Introduction to Data Science Project Report

Jenifer Tabita Ciuciu-Kiss

Eötvös Loránd University, Budapest
Budapest, Egyetem tér 1-3, 1053, Hungary
kuefmz@inf.elte.hu
https://www.elte.hu/

Abstract. This review contains the summary about the final Introduction to Data Science project. This consists of three main parts: modelling, clusterization and frequent pattern matching. In order to do that some prepocessing was needed. The review explains how all that was done by me and what conclusions I made from the results.

Keywords: clusterization · frequent pattern matching · modelling.

1 Preprocessing

1.1 Analization of the given database

Firstly I checked the data manually. As a description of the database was given it was easy to find out what attributes appear and what their meaning is. We have different data about bank customers like the status of their existing checking account, credit amount, present residence, age, housing situation, etc. The last attribute defines whether a customer is a good customer or a bad customer. That is important because later on we would like to define the correlations between the last and other attributes.

Reading the data from the given csv file (project_data.csv) I have recognized that two attributes names are missing, the *X19* and *X20*. To solve this problem, I opened the *.csv* file and wrote the missing part manually.

After reading the data from the given database to the *data* variable, I have checked whether it contains missing values. For this I have used the *data.info()* command which also shows the type of the attributes and the data-frame's shape. Now we know that the data-frame has *1000* rows (data from 1000 customers) and *21* attributes. The type of the data is either *int64* or *object*. For every property we know exactly what information it contains, which makes the exploratory data analysis easier.

For checking the beetling data I made some histograms and I did not see any unexpected values, so I concluded that there are no beetling values in the database. As in the last row (named Y) we have 1-2 values, I subtracted 1 from them to get 0-1 values and to handle the data as a Boolean.

1.2 Exploratory data analysis

From now on, I am going to mark in the figures the *good* customers with *blue* and the *bad* customers with *red*. I made many analyses on the data, but I am going to present only the ones that can help in making some conclusions.

Knowing that the database has no missing values, we can make some plots for a better understanding of the data [1]. First of all we would like to know the number of the good and bad customers. It is very easy to find out that there are 700 good and 300 bad customers, which means that 70% of the customers are good and 30% of them are bad. I have counted the number of good and bad customers and I have printed it out.

```
The number of good customers: 700
The number of bad customers: 300
```

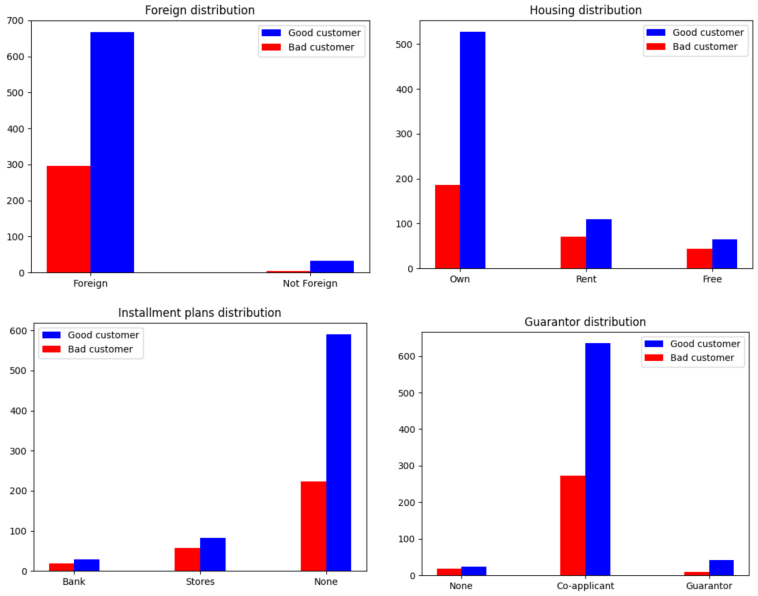

Fig. 1. Good and bad customer distribution, and their Housing, Installment Plans and Guarantors distribution

In figure 6. the upper left plot shows the foreignness distribution between good and bad customers. We can see that most of the customers are foreign so I suppose there is no correlation being foreign and being a good customer.

I have checked whether the housing situation of the customer is in a relation with being a good or bad customer. I can see that in Figure 1. on the upper right plot that most of the good customers own a house. That means that owning a house and being a good customer have a high correlation.

We can see correlation between being a good customer and having no other installment plans in Figure 1. on the bottom left plot. This could possibly mean that the customers who do not have other plans take more care on their debt.

Figure 1. show in the bottom right plot that most of the good customers have a co-applicant an for the credit. This also let us conclude the correlation between being a good customer and having a co-applicant for the credit.

