MLND Capstone Project: Predicting Home Loan Default

Lancelot

21 October, 2018

I. Definition

Project Overview

In this project we will build a model to predict the probability of default for a given home loan application. This is based on the Kaggle competition Home Credit Default Risk [1].

Home Credit is a global loan provider. Their vision is to provide a safe environment to give loans to people especially the unbanked population. This group of people has been unfairly treated by unscrupulous money lenders and Home Credit wants to change that.

However, due to the lack of credit scores of these unbanked population, Home Credit needs to rely on alternative data to help them predict their potential clients' repayment abilities. Traditional approach using credit scores are mainly out of scope and Home Credit is looking for innovative machine learning based models to help them predict the probability of an applicant defaulting his loan.

Problem Statement

The problem we are solving is a binary classification problem. We are given a set of data that is linked to an application, (examples include his income, age, education, his previous repayment history, his monthly snapshots of credit card loans, cash etc), and our task is to predict the probability that the applicant will not repay the loan.

We shall be approaching the problem with the following steps

- Exploratory data analysis - identify numerical vs categorical data, understand general trends, distribution of data, checking for outliers
- Data preprocessing - perform one hot encoding for categorical data, normalize the data

- Feature engineering - build new features by combining the existing features simply based on intuition
- Data preparation - split data into training vs validation set, resample the training data if necessary
- Model training & cross-validation - in this step we will train three models - Random Forest, XGBoost, LightGBM, and obtain their validation scores
- Hyperparameter tuning - based on the model with the best validation scores, we will optimize the parameters
- Check for model bias-variance - plot the learning curve to see if model overfit or underfit

Metrics

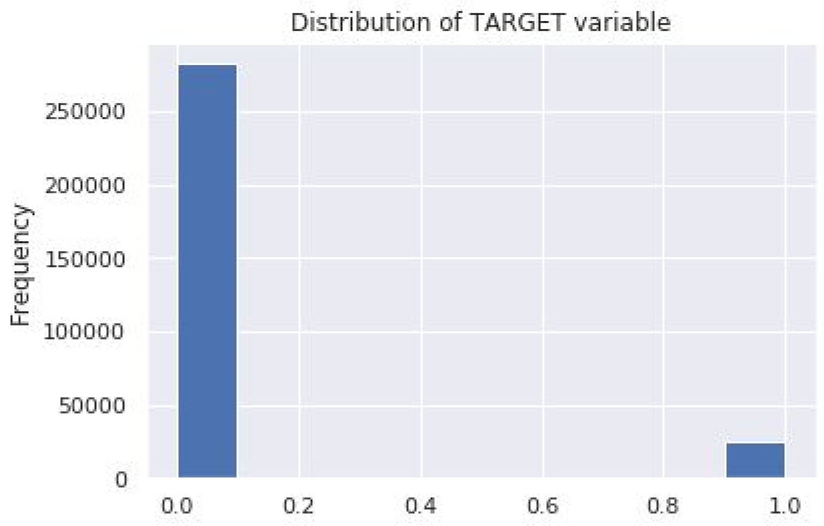
To measure the goodness of the model, we shall be using the metric Area Under Curve for Receiver Operating Characteristics (AUC-ROC). This metric combines the true positive rate and the true negative rate into a single figure, which would be much better than say “accuracy”.

II. Analysis

Data Exploration

The dataset we will be using is a table consisting of 307511 rows and 121 columns including the TARGET variable. This is a small to mid-size table hence does not required any big data techniques or nor we do need to run it on GPU.

The variable we have to predict is the TARGET variable. It can be either 1 or 0, where 1 indicates that the client has not repaid the loan. Looking at the distribution of TARGET, we see an imbalanced dataset.



