Alessia Taboga

February 2019

Udacity Machine Learning Engineer Nanodegree
CAPSTONE PROJECT
PREDICTING LOAN DEFAULT FOR LENDING CLUB

1. Definition

1.1 Project Overview

One well known application of machine learning algorithms is predicting loan default [1, 2, 3, 4]. For this project, I built and compared several supervised machine learning models to predict bad loans for the Lending Club company (data available on Kaggle website [5]).

Lending Club is an American peer-to-peer lending company, which connects borrowers with investors online [6,7]. Potential borrowers apply for loans by providing personal and financial information and details about the loans they would like to obtain. Lending club evaluates the applicant's credit score, credit history, requested loan amount, debt-to-income ratio and decides whether or not to approve the loan. If it is approved, the Lending Club assigns to it a credit grade, which establishes interest rate and fees.

Investors can search through the list of loans, having access to some of the borrower's details as well as the loan amounts, grades and purposes. They can than decide if and how much to fund each borrower, hoping to be repaid with interest. If the loan is not repaid, the investors will lose money.

Although decisions can be made on the basis of some statistical methods for the more obvious loan acceptance/reject cases, decisions may become difficult in less clear cases. Denying the loan in all these uncertain situations would prevent losses for the investors, but reduce the number of good and active customers for the Lending Club. Lending Club makes money by charging fees to both borrowers and investors.

Data science can help. Available data of past loans (data with details of borrowers and information saying if they were able to repay or not their loans) can be used to build machine learning models capable of predicting if the borrowers will default or not. The model can also provide a structure so that reasons behind a denied loan can be explained to the borrower if necessary.

1.2 Problem Statement

The aim of the project was to build a supervised machine learning model able to predict which borrowers will not pay back their loans, using available data from past loans.

In order to achieve this goal, the following tasks were carried out:

1. Lending Club data were downloaded from Kaggle website [5].
2. All the features in the data were analysed in order to understand their meaning, their distribution, their relationship with the target (good/bad loans), their correlations, their type and the potential presence of missing values and outliers. Some of the original features were dropped as they would not have added value to or would have reduced the performance of the machine learning algorithm.
3. Data were pre-processed to deal with outliers, missing values, categorical variables and necessary transformations.
4. Data were split into training and testing datasets (0.80% and 0.20% respectively).
5. Data were standardised.
6. Four different initial classification models were built (Gaussian Naïve Bayse, Logistic Regression, Random Forest Classifier, AdaBoost).
7. The initial models were evaluated and compared.
8. Tests to improve the model performances were carried out (changing/tuning hyperparameters or trying to balance the provided data).
9. Results of the various models were again evaluated and compared.
10. The most important features for the models were determined.

An optimal model is expected to perform well in recalling the borrowers that will not pay back their loans (we want a high proportion of the actual bad loans to be classified as bad loans), as they cause the main loss for the investors. At the same time, the model should still have an acceptable level of precision (we do not want many of the actual good loans to be classified as bad loans), as Lending Club will not want to deny loans to many good and active customers.

The model should provide also a simplified structure to explain to rejected borrowers why they have been denied a loan.

1.3 Metrics

For the scope of the project, the model should be able to predict loans accurately. Accuracy is indeed a metric commonly reported to compare and assess classification algorithms. It measures how often the classifier makes the correct prediction, being the ratio between the number of correct predictions to the total number of predictions:

$$accuracy = \frac{TruePositives + TrueNegatives}{TotalNumberPredictions}$$

where:
True Positives (TP) are the borrowers that were not able to pay back (bad loan) and were correctly predicted as not paying back (bad loan);

True Negatives (TN) are the borrowers that were able to pay back (good loan) and were correctly predicted as paying back (good loan).

However, accuracy is not a very good metric by itself for this project. The Lending Club data downloaded from Kaggle website are highly skewed in their classification distribution. The percentage of borrowers with bad loans is only 7.6%. The model could miss-classify all or most of the bad loans and have a very high accuracy score.

Therefore, it is important to look also at other metrics such as Precision and Recall.

Precision tells what proportion of the borrowers classified as not able to pay back (bad loans) were actually not able to pay back.

$$precision = TruePositives/(TruePositives + FalsePositives)$$

where:
False Positives (FP) are the borrowers that were able to pay back (good loan) but have being classified as not able (bad loan)

Recall tells what proportion of borrowers that were not actually able to pay back (bad loans) were classified by the model has being not able to pay back.

$$recall = TruePositves/(TruePositives + FalseNegatives)$$

where:
False Negatives (FN) are the borrowers who have being classified as paying back (good loan) when in reality they did not (bad loan).

As already mentioned, recall (or sensitivity) will be more important than precision for this project. The ideal model should be able to identify correctly all the bad loans, as they cause more economic loss for the investors compared to the reduction in number of good customers for the Lending Club. However, precision cannot become so low that Lending Club will wrongly deny loans to too many good and active customers.

When running the project in Python, I used scikit learn metrics library to print confusion matrix, classification report and also accuracy and precision scores [8-12]. The classification report includes also F1 (which is a harmonic mean of precision and recall and therefore takes both equally into account).

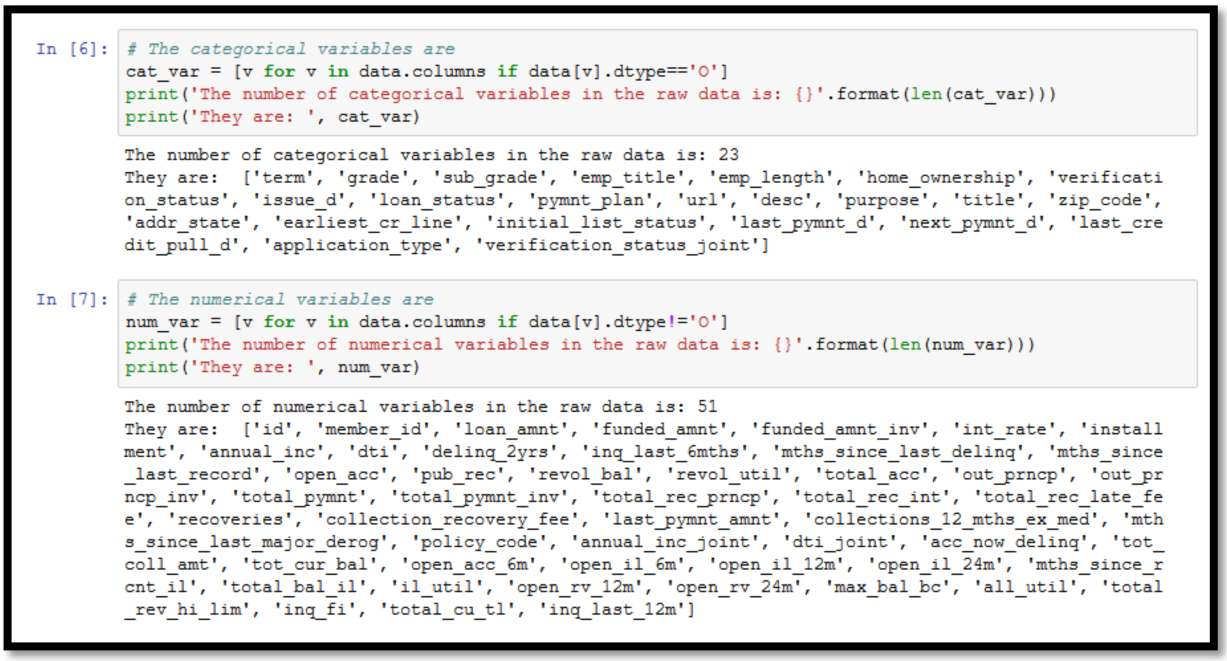
These are also the metrics which are reported in several Kaggle kernels which I can use as comparison for the performances of my models.

1.4 Data Exploration

The Lending Club data used for the project were downloaded from Kaggle website [5]. The ‘loan.csv’ file contains data for all loans issued through 2007-2015, including information about loan status and latest payment. A separate file, ‘LCDataDictionary.xlsx’, provides description of all variables.

The input data have 887379 records and 74 variables. 5 of the features described in the dictionary are in reality not present in the provided data ('fico_range_high', 'fico_range_low', 'is_inc_v', 'last_fico_range_high', 'last_fico_range_low').

