Machine Learning Engineer Nanodegree

Lending Club *Investor ML*

Predict whether individual loan application will be charged off?

Sriharsha M S
April 23rd, 2018

I. Definition

Project Overview

LendingClub is an online financial community that brings together creditworthy borrowers and savvy investors so that both can benefit financially. They replace the high cost and complexity of bank lending with a faster, smarter way to borrow and invest. Since 2007, LendingClub has been bringing borrowers and investors together, transforming the way people access credit. Over the last 10 years, LendingClub helped millions of people take control of their debt, grow their small businesses, and invest for the future.

LendingClub balances different investors on its platform. The mechanics of the platform allow LendingClub to meet the objectives of many different types of investors, including retail investors. Here's how it works. Once loans are approved to the LendingClub platform, they are randomly allocated at a grade and term level either to a program designed for retail investors purchasing interests in fractions of loans (e.g. LendingClub Notes) or to a program intended for institutional investors. This helps ensure that investors have access to comparable quality loans no matter which type of investor they are. LendingClub goal is to meet incoming investor demand for interests in fractional loans as much as possible.

The design of LendingClub platform emphasizes how important retail investors are to LendingClub. For LendingClub retail investors are key component of our diverse marketplace strategy. Retail investors are—and will always be—the heart of the LendingClub marketplace.

This project "*Investor ML*" tool specifically built keeping retail investors in mind. *Investor ML* aims to predict probability risk of borrower being charged off and not fully pay loan amount, called "**Risk Rate %**". LendingClub can provide "**Risk Rate %**" predictions from Investor ML for each loan of borrower as an additional indicator for investors to make investment decisions. Retail investors use loan information, borrower information like fico score, loan grade etc. and decide to invest in fractional loans. Additional statistic "**Risk Rate %**" learned from historical loans may also act as additional information and help retail investors to diversify their investment.

The datasets are provided by LendingClub, we will use lending data from 2007-2011 and be trying to classify and predict whether or not the borrower paid back their loan in full. Data is available to download https://www.lendingclub.com/info/download-data.action.

Problem Statement

The two most critical questions in the lending industry are: 1) How risky is the borrower? 2) Given the borrower's risk, should we lend him/her? The answer to the first question determines the interest rate the borrower would have. Interest rate measures among other things (such as time value of money) the riskiness of the borrower, i.e. the riskier the borrower, the higher the interest rate. With interest rate in mind, we can then determine if the borrower is eligible for the loan.

Investors (lenders) provide loans to borrowers in exchange for the promise of repayment with interest. That means the lender only makes profit (interest) if the borrower pays off the loan and don't get charged off. However, if borrower doesn't repay the loan and get charged off, then the lender loses money.

We will address the second question indirectly by trying to predict if the borrower will repay the loan fully or get charged off. Using the historical dataset of all loans, we have the ground truth of loans that are charged off and fully paid. We will build machine learning classifier to learn from the loans and use it to predict the "Risk Rate %" off loan being charged off.

Investor ML is supervised learning, binary classification problem. *Investor ML* classifier model fits the training data and learns to classify the loans target variable charged off or fully paid.

Investor ML solution will consist of,

1. Exploratory analysis and understanding of historical loans of data
2. Build intuitions select and prepare features, treat missing values, scale numerical features, encode categorical features.
3. Prepare target label of charged off or not, study the class imbalance and handle if class imbalance is an issue.
4. Build multiple classifiers use performance metric to choose the ensemble models.
5. Save the model so that saved model can be used to infer loan and provide probability of "Risk Rate%".

Metrics

We will also use ROC AUC statistic for model comparison to choose the model based in class imbalance solution to compare between model-not balanced and model-balanced.

Before presenting the ROC curve (= Receiver Operating Characteristic curve), the concept of **confusion matrix** must be understood. When we make a binary prediction, there can be 4 types of outcomes:

- We predict 0 while we should have the class is actually 0: this is called a **True Negative**, i.e. we correctly predict that the class is negative (0). For example, an antivirus did not detect a harmless file as a virus.
- We predict 0 while we should have the class is actually 1: this is called a **False Negative**, i.e. we incorrectly predict that the class is negative (0). For example, an antivirus failed to detect a virus.
- We predict 1 while we should have the class is actually 0: this is called a **False Positive**, i.e. we incorrectly predict that the class is positive (1). For example, an antivirus considered a harmless file to be a virus.
- We predict 1 while we should have the class is actually 1: this is called a **True Positive**, i.e. we correctly predict that the class is positive (1). For example, an antivirus rightfully detected a virus.

To get the confusion matrix, we go over all the predictions made by the model, and count how many times each of those 4 types of outcomes occur:

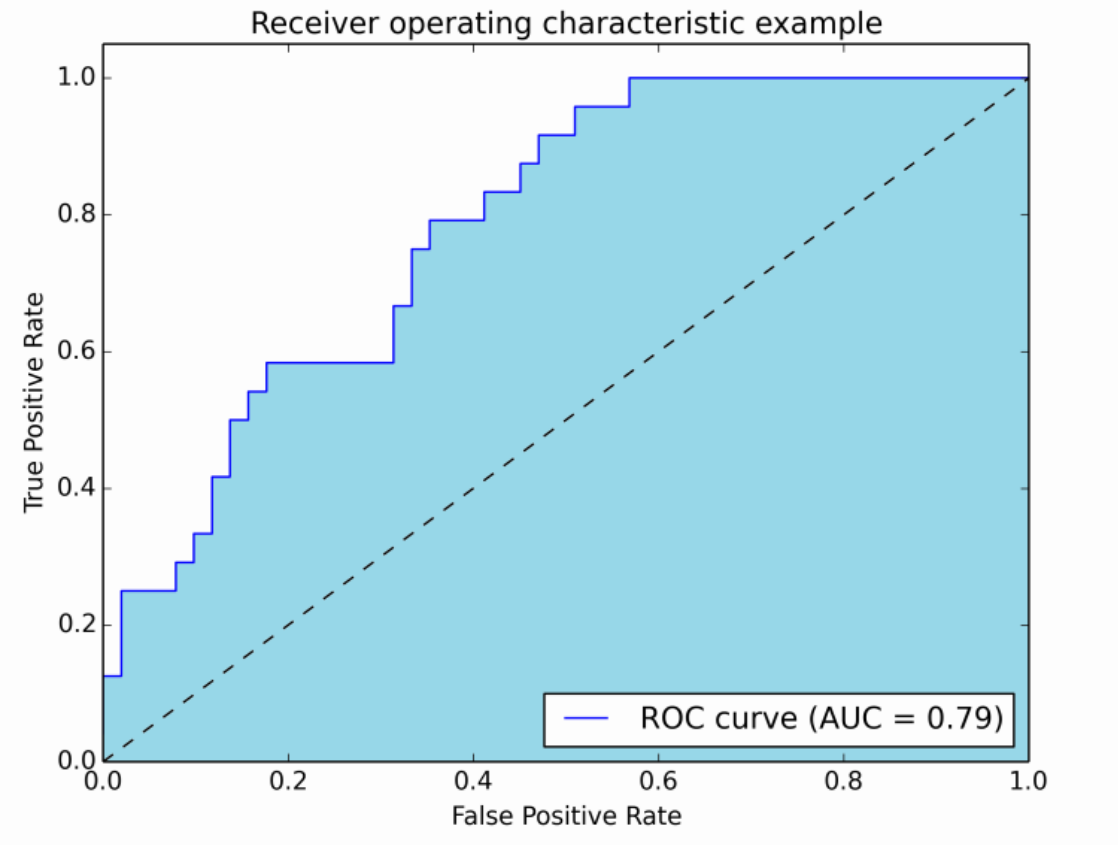
		Predicted class	
		Class 1	Class 0
Actual class	Class 1	10 true positives (TP)	2 false negatives (FN)
	Class 0	3 false positives (FP)	35 true negatives (TN)

In this example of a confusion matrix, among the 50 data points that are classified, 45 are correctly classified and the 5 are misclassified.

Since to compare two different models it is often more convenient to have a single metric rather than several ones, we compute two metrics from the confusion matrix, which we will later combine into one:

- True positive rate (**TPR**), aka. sensitivity, hit rate, and recall, which is defined as TPTP+FNTPTP+FN. Intuitively this metric corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. In other words, the higher TPR, the fewer positive data points we will miss.
- False positive rate (**FPR**), aka. fall-out, which is defined as FPFP+TNFPFP+TN. Intuitively this metric corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points. In other words, the higher FPR, the more negative data points will be misclassified.

To combine the FPR and the TPR into one single metric, we first compute the two-former metrics with many different threshold, then plot them on a single graph, with the FPR values on the abscissa and the TPR values on the ordinate. The resulting curve is called ROC curve, and the metric we consider is the AUC of this curve, which we call AUROC.
The following figure shows the AUROC graphically:



In this figure, the blue area corresponds to the Area Under the curve of the Receiver Operating Characteristic (AUROC). The dashed line in the diagonal we present the ROC curve of a random predictor: it has an AUROC of 0.5. The random predictor is commonly used as a baseline to see whether the model is useful.

