Machine Learning Engineer Nanodegree

Capstone Project: Predicting Loan Defaults

Scott R Smith
March 9, 2019

I. Definition

Project Overview

Granting credit to borrowers has been a tricky business problem since the beginning of banking. Credit is usually evaluated and granted based upon risk of the loan being paid back. The level of risk both determines if credit is granted as well as the lending terms (interest rate, amount, etc.). Lending terms are calculated on default rates in a way so paid back loans cover defaulted loan costs and keep the financial institution profitable. Currently, lenders such as banks and credit unions, use a FICO score and borrower financial information to determine if the borrower is a credit risk and has the cash flow to pay the loan back.

The age of machine learning has brought about a new breed of lender categorized as FinTech. FinTech is non-bank entities lending online using fully automated processes. The Financial Services of the market is using Machine Learning. For example, the FICO score now uses ML to increase accuracy. Lending Club had a Kaggle competition where data scientists were invited to use Lending Club Data to predict loan defaults. (Top score was an AUC of 0.718)

There have been numerous studies using Machine Learning for credit scoring. This study "Benchmarking state-of-the-art classification algorithms for credit scoring" benchmarked 41 classifiers including Random Forest, Naive Bayes, Linear, Boosted, and k-nearest neighbor where they concluded that "heterogeneous ensembles classifiers perform...more accurately than industry standards". Another study, An experimental comparison of classification algorithms for imbalanced credit scoring data sets, looked at Logistic Regression, Decision Trees, SVM, and k-nearest neighbors. This study looked at logistic regression in credit scoring.

Machine Learning has greatly improved credit scoring as the studies show, with FinTechs such as OnDeck, Prosper, LendingClub and Kabbage are offering unsecured business and consumer loans based on advanced, proprietary, ML algorithms. One characteristic of these algorithms is they use supplemental data in addition to Machine Learning. For example, Kabbage uses UPS delivery data and social media as part of its credit score. OnDeck uses accounting data and social data in addition to traditional credit data. These supplemental features are used to increase prediction accuracy further.

The goal of this project is to predict if a borrower will default on the loan.

Project Files

Files for this project are located on GitHub: https://github.com/scottrsmith/Loan-Defaults

Datasets and Inputs

I will be using the Lending Club loan dataset, from 2007 to Q2 2018, posted on Kaggle. This is a very large dataset, almost 3 GB uncompressed, in two data files.
https://www.kaggle.com/wordsforthewise/lending-club

These two data files are accepted loans and rejected loans. Accepted loans have data for current active loans, completed loans (paid off) and defaulted loans. This dataset has the credit data used to make the initial, approved, credit decision, borrower information (state and zip code, loan purpose, etc.) as well as data on current payment status, default status, Lending Club scoring and loan terms. The second dataset is for rejected loans. This data set is of limited value since it does not contain all of the data to make the initial credit decision so it will not be used.

From the accepted loans file, we will only use data used for the initial credit decision, including credit score and borrower attributes and loan status (defaulted or completed). We will not use loans that are current. We will also limit our data to two years of data (2016 and 2017). This will make the file more manageable and help with feature engineering as we experiment with external data to improve accuracy.

I will also be exploring using external data to supplement existing demographics. This includes the cost of living by state[1] or zip code[2], salary information by job type[3], unemployment rates[4], etc.

Problem Statement

In this project, we will determine if an approved and funded borrower will default on a loan. We will use Lending Club data that contains information about the granted loans that have defaulted and loans paid back. We will not use datas on current loans since we do not know if these will default or not. All of these loans were run through the Lending Club credit algorithms, graded and approved. While most are paid off, some default. This is by design since a certain percentage of loans are expected to default.

[1] https://www.statista.com/topics/768/cost-of-living/,
https://www.numbeo.com/cost-of-living/region_rankings.jsp?title=2016&region=021
[2] https://www.irs.gov/statistics/soi-tax-stats-individual-income-tax-statistics-2016-zip-code-data-soi
[3] https://www.bls.gov/oes/2017/may/oes_nat.htm
[4] https://www.census.gov/econ/geo-zip.html

This is a supervised learning, binary classification problem with results that either predict default or not. Success will be measured by how well we predict default based upon a measure of the accuracy of our predictions. I will also use feature engineering and explore using 3rd party data to supplement the credit data to help with prediction.

This project will go through the following design and execution tasks.

1. Examine the data.
   For this project, I only want data that would be available in an application. Two reasons: One, we do not have complete original credit data for rejected applications. Second, we want to remove any Lending Club generated data especially data that is highly correlated to defaults (for example, higher interest rates correlate to default rates.)

2. Remove unneeded columns
   Remove columns related to post-loan acceptance. This consisted of highly correlated data to defaults as well as ongoing credit checks, late payment tracking, etc.

3. Prep data for initial, default training
   Clean the data. Removing or flag missing values, one-hot encode, remove out of range data, etc.

4. Train data for baseline
   Create the baseline for models. This will be based upon default settings for the various classification models. I will create a pipeline of the various ML Classification models including Logistic Regression (L1 and L2), Random Forest, Gradient Boosting, Decision Tree, KNeighbors, SGD, Bagging AdaBoost, and GaussianNB.

   From the initial training, pick the top 3 or 4 to further analyze

5. Feature Engineer
   This will include feature engineering with the existing data (re-bucketing, indicator variables), as well as trying to add external data. For example, the cost of living by state[5] or zip code[6], salary information by job type[7], unemployment rates[8], etc.

6. Run training and tune Hyperparameters
   As we feature engineer, I will track the changes to the scoring and run the training multiple times to gauge improvements (or not), to the default models. Training will be in multiple steps:

[5] https://www.statista.com/topics/768/cost-of-living/, https://www.numbeo.com/cost-of-living/region_rankings.jsp?title=2016&region=021
[6] https://www.irs.gov/statistics/soi-tax-stats-individual-income-tax-statistics-2016-zip-code-data-soi
[7] https://www.bls.gov/oes/2017/may/oes_nat.htm
[8] https://www.census.gov/econ/geo-zip.html

- Run the model defaults.
- Tune hyperparameters to get maximum results (based upon the metrics covered below)
- Explore Ensemble Methods (Bagging and AdaBoost) using different base estimators
- Explore Stacking

7. Evaluate results. Revisit steps 5 through 6 until the target evaluation metrics are exceeded.

Metrics

Credit Scores, like FICO, accurately predict delinquency rates. For FICO scores greater than 650 the average delinquency is around 4%[9]. The Lending Club data for 2016-2017 has an 8.3% default rate overall ALL of the data segments. Using the two target segments (paid vs defaulted) the number of loans was 140,379 paid vs 38,413 defaulted and calculates out as a 21.48% default rate. With these numbers, I will use 21% as the baseline for actual default rates and 8.3% for predicted to create target metrics.

From these numbers I get the following confusion matrix:

Confusion Matrix N = 178792	**Predicted Paid (0)**	**Predicted Default (1)**
Actual Paid (0)	140,379 (True -)	23,573 (False +)
Actual Default (1)	0 (False -)	14,840 (True +)

Note on the confusion matrix numbers. We had a total of 178,792 loans with actual default of 38,413. Giving 140K True negatives and 14,840 actual defaults, with 23,573 false positives.

Since this will be a classification problem, we want to look at precision and recall:

- Precision = Of the Defaults, how many were actual defaults
- Recall = Out of all the defaults calculated, how many were correct

The above confusion matrix gives the following scores:

Recall = 1.0,   Precision = 0.386,   F1 = 0.55,   f-beta(β=.25) = .44

These metrics (scores) above, are what we will use to gauge the effectiveness of our models. These scores are all based upon the confusion matrix above. For the confusion matrix, “each

[9] http://www.wvasf.org/presentation_pdfs/John_Meeks_-_WV_Asset_Building_Charleston_102811.pdf

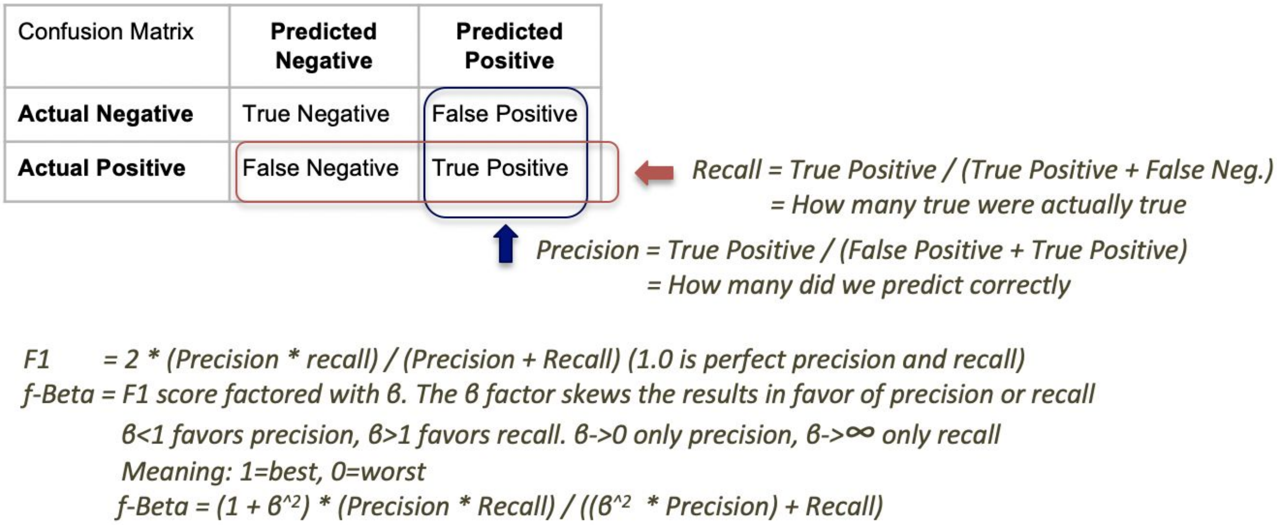
row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class”[10]

We will be focusing on Precision[11], f-beta[12], and AUC[13] scores for measuring the various Machine Learning models based upon the Confusion Matrix.