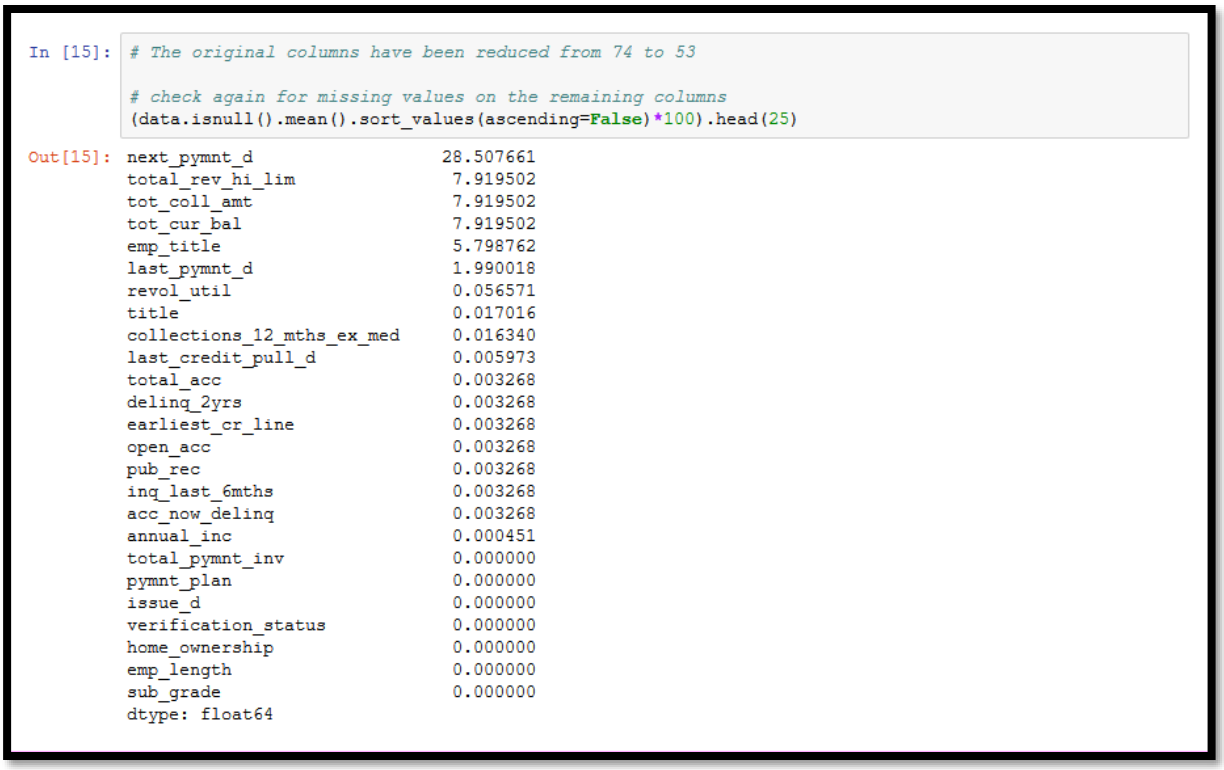
An initial data exploration revealed that 23 of these variables are categorical, while the other 51 are numerical. Categorical variables need to be changed into numerical for machine learning algorithms (for example with hot-encoding, mapping or feature engineering). Some of the categorical features (‘issue_d’, ‘earliest_cr_line’, ‘last_pymnt_d’, ‘next_pymnt_day’, ‘last_credit_pull’) are dates and Pandas datetime can be used to change them to numerical.



```
In [6]: # The categorical variables are
        cat_var = [v for v in data.columns if data[v].dtype=='O']
        print('The number of categorical variables in the raw data is: {}'.format(len(cat_var)))
        print('They are: ', cat_var)

        The number of categorical variables in the raw data is: 23
        They are:  ['term', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'verificati
        on_status', 'issue_d', 'loan_status', 'pymnt_plan', 'url', 'desc', 'purpose', 'title', 'zip_code',
        'addr_state', 'earliest_cr_line', 'initial_list_status', 'last_pymnt_d', 'next_pymnt_d', 'last_cre
        dit_pull_d', 'application_type', 'verification_status_joint']

In [7]: # The numerical variables are
        num_var = [v for v in data.columns if data[v].dtype!='O']
        print('The number of numerical variables in the raw data is: {}'.format(len(num_var)))
        print('They are: ', num_var)

        The number of numerical variables in the raw data is: 51
        They are:  ['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'int_rate', 'install
        ment', 'annual_inc', 'dti', 'delinq_2yrs', 'inq_last_6mths', 'mths_since_last_delinq', 'mths_since
        _last_record', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'out_prncp', 'out_pr
        ncp_inv', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_late_fe
        e', 'recoveries', 'collection_recovery_fee', 'last_pymnt_amnt', 'collections_12_mths_ex_med', 'mth
        s_since_last_major_derog', 'policy_code', 'annual_inc_joint', 'dti_joint', 'acc_now_delinq', 'tot_
        coll_amt', 'tot_cur_bal', 'open_acc_6m', 'open_il_6m', 'open_il_12m', 'open_il_24m', 'mths_since_r
        cnt_il', 'total_bal_il', 'il_util', 'open_rv_12m', 'open_rv_24m', 'max_bal_bc', 'all_util', 'total
        _rev_hi_lim', 'inq_fi', 'total_cu_tl', 'inq_last_12m']
```

Example code from my ipython notebook

There are several features with missing values in the data. For 21 of them, the percentage of missing values is above 50%. I therefore decided to drop these columns at an early stage (original features were so reduced from 74 to 53). Missing values needed to be dealt with before running the machine learning models. Data exploration/visualisation was carried out also to identify the best solution for dealing with the missing values (for example if filling them with mode/median, or discarding the entire samples). After dropping the columns with more than 50% of missing values, the remaining features with missing values, expressed in percentage, were the first 21 on the list below:



```
In [15]: # The original columns have been reduced from 74 to 53

         # check again for missing values on the remaining columns
         (data.isnull().mean().sort_values(ascending=False)*100).head(25)

Out[15]: next_pymnt_d                  28.507661
         total_rev_hi_lim               7.919502
         tot_coll_amt                   7.919502
         tot_cur_bal                    7.919502
         emp_title                      5.798762
         last_pymnt_d                   1.990018
         revol_util                     0.056571
         title                          0.017016
         collections_12_mths_ex_med     0.016340
         last_credit_pull_d             0.005973
         total_acc                      0.003268
         delinq_2yrs                    0.003268
         earliest_cr_line               0.003268
         open_acc                       0.003268
         pub_rec                        0.003268
         inq_last_6mths                 0.003268
         acc_now_delinq                 0.003268
         annual_inc                     0.000451
         total_pymnt_inv                0.000000
         pymnt_plan                     0.000000
         issue_d                        0.000000
         verification_status            0.000000
         home_ownership                 0.000000
         emp_length                     0.000000
         sub_grade                      0.000000
         dtype: float64
```

Example code from my ipython notebook

Three features (‘id’, ‘member_id’, ‘url’) are characterised by unique identification values different for each customer. These features were dropped as they would not add any value to the model.

The numerical feature ‘policy_code’ is instead constant (always equal to 1). The feature ‘pymnt_plan’ (with two categoriese ‘y’ and ‘n’) is almost constant for all the customers, as there are overall only 10 customers with category ‘y’. The feature ‘total_rec_late_fee’ had a value different than 0 only for 1.4% of the data. These three features do not hold useful information for the model and therefore were dropped.

The numerical features ‘recoveries’ and ‘collection_recovery_fee’ had values of 0 for most of the customers (values different than 0 for less the 3% of the customers). When analysing the distribution of the non zero customers versus the target, I noticed that they were all in the bad loan category. These two features define in a way the target because only customers with charge off status will have values different than zero. Therefore I dropped also these two features.

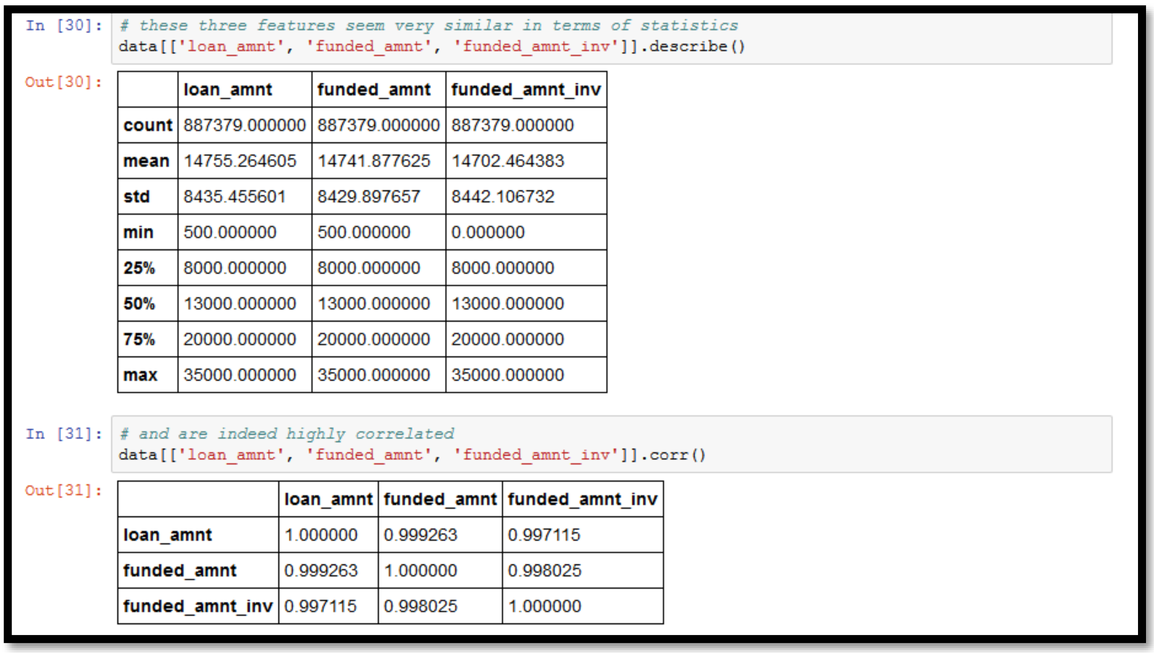
The features ‘emp_title’, ‘title’ and ‘zip’ code have too many categories and were dropped as they could cause decrease of model performance. The feature ‘title’ documents the reason for asking the loan as provided by the customer, but there is already a feature ‘purpose’ which is more useful (lower number of unique values and clearly defined loan categories). The information provided in ‘zip’ feature (935 categories) is already underlying in the ‘addr_state’ feature which has less categories, 51. For the feature ‘addr_state’, only some exploratory plots were made. Then the feature was dropped, as influenced by population density and more important for geographical insights behind the scope of this version of my project.

The feature 'sub-grade' provides only more specifics of the categories present in 'grade' and therefore it was dropped.

The categorical feature 'home_ownership' has some rare labels and the model may benefit from grouping them together.

Plots and statistical analysis showed presence of outliers in several features ('annual_inc', 'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'total_pymnt', 'total_pymnt_inv', 'total_rec_int'). These needed to be addressed before running the machine learning algorithms.

