Task2: Documentation

January 30, 2019

Diego Kozlowski

The following document will present a brief summary of some key-points from the second-task's workflow.

1 Feature engineering

1.1 Handling missing data

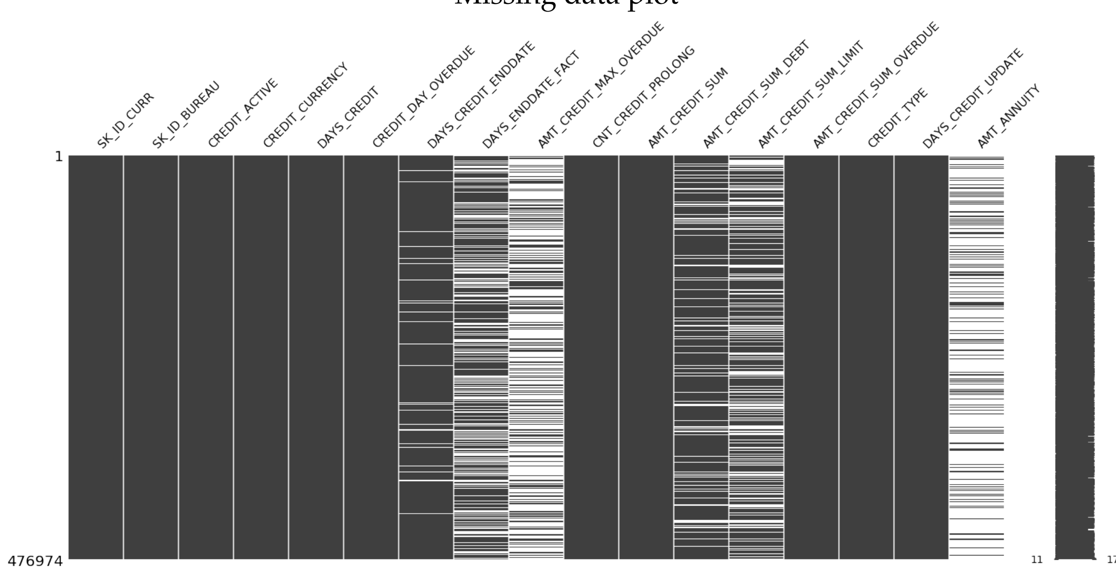
A revision of both datasets shows that there are no *NaN* values on the main data set, but there are plenty in the additional dataset:

```
additional_df shape: (476974, 17)
additional_df shape dropping NA: (12920, 17)
```

Dropping those rows with *NaN* would imply a big lose of information. Because of this, I studied the behavior of this missing data, in order to find a better way to address the problematic.

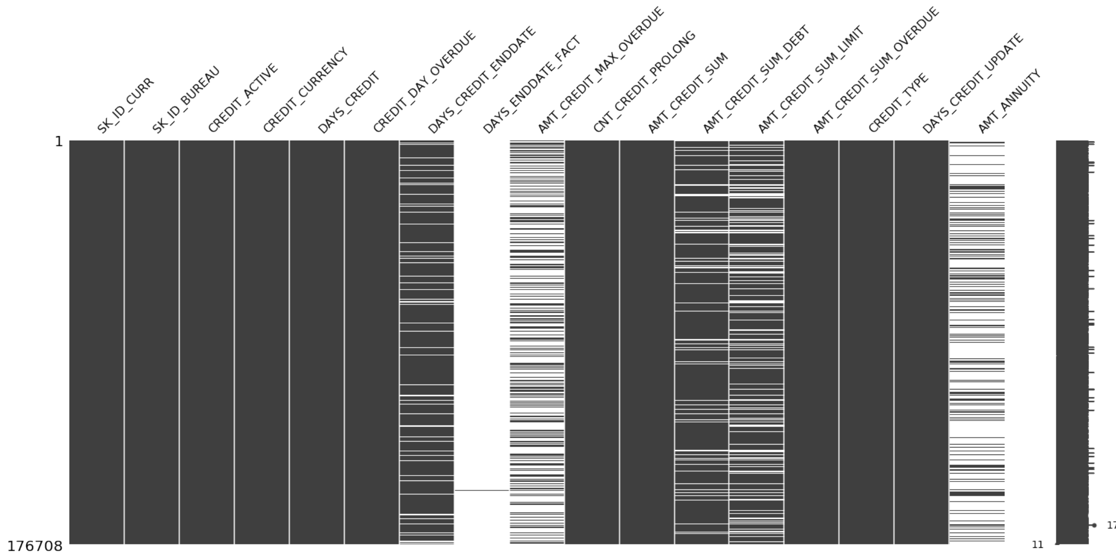
in 1.1 we can see a missing-data plot that reflects in white the NaN on the dataset, for each feature

