Memoria Métodos de clasificación

Nerea Jiménez González
Yhondri Acosta Novas

Abril 2020

UNIVERSIDAD
COMPLUTENSE
MADRID

Índice

Índice de figuras

1. Detalles de la implementación

1.1. Lenguaje utilizado

Para la implementación, se utiliza el lenguaje JAVA.

1.2. Procedimiento seguido para la implementación

1.3. Ampliaciones realizadas

Para la ampliación hemos realizado tanto el algoritmo de Bayes como el de Lloyd.

1.3.1. Algoritmo de Bayes

Este algoritmo no necesita ningún parámetro extra, por lo que sólo recibe los datos de las muestras agrupadas por clases.
Lo primero que calculamos son las medias de cada clase mediante la función *calculateAverage*. Para calcular las medias sumamos todas las matrices de muestras de cada clase con la función *plusArray*, y despúes dividimos el resultado entre las cantidad de muestras de dicha clase con la función *divide*. Estas dos funciones pertenece a la clase *Matrix*. Hacemos esta operación para cada clase, y guardamos los resultados en las variables *averageClassX*, donde x es el número de la clase a la que corresponda la media calculada.
Una vez tenemos ambas medias, ya podemos saber a qué clase pertenece una muestra dada. Para ello utilizamos la clase *whichClassBelongTo*, que recibe una matriz con los datos de la muestra a clasificar.
En esta clase primero restamos a la muestra el centro de la clase que queremos comprobar si pertenece a esta con la función *minus*, y después calculamos la distancia con la función *distanceBayes*. La distancia más pequeña será la que decida a que clase pertenece nuestra muestra.

1.3.2. Algoritmo de Lloyd

Este algoritmo recibe los parámetros de tolerancia, número máximo de iteraciones y los centros a usar. La razón de aprendizaje no es un parámetro como tal ya que es una constante global. También reciben los datos de las muestras.
Lo primero que hacemos es actualizar los centros, para ello usamos la función

updateCenters. Esta función realiza las iteraciones necesarias actualizando el centro que se vaya eligiendo. La condición de parada para dejar de iterar es o bien que llegue hasta el máximo de iteraciones, *maxIterations*, o bien que ya no sea necesario seguir actualizando. Para esta segunda condición usamos la función *keepUpdating*, la cuál calcula la distancia Euclidea y si es menor que la toleracia calculandola con ambos centros, devuelve true, de forma que se para de iterar. En caso contrario, devuelve false y se sigue iterando si no ha llegado a *maxIterations* iteraciones.
Cada iteración recorre todas las muestras de ambas clases. Para cada muestra se decide que centro actualizar, para lo cual utilizamos la función *whichCenterToUpdate*. Esta función recibe la muestra en forma de array la cual va a decidir que centro vamos a actualizar. Para ello se realizal los cálculos necesarios para obtener la distancia con ambos centros. La distancia más pequeña decide que centro actualizar.
Una vez sabemos que centro vamos a actualizar, utilizamos la función *updateCenterChoosed*, a la cuál pasamos el centro que hemos elegido y la muestra que hemos utilizado para ello. Esta función actualiza el centro mediante la función $c_j(k+1) = c_j(k) + \gamma(k)[x(k) - c_j(k)]$.
Los centros actualizados se van guardado en la matriz *updateCenters*, mientras que la matriz *oldCenters* contiene los valores de los centros antes de comenzar la iteración. Esto lo hacemos porque necesitamos saber ambos valores para la función *keepUpdating*. La matriz *updateCenters* pasa a ser oldCenters al comienzo de una nueva iteración.
Una vez tenemos los centros actualizados, ya podemos saber a que clase pertenece una muestra dada. Para ello utilizamos la función *whichClassBelongTo*, la cual recibe una matriz con los datos de la muestra a clasificar. Para saber a que clase pertenece, calculamos las distancias respecto a cada centro, y aquella con valor más pequeño es a la que pertenece la clase.

2. Código ejecutable

2.1. Simulación

3. Manual de usuario

Memoria Práctica 3
Mayo 2020
Ult. actualización 29 de abril de 2020