Minería de Datos, Ingeniería Informática
Universidad de Valladolid

Práctica de Mapas Auto-organizados

Daniel González Alonso

3 de noviembre de 2016

1. Introducción

En este documento se explican los pasos desarrollados, los resultados obtenidos y la valoración en general de los resultados de la práctica de Mapas Auto-organizados (SOM) y SOM + MLP de la asignatura Minería de Datos de Ingeniería Informática, Universidad de Valladolid.

Para la creación del Mapa Auto-organizado se empleó Octave en su versión 4.0.3 para Windows. Para la clasificación con el Perceptrón Multicapa se empleó WEKA en su versión 3.8.0.

2. Datos iniciales

Los ficheros de datos que se daban constaban de datos con 40 propiedades y su respectiva clase codificada mediante 10 valores entre 0.1 y 0.9, siendo el mayor (siempre 0.9) el número de la clase. Se aportaba por un lado un fichero de entrenamiento llamado `dígitos.entrenamiento.normalizados.txt` con 270 instancias y por otro un fichero de test llamado `dígitos.test.normalizados.txt` con 70 instancias.

La implementación del mapa auto-organizado se encuentra en el fichero `SOM.m`. Para la creación del mapa auto-organizado, se decidió que el número de filas del mapa fuese de 12 y el de columnas de 8, por otro lado se emplearon 50 épocas.

Lo primero que hice para crear el mapa fue cargar el fichero de entrenamiento en una matriz, añadirle una columna al final llena de unos y después la matriz por filas. Este

proceso se conoce como normalización extendida, y sus objetivos son, por un lado que los datos tengan norma 1 y por otro, al añadir una dimensión más, evitar solapamientos entre los datos.

Una vez que tenemos los datos cargados, continué creando la matriz que contenía los pesos de las neuronas del mapa. Ésta matriz, tenía que tener el número de neuronas como número de filas (12 filas * 8 columnas) y el número de propiedades después de la normalización extendida como número de columnas (41). Los valores de los pesos se inicializan con valores generados aleatoriamente entre -0.5 y 0.5, y después se normalizan por filas, es decir, por neuronas.

3. Fase de aprendizaje

Una vez que tenemos todos los datos, podemos empezar con la fase de aprendizaje, esta fase es un proceso iterativo de 50 épocas, donde en cada época por cada dato del conjunto de instancias tras la normalización extendida se hace lo siguiente:

1. Calculamos la neurona ganadora: la neurona ganadora es aquella que para la entrada y época actual, da como resultado la menor distancia. La distancia la calculamos mediante el coseno del ángulo puesto que los cálculos parecían más sencillos ya que:

$$\cos(\overline{x}^v, \overline{w}_i) = \sum_{j=0}^{n-1} x_j^v w_{ij} \tag{1}$$

   Siendo $\overline{w}_i$ el vector de pesos de una neurona y $\overline{x}^v$ la entrada actual. La neurona ganadora con éste cálculos de las distancias es aquella que cumple:

$$I = \max_{i \epsilon H} \cos(\overline{x}^v, \overline{w}_i) \tag{2}$$

2. Calcular el conjunto de neuronas $N(I)$: El conjunto de neuronas $N(I)$ son todas aquellas neuronas dentro de un cuadrado alrededor de la neurona ganadora, hay que tener en cuenta que para calcular este conjunto, en caso de que la neurona ganadora este cerca de un borde, parte del cuadrado puede estar fuera del mapa, en este caso, dentro del conjunto $N(I)$ también se incluirán las neuronas situadas en el otro lateral del mapa de forma cíclica. El *"radio"* del cuadrado se calcula inicialmente como mín$(n_filas, n_columnas)/2$ y éste decrece en cada época hasta llegar a 0, esto quiere decir que cuando llegue a 0 el conjunto a actualizar solo incluirá a la neurona ganadora.

3. Actualizar el conjunto de neuronas $N(I)$: La actualización de este conjunto de neuronas consiste en actualizar los pesos de las neuronas de acuerdo con la siguiente fórmula:

$$\overline{w}_i(t+1) = \frac{\overline{w}_i(t) + \alpha(t)\overline{x}^v}{\|\overline{w}_i(t) + \alpha(t)\overline{x}^v\|} \tag{3}$$

El valor de $\alpha(t)$ lo calculo como:

$$\alpha(t) = \frac{\alpha_0}{1 + \frac{t}{P}} \tag{4}$$

Siendo t la iteración actual, P el número de instancias, y α_0 el valor 25.

4. Creación de los ficheros

Una vez acabado el proceso de aprendizaje, se han de crear dos ficheros CSV en forma de tabla, los cuales contendrán en las columnas las neuronas (número de filas * número de columnas), y por filas cada instancia del conjunto de datos, de forma que el contenido de la celda x_{ij} en el fichero sea la distancia de la neurona j a la instancia i. Además, también hay que añadir una última columna que contendrá la clase con la que ha de quedar clasificada esa instancia. El primero de los ficheros contendrá las distancias al conjunto de entrenamiento y se ha llamado `distancias_entrenamiento.csv` el segundo contiene las distancias al conjunto de test, y se ha llamado `distancias_test.csv`. Para elaborar estos ficheros lo único que hay que hacer es tomar los datos, hacerles la normalización extendida de nuevo y aplicar la fórmula número 1. Además después también se aplicó un filtro que elevaba todos los datos de las distancias generadas anteriormente a la cuarta.

5. Clasificación con WEKA

Una vez que tenemos los ficheros CSV, la siguiente parte de la práctica consistía en clasificar los datos mediante un Perceptrón Multicapa en el programa WEKA. Para ello, primero hice una fase de prepocesado tanto en el fichero de entrenamiento como en el de test mediante el filtro *NumericToNominal* en la columna de clase, generando los ficheros `distancias_entrenamiento.arff` y `distancias_test.arff`. Una vez con estos ficheros, los introduje de nuevo en WEKA (sin normalizar para no perder el filtro de elevar a la cuarta) en un clasificador *MultilayerPerceptron* con 500 épocas. Los resultaros obtenidos fueron bastante impresionantes, con tan solo 4 de las 70 entradas del fichero de test mal clasificadas, lo que hace un 94.2857 % de aciertos.

6. Clasificación con etiquetado por Neuronas

Como la parte de la implementación del etiquetado por neuronas en Octave no parecía muy complicada, decidí también implementarlo (el código está comentado en el código del fichero `SOM.m`). Para este etiquetado, lo único que tenía que hacer era crear una matriz a la que llamé *m_salida_esperada* donde asignar a la mejor neurona para cada instancia su correspondiente clase en el fichero de datos. Después usé el fichero de test para comprobar los resultados anteriores, comparando la clase de cada instancia con la clase predicha para ella en la matriz *m_salida_esperada*. Los resultados de este etiquetado son algo inferiores respecto al etiquetado con WEKA, con resultados variando entre un 82.857 % y un 90 % de aciertos.