Missing data plot



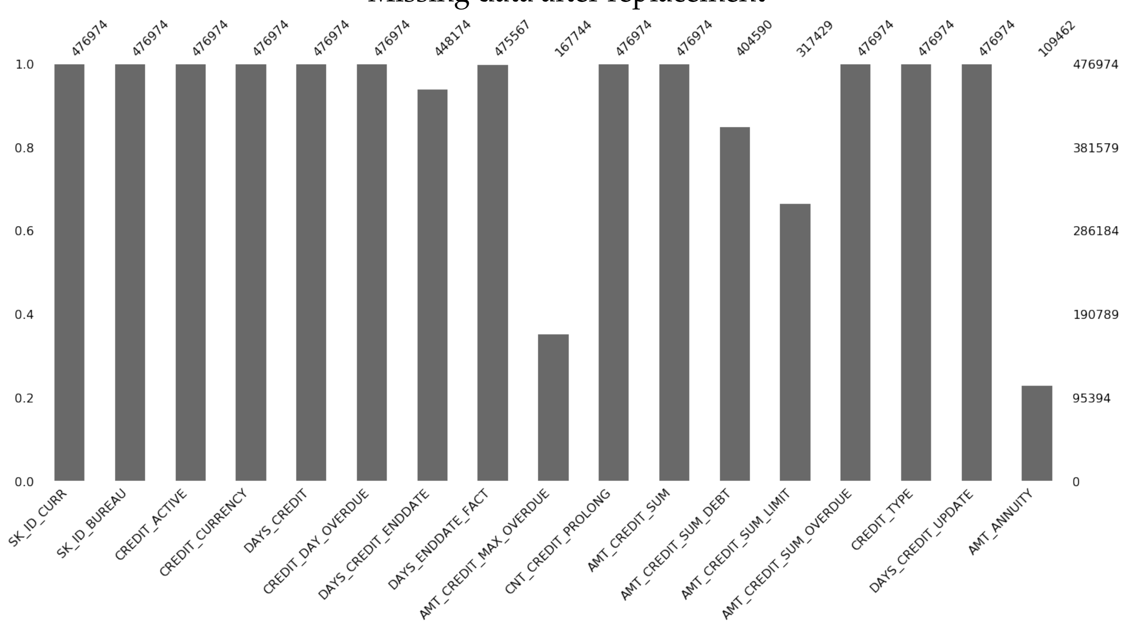
there are three variables that present significantly more *NaN* than the rest. I study the relation between those variables and the rest. In 1.1 it can bee seen that when the variable *CREDIT* is 'Active' there is no information on DAYS_ENDDATE_FACT.

Missing data plot when CREDIT is 'Active'



Given this, I decided to replace this *NaN* with '1'
After this, we can see in 1.1 that the *NaN* problem in this variable is solved.

Missing data after replacement



Because there are two variables with many *NaN*, I decided to remove them, so the consistency of the dataset is increased.

The final step for dealing with the missing data was to remove those rows which had missing values:

```
Orignal shape: (476974, 17)
Final shape: (300948, 15)
```

1.2 Feature selection for the Additional data

In order to use the additional information, I calculate the following summary measures by SK_ID_CURR, for the numerical variables:

- mean
- median
- sum
- count

1.2.1 Categorical Data

First, I exclude SK_ID_BUREAU as it is an ID.

