DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Métodos Numéricos

Trabajo Práctico 2

Reconocimiento de dígitos.

Resumen

En este trabajo pondremos en práctica distintos algoritmos para reconocer una cierta cantidad de dígitos manuscritos. Se trabajará con una base train, a la cual le realizaremos particiones para entrenar los algoritmos. Se implementará el metodo de kNN(k-Nearest Neighbors). Debido a que este es sensible a la dimensión de los objetos a considerar, además se implementará el método de Análisis de Componentes Principales para reducir el tamaño de dichos objetos. Se llevarán a cabo experimentaciones para poder determinar los parámetros óptimos para cada método. Al final del trabajo, se llegarán a conclusiones sobre lo descubierto.

Integrante	LU	Correo electrónico
Armagno, Julián	377/12	julian.armagno@gmail.com
More, Ángel	931/12	angel_21_fer@hotmail.com
Pinzón, Germán	475/13	pinzon.german.94@gmail.com
Porto, Jorge	376/11	cuanto.p.p@gmail.com

Palabras claves:

Learning Machine. kNN. Análisis de Componentes Principales. Auto-Valores. Auto-Vectores. Método de las Potencias. K-fold cross validation

Índice

1. Introducción Teórica

1.1. k Nearest Neighbor y Principal Component Analysis

En el presente trabajo utilizaremos los métodos de kNN y PCA para llevar a cabo el reconocimiento de dígitos manuscritos. Comenzaremos explicando resumidamente los métodos utilizados, de una manera más general. Dado un vector v y un conjunto de vectores U, queremos saber a que clase pertenece v. Nosotros sabemos a que clase pertenece cada vector $u_i \in U$, entonces una manera sencilla de "estimar" a que clase pertenece v en base a la información que poseemos, es comparándolo con cada vector de U. Una manera de efectuar esta comparación es tomando la distancia (norma 2) entre v y cada vector $u_i \in U$ y quedarnos con la clase del u_m que minimice esa distancia sobre todos los demás. Lo que hace el algoritmo kNN es generalizar un poco esta idea. En lugar de quedarnos con la clase del u_m que minimice la distancia a v, kNN toma las clases de los k vectores $u_1, u_2, ..., u_k$ cuyas distancias sean mínimas y elige entre ellas la clase que más aparezca. Para conocer el parámetro k es necesario realizar experimentos probando distintos valores y tomar una decisión en base a la calidad de los resultados.

El algoritmo kNN es conceptualmente fácil de entender e implementar, lo cual constituye una ventaja, pero su desventaja principal es el tiempo de cómputo que requiere. Esto se debe a que, como los vectores representan imágenes, la dimensión de estos vectores suele ser grande. Dado que la idea detras del reconocimiento de imágenes en este trabajo radica en utilizar una base de datos relativamente grande (de ahora en más dbTrain) para determinar que tipo de imagen es la que estamos tratando de reconocer, el costo de computar estas distancias se multiplica por la cantidad de imágenes que tenemos en nuestra base de datos. Vemos que este método, así como está planteado, no escala.

Como ya dijimos, por cuestiones de performance no es conveniente utilizar kNN de manera directa con los datos que tenemos. Lo que vamos a intentar hacer entonces, es realizar una transformación de nuestros datos de manera tal que luego, cuando queramos aplicar kNN no nos resulte tan costoso. Recordemos que nuestros datos son vectores en un espacio $\mathbb{R}^n$, entonces lo que vamos a querer hacer es transformarlos a vectores en otro espacio $\mathbb{R}^\alpha$ tal que $\alpha < n$. Pero también vamos a querer que esta transformación no "cambie por completo" a nuestros vectores, en el sentido de que sigan representando las imágenes que representaban o al menos conserven las "partes relevantes" de ellas. Para poder llevar a cabo esta transformación realizaremos los siguientes pasos:

1. Tomamos los vectores de dbTrain que usaremos para comparar la imagen que queremos reconocer en forma matricial (cada vector una fila de la matriz) y hallar la matriz de covarianza $M \in \mathbb{R}^{nxn}$.

2. Hallar una matriz $P \in \mathbb{R}^{nxn}$ que nos permita disminuir la covarianza de los datos de dbTrain. Esto equivale a buscar las variables que tengan la mayor varianza entre sí y covarianza 0. El objetivo de esto es disminuir la redundancia en nuestros datos.

3. Sea P' la matriz P con las primeras α columnas (este parámetro se fija mediante experimentación), es decir $P' \in \mathbb{R}^{nx\alpha}$, y x_i la i-ésima imagen de dbTrain, realizamos el producto $x'_i = P'^t \widehat{x}_i$, ahora x'_i es nuestra nueva i-ésima imagen de train y está en el espacio $\mathbb{R}^{nx\alpha}$.

