Preprocessing

Cleaning

Several modifications and cleaning of the original training data needed to be made. Besides the obvious formatting and parsing of numerical values (e.g. $5,000 -> 5000), some choices had to be made.

- Removal of rows that were missing the target value (*interest rate*), and rows with too many missing values

- *years of experience* was originally entered as categories ("<1 year", "1 year", "2 years", ...). These were converted to numerical values for better performance in linear models.

- *loan grade* and *subgrade* were converted from letters "A" through "G" to numerical values 1.0 through 7.0. Sub-grades "A1", "A2", "A3", were mapped to 1.2, 1.4, 1.6, etc., forming a continuous numerical scale. This mapping relies on the assumption that the mean *interest rate* monotonically changes with *subgrade*. Analysis of the average *interest rate* shows this to be a valid assumption.

- Several fields contained text that was entered in by hand (sometimes by the loan recipient). While there is opportunity to extract deeper information through text mining techniques, I took a simpler and faster approach. I simply stemmed the text in each field, and sorted the words alphabetically (e.g. "credit card refinanced" and "refinancing credit card" both become "card credit refinanc").

  I took a look at the most frequent strings in the features *loan title* and *employer*. Many strings were repeated over 1000 times throughout the data set. I used these values as categories with all other strings set to an "other" category. This led to a total of 25 *loan title* categories and 13 *employer* categories.

  These categories could be better grouped, but would require making more assumptions, such as "RN" == "Registered Nurse", or "truck driver" == "driver". I was not willing to make these assumptions and did not consolidate categories this way, but this could be considered for future modeling.

  It should also be noted that *loan title* and *reason for loan* features appeared highly redundant with the *loan category* feature, so categorizing values this way will have minimal loss of information.

- *Zip codes* were originally entered as categorical values with the 3 first digits of the zip code. Originally, this was 878 unique 3-digit values. For simplification and faster runtimes, I reduced the number of zip code categories to 10 categories, using only the first single digit of the zip code. Since zip codes are structure regionally, the first digit still describes the east-west location within the U.S. More digits would give more precise

regional location, but at the cost of more categories. A more precise model would include all 3 digits.

Creating new features

There are likely to be interesting relationships between features that may be useful for predicting *interest rate*. Analysis of the relative importance of features in the Random Forest model confirmed that several new features (*issue year, funded/income, open line fraction, credit tenure, requested/income, credit balance/funded*) had relatively high importance (Fig. 2). These features were part of the top 20 features ranked by importance. The total set of new features introduced are:

- issue month of loan (as a category)
- issue year of loan*****
- credit tenure = loan issue date - earliest credit date
- loan amount funded / loan amount requested
- loan amount requested / borrower income
- loan amount funded / borrower income
- revolving credit balance / borrower income
- revolving credit balance / loan amount request
- revolving credit balance / loan amount funded
- number of open credit lines / total number of credit lines in credit report
- has delinquency (see: Inputing Missing Values section)
- has prev public record (see: Inputing Missing Values section)

:***** Some assumptions had to be made for *issue year of loan*. The year the loan was given could be treated as either a categorical feature, treated as a numerical feature, or removed entirely. To treat the feature as numeric, there should be reason to expect that order of the years are important, or that the average interest rates given smoothly change from year to year. Consider the average *interest rate* as a function of *issue year*:

Year	Avg(Interest Rate)
2007	10.29%
2008	11.15%
2009	12.18%
2010	11.73%
2011	12.25%
2012	13.63%
2013	14.54%
2014	14.03%

It is true that the interest rate appears to have a linear relationship with the year that the loan was issued. However, this increase with year should not be expected *ad infinitum*. Interest

rates are related to economic conditions of the year and the policies of the Federal Reserve. An economic crash or boom could drastically affect the interest rate given in a particular year. In fact, I found *loan issue year* to be the fifth most important feature in the Random Forest model. Given that the holdout set contains issue dates from early 2015 only, it may be a safe to assume interest rates will follow closely with 2014, in which case both numerical or categorical treatment of *issue year* can be used. For the linear model, a numerical feature is best.

