DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

"Si nos organizamos aprobamos todos..."

Métodos 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. Introducción

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 cuál 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)
```

Una suposición 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. Ahondaremos mas en detalle en la siguiente sección, cuando pongamos a

prueba cual es la mejor cantidad de vecinos para este algoritmo.
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 descrito anteriormente.

2.2. Optimización mediante Análisis de componentes principales

El Análisis de Componentes Principales o PCA es un procedimiento probabilístico que utiliza una transformación lineal ortogonal de los datos para convertir un conjunto de variables, posiblemente correlacionadas, a un nuevo sistema de coordenadas conocidas estas como componentes principales tal que la mayor varianza de cualquier proyección de los datos queda ubicada como la primer coordenada (llamado el primer componente principal, aquella que explica la mayor varianza de los datos), la segunda mayor varianza en la segunda posición, etc. En este sentido, PCA calcula la base más significativa para expresar nuestros datos. Recordemos que una base es un conjunto de vectores linealmente independientes tal que en una combinación lineal, puede representar cualquier vector del sistema de coordenadas.
De esta manera entonces, será fácil quedarnos con los λ componentes principales que concentran 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 haremos es, a partir de nuestra base de datos de elementos etiquetados, será construir la matriz de covarianza M de tal manera que en la coordenada M_ij se obtenga el valor de la covarianza del pixel i contra el píxel j. Luego, utilizando el método de la potencia, procederemos a calcular los primeros λ autovectores de esta matriz. Una vez obtenidos los autovectores, multiplicando cada elemento por los λ autovectores, obtendremos un nuevo set de datos. Sobre este set de datos, ahora aplicaremos el algoritmo KNN nuevamente y lo que esperamos ver es un mayor número de aciertos, ya que hemos quitado ruido del set de datos (mediante esta base que mencionamos al principio). Esto se suma a mejores tiempos de ejecución, ya que hemos reducido la dimensionalidad del problema.
Generalizando entonces, los supuestos de PCA son:

- Linealidad: La nueva base es una combinación lineal de la base original.
- Media y Varianza son estadísticos importantes: PCA asume que estos estadísticos describen la distribución de los datos sobre el eje.
- Varianza alta tiene una dinámica importante: Varianza alta significa señal. Baja varianza significa ruido.
- Las componentes son ortonormales.

Si algunas de estas características no es apropiada, PCA podría producir resultados pobres. Un hecho importante que debemos recordar: PCA devuelve una nueva base que es una combinación lineal de la base original, limitando el número de posibles bases que puedan ser encontradas.

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. En una primera iteración se toma un bloque para testear y los bloques restantes para entrenar a nuestro modelo, observando los resultados obtenidos. En la siguiente, se toma el segundo bloque para testear y los restantes como dataset de entrenamiento. La metodología se repite k veces hasta iterar todo el conjunto de datos. Finalmente, se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado de error y poder evaluar la performance del método de entrenamiento.

Esta técnica, que es una mejora de la técnica de holdout donde simplemente se divide el set de datos en dos conjuntos (uno para entrenamiento y otro para testing), trata de garantizar que los resultados obtenidos sean independientes de la partición de datos contra la que se está evaluando porque ofrece el beneficio de que los parámetros del método de predicción no pueden ser ajustados exhaustivamente a casos particulares. Es por esto que se utiliza principalmente en situaciones de predicción, dado que intenta evitar que el aprendizaje se realice sobre un cuerpo de datos específico y busca obtener respuestas más generales.

La única desventaja que presenta es la necesidad esperable de correr los algoritmos en varias iteraciones, situación que puede tener un peso significativo si el método de predicción tiene un costo computacional muy alto durante el entrenamiento.

Nosotros realizamos todos los experimentos para un $K = 10$, es decir, siempre experimentamos en 10 particiones distintas y nunca variamos el K. Tomamos la primer partición de las 10 y utilizamos las otras 9 para entrenar, luego tomamos la segunda partición y utilizamos la primer partición y las otras 8 particiones para entrenar, y así sucesivamente, hasta poder testear con todas las distintas particiones.

2.3.1. ¿Qué pasaría si variamos la cantidad de particiones del k-fold?

Si particionamos en menos conjuntos, es decir, elegimos un K chico, lo que sucede es que vamos a tener particiones más grandes y eso resultará en que la base de entrenamiento será más pequeña. Luego el modelo no será tan robusto y se esperaría que las predicciones sean peores. Por ejemplo, si tengo una base de 100 imágenes y las divido en 5 particiones, cada partición va a tener 20 imágenes, van a ser utilizadas 20 imágenes para el testeo y 80 para el entrenamiento. En cambio si parto la base de datos en 2 particiones, van a quedar 50 imágenes para testear y 50 para el entrenamiento. Entonces, a mayor cantidad de imágenes para entrenar, mayor va a ser la probabilidad de tener una mayor cantidad de aciertos.

Ese mismo razonamiento se puede utilizar a la inversa, si tengo mayor cantidad de particiones, voy a tener mayor cantidad de imágenes para entrenar y es muy probable que la los resultados obtenidos sean más fiables.

En otras palabras:

- A mayor cantidad de particiones, mayor cantidad de imágenes de entrenamiento y mayor fiabilidad.
- A menor cantidad de particiones, menor cantidad de imágenes de entrenamiento y menor fiabilidad.

