Introduction to Data Science Assignment*

Gábor Vitrai - ABIOWE

[1] University of Eötvös Loránd, Budapest, Hungary http://www.inf.elte.hu
[2] Faculty of Informatics

Abstract. This paper was created for the Introduction to Data Science Subject as a final - end of year - assignment. The Data set this paper describes can be found in the reference list. The aim of the project is to analyse the data, and: 1) Develop a prediction model to classify the customers as good or bad 2) Cluster the customers into various groups 3) Provide some ideas on how frequent pattern mining could be utilized to uncover some patterns in the data and/or to enhance the classification.

Keywords: Data Science · Clustering · Regression · Frequent Pattern Mining

1 The Data Set

The data set which we will work with is the famous Bank loan set, often called the German credit data set. The included problem is the following: We have to classify the customers to two groups, good or bad. This sounds easy but there is a unique problem: Giving a loan to a bad customer marked as a good customer will result in a greater cost to the bank, than denying a loan to a good customer marked as a bad customer.

1.1 Data Analysis

First View After reading the data, with the help of some basic built in functions we can easily get some primary information of the records. We can see that we have 1000 customer records. If we want to take a closer look, we can read the following information: count: 1000, mean: 35,54, std: 11,37, min: 19, max: 75 and ect. To get an even better view, we can use the / *seabron.pairplot (dataFrame,hue='Y')* / function to get comprehensive plot collection.

Counting the data There are two classes distinguished. If a customer is good, it is marked with 1, if not, then with 2. A total of 70 percent of the records contains good customers, and 30 percent of examples are bad customers.

A cost matrix gives different penalty to each bad classification error for the positive class. A cost of **5** is applied to a false negative - marking a bad customer as good - and a cost of **1** is assigned for a false positive - marking a good customer as bad.

* Semester: 2020/21-1

Table 1. Count of customers

Category	Count	Percentage
1 - Good Customer	**700**	70,00
2 - Bad Customer	**300**	30,00

1.2 Numerical Data

The data set can be divided into two category in the meaning of attributes. The basic numerical columns are easier to deal with. These are simple values like the age of the customer or the the customer's credit in the bank.

1.3 Categorical data

We can see that the remaining categorical columns are encoded. This is marked with an ”Axx” format, where “x” are integers for different labels. Some sort of further encoding of the categorical variables will be required.

2 The Models

In this section, I will write about the process of creating, testing, evaluating and fine tuning of different models. First we divide the data into two groups. The target - in our case it is a vector - is containing values 0 or 1. This shows us how we classify the customers to the ”Good” or ”Bad” category. We will use this data to train our models, so they can predict this result by themselves. The features are the data consisting of every other attribute.

2.1 Training the Test Split

Scaling We start with standardizing our values. For this we use a StandardScaler() and/or a MinMaxScaler() from Scikit learn. Scaling data is the process of increasing or decreasing the magnitude according to a fixed ratio, in other words we change the size but not the shape of the data. Features are basically the column names and the respective data in that column. As an example we can see how the age of the customer is scaled by different scalers:

Table 2. Scaling results

Age of Customer	Standard scaling	Min-max scaling	Max-abs scaling	Robust scaling
67	2.766456	0.857143	0.893333	2.266667
22	-1.191404	0.053571	0.293333	-0.733333
49	1.183312	0.535714	0.653333	1.066667
45	0.831502	0.464286	0.600000	0.800000
53	1.535122	0.607143	0.706667	1.333333

Encoding: After scaling, we have to encode our data. Numerical data is not needed to be encoded, so we will only encode the categorical data. For this, we use OrdinalEncoder(). It is a natural encoding for ordinal variables. By default, it will assign integers to labels in the order that is observed in the data, which is perfect for us.

Filling missing data: In our practical project, we do not have missing data, but we will add a SimpleImputer to the code as well. The aim is to make the code work even if we have NaN data in the future. The Imputer will replace the missing values with median values, so the statistical values will not change dramatically.