4. Sea x una imagen vectorizada cuya clase queremos reconocer, realizamos el producto $x' = P'^t \widehat{x}$ donde $\widehat{x} = (x - \mu)/(\sqrt{n-1})$, μ es la media de las imágenes de dbTrain y n la cantidad de imágenes de dbTrain. Como ahora $\widehat{x} \in \mathbb{R}^\alpha$ y los vectores de dbTrain están en el mismo espacio, podemos aplicar kNN con x' y los x'_i.

Para obtener P lo que hacemos es hallar la base ortonormal de autovectores de M. Sabemos que dicha base existe por ser M simétrica. Entonces la columna i de P va a ser el vector v_i, el i-ésimo autovector de M. Esta base lo que nos permite hacer es "observar" a nuestros datos desde otro lugar, o sea ahora nuestros ejes de coordenadas van a estar en las direcciones donde más varianza existe entre los datos.

Una vez completados estos tres pasos, cuando queramos reconocer a que clase pertenece un vector v, trabajamos con $v' = P'^t v$ y, dado que ahora v' es un vector de $\in \mathbb{R}^\alpha$ al igual que los vectores de dbTrain podemos aplicar kNN.

1.2. Método de la potencia y deflación

En la sección anterior explicamos los métodos que nos permiten reducir la dimensión de las imágenes con las cuales vamos a trabajar y, en base a cierta informacion que tenemos, efectuar comparaciones para predecir a que clase pertenece una imagen. Vimos que, uno de los pasos de PCA es obtener una matriz P cuyas columnas son los autovectores de otra matriz M. El método de la potencia, junto con el método de deflación, nos permite encontrar los autovectores y autovalores asociados a la matriz M.

Dada una matriz $A \in \mathbb{R}^{nxn}$ cuyos autovalores $\lambda_1, ..., \lambda_n$ cumplen $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ... \geq |\lambda_n|$ y $v_1, ..., v_n$ los autovectores asociados, el Método de la potencia estimará v_1 y λ_1. Decimos estimará porque para tener el valor exacto deberíamos calcular un límite, entonces vamos a "simular" este límite computacionalmente con lo cual el resultado puede tener cierto error. Este método necesita de un vector inicial x_0 y un valor de k relativamente grande y lo que va a hacer es calcular $x_{i+1} = \frac{Ax_i}{||Ax_i||_2}$ y $\widehat{\lambda}_1 = \frac{\Phi(Ax_{i+1})}{\Phi(Ax_i)}$ desde $i = 1$ hasta k. El resultado será $x_{k+1} = \widehat{v_1} \approx v_1$ y $\widehat{\lambda}_1 \approx \lambda 1$. Para que esto se cumpla es importante aclarar que la función $\Phi : \mathbb{R}^n \to \mathbb{R}$ que usamos debe cumplir las siguientes propiedades:

- Debe ser continua
- Si $v \in \mathbb{R}^n$ y $v \neq 0$ entonces $\Phi(v) \neq 0$
- Si $v \in \mathbb{R}^n$ y $\alpha \in \mathbb{R}$ entonces $\Phi(\alpha v) = \alpha \Phi(v)$

El Método de la potencia también pide que el vector inicial x_0 no sea perpendicular a v_1, sin embargo a la hora de implementarlo en una computadora esto puede no ser tenido en cuenta y no traerá grandes consecuencias. Esto se debe a que nuestro resultado va a estar arrastrando cierto error a medida que el método itere y este error va a estar cambiando la dirección de x_i (aunque sea mínimamente) y en ese caso dejaría de ser perpendicular a v_1.

Vimos como funciona el Método de la potencia y que nos devuelve el autovalor de mayor módulo junto con su autovector asociado, sin embargo nosotros queremos obtener todos los autovalores y autovectores de A. Esto se soluciona definiendo una nueva matriz $\widehat{A}$ y aplicando nuevamente el Método de la potencia pero ahora sobre $\widehat{A}$. Supongamos que ya tenemos los valores $\widehat{v_1} \approx v_1$ y $\widehat{\lambda}_1 \approx \lambda_1$ obtenidos al aplicar el Método de la potencia sobre A, entonces para obtener $\widehat{v_2}$ y $\widehat{\lambda}_2$ aplicamos nuevamente el método pero ahora sobre $\widehat{A} = A - \widehat{\lambda}_1 \widehat{v_1} \widehat{v_1}^t$. Para el caso general, si queremos obtener $\widehat{v_{i+1}}$ y $\widehat{\lambda}_{i+1}$, tenemos $\widehat{v_i}$ y $\widehat{\lambda}_i$ y nuestra matriz es $\widehat{A}$, definimos $\widehat{\widehat{A}} = \widehat{A} - \widehat{\lambda}_i \widehat{v_i} \widehat{v_i}^t$ y aplicamos el método sobre $\widehat{\widehat{A}}$. A esto se le llama *deflación*. Los autovalores y autovectores de la nueva matriz que definimos van a ser los mismos que la matriz original (la primera o la que definimos en el paso anterior) excepto por el autovalor de mayor módulo y su autovector asociado.