About these scores:

- Precision measures how well we predict defaults.
- Recall Measures out of all the defaults calculated, how many were correct.
- f-beta measures precision and recall together with the beta(β) allowing us to weight in favor of precision or recall (we will use a beta of .25 to weight towards precision)

Based upon the Confusion Matrix, this is how the selected scores are calculated:



Confusion Matrix	**Predicted Negative**	**Predicted Positive**
Actual Negative	True Negative	False Positive
Actual Positive	False Negative	True Positive

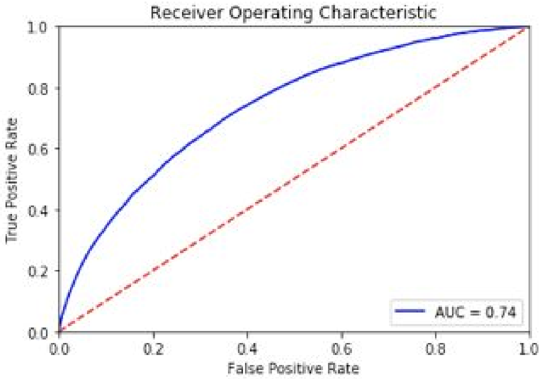
Recall = True Positive / (True Positive + False Neg.)
= How many true were actually true

Precision = True Positive / (False Positive + True Positive)
= How many did we predict correctly

*F1 = 2 * (Precision * recall) / (Precision + Recall) (1.0 is perfect precision and recall)*
f-Beta = F1 score factored with β. The β factor skews the results in favor of precision or recall
β<1 favors precision, β>1 favors recall. β->0 only precision, β->∞ only recall
Meaning: 1=best, 0=worst
*f-Beta = (1 + β^2) * (Precision * Recall) / ((β^2 * Precision) + Recall)*

We will also compute the Area Under the Curve (AUC). The curve is a graph of True Positive Rate vs the False Positive Rate. AUC measures the area under the curve. The more area under the curve, the better, with 1.0 being a perfect score. See Wikipedia for more details.

[10] https://en.wikipedia.org/wiki/Confusion_matrix
[11] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score
[12] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html#sklearn.metrics.fbeta_score
[13] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html#sklearn.metrics.auc



Area Under Curve (AUC).

In our example, the AUC of .74 measures everything under the blue curve. The red line is an AUC of .5 represening random guesses in our classification problem.

Lenders calculate default rates to understand risk exposure and price terms based upon the level of risk. For this, we are interested in higher precision. That is, we want to catch as many defaults as we can to maximize our lending. (We are OK with false positives, just as we are with false negatives.) The goal is to beat the precision of .386 and the f-beta score of .485. (Lending Club's best-guess performance). Using these numbers and the AUC benchmark score we have the following targets:

- AUC score > .7
- Precision score of > .386
- f-Beta score > .44 (β=.25)

The goal of this project is to beat all three metrics to match or beat the estimated Lending Club performance. (Keeping in mind that actual default rates are usually by design and the data will reflect this.)

II. Analysis

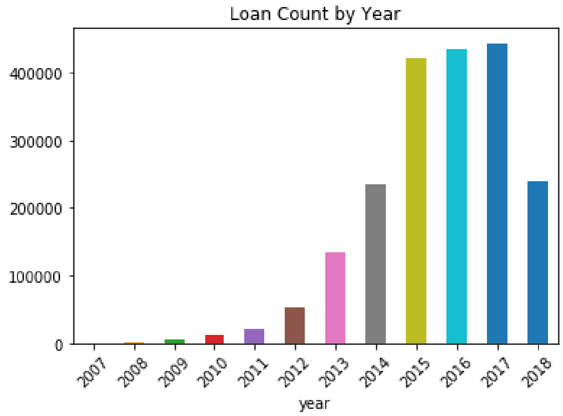
Data Exploration and Exploratory Visualization

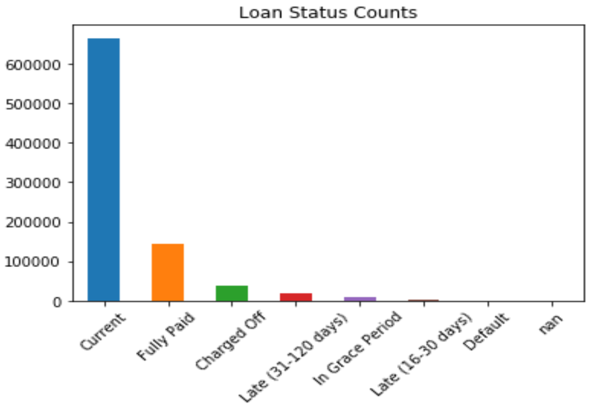
Overview

The Lending Club data set is a very large dataset consisting of 2,004,091 records from 2007 to Q2 of 2018 with a size of 2.3GB.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2004091 entries, 0 to 2004090
Columns: 151 entries, id to settlement_term
dtypes: float64(113), object(38)
memory usage: 2.3+ GB
```

For the amount of data, most of the data is in the years of 2015 through 2017, segmented by loan status (years 2016/2017 shown). The two key segments are 'loan date' and 'loan status'. These will be the two key segments we will use to prepare the data for model training.





Source Data Breakdown

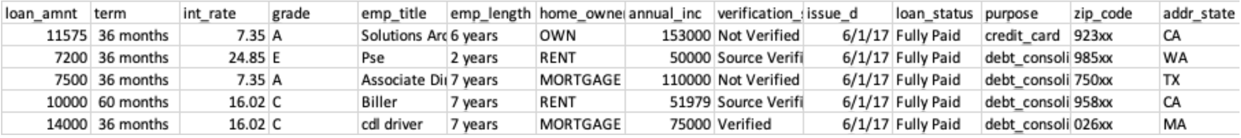
The data falls into 9 general categories based upon who and how it was generated. These include:

- Application Data: Data provided by the applicant
- Joint Account: The loan is for a joint account (co-borrower/signer)
- Second Application: Credit bureau data pulled for the co-signer
- Member Data: Lending Club generated
- Delinquent Loan Servicing: Lending Club generated data on loans past due
- On-going Credit Pull Data: Data pulled during loan servicing
- Ongoing Servicing and Collections: Lending Club data generated during servicing and/or collections
- Underwriting: Data generated by the Lending Club underwriting department
- Credit Bureau Data: Data pulled from credit bureau on the primary applicant

See "Appendix 5: Source Data Fields" for more detail

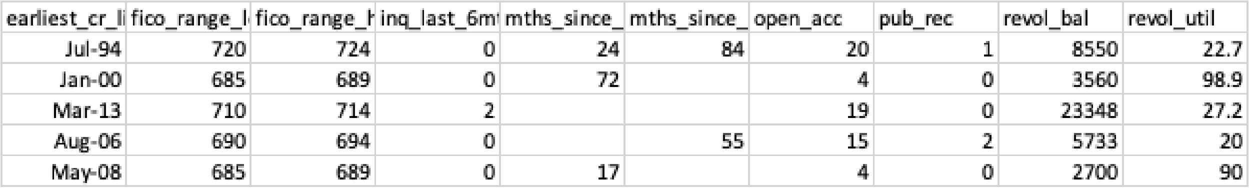
Data Sample

General Data:



loan_amnt	term	int_rate	grade	emp_title	emp_length	home_owne	annual_inc	verification_	issue_d	loan_status	purpose	zip_code	addr_state
11575	36 months	7.35	A	Solutions Arc	6 years	OWN	153000	Not Verified	6/1/17	Fully Paid	credit_card	923xx	CA
7200	36 months	24.85	E	Pse	2 years	RENT	50000	Source Verifi	6/1/17	Fully Paid	debt_consoli	985xx	WA
7500	36 months	7.35	A	Associate Di	7 years	MORTGAGE	110000	Not Verified	6/1/17	Fully Paid	debt_consoli	750xx	TX
10000	60 months	16.02	C	Biller	7 years	RENT	51979	Source Verifi	6/1/17	Fully Paid	debt_consoli	958xx	CA
14000	36 months	16.02	C	cdl driver	7 years	MORTGAGE	75000	Verified	6/1/17	Fully Paid	debt_consoli	026xx	MA

Credit Data:



earliest_cr_li	fico_range_l	fico_range_h	inq_last_6mt	mths_since_	mths_since_	open_acc	pub_rec	revol_bal	revol_util
Jul-94	720	724	0	24	84	20	1	8550	22.7
Jan-00	685	689	0	72		4	0	3560	98.9
Mar-13	710	714	2			19	0	23348	27.2
Aug-06	690	694	0		55	15	2	5733	20
May-08	685	689	0	17		4	0	2700	90

Data Characteristics of Training Data Set

The training data is a subset of the overall data. (limited to 2017-2018 and certain columns as outlined above with 178,792 records)

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 178792 entries, 699274 to 1896216
Data columns (total 81 columns):
```

Categorical Data

There are a number of categorical data points that need to be feature engineered, removed or one-hot encoded.

