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 entre v y cada vector $u_i \in U$ y quedarnos con la clase del u_min 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_min 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 elege 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 tiene la desventaja principal que tiene 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) de información 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 las "partes relevantes" de ellas. Para poder llevar a cabo esta transformación realizaremos los siguientes pasos:

1. Tomar los vectores de dbTrain 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.
3. Sea P' la matriz P con las primeras α columnas (este parámetro se fija experimentación mediante), es decir $P' \in \mathbb{R}^{nx\alpha}$, realizamos el producto $x'_i = P'^t x_i$ donde x_i es la i-ésima imagen de dbTrain y x'_i es esa misma imagen luego de aplicar nuestra transformación. Ahora $x'_i \in \mathbb{R}^\alpha$.

Para obtener P lo que hacemos es obtener 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.

2. Desarrollo

3. Resultados

4. Discusión

5. Conclusiones

6. Apéndices