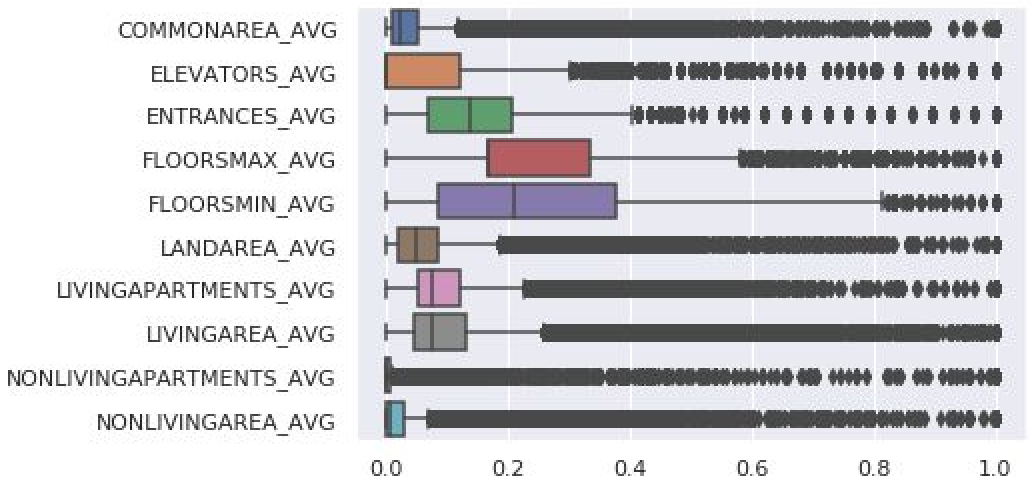
For an imbalanced dataset, accuracy is not a good measure. We have chosen another metric AUC-ROC explained in earlier section. The other consideration we have is that we will apply techniques to prepare the training dataset to get a more balanced dataset

We next look into the data types. Out of 120 columns (TARGET variable excluded)

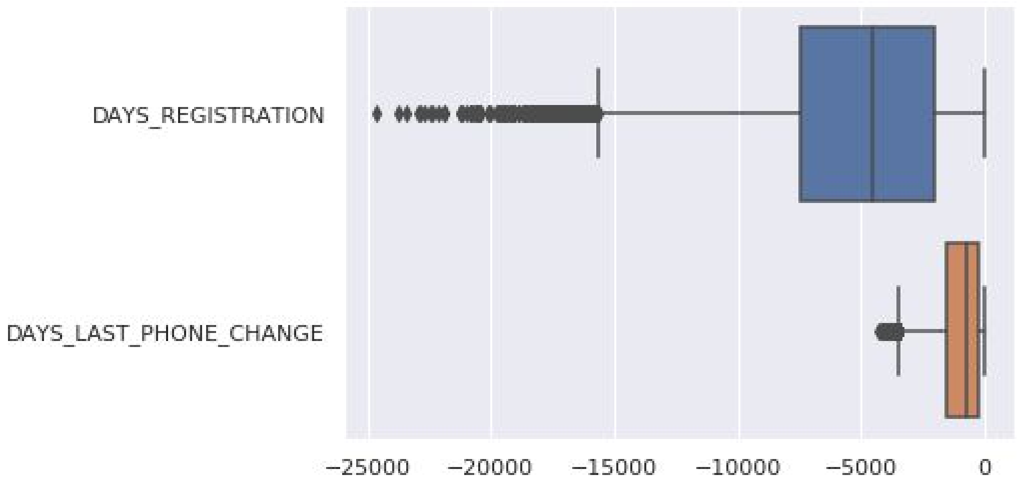
- 65 are floats
- 39 are integers
- 16 are objects

Exploratory Visualization

Among the 65 float variables, 38 of them have more than 50% missing values and 16 of them have more than 60% missing values. In terms of distribution, there is a wide range of values. Some values are distributed between 0 and 1 as shown just below



And some data are purely negative as shown here. These are indicators that the data needs to be standardized first before feeding into a model



As for the integer variables, there is no missing data. However we noted that most of them are binary values - most of them FLAG variables as indicated from their names. In fact only the following variables have more than 3 different unique values - CNT_CHILDREN, DAYS_BIRTH, DAYS_EMPLOYED, HOUR_APPR_PROCESS_START - these variables will be treated as numerical while the rest categorical.