Investor ML, is particularly interested in predicting if the borrower will get charged off i.e. probability of "Risk Rate %". It would seem that using accuracy as a metric for evaluating a particular model's performance would be appropriate.

$$\textbf{Accuracy} = \frac{\sum \boldsymbol{True\ Positive} + \sum \boldsymbol{True\ Negative}}{\sum \boldsymbol{Total\ Population}}$$

Additionally, identifying borrower that fully pay back loan would be detrimental to *Investor ML*, since they are looking to invest on individual who will pay back loan. Therefore, a model's ability to precisely predict those who get charged off is more important than the model's ability to recall. We can use F-beta score as a metric that considers both precision and recall:

$$\mathbf{F\beta} = (1 + \mathbf{\beta}2) \cdot \frac{\mathbf{precision} \cdot \mathbf{recall}}{(\mathbf{\beta}2 \cdot \mathbf{precision}) + \mathbf{recall}}$$

In particular, when β=0.5, more emphasis is placed on precision. This is called the **F**0.5 score (or F-score for simplicity).

We will use Accuracy, F-score metric to compare, measure the variance of the model and choose the final model and predict to “Risk Rate %”.

II. Analysis

Data Exploration

Missing Values

After analyzing all the missing values, decided to keep a threshold of 40% of column population to analyze the data. Original dataset had Dataset has 42542 data points with 151 variables each. After applying threshold of 40% column population dataset changed and dataset has 42542 data points with 60 variables each. More discussion and strategies of handling missing values is discussed on data processing section.

Additional removals

Some identifier features like id, urls are not needed and removed. Based on profiling analysis, remove following features

- Dataset has 3 duplicate rows, lets remove duplicates
- application_type all of them are individual except missing 7
- collections_12_mths_ex_med has all the value 0.0 except missing 152
- fico_range_high is highly correlated to fico_range_low, so let’s remove fico_range_high feature
- initial_list_status has f as value excluding missing 7
- hardship_flag is having all same values as 'N' except missing data
- out_prncp has all feature values as 0.0 except missing values
- out_prncp_inv has all feature values 0.0 except missing values
- policy_code has all feature values 1 except missing values
- tax_liens as all values are 0.0 except 1 and others are missing
- title, purpose have similar values like debt consolidation et al, so remove title use only purpose
- desc is verbose text and some of the desc is like title or purpose. Not using verbose text to analyze so remove

After above step we are left with features dataset that has 42542 data points with 41 variables each.

Data Profiling

Panda profiling was used to do the basic data exploration, following are the details of data exploration.

Dataset info

Number of variables	41
Number of observations	42542
Total size in memory	13.3 MiB
Average record size in memory	328.0 B

Variables types

Numeric	16
Categorical	20
Boolean	4
Date	0
Text (Unique)	0
Rejected	1
Unsupported	0

- acc_now_delinq is highly skewed (γ1 = 103.07) Skewed
- acc_now_delinq has 36 / 100.0% missing values Missing
- addr_state has 7 / 100.0% missing values Missing
- addr_state has a high cardinality: 51 distinct values Warning
- annual_inc is highly skewed (γ1 = 29.035) Skewed
- annual_inc has 11 / 100.0% missing values Missing
- application_type has 7 / 100.0% missing values Missing
- collections_12_mths_ex_med has 152 / 100.0% missing values Missing
- delinq_2yrs has 36 / 100.0% missing values Missing
- delinq_amnt is highly skewed (γ1 = 206.16) Skewed
- delinq_amnt has 36 / 100.0% missing values Missing
- desc has 13299 / 100.0% missing values Missing
- desc has a high cardinality: 28965 distinct values Warning
- dti has 7 / 100.0% missing values Missing
- earliest_cr_line has 36 / 100.0% missing values Missing
- earliest_cr_line has a high cardinality: 531 distinct values Warning
- emp_length has 7 / 100.0% missing values Missing
- emp_title has 2631 / 100.0% missing values Missing
- emp_title has a high cardinality: 30660 distinct values Warning
- fico_range_high is highly correlated with fico_range_low (ρ = 1) Rejected
- fico_range_low has 7 / 100.0% missing values Missing
- grade has 7 / 100.0% missing values Missing
- hardship_flag has 7 / 100.0% missing values Missing
- home_ownership has 7 / 100.0% missing values Missing
- initial_list_status has 7 / 100.0% missing values Missing
- inq_last_6mths has 36 / 100.0% missing values Missing
- int_rate has 7 / 100.0% missing values Missing
- int_rate has a high cardinality: 395 distinct values Warning
- last_credit_pull_d has 11 / 100.0% missing values Missing
- last_credit_pull_d has a high cardinality: 128 distinct values Warning
- last_fico_range_high has 7 / 100.0% missing values Missing
- last_fico_range_low has 7 / 100.0% missing values Missing
- loan_amnt has 7 / 100.0% missing values Missing
- loan_status has 7 / 100.0% missing values Missing
- open_acc has 36 / 100.0% missing values Missing
- out_prncp has 7 / 100.0% missing values Missing
- out_prncp_inv has 7 / 100.0% missing values Missing
- policy_code has 7 / 100.0% missing values Missing
- pub_rec has 36 / 100.0% missing values Missing
- pub_rec_bankruptcies has 1372 / 100.0% missing values Missing
- purpose has 7 / 100.0% missing values Missing
- revol_bal has 7 / 100.0% missing values Missing

- revol_util has 97 / 100.0% missing values Missing
- revol_util has a high cardinality: 1120 distinct values Warning
- sub_grade has 7 / 100.0% missing values Missing
- tax_liens is highly skewed (γ1 = 205.99) Skewed
- tax_liens has 112 / 100.0% missing values Missing
- term has 7 / 100.0% missing values Missing
- title has 19 / 100.0% missing values Missing
- title has a high cardinality: 21258 distinct values Warning
- total_acc has 36 / 100.0% missing values Missing
- verification_status has 7 / 100.0% missing values Missing
- zip_code has 7 / 100.0% missing values Missing
- zip_code has a high cardinality: 838 distinct values Warning
- Dataset has 6 duplicate rows Warning

Basic intuitions formed based on the profiling of the data.
I have reviewed each and every feature its Quantile statistic, Descriptive statistic, Histogram, Common Values, extreme values, I show one feature Dti and its values below, for each feature refer loans_data_profiling.html, go to feature and toggle details.

Feature Analysis:

dti

Numeric

Distinct count	2895
Unique (%)	0.0%
Missing (n)	7
Infinite (%)	0.0%
Infinite (n)	0
Mean	13.373
Minimum	0
Maximum	29.99
Zeros (%)	0.0%

- Statistics

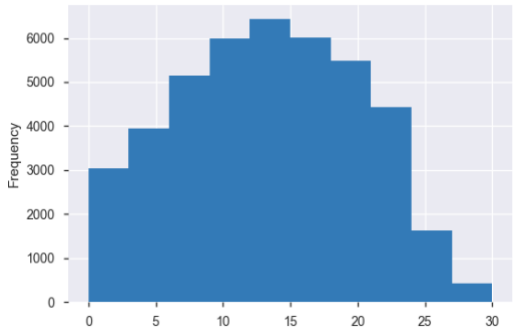
Quantile statistics

Minimum	0
5-th percentile	2.1
Q1	8.2
Median	13.47
Q3	18.68
95-th percentile	23.92

