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

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

December 24, 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 Usage

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.

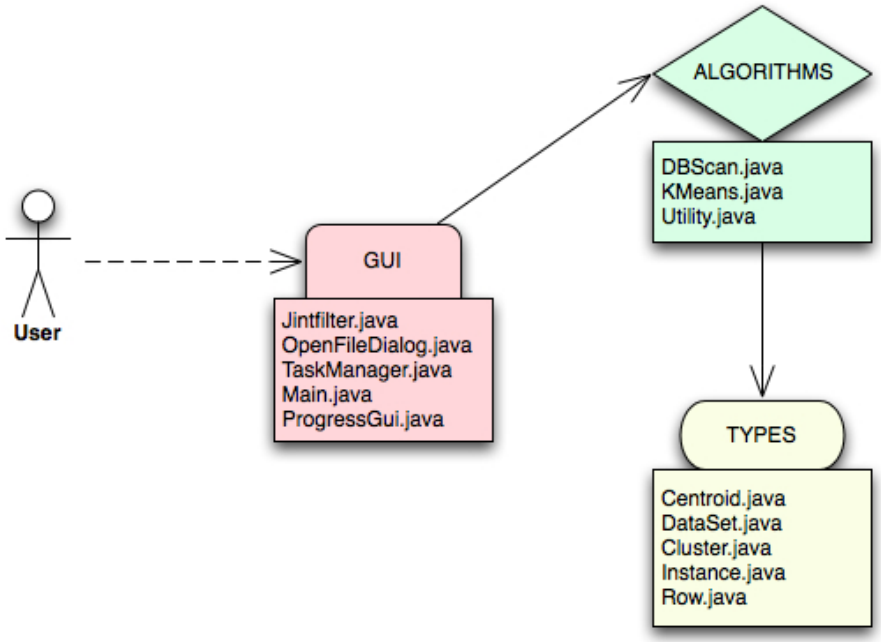

Figure 2.1: General Architecture View of the System

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.

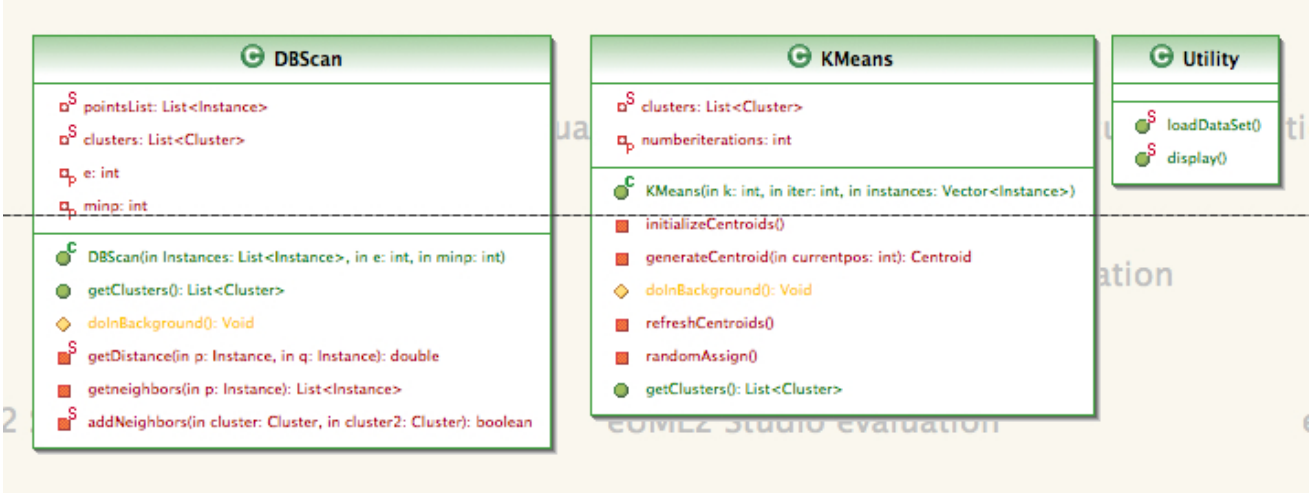

Figure 2.2: Algorithms Package

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.

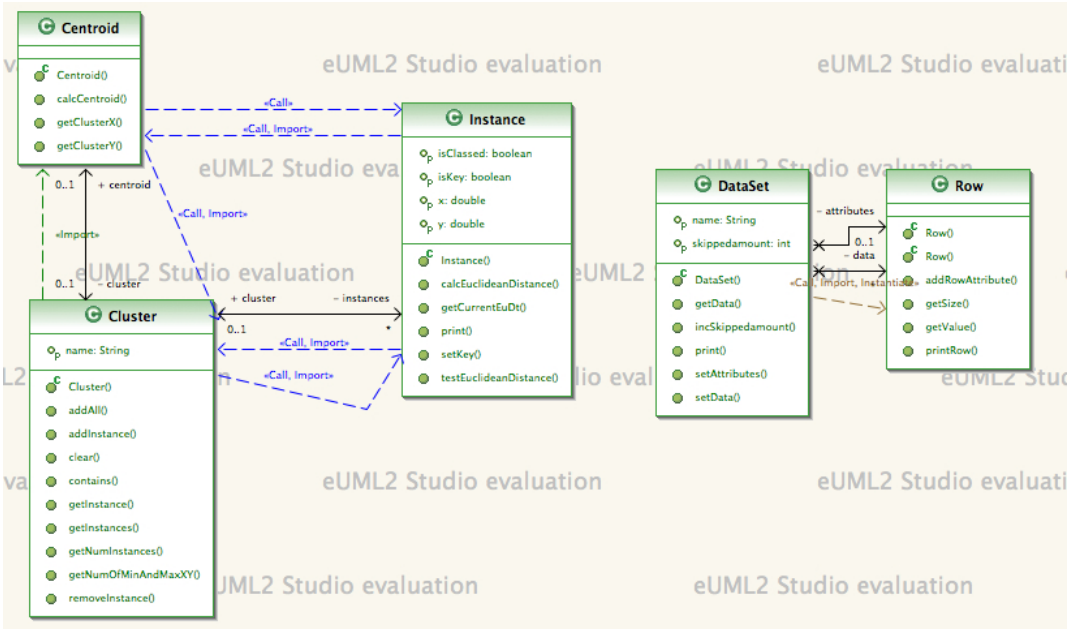

Figure 2.3: Types Package

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