Data Mining Project PT. 2 - k-means and DBSCAN analysis and implementation

Daniel Hanspeter, 6129 Daniel Graziotin, 4801 Thomas Schievenin, 5701

December 27, 2010

Abstract

This report is on the development and analysis of two Data Mining clustering algorithms, namely k-means clustering and DBSCAN, within an interactive graphical program in Java programming language. This report is dived in three parts. In the first part we will present a usage section, explaining how to interact with the program. We will then introduce the architecture of the system, explaining our implementation in detail. In the third part we will present our analysis of the algorithms

1 Getting Started

1.1 Introduction

The DWDM Project part II consists in implementing some Data Mining algorithms to get a deeper understanding of their behavior, their strengths and their weaknesses. We choosed to implement the K-Means and the DB-Scan algorithms performing the same task and then we compared their behaviours.

1.2 Running the application for the first time

First, launch the JAR file. This can be done either by double clicking on the JAR file or by executing the below command in a console window:

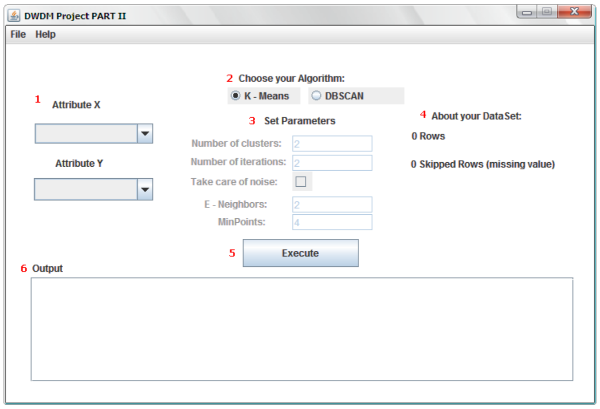
```
java -jar dmproject.jar
```

Our application is tested and supported for Windows XP and Windows 7. There are litte unexpected behaviors when running it under Mac Os X but they are all related to the graphical implementation of the project: the functionalities seem working fine anyway. There are bigger issues with Gnu/Linux, all related to the graphical implementation: some functionalities will not run.

1.3 Quick overview

As the application starts all features are disabled. This is because there is not a dataset loaded yet. The application is composed by the following parts:

1. Select Box Attributes: the user gets the possibility to select the attributes on which the algoritms will perform their calculations;
2. Preferred Algorithm: the user decides which algorithm to perform;
3. Parameters: in case of K-Means algorithm selected, the available parameters to set are the number of clusters, the number of iterations and whether to take care of noise. If the DB-Scan algorithm is selected, we have to specify only the values for E-Neighbours and Min-Points;
4. Dataset information: the program displays the number of rows loaded into memory and the number of skipped rows because of missing values;
5. Execute: this bottom executes the selected algorithm;
6. Output: algorithms' outcome.



● K - Means ○ DBSCAN

Number of clusters: 2
Number of iterations: 2
Take care of noise: □
E - Neighbors: 2
MinPoints: 4

0 Rows

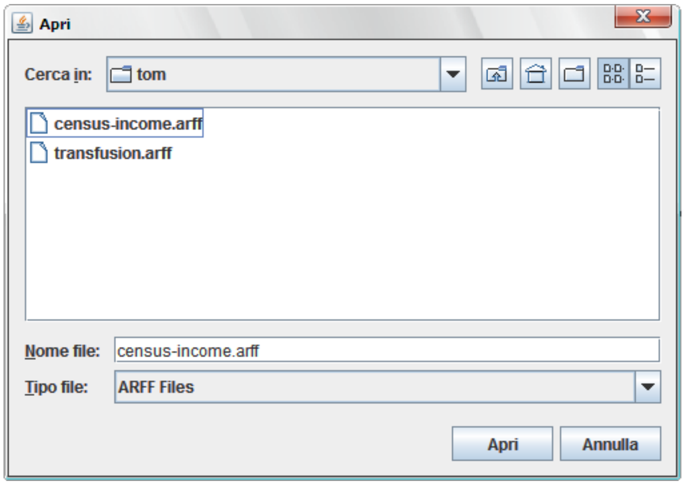
0 Skipped Rows (missing value)

Figure 1.1: Quick overview

1.4 How to run K-Means algorithm

The steps for running the K-Means algorithm are the following:

- First we need to load our dataset into the memory. This can be done by clicking on menu File > Open, then select either "census-income.arff" or "blood-transfusion".arff provided as attachment. Note, the loading windows allows to import only arff file, otherwise our parser is not able to parse its content.



