Sistemas de Gestión de Datos y de la Información
Máster en Ingeniería Informática
Práctica 2

Fecha de entrega: viernes 11 de diciembre de 2015, 16:55h

Objetivos mínimos
Implementar los algoritmos del clasificador k-NN y el *clustering k-means*.

Entrega de la práctica
La práctica se entregará en un único fichero **GrupoXX.zip** mediante el Campus Virtual de la asignatura. El fichero contendrá una carpeta por cada uno de los apartados (*k-NN*, *k-means* e *ID3*) conteniendo un fichero con extensión **.py** con las funciones necesarias. Si se ha realizado el apartado **B** del *k-means* entonces dicha carpeta contendrá también el fichero **PDF** solicitado.

Lenguaje de programación
Python 2.7. En el Campus Virtual podéis encontrar una colección de funciones y librerías Python muy útiles para esta práctica.

Ficheros
Todos los ficheros utilizados en esta práctica, **tanto los de entrenamiento como los de test**, comienzan con una línea *cabecera* en formato CSV donde se definen los nombres de los distintos atributos. Después de la primera línea siguen un conjunto de instancias en formato CSV (una por línea) con tantos atributos como la cabecera. En los algoritmos de aprendizaje supervisado siempre supondremos que la clase es el último atributo de cada instancia.

En el Campus Virtual podéis encontrar ficheros de test y entrenamiento para los distintos apartados:

- *k-NN*: `iris.csv` (entrenamiento), `iris_test.csv` (test).
- *k-means*: `customers.csv`.
- *ID3*: dos conjuntos de entrenamiento, `lens.csv` (pequeño y simple) y `car.csv` (más complejo). No se proporcionan conjuntos de test, aunque si los necesitáis podéis extraer de manera aleatoria algunas instancias del conjunto de entrenamiento.

Calificación
El clasificador k-NN y el algoritmo *k-means* tienen un peso del 30 % cada uno. La generación de árboles de clasificación mediante el algoritmo ID3 tiene un

peso del 40 %. Además de la corrección de los programas se valorará también la claridad del código y la organización en funciones auxiliares reutilizables.

Declaración de autoría
Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, la práctica, el grupo, los autores y una declaración de integridad expresando que el documento es fruto exclusivamente del trabajo de sus miembros. No se permite ningún tipo de colaboración entre distintos grupos.

Clasificador k-NN

A) [Objetivo mínimo, 20 %]

Implementar un clasificador k-NN en Python. Se deben definir como mínimo las siguientes 3 funciones:

1. `read_file (filename)`
   Lee el fichero de texto de ruta `filename` formado por instancias en formato CSV y devuelve una lista de instancias representadas cada una como una lista. Supondremos que todos los atributos son continuos salvo la clase (último atributo) que es categórico. El resultado de invocar a la función `read_file` sería una lista como:

```
[ [1.8, 3.9, 4.8, 'good'],
  [2.0, 5.9, 4.7, 'bad'],
  [5.0, 1.2, 2.8, 'excellent'] ]
```

2. `knn(k, i, c)`
   Dado un valor de `k`, una instancia nueva `i` y un conjunto de entrenamiento `c`, devuelve la clase predicha para `i` utilizando únicamente la moda de la clase de los `k` vecinos más cercanos. No es necesario aplicar normalización a los atributos del conjunto de entrenamiento. La instancia `i` tendrá tantos atributos como las instancias del conjunto de entrenamiento `c`, pudiendo tener o no una clase asociada.

3. `test(k, trainset, testset)`
   Dado un valor de `k`, un conjunto de entrenamiento `trainset` y un conjunto de test `testset` calcula la proporción [0..1] de instancias de `testset` correctamente clasificadas utilizando la función `knn`.

B) [10 %]

Implementar el algoritmo k-NN como tarea MapReduce en *mrjob* para clasificar un conjunto de instancias. El fichero de entrada de la tarea será el conjunto de entrenamiento, mientras que las instancias a clasificar estarán en un fichero separado pero cuya ruta es conocida de antemano. Además queremos minimizar el número de parejas que han de transferirse entre la fase Map y la fase Reduce.

***Pista**: podéis utilizar variables estáticas de clase para almacenar valores que necesitéis en la fase Map o Reduce. Echad un ojo a la funciones* `mapper_init()` *y* `reducer_init ()` *explicadas en* `https://pythonhosted.org/mrjob/job.html` *por si os son de utilidad.*

Clustering k-means

A) [Objetivo mínimo, 20 %]

Implementar el algoritmo de *clustering k-means* en Python. Se debe definir como mínimo las siguientes dos funciones:

1. `read_file (filename)`
   Similar a la función utilizada en el apartado anterior salvo que ahora no existe atributo categórico clase. Por lo tanto tras la cabecera todas las instancias tienen el mismo número de atributos. **Todos los atributos serán continuos**.

2. `kmeans(k, instancias , centroides_ini =None)`
   Recibe un valor de `k`, una lista de `instancias` (con el mismo formato que para *k-NN*) y una lista de `k` centroides iniciales (`centroides_ini`); y devuelve una agrupación aplicando el algoritmo de *k-means*. Concretamente el resultado es una pareja (`clustering` , `centroides`) tal que:

   - `clustering` es un diccionario representando el agrupamiento. Por ejemplo para `k=3` y 5 instancias:

   ```
   { 0: [inst1, inst2, inst3],
     1: [inst4],
     2: [inst5] }
   ```

   - `centroides` es la lista con los centroides de los `k` clústeres finales. `centroide[i]` será el centroide del clúster numero `i`.