Es importante también tener en cuenta, especialmente para aplicaciones más genéricas (por ejemplo detección de objetos de todo tipo en una imagen), tomar conjuntos heterogéneos. Si tenemos particiones con elementos similares entre si es posible que obtengamos mediciones de modelos muy buenos

para detectar una característica pero muy malos para detectar otra. Supongamos que elegimos mal nuestras particiones y elegimos en cada partición todas imágenes correspondientes al mismo número. Entonces nuestras tazas de reconocimiento serían malas y el algoritmo poco efectivo. Esto también aplica a la inversa. Tomando buenas muestras, aunque sean pequeñas, podemos obtener muy buenos resultados si sabemos que elementos tomar y distribuirlos bien en todas las particiones.

2.3.2. Análisis de promedios

Para obtener un resultado global de los conjuntos de testing mencionados anteriormente, procedimos a realizar un promedio de los aciertos obtenidos para cada k. La ventaja de utilizar este promedio, a diferencia de tomar los conjuntos por separado, es que al tener diversos conjuntos, se mitiga el problema de conjuntos para los cuales es muy efectivo la utilización de un k en particular, por las características de ese conjunto. Si el k resulta efectivo en el promedio de los conjuntos, nos indica que es efectivo globalmente y no para un caso en particular.

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)

```
vector vectorInicial= vector(cant_filas(covarianza))
for 1 to cantIteraciones do
    vector nuevo = multiplicar(covarianza,vectorInicial)
    multiplicarEscalar(nuevo,1/norma(nuevo))
    vectorInicial = nuevo
end for
return vectorInicial
```

TP1 6 Vector medias(matriz entrada)

```
vector medias=vector(cant_columnas(entrada))
for i=1 to cant_columnas(entrada) do
    suma = 0
    for j=1 to cant_columnas(entrada) do
        suma + = entrada_{i,j}
    end for
    medias_i = suma/cant_filas(entrada)
end for
return vectorInicial
```

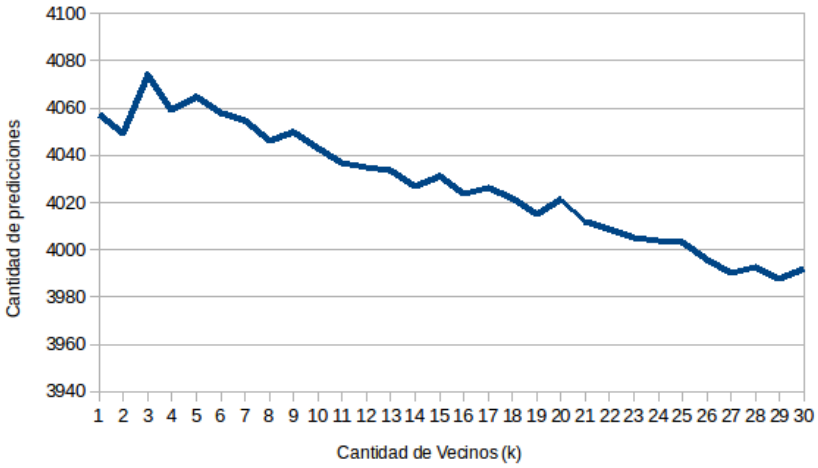
3. Análisis

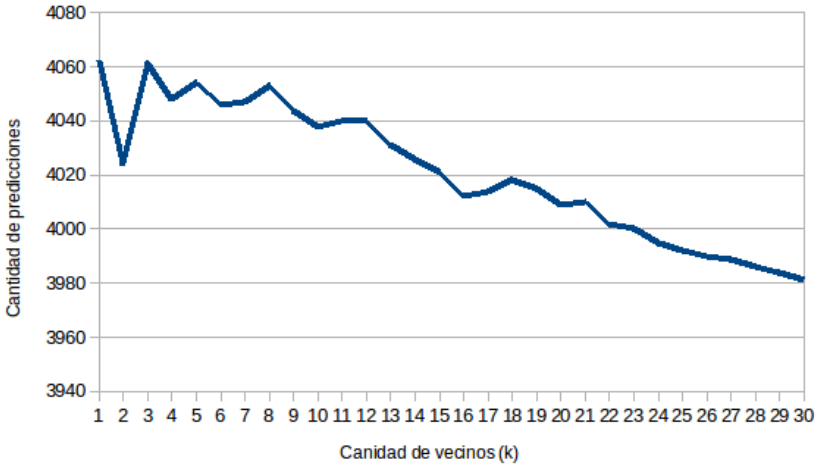
3.1. KNN

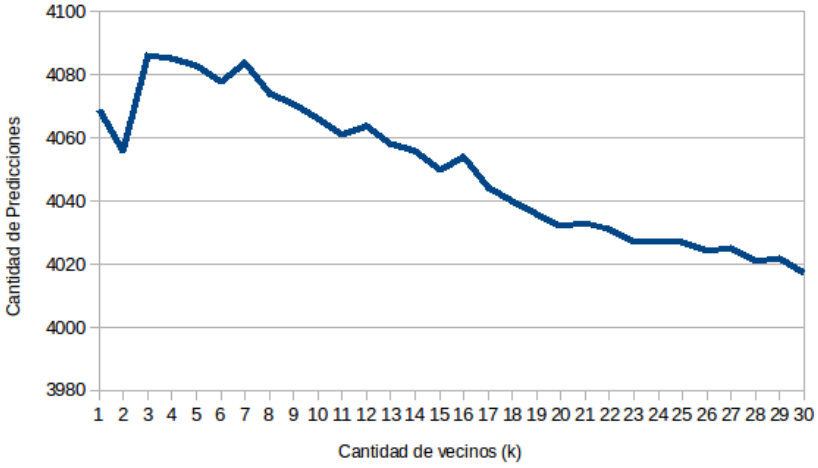
En esta sección definimos:

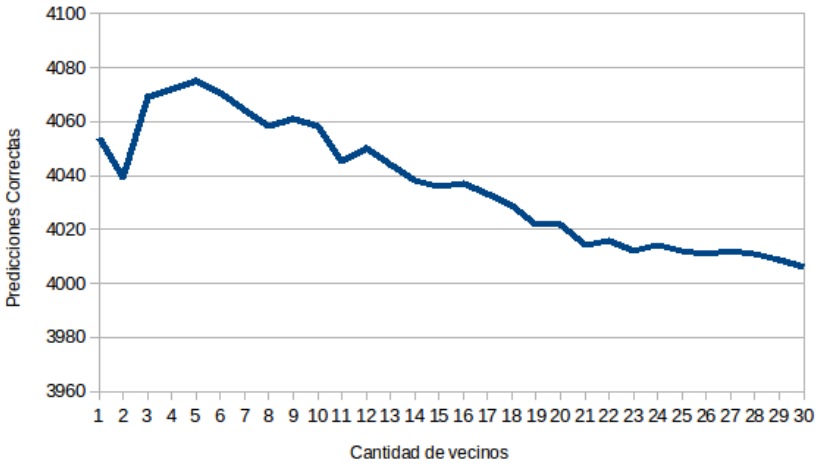
- k: cantidad de vecinos a considerar en el algoritmo kNN.

