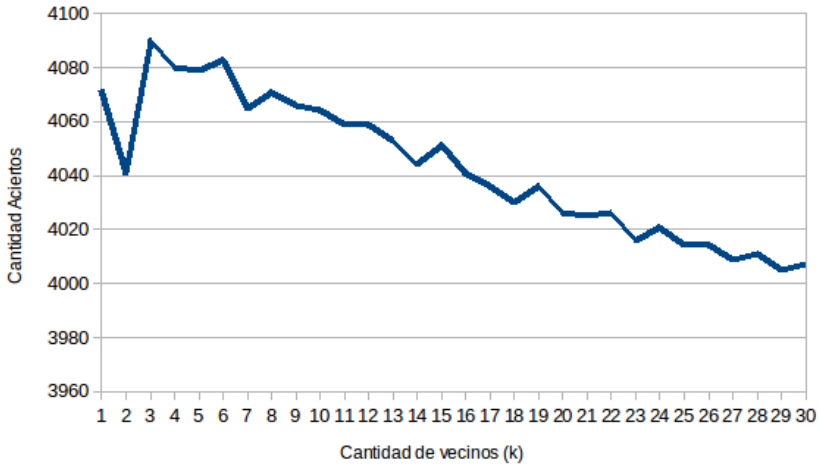
- k: cantidad de vecinos a considerar en el algoritmo kNN.

El análisis sobre el algoritmo KNN (k vecinos más cercanos) se realiza para distintos valores de k. La idea detrás de esta elección de la variable busca entender la variación en la efectividad (cantidad de aciertos) del algoritmo.
Para ello variamos k desde 1 hasta 30 para ver cual era el comportamiento que se obtenía.
Para cada uno de los ks realizamos dos corridas con diferentes k-folds.
El procedimiento de este algoritmo comienza, por cada imágen que queremos averiguar a que dígito pertenece, con su vectorización. Luego resta el resultado a cada uno de los vectores imágen y calcula la norma 2 para saber en cuanto difieren con cada una de las imágenes. Todos esos resultados se acumulan en una cola de prioridad que los ordena de menor a mayor, según las diferencias entre la imágen la cual se quiere averiguar a que clase pertenece y todas las imágenes de la base de datos etiquetada.
Como siguiente paso se toman los k primeros elementos de la cola de prioridad y se verifica a que dígito se corresponden para luego saber cual es el dígito que recibió mas "votosz ver si se produjo un acierto o no. Por lo tanto, una supocición preliminar es que a mayor cantidad de vecinos (o sea, k) menor va a ser la cantidad de aciertos, ya que se empiezan a mirar los elementos de menor prioridad de la cola, eso significa, que se cuentan primero las imágenes que más difieren y eso puede hacer que las chances de acertar el dígito correcto disminuyan.

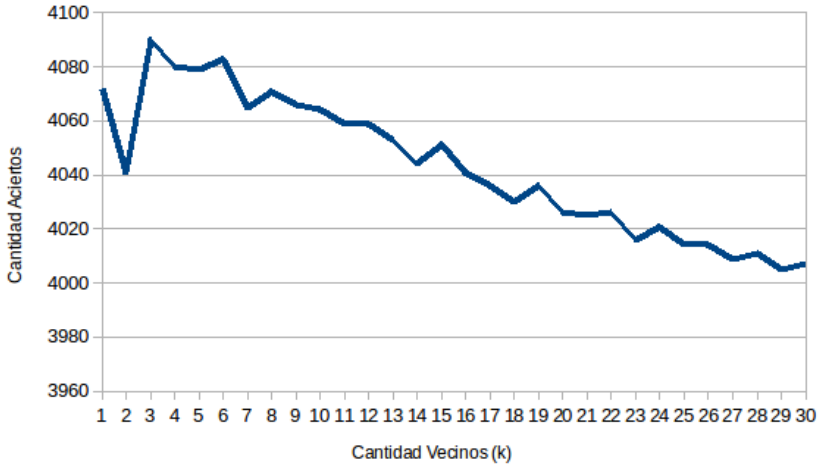
3.1.1. Cantidad de vecinos

Como ya digimos, para analizar cual es el mejor número de vecinos para el cual el algoritmo KNN da una mayor cantidad de aciertos, optamos por variar la cantidad de k vecinos a tomar.

