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 será reconocer imágenes que contienen dígitos a través de la utilización de técnicas simples de Machine learning.
La metodología consiste en lo siguiente:
Tendremos una base de datos con imágenes ya etiquetadas y una base de datos con otras imágenes sin etiquetas. El objetivo se centrará en que el sistema pueda utilizar la base de datos para poder etiquetar de manera correcta la base de datos sin etiquetar.
Para ello utilizaremos primero el método mas intuitivo posible. Esto sucede con cada imagen a etiquetar, buscamos la que más se le parezca en la base de datos etiquetada, y la marcamos con la misma etiqueta. Por supuesto, todavía queda determinar cual es el criterio para decir que dos imagenes se 'parecen'. Eso será descripto en mayor profundidad en el siguiente apartado.
Como segunda idea intuitiva podemos pensar que tal vez tenemos mucha mala suerte (o la base de entrenamiento no es muy buena) y la imagen mas parecida es la de una etiqueta erronea, luego la manera de resolver esto es, en vez de tomar un solo vecino, tomar k vecinos y de esos k vecinos mas cercanos ver que etiqueta es la que más veces se repite. Esta idea básica, es la de KNN que será explicada en mas detalle en el próximo apartado de este trabajo.
Luego, a esta idea intentaremos aplicarle una mejora sustancial utilizando un método probabilístico conocido como PCA que consiste en aplicarle una transformación a la imagen de tal manera de solo quedarnos con aquellas de mayor variabilidad y desechar aquello que pueda estar introduciendo ruido.

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 sus vecinos los k más parecidos (habiendo que definir que es ser "parecido"), y luego de estos k vecinos, determinar cual es la moda.
Para nuestro 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 pseudocodigos sera necesario asumir que todas las estructuras utilizadas almacenan enteros a menos que se indique lo contrario, esto se indica poniendo entre parentesis el tipo de elementos 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)
```

Preliminarmente 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 mejore de alguna manera 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 (quitando la primera) y así sucesivamente.
De esta manera será fácil quedarnos con los λ componentes principales que concentren toda la 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 numero de predicciones.
A fines prácticos, lo que haremos será 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, procederemos a calcular 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 knn nuevamente y lo que esperamos ver es un mayor número de aciertos, ya que hemos quitado ruido del set de datos y mejores tiempos de ejecución, ya que hemos reducido la dimensionalidad.

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
```

2.3. Cross-validation

Para medir la precisión de nuestros resultados utilizaremos la metodología de cross-validation. Esta consiste en tomar nuestra base de entrenamiento y dividirla en k bloques. Ahora tomamos el primer bloque para testear y los demas bloques para entrenar a nuestro modelo y ver que resultados obtenemos. Luego se toma el segundo bloque para testear y los demas de entrenamiento.
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.

3. Análisis

3.1. KNN

Análisis de KNN
Vamos a analizar el algoritmo KNN (k vecinos mas cercanos) para distintos valores de k, fijando un valor de λ, para ver como varía la efectividad (cantidad de aciertos) del algoritmo. Vamos a probar el algoritmo KNN para los siguientes valores:

α: 10 y k: 1, 5, 20, 50, 250.

También vamos a ejecutar para los valores anteriores, 5 pruebas iguales para cada uno, ya que el algoritmo varía la cantidad de aciertos dependiendo de la base de datos que analice y no siempre da el mismo resultado.

Este algoritmo lo que hace es, por cada imágen que queremos averiguar a que dígito pertenece, vectorizarla, restarla a cada uno de los vectores imágen y calcular la norma 2 para saber en cuanto difieren con cada una de las imágenes. Todos esos resultados los vamos metiendo en una cola de prioridad que los ordena de menor a mayor, según las diferencias entre la imágen que quiero averiguar a que dígito pertenece y todas las imágenes de la base de datos.
Luego lo que hacemos es agarrar los k primeros de la cola de prioridad y verificar a que dígito se corresponden para luego saber cual es el dígito que recibió mas ́votosý ver si se produjo un acierto o no. Por lo tanto, 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 mas difieren y eso puede hacer que las chances de acertar el dígito correcto disminuyan.
El algoritmo KNN es muy efectivo ya que tiene aproximadamente entre 85 % y 90 % de aciertos. Pero su déficitésque es muy lento a comparación del algoritmo PCA.

3.1.1. Cantidad de vecinos

Cantidad de vecinos para el KNN.
Vamos a analizar para el algoritmo KNN cual es el mejor número de vecinos para el cual da una mayor cantidad de aciertos. Vamos a probar el algoritmo KNN para los valores siguientes:
k: 1, 5, 20, 50, 250.

