Faculdade de Economia da Universidade do Porto

Master in Data Analytics 2017/2018

Data Bases and Programming

Rui Manuel Santos Rodrigues Leite

K-means Clustering Algorithm

Rúben Filipe (201702570)

Simon Sudora (201701827)

January 12, 2018

Contents

Method Description

Clustering

K-means is an algorithm for solving clustering problems. When we are trying to solve clustering problem, we are trying to find some common characteristics among examples and based on these characteristics categorize them into clusters. Clustering belongs to the category of unsupervised learning problems. It means that we don't know the true class of the examples, but we try to find some. However, we can't evaluate created model precisely based on the data, but instead we have to use our common sense to evaluate how bad or good is created clustering. On the Fig. 1 we can see very simple example of how can be clusters created. Unfortunately, real life examples are usually much more complex

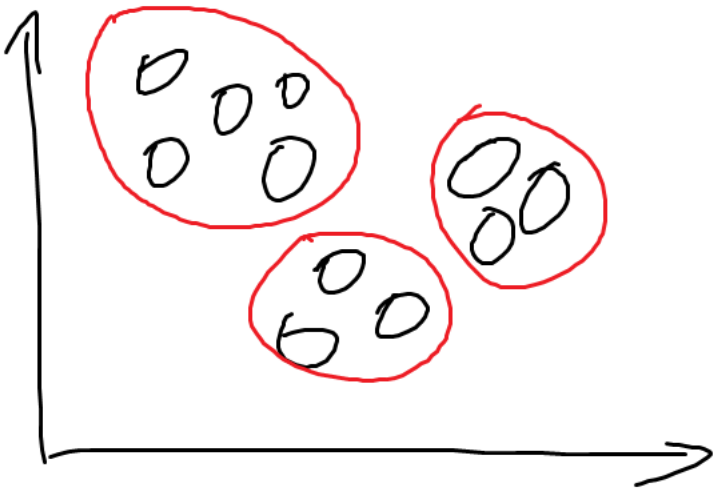

Fig. 1 - Example of possible clusters for the given data points in 2-dimensional space.

K-means

There are more algorithms for solving clustering problems. What is specific for the K-means algorithm, is that it requires k (number of clusters) to have predefined. This might be difficult for user to determine. It also doesn't provide any explanation why it did clustering the way it did. On the other hand, the main advantage of K-means before the others is its **speed**.

The **main idea** of the K-means algorithm:

1. Choose randomly k-instances, which will be your initial centroids. Each cluster has its centroid.
2. For each instance find the nearest cluster (centroid) and assign instance as member of the cluster.
3. Based on the changes in the step 2. update centroids of each cluster by computing the mean of the instances in the cluster.
4. If you have not made any changes in the step 2., then you finished otherwise go to 2. step.

As a distance function is usually used **Euclidean distance**. Standard K-means can deal only with numeric variables. This comes from the fact that Euclidean distance function can measure properly only distance between pair of numerical vectors. To have Euclidean distance computed properly, we also need to have variables in the data set in the same scale. Min max normalization can help in this case. Moreover, Euclidean distance can't be computed between vectors with missing values.

By doing this algorithm we are trying to minimize sum of the distance between every instance and the centroid of its cluster. The K-means algorithm guarantees to converge into a **local optimum**.

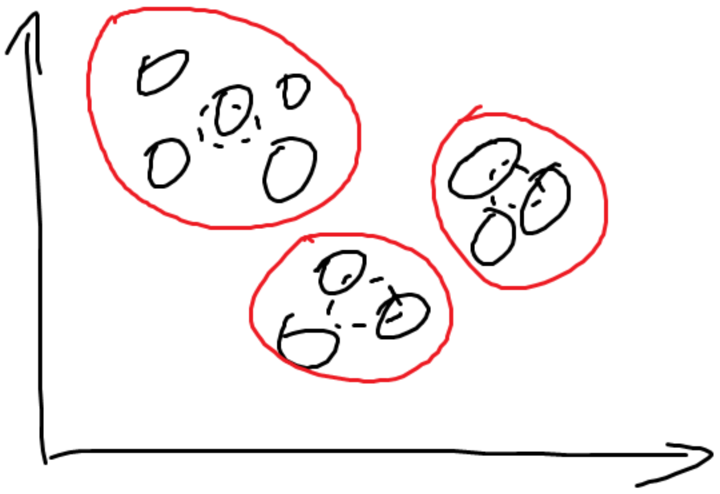

Fig. 2 - Dashed circles represent centroids of clusters.

Implementation

We did implementation of K-means by using programming language R. We implemented K-means as it was described earlier.