The reason why I have chosen the plots seen in Figure 1 is that I have seen some protrusion counting the different properties for good and bad customers. This means that there are correlation between these attributes.

Next I have checked the age distribution with being a good or bad customer. I have used the age later on for clustering as this is one of the few continuous data in the database. Figure 2. shows the results and we can see that there is no correlation between these attributes. The conclusion from the histogram that age is not strongly connected with being good or bad customer.

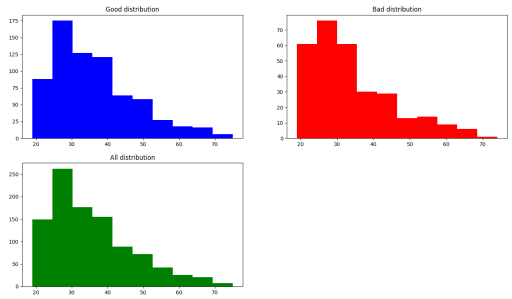

Fig. 2. Age distribution of good and bad customers

2 Baseline results and evaluate models

During the database analysis I have found out that the *FalsePositive* results (marking a customer as good while he/she is actually a bad customer) is five times worse then the *FalseNegative* results (marking them bad while they are good). For this let me introduce some concepts [3]:

```
Recall = TruePositive / (TruePositive + FalseNegative)
Precision = TruePositive / (TruePositive + FalsePositive)
```

In order to handle this, I used the *sklearn.metrics.fbeta_score* [2] function. This function calculates the harmonic mean between precision and recall, and it favors higher recall scores over precision scores.

Another thing that needed to be handled was dividing the data into train and test parts. For that I used the *sklearn.model_selection.RepeatedStratifiedKFold* [2]

function to 10 folds and I repeated it 3 times. It means that a single model will be evaluated 30 times and the mean and standard deviation of these runs will be reported.

I have tried many different models. I am going to show the results that I got and I will also present two of the models in detail, namely the *Linear Regression* and the *SMOTE* algorithms.

First of all, for models I needed to encode the non- number attributes, and I used *pd.get_dummies* algorithm for this.

During the evaluation I plotted the result and I also printed some numbers that I got. As I said before, every algorithm was ran 30 times. The model each time was trained on the 90% of the data and was tested on the remaining 10%. During the testing the accuracy was checked and calculated with the *fbeta_score* measure mentioned before. So there were 30 result scores for each model and the first printed number shows their mean. The second number between parenthesis shows the standard deviation of the model. It is a measure of spread of the scores, and it tells us the concentration of data around the mean value. In case of low values the concentration is high around the mean value, so we would like to see low numbers. The plots shows the mean of the results (green triangle) and the median (yellow line) for each model. Overall the goal is to get as high mean as possible so the yellow line in the plots should be on the upper part.

2.1 SKLearn Models

Firstly, I have evaluated some well known models from the *SKLearn* package. This can be seen in the following:

```
LR: 0.426811 (0.097457)
LDA: 0.497801 (0.103866)
NB: 0.555958 (0.083368)
DTC: 0.502476 (0.086138)
SVC: 0.073061 (0.050515)
KNC: 0.239904 (0.091180)
RFC: 0.370228 (0.134570)
GPC: 0.282091 (0.081094)
```

The result can also be seen in Figure 3. We can see that we got the best result (0.55) for the *GaussianNB* (named NB in the Figure 3.). The *SVC* model gave lowest standard deviation.

2.2 Linear Regression

Linear regression is the simplest type of regression analysis. It is a linear approach to modelling the relationship between a dependent and one or more independent variables. In our case we used a simple linear regression as we had one dependent variable. The relationship between the variables are modeled using linear predictor functions whose unknown parameters are estimated from the data.