A continuación vamos a mostrar los resultados para los valores recién mencionados en forma de graficos.

Una de las cosas que podemos observar es que a menor cantidad de vecinos recorridos, mayor va a ser la cantidad de aciertos, ya que como estoy eligiendo los mejores de la cola de prioridad, voy a obtener mayor presición debido a que los primeros de la cola son los que menos difieren de las imágenes de la base de datos. Por lo tanto, si elijo $k = 1$ voy a obtener la imágen del dígito que más cerca estuvo de la imágen que me pasaron por parámetro, así que no me interesa saber cuales son los resultados de las siguientes imágenes de la cola ya que el primero es el mejor caso posible.

3.1.2. Mejoras en kNN

Cortar el dataset en x valor y comparar para ver si mejora

3.2. PCA

Vamos a probar el algoritmo para distintas medidas de k y α, que van a ser:
k: cantidad de vecinos a considerar en el algoritmo kNN.
α: a la cantidad de componentes principales a tomar.

Vamos a probar el algoritmo para los siguientes valores:

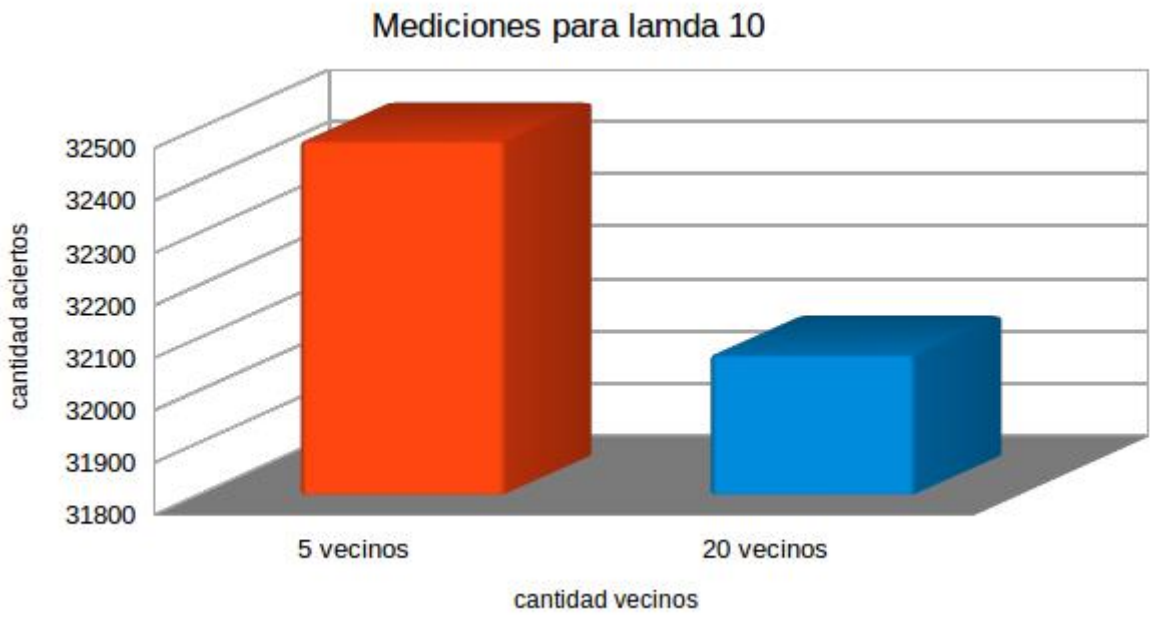
α: 10 y k: 5, 20.
α: 200 y k: 5, 20.
α: 700 y k: 5, 20.
α: 50 y k: 5, 25, 50, 100.