   Si `centroides_ini` está vacío (su valor es `None`) entonces se escogerán los centroides iniciales intentando que estén lo más alejados posible entre sí. El algoritmo será:

   a) Se añade `instancias[0]` como primer centroide.
   b) Elijo como siguiente centroide aquella instancia cuya *minima distancia a los demás centroides es la máxima.*
   c) Termino cuando tengo `k` centroides.

B) [10 %]

Implementar las 3 medidas de cohesión (radio, diámetro y distancia al cuadrado promedio con respecto el centroide) para evaluar la calidad del *clustering* para valores de `k` entre 2 y 20. Visualizar la variación de la cohesión como un gráfico de líneas para cada una de las medidas y razonar cuál podría ser un buen valor de `k`.

Aparte del código, para este apartado hay que entregar un **fichero PDF** incluyendo las 3 gráficas, el valor escogido de `k` y su razonamiento.

Árbol de clasificación ID3

A) [25 %]

Implementar el método ID3 (TDIDT con ganancia de información) para generación de árboles de clasificación. Supondremos que las instancias tienen **todos** los atributos categóricos, los ficheros siguen el formato CSV y que todos los valores de los atributos aparecen en el conjunto de entrenamiento. Para la generación del árbol se debe usar la ganancia de información (o disminución de entropía) como criterio de selección de atributos.

En este apartado se debe crear como mínimo las siguientes funciones:

1. read_file (filename)
   Lee el fichero de texto de ruta filename formado por instancias en formato CSV. **Todos los atributos serán categóricos, incluida la clase** que aparece en último lugar. Recordad que la primera línea es la cabecera que define los nombres de cada atributo. Como resultado esta función devuelve una tupla de 3 elementos (inst , attrib_dic , classes):

   a) inst Lista de instancias representadas como listas.

   b) attrib_dic Diccionario con los valores y posición de cada atributo dentro de la instancia. Ejemplo:

   ```
   { 'age':      (0, ['pre_presbyopic', 'presbyopic', 'young']),
     'astig':    (2, ['no', 'yes']),
     'specRx':   (1, ['high_hypermetropia', 'myopia']),
     'tears':    (3, ['normal', 'reduced']) }
   ```

   c) classes Lista con los nombres de las posibles clases. Ejemplo:

   ```
   ['hard', 'soft', 'no']
   ```

2. id3(inst , attrib_dic , classes , candidates) Toma como parámetros una lista de instancias inst, un diccionario attrib_dic que almacena los posibles valores para cada atributo, una lista classes de posibles clases y una lista de atributos candidatos para ramificar candidates. Esta lista candidates es una lista con *nombres* de atributos tal y como aparecen en attrib_dic . El resultado de esta función será un árbol de clasificación. Podéis representar este árbol como prefiráis, aunque se podría construir de manera sencilla utilizando únicamente tuplas o combinando tuplas y diccionarios.

B) [15 %]

Almacenar los árboles de clasificación generados por la función id3 en un fichero en formato *DOT*. DOT es un formato textual muy sencillo para la descripción de grafos, que luego pueden visualizarse con herramientas como `xdot`. Podéis encontrar más información sobre este lenguaje en:

- `http://www.graphviz.org/Documentation.php`
- `https://en.wikipedia.org/wiki/DOT_language`

```
digraph tree {
  //nodos
  sexo [label="Sexo",shape="box"];
  f1 [label="Futbol"];
  exp [label="Expediente",shape="box"];
  f2 [label="Futbol"];
  p [label="Padel"];

  //aristas
  sexo -> f1 [label="M"];
  sexo -> exp [label="F"];
  exp -> f2 [label="<A"];
  exp -> p [label=">=A"];
}
```

Figura 1: Fichero `tree.dot` representando un árbol de clasificación simple

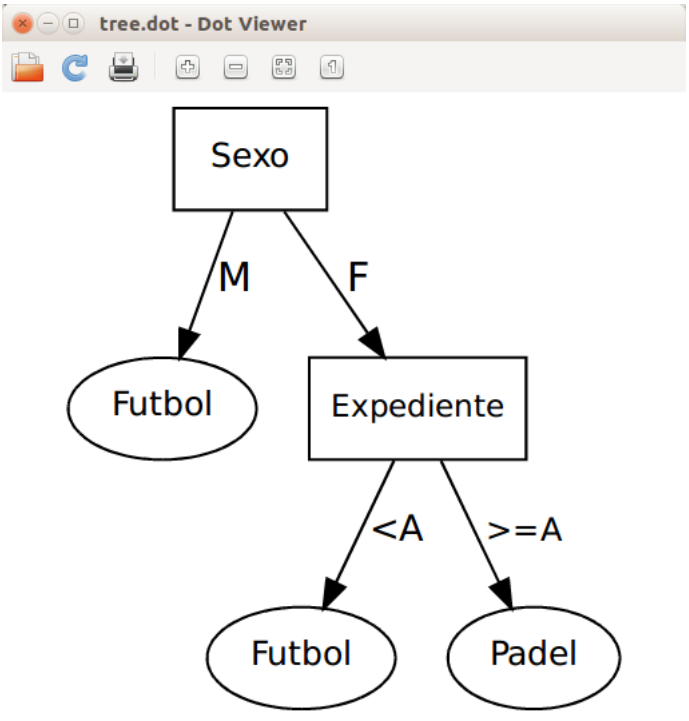

Figura 2: Fichero `tree.dot` visualizado en `xdot`

Un ejemplo muy simple que muestra todos los elementos del lenguaje DOT que necesitáis aparece en la Figura 1, y su visualización en el programa `xdot` en la Figura 2.

Para este apartado se pide implementar la siguiente función:

- write_dot_tree (id3_tree , filename)
  Esta función recibe un árbol de clasificación id3_tree tal y como lo genera la función id3 del apartado A y una ruta filename. Esta función no devuelve nada sino que convierte id3_tree a formato DOT y lo almacena en filename.