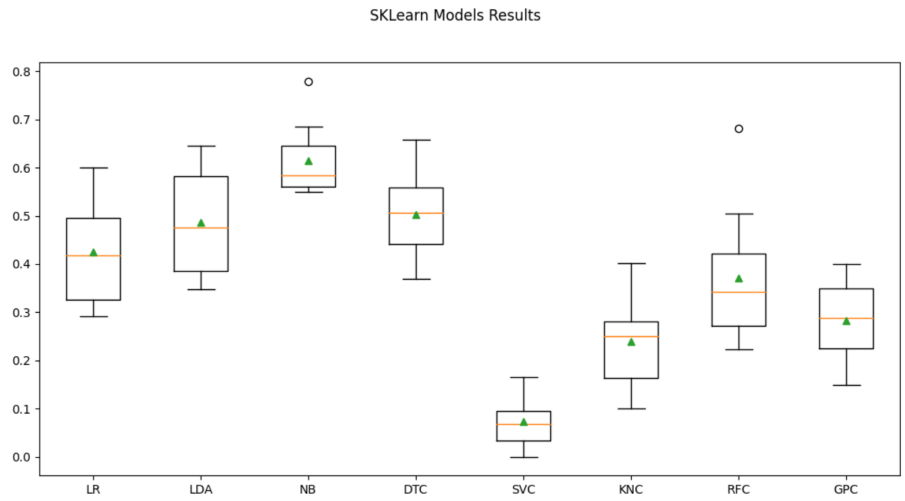

Fig. 3. Distributions of scores for various SKLearn [2] models

Using linear regression for our database I got 0.42 for the mean of the scores which is significantly lower and a only little higher standard deviation (0.09) than the results for the GaussianNB model,

2.3 Under-sampling the data

Under-sampling the data is a technique that is not as widely used as over-sampling. The idea is that not all of the variables have the same importance, so we can give more weight to the important ones. I have tried seven of these algorithms and in average I got the best result using these ones. This is because we have more attributes that do not influence the dependent variable. The result can be seen in the following:

```
TL: 0.653247 (0.071806)
ENN: 0.751309 (0.066780)
RENN: 0.757979 (0.059776)
NCR: 0.728194 (0.078452)
CNN: 0.664882 (0.072878)
AllKNN: 0.743580 (0.039726)
Miss: 0.661069 (0.071445)
```

The results can also be seen in Figure 4. I ran these models many times, but the results were changing all the time (not very significantly but the order of the models score's mean was changing), so I think I cannot make a clear conclusion from these models.

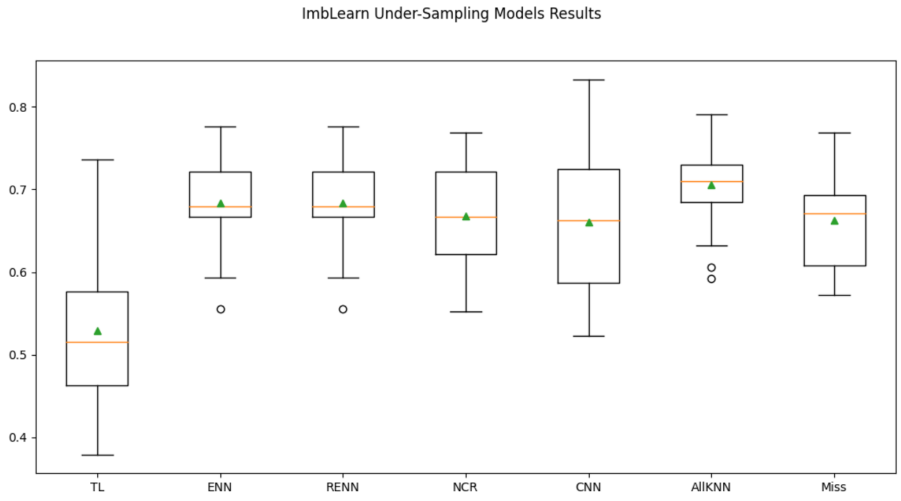

Fig. 4. Distribution of scores for various under-sampling models

2.4 Over-sampling the data

Over-sampling the data is a commonly used model for data analysis. One of the most known algorithms is *SMOTE* and basically all the other algorithms are only subversions of that. Despite the fact that the over-sampling models are more popular I got worse results using this model. The results are the followings:

```
SMOTE: 0.534616 (0.084242)
BLSMOTE: 0.536692 (0.089197)
KMSMOTE: 0.514895 (0.100835)
ROS: 0.705559 (0.064202)
SVMSMOTE: 0.585669 (0.078646)
```

Seen as well in Figure 5. By far the best results in this type were the ones using the *RandomOverSampler* (named ROS).

2.5 SMOTE - Synthetic Minority Over-sampling Technique

It is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

3 Clustering

Clustering [4] itself is not one specific algorithm, but the general task to be solved. I have used two different algorithms during the analysis and I compared