Finally for the object variables, only 5 variables have missing values and the percentage of missing values range from 31% to 68%. For this project we simply set the missing values to a constant.

Algorithms and Techniques

With regards data preprocessing techniques, we will normalize using standard scaling, convert to categorical data using one-hot-encoding, use SMOTE resampling to address the issue of imbalanced dataset. Here we give a short summary of these techniques

- Standard scaling - features values are subtracted by their mean and divided by their variance. This is possible because by law of large numbers, these values follow normal distribution, and so we scaled them so that the features all follow the same standard normal distribution (more or less)
- One hot encoding - this technique is applied for categorical variables to convert them into binary variables. Say we have a field TYPE which can take the values of HOUSING_LOAN or CAR_LOAN. To do one hot encoding we create two features TYPE_HOUSING_LOAN and TYPE_CAR_LOAN which individually can take either 1 or 0 depending on its loan type. This step is normally required as most libraries take only numerical values
- SMOTE stands for Synthetic Minority Oversampling TEchnique. The idea is to focus on the minority cluster and generate additional samples which are close enough in the feature space to those already in the minority cluster. Essentially what we do here is to synthesize more data that look just like those in the minority class [2]

With regards to modeling, we will use Random Forest, XGBoost, LightGBM. Here we give a short summary of these models

- Random Forest is a collection of decision trees. Each decision tree is supposed to predict its own outcome and a vote counting is done on these results. The majority of the votes is then considered the final prediction. Intuitively this works on the basis that some decision trees make good decisions some make bad ones but if we have many trees built based on different features, we could trust the majority of the trees [3]
- XGBoost is similar to Random Forest in that it is an ensemble method. It builds a collection of decision trees as well. The difference is that XGBoost builds trees sequentially and it learns the errors from the previous tree before constructing the new trees [4]
- LightGBM is similar to XGBoost in that it also implements boosting method. However it claims to be more efficient by subsambling the data points which are proved to be more useful, and feature bundling amongst other new approaches [5]

Benchmark

For benchmark model we will predict 0 for everything. This gives us the AUC-ROC score of 0.5

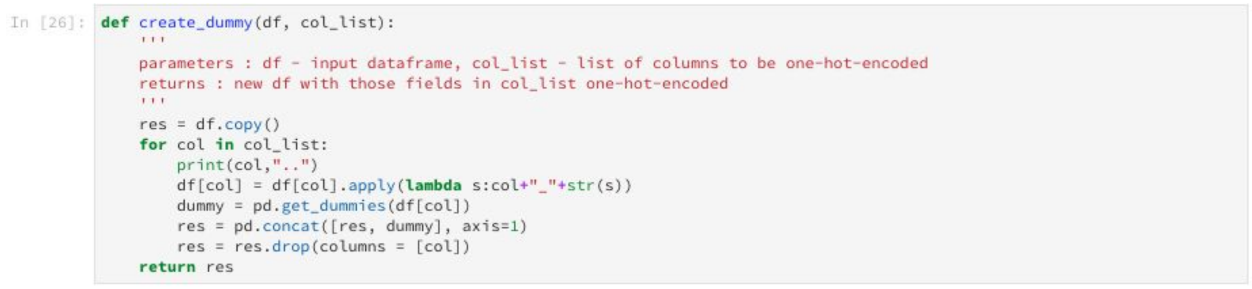
III. Methodology

Data Preprocessing

For data preprocessing, we will be doing the following - data imputation, one hot encoding of categorical variables, normalization of numerical data, splitting of data into training and validation set, oversampling of minority data using SMOTE.