Pipeline: To sequentially apply a list of transforms, we use Pipelines. The steps of a pipeline must be transforms, so we must use fit and transform methods. The purpose of a pipeline is to put together several steps while setting different parameters. In our case, first we have the numerical pipeline so we can use the above mentioned Imputer and Scaler. Using this pipeline, the created ColumnTransformer will allow different columns of the input to be transformed separately and the features generated by each transformer will be concatenated to form a single feature space. We use the created Transformer to fit our features. This will fit all transformers and will transform the data and concatenate the results.

Training: With this **transformed** data we can start the training. The code will split arrays or matrices into random train and test subsets. The result is the Trained data of the **Features** and the **Test** data. The function also returns the Trained and Test data of the **Target** which is crucial information for predictions. With the "test-size" parameter we can set what proportion of the data to include in the test split.

```
target = target = dataFrame["Y"]
X_train, X_test, y_train, y_test =
train_test_split(prepared_data, target, test_size=0.3)
```

2.2 KNN - K Nearest Neighbors:

In KNN classification, an object is classified by a plurality vote of its neighbors with the object being assigned to the class most common among its k nearest neighbors. When k is equal to 1, the object is simply assigned to the class of that single nearest neighbor.

Code sample for model training:

```
# Creating a Classifier instance
knn = KNeighborsClassifier(...)
# Fitting the model to our trained data
knn.fit(X_train, y_train)
```

```
# Evaluating the score
model.score(X_train, y_train)
model.score(X_test, y_test)
# Creating prediction with the model's own default prediction method
y_pred = model.predict(X_test)
# Create the Report of the classification,
containing precision levels and other important values
classification_report(y_test, y_pred)
```

The scores and the created reports looks like the following.
Trained score: **0.728**5714285714285
Test score: **0.746**6666666666667

```
Classification Report:
              precision    recall  f1-score   support
           0       0.75      0.96      0.84       213
           1       0.69      0.23      0.34        87

    accuracy                           0.75       300
   macro avg       0.72      0.59      0.59       300
weighted avg       0.73      0.75      0.70       300
```

Please note, that with each run of the models, the results can differ.

Scores: The scores are calculated by the models, are the mean accuracy on the given test data and labels. In our case, this is a subset accuracy which is a metric, which will allow every label to be correctly predicted.

Classification Report: The most important data we get from the reports are the **precision** values. They tell us what proportion of positive identifications was actually correct.
Recall: What proportion of actual positives was identified correctly? This means that 23% of the customers who got labelled as bad customers, were actually bad customers, but 77% got labelled wrongly. This is actually not as bad as it seems, because with a very basic non-tuned model, we made smaller mistakes with guessing the good customers bad, rather then the bad customers as good. (Which is 96% correct)

2.3 SVM - Support Vector Model

In machine learning, support-vector machines are *supervised* learning models with associated learning algorithms that analyze data used for *classification* and *regression* analysis. It is especially powerful for smaller data sets, so it is worth trying it out on our data.

Results: After running the same basic procedure as in the case of KNN, we get the following results:
Trained score: **0.78**57142857142857
Test score: **0.74**

```
Classification Report:
      precision    recall  f1-score   support
   0       0.77      0.88      0.82       204
   1       0.64      0.44      0.52        96
```

Notice: We can notice, that the results - in average - are better then in the case of the K Nearest Classifier, but to get a better view, lets run the same test with 2 more different models.

2.4 LR - Logistic Regression

Logistic Regression is a *Machine Learning* classification algorithm that is used to predict the probability of a *categorical dependent* variable. (Right now we are using all the four mentioned models for prediction). In *logistic regression*, the dependent variable is a binary variable that contains the data, coded as 0 or 1. With the use of this model we get the following results.
Trained score: **0.77**85714285714286
Test score: **0.74**66666666666667

```
Classification Report:
      precision    recall  f1-score   support
   0       0.77      0.88      0.82       204
   1       0.64      0.47      0.54        96
```

2.5 GPC - Gaussian Process Classifier

Gaussian Processes can be used as the basis for sophisticated non-parametric machine learning algorithms for *classification* and *regression.* It is very similar to SVM, but it is capable of predicting highly calibrated class membership probabilities.
Trained score: **0.73**
Test score: **0.69**

```
Classification Report:
      precision    recall  f1-score   support
   0       0.81      0.71      0.76       204
   1       0.51      0.66      0.58        96
```

2.6 Conclusion: SVM - Additional Training

By looking at the results and the training scores, we can withdraw the conclusion, that the best models without tuning, are the SVM and the LR. In this section we will try to improve the results of the SVM model, which got picked by it's score out of the four models.