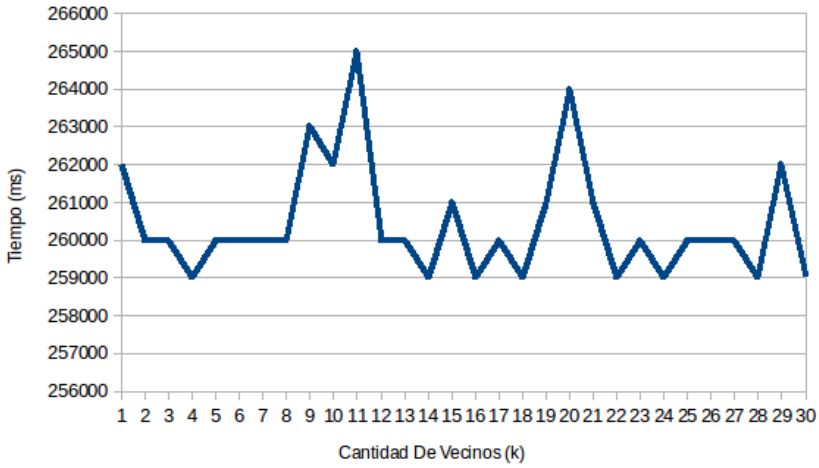
Se prueba entonces el algoritmo KNN para los siguientes valores: $1, 2, 3, 4, ..., 30$.
Para el primer k-fold con el que probamos, que contaba con 4200 imagenes para testear, obtuvimos el siguiente grafico:

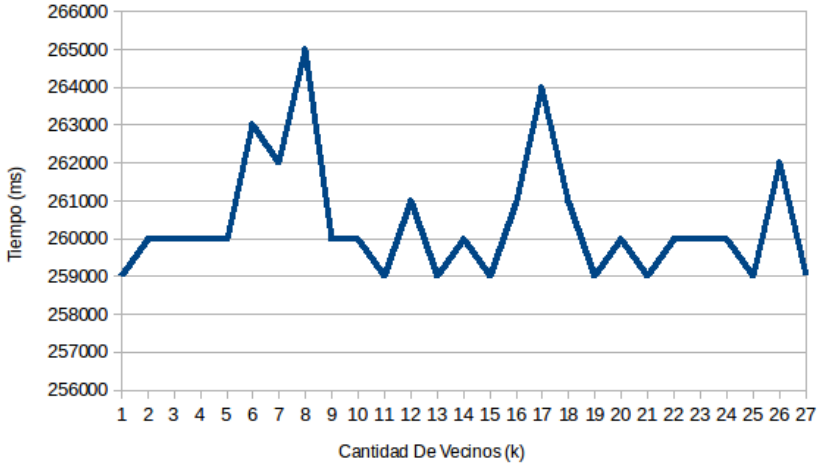


Los resultados son bastante sorprendentes. Para valores muy pequeños de k (k=1,K=2) podemos observar que los resultados son peores que para $k = 3$ lo cual no esperabamos que sucediera. Luego para valores mas grandes si se observa lo que nos decía la intuición, tal que para un gran numero de vecinos se empiezan a perder aquellos que son realmente relevantes.
Aun así estos resultados no nos dejaron satisfechos así que realizamos otro k-fold diferente, con otras 4200 imagenes distintas para testear para ver si obteníamos resultados similares o el k-fold anteriór padecía de alguna particularidad extraordinaria.



Los resultados que obtenemos son muy similares a los anteriores, por lo que concluímos que $k = 3$ es el valor optimo de vecinos a tomar.
Ademas para estas dos tests realizamos una medición de tiempos para ver como se comportaba el algoritmo frente a un cambio en la cantidad de vecinos. Los resultados pueden verse en el siguente grafico:





Como puede observarse, los tiempos de los algoritmos no se ven muy afectados por la variación en la cantidad de vecinos. Es muy probable que esto se deba a que el algoritmo debe comparar a la imagen que se desea comparar contra un numero muy extenso de imagenes, estan en el mismo orden de magnitud, bla bla...

Usamos $k = 3$ porque es el mejor.