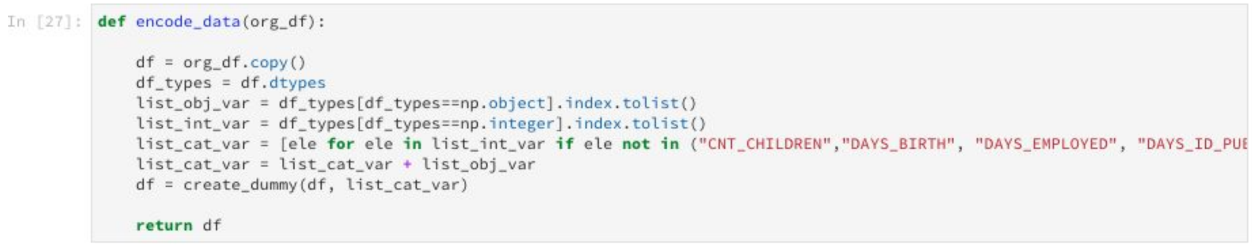
For data imputation we simply fill all NaN with 0.

For the one hot encoding process, we first create a function that takes a dataframe and a list of categorical variables. The function returns a dataframe where the original categorical variables are replaced with new dummy variables. We perform some renaming here so as to create unique dummy variables.



```
In [26]: def create_dummy(df, col_list):
             '''
             parameters : df - input dataframe, col_list - list of columns to be one-hot-encoded
             returns : new df with those fields in col_list one-hot-encoded
             '''
             res = df.copy()
             for col in col_list:
                 print(col,"..")
                 df[col] = df[col].apply(lambda s:col+"_"+str(s))
                 dummy = pd.get_dummies(df[col])
                 res = pd.concat([res, dummy], axis=1)
                 res = res.drop(columns = [col])
             return res
```

Then we need to create another function that decides what columns are considered categorical. All the columns with object variables are considered categorical. In addition, amongst the integer variables, all except a few are in fact FLAG variables which would be considered categorical as well. This was explained in Section II earlier.

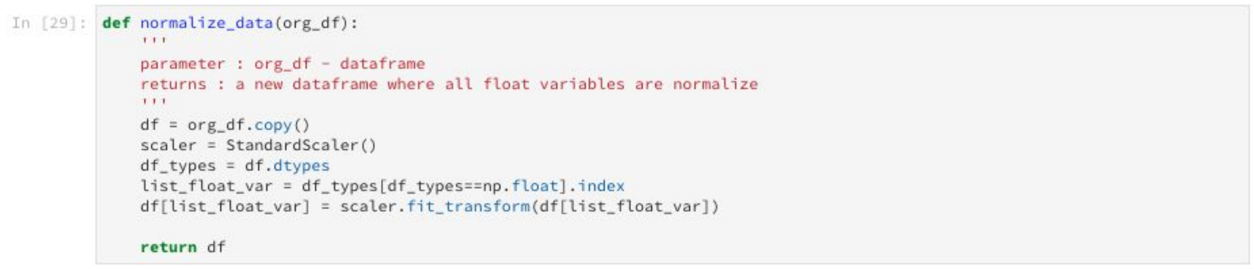


```
In [27]: def encode_data(org_df):

             df = org_df.copy()
             df_types = df.dtypes
             list_obj_var = df_types[df_types==np.object].index.tolist()
             list_int_var = df_types[df_types==np.integer].index.tolist()
             list_cat_var = [ele for ele in list_int_var if ele not in ("CNT_CHILDREN","DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_ID_PUE
             list_cat_var = list_cat_var + list_obj_var
             df = create_dummy(df, list_cat_var)

             return df
```