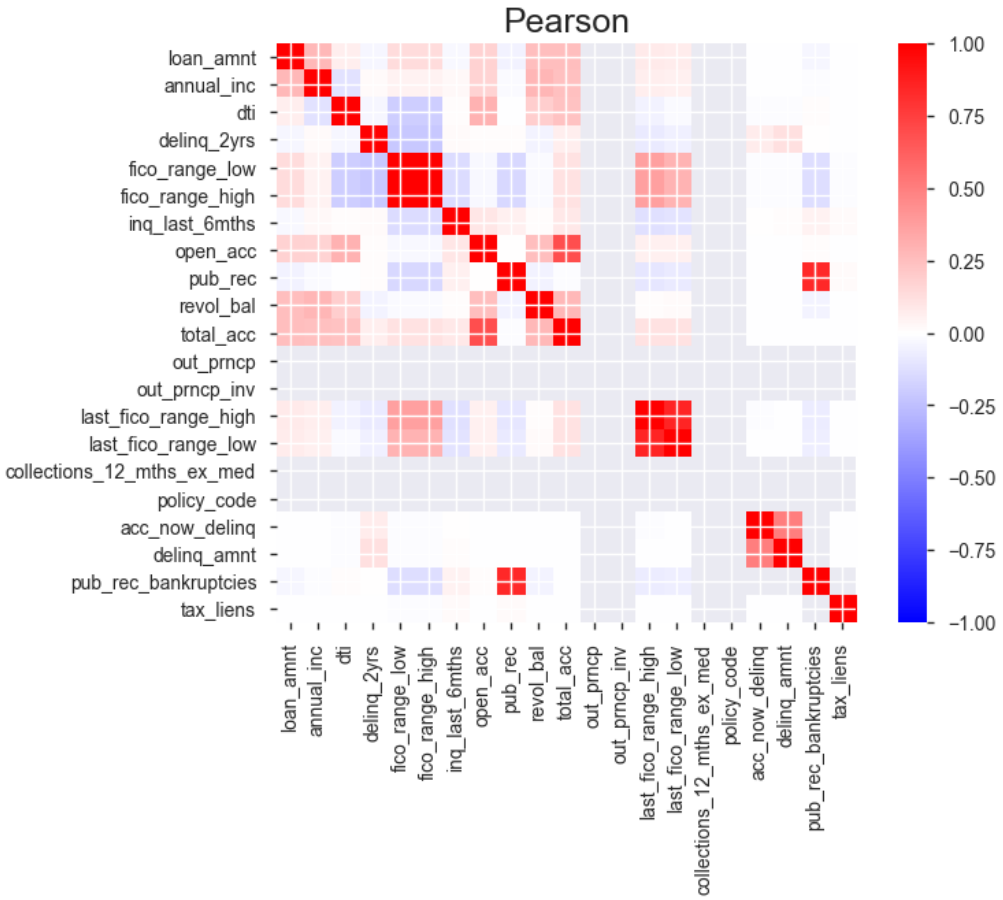
Maximum	29.99
Range	29.99
Interquartile range	10.48

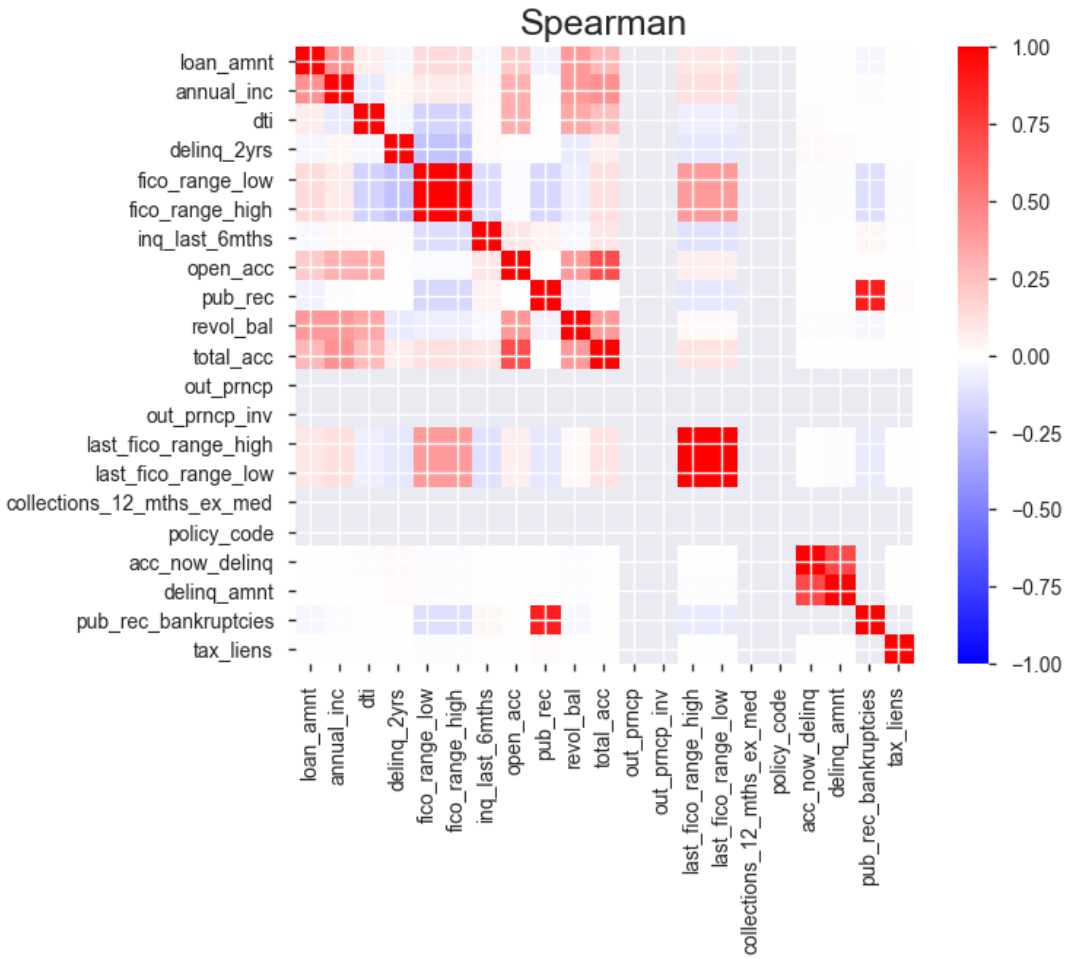
Descriptive statistics	6.7263
Standard deviation	
Coef of variation	0.50298
Kurtosis	-0.85174
Mean	13.373
MAD	5.6401
Skewness	-0.029922
Sum	568820
Variance	45.243
Memory size	332.4 KiB

- Histogram



Correlation plots for features:





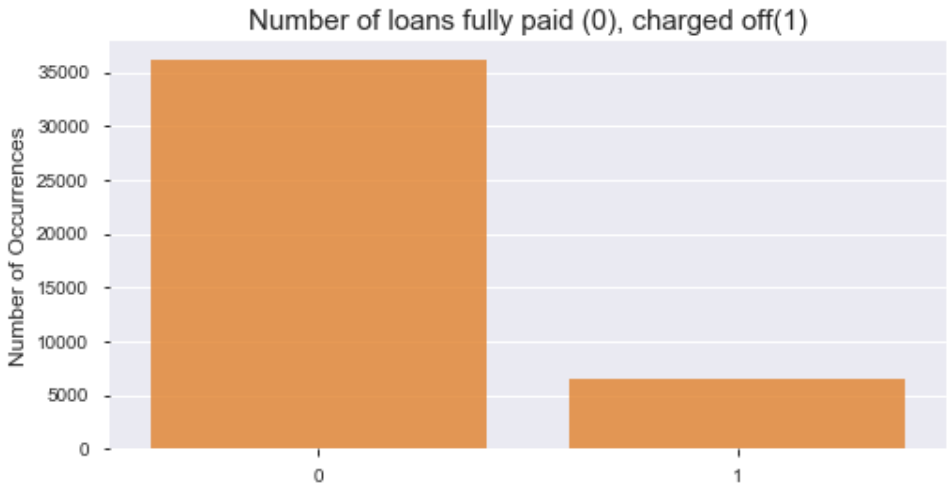
Preparing Target Label

loan_status is converted as target response variable for Investor ML classification problem. Following are the value counts of loan_status field.

Loan_status	Count
Fully Paid.	34116
Charged Off.	5670
Does not meet the credit policy. Status:Fully ...	1988
Does not meet the credit policy. Status:Charge...	761

For our project we just need to know whether loans are charged off or not. We will convert the Fully Paid & Does not meet the credit policy. Status:Fully Paid to 0, Charged Off & Does not meet the credit policy. Status:Charged Off to 1. After doing this change

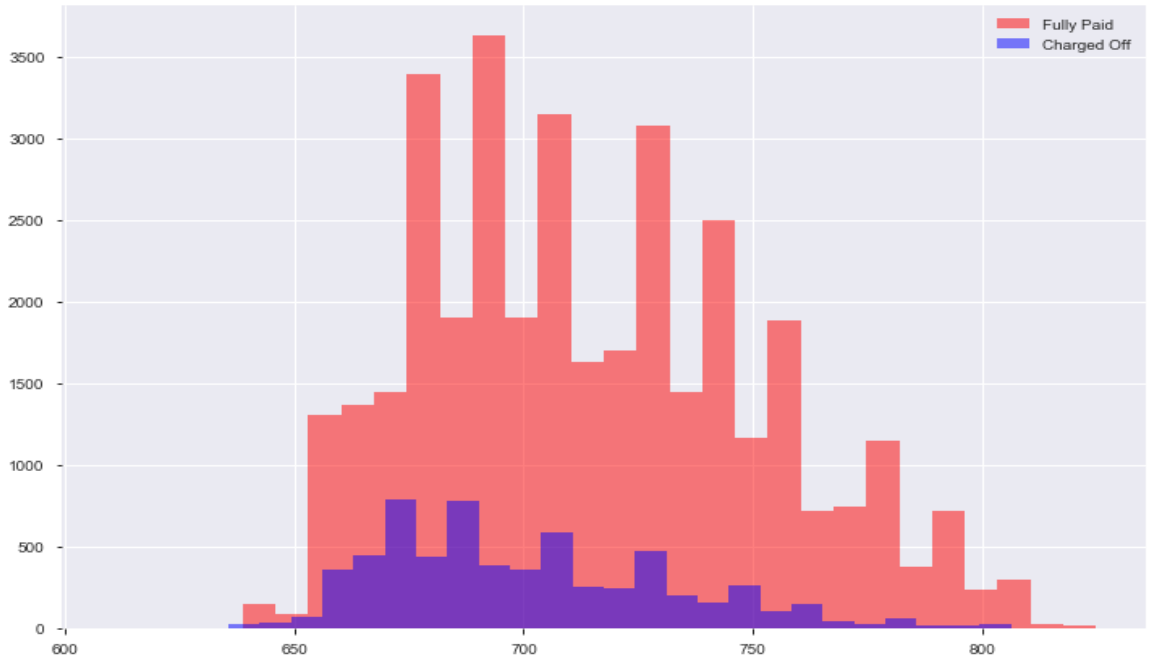
Loan_status	Count
0	36104
1	6431



We see that data set is pretty imbalanced as expected where positive examples (“charged off”) are only ~18%. We will handle imbalanced class issue later.