If we were to predict interest rates for loans given in the distant future, we would not want to use *issue year* as a feature, but instead incorporate outside information such as features measuring economic strength, and features tracing Federal Reserve policies.

Removing features

Features removed included:

- *loan issued date* (transformed into *issued month* and *issued year*)
- *earliest credit date* (transformed into *credit tenure*)
- *borrower ID* (unique for all rows)
- *loan ID* (unique for all rows)
- *reason for loan* (This field contained written descriptions of the reason, which often were redundant with *loan title* and *loan category*. For simplicity, this feature was removed entirely.)

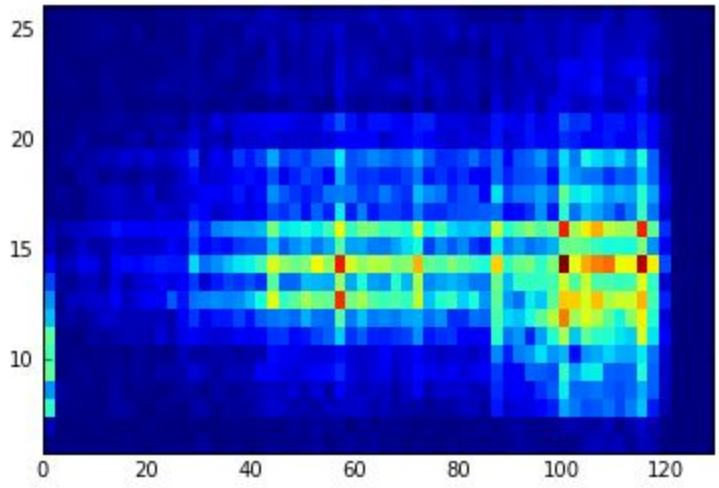
Imputing missing values

There were a total of 7 features that contained missing values. Median values were used to impute these features (*loan grade, loan subgrade, income, years employed, revolving credit utilization, months since delinquency, months since last record*.)

With two features, the fraction of observations with missing values was very high (185K / 400K for *months since delinquency*, and 296K / 400K for *months since last record*). In both of these cases, the absence of the value may have meaning (such as there is no delinquency). In fact, the average *interest rate* for those observations with non-null *months since delinquency* values = 14.6%, while the average *interest rate* for observations with null values = 13.4%. A similar behavior is seen with *months since last record.* Because of this, two boolean features were introduced, indicating whether or not these features had missing values.

Additionally anomalies were seen in the *months since last record* feature (Fig. 1). A value of 0 months could be interpreted as either just recently having a record, or never having had a record. The average *interest rate* for these observations = 10.6%, significantly lower than averages from both the non-null observations (14.8%) and the null observations (13.8%). These may belong to a group of their own. For simplicity, I left these values unchanged, but this should be considered more deeply when building more rigorous models.

Fig. 1: Interest Rate % (y) vs. Months Since Last Public Record (x)



Modelling

To measure the performance of my models and tune model parameters, I split the data set into a training set and a validation set (25%). Also, I created dummy features for categorical variables (e.g. *state=FL*, *state=TX*, …). Since for each feature, exactly one dummy variable will be set to 1, there is a correlated redundancy in the dummy variables. For example, *payment length=36* is False guarantees that *payment length=60* is True. Therefore, for each category feature, I removed one dummy feature.

Benchmark

To begin, I estimated a benchmark RMSE for comparing future models. This benchmark simply predicts all validation *interest rates* as the mean of the training set *interest rates* = 13.9%. Running this on the validation set yielded a validation RMSE = 4.39 percentage points.