Some features seemed very similar/highly correlated when looking at plots and statistics (ex. 'loan_amnt' vs 'funded_amnt' and 'funded_amnt_inv'; 'out_prncp' vs 'out_prncp_inv', 'total_pymn' vs 'total_pymnt_inv). These similarities needed to be checked again at pre-processing stage to avoid multi-collinearity.



```
In [30]: # these three features seem very similar in terms of statistics
         data[['loan_amnt', 'funded_amnt', 'funded_amnt_inv']].describe()
```

Out[30]:

	loan_amnt	funded_amnt	funded_amnt_inv
count	887379.000000	887379.000000	887379.000000
mean	14755.264605	14741.877625	14702.464383
std	8435.455601	8429.897657	8442.106732
min	500.000000	500.000000	0.000000
25%	8000.000000	8000.000000	8000.000000
50%	13000.000000	13000.000000	13000.000000
75%	20000.000000	20000.000000	20000.000000
max	35000.000000	35000.000000	35000.000000

```
In [31]: # and are indeed highly correlated
         data[['loan_amnt', 'funded_amnt', 'funded_amnt_inv']].corr()
```

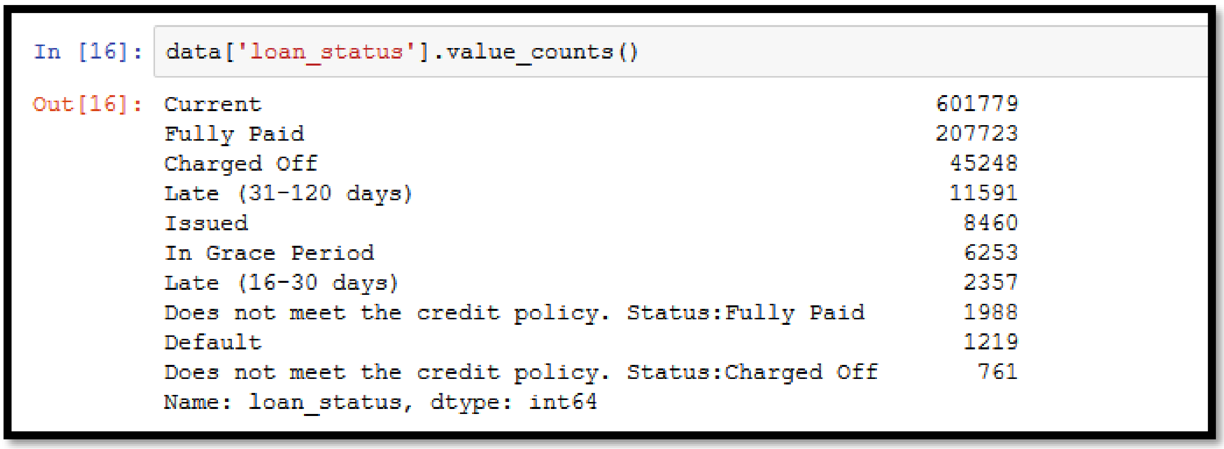
Out[31]:

	loan_amnt	funded_amnt	funded_amnt_inv
loan_amnt	1.000000	0.999263	0.997115
funded_amnt	0.999263	1.000000	0.998025
funded_amnt_inv	0.997115	0.998025	1.000000

Example code from my ipython notebook

The distribution of some continuous features ('out_prncp', 'out_prncp_inv', 'last_pymnt_amnt', 'tot_coll_amnt', 'tot_cur_bal', 'total_rev_hi_lim') seemed to be highly skewed, with most of the samples concentrated on low values, but still characterised by a significant amount of samples with high values which cannot be considered outliers. As such distribution can negatively affect the machine learning algorithms, these features needed to be transformed at pre-processing stage to have a more normal distribution.

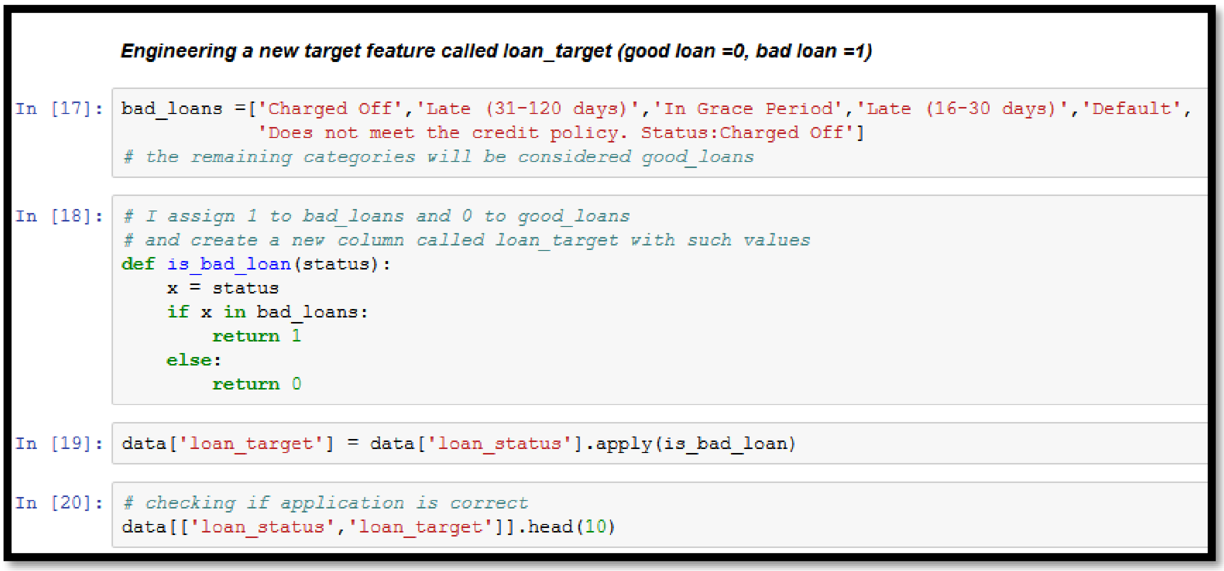
As the aim of the project was to predict loan default, I needed to identify the feature storing such information. The feature is 'loan_status' and has 10 categories, which I described in detail in my ipython notebook.



```
In [16]: data['loan_status'].value_counts()

Out[16]: Current                                                601779
         Fully Paid                                             207723
         Charged Off                                             45248
         Late (31-120 days)                                      11591
         Issued                                                   8460
         In Grace Period                                          6253
         Late (16-30 days)                                        2357
         Does not meet the credit policy. Status:Fully Paid       1988
         Default                                                  1219
         Does not meet the credit policy. Status:Charged Off       761
         Name: loan_status, dtype: int64
```

Example code from my ipython notebook

After understanding the meaning of the categories and similarly to what done in some Kaggle kernels [4], I engineered a new target column (**‘loan_target’)** containing binary values, with 1 meaning bad loan and 0 meaning good loan. I decided to consider bad loans the categories representing default, delays and charged off. This process was done almost at the start, as I wanted to have the target column available to analyse relationships with the various features during my exploratory visualisations.



Engineering a new target feature called loan_target (good loan =0, bad loan =1)

```
In [17]: bad_loans =['Charged Off','Late (31-120 days)','In Grace Period','Late (16-30 days)','Default',
                     'Does not meet the credit policy. Status:Charged Off']
         # the remaining categories will be considered good_loans

In [18]: # I assign 1 to bad_loans and 0 to good_loans
         # and create a new column called loan_target with such values
         def is_bad_loan(status):
             x = status
             if x in bad_loans:
                 return 1
             else:
                 return 0

In [19]: data['loan_target'] = data['loan_status'].apply(is_bad_loan)

In [20]: # checking if application is correct
         data[['loan_status','loan_target']].head(10)
```

Example code from my ipython notebook

The percentage of customers with bad loans in the original data is very low (approximately 7.6%). The target classes are highly imbalanced. Without running any model, we could predict that all customers will repay back their loan and be correct at 92.4% (naïve prediction). Strategies to deal with imbalanced data needed to be tested during model building in order to try to improve model performance.

1.5 Exploratory Visualisation

The exploratory data analysis section of my ipython notebook contains plots for the target and for most of the features. Here I report some interesting data insights (exploring target, company insights, some features that show relationship with target, examples of features requiring transformation).

The two plots below (Fig. 1) show the number of customers per class for the original 'loan_status' feature and for the engineered target column ('loan_target'). The plots clearly show that most of the Lending Club borrowers are in the 'Current' or 'Fully Paid' category for the 'loan_status' feature and therefore in the 'good loan' category for our target. The number of customers with bad loans is small (<10%).

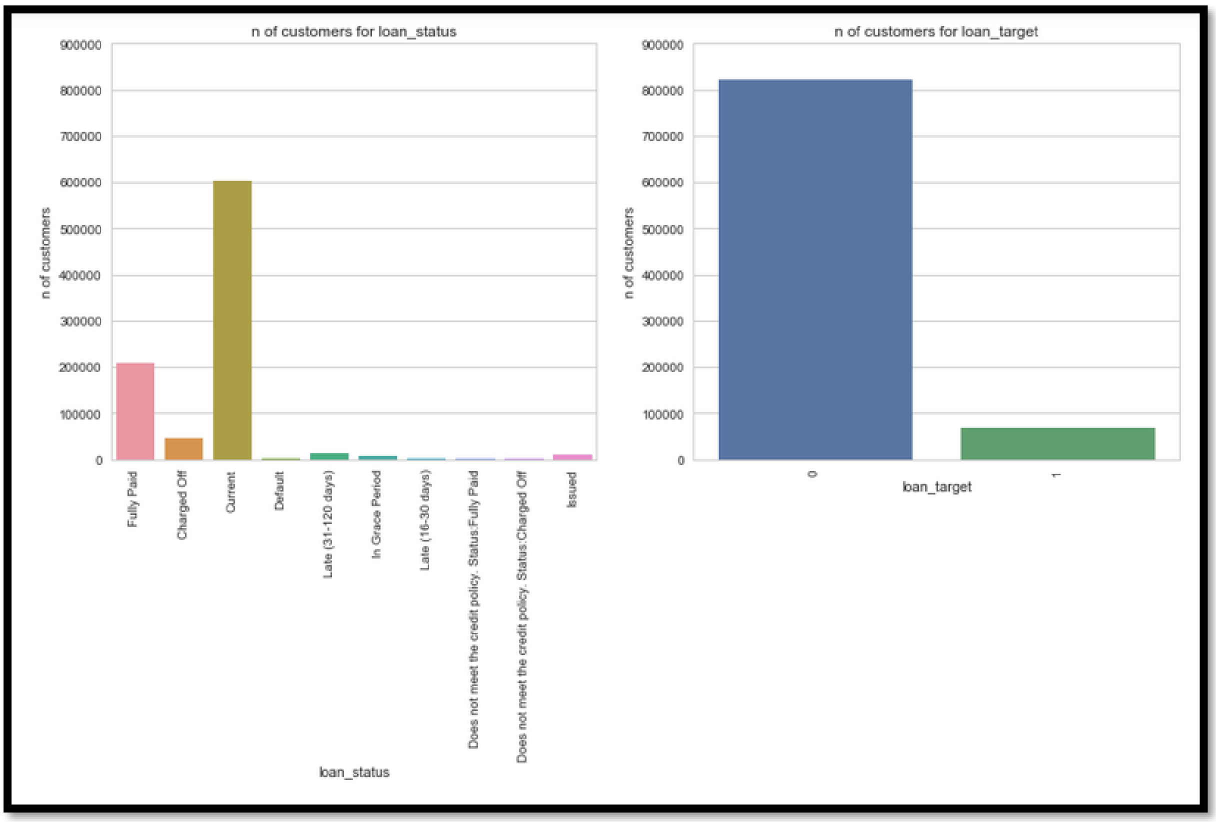
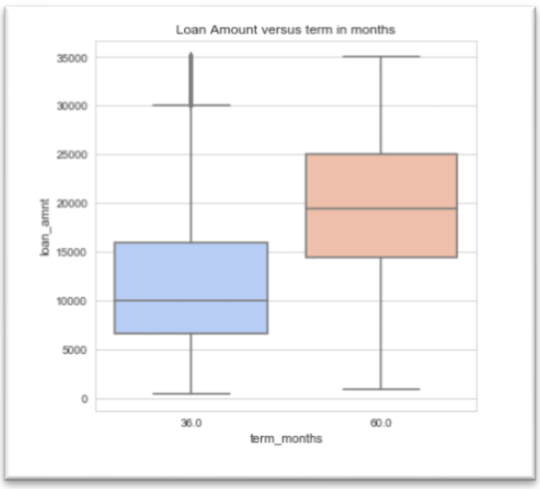