GridSearchCV: For additional improvement we will use **GridSearchCV**, which is an exhaustive search over specified parameter values for an estimator. Here we specify a dictionary with the requested parameters, to which we can train our model. The execution will result in the recommended estimator. The model is trained on the full development set and the scores are computed on the full evaluation set. The execution tested different kernels (e.g.: rbf, poly, sigmoid, linear) and different gamma values. At this point we can use the best suited model for predicting, lets see the results.
Best suitable model: {*'C': 10, 'gamma': 0.01, 'kernel': 'rbf'*}.
Trained score: **0.79**42857142857143
Test score: **0.75**

```
Classification Report:
              precision    recall  f1-score   support
           0       0.82      0.84      0.83       216
           1       0.56      0.51      0.53        84
    accuracy                           0.75       300
   macro avg       0.69      0.68      0.68       300
weighted avg       0.74      0.75      0.75       300
```

Accuracy: **0.74** (+/- 0.04)
Cross validation is a model validation technique to test how the results of a statistical analysis will generalize to an independent data set.
At this point we can notice, that with the basic macro, the results are good, but we can test other macros, and we can try to cross validate our scores. For example we can use the f1_macro as an experiment. With this macro the accuracy drops to 75%. With f_beta scoring, we get a slightly better result:
Accuracy: **0.80**39591861947809 (+/- 0.06)

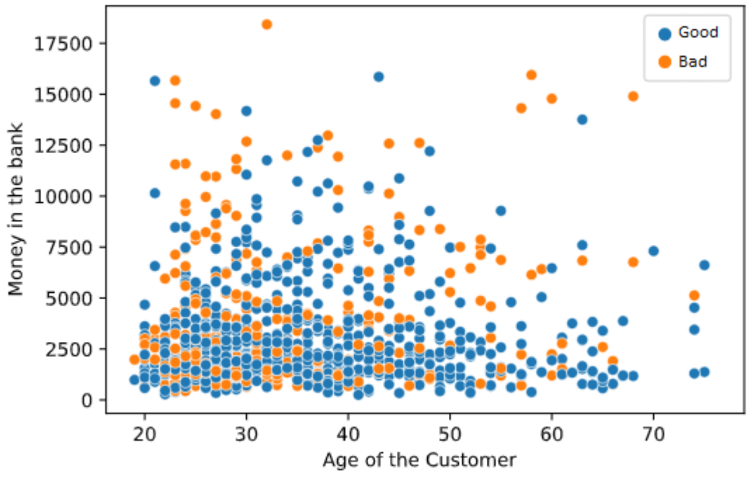
3 Clustering

Clustering is an example of an unsupervised learning model, when we feed non-categorized training data into a model and have our model attempt to categorize that data in some way according to its features. With simpler words the algorithm will try to assign each customer to different groups based on their features.

3.1 Analyzing the data for different information

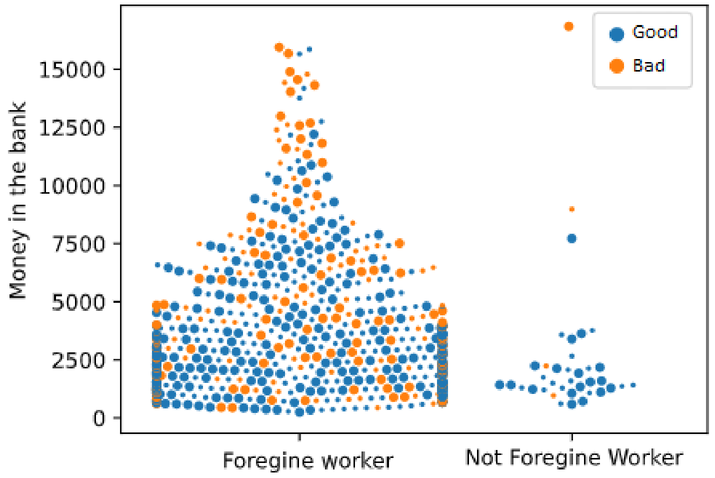
First to decide what feature is worth clustering, lets see a simple plot based on the **Customers money in the bank** (in EUR) and by **Customer's Age**. This will show us, whether there is a correlation between having more money in the bank with becoming a good customer.