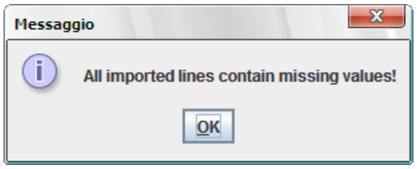
Cerca in: tom

census-income.arff
transfusion.arff

Nome file: census-income.arff
Tipo file: ARFF Files

Figure 1.2: Open window

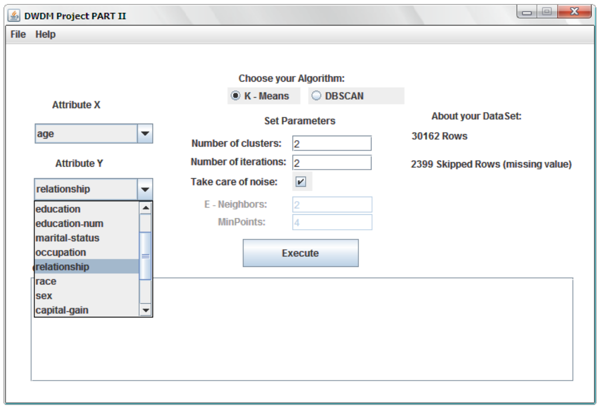
- Important: while importing the dataset, the application checks wheter missing values are present. This information will be diplayed later in the "DataSet information" section described before. In case of a dataset with missing values in every row, the application informs the user that it has no data for running the algorithms.



All imported lines contain missing values!

Figure 1.3: No data popup

- Once the dataset has been loaded, we can decide on which attributes to perform the K-Means algorithm. We selected as attributes "age" and "relationship", 2 clusters and 2 iterations. We also wanted to take care of the noise.



Choose your Algorithm: ● K - Means ○ DBSCAN

Attribute X: age

Attribute Y: relationship

education
education-num
marital-status
occupation
relationship
race
sex
capital-gain

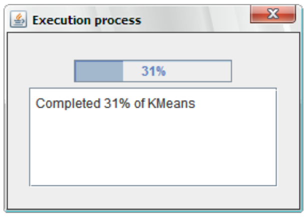
Set Parameters

Number of clusters: 2
Number of iterations: 2
Take care of noise: ☑
E - Neighbors: 2
MinPoints: 4

About your DataSet:
30162 Rows
2399 Skipped Rows (missing value)

Figure 1.4: Selecting attributes

- Click the Execute button;
- In case of big dataset size, the execution of the K-Means algorithm takes a while. For this reason we implemented a progress bar to inform the user how long the execution is going to take.

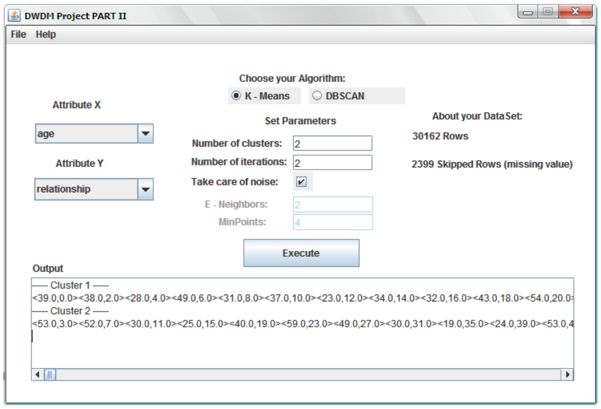


31%

Completed 31% of KMeans

Figure 1.5: Progress bar

- Once the progress bar finishes, the user sees the outcome of the K-Means algorighm:



Choose your Algorithm: ● K - Means ○ DBSCAN

Attribute X: age

Attribute Y: relationship

Set Parameters

Number of clusters: 2
Number of iterations: 2
Take care of noise: ☑
E - Neighbors: 2
MinPoints: 4

About your DataSet:
30162 Rows
2399 Skipped Rows (missing value)