After this, I did an analysis over the categorical variables in order to understand if they can add valuable information. The following tables shows the distribution of the values from the categorical data in the different classes of the target:

```
Out[53]: TARGET  CREDIT_ACTIVE
         0.0     Closed           0.614313
                 Active           0.384567
                 Sold             0.001116
                 Bad debt         0.000004
         1.0     Closed           0.544749
                 Active           0.452858
                 Sold             0.002393
         Name: CREDIT_ACTIVE, dtype: float64
```

```
Out[54]: TARGET  CREDIT_CURRENCY
         0.0     currency 1         0.999645
                 currency 2         0.000328
                 currency 3         0.000018
                 currency 4         0.000009
         1.0     currency 1         1.000000
         Name: CREDIT_CURRENCY, dtype: float64
```

```
Out[55]: TARGET  CREDIT_TYPE
         0.0     Consumer credit                             0.761632
                 Credit card                                 0.212575
                 Mortgage                                    0.012491
                 Car loan                                    0.010719
                 Microloan                                   0.001566
                 Unknown type of loan                        0.000531
                 Another type of loan                        0.000337
                 Loan for business development               0.000108
                 Cash loan (non-earmarked)                   0.000031
                 Loan for the purchase of equipment          0.000004
                 Real estate loan                            0.000004
         1.0     Consumer credit                             0.744323
                 Credit card                                 0.233129
                 Mortgage                                    0.008774
                 Car loan                                    0.007764
                 Microloan                                   0.005424
                 Another type of loan                        0.000266
                 Unknown type of loan                        0.000266
                 Loan for working capital replenishment      0.000053
         Name: CREDIT_TYPE, dtype: float64
```

The distribution over the categories in this variables don't change with respect of the target (except maybe the active/closed status).

As there is no direct way to add this variables to the final dataset, such as summary measures, and from the distribution of the variables it doesn't emerge a clear patron (i.e. all the *car loans* have target=1) I decided to exclude this group of variables.

1.2.2 Feature engineering: final remarks

When I join the two datasets, new NaNs appears as not all the SK_ID_CURR from the main data exists in the cleaned additional dataset.

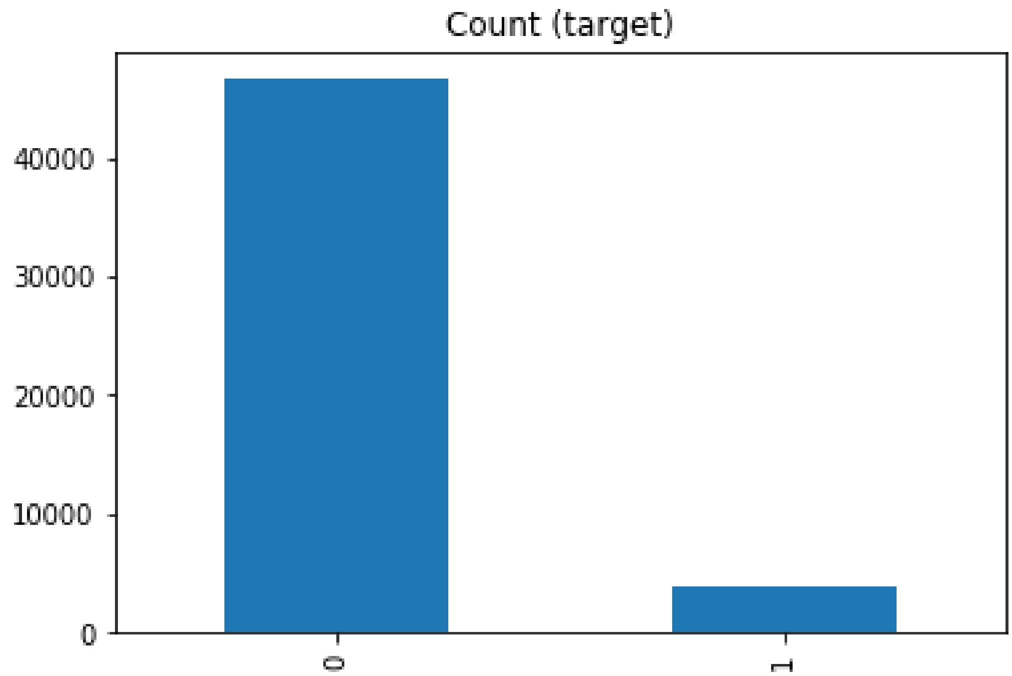
I drop the data with *NaN*. I could imput by the mean or a specific value, but the I consider that the amount of loss is manageable