Fig. 1. Plot Good and Bad Customers without clustering



We can not really tell if there is important information in this plot From similar plots, we could visualize the fact, that owing an apartment can increase the chance of becoming a good customer. In case the customer is **not renting** a flat or a house, they are more likely will get **clustered as good customers**. With the help of plotting we can also see, that foreign workers often have **more money in the bank**, and as we can see, domestic customers are mostly labeled as good customers despite having just little money in the bank.

Fig. 2. Foreign and domestic customers

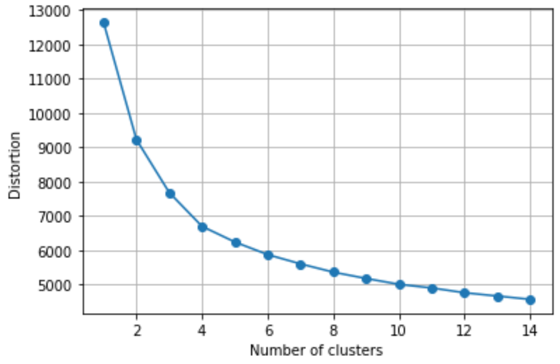


KMeans The k-means clustering method is an unsupervised machine learning technique used to identify clusters of data objects in a data set. There are many different types of clustering methods, but this one is one of the oldest and most approachable.

3.2 The Elbow Method

Upon clustering, one of the biggest question is the number of clusters, so how many groups do we want to cluster the customers into. To determine the "recommended" amount, we use a technique called Elbow method. The Elbow method will create KMeans instances with different amount of clusters and measure the distortion amount for the range of clusters. For our trained data, we can get the following result. On the plot below, we can see that the *elbow point* is at number four, so lets use this number for clustering our customers.

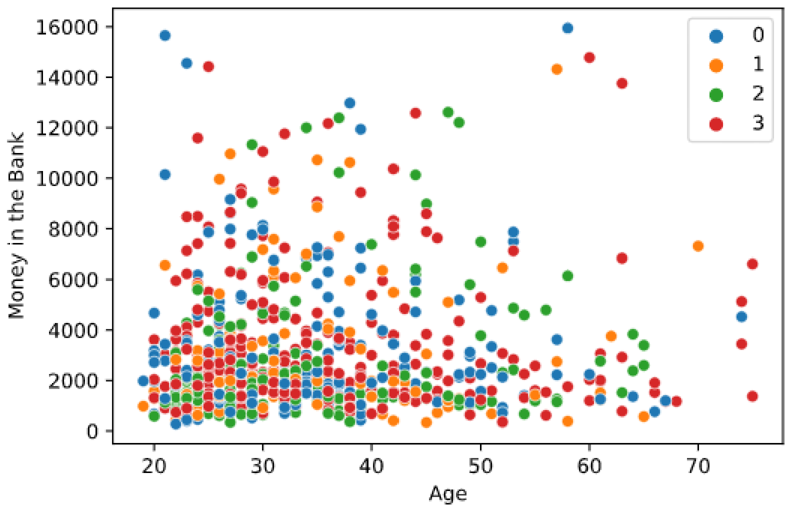
Fig. 3. The Elbow method



3.3 Clustering by All attributes

For clustering we will project our data onto the same space, with the same attributes, so lets see how the KMeans algorithm will cluster our customers into our *four* groups, based on ALL their attributes.