Manageable data sets (data that can be feature engineered or one-hot encoded if used):

- Term, grade, home ownership, verification status, purpose, employee length

Problem datasets include (Data sets that might be very hard to engineer):

- *Zip_code*, with only the first three digits of the zip code provided (897 unique values)
- *Emp_title* (employee title), title as provided by the applicant. There 58,000 unique values provided. This will be a hard set of data to feature engineer

Date-based

There are several variables that are dates that will need to be feature engineered:

- *Issue_d* (issue date of the loan)
- *Earliest_cr_line* (Date of the earliest credit line reporting)

Missing Values

There are a number of columns with missing values. These are all related to credit score data.

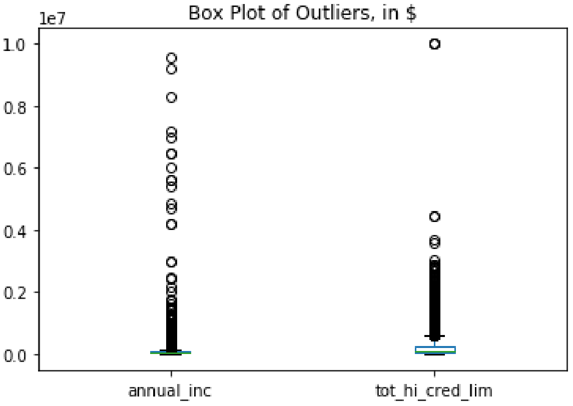
Columns Missing values:

inq_last_6mths,mths_since_last_delinq,mths_since_last_record,revol_util, Mths_since_last_major_derog,open_acc_6m,open_act_il,open_il_12m, mths_since_rcnt_il,total_bal_il,il_util,open_rv_12m,open_rv_24m,max_bal_bc, *inq_fi,inq_last_12m,avg_cur_bal,bc_open_to_buy,bc_util,mo_sin_old_il_acct, mths_since_recent_bc,mths_since_recent_bc_dlq,mths_since_recent_inq, Mths_since_recent_revol_delinq, num_accts_ever_120_pd,num_tl_120dpd_2m, percent_bc_gt_75,open_il_24m*

Outliers

Using Median Absolute Deviation (MAD)[14] for values in the 99.9% quantile, we get the following outliers:

- For annual_inc 15 outliers above 4200000.0
- For tot_coll_amt 2 outliers above 173380.0
- For acc_open_past_24mths 1 outliers above 56.0
- For mo_sin_rcnt_tl 2 outliers above 289.0
- For mort_acc 2 outliers above 37.0
- For num_actv_bc_tl 4 outliers above 29.0
- For num_sats 2 outliers above 80.0
- For num_tl_op_past_12m 4 outliers above 28.0
- For tot_hi_cred_lim 3 outliers above 9999999.0
- For total_bal_ex_mort 3 outliers above 1234429.0
- For total_bc_limit 1 outliers above 1105500.0
- For total_il_high_credit_limit 5 outliers above 1120706.0



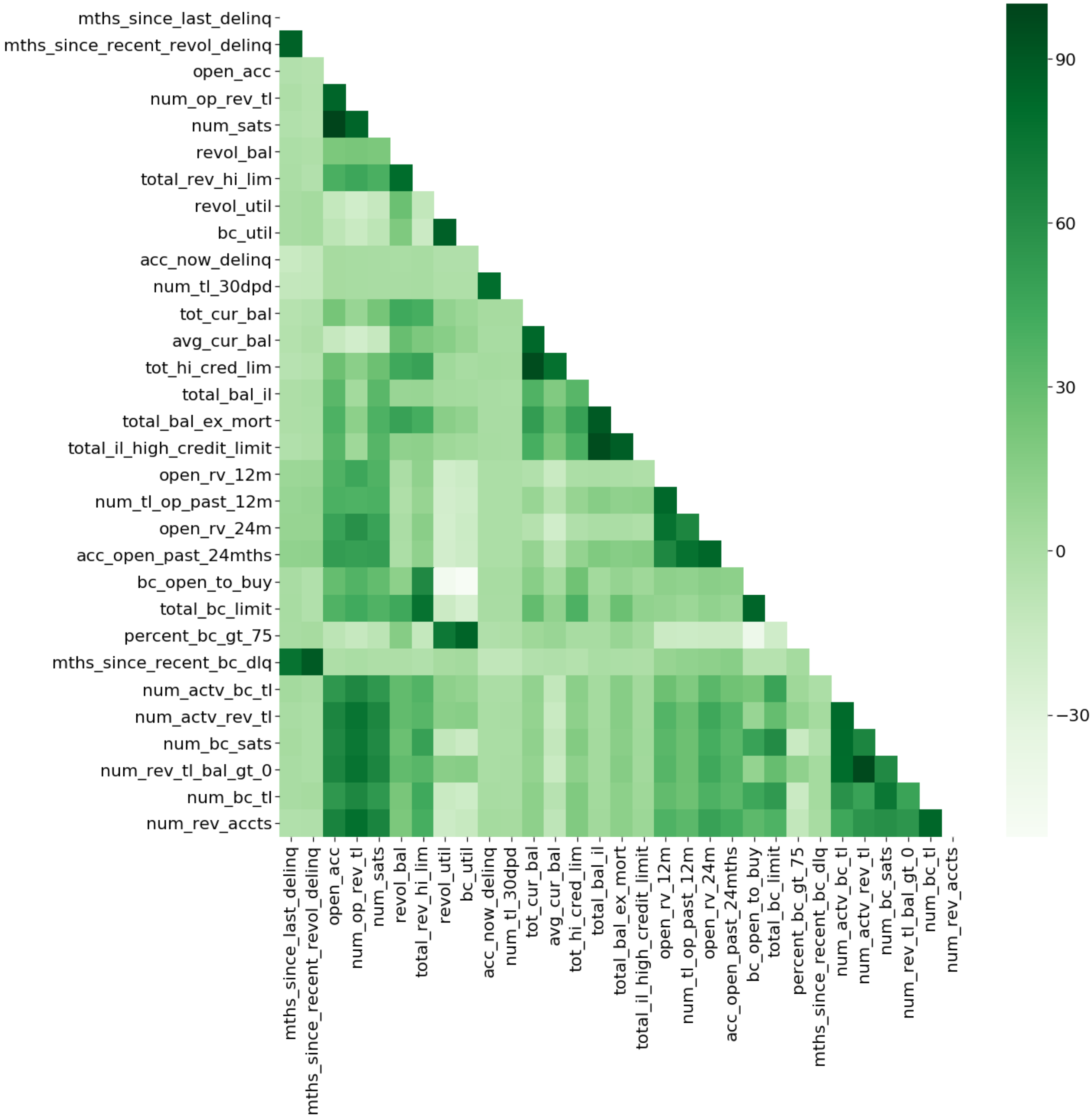
Note: Outliers were kept in the final models

Correlated Data

Correlation within the Credit Bureau data

There are some strong correlations within the data reported from the Credit Bureau. This is to be expected and no action needs to be taken.

[14] http://eurekastatistics.com/using-the-median-absolute-deviation-to-find-outliers/



Strong correlation between *fico_range_low* and *fico_range_high*

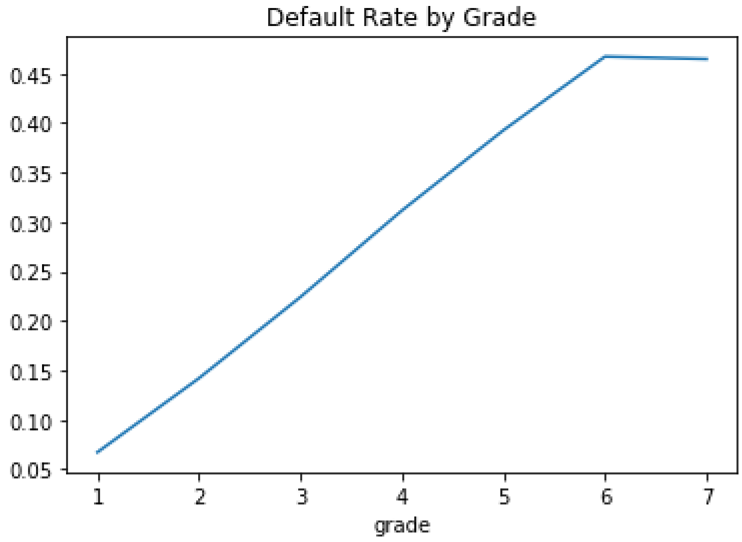
	fico_range_low	**fico_range_high**
fico_range_low	1.0	1.0

fico_range_high	1.0	1.0

Strong correlation between *grade* and *interest* rate

	int_rate	**grade**
int_rate	1.0	0.967013
grade	0.967013	1.0

Strong relationship between *grade* and *default rate*



Mean Grouped by *grade* and *loan_status*

Algorithms and Techniques

Algorithms

Predicting loan defaults is a binary classification problem. That is, will the borrower default or not. This is also a supervised learning problem, that is, we have historical data on loans that have been paid off or defaulted. There is a wide range of classification models we need to evaluate since anyone algorithm will not work for every problem. We will start with the following models for supervised classification problems than continue to refine these with Ensemble Methods.

Linear Models

Linear Models are a regression analysis where there is a linear relationship between independent and dependent variables. Linear Models are used for finding out the relationship between variables and forecasting (dependent variable) - thus finding a linear relationship between the two. For linear models, the cost function is an important concept since it measures the error between the predicted value and actual value. The error is the root mean squared error. The goal is to minimize this error. Different models use different cost functions to calculate mean squared error (MSE).