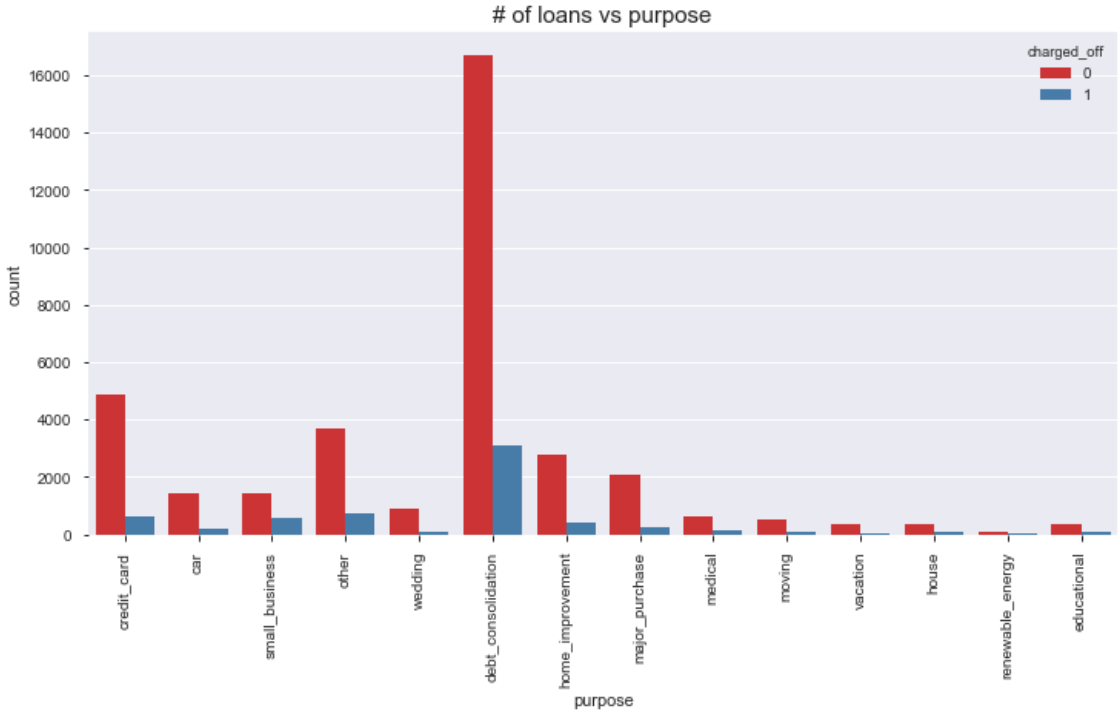
Exploratory Visualization

Analysis of loans based on fico score and whether they are charged off or fully paid. FICO score is key metric credit rating used to learn about the borrower credit, this analysis will help us understand how FICO score relates to borrower ability to fully pay or charged off.



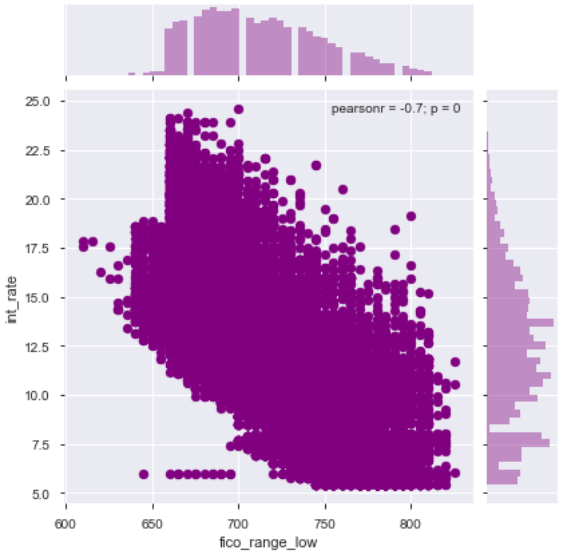
- Though loan applicants have good FICO score, 750+, we can see some loans are not paid off.
- Highest number of loans paid off are in b/w below 660 to 700 FICO score.

Why do borrowers take loan? Using feature purpose, we can plot and see purpose stated by borrower, and how many of the stated purpose loans are charged off or fully paid.



- Most of the loans are being requested for debt_consolidation
- Most not fully paid loans are also of purpose debt_consolidation

How does interest rates correlate to FICO score? Using feature interest rate and FICO, we can plot Pearson correlation plot, which will show how the interest rates are correlated to FICO score.



- Higher the fico score, lower the interest rate, there is a correlation b/w fico score & interest rate.

Algorithms and Techniques

Investor ML is supervised learning, binary classification problem. We will use various Ensemble methods for our classifier. Ensemble methods can be defined as combining several different models (base learners) into final model (meta learner) to reduce the generalization error. It relies on the assumption that each model would look at a different aspect of the data which yield to capturing part of the truth. Combining good performing models, that were trained independently will capture more of the truth than a single model. Therefore, this would result in more accurate predictions and lower generalization errors.

- Almost always ensemble model performance gets improved as we add more models.
- Try to combine models that are as much different as possible. This will reduce the correlation between the models that will improve the performance of the ensemble model that will lead to significantly outperform the best model. In the worst case where all models are perfectly correlated, the ensemble would have the same performance as the best model and sometimes even lower if some models are very bad. As a result, pick models that are as good as possible.

Different ensemble methods construct the ensemble of models in different ways. Below are the most common methods:

- Blending: Averaging the predictions of all models.
- Bagging: Build different models on different datasets and then take the majority vote from all the models. Given the original dataset, we sample with replacement to get the same size of the original dataset. Therefore, each dataset will include, on average, 2/3 of the original data and the rest 1/3 will be duplicates. Since each model will be built on a different dataset, it can be seen as a different model. For example: *Random Forest* improves default bagging trees by reducing the likelihood of strong features to picked on every split. In other word, it reduces the number of features available at each split from *n* features to, for example, *n/2* or *log(n)*features. This will reduce correlation –> reduce variance.
- Boosting: Build models sequentially. That means each model learns from the residuals of the previous model. The output will be all output of each single model weighted by the learning rate λ. It reduces the bias resulted from bagging by learning sequentially from residuals of previous trees (models).
- Stacking: Build k models called base learners. Then fit a model to the output of the base learners to predict the final output.

We'll be using Random Forest (bagging), Gradient Boosting (boosting), XGBoosting (boosting) and Logistic Regression classifiers as base learners.

Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

The implementation of logistic regression in scikit-learn can be accessed from class **LogisticRegression**. This implementation can fit binary, is used for *Investor ML* logistic regression with optional L2 or L1 regularization.

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Similarly, L1 regularized logistic regression solves the following optimization problem

$$\min_{w,c} ||w||_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Random Forest

In random forests (see **RandomForestClassifier** classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging,

its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.
The scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

XGboostClassifier
XGBoost is an optimized distributed gradient boosting system designed to be highly **efficient**, **flexible** and **portable**. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. Xgboost module in python has a sklearn wrapper called XGBClassifier which we will use.

GradientBoostingClassifier
Gradient Tree Boosting or Gradient Boosted Regression Trees (GBRT) is a generalization of boosting to arbitrary differentiable loss functions. GBRT is an accurate and effective off-the-shelf procedure that can be used for classification problems. Gradient Tree Boosting models are used in a variety of areas including Web search ranking and ecology.
The advantages of GBRT are:

- Natural handling of data of mixed type (= heterogeneous features)
- Predictive power
- Robustness to outliers in output space (via robust loss functions)

The disadvantages of GBRT are:

- Scalability, due to the sequential nature of boosting it can hardly be parallelized.