Fig. 1

Regarding the feature 'term', the boxplot below (Fig. 2) shows that the loan amount is generally higher for the borrowers with 60 months term. From the count plot (Fig. 3) it is evident that most of the borrowers have loans with 30 months term and that the percentage of bad loans is higher among the customers with 60 months term.



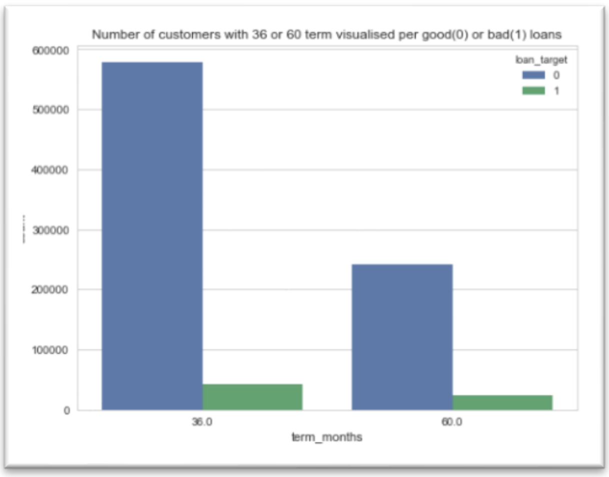


Fig. 2 and 3.

From Figure 4 we can see that most of the customers are in grade B and C, followed by A and D. From a first inspection, it seems that bad loans are more common for customers in grade C and D.

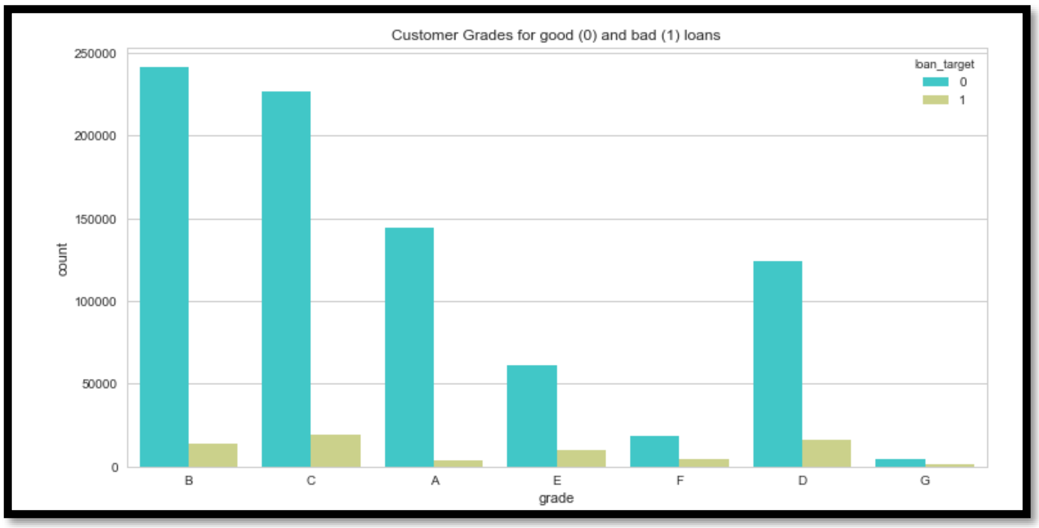


Fig. 4

The number of Lending Club customers has significantly increased over the years (Fig. 5, feature ‘issue_year’), doubling in size from 2013 to 2014 and then again from 2014 to 2015. The proportion of customers with bad loans was worse in the first years and has really decreased in 2015 compared for example to 2013.

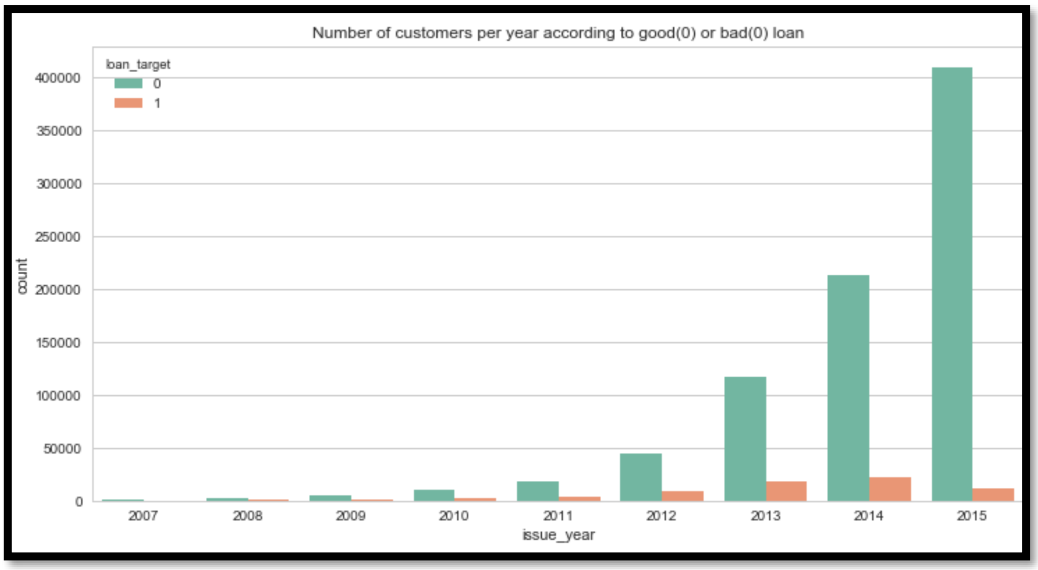


Fig. 5

The borrowers request loans mainly for debt consolidation and to repay credit card (Fig.6, feature 'purpose').

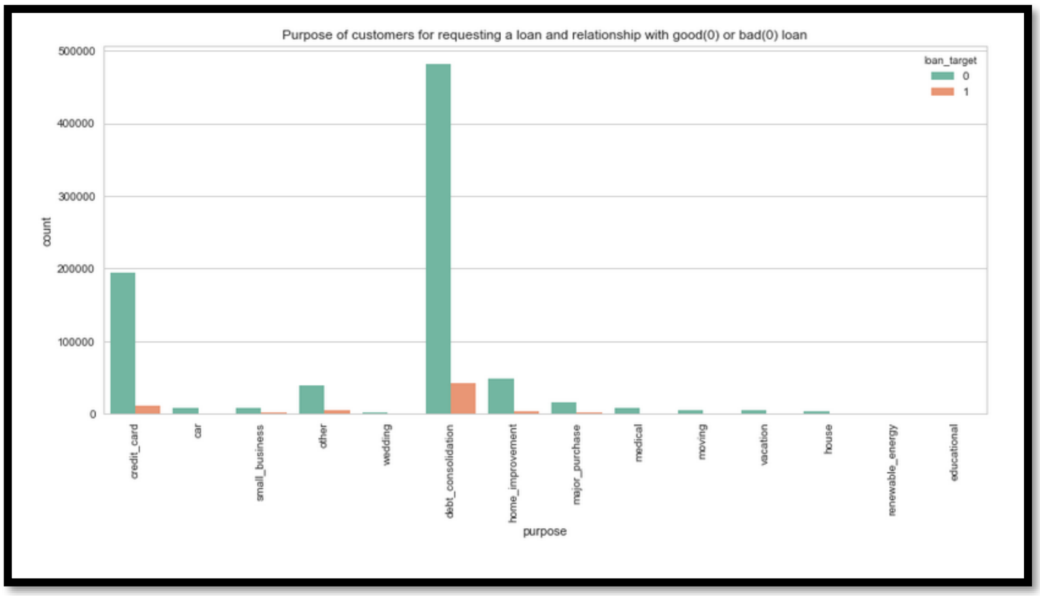


Fig. 6

The peak of the distribution of debt to income ratio (Fig.7, feature 'dti', after dealing with outliers) for customers with bad loans moves towards higher numbers.