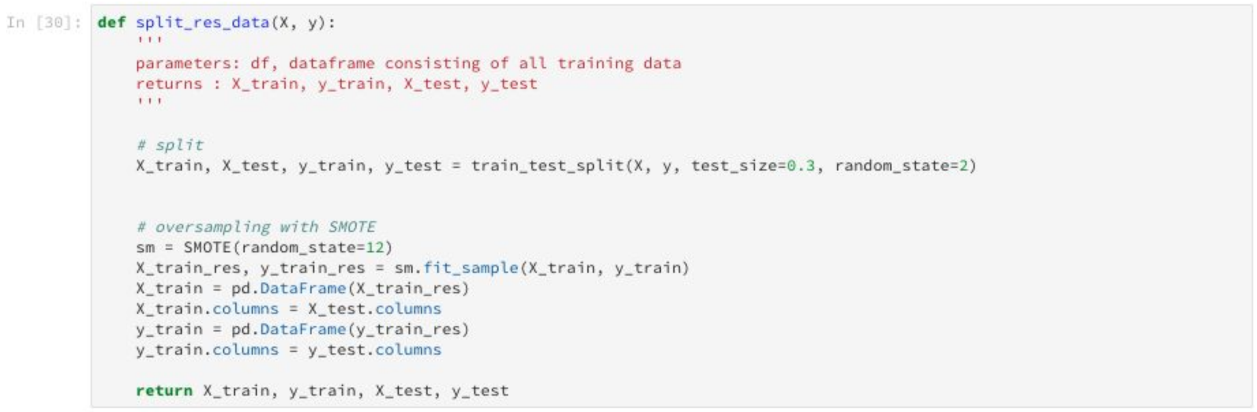
For normalization of float variables, we use the StandardScaler class from the sklearn.preprocessing library, and created a function as follow



```
In [29]: def normalize_data(org_df):
             '''
             parameter : org_df - dataframe
             returns : a new dataframe where all float variables are normalize
             '''
             df = org_df.copy()
             scaler = StandardScaler()
             df_types = df.dtypes
             list_float_var = df_types[df_types==np.float].index
             df[list_float_var] = scaler.fit_transform(df[list_float_var])

             return df
```

We split the data set into 70% training and 30% validation. Using SMOTE class from imblearn library, we apply the resampling on the training set. Here is the function that does exact those steps



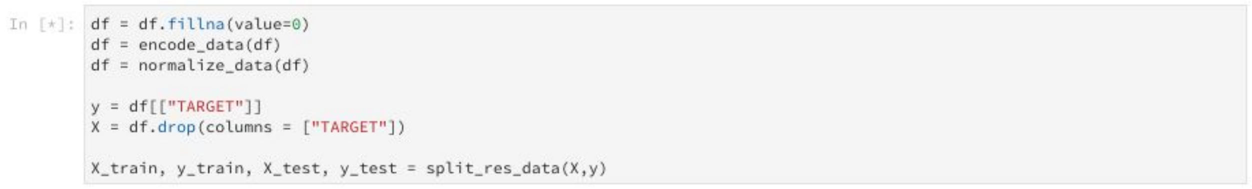
```
In [30]: def split_res_data(X, y):
             '''
             parameters: df, dataframe consisting of all training data
             returns : X_train, y_train, X_test, y_test
             '''

             # split
             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)


             # oversampling with SMOTE
             sm = SMOTE(random_state=12)
             X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
             X_train = pd.DataFrame(X_train_res)
             X_train.columns = X_test.columns
             y_train = pd.DataFrame(y_train_res)
             y_train.columns = y_test.columns

             return X_train, y_train, X_test, y_test
```

We put together all the data preprocessing steps to finally obtain the training sets and validation sets for feature columns and target column, and getting ready for the model fitting stage