There are two types of linear models we will evaluate:

Logistic Regression. “A model that generates a probability for each possible discrete label value in classification problems by applying a sigmoid function to a linear prediction.” (Google Developers) We will use penalty functions of L1 and L2, where L1 uses lasso regression as penalty term to the loss function and L2 uses ridge regression. The major difference between these two is that Lasso shrinks the less important features to Zero (as a form of feature selection) and L2 does not.

Stochastic gradient descent (SGD). SGD is “a technique to minimize loss (error) by computing the gradients of loss with respect to the model's parameters, conditioned on training data. Informally, gradient descent iteratively adjusts parameters, gradually finding the best combination of weights and bias to minimize loss.” (Google Developers)

Ensemble Methods

Ensemble methods combine several ML models that have learned to solve the problem to achieve better predictive performance vs a single model. Different ensemble methods can be used to decrease variance (bagging), bias (boosting), or improve predictions (stacking).

Ensemble methods use weak learners (or base learners) to combine their ‘knowledge’ to create one strong learner.

We will evaluate the following Ensemble Methods:

> **Random Forest.** Random forest fits a number of decision trees (a forest of trees) on various sub-samples of the data and uses the average to improve predictions. Random, comes from randomly picking features to create the trees.
>
> **Gradient Boost.** “Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.” (Wikipedia) For posting, we combine learners that are made sequentially- where subsequent learners learn from previous mistakes (thus the ‘boost’). Learners are generated during the learning process.
>
> **Ada-Boost**. Ada-Boost (adaptive boosting), like Gradient Boost, combines weak learners to make a strong learner. The difference is that Adaboost creates weak learners by changing the sample distribution where Gradient Boosting does not.
>
> **Bagging**. For bagging, we combine many independent learners and use an averaging approach to pick the best. These are independent since we take a random sample of the data.
>
> **Stacking**. Stacking will combine multiple classification models (base classifiers) via a meta-classifier. Each base classifiers is trained on the complete data set and then the meta-classifier is trained on the outputs of the base classifiers.

Naive Bayes

Naive Bayes classifiers are based upon the Bayes’ theorem which “describes the probability of an event based upon prior knowledge of the conditions that might be related to the event.” (Wikipedia). Bayes theorem finds the probability of an event based upon previous events. The “Naive” part is that the data features are independent.

The difference between Naive Bayes classifiers is the assumption made around conditional probability. The two models we will evaluate are GaussianNB and KNeighbors.

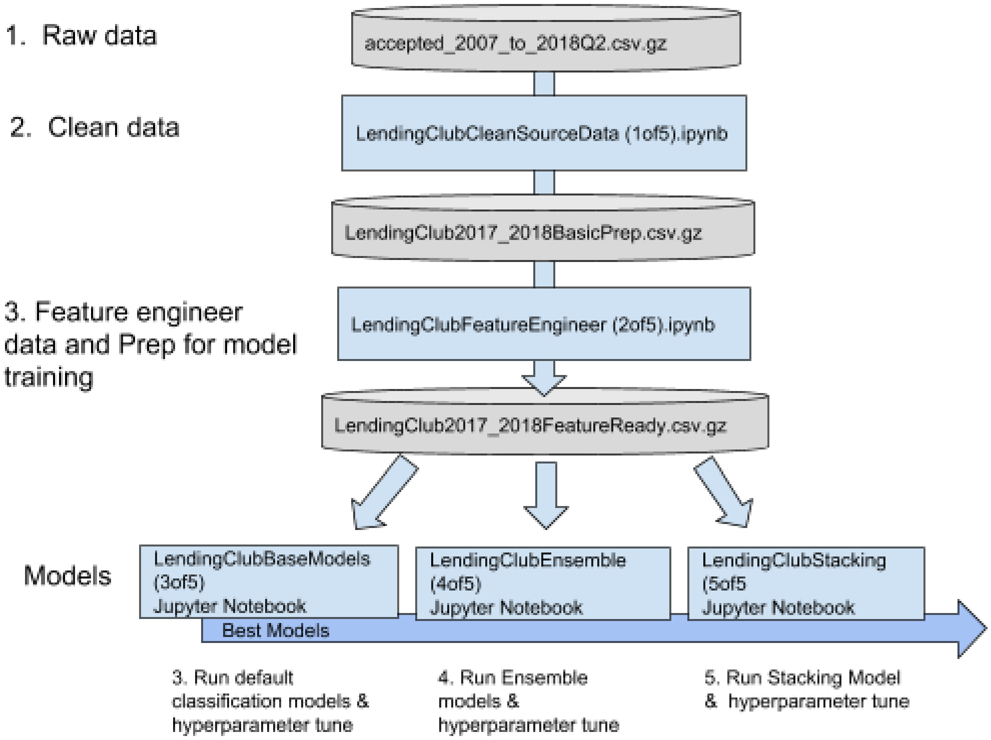
> **GaussianNB**. Assumes that features are distributed based upon a Gaussian distribution (The bell curve we are all familiar with.)
>
> **K-Neighbor**: K-Neighbor makes no assumptions on the distribution of data. It makes a prediction by locating similar cases.

Decision Tree

“Decision trees (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).“ (Wikipedia). Decision Trees are what they sound like. Based upon a decisions, one branch of the tree is taken vs another, and the process repeats itself till a decision is made.

Techniques

The approach to training will be a multi-step process. Once we clean, feature engineer, and prep the data we will run it through the models as follows:


1. Raw data
accepted_2007_to_2018Q2.csv.gz
2. Clean data
LendingClubCleanSourceData (1of5).ipynb
LendingClub2017_2018BasicPrep.csv.gz
3. Feature engineer data and Prep for model training
LendingClubFeatureEngineer (2of5).ipynb
LendingClub2017_2018FeatureReady.csv.gz
Models
LendingClubBaseModels (3of5) Jupyter Notebook
LendingClubEnsemble (4of5) Jupyter Notebook
LendingClubStacking (5of5 Jupyter Notebook
Best Models
3. Run default classification models & hyperparameter tune
4. Run Ensemble models & hyperparameter tune
5. Run Stacking Model & hyperparameter tune

1. Explore General Classification Models.

Run classifiers with default hyperparameters.
Picking the best of the models to tune, optimizing the hyperparameters to get the best f-beta, AUC, and precision scores.

2. Ensemble Methods:

Use the best general classification models to use as a base estimator for Bagging and AdaBoost ensemble methods to improve the performance of the bagging and AdaBoost algorithms

3. Stacking

Last, I will explore stacking. Again, focusing on the best classification models to feed into the stacking model. The goal would be to combine models that would increase both precision and recall to improve the overall f-beta score.

Benchmark

There are many benchmarks on lending data, credit scoring and the like. The most popular is FICO. This uses up to 1500 data points to make an analysis. Default rates run from 15% to 1% for FICO scores from 650 to 850.[15]

The Lending Club dataset is a very popular dataset on Kaggle with many kernels created to evaluate and explore the data. Kernels performing the equivalent analysis are using the AUC score. There are three kernels I picked to be used as a benchmark (these were the most up-voted):

- https://www.kaggle.com/ionaskel/credit-risk-modelling-eda-classification
  AUC, using the trapezoidal rule, of 0.70 and accuracy of .79. Using logistic regression
- https://www.kaggle.com/pileatedperch/predicting-charge-off-from-initial-listing-data
  AUC score of 0.689. Using logistic regression
- https://www.kaggle.com/benesalvatore/predict-default-using-logisitic-regression
  AUC score of 0.7111916771536655. Using logistic regression

Of these. I will use an AUC score, using the trapezoidal rule, of .70 as the benchmark floor. In a Kaggle competition, the top score was an AUC of 0.718.

Also, I will be using the metrics as outlined above as a target. Using these numbers and the AUC benchmark score we have the following targets:

- AUC score > .7
- Precision score of > .386
- f-Beta score > .44 (β=.25)

[15] http://www.wvasf.org/presentation_pdfs/John_Meeks_-_WV_Asset_Building_Charleston_102811.pdf

III. Methodology

Data Preprocessing

We are interested in determining if a loan will be defaulting or not, So we are interested in loans that have been paid or charged off and defaulted. Since we are evaluating based upon information that is known at the time of the loan application we will remove data with on-going servicing, credit checks, etc.

> *Implementation of the Data Preprocessing is located in the Jupyter Notebook: "LendingClubCleanSourceData.ipynb" and requires the file "accepted_2007_to_2018Q2.csv" outputting the file: "LendingClub2017_2018BasicPrep.csv"*

We will first reduce the size to something more manageable. This will consist of:

- Scale back to two years (2016 and 2017)
- Use only paid and charged off/default segments
- Remove/combine any underrepresented segments.
- Take out any Lending Club generated data

Scale back to two years (2016 and 2017)

Remove based upon the **issue_d** (issue date) of the loan, which would correspond to when the initial application was approved.

Remove any underrepresented segments and combine segments as needed
The joint application will be one of the areas that we will remove since it is a small segment of the overall data - both applications that are joint, and the data associated with the second application will be removed.

- Joint account data. We are only training on single applicant loans

Take out any Lending Club generated data

Initial Columns to remove. These are columns to be initially removed. Generally, these are columns that represent post-loan approval data as created by Lending Club.