El análisis sobre el algoritmo KNN (k vecinos más cercanos) se realiza para distintos valores de k, la idea detrás de esto es comprender la variación en la efectividad (cantidad de aciertos) del algoritmo a través del ajust de este parámetro.
Para ello variamos k desde 1 hasta 30 para ver cual es su comportamiento. Para cada uno de los k vecinos realizamos una corrida de cross-validation con 10 conjuntos.
Cada uno de los conjuntos contó con 4200 imágenes a testear. En los siguientes gráficos presentamos algunos de los sets obtenidos:

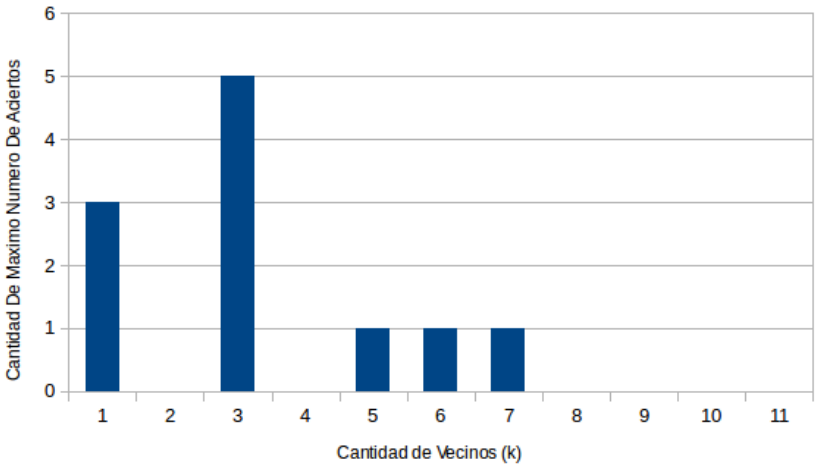

Cantidad de predicciones
4100
4080
4060
4040
4020
4000
3980
3960
3940
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Cantidad de Vecinos (k)


Cantidad de predicciones
4080
4060
4040
4020
4000
3980
3960
3940
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Canidad de vecinos (k)


Cantidad de Predicciones
4100
4080
4060
4040
4020
4000
3980
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Cantidad de vecinos (k)



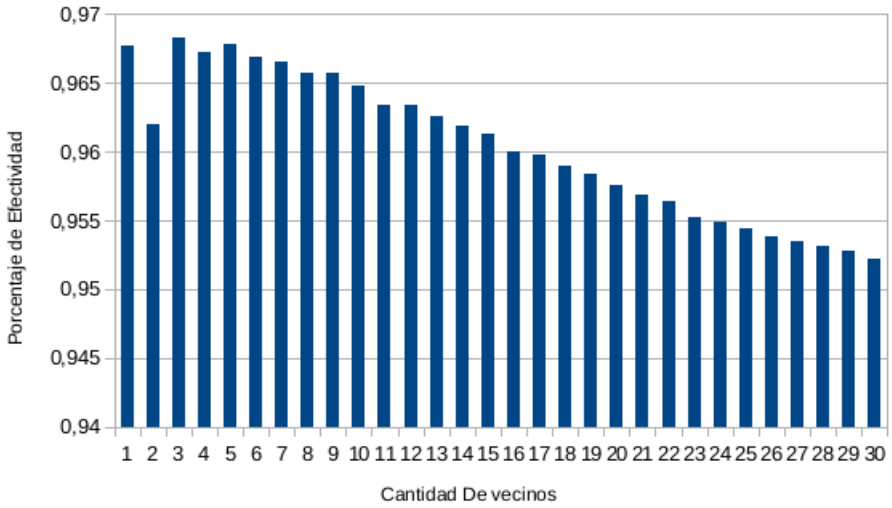
Como se puede ver hay un patrón bastante claro en los experimentos realizados, para valores muy pequeños de k los resultados son bastante buenos. Para $k = 1$ se obtiene el mejor valor para 3 de los sets que testeamos, para $k = 3$ se obtiene que 4 veces es la cantidad de vecinos que produce más aciertos, para el resto de los sets se encuentran también valores cercanos a estos. Para valores más grandes se observa un comportamiento que ratifica nuestra intuición, con un gran número de vecinos se empiezan a perder aquellos que son realmente relevantes y las predicciones empiezan a ser peores. Lo que no se esperaba era el salto en la cantidad de aciertos que vemos en los primeros valores de k pero podemos entonces saber que dentro de los primeros k se encuentran los mejores resultados.