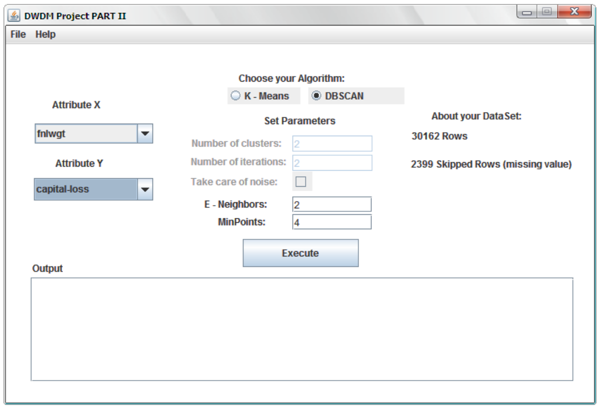
Output

```
----- Cluster 1 -----
<39.0,0.0><38.0,2.0><28.0,4.0><49.0,6.0><31.0,8.0><37.0,10.0><23.0,12.0><34.0,14.0><32.0,16.0><43.0,18.0><54.0,20.0>
----- Cluster 2 -----
<53.0,3.0><52.0,7.0><30.0,11.0><25.0,15.0><40.0,19.0><59.0,23.0><49.0,27.0><30.0,31.0><19.0,35.0><24.0,39.0><53.0,4
```

Figure 1.6: K-Means outcome

1.5 How to run DB-Scan algorigthm

The steps for executing the DB-Scan algorithm are basically the same of the K-Means algorithm. The only things that changen are the parameters to set. In fact, we have to specify the number of E-Neighbors and the number of MinPoints.



DWDM Project PART II

File Help

Choose your Algorithm:

○ K - Means ● DBSCAN

Attribute X

fnlwgt

Attribute Y

capital-loss

Set Parameters

Number of clusters: 2

Number of iterations: 2

Take care of noise: □

E - Neighbors: 2

MinPoints: 4

Execute

About your DataSet:

30162 Rows

2399 Skipped Rows (missing value)

Output

Figure 1.7: DB-Scan Algorithm

2 Architecture

For implementing the system we decided to follow a personalized Model-View-Controller design pattern, that we name "MVA", or Model-View-Algorithm, respectively mapped in the Java packages **types**, **gui** and **algorithms.** This is for a better separation of concerns. Figure 2.1 illustrates with a very simple scheme our general view of the architecture of the system. It is a personalized version of the MVC pattern because we further separated the algorithms from the controller part, putting the non algorithm-related code in the gui package anyway. The user interacts only with the gui component that, based on user input, calls the components of the algorithms package that will do some computation on data defined in the types package and return the result.

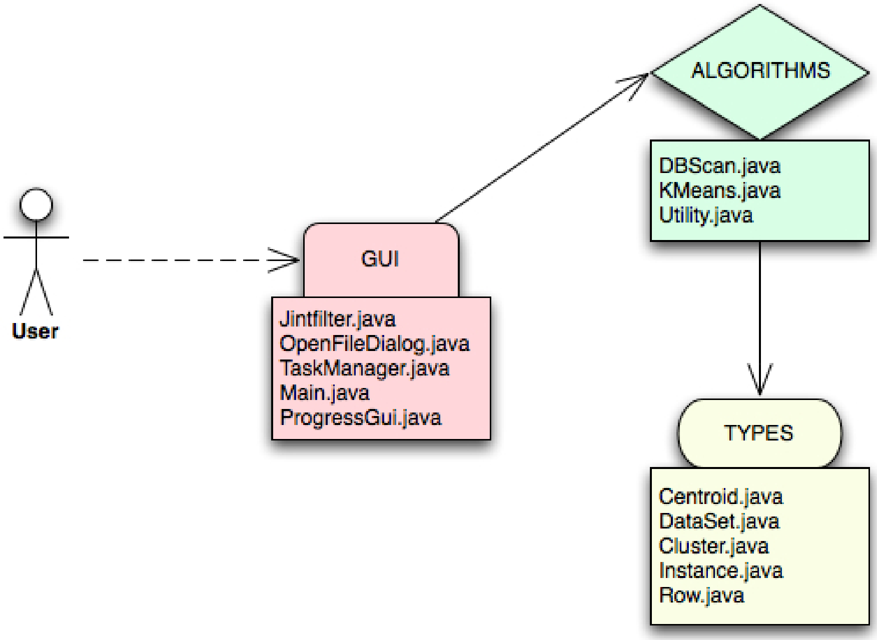


Figure 2.1: General Architecture View of the System

The gui package contains the Main class that is responsible for accessing the program and constructing the graphical user interface using Swing. The Jintfilter is a specialization of Swing's Document object and is used for filtering user input on algorithm parameters. The OpenFileDialog class is a personalized dialog for opening the datasets, in ARFF format. The ProgressGui class is responsible for creating a window containing the status and progress of the operations. The TaskManager class is a wrapper for creating threads that contain the algorithm tasks.