The criteria for removing these fields:

- Second Applicant: We will remove all second applicant data and only focus on single applicant data

- Delinquent Loan Servicing Info: This is data labeled as hardship, debt settlement, etc.
- Ongoing Servicing And Collections: This is data about what current payment balances, late fees, etc.
- Ongoing Credit Pull Data. Since we are only interested in data at the time of the application, this data needs to be removed
- Member data. This is Lending Club data about the member, like member number.

See Appendix 2 for a list of fields removed.
See Appendix 4 for a list of fields kept.
See Appendix 5 for a list of fields used in the final models

After preparing the above, the training data set of 2,004,091 down to 178,821 records

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 178821 entries, 235629 to 2004090
Data columns (total 85 columns):
```

Feature Engineering

These are the following areas where we need to feature engineer

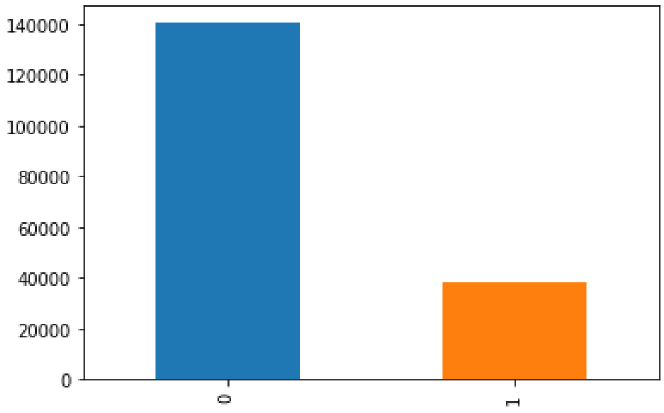
- Target variable: loan_status
- Objects: Term, grade, verification status, purpose, zip_code, addr_state, emp_length, home_ownership, emp_title
- Highly correlated data: fico_range_low, fico_range_high
- Missing values (as listed above)
- Outliers (as listed above)
- Date data: Issue_d, earlient_cr_line
- External data

Implementation Feature Engineering is located in the Jupyter Notebook: “LendingClubFeatureEngineer.ipynb” and requires the file “LendingClub2017_2018BasicPrep.csv” generated from “LendingClubCleanSourceData.ipynb”. This notebook will produce the file: LendingClub2017_2018FeatureReady.csv ready for model training.

Target Training Variable: loan_status

We will train on loan status. For this, we only want defaulted and paid and we will drop anything else.

Re-bucket 'Charged Off','Default' to the value of 1
'Fully Paid', 'Completed' to the value of 0.



Objects

Term

Term is generally generated by underwriting, but could also be requested by the customer. This tends to be the result of credit evaluation, so in our study, we will convert this to an integer in length in months. “60 Months → 60”

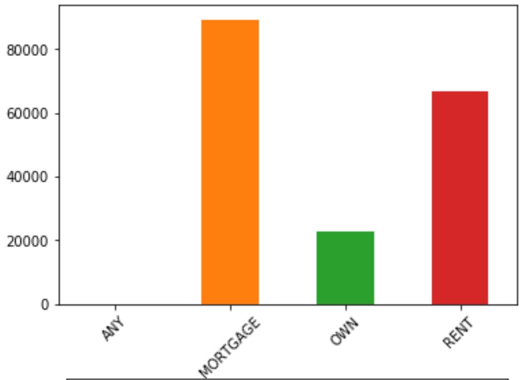
Grade

Grade is highly correlated with default rates and interest rates. Both Interest_Rate and Grade need to be dropped due to a high correlation between them and default rates.

Home Ownership

‘ANY’ is 93 records. We will drop ‘ANY’ values rather than try to re-bucket or train on.

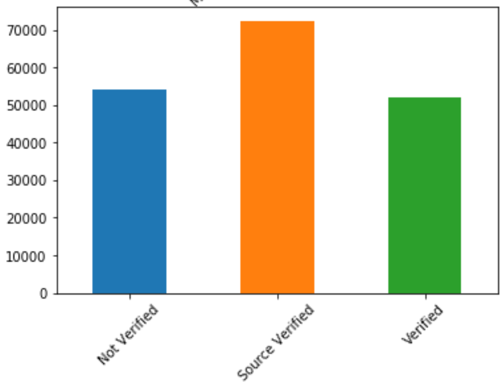
The remaining values will be one-hot encoded.



Verification Status:

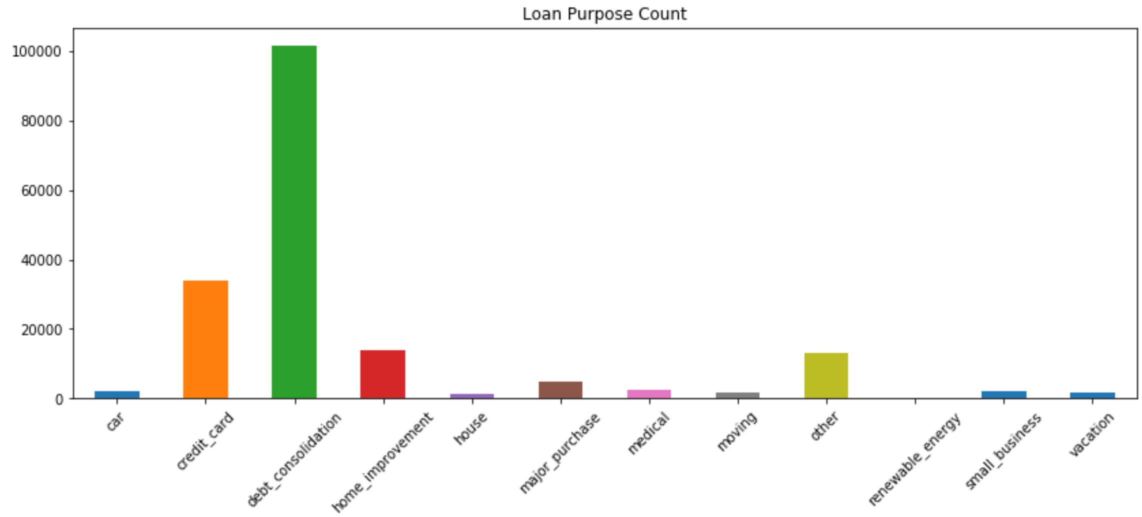
This is an important variable that will be one-hot encoded:

Name: verification_status, dtype: int64



Purpose

The purpose could be an important value. Since this is a limited set of data we will one-hot encode



Addr_state

The State of the borrower could be important relative to the cost of living or other factors. Addr_state can't be feature engineered, so it will be on-hot encoded

Zip_code

The zip code could be important relative to the cost of living or other factors, but there are 897 unique values. *zip_code* will be dropped if it can't be feature engineered.

Emp_length (employment length)

Employment length is important, so we convert this to months Employed. If we do not have a value we will zero fill.

Emp_title (Employ Title)

Employee title could be important, but we have 58,884 unique values. These are supplied by the borrower. At this time it will be hard to feature engineer this so this column will be dropped.

Highly correlated data

Fico

fico_range_low, *fico_range_high* are highly correlated. We will create an 'AvgCreditScore' variable that is the average of the two.

Grade, Interest

These are highly correlated with each other and delinquency rates that they will be dropped (These are calculated by Lending Club)

Missing values

All columns with missing data are related to credit data provided by a credit bureau. It is most likely that these missing values represent no data point, rather than data missing. There are three approaches: marking the values as missing, zero fill, or remove the records.

I choose to zero fill, since the assumption is that most of these values are not reported since there is no data, thus the values would be zero. These columns will be zero filled:

> inq_last_6mths,mths_since_last_delinq,mths_since_last_record,revol_util, Mths_since_last_major_derog,open_acc_6m,open_act_il,open_il_12m, mths_since_rcnt_il,total_bal_il,il_util,open_rv_12m,open_rv_24m,max_bal_bc, inq_fi,inq_last_12m,avg_cur_bal,bc_open_to_buy,bc_util,mo_sin_old_il_acct, mths_since_recent_bc,mths_since_recent_bc_dlq,mths_since_recent_inq, Mths_since_recent_revol_delinq, num_accts_ever_120_pd,num_tl_120dpd_2m, percent_bc_gt_75,open_il_24m

Outliers

The following are outliers in the data set. These are outside of the 99.9% quartile.

- For annual_inc 15 outliers above 4200000.0
- For tot_coll_amt 2 outliers above 173380.0
- For acc_open_past_24mths 1 outliers above 56.0
- For mo_sin_rcnt_tl 2 outliers above 289.0
- For mort_acc 2 outliers above 37.0
- For num_actv_bc_tl 4 outliers above 29.0
- For num_sats 2 outliers above 80.0
- For num_tl_op_past_12m 4 outliers above 28.0
- For tot_hi_cred_lim 3 outliers above 9999999.0
- For total_bal_ex_mort 3 outliers above 1234429.0
- For total_bc_limit 1 outliers above 1105500.0
- For total_il_high_credit_limit 5 outliers above 1120706.0

In the final models no outliers were removed

Date data

Issue_d, earlient_cr_line are two date values remaining. I what could prove value is the length of the credit history. For this, we will engineer the data to create:

CreditHistoryMonths to equal the total months between *earliest_cr_line* and *issue_d*

We will then drop *earliest_cr_line* and *issue_d*

External data

Many FinTech providers build credit models using 'unconventional' data from a variety of data sources. As outlined in the "Project Overview' section, many Fintechs use supplemental data such as social media or area demographic data (such as Kabbage using UPS delivery data and social media as part of its credit score.[16])

In this project, we will test replacing ***zip_code*** and ***addr_state*** (State Code) with demographic data.

Addr_state:

Addr_state (the two-letter state abbreviation), is not that useful, so I replaced the state with the cost of living index:
https://www.money-rates.com/research-center/best-states-to-make-a-living/

I will explore if these types of external variables will help with Modeling Accuracy

Zip_code:

The zip_code field in the Lending Club Data is the first 3 of 5 digits in the zip code. Example: 94107 is 941xx. I have taken IRS data and summarized for these 'zip_codes'. I am replacing zip code with the following data elements:

- PREP Number of returns with paid preparer's signature
- N2 Number of exemptions
- ELDERLY Number of elderly returns
- SCHF Number of farm returns
- N02300 Number of returns with unemployment compensation

Attributes I am looking to associate with charged or paid off loans with the above demographics. (Small farms, vs urban, elderly, vs more well-off.)

File: https://www.irs.gov/statistics/soi-tax-stats-individual-income-tax-statistics-2016-zip-code-data-soi

Additional Columns Dropped

The following columns were dropped:
Grade, int_rate, emp_title, zip_code,
fico_range_low, fico_range_high, issue_d, earliest_cr_line

[16] https://www.bankdirector.com/committees/lending/serving-up-kabbage-to-small-businesses-with-a-side-of-technology/

One-Hot Encoding:

The following columns where 'home_ownership', 'verification_status', 'purpose', ‘addr_state’

Implementation

The implementation process follows what was outlined in the above section ‘Algorithms and Techniques’. After the data was prep, as outlined above, I implemented model training in three steps, with each one building on the previous.

Tracking of my test runs will be found in the sheet ‘Run Tracking.xlsm’

In general, I followed the following algorithm for each step of model tuning:

```
Create a Pipeline of models and a range of hyperparameters for each model:
For each run in the pipeline:
    Run a Grid Search CV. Using Standard scaler
    Score and reporting on the models
```

1. General Classification Models.

Running the following classifiers with default hyperparameters (HP):

Logistic Regression with L1, Logistic Regression with L2, Random Forest, Gradient Boost, Bagging, AdaBoost, GaussianNB, KNeighbors, SGD, Decision Tree

Conduct multiple runs to determine the best HP:

a. Default Hyperparameters, 10% of the dataset
b. GridsearchCV (Long run - a variety of HP), 10% of the Data set
c. GridsearchCV (Best HP so far), Full Data set
d. With External Data Features and without.
e. With standard scaler set and without.
f. Final Tuning with Full runs to get the best model params

2. Ensemble Methods:

Use the best classification models to use as a base estimator for bagging and AdaBoost ensemble methods to improve the performance of the Bagging and AdaBoost algorithms

3. Stacking Ensemble Methods

Last, I will explore stacking. Again, focusing on the best classification models to feed into the stacking model. The goal is to combine the best models that would increase both Precision and Recall to improve the overall f-beta score.

For the stacking Classifier, I ran two approaches. The first was based upon an approach, outlined on Kaggle[17], that is an ensemble method that takes base predictions of a set of classifiers and uses a second-level model to predict the output from the base predictions.

This used L1, L2, GBC, AdaBoost and RFC as first level models, with second level models tried XGB, L2, L3, RFC, GBC, bagging, AdaBoost, GaussianNB, Decision Tree and KNeighbors.

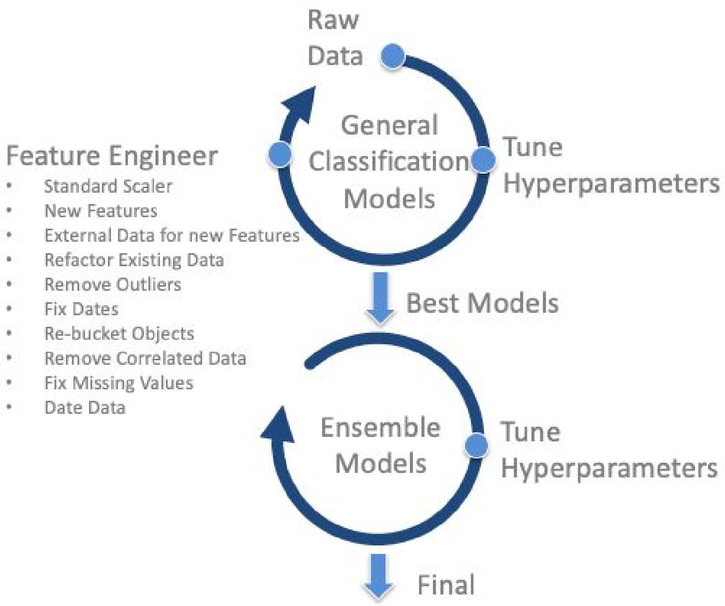
The second approach, I used Mlxtend Stacking Classifier[18]. This required weak learner classifiers and a meta-classifier. The classifiers, picked from the best models from the Ensemble methods and case classifiers, include:
Meta-classifier: bagging with GBC as the base estimator, with classifiers of

- Bagging with Base Estimator: Logistic Regression (Penalty= l2)
- Bagging with Base Estimator: gbc
- Logistic Regression (Penalty= Le)
- Logistic Regression (Penalty= L1)

Refinement

The refinement process generally followed what was outlined in the above section ‘Algorithms and Techniques’. It was an iterative process that flowed as in the diagram on the right.

There were multiple iterations of tuning parameters and testing different feature combinations.



[17] https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python
[18] http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/#api

General Classification Models

The start of the process was to run general Classification modes with a subset of the unmodified data (removing Lending Club generated data, date data, and limited to only two years of data.

The initial runs were done with Default Hyperparameters and 10% of the dataset with the following results:

Model	AUC	Fbeta β=.25	Recall	Precision	Run Time	Accuracy
Logistic Regression (Penalty= L1)	0.697	0.421	0.099	0.528	0.241	0.79
Logistic Regression (Penalty= L2)	0.697	0.422	0.1	0.528	0.059	0.79
rfc	0.625	0.348	0.086	0.429	0.024	0.781
gbc	0.712	0.435	0.086	0.582	0.305	0.792
bagging	0.638	0.394	0.117	0.463	0.237	0.783
AdaBoost	0.702	0.441	0.139	0.51	0.112	0.788
gaussiannb	0.628	0.238	0.915	0.227	0.004	0.32
rfc	0.625	0.348	0.086	0.429	0.024	0.781
decisiontree	0.551	0.29	0.305	0.289	0.03	0.692
sgd	0	0.282	0.278	0.282	0.004	0.696
KNeighbors	0.581	0.309	0.135	0.336	1	0.759
bagging	0.71	0.434	0.076	0.615	18.866	0.794

At this point, there are good results and one, AdaBoost, beat my benchmark goals. From the base run, I selected the best of the models to tune, optimizing the hyperparameters to get the best fbeta, AUC, and precision scores.

Feature Tuning:

- Standard Scaler: Tested with both. Kept Standard Scaler
- New Features: Added new feature: ‘Months since first credit check’
- External Data for new Features: I tested adding two new features. One to replace states with cost-of-living, and one, to replace zip code data with IRS data that measure the number of returns with different characteristics (farm returns, elderly, with investment income) - Both of these were failures.
- Test Outliers. I removed outliers but eventually added them back
- Fix Dates: I fixed any date fields, creating a new field, months since initial credit report
- Re-bucket Objects: Some objects were re-bucketed
- Remove Correlated Data: Fico data were combined into a single feature
- Fix Missing Values.

Tuned Hyperparameters running a pipeline to gridsearchCV to find the best. The final results of tuning and feature engineering are in the following table, with GBC being the best model with an AUC of 7.35, fbeta of .482 and precision of .598, beating my goals.

Model	AUC	Fbeta β=.25	Recall	Precision	RunTime (minutes)	Accuracy
bagging	🟩 0.71	0.434	0.076	🟩 0.615	18.866	0.794
gbc	🟩 0.735	🟩 0.482	0.118	🟩 0.598	1.918	0.796
gaussiannb	0.647	0.341	0.403	0.338	0.097	0.705
rfc	🟩 0.711	🟩 0.458	0.106	🟩 0.576	43.541	0.794
Decision tree	0.676	0.411	0.116	🟩 0.488	0.319	0.786
Logistic Regression (Penalty= L2)	🟩 0.723	🟩 0.46	0.11	🟩 0.576	0.323	0.794
AdaBoost	🟩 0.725	🟩 0.474	0.141	🟩 0.556	3.818	0.794
Logistic Regression (Penalty= L1)	🟩 0.723	🟩 0.461	0.11	🟩 0.576	1.4	0.794

* Items in green beat the baseline targets

Ensemble Methods:

The next stage was to take the best general classifiers and run training on Bagging and AdaBoost with different base estimators.

Model	AUC	Fbeta β=.25	Recall	Precision	RunTime (minutes)	Accuracy
bagging	0.71	0.434	0.076	0.615	6.73	0.794
AdaBoost+l2	0.688	0.39	0.387	0.391	15.227	0.741
AdaBoost	0.725	0.474	0.141	0.556	1.824	0.794
bagging+gbc	0.734	0.434	0.07	0.644	39.862	0.794
bagging+l2	0.721	0.403	0.064	0.601	14.83	0.792
AdaBoost+gbc	0.736	0.504	0.184	0.566	13.329	0.797

* Items in green beat the baseline targets

The best model was AdaBoost with a GBC base estimator with an AUC of 7.36, fbeta of .504 and precision of .566, beating my goals. Measured by fbeta score, this was my best model.

Stacking Ensemble Methods

The last stage was to take the best models from Ensemble and general classifiers to generate a stacked model. I tried multiple models with the best as follows:

Model	AUC	Fbeta β=.25	Recall	Precision
StackingClassifier	0.739	0.499	0.132	0.605
Meta-Classifier: bagging with GBC Base Estimator				
Classifiers: ● Bagging with Base Estimator: Logistic Regression (Penalty= l2) ● Bagging with Base Estimator: gbc ● Logistic Regression (Penalty= Le) ● Logistic Regression (Penalty= L1)				

All told, there were over 6500 runs (review the various tabs in the excel sheet 'Run Tracking.xlsx'). Final hyperparameters can be found in Appendix 3

IV. Results

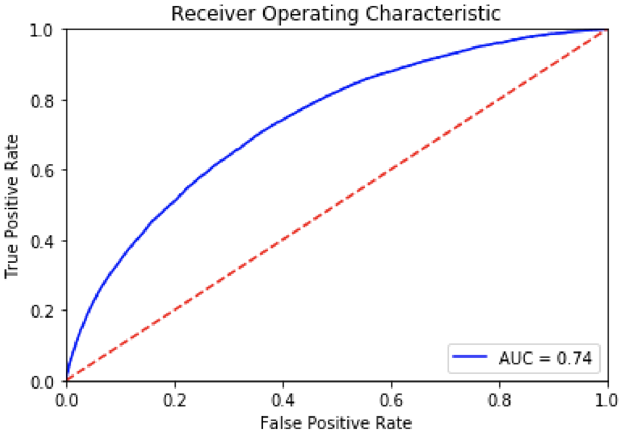
Model Evaluation and Validation

The top three models come one each from the three approaches listed above in the implementation section. Models were scored on the fbeta score with a β=.25, to weight precision with more importance than recall.

Rank	Method	Model	AUC	fbeta β=.25	Recall	Precision
1	Ensemble Method	AdaBoost with gbc As the base estimator	0.736	0.504	0.184	0.566