La distribución del mejor k es la siguiente:



Decidimos luego que el mejor valor para k es el que produce más veces el máximo número de aciertos, es decir $k = 3$. La conclusión que podemos extraer de este resultado es que para ciertos casos puede pasar que la imagen que está en primer lugar de la cola de prioridad no sea la correcta, eso pasaría para el caso en que se tiene un dígito que se parece mucho al primero de la cola, pero estos no son iguales.

Por ejemplo teniendo el dígito x, testeandolo con una base de datos grande en la que ningún dígito x' se parece a x, pero teniendo otro dígito y tal que se parece mucho, ese dígito y va a quedar en la primera posición de la cola de resultados. Recordemos que una vez que comparamos el dígito con toda la base de datos, se arma una cola de prioridad, ordenados semejanza entre dígitos. Como decíamos, ese dígito y puede ser considerado como un 'outlier', ya que hace que el dígito más parecido no sea el resultado correcto. Por lo tanto siempre es mejor elegir más vecinos para saber cuales dígitos aparecen

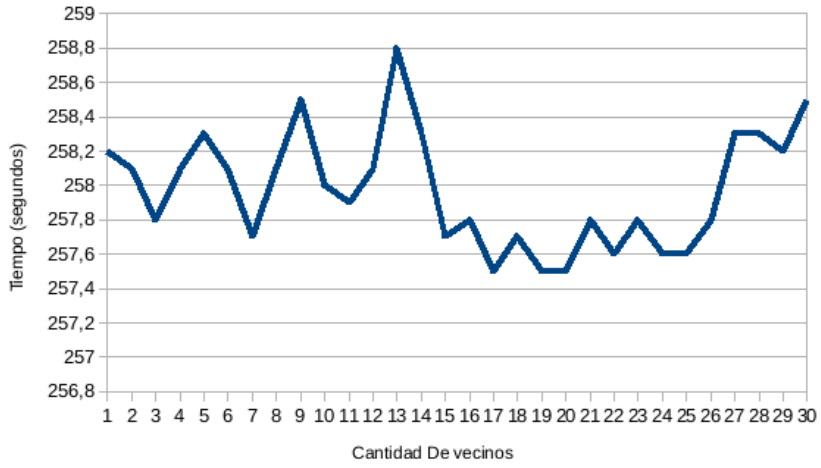
en mayor cantidad y así seleccionar el de mayor cantidad de apariciones.
Teniendo esto en mente decidimos, como explicamos en la sección de desarrollo, observar la efectividad sobre el promedio de los conjuntos de cross-validation. Como se puede observar en la siguiente figura, $k = 3$ es la cantidad de vecinos que acarrea mayor efectividad sobre el global de los conjuntos. Esto indica que no solo $k = 3$ es la mejor elección que mejor se repite en los conjuntos, sino que también es la mejor elección si tomamos todo el dataset, indicandonos que además de ser muy bueno en varios conjuntos, no es una mala elección en el resto.



Pero, ¿Hasta qué cantidad de vecinos conviene tomar? Si elegimos pocos vecinos podríamos tener el problema comentado anteriormente de elegir el dígito más cercano pero no el correcto. Si elegimos muchos vecinos, lo que podría pasar es que estaríamos contando los dígitos que menos se parecen al dígito testeado, o sea, estaríamos eligiendo los peores resultados. Según los resultados obtenidos, nos conviene elegir los primeros 3 vecinos más cercanos, ya que según los resultados testeados es el que más cantidad de aciertos obtuvo.

Otra observación que podemos ver de las imágenes resultantes es que para los valores k pares obtenemos una menor cantidad de aciertos que para los k impares. Esto tiene sentido, ya que si tengo un dígito x a testear y en la cola de prioridad obtengo como los dígitos más cercanos, por ejemplo, los dígitos x, y, x, y. Si elegimos $k = 1$ o $k = 3$ el resultado sería correcto, pero si elegimos un valor k par, podríamos tener un empate entre 2 dígitos y podríamos elegir el dígito incorrecto simplemente al azar, por una elección arbitraria del algoritmo, o por cualquier motivo para desempatar. Entonces por ese simple motivo, podríamos elegir mal el dígito correcto y tener menor cantidad de aciertos. En cambio, para los k impares ese problema no lo tenemos y nos evitaríamos ese inconveniente, por más que después se elija el resultado correcto o no.

Además para estos tests realizamos una medición de tiempos para ver cómo se comportaba el algoritmo frente a un cambio en la cantidad de vecinos. Los valores promediados para cada k pueden verse en el siguiente gráfico:



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 número muy extenso de imágenes que estan en el mismo orden de magnitud. En el grafico da la sensacion que varia mucho la función para distintos valores de k, pero eso pasa porque la escala del eje y es muy chica en comparación a la escala del eje x que es grande, ya que está entre 256 segundos y 259 segundos en el eje y y en el eje x el k varia entre 1 y 30, es por eso tanta variación.