3.2. PCA

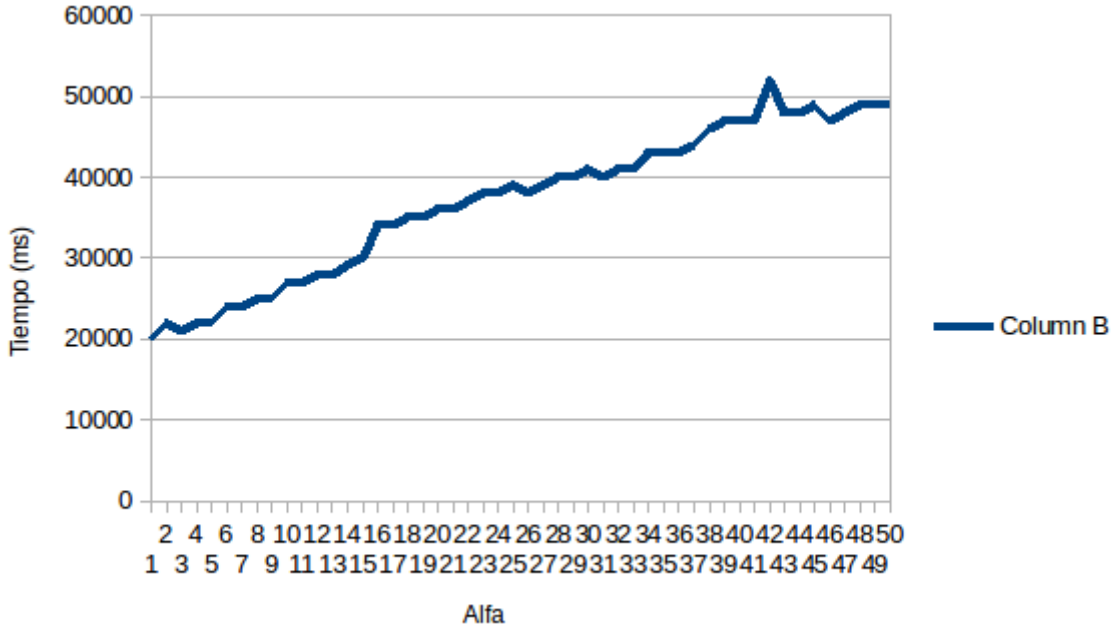
3.2.1. Cantidad de vecinos y α inicial

En esta sección definimos:

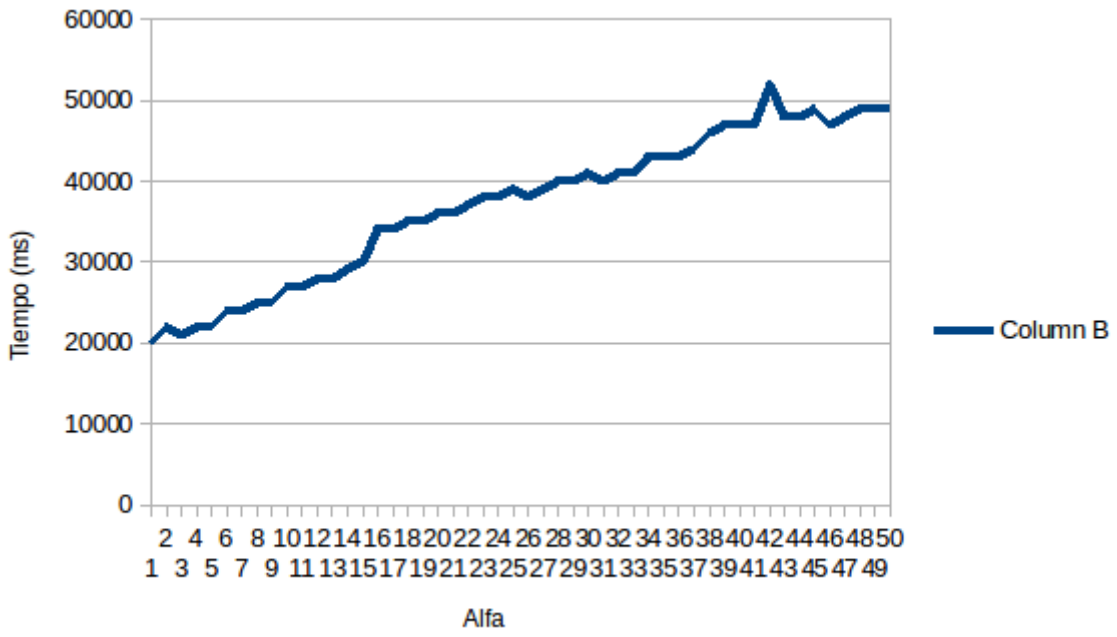
- α: a la cantidad de componentes principales a tomar para el PCA.
- k: cantidad de vecinos a considerar en el algoritmo kNN.