2	Stacking Ensemble Method	Mlxtend StackingClassifier: Meta-Classifier: bagging+gbc Classifiers: - Bagging w/ L2 BE* - Bagging w/ GBC BE - Logistic Regression (Penalty= l2) - Logistic Regression (Penalty= l1)	0.739	0.499	0.131	0.605
3	General Classification Model	Gradient Boosting Classifier	0.735	0.482	0.118	0.598

* Base Estimator

The best model is AdaBoost with GBC as the base estimator with the best fbeta score beating the goals with AUC of 0.736 beating 0.700, Precision of 0.566 beating 0.386, fbeta of 0.504 beating 0.440.



Most of the models tested converged to around the same results and still have the same relative results even with different data sets and feature engineering. This tells us a few things:

- First, The final model is aligned with the solution expectations - having beat our targets and benchmarks.
- The model parameters are solid, having done well with different datasets, unseen data (testing sets held back for evaluation) and hyperparameters (See the runs in the “Run Tracking.xlsx” file.) changes do not greatly affect the results with our results only being the best of a lot of good runs.

This is a good, robust, trustworthy model. However, this is a model tuned to high precision, in a real-world setting for credit evaluation, the predicted defaults would be engineered in a way that met the business goals of the lending institution, which will price lending products based upon their projected risk.

See Appendix 3 for details on the models used and their hyperparameters.

Justification

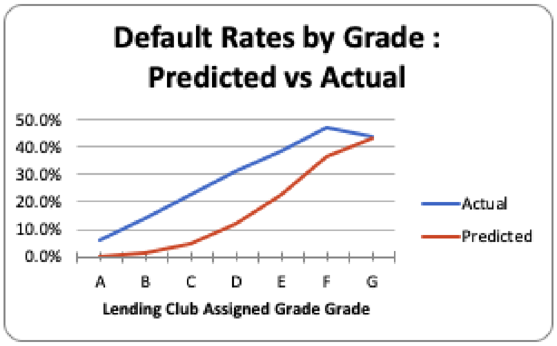
My best results are based upon using the Bagging Classifier with a base estimator using a Gradient Boost Classifier. The benchmark is based upon data provided by Lending Club. The data reflects loans that have already been graded and issued. As I calculated, the estimated baseline numbers are based upon actual performance:

Recall = 1.0, Precision = 0.386, f1 = 0.55, f-beta(β=.25) = .44

This data reflects the actual Lending Club credit risk profile for loans. It would be expected that the model they use is already optimized and I would not expect a large difference in my numbers vs Lending Club. I would also expect my results to mirror the Lending Club results and others who have evaluated the data.

The second benchmark is measured by others who have run models against the data (Using the AUC score). With both benchmark sources I have the following results.:

	AUC	fbeta β=.25	Precision
Benchmark	.7	.44	.368
Best Score	.736	.504	.566



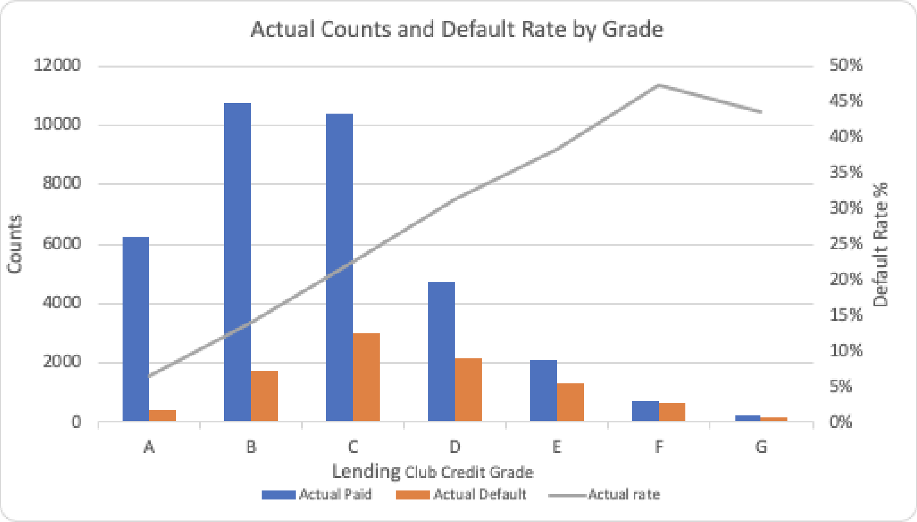
My results mirror and beat the benchmark results. When evaluating predicted default rates against Actual Lending Club default rates by the Lending Club Grade, we can see that predicted results follow actual, solving the problem.

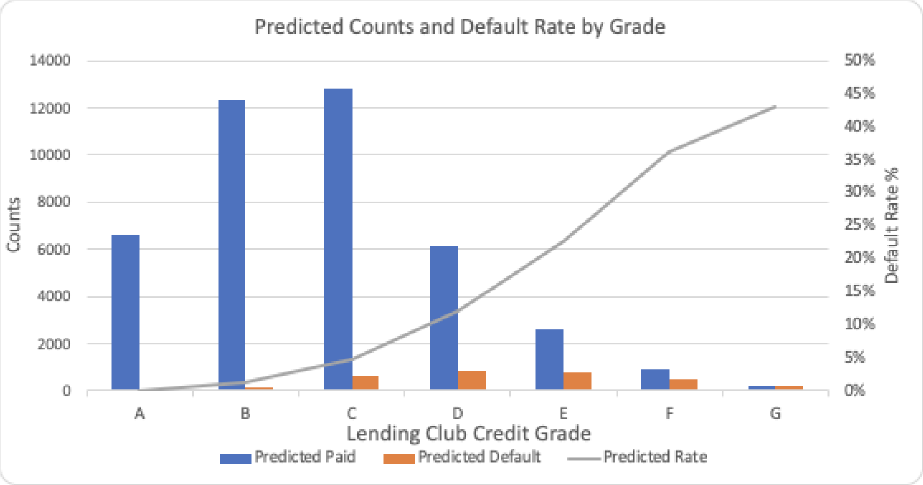
V. Conclusion

Free-Form Visualization

The main goal of the project is to create a model that accurately predicts default rates. Since we are using data that has already been evaluated and graded we can measure model results against actual performance.

In the two graphs, we can both actual default rates by grade and predicted default rates. We can see that these generally mirror each other. Both reflecting higher default rates by Credit Grade and the general slope of the line (the increase in default rates vs lower credit Grades)





Reflection

Overall, this was a very interesting project. The objective was to accurately predict default rates. The method was to use increasingly more complex models, using the results of previous models to tune subsequent models. I also wanted to explore feature engineering and the ability to feature engineer and use outside data to help supplement results.

What I discovered is the data tells a story that is hard to improve upon. While I did extensive feature engineering and tried, and failed, to improve results with external data it could be seen that a major breakthrough (like a fbeta score of .8) was not to happen.

In addition, the more complex stacking models show the potential to create better results, but my lack of experience proved difficult in tuning these models.

While results met my expectations, I think the core of this model is not ready for a real-world setting. In the real world, the goal in credit evaluation is to grade a loan application by risk (probability of default) and create terms and pricing that reflect those probabilities since is impossible to predict, with 100% accuracy, if a borrower will default.

Improvement

Intuitively, I believe that the model can be improved upon, but in my experience this became difficult. The best illustration of this is the accuracy score. The accuracy score did not change much from .79. Models could be tuned, for example, to improve recall or precision, but the accuracy score did not change.

My hope with using stacking ensemble methods to combine models that had high accuracy and other models with high precision that I could combine the best of ‘both’ worlds to achieve breakthrough fbeta and F1 scores. This did not happen, but it is possible to improve upon my results with a better understanding of how these models work and how to best tune for them.

Appendix 1. Running the Code

Running the Code

The completed project is run in multiple steps.

This is a 5-step process. Steps 1 and 2 prepare the data and steps 3 to 5 are different models. The file LendingClub2017_2018FeatureReady.csv.gz is provided, so you can start at step 3

1. LendingClubCleanSourceData (1of5).ipynb
    a. Do the first pass of data cleaning, reducing the data set to the segments needed
    b. Takes the raw file, accepted_2007_to_2018Q3.csv.gz,
    c. outputs: LendingClub2017_2018BasicPrep.csv.gz

2. LendingClubFeatureEngineer (2of5).ipynb
    a. Do feature engineering and pre-processing to get the data 'model ready'
    b. Takes: LendingClub2017_2018BasicPrep.csv.gz
    c. Outputs: LendingClub2017_2018FeatureReady.csv.gz

3. LendingClubBaseModes (3of5).ipynb
    a. Takes: LendingClub2017_2018FeatureReady.csv.gz (this file is provided)
    b. Run Classifiers with default hyperparameters.
    c. Run GridSearchCV on Classifiers to tune hyperparameters.

4. LendingClubEnsemble (4of5).ipynb
    a. Takes: LendingClub2017_2018FeatureReady.csv.gz (this file is provided)
    b. Run Ensemble methods with best models, from step 3 to find the best base estimators,

5. LendingClubStacking (5of5).ipynb
    a. Takes: LendingClub2017_2018FeatureReady.csv.gz (this file is provided)
    b. Run Stacking methods with best models, from step 3 to find the best classifiers and meta-classifier.
    c. Using: http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

Appendix 2. Deleted Columns

Columns to be deleted from the initial dataset