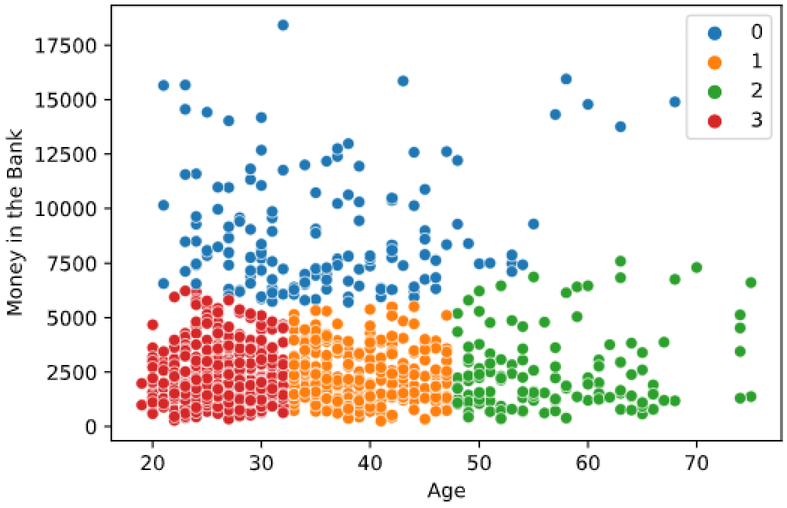
Fig. 4. Clustering by All Attributes



3.4 Clustering by Specific Attributes

We could tell, that through a human's eye, we can not see any correlation from the outcome of the clustering, but the computer found the important relations. Now let cluster the data based only on **Age** and **In bank credit**.

Fig. 5. Clustering by Specific Age and Money



As we can see the clusters formed 4 groups. The red one (#3) is the youth, with average amount of money (*0-6000 EUR*). The Orange groups is describing the middle-age adults, and the green one is the elderly. We can also notice that the bank has less customers at this age. The fourth (#0) and most interesting group is the blue one, which represents the richer layer, in all the age groups. (*6000-18000 EUR*)

4 Frequent pattern mining

Frequent pattern mining is part of knowledge discovery in databases and it describes the task of finding the most frequent and relevant patterns in large data sets. In this section we will use a pattern mining library to show a simple example. An FP growth algorithm represents the database in the form of a tree, called a frequent pattern tree. The structure of the tree will maintain the association between the item sets. Then, the item sets of the fragmented patterns are analyzed.

Example: Lets see an example for the given data set.
First of all, we have to prepare the data set we want to work with. This case we need the **categorical data *and* the target** vector. After importing the library to the project, we can select the feature set and tell the algorithm what confidence level should it use. After the algorithm finished, the output is an array of the following structure:

```
[{'A32'}, {'A201'}, 0.9622641509433962]
```

This shows us that the existence of '**A23**' value will imply '**A201**' with a **96,2%** chance. In our project we want to see what patterns will lead to classifying a customer as good. With this filtering, the result is the following:

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

The most important things to look for are the patterns which imply a good customer. In our case, good customers are marked with "**0**". As I mentioned before, the overall chance of a customer being classified as good customer is around 70%. As we can see these patterns just slightly improves that. Now we can start looking for attributes causing improvements. On the top, we can immediately see, that the **'A152'** parameter helps a lot. (This number means that the customer owns a house). The **'A101'** appears many times (No other debtors or guarantors for the credit) in the middle of the range, but it does not effect the probabilities heavily.
With this example we can see why Frequent pattern mining can be useful to enhance classification procedures. It is very simple to find correlations and this technique can even help to spot new points for analysing the data.

References

1. SVC Classification, http://www.tryanalyticsblog.com/svm-classification-credit/. Last accessed 10 Dec 2020
2. K Means Clustering, https://towardsdatascience.com/k-means-clustering-of-university-data-9e8491068778. Last accessed 8 Dec 2020
3. GridSearchCV, https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/. Last accessed 8 Dec 2020
4. Predictive Models, https://financetrain.com/case-study-german-credit-steps-to-build-a-predictive-model/. Last accessed 8 Dec 2020
5. Classification, https://machinelearningmastery.com/imbalanced-classification-of-good-and-bad-credit/. Last accessed 10 Dec 2020
6. Data Prediction, https://www.kaggle.com/hendraherviawan/predicting-german-credit-default. Last accessed 9 Dec 2020
7. One-Hot Encoding, https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/. Last accessed 8 Dec 2020
8. Scaling, https://medium.com/@stallonejacob/data-science-scaling-of-data-in-python-ec7ad220b339. Last accessed 10 Dec 2020
9. Fp-Growth, Frequent Pattern Mining, https://towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3. Last accessed 11 Dec 2020