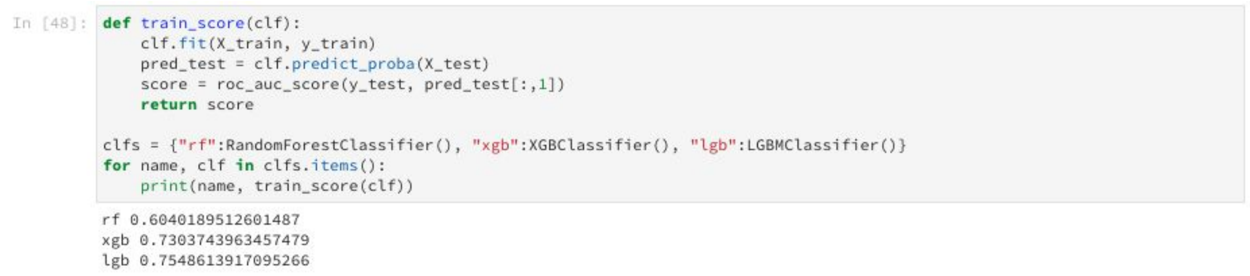
```
In [*]: df = df.fillna(value=0)
        df = encode_data(df)
        df = normalize_data(df)

        y = df[["TARGET"]]
        X = df.drop(columns = ["TARGET"])

        X_train, y_train, X_test, y_test = split_res_data(X,y)
```

Implementation

We use the following three models - Random Forest, XGBoost, LightGBM. The implementations we chose were from sklearn.ensemble [6], xgboost [7] and Microsoft/LightGBM [8].   Default settings were used and the AUC_ROC score for each model calculated using the validation set is as follow. The best performer is LightGBM with a score of AUC_ROC score of 0.75486



```
In [48]: def train_score(clf):
             clf.fit(X_train, y_train)
             pred_test = clf.predict_proba(X_test)
             score = roc_auc_score(y_test, pred_test[:,1])
             return score

         clfs = {"rf":RandomForestClassifier(), "xgb":XGBClassifier(), "lgb":LGBMClassifier()}
         for name, clf in clfs.items():
             print(name, train_score(clf))
```

```
rf 0.6040189512601487
xgb 0.7303743963457479
lgb 0.7548613917095266
```