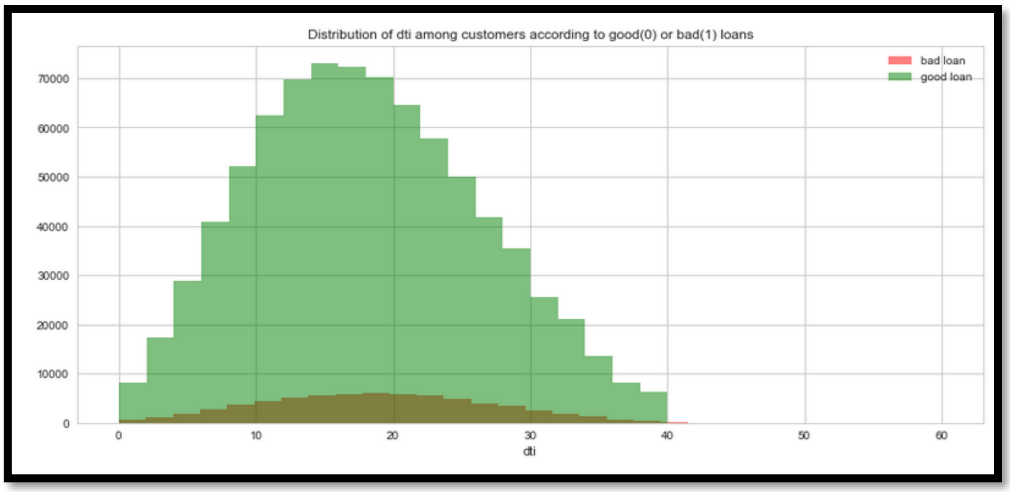
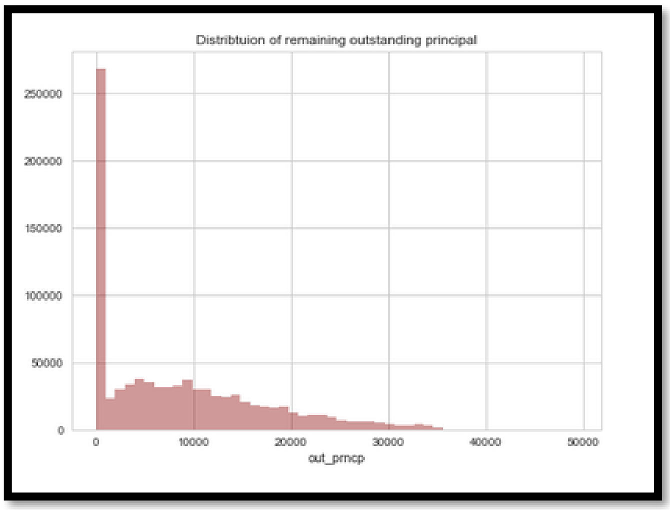


Fig. 7

As metioned before, during data exploration it was noted that some features have a skewed distribution with many samples around low values but also significant number of samples in much higher value ranges. This kind of distribution effects negatively model performances and needs to be cahnges at pre-processing stage. Fig. 8 and 9 show two examples (for ‘out_prncp’ and ‘last_pymnt_amnt’).



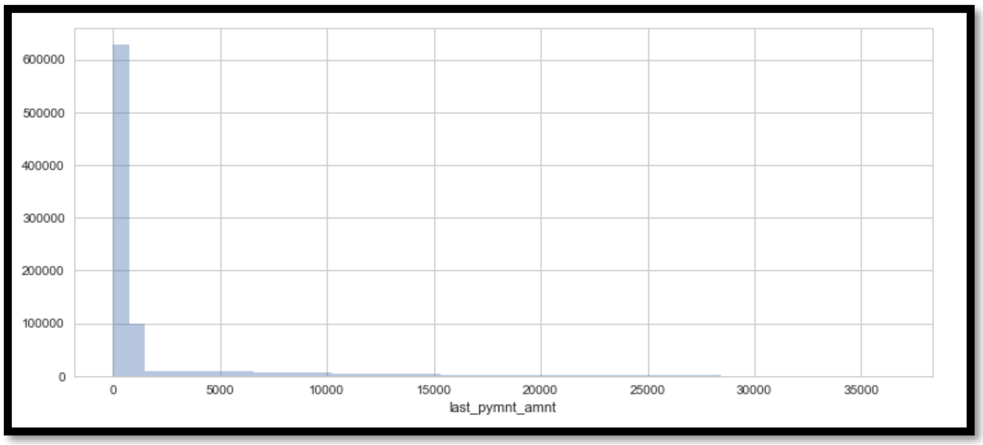


Fig. 8 and 9

1.6 Algorithms and Techniques

For this project, I wanted to be able to predict good (0) or bad (1) loans. Target information are stored in the data, in column 'loan_target'. This is therefore a supervised binary classification problem.

I tried and tested four different supervised learning models: Gaussian Naïve Bayse (GaussianNB), Logistic Regression, Random Forest Classifier and AdaBoost Classifier. All are available in scikit-learn [9].

Gaussian Naïve Bayse [1, 14, 15] is a powerful classification algorithm based on the probabilistic Bayse theorem. This theorem calculates the posteriory probability of our hypothesis (=bad loan) given the a-priori knowledge (=our features). The term naïve is used to underline that every attribute is considered independent from each other. Therefore, attributes which are highly correlated may affect the model performance because they will be 'voted' twice (in pre-processing I checked to reduce multi-collinearity). However, the model can still perform well. The advantages of Naïve Bayse are that it is very simple to understand and to implement. Moreover, the model trains fast and without requiring lots of data. It works well when there are many features, as in this data. Some disadvantages are that it assumes a Gaussian distribution of the variables (I improved the data on this by removing outliers). Moreover, it is not reliable if the new observations have very different distributions compared to training set.

I chose this model as I wanted a fast and simple starting point.

Logistic Regression [1, 16, 17, 18] is one of the most commonly used algorithms for binary classification problems such as default prediction and offered a solid baseline for me to compare with other more complex algorithms. Logistic Regression is based on the logistic or sigmoid function. The values of the features are combined linearly using weights to provide the probability of default, which can be mapped to a binary discreate output, 0 or 1, based on threshold. It is a very efficient algorithm, easy to train, implement and interpret, and requires limited computing resources. Being a linear algorithm, it assumes linear relationships between the input variables and the target. Highly correlated variables will affect negatively the model performance, causing overfitting (in pre-processing I checked for multi-collinearity). It may fail to converge if the data are very sparse.

The hyper-parameters are mainly linked with regularisation (l1 and l2) and strength of regularisation (parameter 'C').

As the data are really imbalanced, in my case I was interested in testing the parameter class_weight. If this is set to 'balanced' instead of default, a weighting on the class should be applied based on its frequency.

Random Forest Classifier [19] is another commonly used algorithm, due to its simplicity. It aggregates the results of an ensamble of Decision Trees to provide better predictions in terms of accuracy and stability. The observations and features used to build the trees are randomly selected and then results are averaged. This approach reduces the risk of overfitting. The default parameters often provide good results. A high number of trees (estimators) can slow down the model building, but the algorithm can provide good accuracy and run efficiently on large datasets.

Some of the important hyper-parameters of Random Forest, which are described in scikit-learn, are:

n_estimators = number of trees

max_depth = maximum depth of the trees

min_sample_split = minimum number of samples required to split an internal node

criterion = criteria to measure the quality of the split (‘gini’ or ‘entropy’)

class_weight = if None all classes are given the same weight, if ‘balanced’ the weights are adjusted based on class frequency (if ‘balanced_subsample’ the weights are computed based on boostrap sample)

AdaBoost Classifier [1, 20, 21] is a boosting method which combines multiple weak learners of the same type (usually low level Decision Trees) to create a better model. This meta-algorithm works iteratively: each model is built from the previous one by focusing on correcting the samples that were badly classified. The badly classified points are given higher weights. When all the weak learners have been created or there cannot be further improvements, all the outputs from the different models are combined, using a weighted ‘vote’ based on model performances. Due to this methodology, AdaBoost has the advantage of correcting for its own mistakes. However, it is sensitive to poor quality of training data, noisy data and outliers (I pre-processed the data in that respect).

I wanted to test this algorithm to compare it mainly against Random Forest.

Some of the important hyper-parameters, described in scikit learn, are:

n_estimators = maximum number of estimators at which boosting is terminated

learning_rate = value that shrink the contribution of each estimator

As the target classes are really imbalanced apart from trying the parameters class_weight=’balanced’ available in Logistic Regression and RandomForest, I tested **over sampling with SMOTE** (Synthetic Minority Oversampling Technique) [22, 23]. This algorithm creates synthetic observations based on the minority class by firstly calculating the k nearest neighbours and then by creating random synthetics along the line that connects the current observation to the k neighbours.

10 k fold cross-validation was run to prevent overfitting. In this technique the training data are divided in 10 subsets and each time 1 different subset is used for testing and the other 9 for training. The average error across all k trials is then computed. In this way we do keep the testing data unseen by the model.

Sciki learn **RandomisedSearchCV** was tested in order to tune model hyper-parameters. In the feedbacks of my Udacity Projects, reviewers suggested to try it instead of GridSearchCV as it runs much faster.

1.7 Benchmark

The most basic benchmark for my model is the Naïve prediction (what happens if we predict according to the majority that all customers will have good loans). In this case, we would be correct at 92.42%. As a starting point, the accuracy of my model must be better than this.

A second benchmark can be my basic Logistic Regression model (with default parameters), as this algorithm is commonly used for predicting default.

Many kernels are available on the Kaggle website for the Lending Club data [2]. Most of them focus on data visualisation and analysis, but some report also machine learning models (I have listed some of this kernels in the References). I want to benchmark my results against some of the Kaggle kernels checking precision and recall, because more important than just accuracy for this topic. However, proper comparison can be difficult.

2. Methodology

2.1 Data Pre-processing

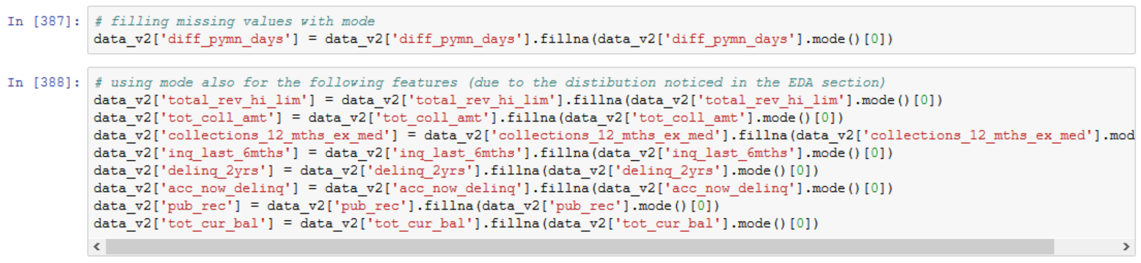
Missing values

During data exploration, some features with missing data were identified. The features with more than 50% of missing values were dropped at the beginning of the project.