Luego de los test decidimos que vamos a usar $k = 3$ porque nos dio los mejores resultados, basado en la cantidad de aciertos.

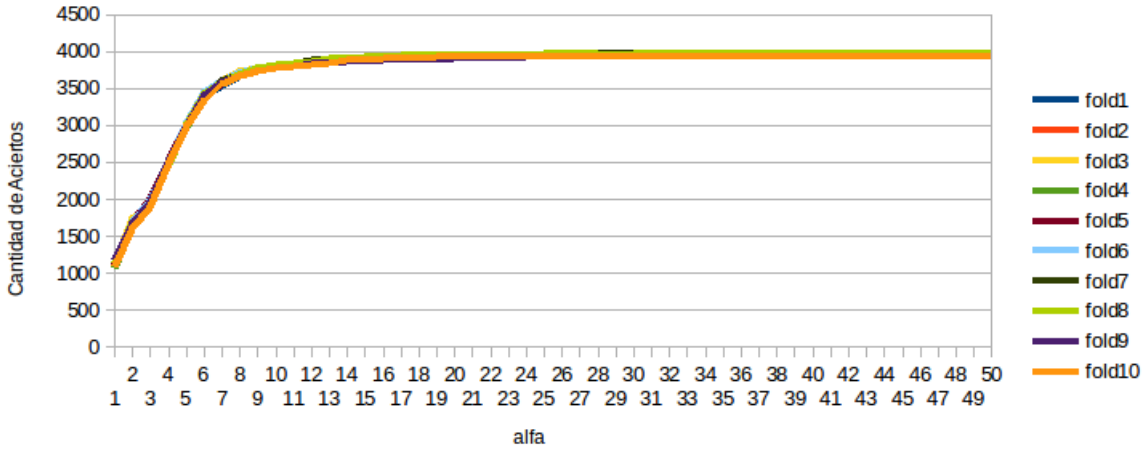
3.2. PCA

3.2.1. Búsqueda del mejor valor de α

En esta sección definimos:

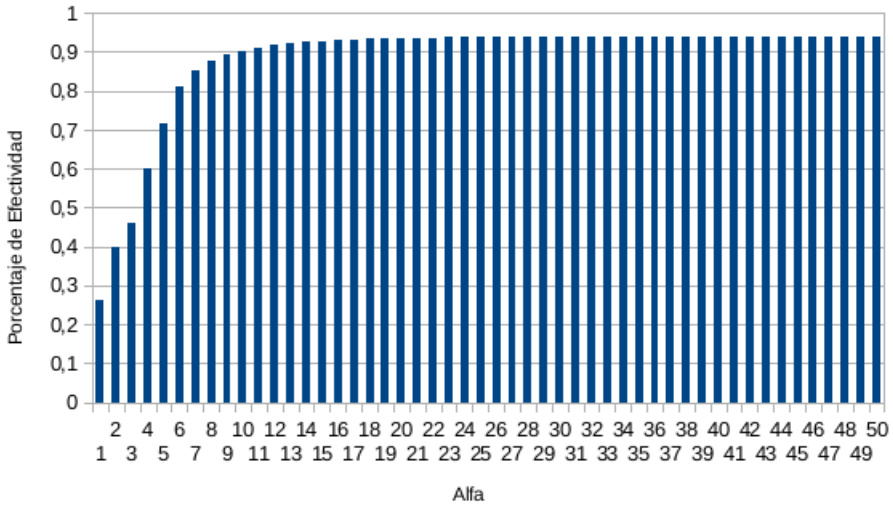
- α: a la cantidad de componentes principales a tomar para el PCA.
- k: cantidad de vecinos a considerar en el algoritmo kNN.

En primera instancia vamos a utilizar el método de cross-validation para intentar determinar el mejor α posible. Supondremos en este momento que el mejor k para este caso es el encontrado en la sección anterior (aunque esto podría no ser así) y luego testeamos si esto es así o si para el α encontrado existe algun otro k que mejora la cantidad de predicciones del sistema.
Por lo tanto enfocamos nuestro análisis en obtener un valor óptimo de α. Para este fin, partimos el conjunto de datos de entrenamiento en 10 subconjuntos y aplicamos cross-validation. 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 el método de PCA las ordena en base a su relevancia, se alcance un valor óptimo sin necesidad de considerarlas todas. Para esta partición de los datos de entrenamiento con 4200 imágenes para testear, tomamos α desde 1 hasta 50 y graficamos lo obtenido:

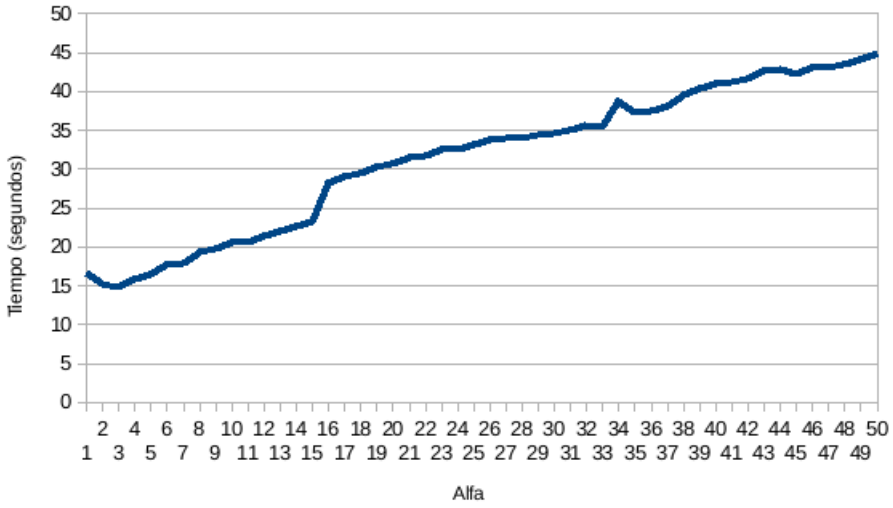


Puede verse que para valores pequeños, aumentar en uno el α produce un gran aumento de aciertos. Por ejemplo, considerando el primer set para α igual a 1 se obtienen 1112 aciertos, mientras que para α igual a 2 se obtienen 1680 aciertos, esto representa un 52 % más de aciertos.
