Minería de datos Práctica 1:Clustering knn-means

Jose Ignacio Sánchez
Josu Rodríguez

24 de octubre de 2014

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. Introducción

El objetivo principal de esta práctica es obtener la capacidad de formular un algoritmo de aprendizaje automático de clasificación ***No-Supervisada***. Por otra parte, se trabajarán la capacidad de sintetizar uns técnica de aprendizaje automático no-supervisado, conocer su coste computacional así como sus limitaciones de representación y de inteligibilidad

2. Recursos

- PC con aplicación Weka.
- Bibliografía.
- Librerías de Weka.
- Manual de Weka.
- Guía de la práctica.
- Ficheros para los datos de la práctica: food.arff, colon.arff.
- Otros ficheros que no están en formato *.arff*:
  - En formato *.txt*: ClusterData.atributos.txt (este fichero si tiene la clase asociada para evaluar la calidad del *clustering* en ClusterData.clase.txt).
  - E formato *.csv* bank-data.csvclustering

3. Clasificación NO-supervisada o *Clustering*

(*Definición) 3.1 Se considera* ***clasificación no-supervisada*** *cuando el conjunto de entrenamiento no están las instancias etiquetadas con el valor de la clase. Es un experimento exploratorio, que trata de agrupar las instancias en grupos definidos por similitud entre las características de las instancias que pertenecen al mismo grupo y disimilitud entre las que pertenecen a grupos distintos. Tecnicamente estos grupos son llamados Clusters.*

3.1. Clustering *k-means*

4. Diseño

Estructuramos la ejecución del algoritmo en fases como se puede ver en la figura 1 , las cuales se detallan a continuación.

Primera fase: carga de datos y configuración

Inicialmente se carga la configuración establecida por el usuario en el fichero **kmeans.conf**, es decir: path del fichero y su formato, tipo de inicialización para el *codebook*, número de clusters, distancia a utilizar, número de clústers deseados, elección manual o automática sobre la normalización y diversas opciones más, especificando datos sobre el fichero que se utilizará para las instancias.

Segunda fase: Preproceso de datos

En el preproceso de datos se normalizará o no, dependiendo del parámetro indicado por el usuario. Si el parámetro es 0 no se normalizará, si es 1 se normaliza y si es 2 se hará uso del método experimental para decidir la idoneidad de la normalización.

Tercera Fase: Algoritmo K-means

En esta fase se implementa el algoritmo **K-means**.

1. En primer lugar inicia los *centroides* con el criterio establecido por el usuario, o la matriz de bits de pertenencias.
2. Recorre las instancias del conjunto y calcula la distancia a cada uno de los *codeword* actualizando la matriz de bits de pertenencia,el valor del bit es uno si es el centroide más cercano a la instancia.
3. Se calcula de nuevo el vector promedio para cada cluster.
4. Iterar los pasos dos y tres hasta converger.

Cuarta Fase: Evaluación

En esta fase se tratará de automatizar la evaluación del algoritmo frente a los datos obtenidos con distintas ejecuciones, variando los parámetros o incluso con los resultados obtenidos con el modelo de **K-means** que ya implementa el API de **weka**.