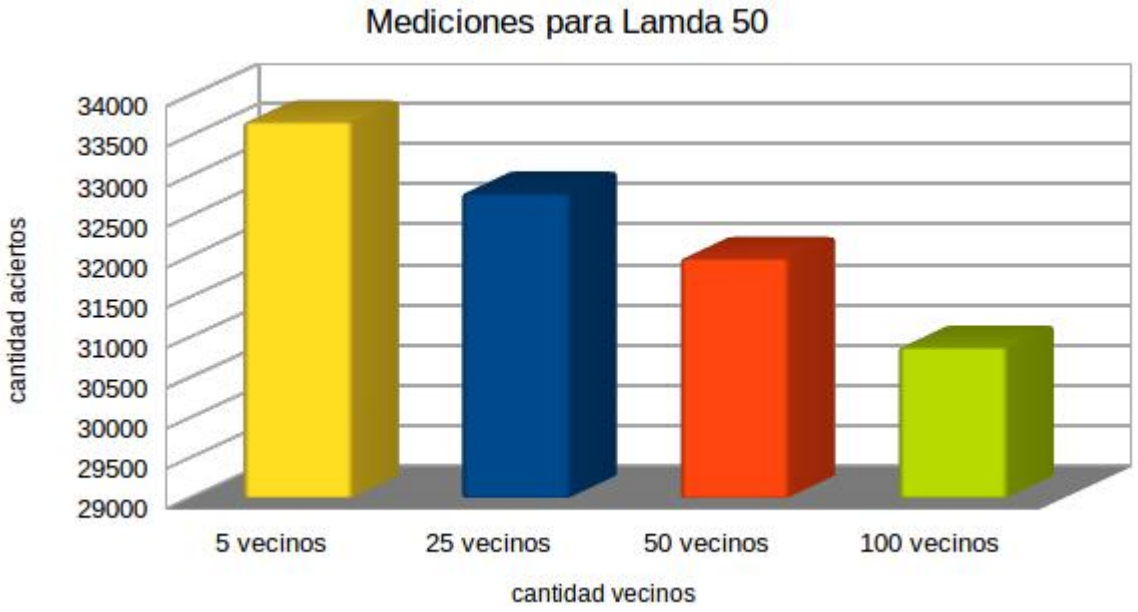
Lo que vamos a probar es, fijando un valor de λ, para que cantidad de vecinos vamos a tener la mayor cantidad de aciertos y así maximizar la cantidad de aciertos. Después de aplicar el algoritmo PCA, aplicamos el KNN y armamos una cola de prioridad para los resultados de aplicar el algoritmo KNN. Lo que se hace es tomar dos imágenes, restarlas y aplicarle la norma 2 para saber en cuanto difiere una imágen de la otra. En la cola de prioridad estan adelante 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:

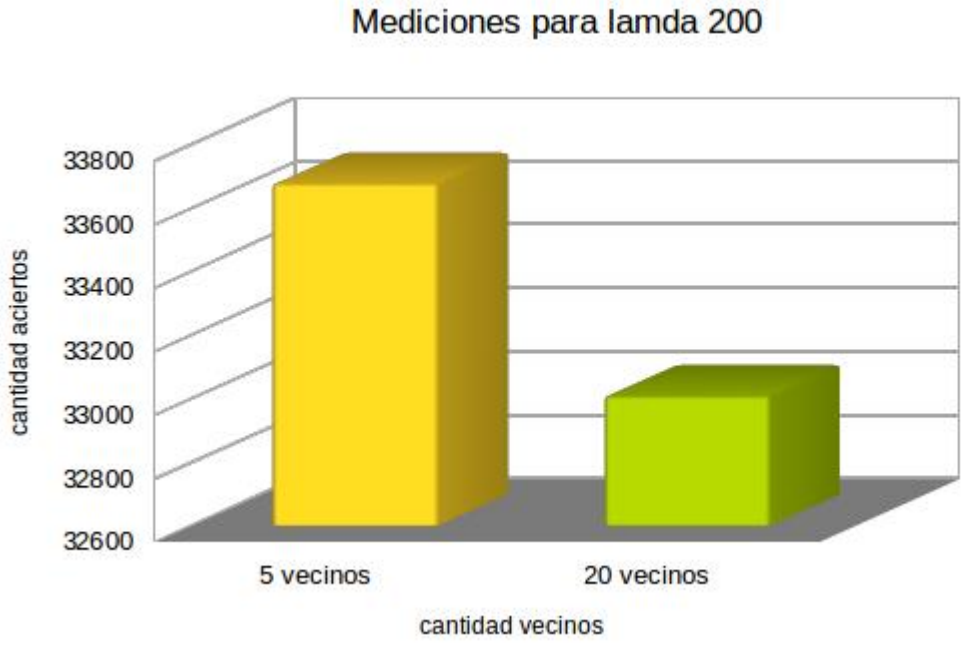
- Que sea beneficioso ya que a mayor cantidad de pruebas vamos a tener mas aciertos
- Que sea malicioso ya que a mayor cantidad de pruebas vamos a obtener peores datos, o sea, vamos a mirar las imagenes que menos coiciden con la imágen de prueba de la base de datos.