En primera instancia vamos a utlilizar k-folds para intentar determinar el mejor α posible. Supondremos en este momento que el mejor k para este caso es tambien 3 (aunque esto podría no ser asi) y luego testearemos si esto es asi o si para el α encontrado existe algun otro k optimo.
Enfocamos nuestro análisis en obtener un valor óptimo de α. Para este fin, realizamos varias corridas

tratando de maximizar la performance del algoritmo. Dado que este parámetro representa la cantidad de componentes principales a tener en cuenta y teniendo en mente el funcionamiento del algoritmo de PCA, es esperable que valores pequeños no sean beneficiosos (teniendo en cuenta que el máximo a considerar es bastante elevado), pero dado que PCA las ordena en base a su relevancia, se alcance un valor óptimo sin necesidad de considerarlas todas. Para un mismo k-fold con 4200 imagenes para testear, tomamos α desde 1 hasta 50 y graficamos lo obtenido:



Puede verse que para valores pequeños, aumentar en uno el α produce un gran aumento de aciertos. Por ejemplo, para α igual a 1 se obtienen 1107 aciertos, mientras que para α igual a 2 se obtienen 1693 aciertos, esto es un 52 % mas de aciertos.
Para valores mas grandes de α (al rededor de $\alpha = 12$) esta tendencia empieza estabilizarse. Por ejemplo para $\alpha = 12$ se obtienen 3865 imagenes correctamente predecidas, pero para $\alpha = 13$ se obtienen 3881 imagenes correctas, esto es el crecimiento de aciertos es de menos de un 1 %.
Ademas, para este k-fold realizamos una medición de tiempos, que se puede ver a continuación:



En este grafico se puede ver que aumentar el α produce un aumento lineal de los tiempos de ejecución, de lo que se desprende que aumentar la cantidad valores principales no resulta gratuito y tiene cierto costo asociado.
Ademas aumentar de manera desmedida el α puede probocar lo que en machine learning se denomina 'Overfiting'.
Debido a todas las razones expuestas consideramos que con α igual a 14 será el mejor valor que podemos tomar.

La segunda prueba a realizar es, fijando un valor de α, analizar para que cantidad de vecinos se obtiene la mayor cantidad de aciertos. Después de aplicar el algoritmo PCA, se aplica el algoritmo KNN, armando una cola de prioridad para los resultados de aplicar el algoritmo KNN. Lo que se hace es tomar dos imágenes, restarlas y aplicarle la norma 2 para saber en cuanto difiere una imagen de la otra. En la cola de prioridad se encuentran por delante los valores más chicos, o sea, las imágenes del test que más cerca de coincidir están con respecto a la imágen de la base de datos. Por lo tanto, si elegimos una mayor cantidad de vecinos, pueden pasar dos cosas:

- Que sea beneficioso ya que a mayor cantidad de pruebas vamos a tener mas aciertos
- Que sea malicioso ya que a mayor cantidad de pruebas vamos a obtener peores datos, o sea, vamos a mirar las imágenes que menos coiciden con la imagen de prueba de la base de datos.

4. Conclusiones

El análisis realizado nos lleva a sacar una serie de conclusiones en base a lo experimentado.
El algoritmo KNN presenta una efectividad más que aceptable, entendiendo que es una técnica que cuenta con varios años de antiguedad.
Consideramos importante destacar que esto depende en gran medidad de la variación de los datos a analizar. En aquellos conjuntos donde la varianza es elevada y los datos se encuentran muy dispersos, promediar el resultado en base a sus vecinos más cercanos puede no resultar la mejor técnica a implementar.
Teniendo en cuenta esto, la relación costo-beneficio de la implementación y ejecución previa de una optimización como la del algoritmo de PCA, resulta mínima. En todos los casos los resultados mejoraron, dado que concentrar los factores relevantes en componentes especificas del set de datos favorece la vecindad de la que el algoritmo de KNN hace uso.
Dada la característica principal del algoritmo de PCA (ordenar las componentes principales en base a su relevancia), se permite ajustar la cantidad de datos a considerar, dando lugar a una mejora no solo en los resultados, sino también a la performance y al uso de memoria. Como vimos durante nuestro análisis, la cantidad óptima está bastante por debajo del máximo y no tienen ningún beneficio considerar una mayor cantidad de estas.
Si bien estos algoritmos nos muestran resultados interesantes, los tiempos de ejecución utilizando hardware moderno son altos. Este se puede mejorar utilizando técnicas de paralelización utilizando multiples cores o también utilizando instrucciones SIMD (si el procesador lo soporta), sin embargo estos tiempos no son aceptables para aplicaciones que requieren realizar OCR en ¨tiempo real.
Como se menciona al comienzo del trabajo, el preprocesamiento de las imágenes es otro factor que puede mejorar la eficiencia algorítmica. Así como PCA quita ruido del dataset, es posible homogeneizar las imágenes por separado aplicando otros filtros.
Si bien el propósito del trabajo busca encontrar dígitos en imágenes este mecanismo se puede utilizar de un modo muy parecido para encontrar otras características tanto en imágenes como en audios y así etiquetar según clases que no tienen que ver necesariamente con la extracción de dígitos.