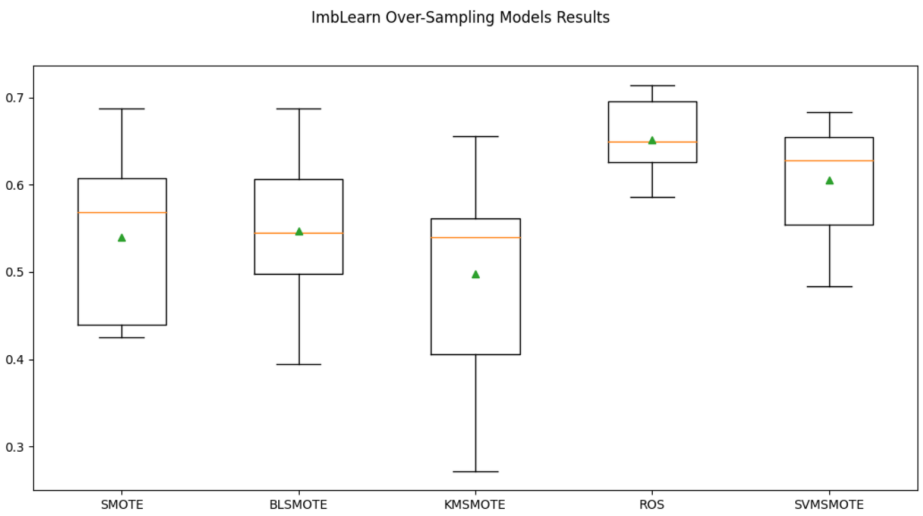

Fig. 5. Distribution of scores for various over-sampling models

them. These are the *K-Means* and the *MiniBatch K-Means* algorithms. In the followings I am going to explain their main concepts.

3.1 K-means

K-means is an algorithm which divides the data points into k groups so that:

- each group has a central point associated
- the sum of distances from points to their centrum is minimized

From these it trivially follows that:

- the centrum point is the average of the points inside the cluster
- each point is in the cluster whose centrum it is closest to

An iterative algorithm of two steps can be created: first we assign each point to the closest centrum (Starting values for the means are needed by the algorithm to be used in the first step), then we recalculate the means of the clusters. The steps are repeated until there is no change.

3.2 MiniBatch K-Means

MiniBatch K-Means is a modified, non-deterministic version of K-means. First, a mini-batch is formed using random samples from the dataset. Then, each sample in the batch updates the nearest centroid, pulling it towards itself. Samples are then reassigned to the closest centroid. The algorithm is repeated until there is only a small change in steps.

3.3 Comparison

The comparison of K-Means and MiniBatch K-Means on the dataset can be seen in Figure 6. As MiniBatch K-Means is non-deterministic, there is a chance that it provides bad results in a low number of trainings.

As the attributes in the dataset have different dimensions (age is in years, credit is in euros) which have different ranges, the data was normalized, so that the minimum value corresponds to 0, and the maximum to 1.

The given dataset contained only 3 attributes that were continuous (X02, X05, X13) and I made the clustering using all combinations made of two attributes ((X02, X05), (X05, X13), (X02, X13)). The figure 6 shows the result using the X05 and X13 attributes, namely the *credit amount of the customers* and the *age of the customers*. We can see that the clustering worked pretty well for the normalized data. Normalization was done using a min-max normalization. The MiniBatch K-Means algorithm gave somewhat different results than the the basic K-Mean one, because as I have previously said it contains some randomization. I have run this clustering many times as well and approximately half of the times I got the same result for the to algorithms with no differences. In Figure 6. some different points can be seen and the centrums are also not in the same place.

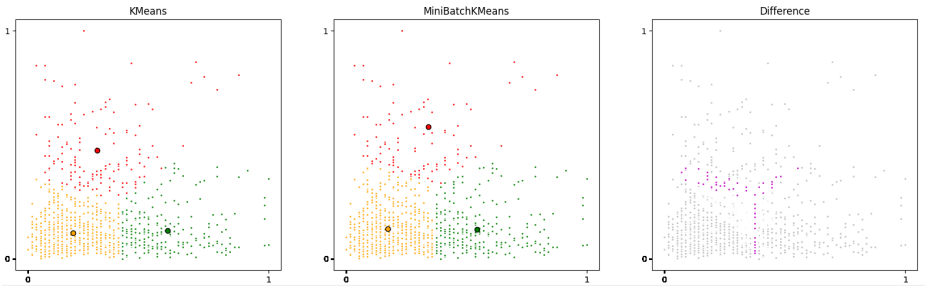

Fig. 6. Dividing the data into 3 clusters by age and credit amount. Age is normalized between 19-75 years, credit amount between €250-€18424

4 Frequent pattern mining

Frequent pattern mining [5] is a technique which finds the most frequent and relevant patterns in large datasets. It looks for subsets that have a frequency in the dataset above a (given or computed) threshold. This algorithms do not work very well on numbers so I did the frequent pattern mining only on the *object* type attributes, and the last (Y) attribute to find the patterns related to that.