Also, I split in train and test and validation (0.8,0.1,0.1). The latter one will be used for model comparison. Besides I need to One Hot encode categorical variables

1.3 Class imbalance

A quick analysis over the distribution of the classes in the train set show the following:

```
Class 0: 46565
Class 1: 3936
Proportion: 11.83 : 1
```



The target is heavily unbalanced. Here is important to recognize the problem objective, because this determines how to proceed. The Documentation states that the goal is to *predict whether to offer loans to clients* but we do not know the relative costs of **false positives** and **false negatives**.

I will make the following **assumption**: The main goal is to predict well the **positive** class, because it imply more costs, i.e. the target is the defaulted loans. Hence a correct prediction of this is more valuable than a correct prediction of the class 0.

Because of the previous assumption I will *resample* the dataset in order to balance the classes. In order to avoid loosing more data, I decided to use **random over-sampling**.

The final training set has the following distribution:

```
Random over-sampling:
1    46565
0    46565
Name: target, dtype: int64
```

2 Modelling

I will try three different models, from the simplest to the most advanced techniques:

- Logistic regression
- Naive Bayes
- Gradient Boosting Machine

I decided to use a Gradient Boosting Machine (from the same family of models of the famous XGBOOST) and not a Deep Learning approach because of the input data. In my experience structured data is best fitted by ensembles of trees, while is better to approach unstructured problems(image, audio, etc) with neural networks.

2.1 Results

Logistic regression

```
Out[Accuracy]: 0.5980042765502495

Confusion Matrix:
[3114 2046]
[ 210  242]
              precision    recall  f1-score   support

           0       0.94      0.60      0.73      5160
           1       0.11      0.54      0.18       452

   micro avg       0.60      0.60      0.60      5612
   macro avg       0.52      0.57      0.46      5612
weighted avg       0.87      0.60      0.69      5612
```

Naive Bayes

```
Out[Accuracy]: 0.5181753385602281

 Confusion Matrix:
[2609 2551]
[ 153  299]
              precision    recall  f1-score   support

           0       0.94      0.51      0.66      5160
           1       0.10      0.66      0.18       452

   micro avg       0.52      0.52      0.52      5612
   macro avg       0.52      0.58      0.42      5612
weighted avg       0.88      0.52      0.62      5612
```

Gradient Bosting Machine

```
Out[Accuracy]: 0.6534212401995724
```

```
Confusion Matrix:
[3394 1766]
[ 179  273]
              precision    recall  f1-score   support

           0       0.95      0.66      0.78      5160
           1       0.13      0.60      0.22       452

   micro avg       0.65      0.65      0.65      5612
   macro avg       0.54      0.63      0.50      5612
weighted avg       0.88      0.65      0.73      5612
```

The two best models are the Naive Bayes and the Gradient Boosting Machine. There is an important trade-off between the recall of the two classes. In order to optimize this, it would be necessary more information of the costs involved in the different types of mistakes. This would modify both the resampling and the model selection (between Naive Bayes and the Gradient Boosting Machine)

2.2 Final Model

Based on the results on the validation set, I choose the Gradient boosting machine model, and test it over the test-set

```
Confusion Matrix:
[3798 1941]
[ 204  292]
              precision    recall  f1-score   support

           0       0.95      0.66      0.78      5739
           1       0.13      0.59      0.21       496

   micro avg       0.66      0.66      0.66      6235
   macro avg       0.54      0.63      0.50      6235
weighted avg       0.88      0.66      0.73      6235
```

3 Final notes

This workflow could be enhanced by using a pipeline from *sklearn.pipeline* and cross-validation for the fine tuning of the final model. This would also be helpful in order to choose if the missing data should be excluded (like I did) or imputed, and how (mean, 0, or maybe some imputation algorithm).

Given the limited time for this exercise and that the Boosting Machine default parameters are known to be robust, I decided to skip this part. The decision of excluding the missing values was supported with the validation set, but is not showed for the clarity of the code