Modeling would be consisted of following parts:

- Strategies to deal with missing values. Discussed later in details in Data processing section.
- All identified numerical features are scaled and categorical variables are one-hot encoded, details discussed in Data processing section.
- Strategies to deal with imbalanced datasets. Discussed later in details in Implementation section.
- Build ensemble models. Discussed later in details in Implementation and refinement section.
- Model will be chosen based on runtime, metrics discussed Metrics section.
- Entire data will be fit and final model will be saved, final saved model can be used to infer “Risk Rate %” of a new borrower loan application to predict loan_status being charged off.

Benchmark

Looking at the distribution of classes (those who get charged off, and those who pay back), it's clear most individuals fully pay back loan. Suppose if we were building model to predict whether an individual loan will be pay back. This can greatly affect accuracy, since we could simply say "this person will pay back loan" and generally be right, without ever looking at the data! Making such a statement would be called naive, since we have not considered any

information to substantiate the claim. It is always important to consider the naive prediction for your data, to help establish a benchmark for whether a model is performing well. That been said, using that prediction would be pointless: If we predicted all people pay back loan, *Investor ML* would identify all loan application as fully payable loans and would not answer or help us find “Risk Rate %” as all loans are paid back.

Benchmark metrics of Naïve Predictor:
Looking at the distribution of classes (those who charged off, and those who fully pay), it's clear most individuals fully pay back loan. This can greatly affect accuracy, since we could simply say "this person will pay back loan" and generally be right, without ever looking at the data! Making such a statement would be called naive, since we have not considered any information to substantiate the claim. Also, as we are trying to predict "Risk Rate%", we can’t say loan gets charged off. To define a naive predictor, we will use Gausian NB model to help establish a benchmark predictor. Using this as benchmark, we can use all other linear or ensemble models to compare to say whether a model is performing well or not.

Gaussian Naive Bayes
GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

Model Name	Accuracy	F-score
Naive Predictor	0.557071	0.2567

Going through the metrics of Naïve Predictor we can say that it has poor accuracy of 0.5570 & bad f-score of 0.2567.

III. Methodology

Data Preprocessing

Feature selection based on intuition

- Based on above profiling, by spending some time, going through each feature and analyzing each feature, we can first identify the categorical features & numerical features that are populated well to be used for modeling.
- Pandas profiling also profiles the details of highly correlated features using which we don’t need to select some features if they are highly co-related, for example, fico_range_high & fico_range_low, they both are highly co-related so we can use one of them for our analysis and modeling.

Selected categorical features, intuitions for selection
- addr_state: Not selected, as this feature as high cardinality, has 51 distinct values, though the state might be a good indicator, am hesitant use this feature for first pass as using this one hot encoding features may increase by 51.
- earliest_cr_line: Not selected, should be converted to numerical by calculating the number of months from the loan date, not considered for first pass.
- emp_length: Selected, this feature might help us know duration of employment and how it influences paying off loan.
- emp_title: Selected, as purpose gives better indicator and also features will explode using this feature and one hot encoding.
- grade: Selected, LC grade might influence paying off loan.
- home_ownership: Selected, this feature might help us know how home_ownership influence paying off loan.
- int_rate: Selected, different interest rate might influence loan getting paid off, lower interest rate means less amount to be paid off, so loan may have been paid
- last_credit_pull_d: Not selected, converting the loan application month to last credit pull month might be helpful feature
- purpose: Selected, purpose might indicate the usage, which might corelate to loan being paid off.
- sub_grade: Selected, may be good indicator of paying off loan.
- term: Selected, may be longer term, less monthly commitment and may paying off loan.
- title: Not selected, purpose is better feature than title, too many categories to convert as one hot encode features.
- url: Not selected, url identifier.
- verification_status: Selected, indicator, better verification state, likely to pay off loan.
- zipcode: Not selected, indicates living neighborhood and changes of pay off loan. Too many zip codes to convert as one hot encode features.

Selected numerical features, intuitions for selection
- acc_now_delinq: Selected, considering this feature, as once we add more data, it might be valuable.
- annual_inc: selected, income informs how good the loan applicant has the ability to pay back.
- delinq_2yrs: selected, if applicant delinquency in past 2 yr may be good indicator.
- delinq_amnt: Not selected, as most of the values is 0, 36missing and 2 other values may.
- dti: Selected, dti may be a good indicator of borrower's debt to income ration and informs the ability of applicant to pay back loan.
- fico_range_high : Not selected, as its highly correlated to fico_range_low, which will be used.
- fico_range_low: Selected, it might tell credit rating of applicant, which by itself is a good measure of loan applicant ability to pay back.
- inq_last_6mths: Selected, shows multiple credit enquire, which may indicate pursuance of loan applicant for various debts.
- last_fico_range_high: Not selected, as it is highly correlated to last_fico_range_low and last_fico_range_low will be used.

- last_fico_range_low: Selected, this feature might help us know how loan applicant maintained his credit.
- loan_amnt: Selected, higher the loan_amnt, less likely loan will be paid off.
- mths_since_last_delinq: Selected, more month's good loan applicant.
- open_acc: Selected, Median open acc is better.
- pub_rec: Selected, derogatory public records, less likely to pay off loan.
- pub_rec_bankruptcies: Selected, more/any bankruptcy less likely to pay off loan.
- revol_bal: Selected, more revol_bal, likely to pay off loan.
- revol_util: Selected, more revol_util, likely to pay off loan.
- tax_lines: Not selected, all 42429 of tax_lines is 0, and 1 is 1 and 112 missing.
- total_acc: Selected, median account is a good indicator.

Missing Values

During data exploration we learnt that there are missing values in the dataset. We have to treat missing values differently for both numerical features and categorical features.

1. **Handling numerical features missing values**

   Following are missing numerical features. Dataset has 42535 data points with 15 numerical variables each.

   acc_now_delinq 29
   annual_inc 4
   delinq_2yrs 29
   int_rate 0
   dti 0
   fico_range_low 0
   inq_last_6mths 29
   last_fico_range_low 0
   loan_amnt 0
   open_acc 29
   pub_rec 29
   pub_rec_bankruptcies 1365
   revol_bal 0
   revol_util 90
   total_acc 29

   There are several strategies to impute missing values like, median, mean and mode imputation. As all above features are sensitive information related to individual financials, imputing mean, median strategies might not be right, so it is better to impute 0 for NA values than mean, median or mode strategies. Let's impute missing values as 0.

2. **Handling categorical features missing values**

   Following are missing numerical features. Dataset has 42535 data points with 7 numerical variables each.

   emp_length 0
   grade 0

home_ownership 0
purpose 0
sub_grade 0
term 0
verification_status 0
No missing values in categorical features.

Normalizing Numerical Features
It is often good practice to perform some type of scaling on numerical features. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning. I will use StandardScaler to scale the features around the mean.

One-hot encoding of categorical features
Above we found that there are about 7 categorical features that are non-numeric. Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (called *categorical variables*) be converted. One popular way to convert categorical variables is by using the **one-hot encoding** scheme. One-hot encoding creates a *"dummy"* variable for each possible category of each non-numeric feature. For example, assume someFeature has three possible entries: A, B, or C. We then encode this feature into someFeature_A, someFeature_B and someFeature_C.

	someFeature		**someFeature_A**	**someFeature_B**	**someFeature_C**
0	B		0	1	0
1	C	----> **one-hot encode** ---->	0	0	1
2	A		1	0	0

After above preprocessing, we have dataset of 42535 records, 93 features, 1 response variable with value 1 or 0.

Implementation

Shuffle and Split Data
Now all categorical variables have been converted into numerical features, and all numerical features have been normalized. As always, we will now split the data (both features and their labels) into training and test sets. 80% of the data will be used for training and 20% for testing.

Strategies to deal with imbalanced datasets
Classification problems in most real-world applications have imbalanced data sets. In other words, the positive examples (minority class) are a lot less than negative examples (majority class). We can see that in spam detection, ads click, loan approvals, etc. In our example, the positive examples (people who charged off) were only ~18% from the total examples. Therefore, accuracy is no longer a good measure of performance for different models because if we simply predict all examples to belong to the negative class, we achieve 81% accuracy.

Better metrics for imbalanced data sets are AUC (area under the ROC curve) and f1-score. However, that's not enough because class imbalance influences a learning algorithm during training by making the decision rule biased towards the majority class by implicitly learns a model that optimizes the predictions based on the majority class in the dataset. As a result, we'll explore different methods to overcome class imbalance problem.

Under-Sample: Under-sample the majority class with or w/o replacement by making the number of positive and negative examples equal. One of the drawbacks of under-sampling is that it ignores a good portion of training data that has valuable information. In our example, it would lose around ~27000+ examples. However, it's very fast to train.

Over-Sample: Over-sample the minority class with or w/o replacement by making the number of positive and negative examples equal. We'll add around ~27000+ samples from the training data set with this strategy. It's a lot more computationally expensive than under-sampling. Also, it's more prune to overfitting due to repeated examples.