Para valores más grandes de α (alrededor de $\alpha = 12$) esta tendencia empieza a estabilizarse. Por ejemplo para $\alpha = 12$ se obtienen 3845 imágenes correctamente predecidas, pero para $\alpha = 13$ se obtienen 3869 imágenes correctas, esto es el crecimiento de aciertos es de menos de un 1 %.
Graficamos la variación del α hasta el valor $\alpha = 50$ ya que nos dimos cuenta que para valores mayores de 50 la cantidad de aciertos no variaba mas del 1 % (como dijimos anteriormente) por lo tanto, no tenia sentido seguir testeando y sumar esa información que no aportaba nada.

Procedimos también a realizar un análisis de la efectividad global de los conjuntos de cross-validation para $k = 3$ a través de su promedio. Como se puede observar, a medida que crece k, la mejora en la efectividad respecto a $k - 1$ es cada vez más pequeña. Podemos observar como a partir de $k = 12$ se encuentra una asíntota respecto del eje x, esto indica que tomar α entre 10 y 14 es un buen punto para obtener buenos resultados sin desperdiciar recursos.



Además, para este $k - fold$ medimos los tiempos de ejecución y los promediamos para poder ver de que manera varía la ejecución de los algoritmos en función de α:

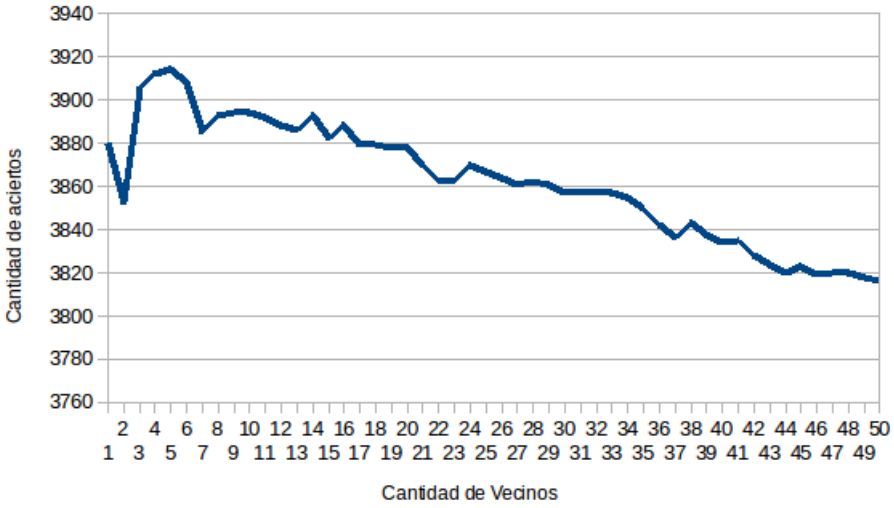


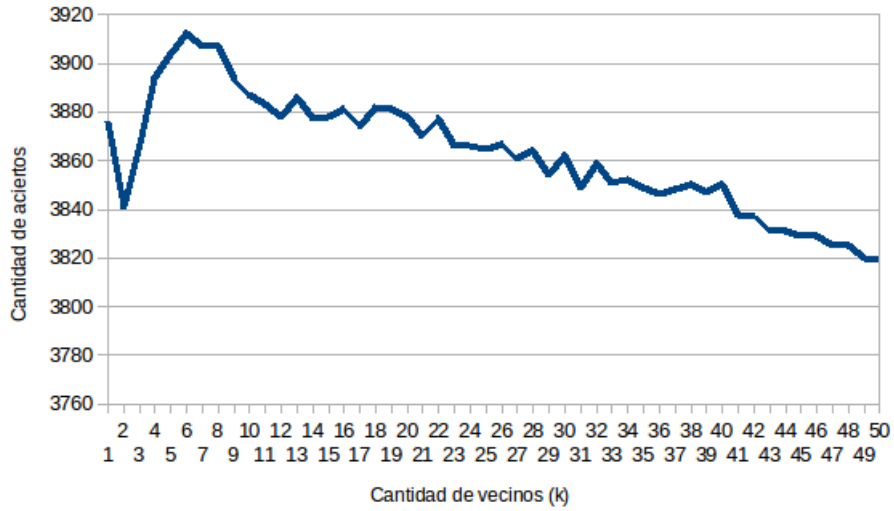
Cabe aclarar que estos tiempos no contemplan todo lo que se considera el 'entrenamiento' del sistema, es decir, todo el preprocesamiento que resultará en encontrar los valores principales. La justificación de esto es que el procedimiento se realizará una vez, para entrenar el sistema y luego, al momento de clasificar las nuevas imágenes este tiempo podrá ser despreciado.
En este gráfico 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 en términos de tiempo de ejecución y tiene cierto costo asociado.