Data structures

Implementation is done by using 2 main data structures:

- *data.set* (data frame) – This data frame stores the input dataset. Moreover, this data structure is also used to hold the id of the cluster that each instance is assigned to. This value is stored in the column "cluster".
- *centroids* (data frame) – This data frame stores centroid instances. It has always k rows (one row for each cluster) and It has the same structure as data.set – all the columns from the original data set + cluster id.

Functions

The code is divided into 2 files:

- *k_means.R* - In this file is implementation of the K-means algorithm and it includes following functions:
    - *choose.init.centroid.instances (data.set, k, seed)* – This function takes dataset from the input and chooses randomly (uniform distribution) k instances to be initial centroids of clusters. It creates new column - cluster in the dataset. This variable determines the cluster of instance by assigning the id of the cluster to the instance. Only centroids have the true id of cluster, the other instances have neutral value (-1) because they have not been assigned to any cluster yet.
    - *min.max.normalization(x)* – This function does the min max scaling of the vector x.
    - *euclidean.dist(x1, x2)* – This function computes and returns the Euclidean distance of the vectors x1 and x2.
    - *find.nearest.cluster(instance, centroids)* – This function finds and returns cluster id of the cluster whose centroid has the smallest Euclidean distance from instance on the input.
    - *assign.instancese.to.clusters(data.set, centroids)* – This function goes through all the instances in the dataset assign the id of the nearest cluster to each of them.

    - *compute.new.centroids(data.set, centroids)* – This function goes through all the cluster and computes new centroids for them. Centroids are computed by doing mean of each column in the dataset but only from the instances that are assigned to the particular cluster.
    - *k.means(data.set, k, max.iter, seed)* – This is the main function of the implementation. To do the K-means clustering user should call this function. As parameters of this function user should put input dataset (only with numerical values, because Euclidean function can't compute distance properly for the nominal values), number of clusters that should be created, maximum number of iteration in which K-means must converge and random seed that will be used for choosing initial centroids.

      The main loop of the function performs K-means clustering as it was described in the paragraph "The main idea of the K-means algorithm". Besides this there is also one more loop. The role of this loop is to try different seed for choosing initial centroids in case the K-means did not converge in deserved number of iterations. Moreover, this function also checks if all the values in the input dataset are numerical and if there are no missing values. During the iterations of K-means it prints out info massages for user about the current iteration number and number of changes that were made in the iteration.

- *tests.R* - Code in this file serve for doing tests with implemented K-means algorithm. It also includes 2 functions for computing the performance and accuracy of the implemented K-means:
    - *find.cluster.true.class.binding (data.set)* – For each true class, it finds the most frequent cluster id. The most frequent cluster id is then changed to corresponding true class e.g. in iris dataset there are 3 classes (setosa, versicolor, virginica). This function finds cluster id that correspond to setose, versicolor and virginica.
    - *compute.accuracy(data.set)* – This function simply computes and returns the percentage of cases in which K-means algorithm determined the cluster correctly.

Experiments

Method

As we know clustering is unsupervised learning problem, thus it's not easy to evaluate how good and accurate algorithm is. However, one way how to evaluate the results of clustering is to choose dataset that is determined for classification and compare true classes from the original dataset and clusters created by the K-means(or another clustering method). To do the described evaluation of K-means we decided to use well known iris dataset. We also needed some referral value to compare accuracy of our implementation with. For this purpose, we decide to use existing implementation of the K-means from R library stats.

Results

K-means from stats with the following parameters iter.max = 100, nstart = 5, algorithm = "Hartigan-Wong" achieved the accuracy 89,3%. This was our referral value for a good accuracy.

During the experiments with our implementation we encountered a problem related to a random seed. When we chose a "bad" seed, K-means got stuck in local optimum and his final accuracy was very bad.

Example: seed = 100, accuracy: 22%

But when we chose a “good” seed, our implementation of K-means had almost the same accuracy as the implementation from the stats library.

Example: seed = 5, accuracy: 88,7%

We also noticed that K-means converge always quite fast (5-15 iterations) regardless of chosen seed.

Conclusion

As can be seen on experiments, our implementation performs quite well on the iris data set and it can compete with the built in R implementation of K-means algorithm. The only difficulties can cause “badly” chosen seed, thus it’s always necessary to try more than one seed.

Possible future work and improvements

The problem related to the random seed can be solved by using a heuristic for choosing the initial centroids instead of choosing them randomly.

It will be also good to test our implementation on more datasets (bigger and more complex) and try to compare our implementation with implementation from stats library not only in terms of accuracy but also in terms of speed.

Sources

[1] João Gama (2017). Data mining I: Cluster analysis [Powerpoint slides].