**Problem**

Peer-to-peer lending marketplaces like LendingClub and Prosper Marketplace are preferred for their ease-of-use, off-the-shelf credit risk assessments score risk in grouped buckets. They are incentivized to increase the shear number of transactions taking place on their platforms. However, on a loan-by-loan basis, this is inefficient given each loan’s uniqueness and the sheer amount of data collected from borrowers. We are hoping to become more confident in their risk-reward assessments. What is the underlying risk that a borrower fails to make required payments, leading to a loss of principal and interest? Given the terms of the loan, what return can an investor expect?

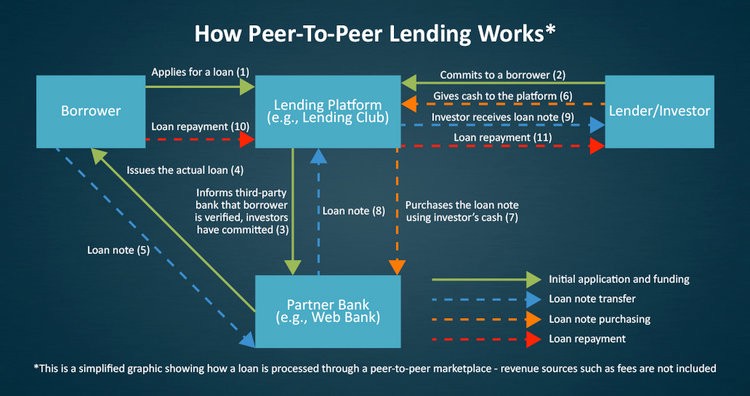
**Solution**

Given a user’s available funds, maximum risk tolerance, and minimum desired annualized return, recommend the best loans to invest in.

**Model**

1. Scoring risk by predicting the probability that a loan defaults
2. Predicting annualized returns

Peer-to-Peer Lending (P2P): The practice of lending money to individuals or businesses through online services that match lenders with borrowers. Because of their online-only nature and low overhead, generally, lenders can earn higher returns compared to savings and investment products offered by banks, while borrowers can borrow money at lower interest rates.



**Probability of Default**: an estimate of the likelihood that a borrower will be unable to meet its debt obligations.

**Annualized Return**: returns over a period scaled down to a 12-month period.

**\*\*\***The formula I used: AR = (xTP / xLA) ^ (365/D) — 1, where xLA is the loan amount, xTP is the total payment made by the borrower, and D is the number of days between loan funding and date of last payment. **\*\*\***

**Predicting the Probability of Default**

To assign a quantity to each loan that measures its risk level (risk that the borrower will default on the loan). All data is classified as 1 if it was fully paid and 0 if it was defaulted. Instead of a classification problem, we will output probabilities that it falls in each class (probabilistic approach).

Models to try:

* Logistic Regression (.predict\_proba() outputs the probability it belongs to class 1), e.g. not default, need to take 1 - % to know the probabilities it defaults).

Keep all features or do dimensionality reduction using PCA?

Using Feature Selection or using Deep Learning…

How can we encode string into features?

**Predicting Returns**

I used a very simple Annualized Return calculation based on the available data from LendingClub. I decided against using LendingClub’s complicated Adjusted Net Annualized Return computation due to interpretability, though as I continue iterating my model I am open to refined calculations that utilize more traditional financial loss estimates. The element of time as it relates to loan terms and payments sits within this calculation. Macro information like the federal interest rate, inflation and the time value of money is inherent in the interest rate quoted to the borrower, though there is certainly room to dive into time series and survival analysis with increased rigor to productionalize this model for financial institutions.

Linear Regression worked decently on the training set, but was overfit and did not generalize well to the future set or the test set within the same period. Ridge Regression, which is similar but applied L2 regularization fixed the overfitting issue in my 2nd iteration. That said, I wanted to know if my evaluation metrics would improve using something that could make regression decisions differently, and if there was a model that wouldn’t be as reliant on total payment so that my prediction could be used for a brand new listing on LendingClub without payment history. Random Forest was the perfect candidate given the overfitting problem and its proven ability to bootstrap subsamples of features and aggregate many decision trees. In simple terms, a Random Forest would be able to construct and combine many not-so-good models that are not correlated with each other to create a single, decent model — all while being equipped to handle my 1,107 features without much additional preparation besides finding the best parameters. Given the power of a Google Cloud GPU, I let GridSearchCV handle this by searching for the best combination.

**Underperformers**: 1 Linear Regression model, 3 Ridge Regression Models, 1 Keras Neural Network, 4 Random Forest Regression models (using Scikit-Learn)

**Best** **Performer**: Random Forest Regression v5 with GridSearch Cross-Validation

**Best Model Evaluation Metrics (Test Error — not including payment history):**

* R-Squared: 0.56
* Mean Squared Error: 0.02
* Root Mean Squared Error: 0.16

The best Random Forest regression model achieves a root-MSE of 0.16 on the test set, which implies that the predicted annualized return is estimated to differ from the true annualized return by 0.16. While this may appear very large at first, the model can be very useful in formulating a loan selection strategy. Loan defaults usually happen soon after loan funding, and the chance of default decreases as more payment is made. As a result, I recommend loans with annualized return predictions higher than a reasonable threshold set by the user. Intuitively, the threshold can serve as a parameter investor can tune according to their investment account size: the bigger the minimum annualized return (higher return) and the smaller the maximum probability of default is (lower risk), the more stringent the overall loan selection is, so less total dollars can be invested. Hopefully, the annualized return will be higher due to investing in loans more selectively.

That said, we can become much more confident in our prediction if we were to know something about the borrowers’ payment history, though in practice we won’t be able to incorporate initial borrow payment behavior as an investor. That said, with a bit of payment history to work with, the model improves significantly:

**Evaluation Metrics (Test Error — if payment history is included):**

* R-Squared: 0.97
* Mean Squared Error: 0.001
* Root Mean Squared Error: 0.04