Además aumentar de manera desmedida el α puede provocar lo que en la jerga de machine learning se denomina 'Overfitting' o Sobreajuste, que consiste en que el modelo en vez de buscar patrones sobre los cuales predecir, empieza a 'memorizar' de alguna manera los datos de entrenamiento. Esto puede conducir a que, si bien en la etapa de desarrollo se obtienen buenos niveles de predicción, cuando el modelo es puesto a prueba en algún caso real su desempeño no es el esperado[1].
Debido a todas las razones expuestas consideramos que α igual a 14 será el mejor valor que podemos tomar.

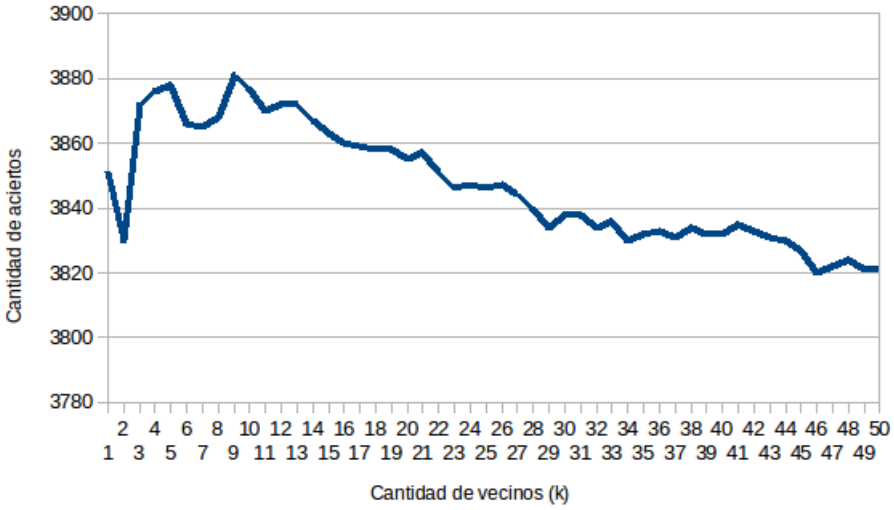
3.2.2. Búsqueda del mejor valor de k

La segunda prueba a realizar es, fijando un valor de α, analizar para que cantidad de vecinos se obtiene la mayor cantidad de aciertos.
Para esto tomamos $\alpha = 14$, volvemos a dividir el conjunto de datos en 10 sets y realizamos cross validation sobre estos, variando el k desde 1 hasta 30.
En el siguientes gráficos mostramos el resultado obtenido para los tres de los sets cuando se varía el número de vecinos:

[1] A Few Useful Things to Know about Machine Learning, Pedro Domingos, Department of Computer Science and Engineering, University of Washington

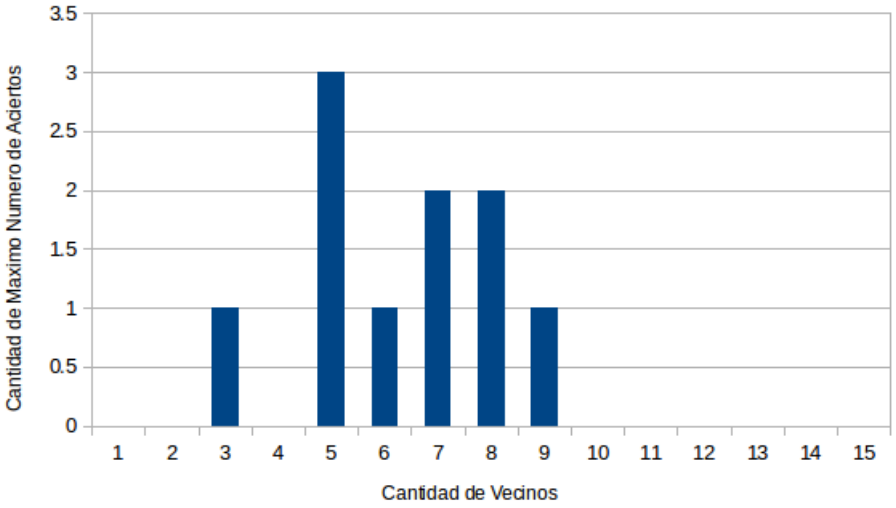




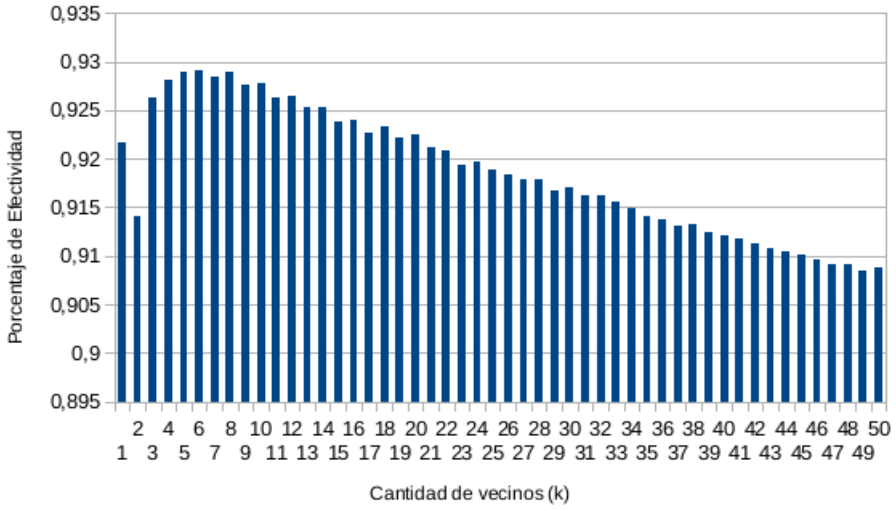


En total, de los diez sets en tres de ellos se encontró que el número de vecinos que maximiza la cantidad de aciertos era 5, en dos se encontró que el máximo fue 7 y 8, y en otros restantes el mejor

fue 3, 6 y 9 vecinos, cada uno.