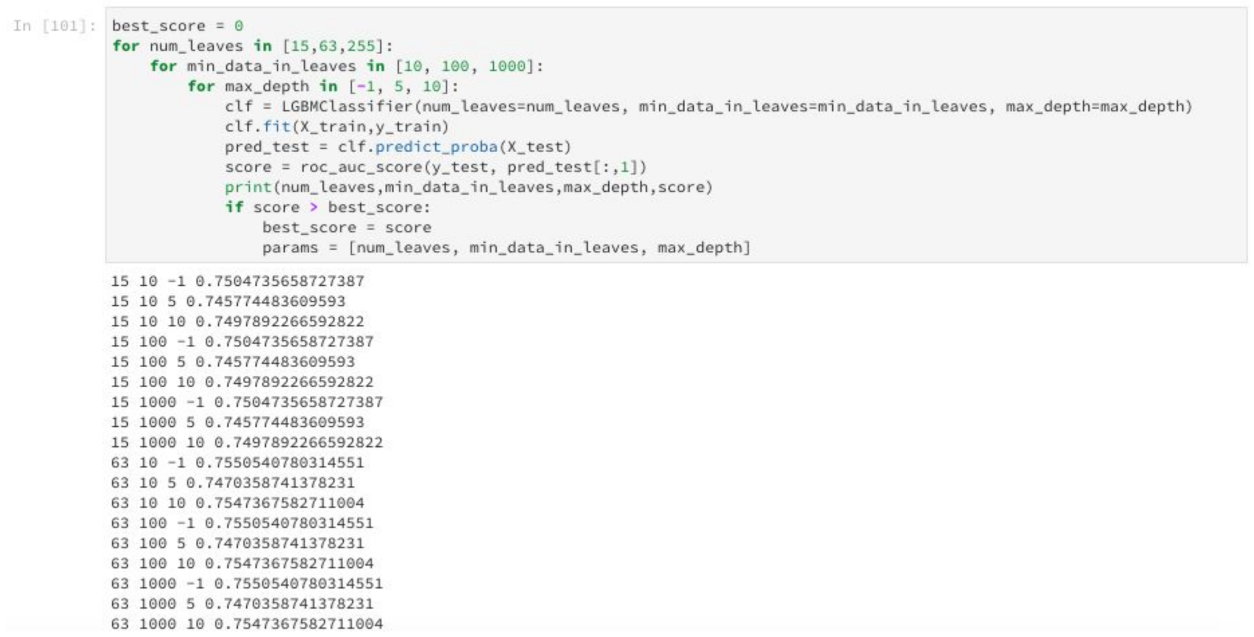
Refinement

Based on the the fact that LightGBM has the best performance using the default settings, we will next move to tune the hyperparameters for LightGBM.

We proceed to tune the parameters num_leaves, min_data_in_leaves and max_depth according to the suggestions outlined in the official documentation of LightGBM [9]. We perform a simple grid search across the following parameter space

- Num_leaves - 15, 63, 255
- Min_data_in_leaves - 10, 100, 1000
- Max_depth : 5, 10, unrestricted

The best parameters would be num_leaves = 63, min_data_in_leaves = 10, and unrestricted max_depth. And the resulting AUC_ROC is 0.755 which is slightly better than the model with default settings. The code to perform the search is as follow



```
In [101]: best_score = 0
          for num_leaves in [15,63,255]:
              for min_data_in_leaves in [10, 100, 1000]:
                  for max_depth in [-1, 5, 10]:
                      clf = LGBMClassifier(num_leaves=num_leaves, min_data_in_leaves=min_data_in_leaves, max_depth=max_depth)
                      clf.fit(X_train,y_train)
                      pred_test = clf.predict_proba(X_test)
                      score = roc_auc_score(y_test, pred_test[:,1])
                      print(num_leaves,min_data_in_leaves,max_depth,score)
                      if score > best_score:
                          best_score = score
                          params = [num_leaves, min_data_in_leaves, max_depth]
```

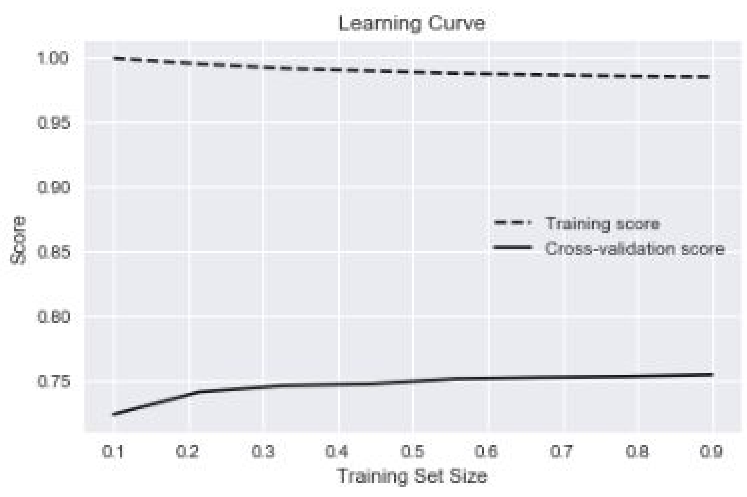
```
15 10 -1 0.7504735658727387
15 10 5 0.745774483609593
15 10 10 0.7497892266592822
15 100 -1 0.7504735658727387
15 100 5 0.745774483609593
15 100 10 0.7497892266592822
15 1000 -1 0.7504735658727387
15 1000 5 0.745774483609593
15 1000 10 0.7497892266592822
63 10 -1 0.7550540780314551
63 10 5 0.7470358741378231
63 10 10 0.7547367582711004
63 100 -1 0.7550540780314551
63 100 5 0.7470358741378231
63 100 10 0.7547367582711004
63 1000 -1 0.7550540780314551
63 1000 5 0.7470358741378231
63 1000 10 0.7547367582711004
```

The reason why we implemented the Grid Search instead of using standard library is that it is not clear how to ensure that the synthetic data generated in the training set does not get leaked into the validation set if we use the standard library available.