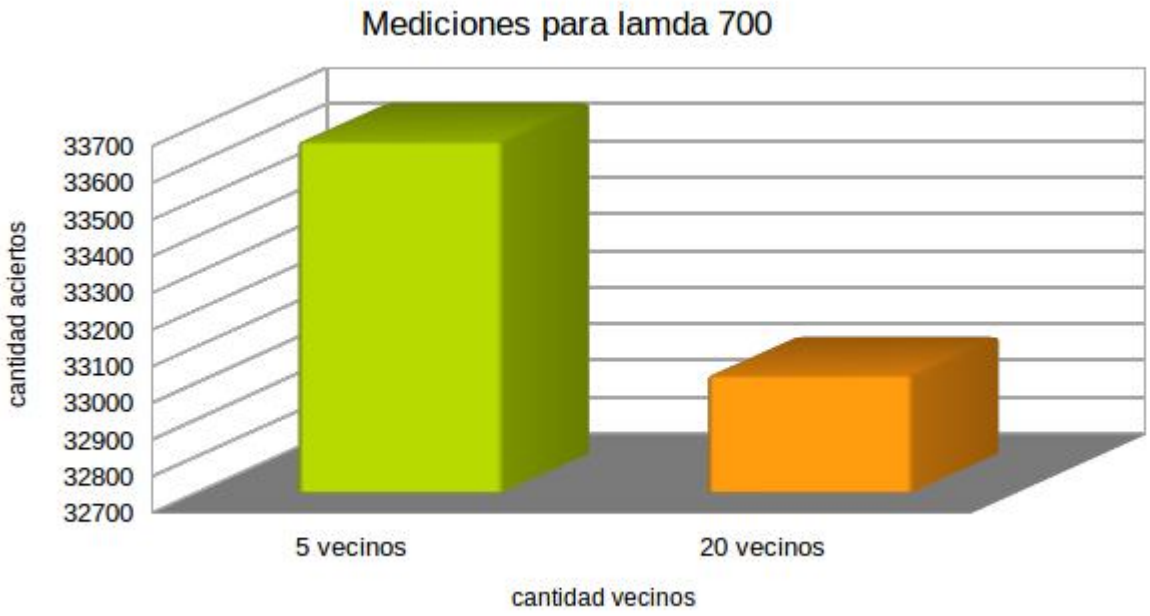
La conclusión que se desprende es que a mayor cantidad de vecinos, el algoritmo empieza a funcionar peor, ya que estamos mirando los vecinos que menos coinciden con la base de datos, debido a que la cola de prioridad los ordena según la menor cantidad de diferencias entre la imágen obtenida y las imágenes de la base de datos.

Por lo tanto, cuanto mas vecinos se exploran, menor cantidad de aciertos hay. Eso se puede ver a través de las siguientes mediciones realizadas para distintos k vecinos y fijando un valor de λ.




Mediciones para Lamda 50
34000
33500
33000
32500
32000
31500
31000
30500
30000
29500
29000
cantidad aciertos
5 vecinos
25 vecinos
50 vecinos
100 vecinos
cantidad vecinos


Mediciones para lamda 200
33800
33600
33400
33200
33000
32800
32600
cantidad aciertos
5 vecinos
20 vecinos
cantidad vecinos



Las pruebas para $\lambda = 10$, $\lambda = 50$ y $\lambda = 200$ las corrimos 10 veces y calculamos un promedio de esas diez. Ese es el valor que se refleja en la cantidad de aciertos.

Las pruebas para $\lambda = 700$ las corrimos 4 veces y calculamos un promedio de esas cuatros. Ese es el valor que se refleja en la cantidad de aciertos.

Para verificar los datos anteriores, revisar el archivo "mediciones y graficos.[en] el directorio ïnforme-resultadostest".

3.2.1. Lambda inicial

3.2.2. Algo más que no me acuerde

4. Resultados

4.1.

5. Conclusiones

1) El algoritmo KNN es muy efectivo ya que tiene aproximadamente entre 85 % y 90 % de aciertos. Pero su 'deficit' es que es muy lento a comparacion del algoritmo PCA.

2) El algoritmo PCA tiene un pequeño menor porcentaje de efectividad con respecto al algoritmo KNN pero es muchisimo mas rapido en tiempo de ejecucion, por lo que vale la pena perder un poquito de efectividad a cambio de muchisimo menor tiempo de ejecucion del PCA.

3) Cuanto mas vecinos exploro, menor es la cantidad de aciertos ya que estoy mirando los vecinos que mas difieren entre si y eso hace que pueda equivocarme a la hora de elegir el digito correcto.

6. Apendice