Capstone Project Machine Learning Engineer Nanodegree	Andrii Stelmashenko February 13th, 2018

Definition

Project Overview

Loans are very popular in US. People take loans for education, property, car, etc. The definition of loan is:

A loan is money, property or other material goods that is given to another party in exchange for future repayment of the loan value amount along with interest or other finance charges [1].There is a risk that a lender may not receive given credit back, it's called 'Credit Risk' [2]. Lenders, of course, want to minimize credit risks, for that they calculate 'Credit Score' [3]. A credit score is a number representing the creditworthiness of a person, the likelihood that person will pay his or her debts. Credit score is composed from several components related to a person, one of major components is 'Credit History'. A credit history is a record of a borrower's responsible repayment of debts.

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders. It's possible to use alternative data, including telco and transactional information, to predict their clients' repayment abilities, which can help people to get loans from trustworthy lenders.

Currently they rely on statistical methods (they do not provide any additional details) and want to use modern ML based approach which may perform better comparing to currently used.

Problem Statement

The goal of this project is to predict probability that a person will pay it's debts based on related to the person financial information, some kind of alternative to 'Credit Score'.

Dataset is provided by competition organizer - HomeCredit company, and it is hosted on kaggle platform. HomeCredit_columns_description.csv file contains descriptions for the columns in the various data files. There are 221 row in the file with human-readable description for each column. There is TARGET column: ‘Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)’, this column means that it is labeled data with possible two classes thus it is supervised learning binary classification problem.

There number of algorithm to solve this type of problem like SVM, Neural Networks, Ensemble Methods (GBDT), Bayesian Based methods, etc. To pick proper algorithm for the task requires to understand which one fits best and it mostly depends on data set. Main factors to make decision usually are data set size, number of features, data quality, feature value type (categorical or numerical) and many more nuances.

From the first look at the data, there are 307511 rows in the training data set which seems enough for complex algorithms which require much data to learn. Data preprocessing is specific to an algorithm e.g. some algorithms natively work with categorical features another ones require preprocessing of categorical features. After data preprocessing model will learn and then it’s hyper-parameters fine-tuned to choose final model parameters.

Metrics

Submissions are evaluated on area under the ROC curve [4][21] between the predicted probability and the observed target. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The evaluation logic is the next: submitted results are actually two probability distributions of two classes - 0: the loan was repaid or 1: the loan was not repaid. ROC curve will show how we did that split, it visualizes all possible classification thresholds. The key point to note is the area under curve (AUC) is the highest when the two curves are farthest with little overlap. AUC represents the probability that a classifier will rank a randomly chosen positive observation higher than a randomly chosen negative observation.

The result will help HomeCredit business to decide if they want e.g. to minimize False Positive Rate or maximize True Positive Rate. Below is the illustration [22] of ROC curve and possible split of Positive and Negative points.

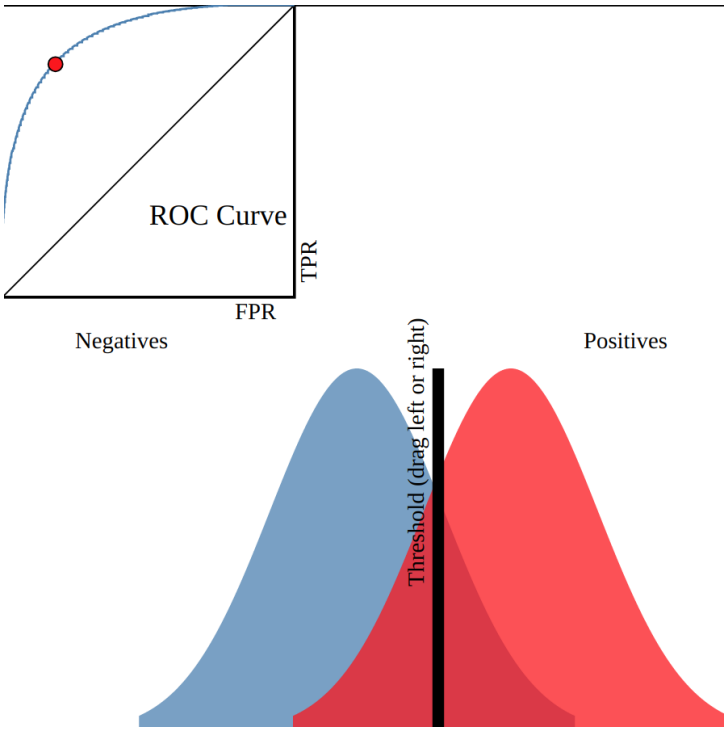

Figure 1 – ROC illustration

Analysis

Data Exploration

Dataset is provided by competition organizer. HomeCredit_columns_description.csv file contains descriptions for the columns in the various data files. There are 221 row in the file with human-readable description for each column. Relations between files are provided as a diagram:

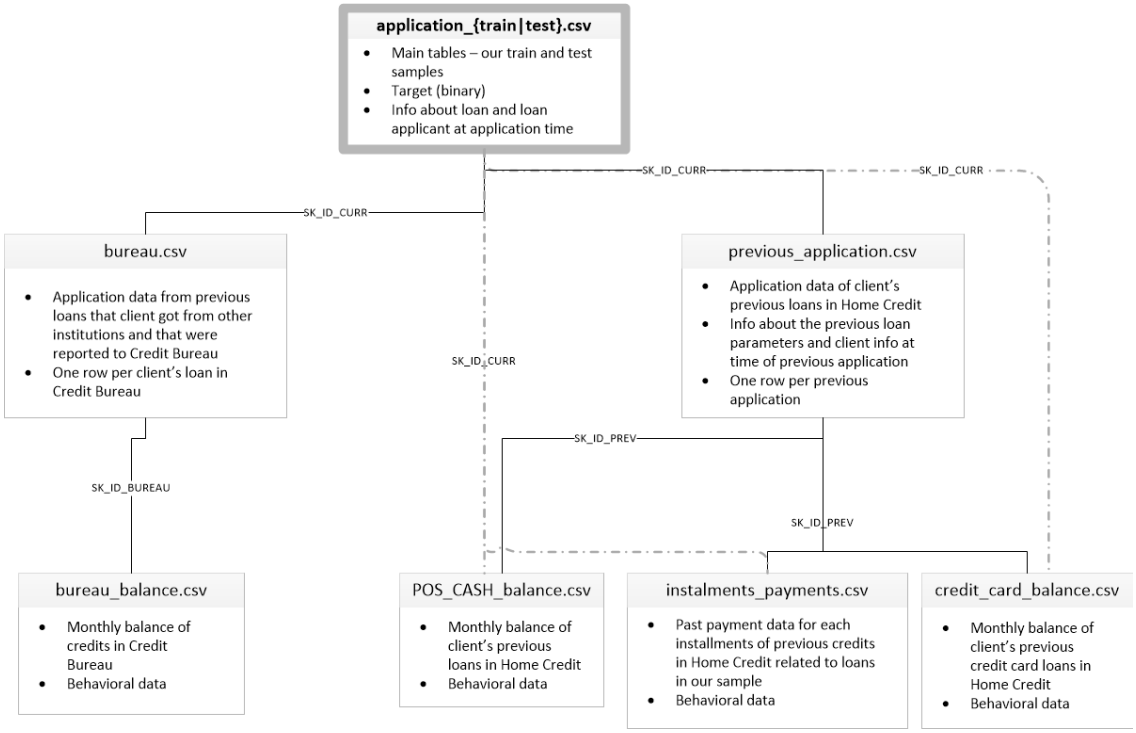

Figure 2 – Input data relations

Data files are split, later all these files should be concatenated using techniques like SQL Left Join[5] into single csv file. Later in text analysis is provided only of some of columns which are discovered to be interesting, general analysis will be done only on one column as an example, other similar columns will be skipped to reduce document size.

- application_train/test: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.
- bureau: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- bureau_balance: monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- previous_application: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- POS_CASH_BALANCE: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- installments_payment: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

All columns are of one of three data types: int, float, object. These types correspond to logical data type of each column: continuous, numerical discrete and categorical. Categorical columns are represented by object and int types. Let’s count unique values:

```
app_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```

Partial output:

```
NAME_CONTRACT_TYPE              2
NAME_EDUCATION_TYPE             5
WEEKDAY_APPR_PROCESS_START      7
ORGANIZATION_TYPE              58
```

Some columns have only 2 unique values and some have more - 58. Most algorithms work only with numerical values, so it is necessary to encode string values to numerical. Two most used techniques for that are Label Encoding and One-Hot-Encoding.

Columns of continuous type should be analyzed on anomalies. Anomaly is the deviation in a quantity from its expected value[6]. Since it’s often unclear what is ‘expected value’ there is number of statistical methods how to detect anomaly or outlier, Tukey’s mdethod[7] is one of them. It is based on interquartile range (IQR), the range is the next:

```
[Q1 - k * (Q3 - Q1), Q3 + k * (Q3 - Q1)]
```

Where `k=1.5` indicates an outlier, `k=3` means value is “far out”.

Let’s take AMT_INCOME_TOTAL column and apply IQR, the column has 3014 values which are far out from the rest. Executing outliers search for all continuous columns gives 3381 outlier when `k=3` and 25350 when `k=1.5`. AMT_INCOME_TOTAL contains most outliers. The column holds income of a client. These data points are very important because if client’s income is big then most likely it will take big loan, if we look at such clients (see Data_Exploration.ipynb):

```
app_train.loc[income_total_outliers]['TARGET'].astype(int).value_counts()
```

```
0    2852
1     162
```

There is portion of rich clients which do not return loans back.

It depends on algorithm chosen if removing outliers is necessary, some algorithms are sensitive to outliers and some are not. In case of this projects outliers may be very important and it is better to chose algorithms which are not sensitive to outliers.

Missing values is another problem in real world projects. There are a lot of columns where missing values percentage goes up to 60:

Column Name	NaN %	NaN Count
COMMONAREA_AVG	69.872297	214865
NONLIVINGAPARTMENTS_MODE	69.432963	213514
NONLIVINGAPARTMENTS_MEDI	69.432963	213514
...		

Table 1 – Missing values

Having missing values in a dataset can cause errors with some machine learning algorithms. There are number of options what to do with missing values: drop rows or event whole columns where there are to many missing values, impute missing values, e.g. use mean or zeros.

Research on numerical columns has done using Five Number summary [9]. Let's look at AMT_CREDIT column:

```
app_train['AMT_CREDIT'].describe()
```

```
count    3.075110e+05
mean     5.990260e+05
std      4.024908e+05
min      4.500000e+04
25%      2.700000e+05
50%      5.135310e+05
75%      8.086500e+05
max      4.050000e+06
```

It allows to see selected feature value range and how it changes. The 1st and 3rd quartiles give a sense of the spread of the data, especially when compared to the minimum, maximum, and median.

Among numerical features anomaly found in DAYS_EMPLOYED column. It has group of abnormal value 365243 which is how many days before the application the person started current employment. This values means a person worked 1000 year before application, it seems like this number is sort of flag information is unknown or is not filled by some reason.

Exploratory Visualization

Target column is binary, it takes either 0 or 1 values. Let's look it histogram plot.

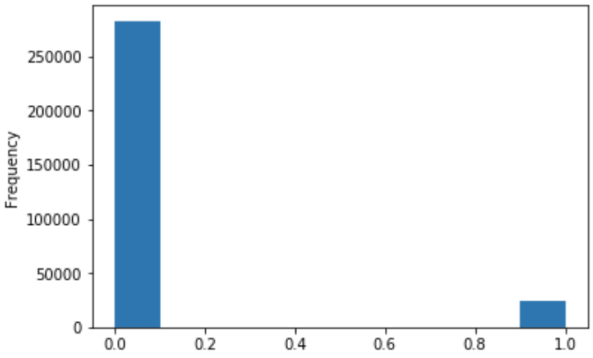

Figure 3 – Target distribution

From the fig. 2 it is seen that there is imbalanced class problem[8].

Missing values, was mentioned already above, can be visualized using data-dense matrix:

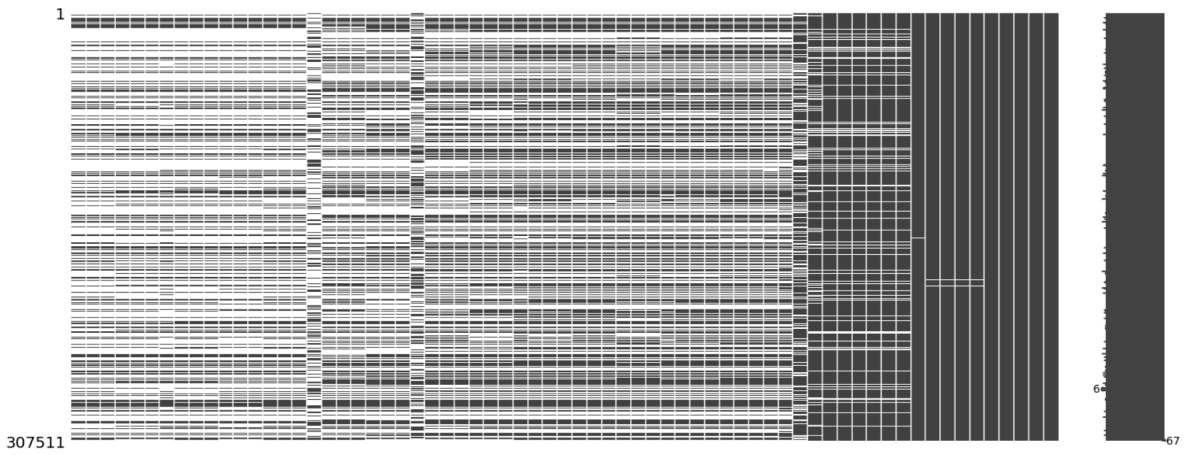

Figure 4 – Missing values matrix

It shows that most rows have missing values in a lot of columns at the same time (see horizontal blank lines). Columns are sorted from the biggest portion of missing values to the smallest:

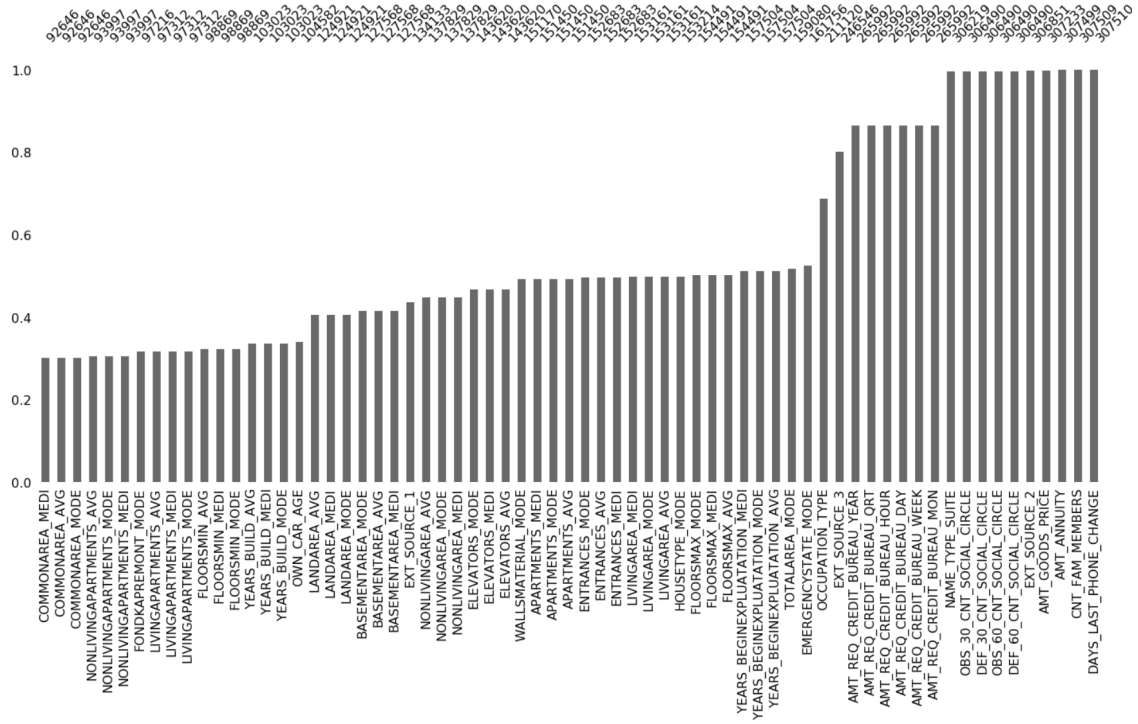

Figure 5 – Missing values barplot

The bar plot shows which columns have the biggest portion of missing values, which are mostly related to apartment/house. Certain algorithms require missing values to be filled in and some others can work with missing values. The decision what to do with missing values will be made after algorithm has been chosen.

There is certain amount of correlations exist in the data set (see Figure 5 below). On the below heatmap are shown only columns with correlations. There are columns describing apartment/house aspects and they are represented as _AVG, _MODE and _MEDI, which are average, modus and median. Correlation value is close or equal to 1 which is very strong correlation type and may considered as duplicates. These columns are candidate for removal e.g. using principal components analysis technique or even, just dropping the columns _MEDI and _MODE leaving only _AVG ones.

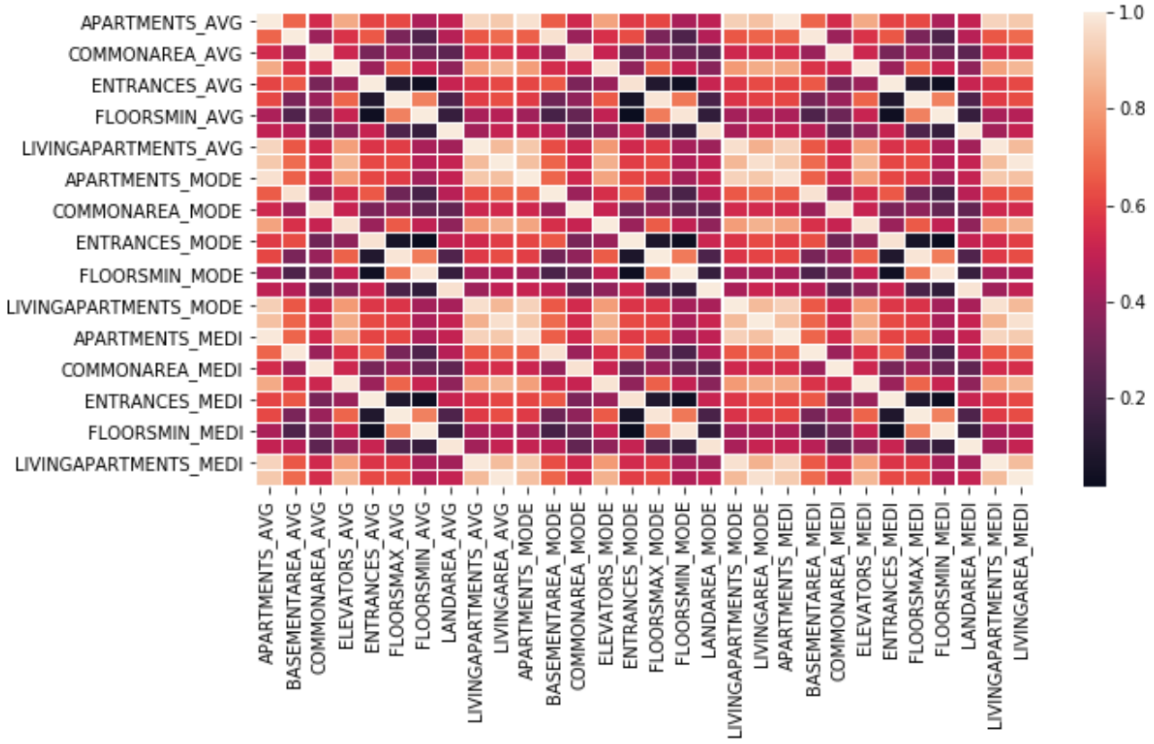

Figure 6 – Correlations

One more candidate for visualization is DAYS_EMPLOYED column, it's worth to look at hist plot with and without outliers:

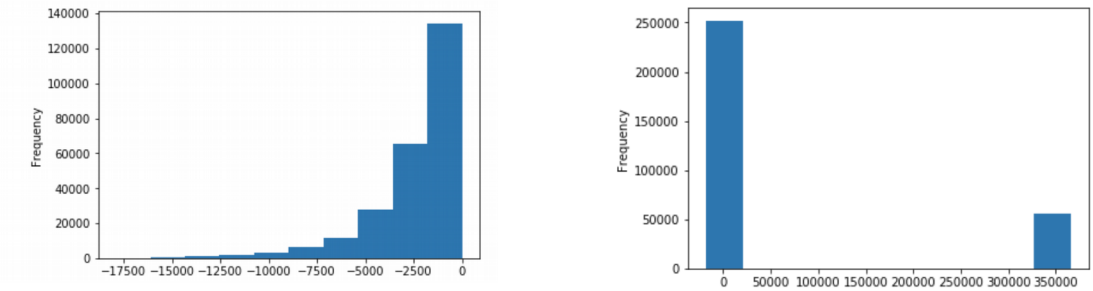

Figure 7 – Days employed column with and with out abnormalies