Second Applicant sec app fico range low sec app fico range high sec app earliest cr line sec app inq last 6mths sec app mort acc sec app open acc sec app revol util sec app open act il sec app num rev accts sec app chargeoff within 12 mths sec app collections 12 mths ex med sec app mths since last major derog	**Delinquent Loan Servicing info** hardship flag hardship type hardship reason hardship status deferral term hardship amount, start date, end date hardship length hardship dpd, loan status payment plan start date orig projected additional accrued interest hardship payoff balance amount hardship last payment amount debt settlement flag debt settlement flag date settlement status, date, amount settlement percentage settlement term]	**Ongoing Servicing And Collections** initial list status out prncp out prncp inv total pymnt total pymnt inv total rec prncp total rec int total rec late fee recoveries collection recovery fee last pymnt d last pymnt amnt next pymnt d]
ongoingCreditPullData last credit pull d last fico range high last fico range low] **Underwriting** installment application_type term	**joint account data** annual inc joint dti joint verification status joint all util pct tl nvr dlq revol bal joint total cu tl]	**member data** member id funded amnt funded amnt inv sub grade pymnt plan policy code url sub grade disbursement method title

Appendix 3. Hyperparameters

The top three models Hyperparameters:

Method	Model	Parameters
Ensemble Method	AdaBoost with gbc As the base estimator	Bagging:

		'max_features': [.75] 'max_samples': [.75] 'n_estimators': [10] GBC: ‘n_estimators': [300], 'min_samples_split': [500], loss': ['deviance'], 'min_samples_leaf': [25], 'max_features': ['sqrt'], 'max_depth': [6], 'learning_rate':[.1], 'subsample': [.8], 'validation_fraction' :[0.1], 'n_iter_no_change' :[10], 'tol':[0.001]

Stacking Ensemble Method	Mlxtend StackingClassifier	use_probas = True
		average_probas = True
		use_features_in_secondary = True
		use_clones = True

	Meta Classifier: bagging with gbc base estimator	Bagging: Defaults GBC: Defaults
	Bagging w/ L2 Base Estimator	Defaults
	Bagging w/ GBC Base Estimator	Bagging: Defaults GBC: Defaults
	Logistic Regression (Penalty= l2)	Defaults
	Logistic Regression (Penalty= l1)	Defaults
General Classification Model	Gradient Boost	GBC: ‘n_estimators': [300], 'min_samples_split': [500], loss': ['deviance'], 'min_samples_leaf': [25], 'max_features': ['sqrt'], 'max_depth': [6], 'learning_rate':[.1], 'subsample': [.8], 'validation_fraction' :[0.1], 'n_iter_no_change' :[10], 'tol':[0.001]

Appendix 4: Data Columns to keep

Data Columns to initially keep:
Color Key: Underwriting, Reported Credit bureau Data, Application Data

Loan amnt	mths since last record	open rv 12m

emp title (object)
emp length (object)
home ownership (object)
annual inc
purpose (object)
zip code (object)
addr state (object)

verification status (object)
issue d (date)
loan status (object)
grade (object)
int rate
term (object)

dti
delinq 2yrs
earliest cr line (date)
fico range low
fico range high
inq last 6mths
mths since last delinq

open acc
pub rec
revol bal
revol util
total acc
mo sin old il acct
mo sin old rev tl op
mo sin rcnt rev tl op
mo sin rcnt tl
mort acc
delinq amnt
collections 12 mths ex med
mths since last major derog
mths since recent bc
mths since recent bc dlq
mths since recent inq
mths since recent revol delinq
mths since rcnt il
acc now delinq
tot coll amt
tot cur bal
il util

open rv 24m
max bal bc
total rev hi lim
inq fi
percent bc gt 75
inq last 12m
acc open past 24mths
avg cur bal
bc open to buy
bc util
chargeoff within 12 mths
pub rec bankruptcies
tax liens
tot hi cred lim
total bal ex mort
total bc limit
total il high credit limit
open acc 6m
open act il
open il 12m
open il 24m
total bal il

Appendix 5: Source Data Fields

Data that is color coded will generally be kept for further analysis and/or feature engineering: Underwriting, Credit Data, Application Data

Application Data
loan amnt
emp title (object)
emp length (object)
home ownership (object)
annual inc
purpose (object)
zip code (object)
addr state (object)

joint account data
annual inc joint
dti joint, all util
verification status joint
pct tl nvr dlq , total cu tl]
revol bal joint

Second Applicant
sec app fico range low
sec app fico range high
sec app earliest cr line
sec app inq last 6mths
sec app mort acc
sec app open acc
sec app revol util
sec app open act il
sec app num rev accts
sec app chargeoff within 12 mths
sec app collections 12 mths ex med
sec app mths since last major derog

Member data
member id
funded amnt
funded amnt inv
sub grade
pymnt plan
policy code
url
sub grade
disbursement method
titl

ongoingCreditPullData last credit pull d last fico range high last fico range low] Underwriting Installment application_type verification status (object) issue d (date) loan status (object) grade (object) int rate term (object)	**Delinquent Loan Servicing info** hardship flag, type, reason, status deferral term hardship amount, start date, end date hardship length hardship dpd, loan status payment plan start date orig projected additional accrued interest hardship payoff balance amount hardship last payment amount debt settlement flag debt settlement flag date settlement status, date, amount settlement percentage settlement term]	**Ongoing Servicing & Collections** initial list status out prncp out prncp inv total pymnt total pymnt inv total rec prncp total rec int total rec late fee recoveries collection recovery fee last pymnt d last pymnt amnt next pymnt d]
Credit Bureau Data _dti delinq 2yrs earliest cr line (date) fico range low fico range high inq last 6mths mths since last delinq acc now delinq tot coll amt tot cur bal il util mths since last record open acc pub rec revol bal revol util total acc	mo sin old il acct mo sin old rev tl op mo sin rcnt rev tl op mo sin rcnt tl mort acc delinq amnt collections 12 mths ex med mths since last major derog mths since recent bc mths since recent bc dlq mths since recent inq mths since recent revol delinq mths since rcnt il open rv 12m open rv 24m max bal bc total rev hi lim inq fi	percent bc gt 75 inq last 12m acc open past 24mths avg cur bal bc open to buy bc util chargeoff within 12 mths pub rec bankruptcies tax liens tot hi cred lim total bal ex mort total bc limit total il high credit limit open acc 6m open act il open il 12m open il 24m total bal il

Appendix 6: Final Schema

This is the of fields and engineered features used in the final models

Data Columns to initially keep:
Color Key: Underwriting, Reported Credit bureau Data, Application Data, Feature Engineered

Loan amnt emp length (object) home ownership (object) annual inc	mths since last record open acc pub rec revol bal	open rv 12m open rv 24m max bal bc total rev hi lim

purpose (object)
addr state (object)

CreditHistoryMonths
AverageCreditScore

verification status (object)
issue d (date)
loan status (object)
grade (object)
int rate
term (object)

dti
delinq 2yrs
earliest cr line (date)
fico range low
fico range high
inq last 6mths
mths since last delinq

revol util
total acc
mo sin old il acct
mo sin old rev tl op
mo sin rcnt rev tl op
mo sin rcnt tl
mort acc
delinq amnt
collections 12 mths ex med
mths since last major derog
mths since recent bc
mths since recent bc dlq
mths since recent inq
mths since recent revol delinq
mths since rcnt il
acc now delinq
tot coll amt
tot cur bal
il util

inq fi
percent bc gt 75
inq last 12m
acc open past 24mths
avg cur bal
bc open to buy
bc util
chargeoff within 12 mths
pub rec bankruptcies
tax liens
tot hi cred lim
total bal ex mort
total bc limit
total il high credit limit
open acc 6m
open act il
open il 12m
open il 24m
total bal il