EasyEnsemble: Sample several subsets from the majority class, build a classifier on top of each sampled data, and combine the output of all classifiers. More details can be found here.

Synthetic Minority Oversampling Technique (SMOTE): It over-samples the minority class but using synthesized examples. It operates on feature space not the data space.

Following are the results of AUC obtained after employing different balancing techniques discussed above.
Original model's average AUC: 0.825141686061
Under-sampled model's average AUC: 0.844144632797
Over-sampled model's average AUC: 0.836151343324
EasyEnsemble model's average AUC: 0.872006664839
SMOTE model's average AUC: 0.830762241789
EasyEnsemble method has the highest 10-folds CV with average AUC = 0.872006664839. So, we use EasyEnsemble technique to handle the imbalanced class problem.

Creating a Training and Predicting Pipeline

A training and predicting pipeline that allows you to quickly and effectively train models using various sizes of training data and perform predictions on the testing data. We will also chart and generate metrics table to capture the running time, performance of models based on various performance metrics discussed earlier. As discussed in Algorithms & Techniques section, using training & predictive pipeline, we will evaluate

- Logistic Regression (linear classifier)
- GradientBoostingClassifier (Ensemble boosting decision tree classifier)
- XGBoostClassifier (Ensemble boosting parallel decision tree classifier)

XGBClassifier trained on 340 samples.

XGBClassifier trained on 3402 samples.
XGBClassifier trained on 34028 samples.
LogisticRegression trained on 340 samples.
LogisticRegression trained on 3402 samples.
LogisticRegression trained on 34028 samples.
GradientBoostingClassifier trained on 340 samples.
GradientBoostingClassifier trained on 3402 samples.
GradientBoostingClassifier trained on 34028 samples.

	model	matthews_corrcoef	roc_auc_score	precision_score	recall_score	f1_score	accuracy	fscore
1	Naive Predictor	0.201287	0.696211	0.220243	0.75972	0.341489	0.557071	0.2567
2	XGBClassifier340	0.441011	0.858439	0.367037	0.846812	0.512109	0.756083	0.413943
3	XGBClassifier3402	0.450757	0.87416	0.370445	0.861586	0.51812	0.757729	0.418113
4	XGBClassifier34028	0.462447	0.884929	0.379299	0.866252	0.527587	0.765487	0.427344
5	LogisticRegression340	0.376855	0.824225	0.360721	0.699844	0.476065	0.767133	0.399432
6	LogisticRegression3402	0.420559	0.848994	0.376618	0.769051	0.505624	0.772658	0.419423
7	LogisticRegression34028	0.459705	0.870451	0.405491	0.792379	0.536457	0.792994	0.449374
8	GradientBoostingClassifier340	0.418919	0.843553	0.340653	0.868585	0.489376	0.72599	0.387793
9	GradientBoostingClassifier3402	0.446944	0.871031	0.370395	0.852255	0.516372	0.758669	0.417619
10	GradientBoostingClassifier34028	0.464679	0.884464	0.380757	0.867807	0.529286	0.766663	0.428901

We can see from the above charts and data that, EasySampling, LogisticRegression performs better than, XGBoostClassifier & GradientBoostingClassifier. Considering running time, 100% training size data, variance (train / test of accuracy score is close), F-score.

Refinement

Model Tuning for hyper parameters

From above implementation section it is clear that using Easy Sampling and LogisticRegression gives the best performance of the model. But we have not tuned any hyper parameters of the Logistic Regression or Easy Samplings BalancedBaggingClassifier following are the hyper parameters selected to tune.

Logistic Regression

solvers (['newton-cg', 'lbfgs', 'liblinear', 'sag']).

'C': [0.1, 1.0, 1.5]

BalancedBaggingClassifier

n-estimators : range(20,81,10)

Used GridSearch with Cross validation optimizing for custom fbeta_score found following tuned parameters from the best estimator as.

Logistic Regression

solvers (['sag']).

'C': [1.5]

BalancedBaggingClassifier

n-estimators : [10]

Following is the result of tuned model with tuned hyper parameters.

```
Unoptimized model ------
Accuracy score on testing data: 0.7931
F-score on testing data: 0.4495
Optimized Model ------
Final accuracy score on the testing data: 0.7936
Final F-score on the testing data: 0.4507

Best Classifier ------
BalancedBaggingClassifier(base_estimator=LogisticRegression(C=1.5,
class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1,
max_iter=100, multi_class='ovr', n_jobs=1, penalty='l1', random_state=10,
solver='liblinear', tol=0.0001, verbose=0, warm_start=False), bootstrap=True,
bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10,
n_jobs=1, oob_score=False, random_state=10, ratio='auto', replacement=False,
verbose=0, warm_start=False)
```

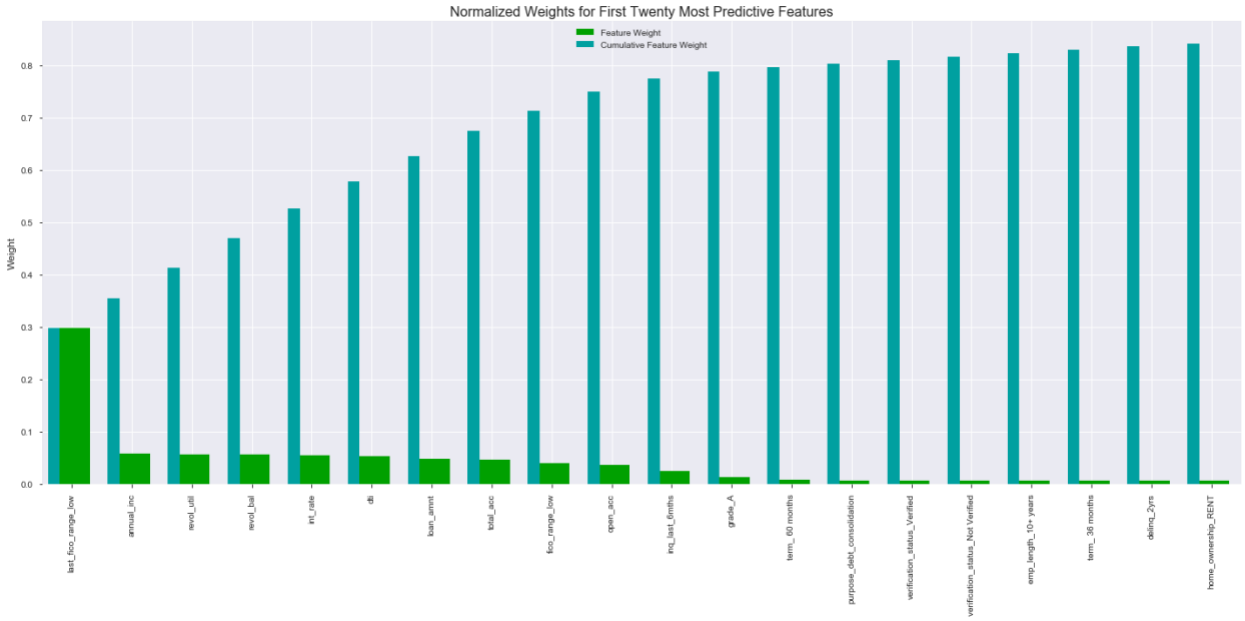
Updated performance metric for GridSearchTunedLR classifier

	model	matthews_corrcoef	roc_auc_score	precision_score	recall_score	f1_score	accuracy	fscore
1	Naive Predictor	0.201287	0.696211	0.220243	0.75972	0.341489	0.557071	0.2567
2	XGBClassifier340	0.441011	0.858439	0.367037	0.846812	0.512109	0.756083	0.413943
3	XGBClassifier3402	0.450757	0.87416	0.370445	0.861586	0.51812	0.757729	0.418113
4	XGBClassifier34028	0.462447	0.884929	0.379299	0.866252	0.527587	0.765487	0.427344
5	LogisticRegression340	0.376855	0.824225	0.360721	0.699844	0.476065	0.767133	0.399432
6	LogisticRegression3402	0.420559	0.848994	0.376618	0.769051	0.505624	0.772658	0.419423
7	LogisticRegression34028	0.459705	0.870451	0.405491	0.792379	0.536457	0.792994	0.449374
8	GradientBoostingClassifier340	0.418919	0.843553	0.340653	0.868585	0.489376	0.72599	0.387793
9	GradientBoostingClassifier3402	0.446944	0.871031	0.370395	0.852255	0.516372	0.758669	0.417619
10	GradientBoostingClassifier34028	0.464679	0.884464	0.380757	0.867807	0.529286	0.766663	0.428901
11	**GridSearchTunedLR**	**0.462095**	**0.870490**	**0.406598**	**0.795490**	**0.538138**	**0.793582**	**0.450661**