Dependencias

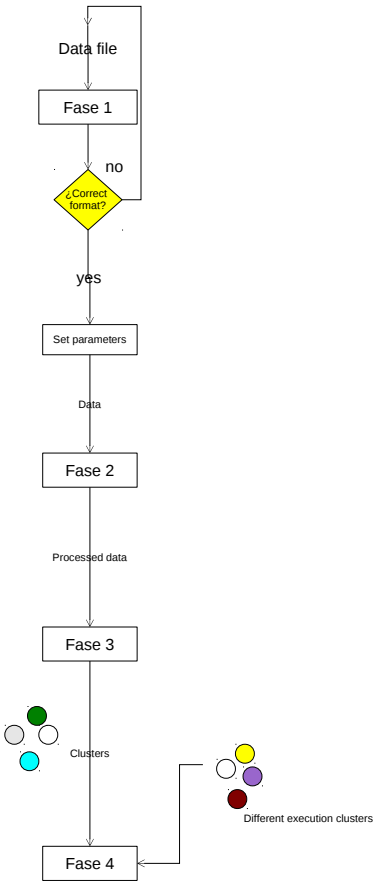

Figura 1: Dependencias del sistema

4.1. Algoritmo en pseudocódigo

```
Let k be the number of clusters to partition the data set
Let X = x_1, x_2, ..., x_n be the data set to be analyzed
Let M = m_1, m_2, ..., m_k be the code-book associated to the clusters
Let dist(a, b) be the desired distance metric
Let B = B_11, B_12, ..., B_nk be the temporary pertenece bit matrix

Ensure: C = C_1, C_2, ..., C_k set of clusterized instances

Begin:

  //randomly initialize the first centroids
  for each m_j
    m_j = randomsample(X)
  end

  //assign dataset instances to each cluster generated by the centroids
  for each x_n
    B_nj = 1 si argmin dist(x_n, m_j) = m_j \foreach m_j si no B_nj = 0
  end

  for each B_nj
    if B_nj == 1
      C_j.add(x_i)
    end
  end

  //iterate the algorithm generatin new centroids based on previously clusterized instances until
      there are no changes between iterations
  while changes in M do
    for each m_j
      m_jnew = calculatecentroid(C_j)
      if m_jnew == m_j
        changes = false
      else
        changes = true
      end
      m_j = m_jnew
    end

    for each x_n
      B_nj = 1 if argmindist(x_n, m_j) = m_j \foreach m_j else B_nj = 0
    end

    for each B_nj
      if B_nj == 1
        C_j.add(x_i)
      end
    end
  end

  return C = C_1, C_2, ..., C_k
end
```

5. Implementación

5.1. Análisis del conjunto de datos para decidir la conveniencia de la normalización

Debido a las dudas surgidas en torno a la normalización de los atributos y su conveniencia, consideramos adecuado para la tarea tratar de buscar algún método que fuese en alguna manera indicador de la utilidad de la normalización.

En un principio nuestro planteamiento se basaba en utilizar la media de la desviación típica de los valores de cada atributo con el objeto de poder analizar los rangos de las diferencias entre valores de cada atributo. Sin embargo la media por su cuenta no nos es útil, ya que por ejemplo, si la media es alta pero la desviación es baja, en realidad, puede no haber mucha variación en los rangos. Esto nos llevó a hallar el Coeficiente de Variación

$$C_V = \frac{\sigma}{\bar{x}} \tag{1}$$

De esta forma logramos un indicador más preciso sobre el rango que buscamos ya que nos indica la proporción de la variabilidad de las desviaciones, en lugar de la mera cantidad de desviación.

La motivación de este análisis viene dada más por el interés de hallar una forma de conocer la utilidad que pueda tener la normalización en un conjunto de datos, ya que objetivamente, el beneficio principal que puede tener es que si no afecta demasiado al resultado final, nos puede interesar más tener los atributos en su rango numérico inicial (p.ej.: es más visual ver un gráfico con edades entre 0 y 100 que entre 0 y 1).

Por otra parte, hemos tratado de hallar una cifra del Coeficiente de Variación que esté comunmente aceptada como baja, pero no hemos encontrado ninguna fuente fidedigna para ello. Por lo tanto, hemos decidido establecer una cifra pequeña (0,1) teniendo en cuenta que ante la posibilidad de que haya rangos muy variados, puede ser más conveniente normalizar.

Huelga decir que este método y sus resultados son experimentales, pese a tener cierta base empírica carecemos de la certeza sobre su eficacia, siendo nuestro objetivo principal investigar respecto a la normalización en lugar de simplemente aplicarla por estar aceptada como conveniente.

5.2. Problemas encontrados y soluciones adoptadas

Problema con la normalización de los atributos En un comienzo, para normalizar, nuestra intención era aplicar la función Z-Score sobre los atributos de las diferentes instancias. Sin embargo, de esta forma se obtenían resultados fuera del rango de [-1, 1], que es incorrecto para esta normalización. No se consiguió localizar el error, desconociendo si se trataba de una formulación incorrecta o un bug de programación. En lugar de ello se ha realizado una proyección lineal al intervalo [0,1] y de esta forma se ha conseguido unificar los atributos en el mismo intervalo.

Problema de generación de excesivas instancias Una vez estructurado todo el código y con todos los módulos programados, nos encontramos con el problema de que durante la ejecución, se generaban demasiadas instancias. Esto se debía a que en cada iteración del algoritmo, los clusters no eran reseteados correctamente, y cada vez que se añadía una nueva instancia en lugar de eliminar la que estaba presente en su lugar, desplazaba esta última y se insertaba en su posición. Para corregir esto sencillamente se modificó la función de añadir instancias para que esta sobreescribiese la presente, y además, se resetean los clusters en cada iteración.