With out abnormal points distribution is more "gaussian", many algorithms make assumption that data distribution of numerical columns is "gaussian", which may affect model performance.

It's also useful to visualize outliers to see how far from other values they are, below is boxplot of OWN_CAR_AGE column:

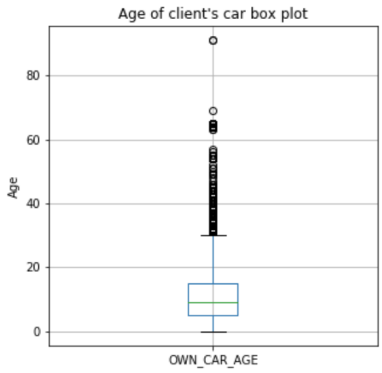

Figure 8 – Outliers visualization

From the plot it is possible to evaluate how many outliers are there outside of IQR.

There are algorithms sensitive to numerical features distribution, e.g. let's take a look at AMT_CREDIT:

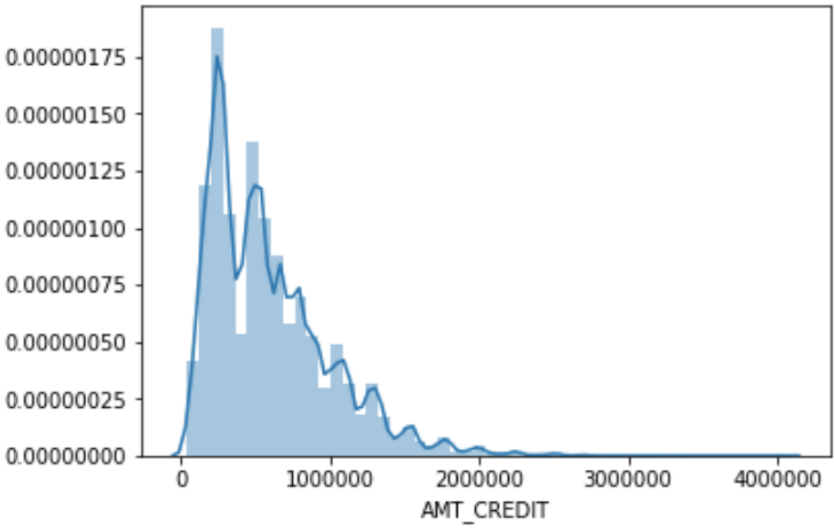

Figure 9 – AMT_CREDIT distribution

If an algorithm makes assumption that numerical features are distributed by Gaussian law, then it may perform better it is true. To make feature distribution more like Gaussian log transform should be applied to the feature:

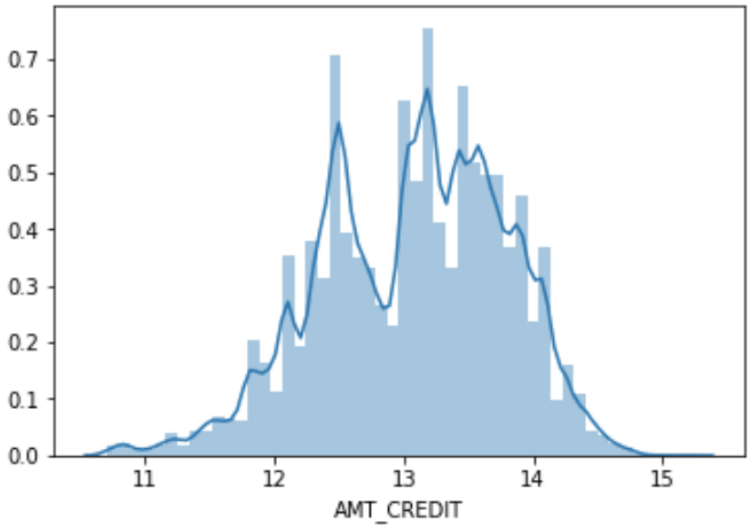

Figure 10 – AMT_CREDIT distribution after logarithm transformation

Algorithms and Techniques

There are a lot of algorithms for supervised ML classification tasks. Ensemble methods proved to work good on high dimensional datasets with mixed features (categorical and continuous). Gradient Boosting Decision Trees is one of them, LightGBM[10] is one of the implementations.

The data contains both numerical and categorical features, so boundary may be a complex and not easy to find function. In this case it may be more effective to find an ensemble of simple classifiers instead of a single complex model which tries to find that very complex decision boundary function. Boosting technique uses simple classifiers, called weak learners, which will do simple splits (most likely having high bias problem) changing data points importances. It adds new weak learner at sequentially improving loss function of final model, where the final model is a combination (ensemble) of weak learners:

$$\hat{y} = \sum_{m=1}^{M} f_m(x)$$

where y-hat is approximation of target function and f m-th is weak leaner.

The boosting strategy is greedy in the sense that choosing $f_m(\mathbf{x})$ never alters previous functions. We could choose to stop adding weak models when $\hat{y} = F_M(\mathbf{x})$'s performance is good enough or when $f_m(\mathbf{x})$ doesn't add anything.

Lightgbm is modern implementation of GBDT idea. It can natively handle categorical features and sparse datasets (see [10] section 4). It is very fast, it uses techniques Gradient-based One-Side Sampling and Exclusive Feature Bundling to deal with large number of data instances and large number of features respectively. It supports parallel execution out of the box [11]. Tree based models are interpretable [12], [13] which is strong side.