The result of the following algorithms has the following structure. They show the pattern found in the database which cause the secondly shown pattern and

then its probability. As I was looking for the frequent patterns that are in relation with the last attributes (being good or bad customer), I am going to describe the results where the caused attribute is this.

4.1 FP Growth

I tried using the frequent pattern mining tool of chonyy/fpgrowth_py [7]. This algorithms looks for higher frequency and it does not check all the combinations of the attributes. It analyzes frequent patterns in a provided dataset and prints them out in the following format:

```
[{'A173'}, {'A201'}, 0.9698412698412698],
```

meaning that the presence of *A173* in a sample implies *A201* with a probability of 0.969... Clusters of multiple variables can also occur in frequent patterns, like

```
[{'A143', 'A101'}, {'A201', 'A152'}, 0.6792452830188679]
```

In order to predict good or bad customers, we use the patterns including '0's or '1's which correspond to these attributes. As most customers are good (Figure 1.2), there are no frequent patterns about bad customers, as expected. The patterns which imply a good customer (i.e. the '0' is on the right hand side, alone) are as follows:

```
[{'A152'},                {'0'},     0.7391304347826086]
[{'A201', 'A152'},        {'0'},     0.7328467153284671]
[{'A143'},                {'0'},     0.7248157248157249]
[{'A143', 'A101'},        {'0'},     0.7250673854447439]
[{'A201', 'A143', 'A101'}, {'0'},    0.7186629526462396]
[{'A201', 'A143'},        {'0'},     0.7161125319693095]
[{'A101'},                {'0'},     0.7001102535832414]
[{'A201', 'A101'},        {'0'},     0.6943181818181818]
[{'A201'},                {'0'},     0.6926272066458983]
```

As the overall probability of a (random) customer being good is 0.7, these patterns only weakly improve (or in some cases decrease) its chance. We can see that the biggest improvement comes from *A152* (customer owns a house) and *A143* (customer has no other installment plan). However, *A101* (no other guarantor for the credit) is basically neutral (very small improvement), whereas *A201* (the customer is a foreign worker) decreases the probability of having a good customer.

4.2 Apriori

The Apriori algorithm tool [6] starts with finding frequent items, and grows them into larger sets. From those sets, association rules can be deduced which highlight the overall trends in the database.

The output of the tool is very similar to the one given by the previous FP Growth tool. The algorithm produced a huge number of frequent item sets, a lot of them implying being a good customer. From those, only the ones with the highest probability are listed here:

```
('A152', 'A173', 'A143', 'A14')                 ==> ('0') , 0.944
('A101', 'A93', 'A143', 'A14')                  ==> ('0') , 0.944
('A173', 'A101', 'A152', 'A14', 'A143')         ==> ('0') , 0.942
('A93', 'A143', 'A14')                          ==> ('0') , 0.942
('A201', 'A101', 'A93', 'A14', 'A143')          ==> ('0') , 0.942
('A201', 'A173', 'A152', 'A14', 'A143')         ==> ('0') , 0.942
('A201', 'A93', 'A143', 'A14')                  ==> ('0') , 0.939
('A201', 'A173', 'A101', 'A152', 'A14', 'A143') ==> ('0') , 0.939
```

We can see that patterns with very high probability were found. Some of these correlations were also presented individually in the 1.2 subsection. We can see that the Apriori algorithm finds pattern probabilities and it considers bigger patterns then FP Growth as well. This also means that Apriori is slower which was not a problem for this dataset as it consists of only 1000 rows but it could be a problem for larger dataframes.

5 Further steps

Many other analysis could be done on the data. The relationships between the attributes have many possibilities. The models could be also handled in many different was for example other scoring and testing algorithms could be used. Clustering could be done with many other algorithm and they all could be compared to each other. Frequent pattern mining could also be checked not only for the dependent attribute, but it could help to know understand the relations between the attributes more.

References

1. "Predicting Credit Risk - Model Pipeline." Kaggle, 3 Nov. 2018, http://www.kaggle.com/kabure/predicting-credit-risk-model-pipeline
2. Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of Machine Learning Research 12 (2011): 2825-2830.
3. Brownlee, Jason. "Develop a Model for the Imbalanced Classification of Good and Bad Credit." Machine Learning Mastery, 27 Aug. 2020, http://machinelearningmastery.com/imbalanced-classification-of-good-and-bad-credit/.
4. Cluster analysis - Wikipedia https://en.wikipedia.org/wiki/Cluster_analysis
5. Association rule learning - Wikipedia https://en.wikipedia.org/wiki/Association_rule_learning
6. Apriori - Github https://github.com/asaini/Apriori
7. fpgrowth_py - Github https://github.com/chonyy/fpgrowth_py