En el siguiente gráfico expresamos cómo se distribuyeron los máximos en cada conjunto del cross-validation:

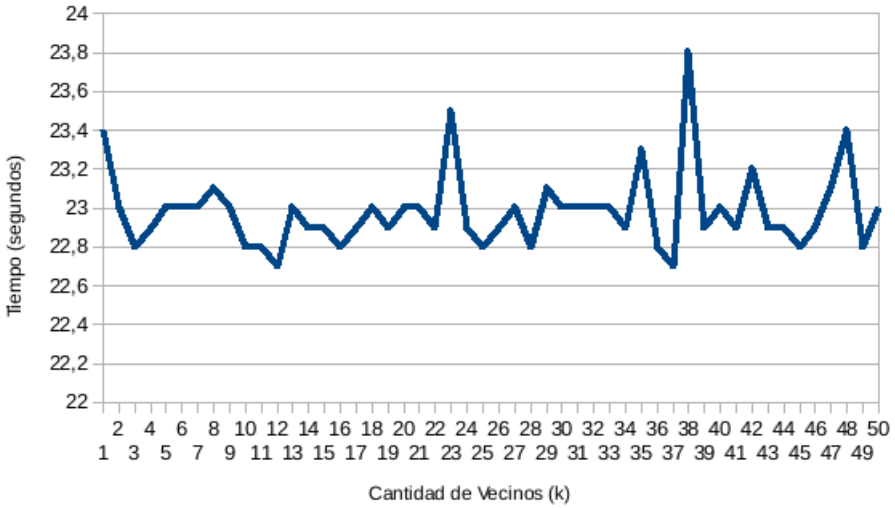


Tomamos como en el resto de los experimentos el promedio de efectividad global entre los conjuntos de cross-validation. El testing se hizo variando k utilizando un α fijo igual a 13 (recordando que vimos que tomar valores entre 10 y 14 son buenas opciones respecto a efectividad y recursos utilizados). Entonces podemos corroborar con la figura a continuación que el pico de efectividad se encuentra tomando valores de k entre 3 y 9. Es interesante observar que utilizando más de 20 vecinos resulta contraproducente, ya que obtenemos peores resultados que tomando simplemente 1, cosa que no ocurre en kNN.



De esto podemos determinar que el k óptimo se encuentra en un rango entre 3 y 9.

Además medimos los tiempos y los promediamos para obtener una idea de cómo afectan las variaciones de k a este método:



Como podemos ver, el número de vecinos continúa sin afectar mayormente los tiempos de ejecución incluso luego de haber reducido la dimensión de la entrada.

4. Resultados

4.1. Resultados del testeo

Del apartado de experimentación podemos deducir que el mejor método de predicción es el método de *knn* que, con $k = 3$, obtiene alrededor de un 96 % de aciertos. Sin embargo esto viene asociado con tiempos de ejecución muy altos y que podrían no resultar apropiados. Recordemos que en el apartado anterior determinamos que para clasificar 4200 imágenes obtuvimos tiempos cercanos a los 4,35 minutos en máquinas modernas.
También vimos que el PCA, si bien no llega a tasas tan altas de predicción como el KNN, obtiene resultados aceptables para $k = 5$ y $\alpha = 14$ encontrados, que rondan entre un 92 % y 90 %. La ventaja de este método es que sus tiempos de ejecución son mucho menores que los del KNN, clasificar 4200 imágenes tardaba 0,36 minutos, o sea un 8,4 % de lo que tardaba la metodología de KNN!.
Luego a la hora de elegir alguna de estas dos metodologías de resolución uno debe sopesar que es lo que más le interesa, resultados rapidos pero con una menor tasa de aciertos o resultados más precisos pero que requieren de tiempos elevados de procesamiento.

5. Conclusiones

El análisis realizado nos lleva a sacar una serie de conclusiones en base a lo experimentado.
El algoritmo KNN presenta una gran efectividad, entendiendo que es una técnica que cuenta con varios años de antiguedad. Sin embargo, los tiempos necesarios para todas las comparaciones resultan considerablemente elevados. Como dato importante de destacar, entendemos que la efectividad de este algoritmo 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. Lo mismo podria ocurrir en situaciones donde los datos se äsemejen̈demasiado por la elección de las características a medir (situación que podría mitigarse eligiendo nuevas formas de representar los datos o realizando un preprocesamiento previo a estos).
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. Si bien es cierto que, como pudimos observar en el análisis, se pierde una efectividad de alrededor de un dígito, atribuimos este comportamiento a algunas de los supuestos que mencionamos que asumía el algoritmo de PCA. Sin embargo, 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 mas que considerable en la performance de aplicar sobre estos el algoritmo KNN y el 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.
Como resultado de esta característica, los tiempos de análisis se reducen drasticamente, todavía lejos de poder implementar este tipo de soluciones en 'tiempo real' pero mucho más cercanos que utilizando solo el algoritmo de KNN.
Como se menciona al comienzo del trabajo y de este apartado, 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.