Continuous values handling is one of the weaknesses of tree based models, LGBM uses discretization technique which leads to information loss, it is not critical in most cases.

Another algorithm considered to test on the data was Neural Networks, Multi Layered Perceptron architecture [23]. Under the hood it is a non-linear composition of layers of functions. Where each layer has it's weights which are learned during learning process. This approach can find really complex hyper-plane boundary. The complexity of hyper-plane or, another words, complexity class of approximation function, depends on number of nodes and layers. It is very powerful approximation tool. However in case of this project, data might be not splittable by hyperplane. If imagine two data points they may be of different class just because of human factor which led to one guy to return loan and another guy do not return loan. Also Neural Networks require all data columns to be numerical and categorical features one-hot-encoded, and may perform bad if there are big number of categorical features.

Benchmark

A naive base line would be a random guess, taking into account we use ROC as a metric, random guess will give us a straight line because true positive rate and false positive rate will be the same. In addition to the straight line it'd be nice to add a simple model result to compare to, e.g. Logistic Regression [17].

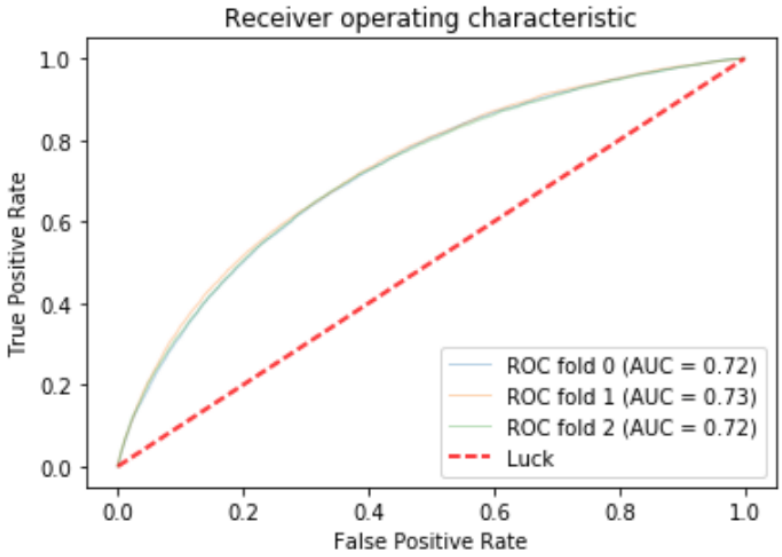

Figure 8 – Base Line

Logistic regression gives 0.72 AUC value, it is obtained based on the data, missing values are fulfilled with median imputation strategy. Red dash line corresponds to random guess result.

Methodology

Data Preprocessing

Another important thing is to use all provided data. Additional files provided represent applicants additional data, and relates as one-to-many to application train/test files rows. A simple strategy of merging these additional data would be to do a 'left join' [5], however relation is one-to-many does not allow us to do that without duplicating rows. A solution to that can be averaging values grouped per applicant for numerical values and taking most frequent values for categorical features, this is what is done with provided files. This can be considered as Feature Engineering and it's hard to predict how it will affect algorithm performance. For below code examples see Data_Merging.ipynb notebook.

Numerical feature aggregation example, load bureau data