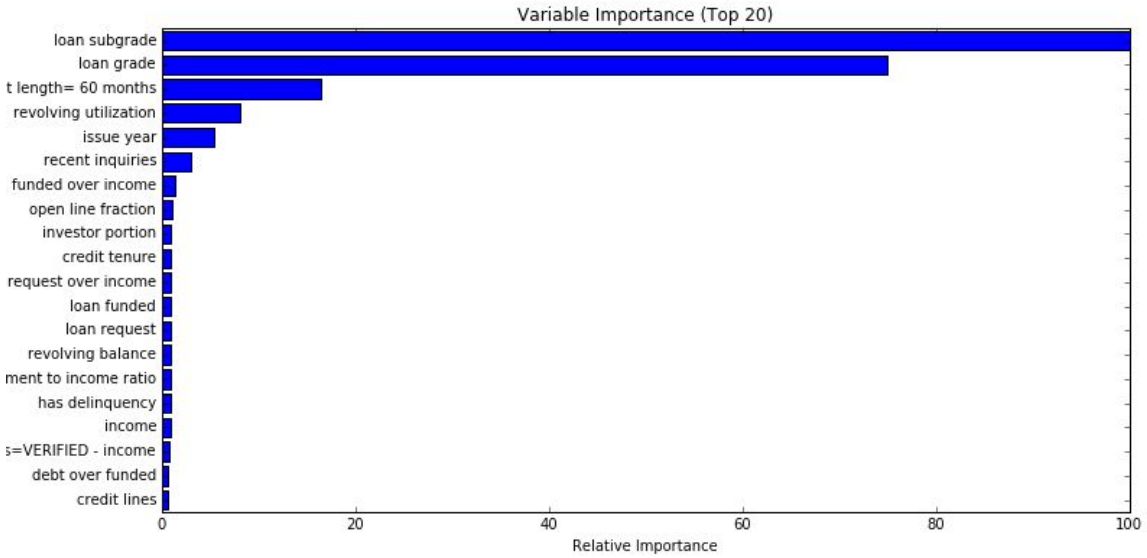
Linear Model

I built two least squares linear models. One with the full selection of features, and one with only 6 of the most important features (*loan subgrade, issue year, payment length= 60 months, revolving utilization, recent inquiries*). The simple linear model achieved a validation set RMSE = 1.87 percentage points, significantly better than the benchmark. The full linear model which was used for Holdout predictions achieved a validation set RMSE = 1.76 percentage points.

Random Forest Regression

The second model I built was a random forest ensemble regression. After tuning parameters, I ended up with a final model with 300 trees, each using a maximum of 30% of the available features, and with a minimum of 6 observations allowed per leaf. The resulting validation RMSE = 1.25, and was the best validation RMSE that I was able to achieve during this project. The feature importance for the 20 most important features are plotted here.

Fig. 2: Random Forest Model Feature Importance



Model Choice

Random forest models, and decision trees in general, perform best when observations are nonlinear and can be split easily with orthogonal cuts in features. However, tree-based models are still capable of building good models when observations, with the right parameter tuning. On the other hand, least squares linear regression models are strongest when observations are linear with features, and suffer otherwise.

In terms of computation time, the linear model is fit the training set in a significantly shorter time, running on my machine in 3 seconds compared to 300 seconds for the random forest model. Additionally, the random forest model requires tuning, which increases the time to build a model, whereas the linear least squares model does not require tuning. A redeeming factor for the random forest models is that it benefits from multi-core parallelization, if available.

One may choose a linear model as the results can be more interpretable and transparent than with the random forest model. This is especially true if you need to be held accountable for why specifically certain decisions are made based on your model predictions. With deep trees, tree-based models may have overly complicated interpretations.

A gradient boosting tree model was also attempted. However, given that the boosting tree models took me a longer time to fit the data, had more parameters to tune, and achieved similar RMSE as the random forest model, I had forgone boosting tree modeling. Additionally, random forest is not prone to overfitting, while gradient boost can be overfit. That being said, with more computation time and tuning, a boosting tree model could potentially produce a higher performance model.