IV. Results

Model Evaluation and Validation

We started with three models, then selected the best one based on default settings. We then fine tuned the hyperparameters to get our final model. Now we want to see how robust our model is, and whether it is overfitting or underfitting. To do that we plot the learning curve



The model learns very well on the training data as the training score shows a consistent high score of well beyond 0.95. However the validation score is barely around 0.75 and it is rather insensitive to the training data set. This is indicative that the model might be overfitting the training data to some extent.

Justification

The AUC_ROC of our model is 0.755 and this is better than our benchmark model with the score of 0.5, thus making our model more useful than a simple model.

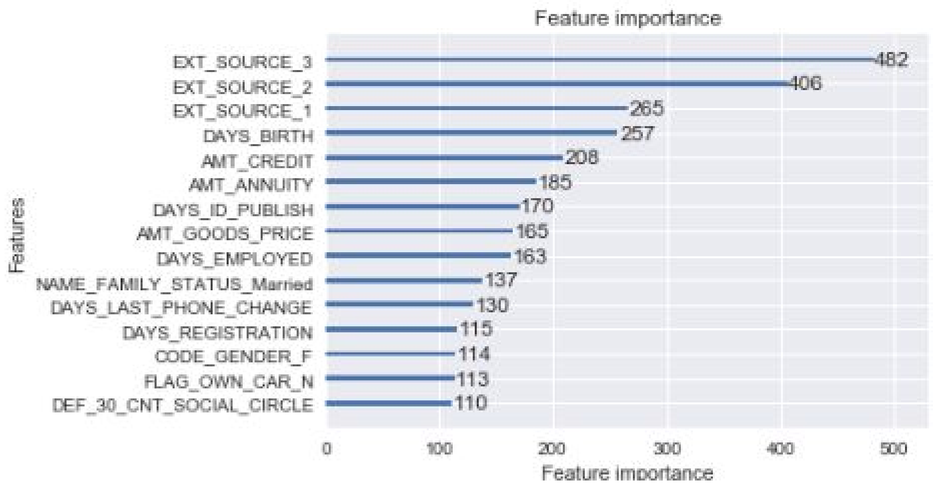
V. Conclusion

Free-Form Visualization

Here we visualize the feature importance based on the top features. This gives us a way to interpret and explain the model. It is also a form of validation. As shown below, the following are the top features

- EXT_SOURCES - this could be external funding or collateral, hence it makes sense that these are indicators for probability of default
- DAYS_BIRTH - this could be age
- AMT_CREDIT - larger amount might mean higher chance of default possibly?

- AMT_ANNUITY - this could an indicator of financial status hence a predictor of default



Reflection

The capstone is really the highlight of the entire program, where I had to a chance to carry out an entire pipeline of machine learning project from data preprocessing to model tuning and finally model validation. While the results of the final model is nothing state-of-the-art, I have developed a sense of what could go wrong and what potential improvements are available. During the process I have picked up debugging skills, and also practical skills on reading documentation and installing new libraries. Overall, this has laid a good foundation and a good framework for me to further strengthen my skills in applying machine learning to solve business problems.

Improvement

There are several areas that we can improve. First is feature engineering. In this project we had not done feature engineering. We could have done this via manual feature engineering or stack a neural network model to learn these features. Second is explore models with lower complexity to address the issue of overfitting.

References

1. https://www.kaggle.com/c/home-credit-default-risk
2. https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets
3. https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

4. https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
5. http://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/
6. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
7. https://xgboost.readthedocs.io/en/latest/#
8. https://lightgbm.readthedocs.io/en/latest/
9. https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html