The feature 'annual_inc' has only 4 missing values. Looking at the values of other features for these samples, I could assume that filling the missing value of the annual income with the mode/median would be wrong as these customers are probably on the low earning range (ex student). Due to the high number of samples in the data and the fact that these 4 are not in the bad loan category, I decided it was safe to drop them.

The engineered feature 'diff_pymnt_days' had 30% of missing values (coming from many missing data in the original feature 'next_pymnt_d'). To understand better the source of these missing data and decide how to deal with them I created a temporary binary flag column ('diff_pymn_d_null') for checking correlations with target and other features. I initially filled the missing values with the mode, but then noticed an high correlation between my flag feature and 'out_prnp' and the feature was dropped.

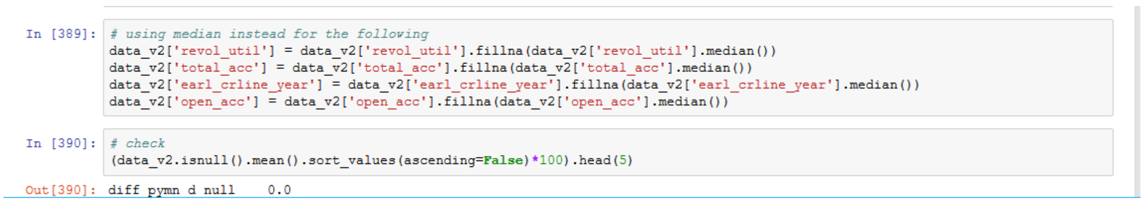
For the remaining features, the missing values were filled either with mode or median.



```
In [387]: # filling missing values with mode
          data_v2['diff_pymn_days'] = data_v2['diff_pymn_days'].fillna(data_v2['diff_pymn_days'].mode()[0])
```

```
In [388]: # using mode also for the following features (due to the distibution noticed in the EDA section)
          data_v2['total_rev_hi_lim'] = data_v2['total_rev_hi_lim'].fillna(data_v2['total_rev_hi_lim'].mode()[0])
          data_v2['tot_coll_amt'] = data_v2['tot_coll_amt'].fillna(data_v2['tot_coll_amt'].mode()[0])
          data_v2['collections_12_mths_ex_med'] = data_v2['collections_12_mths_ex_med'].fillna(data_v2['collections_12_mths_ex_med'].mod
          data_v2['inq_last_6mths'] = data_v2['inq_last_6mths'].fillna(data_v2['inq_last_6mths'].mode()[0])
          data_v2['delinq_2yrs'] = data_v2['delinq_2yrs'].fillna(data_v2['delinq_2yrs'].mode()[0])
          data_v2['acc_now_delinq'] = data_v2['acc_now_delinq'].fillna(data_v2['acc_now_delinq'].mode()[0])
          data_v2['pub_rec'] = data_v2['pub_rec'].fillna(data_v2['pub_rec'].mode()[0])
          data_v2['tot_cur_bal'] = data_v2['tot_cur_bal'].fillna(data_v2['tot_cur_bal'].mode()[0])
```

Example code from my ipython notebook



```
In [389]: # using median instead for the following
          data_v2['revol_util'] = data_v2['revol_util'].fillna(data_v2['revol_util'].median())
          data_v2['total_acc'] = data_v2['total_acc'].fillna(data_v2['total_acc'].median())
          data_v2['earl_crline_year'] = data_v2['earl_crline_year'].fillna(data_v2['earl_crline_year'].median())
          data_v2['open_acc'] = data_v2['open_acc'].fillna(data_v2['open_acc'].median())

In [390]: # check
          (data_v2.isnull().mean().sort_values(ascending=False)*100).head(5)

Out[390]: diff pymn d null    0.0
```

Example code from my ipython notebook

Outliers

During exploratory data analysis, outliers were identified from plots and statistical analysis (ex using 3 times the inter quantile to calculate thresholds). The potential outliers were compared against good/bad loan target to make sure that they were not good indicators of the bad loan category. If not, the feature was capped at outlier threshold value. This was applied to the following columns: annual_inc', 'dti' ('dti' had also two very high bad records which I dropped), 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'total_pymnt', 'total_pymnt_inv' and 'total_rec_int' .

Figure 10 shows an example for annual income bevore vs afer dealing with outliers.

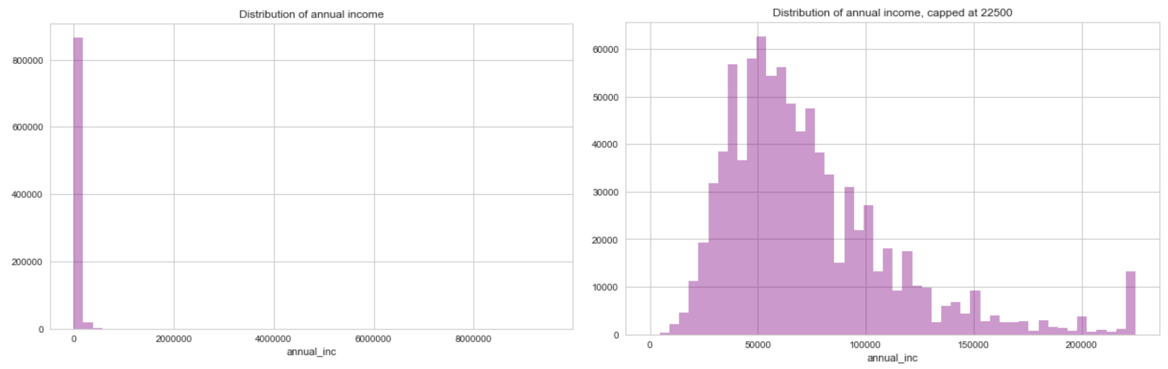
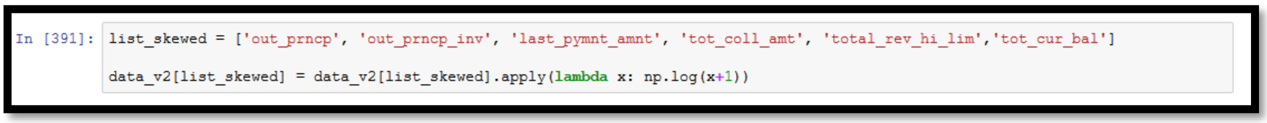


Fig 10

Transformed skewed continuous features

After EDA, I knew which features needed to be transformed to obtain better value distributions, to have values concentrated in a smaller range and following a more 'normalised' shape('out_prncp','out_prncp_inv','last_pymnt_amnt''total_coll_amnt''tot_cur_bal''total_rev_hi_lim'). A log transformation was applied (result example in Fig 11).



```
In [391]: list_skewed = ['out_prncp', 'out_prncp_inv', 'last_pymnt_amnt', 'tot_coll_amt', 'total_rev_hi_lim','tot_cur_bal']

          data_v2[list_skewed] = data_v2[list_skewed].apply(lambda x: np.log(x+1))
```

Example code from my ipython notebook

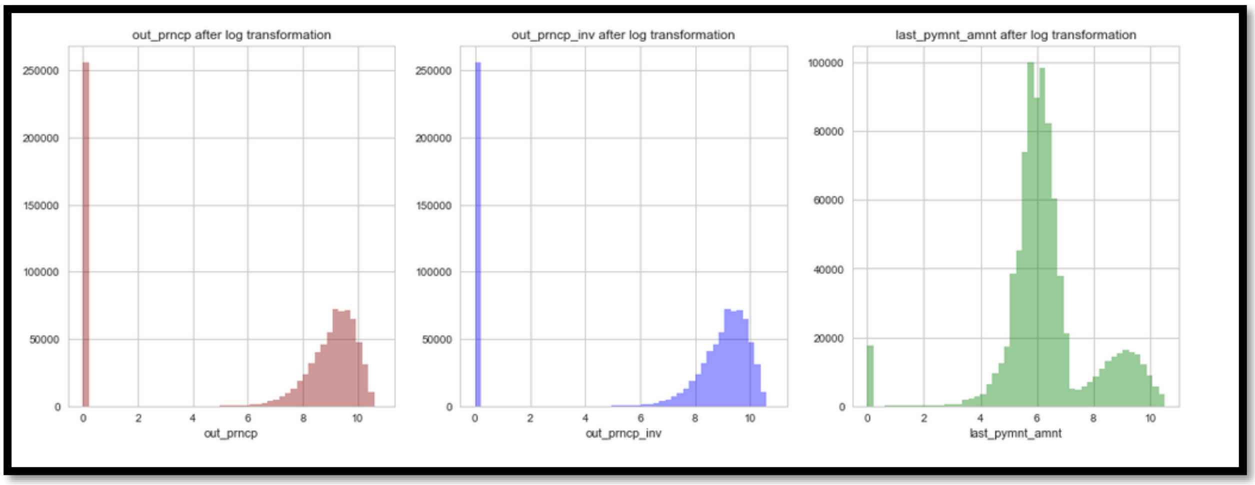
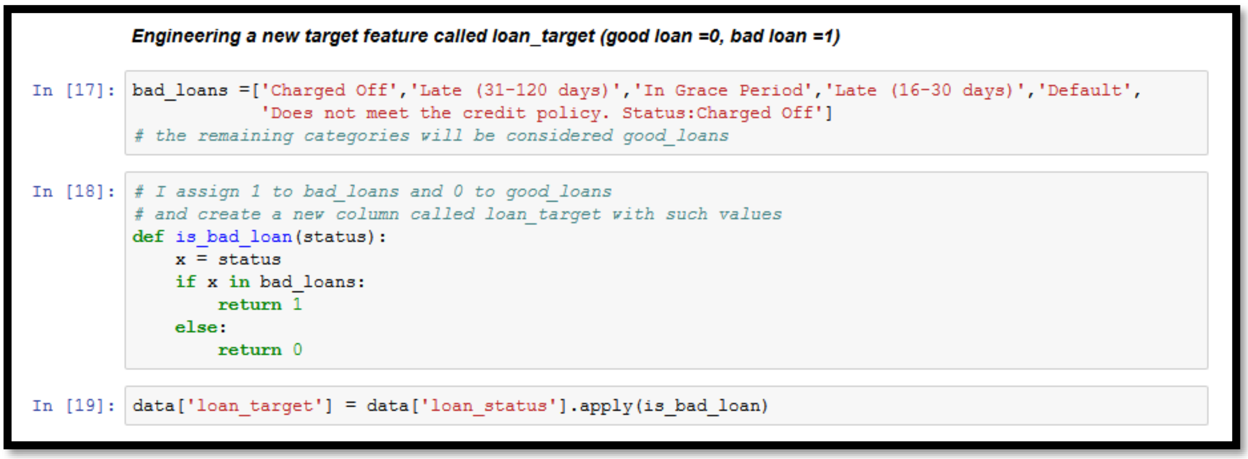


Fig. 11

Feature Engineering and Dealing with categorical variables.