Feature importance

An important task when performing supervised learning on a dataset is determining which features provide the most predictive power. By focusing on the relationship between only a few

crucial features and the target label we simplify our understanding of the phenomenon, which is most always a useful thing to do. In the case of this project, that means we wish to identify a small number of features that most strongly predict whether an individual will be charged off or not. Using ensemble model RandomForestClassifier, we can find out the top twenty features,



Top 20 important features

Importances	**Features**
0.246455	last_fico_range_low
0.0572329	annual_inc
0.0544118	revol_util
0.0542158	dti
0.0523224	int_rate
0.0520493	revol_bal
0.0496855	loan_amnt
0.0466982	total_acc
0.042266	fico_range_low
0.0408149	open_acc
0.0281177	inq_last_6mths
0.00973854	term_ 36 months
0.0094579	verification_status_Not Verified
0.00941782	purpose_debt_consolidation
0.00903726	delinq_2yrs
0.00870967	term_ 60 months
0.00869922	home_ownership_MORTGAGE
0.00810152	emp_length_10+ years
0.00795508	verification_status_Verified
0.00786212	home_ownership_RENT

Feature selection of using top 20 features wasn't prudent in our case, so we will use the tuned best classifier with full dataset instead of top 20 feature

Final Model trained on full data

Final Model Train time 1.84824609756 s full data

Accuracy on testing data: 0.7936
F-score on testing data: 0.4507

Final Model trained on reduced to 81 features data

Final Model Train time 0.940053939819 s full data

Accuracy on testing data: 0.7921
F-score on testing data: 0.4472

Model Ensemble

We'll build ensemble models using four different models as base learners:

- LogisticRegression
- GradientBoostingClassifier
- XGBoostClassifier
- RandomForestClassifier

The ensemble models will be built using two different methods:

Blending (average) ensemble model. Fits the base learners to the training data and then, at test time, average the predictions generated by all the base learners. Use VotingClassifier from sklearn that: Fits all the base learners on the training data, at test time, use all base learners to predict test data and then take the average of all predictions.

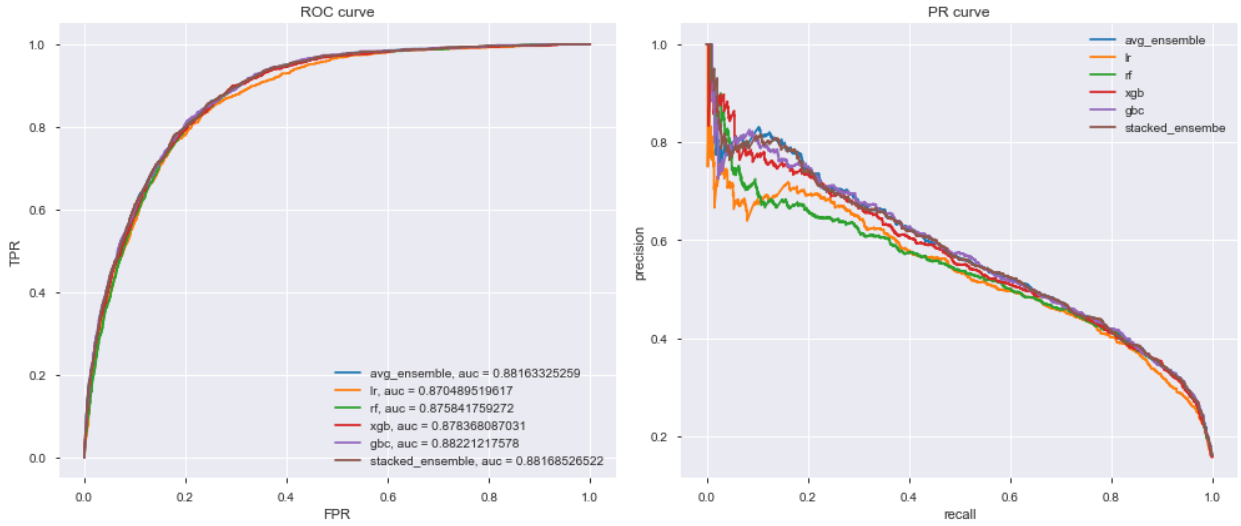
Stacked ensemble model: Fits the base learners to the training data. Next, use those trained base learners to generate predictions (meta-features) used by the meta-learner (assuming we have only one layer of base learners).

There are few different ways of training stacked ensemble model:
Fitting the base learners to all training data and then generate predictions using the same training data it was used to fit those learners. This method is more prune to overfitting because the meta learner will give more weights to the base learner who memorized the training data better, i.e. meta-learner won't generate well and would overfit. Split the training data into 2 to 3 different parts that will be used for training, validation, and generate predictions. It's a suboptimal method because held out sets usually have higher variance and different splits give different results as well as learning algorithms would have fewer data to train. Use k-folds cross validation where we split the data into k-folds. We fit the base learners to the (k -1) folds and use the fitted models to generate predictions of the held-out fold. We repeat the process until

we generate the predictions for all the k-folds. When done, refit the base learners to the full training data. This method is more reliable and will give models that memorize the data less weight. Therefore, it generalizes better on future data.

We'll use logistic regression as the meta-learner for the stacked model. Note that we can use k-folds cross validation to validate and tune the hyperparameters of the meta learner. We will not tune the hyperparameters of any of the base learners or the meta-learner; however, we will use some of the values recommended by the Data-driven advice for applying machine learning to bioinformatics problems and https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.

We will use ROC chart to choose the ensemble model. Observing the below ROC plotted generated from ensemble models fit for test data.



We can see from the chart above; stacked ensemble model has improved the performance.

Final model test METRICS
Accuracy on testing data: 0.8723
F-score on testing data: 0.5579
Accuracy: 87.23%
Recall: 39.27%
Precision: 62.35%
F1: 48.19%
[[6916 305]
[781 505]]

	precision	recall	f1-score	support
0	0.90	0.96	0.93	7221
1	0.62	0.39	0.48	1286

avg / total 0.86 0.87 0.86 8507

Updated performance metrics after ensemble model selection

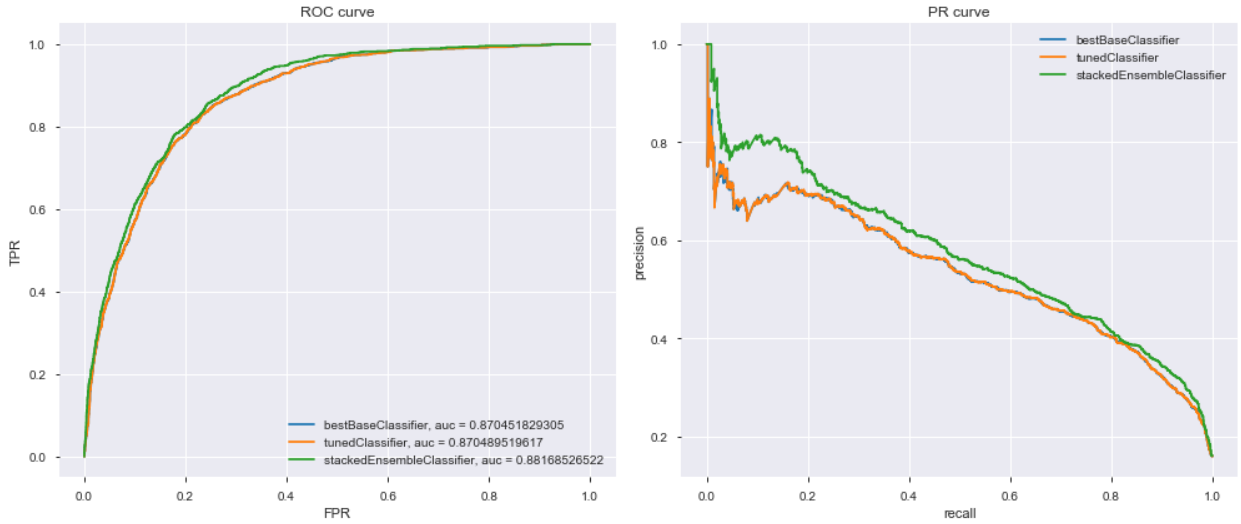
	model	matthews_corrcoef	roc_auc_score	precision_score	recall_score	f1_score	accuracy	fscore
1	Naive Predictor	0.201287	0.696211	0.220243	0.75972	0.341489	0.557071	0.2567
2	XGBClassifier340	0.441011	0.858439	0.367037	0.846812	0.512109	0.756083	0.413943
3	XGBClassifier3402	0.450757	0.87416	0.370445	0.861586	0.51812	0.757729	0.418113
4	XGBClassifier34028	0.462447	0.884929	0.379299	0.866252	0.527587	0.765487	0.427344
5	LogisticRegression340	0.376855	0.824225	0.360721	0.699844	0.476065	0.767133	0.399432
6	LogisticRegression3402	0.420559	0.848994	0.376618	0.769051	0.505624	0.772658	0.419423
7	LogisticRegression34028	0.459705	0.870451	0.405491	0.792379	0.536457	0.792994	0.449374
8	GradientBoostingClassifier340	0.418919	0.843553	0.340653	0.868585	0.489376	0.72599	0.387793
9	GradientBoostingClassifier3402	0.446944	0.871031	0.370395	0.852255	0.516372	0.758669	0.417619
10	GradientBoostingClassifier34028	0.464679	0.884464	0.380757	0.867807	0.529286	0.766663	0.428901
11	**GridSearchTunedLR**	**0.462095**	**0.870490**	**0.406598**	**0.795490**	**0.538138**	**0.793582**	**0.450661**
12	**Stacked Ensemble Final Model**	**0.427706**	**0.881685**	**0.623457**	**0.392691**	**0.481870**	**0.872340**	**0.557888**

IV. Results

Model Evaluation and Validation

Let's check best base classifier, tuned base classifier and final stacked model. Compare their results for various different set of inputs, to validate robustness and obtain their outputs. Also plot the ROC curves to perform sensitivity analysis.