Si bien vimos que para aplicar el método de la potencia en una matriz A se tiene que cumplir que A tenga un autovalor mayor estricto (de multiplicidad 1) en módulo a todos los demás, para poder hacer deflación iterativamente y obtener todos los autovalores y autovectores de A es necesario que valgan las desigualdades de manera estricta: $|\lambda_1| > |\lambda_2| > |\lambda_3| > ... > |\lambda_n|$. Esto debe ser así para que cada nueva matriz que definimos cumpla las hipótesis que requiere el Método de la potencia.

2. Desarrollo

2.1. k-Nearest Neighbors (kNN):

El primer método utilizado es kNN. Como se dijo en la introducción teórica, este método es muy intuitivo, se basa en tomar a cada imagen como un punto en un espacio, y se hace una votación de la moda de los k vecinos cercanos. Todas las imagenes de dbTrain (28x28 pixeles), se encuentran guardadas vectorizadas en una matriz de $\mathbb{R}^{42000x785}$. La primera columna establece el label de cada imagen, las columnas restantes son los pixeles.
Mediante el K que determine la partición, vamos a dividir esa matriz, en dos, una de train, con $(42000/K) * (K-1)$ imágenes y otra de test, con $42000/K$ imágenes. A su vez, vamos a eliminar la primera columna de cada matriz, y nos las vamos a guardar en 2 vectores diferentes. Asi podremos comparar cada imagen de la matriz de test, con todas las imágenes de la matriz de train y al mismo tiempo tener guardados los labels correspondientes a cada imagen. Finalmente la idea algoritmica se detalla en el siguiente pseudocódigo:

Algorithm 1 kNN(int k, matriz Test, matriz Train, vector digTest, vector digTrain)

```
reconocidos = 0
for z = 0; z <Test.filas() ; z++ do
    vector<pair<int, int> > normas2AlCuadrado
    for m = 0; m < Train.filas(); m++ do
        \\ calculamos las distancias
        distanciaAlCuadrado = 0
        for i = 0; i <Test.columnas() ; i++ do
            distanciaAlCuadrado += (Test[z][i] - Train[m][i]) *(Test[z][i] - Train[m][i])
        end for
        if normas2AlCuadrado.size() < k then
            \\ colocamos las primeras k normas
            pair<unsigned int, int> a
            a.first = digTrain[m]
            a.second = distanciaAlCuadrado
            normas2AlCuadrado.pushback(a)
        else
            if haymayor(normas2AlCuadrado, distanciaAlCuadrado then
                \\ si ya tengo k voy sacando las mayores distancias
                int posmayor = dondemayor(normas2AlCuadrado)
                normas2AlCuadrado[posmayor].first = digTrain[m]
                normas2AlCuadrado[posmayor].second = distanciaAlCuadrado
            end if
        end if
    end for
end for
\\ Ahora hago la votación entre los k vecinos
digitos[10] =  0, 0, 0, 0, 0, 0, 0, 0, 0, 0
for int t = 0; t < k; t++ do
    digitos[normas2AlCuadrado[t].first]++
end for
ganador = digitos[0]
for int x = 0; x < 10; x++ do
    if digitos[x] > digitos[ganador] then
        ganador = x
    end if
end for
\\ Si en la votación gano el verdadero label de la imagen de test, sumamos reconocidos
if ganador == digTest[z] then
    reconocidos++
end if
tasaDeReconocimiento = reconocidos/Test.filas()
```

2.2. Principal Component Analysis (PCA):

El segundo método utilizado es PCA. Dado que las imagenes estan representadas por vectores y las dimensiones de de los mismos pueden ser muy grandes, a la hora de analizar las imagenes se requiere una gran cantidad de tiempo de computo así como de memoria. PCA tiene como objetivo redimensionar dichos vectores, siempre que se obtenga un determinado compromiso entre las nueva cantidad de variables y la calidad de las imagenes que ahora se representan. Para esto PCA construye un nuevo sistema de ecuaciones donde en el eje i se representa la i-esima varianza de mayor valor para el conjunto de original de imagenes dadas (representando la i-esima componente principal). De esta manera cada variable esta correlacionada siendo las primeras las de mayor importancia, por lo que se pueden obviar las últimas componentes (ya que serían las que menos importancia

tienenn) reduciendo el número de variables. Para poder llevar a cabo esta transformacion lineal es necesario construir la matriz de covarianza para el conjunto de valores que representan a las imagenes. Para esto dadas dos coordenadas x_i, x_j su covarianza se obtiene como:

$\sigma_{x_j x_k} = (1/(n-1)) \sum_{i=1}^{n} (x_j^i - \mu_j)(x_k^i - \mu_k) = (1/(n-1))(x_k^i - \mu_k v)^t (x_j^i - \mu_j v)$
con v^t *= (1,.....,1)*

De esta forma la covarianza entre dos muestras puede ser calculado como el producto entre dos vectores. Por lo que si A ϵ $\Re^{nx784}$ representa n imagenes de train vectorizadas de 784 variables cada una, la covarianza pueden ser calculadas como:

$M_x = A'^t * A'$; *con* $A' = A/\sqrt{n-1}$ *y* $M_x \epsilon \Re^{784x784}$

obteniendo en la diagonal la varianza de cada coordenada. Con el fin de disminuir las redundancias, covarianza, se plantea un cambio de base. Para esto se diagonaliza la matriz M_x, de forma tal que:

$M_x = P * D * P^t$; *D diagonal y P matriz ortogonal con los autovectores de A como columnas**

Para obtener los autovectores de M_x se empleó el metodo de la potencia* .Dado que la misma es una técnica iterativa que converge al autovector asociado se estableció que la misma se detenga luego de alcanzar un maximo de 3000 iteraciones o cuando la diferencia entre cada coordenada del vector obtenido en la iteración k respecto de la k+1 sea muy poca (en este caso optamos por 0.0000009). Una vez obtenidó el primer autovector procedimos a calcular el segundo para esto aplicamos el metodo de deflación* ambas técnicas serán aplicadas un numero α de veces. En la sección experimentación se evaluara como se comportan los resultados y el tiempo obtenidos al variar este valor. Una vez obtenidos los α autovectores se procedió a llevar a cabo el cambio de dimensionalidad tanto para las imagenes utilizadas como train como para las de test. Para esto se multiplico cada autovector obtenido por las imagenes vectorizadas en A':

$v_i^t * a^{(i)}$; v_i *el i-esimo autovector y* a_i *la i-esima imagen vectorizada*

lo que matricialmente se puede obtener al hacer: A'*P (llamaremos a la matriz resultante $TcTrain$). De esta forma la matriz TcTrain ϵ $\Re^{n*\alpha}$. Para poder comparar las imagenes de test (matriz B), con estos cambios aplicados a las de train, es necesario llevar a las imagenes de test a las mismas dimensiones, por lo que a cada una se le restara la media que corresponda y dividirá por $\sqrt{n-1}$, obteniendo B'. Una vez realizado esto procedemos a cambiar la base tal como se hizo para las de train: TcTest = $B' * P$

Finalizado los cambios de bases obtenemos imagenes en un mismo espacio vectorial cuyas dimesiones son menores a las originales. Pudiendo aplicar distintas tecnicas para llevar a cabo el reconocimiento de los dígitos. Optaremos por aplicar KNN para así poder comparar las diferencias entre tiempo de computo y analisis cuando se busca reconocer dígitos utilizando PCA + KNN y solo KNN.

A continuación se muestra un pseudocódigo de los procedimientos descriptos anteriormente:

Algorithm 2 PCA(matriz Train, matriz Test, int α)

```
1: n = cantidad de imagenes de train
2: \\Se procede a calcular el promedio de las imagenes de train
3: vectorPromedio ← crear(784)
4: for i = 0; i < 784; i++ do
5:     sum = 0
6:     for j = 0; i < n; j++ do
7:         sum = sum + Train[j][i]
8:     end for
9:     promedio[i] = sum
10: end for
11: \\ calculamos la matriz de covarianza:
12: for i = 0; i < n; i++ do
13:     for j = 0; j < 784; j++ do
14:         Train[i][j] = Train[i][j] - promedio[j]
15:     end for
16:     Train[i][j] = Train[i][j]/√(n − 1)
17: end for
18: matriz covarianza ← Train^t * Train
19: \\ obtenemos P la matriz con los α autovectores, de la matriz de covarianza, como columna:
20: P ← metodo de la potencia y de deflacion (covarianza, α)
21: for i = 0; i < cantidad de imagenes de test; i++ do
22:     for j = 0; j < 784; j++ do
23:         test[i][j] = test[i][j] - promedio[j]
24:     end for
25:     test[i][j] = test[i][j]/√(n − 1)
26: end for
27: \\ realizamos la transformación caracteristica para train (TcTrain) y para test (TcTest)):
28: TcTrain ← Train * P
29: TcTest ← Test * P
30: \\ finalmente procedemos a reconocer los dígitos:
31: Knn(tcTriain, TcTest)
```

3. Resultados

4. Discusión

5. Conclusiones

6. Apéndices