The algorithms package, better illustrated in Figure 2.1, contains the pure, type independant implementations of the clustering algorithms, k-means and DBSCAN. They make use of generalized data types defined in the types package, in order to have an abstraction of the data that could be loaded from any datasource, being it an ARFF file, a CSV, and XML or a Database. The Utility class actually contains the method for loading the datasets and converting them in objects declared in types package.

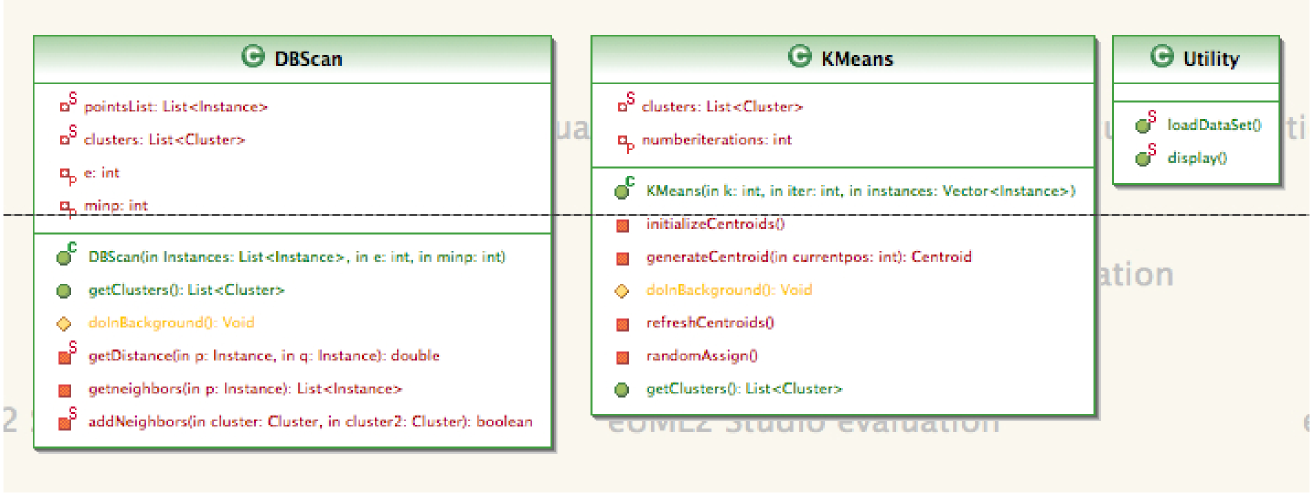

DBScan
pointsList: List<Instance>
clusters: List<Cluster>
e: int
minp: int
DBScan(in Instances: List<Instance>, in e: int, in minp: int)
getClusters(): List<Cluster>
doInBackground(): Void
getDistance(in p: Instance, in q: Instance): double
getneighbors(in p: Instance): List<Instance>
addNeighbors(in cluster: Cluster, in cluster2: Cluster): boolean
KMeans
clusters: List<Cluster>
numberiterations: int
KMeans(in k: int, in iter: int, in instances: Vector<Instance>)
initializeCentroids()
generateCentroid(in currentpos: int): Centroid
doInBackground(): Void
refreshCentroids()
randomAssign()
getClusters(): List<Cluster>
Utility
loadDataSet()
display()

Figure 2.2: Algorithms Package

The types package, represented in Figure 2.3, contains pure Java representation of the data that can be loaded by the algorithms. They are an abstract representation of an ARFF file but they are completely independent from their structure. They act as wrappers for the algorithm and it would be a matter of changing the Utility.loadDataSet() method for loading any kind of data format. The DataSet class represents a data format, e.g. a dataset, that is composed by rows, defined in Row class. The other classes are related to the algorithm own data types, for example Cluster and Centroid.