From the above chart we can observe that though stackedEnsembleClassifier is high cost time in training and predict, it is better in terms of variance and better performance metrics of accuracy & F-score. We can also see that model is robust for varied inputs. ROC chart shows the sensitivity analysis for the stackedEnsembleClassifier.



Above ROC plot shows the sensitivity analysis and stackedEnsembleClassifier with AUC 0.88168 is better than bestBaseClassifier and tunedClassifier.

Justification

By looking at the metrics we discussed in Metrics section, for naïve predictor, unoptimized model, optimized model and final model (stackedEnsembelClassifier),

Metric	Benchmark Predictor	Unoptimized Model	Optimized Model	Stacked Ensemble Final Model
Accuracy Score	0.557071	0.793112	0.793582	0.87234
F-score	0.2567	0.449532	0.450661	0.55788

It is clear that Stacked Ensemble Final Model proves out to be the best model for *InvestorML.* We can also look at the additional metric to validate our results.

Metric	Benchmark Predictor	Unoptimized Model	Optimized Model	Stacked Ensemble Final Model
matthews_corrcoef	0.201287	0.459866	0.462095	0.427706
roc_auc_score	0.696211	0.870452	0.87049	0.881685
precision_score	0.220243	0.405653	0.406598	0.623457
recall_score	0.75972	0.792379	0.79549	0.392691
f1_score	0.341489	0.536598	0.538138	0.48187

accuracy	0.557071	0.793112	0.793582	0.87234
fscore	0.2567	0.449532	0.450661	0.557888

- In terms of accuracy, F-score Stacked Ensemble Final Model is much better.
- The Stacked Ensemble Final Model has larger accuracy and F-score compared to Benchmark Naive Predictor, unoptimized model, optimized model.

We will use Stacked Ensemble Final Model to predict whether a loan by individual will be charged off.

We will save the model using a joblib so that we can use this saved model, infer a new loan will be paid or not based on application data and use *Investor ML* in real time. Final model, we fit using all data available.

Test saved model, final model results on par with our best model, performance metric is increased due to increase in data as we used all the data available.
Accuracy on testing data: 0.8969
F-score on testing data: 0.6729

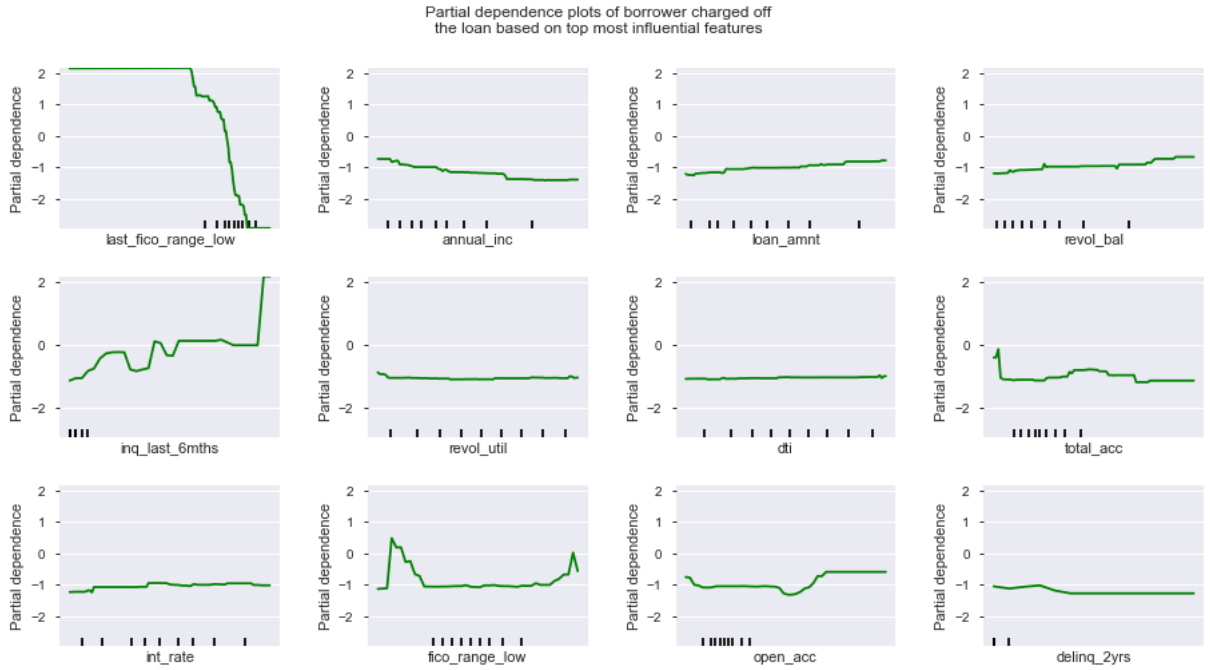
For any new loan application data of a new loan, features should be transformed to transformed final scaled 15 numerical features and transformed once hot encoded categorical features.
Using these new features as input to the saved model loaded, we can predict prob “Risk Rate%” for each loan.

“Risk Rate %” is the additional statistic available for investor to choose a particular loan for investment.

V. Conclusion

Free-Form Visualization

Partial dependence plots to see what the most important features and their relationships with are whether the borrower will most likely pay the loan in full before mature data. we will plot only the top 8 features to make it easier to read. Note that the partial plots are based on Gradient Boosting model.



As expected borrowers with lower annual income and less FICO scores are higly likely to get charged off; however, borrowers with lower interest rates (riskier) and higher revol_bal are more likely to pay the loan fully.

Reflection

Most classification problems in the real world are imbalanced. Also, almost always data sets have missing values. In this project, we covered strategies to deal with both missing values and imbalanced data sets. We also explored different ways of building ensembles which can give better performance. Below are some interesting learnings.

- There is no definitive guide of which algorithms to use given any situation. What may work on some data sets may not necessarily work on others. Therefore, always evaluate methods using cross validation to get a reliable estimate.
- Sometimes we may be willing to give up some improvement to the model if that would increase the complexity much more than the percentage change in the improvement to the evaluation metrics.
- EasyEnsemble usually performs better than any other resampling methods.
- Missing values sometimes add more information to the model than we might expect.
- Though important features have higher predictive power if runtime and performance is not much impacted we can use all features to get the best predictions from the model
- Using voting classifier or stacked ensemble might increase the complexity of the model, provided we are getting better predictions, it's better to use these advanced techniques to choose the best model.
- Note that loan application data of a new loan should be transformed to transformed final scaled 15 numerical features and transformed once hot encoded categorical

features. Using these new features as input to the final stacked ensemble model, we can predict prob "Risk Rate%" for each loan.
- "Risk Rate %" is the additional statistic available for investor to choose a particular loan for investment.

Improvement

Following are some of the additional improvements that can be done,

- In some classification problems, False Negatives are a lot more expensive than False Positives. Therefore, we can reduce cut-off points to reduce the False Negatives.
- When building ensemble models, try to use good models that are as different as possible to reduce correlation between the base learners. We could've enhanced our stacked ensemble model by adding Dense Neural Network and some other kind of base learners as well as adding more layers to the stacked model.
- Add binary features for each feature that has missing values to check if each example is missing or not.
- Use more data to train model, this will generalize by regularizing the model by learning about more input training data.
- Use deep learning rnn (LSTM) networks to evaluate if it provides better performance than the chosen model.
- Tune additional hyper parameters to see if the results are improved.
- Use hardware accelerator like GPU to improve the model training and inferring to reduce the run time of model.

VI. References

1. http://economics.ucr.edu/job_candidates/Bagherpour-Paper.pdf
2. http://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/tsmcb09.pdf
3. https://www.lendingclub.com/info/download-data.action
4. https://www.lendingclub.com
5. http://scikit-learn.org/stable/index.html
6. http://contrib.scikit-learn.org/imbalanced-learn/stable/api.html
7. https://github.com/pandas-profiling/pandas-profiling
8. http://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn
9. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
10. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/