Instancias repetidas en los gráficos Al igual que en el anterior problema, nos encontramos con que a la hora de graficar, se marcaban más instancias de las que realmente había. La solución también pasó por realizar un reinicio de la matriz de bits en cada iteración del algoritmo.

Ausencia de representación para el cluster 0 Una vez incorporado el código necesario para la generación de gráficos su funcionamiento era correcto, pero no mostraba los datos del cluster 0. El problema radicaba en que a la hora de tratar la matriz de bits de pertenencia, el primer caso se trataba fuera de un bucle y por ello no se hacían las actualizaciones correctas. Una vez corregido esto el gráfico se crea correctamente.

6. Validación del *software*

6.1. Diseño del banco de pruebas

7. Análisis de resultados

7.1. Modificando inicializaciones

7.2. Modificando distancia Minkowski

7.3. Criterios de convergencia

7.3.1. Número fijo de iteraciones

7.3.2. Disimilitud entre *codebooks*

7.4. Distintas métricas

7.4.1. Manhattan

7.4.2. Euclídea

7.4.3. Minkowski

8. Clasificación supervisada respecto de

9. Conclusiones

9.1. Técnicas de clustering: motivación

Tal y como se ha descrito anteriormente en este documento, el propósito de la clasificación no supervisada es, opuestamente a la clasificación supervisada, **descubrir** información, en lugar de predecirla. Los campos y casos a los que es aplicable son múltiples y diversos: patrones de comportamiento en sociología, análisis genético, reconocimiento facial, etcétera. Todo área de conocimiento capaz de recopilar gran cantidad de datos es susceptible y puede beneficiarse de metodologías de clustering.

9.2. Conclusiones generales

Debido a la modularidad del software realizado, nos ha resultado relativamente sencillo encontrar los orígenes de los diversos errores y poder solucionarlos así como añadir varios módulos (p. ej.: generación de gráficos e informe) a lo largo del desarrollo, teniendo la estructura principal del código ya hecha. Además de esto, varias de las clases generadas son genéricas y pueden ser utilizadas en otros proyectos, como las funciones estadísticas de Statistics.java o la evaluación de clusters de Evaluation.java.

Por otra parte, la realización de esta práctica nos ha llevado a conocer en mayor profundidad el funcionamiento de un algoritmo de cierta complejidad, haciendo un uso mínimo de librerías externas. De no haber sido así, no se habría descubierto el algoritmo del coeficiente Silhouette, y aún menos haberlo implementado sin acudir a código ajeno. También cabe destacar la capacidad para manejar diversas que se ha obtenido, así como el uso de varias funciones estadísticas para calcular si es apto normalizar o no (experimental).

9.3. Puntos débiles y propuestas de mejora

Dadas la complejidad y la diversidad de funciones implementadas en el software, ha habido ciertos aspectos en los que no ha sido posible centrarse tanto como cabía. Podría destacarse el rendimiento del mismo, que pese a ser correcto y no tomar demasiado tiempo con volúmenes grandes de datos, no ha sido puesto a revisión

exhaustiva por lo que podría ser mejorado.

Además de esto, la cantidad de métodos de evaluación que se han podido implementar son limitados. Sin embargo no lo consideramos un punto débil destacable, ya que el algoritmo implementado, tras su análisis y diversas pruebas, nos ha parecido consistente y efectivo.

- Conclusiones a la vista de los resultados más relevantes.

10. Valoración subjetiva

1. ¿Has alcanzado los objetivos que se plantean?
2. ¿Te ha resultado de utilidad la tarea planteada?
3. ¿Qué dificultades has encontrado?Valora el grado de dificultad de la tarea.
4. ¿Cuánto tiempo has trabajado en esta tarea? Desglosado:

Coste temporal	
Diseño de software	
Implementación de software	
Tiempo trabajando con Weka	
Búsqueda bibliográfica	
Informe	1

5. Sugerencias para mejorar la tarea. Sugerencias para que se consiga despertar mayor interés y motivación en los alumnos.
6. Críticas(constructivas).

Referencias

[1] Andrew Ng. Machine learning, 2014.