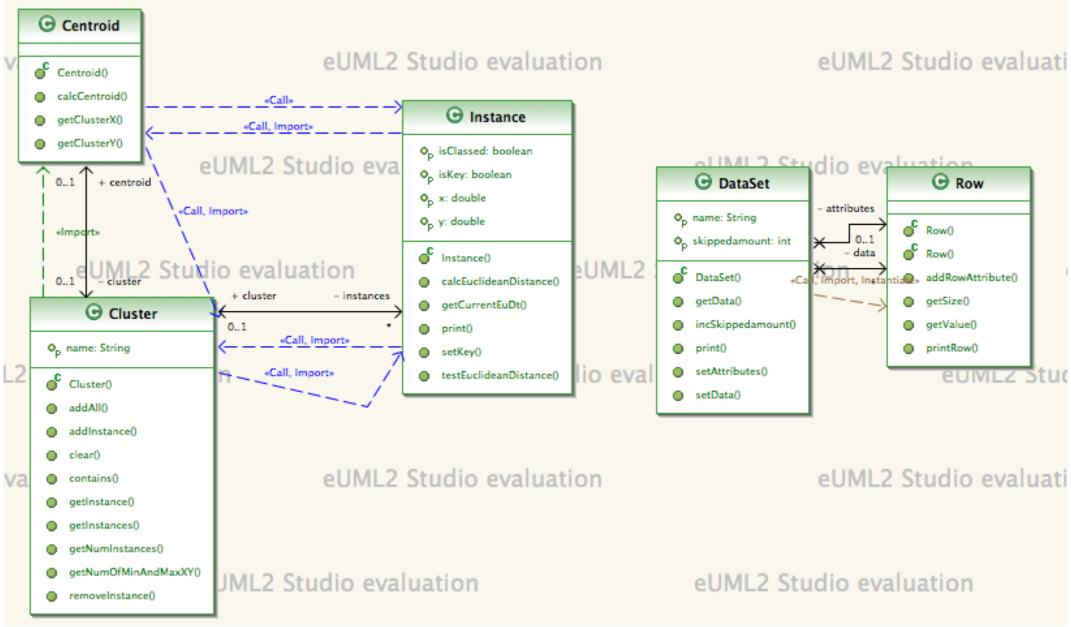


Figure 2.3: Types Package

All the code has been clearly documented using Javadoc documentation. For a deep understanding of the functioning of each method, please see the attached archive file containing the HTML exported documentation.

Regarding the additional method we created in order to achieve the results, they are all documented in the code and in the documentation, but we will mention here two of theme. The first one is a method for removing noise from the datasets. It begins at line 143 of it.unibz.algorithms.KMeans. It is inspired by Barca, J.C. Rumantir, G. - A Modified K-means Algorithm for Noise Reduction in Optical Motion Capture Data.

In order to take care of noise, we don't reduce it before executing the algorithm, but we added an additional check during the execution. On every end of iteration, we try to find out how compact a cluster is. If instances inside a cluster overcome the average compactness of the clusters in the dataset, this cluster is flagged as noise. In modification to the classical K-means algorithm, we try to add an automatic determination of the optimum number of clusters we should have. A cluster is considered noise if it only has a few instances in it. The minimum number of instances in a cluster, or the cluster size, are set such that it minimizes the degree of false positives (i.e. data clusters incorrectly classified as noise) and false negatives (i.e. noise clusters incorrectly classified as data). The following is a code snippet that handles the problem:

```
// Find corners of compactness based on instances with
// minimum and maximum X and Y values
compactness = clusters.get(j).getNumInstances() / (clusters.get(j).getNumOfMinAndMaxXY());
// Calculate cluster compactness, that is number of data points / compactness window size
if (compactness > instances.size() / clusters.size())
        clusters.get(j).setIsnoise(true);
else
        clusters.get(j).setIsnoise(false);
```

3 Analysis

3.1 Comparison of the algorithms

Two clustering algorithms - namely the k-means and the DBSCAN - were examined and their clustering performances were compared. First of all, in the DBSCAN algorithm the user does not have to specify the K value, which drastically affects the outcome in the K-means algorithm. The DBSCAN deals with noise data and outliers. On the other hand, the K-Means algorithm is affected by noisy data. Additionally, in the DBSCAN the ordering of the points in the database is not so important. We found that the DBSCAN algorithm has a superior performance.

3.2 An algorithms analysis

3.2.1 K-Means Properties

- Pro: k-means is relatively scalable and efficient in processing large data sets
- Pro: the computation complexity of the algorithm is O(objects*clusters*iterations)
- Con: it can be applied only when the mean of a cluster is defined
- Con: the user must specify a value for K parameter
- Con: it is not suitable for discovering clusters with nonconvex shapes or clusters of very different size
- Con: it is sensitive to noise and outlier data points (can influence the mean value).

3.2.2 DB-Scan Properties

- Pro: it does not force you to choose the number of clusters in the data a priori, as opposed to k-means
- Pro: it can find arbitrarily shaped clusters. It finds clusters completely surrounded by a different cluster
- Pro: it considers noisy values
- Pro: it requires just two parameters for working
- Pro: it is mostly insensitive to the ordering of the points in the database

- Con: the distance measure in the function for finding neighbours is crucial. We used the euclidean distance measure and especially for high-dimensional data, this distance metric can be rendered almost useless
- Con: varying densities affects the outcome of this algorithm (hierarchical data sets)