While exploring data, it was noted that some features could/should be engineered.

As already explained, a loan_target column (good loan = 0, bad loan = 1) was created from the original 'loan_status' feature (which was dropped at the beginning of pre-processing stage).



Engineering a new target feature called loan_target (good loan =0, bad loan =1)

```
In [17]: bad_loans =['Charged Off','Late (31-120 days)','In Grace Period','Late (16-30 days)','Default',
                     'Does not meet the credit policy. Status:Charged Off']
         # the remaining categories will be considered good_loans

In [18]: # I assign 1 to bad_loans and 0 to good_loans
         # and create a new column called loan_target with such values
         def is_bad_loan(status):
             x = status
             if x in bad_loans:
                 return 1
             else:
                 return 0

In [19]: data['loan_target'] = data['loan_status'].apply(is_bad_loan)
```

Example code from my ipython notebook

A new numerical feature 'term_months' (with values of 36 or 60) was engineered from the original feature 'term' ('36 months', '60 months'), which was dropped.

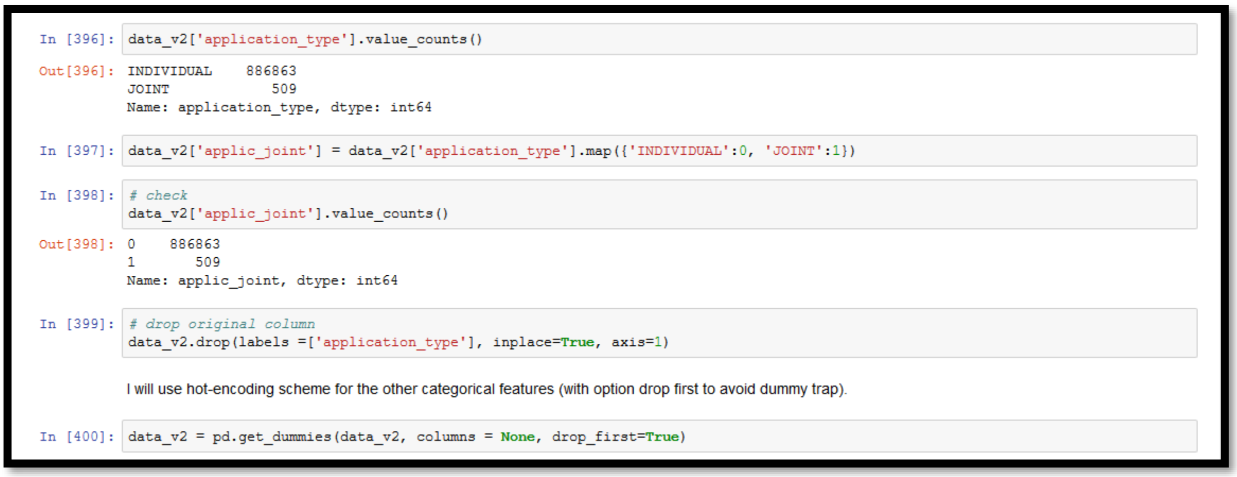
The feature 'emp_length' was changed from categorical to numerical by dropping symbols and the word year/s. Moreover, the categories 'n/a' and '<1' were replaced with 0.

Pandas datetime was used to engineer the new feature 'issue_y' from the original 'issue_d' and 'earl_crline_year' from 'earliest_cr_line'. The original features were dates in the form of Month-Year and I retained only the year. After, I dropped the original features.

I initially tried to engineer a new feature called 'diff_pymn_days' changing 'next_pymnt_d' and 'last_pymnt_d' with Pandas datetime and calculating days of difference between the two (and then capped at 186 days).

A new numerical binary feature 'applic_joint' was created from the original categorical feature 'application_type'. The only two categories 'INDIVIDUAL' and 'JOINT' were mapped to 0 and 1 respectively.

The remaining categorical features ('grade', 'home_ownership', 'verification_status', 'purpose' and 'initial_list_status') were hot-encoded dropping the first column to avoid the dummy trap.



```
In [396]: data_v2['application_type'].value_counts()

Out[396]: INDIVIDUAL    886863
          JOINT            509
          Name: application_type, dtype: int64

In [397]: data_v2['applic_joint'] = data_v2['application_type'].map({'INDIVIDUAL':0, 'JOINT':1})

In [398]: # check
          data_v2['applic_joint'].value_counts()

Out[398]: 0    886863
          1       509
          Name: applic_joint, dtype: int64

In [399]: # drop original column
          data_v2.drop(labels =['application_type'], inplace=True, axis=1)
```

I will use hot-encoding scheme for the other categorical features (with option drop first to avoid dummy trap).

```
In [400]: data_v2 = pd.get_dummies(data_v2, columns = None, drop_first=True)
```

Example code from my ipython notebook

Checking for multi-collinearity

A correlation matrix was created and examined to identify features with high correlation values which would influence negatively the performance of some models. As a result, the features 'out_prncp_inv', 'total_pymnt_inv', 'installement', 'total_rec_prncp', 'diff_pymn_days', 'diff_pymn_d_null' were dropped (they had correlation values with other features >0.9).

Final data

At the end of pre-processing the total number of records was 887372 and the total number of columns (features plus target) was 51.

2.2 Implementation

After the pre-processing, the final data were separated into a feature matrix (X) and target (y). Then they were shuffled and split into training and testing datasets using scikit learn train_test_split (keeping 20% of data for training). The features were scaled using Scikit learn StandarScaler, with fitting done on the training data.

The first models for the four classifiers (Gaussian Naïve Basye, Logistic Regression, Random Forest Classifier and AdaBoost Classifier) were built with basic/default hyper-parameters and no correction for imbalanced classes.

Running time for the classifer.fit function plus confusion metrics, classification report and accuracy/precision/recall scores were derived for all models.

10 k-fold cross validation was run to prevent overfitting.

The performance metrics for these **models1** is shown on Table1.

Tab 1.	**MODELS 1 (basic)**			
metrics	**Gausssian Naïve Bayse**	**Logistic Regression**	**Random Forest Classifier**	**AdaBoost Classifier**
Accuracy on training set	0.862022237029	0.962807280493	0.994118865131	0.96031677835
Accuracy on testing set	0.862775038738	0.962248203972	0.970147908156	0.950678828004
Precision	0.215900725628	0.914226953787	0.986209335219	0.910585817061
Recall	0.303070563921	0.557794508415	0.617581930912	0.523176852672
F1	0.25	0.69	0.76	0.66
Training time	1.687383890	19.7390229701	58.141054153	170.031042337

Random Forest Classifier is the model which provides higher recall (around 0.618). It also has the highest precision (0.986) and testing accuracy (0.970). However, the very high accuracy on training set may suggest some overfitting.

Although Gaussian Naïve Bayse had the advantage of running very fast, its performance is very poor and it is the worse among the models (recall and precision of only 0.303 and 0.216 respectively).

The performance of Logistic Regression and AdaBoost was similar, with the first being a bit better and AdaBoost being much slower to run.

2.3 Refinement

My aim after seen the results of the first models was to try to increase recall as it was the most important metric (but keeping still a reasonable level of precision). One of the reasons why the recall is low could be the class imbalance.

I tested the following to see which option would provide the best results:

Models2) Use the hyper-parameter **class_weight = balanced**, which is available for Logistic Regression and for RandomForest;

Models3) Over sample the data using SMOTE algorithm and then run the models with same hyper-parameters as in Models1;

Models4) Run RandomizedSearchCV to tune hyper-parameters for RandomForest and AdaBoost (on no over sampled data). The search was run on subset of data (50000 samples) to speed up the testing for the hyperparameter tuning.

Tab 2. MODELS 2 (class_weight=balanced)

metrics	**Gausssian Naïve Bayse**	**Logistic Regression**	**Random Forest Classifier**	**AdaBoost Classifier**
Accuracy on training set		0.846479137114	0.994114639116	
Accuracy on testing set		0.84760670517	0.970069023806	
Precision		0.299459202472	0.990588515606	
Recall		0.743873634485	0.613743726011	
F1		0.43	0.76	
Training time		17.387309074	56.927880048	

Tab 3. MODELS 3 (same parameters of Models1 but on over sampled data)

metrics	**Gausssian Naïve Bayse**	**Logistic Regression**	**Random Forest Classifier**	**AdaBoost Classifier**
Accuracy on training set	0.698780212678	0.809032858955	0.998673050657	0.9052235543036
Accuracy on testing set	0.817455979715	0.853128609663	0.969567554725	0.9091083225116
Precision	0.214348498166	0.307569329152	0.939950318609	0.439157676544
Recall	0.5219995866549	0.738411573664	0.642382639504	0.688072040154
F1	0.30	0.43	0.76	0.54
Training time	3.015399217	33.8912770748	141.491475582	452.978650331

Tab 4.	MODELS 4 (parameter tuning, original final data)			
metrics	Gausssian Naïve Bayse	Logistic Regression	Random Forest Classifier	AdaBoost Classifier
Accuracy on training set			0.976244441095	0.964104651802
Accuracy on testing set			0.971117058741	0.963268065925
Precision			0.99366940211	0.94102145815
Recall			0.625627398878	0.553513433717
F1				
Training time			263.89823889	261.4085471630

3. Results

3.1 Model Evaluation and Validation

The Gaussian Naïve Bayse model performed better in terms of recall after data over-sampling (from 0.216 to 0.521), however precision remain basically the same and very low (around 0.21). The accuracy also decreased after over-sampling, becoming quite low on the training dataset (0.699).

The recall for Logistic Regression increased both after changing to balanced mode and after over-sampling (passed from 0.578 of Model1 to 0.743 and 0.738 of Models2 and 3). The balanced mode option has the advantage of running faster compared to over-sampled model3. The precision and accuracy dropped. In particular precision became quite low (from 0.914 of Model 1 to approx. 0.30 of Models 2 and 3). Although we are more interested in recall, this very low precision is a bit worrying as Lending Club would lose many good customers.