```
bureau = pd.read_csv('input/bureau.csv.zip')
bureau_by_skid = bureau.groupby('SK_ID_CURR')
```

```
avg_bureau = bureau_by_skid.mean()
# merge bureau data and application data
app_train = app_train.merge(avg_bureau, how='left', on='SK_ID_CURR')
```

For categorical values it's possible to find most frequent value:

```
credit_card_balance = pd.read_csv('input/credit_card_balance.csv.zip')
max_status = credit_card_balance.groupby('SK_ID_CURR').agg(most_frequent)
credit_card_balance['MAX_STATUS'] = max_status['NAME_CONTRACT_STATUS']
app_train = app_train.merge(credit_card_balance, how='left', on='SK_ID_CURR')
```

This way all numerical and categorical features of additionally joined tables were processed. Additionally conflicting column name were renamed:

```
cols_rename = {'MONTHS_BALANCE': 'CC_MONTHS_BALANCE',
               'NUNIQUE_STATUS': 'CC_NUNIQUE_STATUS',
               'SK_DPD': 'CC_SK_DPD',
               'SK_DPD_DEF': 'CC_SK_DPD_DEF'}
credit_card_balance = credit_card_balance.rename(index=str, columns=cols_rename)
```

Few more new features added are counts, e.g. count of previous applications for each applicant, this gives additional information which otherwise would be lost during averaging and left join operations:

```
cnt_previous_app = previous_application.groupby('SK_ID_CURR').count()
avg_previous_app['NUM_APPS'] = cnt_previous_app['SK_ID_PREV']
```

LightGBM can natively process missing values [11] the same way as it handles sparse matrices [18], so there is no need to impute missing values and even do e.g. One-Hot-Encoding just specify categorical features column names. The thing was done translating `object` column types to `categorical`

```
cat_features = train.select_dtypes('object').columns.tolist()
for col in cat_features:
    train[col] = train[col].astype('category')
    test[col] = test[col].astype('category')
```

Redundant columns found on Data Exploration stage are removed – columns with _MEDI and _MODE suffixes, it reduces data dimensionality from 197 to 177 columns.

LightGBM is not sensitive to outliers because it uses histogram-based algorithms, which buckets continuous feature values into discrete bins.

Finally, preprocessed and merged data is stored in two files: train.csv and test.csv

```
app_train.to_csv('input/train.csv', index=False)
```

Implementation

Development is done in Jupyter Notebook [24], using python 3.x as main language. All the code related to Lightgbm is in the Lightgbm.ipynb notebook. To load data and process data Pandas library[25] is used:

```
train = pd.read_csv('input/train.csv')
print(train.shape)
Out: (307511, 199)
```

Train and test data are loaded the same way from train.csv and test.csv, see "Data Preprocessing" section.

Supervised learning algorithms require data to be split into **X** and **y**, where **X** is matrix, each row is a feature vector and **y** is a vector of scalar values. In our case train data should be split into **X** and **y**:

```
targets = train[['TARGET']]
train_ids = train['SK_ID_CURR']
train = train.drop(columns=['SK_ID_CURR', 'TARGET'])
test_ids = test['SK_ID_CURR']
test = test.drop(columns=['SK_ID_CURR'])
```

Additionally, service columns 'SK_ID_CURR' which is unique for each row are removed, because it does not help algorithm to identify which category that row belongs to. Now both train and test sets have the same number of columns 177:

```
print(train.shape)
print(test.shape)
Out: (307511, 177)
Out: (48744, 177)
```

Lightgbm algorithm nateively works with categorical features. To process those it needs a parameter 'categorical_feature' set. It’s easy to extract them:

```
cat_features = train.select_dtypes('object').columns.tolist()
```

The first run of Lightgbm algorithm was done on almost all default parameters:

```
base_parameters = {
    'learning_rate': 0.1,
    'n_estimators': 5000,
    'num_leaves': 31,
    'max_depth': -1,
    'min_child_samples': 20,
    'reg_alpha': 0.1,
    'reg_lambda': 0.1,
    'max_bin': 255
}
# set base parameters active
params = base_parameters
```

To evaluate how model works cross validation is used. K-Fold technique applied, it works the next way, train data data is split into k folds, then one of folds is chosen to be validation set others to be train set. Models are trained and evaluated with each fold, e.g. model1 trained on Fold1 + Fold2 + Fold3 + Fold4 and evaluated on Fold5. The AUC scores are collected for each model and summarized.

```
k_fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
# arrays to store validation and train score for each model
valid_scores = []
train_scores = []

for train_indices, valid_indices in k_fold.split(features, targets):
    model = lightgbm.LGBMClassifier(n_estimators=params['n_estimators'], objective='binary',
                                    class_weight='balanced', learning_rate=params['learning_rate'],
                                    num_leaves=params['num_leaves'], max_depth=params['max_depth'],
                                    reg_alpha=params['reg_alpha'], reg_lambda=params['reg_lambda'],
                                    min_child_sample=params['min_child_samples'],
                                    subsample=0.8, n_jobs=6, random_state=4242,
                                    max_bin=params['max_bin'])

    train_features, train_labels = train.iloc[train_indices], targets.iloc[train_indices]
    valid_features, valid_labels = train.iloc[valid_indices], targets.iloc[valid_indices]

    # The ‘balanced’ mode uses the values of y to automatically adjust weights inversely
    # proportional to class frequencies in the input data as
    # n_samples / (n_classes * np.bincount(y))
    model.fit(train_features, train_labels, eval_metric='auc', categorical_feature=cat_features,
                  eval_set=[(valid_features, valid_labels), (train_features, train_labels)],
                  eval_names=['valid', 'train'], early_stopping_rounds=100, verbose=100)
    valid_score = model.best_score_['valid']['auc']
    train_score = model.best_score_['train']['auc']
    valid_scores.append(valid_score)
    train_scores.append(train_score)
```

This gave the next plot, iterations on X axis and AUC score for validation and train sets on Y axis:

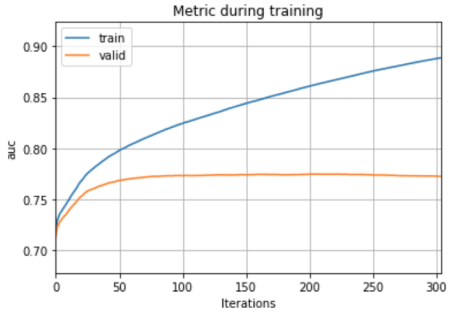

Figure 9 – Validation vs Train AUC metric

From the graphic it is obvious that model is overfitting, because validation score stops growing after 50 iterations while train score continues growing, which means that the model ‘remembers’ train data. Best iteration gives:

```
train's auc: 0.86201
valid's auc: 0.774841
```

Execution on test data gives:

```
test’s auc: 0769
```

LightGBM implementation has many parameters [19]. The most important are:

- num_leaves – main parameter to control complexity. Recommended to to less then 2^(max_depth) to prevent overfitting
- min_data_in_leaf – parameter to prevent overfitting, it’s value depends on train samples and num_leaves. Setting it larger prevents trees to grow too deep which may cau under-fitting
- max_depth – limiting tree depth, if there is not limit (-1) it may cause overfitting
- max_bin – max number of bins that feature values will be bucketed in
- l1 and l2 regularization

This is a GBDT algorithms tend to overfitting if it is not restricted on trees growth. There is a set of recommendation on the official documentation how to deal with overfitting through parameters few of them were applied:

- l1 and l2 regularization parameters
- max_depth restricted
- num_leaves decreased
- data set is big enough, it allows to train on small learning rate

Refinement

From data exploration there is a fact that this is imbalanced data set, so it is important to use stratified splits into training and validation folds. ‘Stratified’ here means that while splitting data set proportion on 0 and 1 will be preserved in training and validation sets. Below code example are from Lightgbm.ipynb notebook.

First result on application_train/test data sets was auc=0.746, parameters:

```
n_estimators=10000,
objective='binary',
class_weight='balanced',
learning_rate=0.0003,
num_leaves=31,
max_depth=-1,
reg_alpha=3,
reg_lambda=5
```

After merging other files to the application_train/test and on the same hyper parameters result improved to auc=0.767.

Parameters tuning is very important stage to find proper ones for the data set. Random search for that is proper tool in sklearn library. It is proved that random search may give better parameters then grid search [20]. Out of the box implementation did not work (by some reason kernel died), so custom implementation was done based on sklearn ParameterSampler and Kfold to split train and validation sets.

Parameters to try look like this:

```
params = {
    'learning_rate': [0.01, 0.1],
    'n_estimators': [8000, 16000],
    'num_leaves': [16, 32],
    'max_depth': [-1, 4, 7],
    'min_child_samples': [10, 20, 40],
    'reg_alpha': [0.1, 0.5, 1.0],
    'reg_lambda': [0.1, 0.5, 1.0],
    'min_data_in_leaf': [8, 16, 32],
    'max_bin': [128, 256]
}
```

This process of searching parameters has been run iteratively, at start big learning rate was chosen to understand which hyper parameters perform good and then learning rate was decreased to fine tune parameters. It ended up with the next parameters:

```
n_estimators=10000,
objective='binary',
class_weight='balanced',
learning_rate=0.0003,
min_child_samples=160
num_leaves=31,
max_depth=7,
reg_alpha=2.0,
reg_lambda=2.0
```

This improved results from auc=0.767 to acu=0.773 on the test set.

Results

Model Evaluation and Validation

There are three main parameters to improve accuracy:

```
n_estimators=10000,
learning_rate=0.0003,
num_leaves=31
```

Number of leaves controls complexity of each weak learner, the maximum number of weak learners is controlled by number of estimators. Making these two parameters big is a way to improve algorithm performance and the same time makes it easy to go into overfitting problem. Low learning rate helps to avoid overfitting by setting learning rate low algorithms learns slowly.

To control overfitting there is also set parameters. Min child sample and max depth control weak learner complexity and reg_alpha, reg_lambda are l1 and l2 regularization parameters.

```
min_child_samples=160
max_depth=7,
reg_alpha=2.0,
reg_lambda=2.0
```

Another important parameter is class weight. From the exploration we know the data set is imbalanced, to properly process this fact lightgbm allows to set the ‘balanced’, it mode uses the values of **y** to automatically adjust weights inversely proportional to class frequencies in the input data.

```
class_weight='balanced'
```

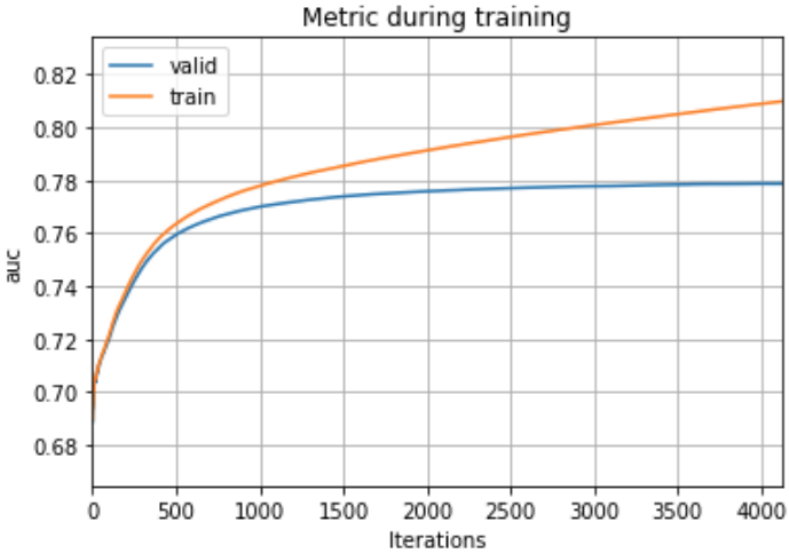

Figure 10 – Validation vs Train AUC metric

Parameter tuning helped to achieve a balance between high bias and high variance problems, see Figure 10 above. Validation and train data score are close enough to each other compared to initial model performance, see Figure 9.

To make sure algorithm is robust train and validation sets were used. Train data sets was split into Train and Validation using K-Fold technique. Train data set is to train algorithm parameters, validation data set is to find hyper-parameters. And finally test unseen data was used to evaluate algorithm performance. This approach makes sure algorithm is robust and has good generalization. Final scores are:

```
Train score: 0.81667
Validation score: 0.779629
Test score: 0.773
```

Test and validation scores have close values which means model has good enough generalization.

Justification

Base line solution result was auc=0.72. Final algorithm result is auc=0.773 which is not that big but still good enough improvement, taking into account that each additional tenth of improvement took more and more time finding more appropriate parameters and training time using small learning rate. And it is much better then random guess result auc=0.5.

Conclusion

Free-From Visualization

Lightgbm algorithm gives one more useful thing feature importance. After algorithm is run trained it's possible to plot it:

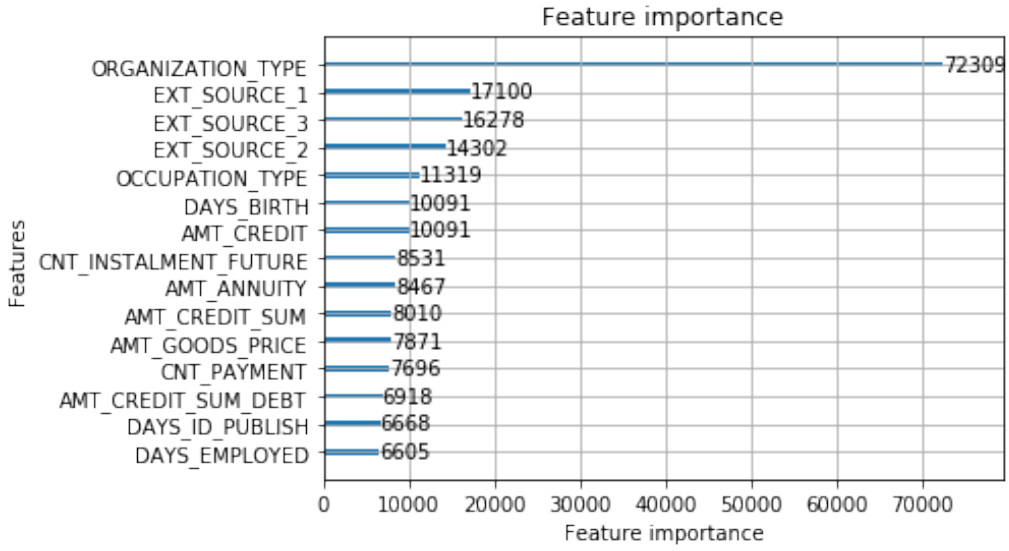

Figure 11 – Feature Importance

From the plot it is seen that organization type is the most important feature with much higher importance value then other features. And it makes sense, if an applicant has a good job, it implies good and stable revenue, then most likely an applicant will return loan back.

That fact organization type has much higher importance is called 'dominant feature' and it may cause that algorithm does not pay attention on other less important features. In this case it is needed to make weak learners complex enough to pay attention to minor features and same time find trade off so that model does not have high variance problem.

Reflection

This was a supervised classification problem with only two classes to predict. Data exploration and analysis is a very important part, it really depends on data what is possible to do, what algorithms to apply. Based on candidate algorithms it is needed to cleanup and preprocess data set. When data is ready the enjoyable part comes in – playing with chosen algorithm to perform good and data set, fine-tuning hyper parameters to achieve goals which in this case was good metric and generalization.

From technical perspective the complexity of this project was in provided data, it has a lot of issues to be addressed: missing values, a lot of additional data which do not relate as one-to-one to a row (applicant) and it is not clear if additional data helps or makes results worse. The most complex part was to find meaningful way how to combine data, create new feature, so that at least some information is retained. The most interesting part was working with model, parameters tuning, to find trade-off between high bias and high variance problems.

Improvement

There is a room to improve the project. Fine tuning parameters through random search is limited only by time, this means it's very hard to find extremely optimal parameters, it's highly probably that final parameters will be some local optimal.

Another way to improve performance is feature engineering. Using provided additional files with data it's possible to create new features which do not correlate with existing features, these new features is additional information which may improve performance.

Ensemble of models will most likely improve results as well. It's possible to combine results of heterogeneous models e.g. using Stacking [16]. This very time and resource consuming because requires training another algorithm and hyper-parameters tuning.

Resources

1. Investopedia – Loan (https://www.investopedia.com/terms/l/loan.asp)
2. Wikipedia - Credit Risk (https://en.wikipedia.org/wiki/Credit_risk)
3. Wikipedia - Credit Score (https://en.wikipedia.org/wiki/Credit_score_in_the_United_States)
4. Wikipedia - ROC curve (https://en.wikipedia.org/wiki/Receiver_operating_characteristic)
5. Wikipedia - LeftJoin (https://en.wikipedia.org/wiki/Join_(SQL))
6. Wikipedia – Anomaly (https://en.wikipedia.org/wiki/Anomaly_(natural_sciences))
7. Wikipedia – Tukey's fences (https://en.wikipedia.org/wiki/Outlier#Tukey's_fences)
8. Archiv.org – Imbalanced Class Problem (https://arxiv.org/pdf/1305.1707.pdf)
9. Wikipedia – Five Number summary (https://en.wikipedia.org/wiki/Five-number_summary)
10. LightGBM (https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf)
11. Lightgbm – Features (http://lightgbm.readthedocs.io/en/latest/Features.html)
12. GBDT interpretability (https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27)
13. LightGBM interpreter tool (https://github.com/slundberg/shap)
14. Embeddings (https://developers.google.com/machine-learning/crash-course/embeddings/categorical-input-data)
15. Hashing (https://alex.smola.org/papers/2009/Weinbergeretal09.pdf)
16. Wikipedia – Stacking Ensembel (https://en.wikipedia.org/wiki/Ensemble_learning#Stacking)
17. Wikipedia – Logistic Regression (https://en.wikipedia.org/wiki/Logistic_regression)
18. Lightgbm – Categorical Features Support (https://www.researchgate.net/publication/242580910_On_Grouping_for_Maximum_Homogeneity)
19. Lightgbm Parameters (http://lightgbm.readthedocs.io/en/latest/Parameters.html)
20. Random Search (http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf)
21. ROC Curve Explaine (https://www.dataschool.io/roc-curves-and-auc-explained/)
22. ROC (http://www.navan.name/roc/)
23. MLP (https://en.wikipedia.org/wiki/Multilayer_perceptron)
24. Jupyter Notebook (https://jupyter.org/)
25. Pandas (https://pandas.pydata.org/)