Random Forest Classifier showed almost the same performance in Model1 and 2 (no evident improvement with balanced mode and still possible overfitting). Recall increased after over-sampling (from 0.61-2 to 0.64) with some loss in precision (from 0.98-99 to 0.93) but not in accuracy. There is still a risk of overfitting in training data. The model 4 with hyper-parameter tuning was characterised by a recall of 0.62 similar to the others, good precision (0.99, higher than the 0.93 of Model3) and importantly by a training accuracy now at 0.976 more similar to testing. Therefore the risk of overfitting on training has been reduced. This is probably the best model in terms of overall performance.

The recall for AdaBoost classifier increased significantly from 0.52 to 0.688 after over-sampling, but precision dropped from 0.910 to 0.688. Performance improved after parameter tuning from Model1 to Model4 in all the metrics.

Overall**, Radom Forest Classifier and AdaBoost Classifier Models 4** (after parameter tuning) offered the best compromise in terms of high accuracy, high precision, and recall. Random Forest is better as the recall is 0.626 and runs much faster than AdaBoost. The new hyper parameters succeded in reducing the training accuracy to avoid overfitting.

The best hyper parameters for the **Random Forest Classifier Model 4** model were: criterion = 'entropy', max_depth = 25, min_sample_split = 10 and n_estimators = 50.

As mentioned above to speed up my testing I run the RandomizedSearchCV on the first 50000 samples. When I tested on less samples, I was having very similar results.

10 k-fold cross validation provided accuracy of approximately 0.971462 with a standard deviation of 0.000632, which should support no overfitting and good accuracy on different runs.

This model still has a relatively low recall value (approx. 0.62). More testing would be required to try to improve it even more. For example, I could try other ways to balance data, improving the way I used SMOTE algorithm, or I could test GridSearchCV to see if it provides different and better hyper-parameters. Moreover, as I can look at the most important features, I can try to reduce the number of features in input, keeping the most relevant ones.

3.2 Justification

My final Model provides an accuracy higher than for a naïve prediction. It performs also better than my basic Logistic Regression Model 1 in all the metrics (accuracy, precision and recall), although takes more time to run. It also performs much better in recall than the Random Forest model shown on Kaggle kernel [a]. That was characterised by accuracy and precision of 1, but recall of only 0.6. On the same kernel[a] a boosting algorithm called lightGBM is used and it seems to have provided accuracy score 0.9997, precision 0.98 and recall 0.82. The recall is much higher than the 0.62 of my model. As a note, in the future I should try to test this algorithm on my final data.

On kernel[b], taking into consideration that here bad loans=0 and good loans=1, a support vector machine algorithm had been run before and after data oversampling with SMOTE and it provided in both cases a recall of 0.65 (a bit higher than mine 0.62) and a much lower precision (0.51) compared to mine (0.99). For the same kernel, both RandomForest and XGBOOST,with default prameters, seem to provide much better results than mine (0.99 in precision and 0.71 in recall) but looking at the confusion matrix it is clear that the number of samples has been reduced.

It is not so easy to compare my results with the ones found in Kaggle. However, looking at the various kernels, it seems that I should be testing Boosting algorithms and also different ways of applying SMOTE, which is new for me.

4. Conclusion

In summary, for this project I worked on the Lending Club data with the aim of building a supervised classification model able to predict good/bad loans.

I analysed all the features in input to build an intuition of the relationship between the features and the target and to gather insights about the borrowers. I engineered the target and other features. In pre-processing stage, I dealt with missing values, outliers, categorical features, feature transformation and multi-collinearity. The high number of features for a domain which is far from my expertise posed a challenge and requested more time than I anticipated.

After pre-processing, the data were shuffled and split into training and testing dataset (80 and 20%) and standardised to be ready for model building.

I run some tests for four different supervised classification algorithms:

1) Using basic/default hyper-parameters
2) Using class_weight='balanced' hyper parameter
3) Using over sampled data to try to balance the classes in input
4) Using new tuned hyper-parameters

My final preferred model was a Random Forest Classifier with tuned parameters which provided the following performance results:

metrics	**Random Forest Classifier (Model4)**
Accuracy on training set	0.976244441095
Accuracy on testing set	0.971117058741
Precision	0.99366940211
Recall	0.625627398878
Training time	263.89823889

The performance was good in accuracy (in both training and testing data) and in precision. However, I did not manage to increase recall more than 0.625. Recall is the most important metrics for this loan prediction problem. Although more work can be carried out in trying to process better the input data and possibly gather the information which were missing (ex fico score), I believe that the main problem is the imbalanced nature of the loan class (Fig. 12). The number of samples for bad loans is too small for the model to train on them and then predict them.

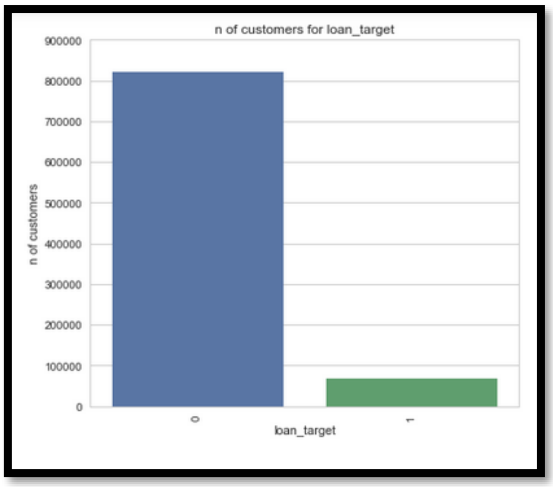


Fig. 12

I should do more research on how to balance better the data, looking at other ways to use SMOTE or searching for other techniques. It would be interesting to see also how a neural network would perform.

The Random Forest Model provided also details on the most important features used for prediction (Fig. 13). These provide a structure to explain why borrowers are denied loans in case needed. Moreover, a further modelling test could be carried out by selecting only the most important features as input to the model.

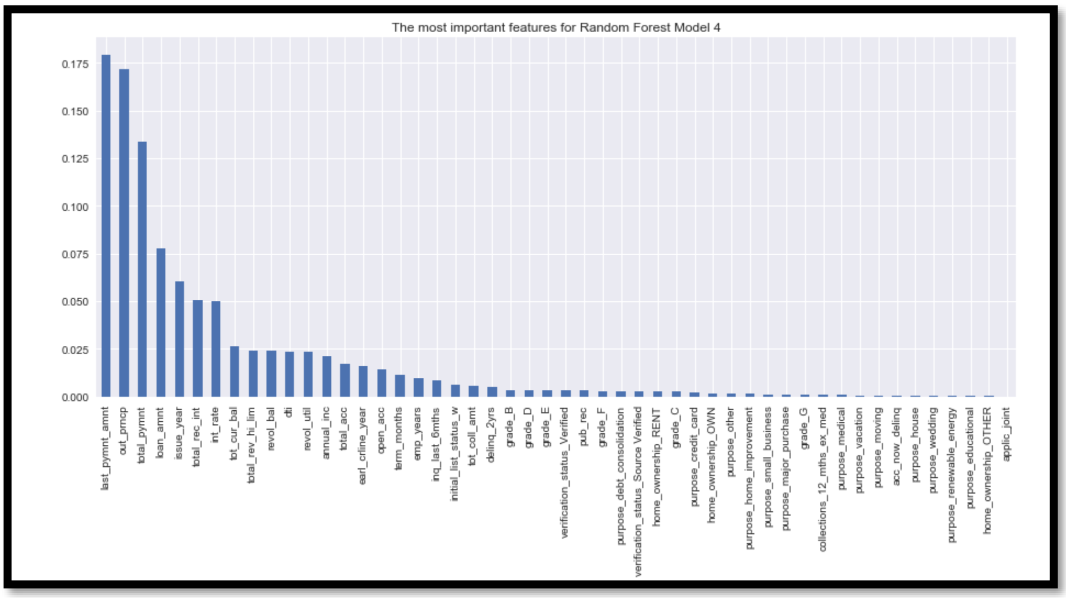


Fig. 13

A think to bear in mind is also that the problem of predicting default is often considered in terms of probability and not just a binary good/bad, yes/no topic. In the future, I could explore considering probability of a customer to default and change the thresholds of definition of default.

5. References

[1] I.H. Witten, F. Eibe, M.A. Hall, 2011. ‘Data Mining: practical machine learning tools and techniques’, 3rd edition.
[2] https://towardsdatascience.com/predicting-loan-repayment-5df4e0023e92
[3] https://github.com/topics/loan-default-prediction
[4] https://www.kaggle.com/wendykan/lending-club-loan-data/kernels
[5] https://www.kaggle.com/wendykan/lending-club-loan-data
[6] https://www.lendingclub.com/
[7] https://en.wikipedia.org/wiki/Lending_Club
[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
[9] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[10] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
[11] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
[12] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
[13] https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
[14] https://machinelearningmastery.com/naive-bayes-for-machine-learning/
[15] https://machinelearningmastery.com/better-naive-bayes/
[16] https://machinelearningmastery.com/logistic-regression-for-machine-learning/
[17] https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
[18] https://machinelearning-blog.com/2018/04/23/logistic-regression-101/
[19] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
[20] https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/
[21] https://www.analyticsvidhya.com/blog/2015/05/boosting-algorithms-simplified/
[22]https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html
[23] http://www.dataminingapps.com/2016/11/what-is-smote-in-an-imbalanced-class-setting-e-g-fraud-detection/

Example of Lending Club kernels with classification models on Kaggle website:
[a] https://www.kaggle.com/pragyanbo/a-hitchhiker-s-guide-to-lending-club-loan-data
[b] https://www.kaggle.com/wsogata/good-or-bad-loan-draft
[c] https://www.kaggle.com/deepanshu08/prediction-of-lendingclub-loan-defaulters
[d] https://www.kaggle.com/jlrsource/predicting-loan-status-with-python
[f] https://www.kaggle.com/kacpiw/who-will-default-on-a-loan-logistic-regression
[g]https://www.kaggle.com/aamirsiddiqui/lending-club-loan-machine-learning