1 Introduction [15 points]

- Group members: Benjamin (Ben) Juarez, Kyle McGraw, Dallas Taylor
- Kaggle team name: Darth Jar Jar
- Ranking on the private leaderboard: 8
- AUC score on the private leaderboard: 0.70335
- Colab link: Colab Code
- Piazza link: Piazza Post
- Division of labor:
  - *Data Wrangling*: Kyle initiated the data processing and wrangled the majority of the data. Ben and Dallas also helped finish up the data processing step. Ideas and methods were discussed as a team throughout the whole process.
  - *Modeling*: Each team member took the time and effort to research and create their own models. Each team member had at least one submission.
  - *Other Analytics*: Dallas and Kyle spent extra time analyzing and optimizing the hyperparameters for their respective models. Ben spent extra time performing feature analysis.
  - *Report*: Workload was distributed equally.
  - **Overall**: We collaborated very well and each team member had a similar workload. No significant discrepancy in the division of labor.

2 Overview [15 points]

Models and techniques attempted

Over the span of this project, we tried XGBoost models, a Random Forest Regressor model, and we even attempted a basic Linear Model. We found out quickly that Random Forest and XGBoost were the better choices for this project, and our highest performing model ended up using XGBoost. For each model, we used basic training and validation procedures. With the hyperparameters for each model, we initially manually manipulated the values to get a better sense of the effects before moving onto more advanced hyperparameter optimization techniques such as randomized searches and grid searches. Specifically with the XGBoost models, we also looked at feature importance, but we did not have the time to fully incorporate our findings with this into the final models. In regards to anything out of the ordinary, it seems clear that the choice of a Linear Model would not be ideal for this type of problem, but it was useful to see the differences in performance.

Work timeline

Our timeline started on Sunday night as our group got together after the necessary set up was performed (merged Kaggle teams, shared colab notebook, etc.). On this first night, we examined the data, researched and brainstormed potential models that would work well with the data, and developed a game plan for the next few days. The next day (Monday), we processed and wrangled the data, and by the end of the day, we had our first basic models complete including our first submission. On Tuesday, we continued the development and improvement of our models with multiple submissions throughout the day. We also took some time to perform randomized/grid searches to optimize hyperparameters as well as doing some analysis of the features. By the end of Tuesday, we were finalizing our two final models. On Wednesday, we had a few final submissions to check for any last improvements before selecting our final two models and seeing the competition results.

3 Approach [20 points]

Data exploration, processing and manipulation

We started out by exploring the data by looking at the type and number of unique inputs in each of the columns. We handled both the train and test sets identically, with numerical columns kept the same and categorical columns made into numerical columns. Categorical columns were either directly translated to numbers if there was a way that made sense (for example, emp_length was <1 year, 1 year, ..., 10+ years and could be encoded as 0-10), otherwise they were encoded using one-hot encoding (for example, ownership had columns for rent, mortgage, own, other, none) or dropped entirely (if there were too many unique categories, such as emp_title). For addr_state, since there were 50 unique categories, which was too much for one-hot encoding but still seemed like an important variable to keep, we used target encoding. We encoded all missing data as a unique identifier (-1) for all the columns, but had planned to do imputation on the missing data given more time.

Details of models and techniques

We decided to try out XGBoost for this project, because from what we had heard about it as an implementation of gradient boosted decision trees designed for speed and performance, it seemed like a good fit. We thought that decision trees would likely be the best model types for the dataset (hence our choice of random forest for our other model), and we wanted to try out a more complex model that used boosting in addition to our random forest.

We thus decided to also utilize Random Forest as an almost baseline performance related to decision trees. Creating a Random Forest model allows us to directly compare different model types that utilize complex decision tree functionality.

Iteratively testing different hyperparameters allowed for a quicker turnaround for checking potential improvements to the model, however, this process of guessing and checking left us vulnerable to missing out on deeper optimization using randomized searches and grid searches. However, performing these searches takes a lot of time, so the process with this step of the project would involve much longer waiting periods, but it did help us improve our performance more significantly. So, these different processes for testing hyperparameters had their pros and cons, but we seemingly utilized both techniques efficiently.

4 Model Selection [20 points]

Scoring

For both our XGBoost model and RandomForest model, we used the RandomizedSearchCV and GridSearchCV from the sklearn modelselection package. The optimization objectives we used for these was binary logistic because we are trying to pick the probability of a 0 or 1 class. Because the final scoring for our models in kaggle were roc auc, we also used roc auc to score the models we were running with RandomizedSearchCV and GridSearchCV. However, while we used roc auc average cross-validation scores to compare the different XGBoost and RandomForest models and select the best XGBoost parameters, we mostly (un-ideally) compared the different types of models using their public kaggle scores. If we were to try even more different models (that we talk about in the challenges section) than we did then we would need compare the models differently so as to not overfit to the testing set.

Validation and test

For both our XGBoost model and RandomForest model, we just split the data into cross-validation sets using the RandomizedSearchCV and GridSearchCV functions. We didn't really split the training data into validation or test sets, which is something we would have done given more time.

5 Conclusion [20 points]

Insights

With the highest performing XGBoost model, we performed some analysis on the feature importance. With this feature importance, we looked at three different measures: weight, gain, and cover. Considering the results of each of these measurements (plots seen in Google Colab Code), we see that the features of annual_inc, grade, sub_grade, int_rate, and dti (debt to income ratio) were the most influential features which logically makes sense when we consider the context of the problem regarding loan payoffs. With the XGBoost model, these conclusions were determined using the plot_importance() method along with changing the parameter importance type between weight, gain, and cover. Weight is related to how many times a feature is utilized to split the data across the decision trees. Cover is similar to weight, but this measure is also weighted by how many training points pass through the data splits. Gain is related to the the average training loss reduction that was gain when a feature is utilized for splitting. In terms of the weight measure, let us present our top 10 list of features by importance is as follows: *(1)* annual_inc *(2)* int_rate *(3)* sub_grade *(4)* addr_state *(5)* dti *(6)* term *(7)* loan_amnt *(8)* purpose_small_business *(9)* revol_bal *(10)* installment. Again, these ranking are in term of weight importance. With more time, we could have provided a deeper analysis of feature importance such that we could feel more confident about these rankings as well as incorporating them properly into an updated model.

One valuable takeaway from this project is that different tree-based models can have varying levels of accuracy and performance, but demonstrate the ability to perform very well and very similarly to each other once the hyperparameters are effectively tuned. Additionally, it is valuable to observe the importance of data wrangling and manipulation in order to produce the "best" possible dataset for training, where we want to maintain maximum correlation while minimizing necessary calculations.

Challenges

If we had more time, we would have handled missing data better. Instead of just encoding the missing data as -1, we would have taken the time to replace the missing data using imputation. While missing data might be a predictor, it would have been better to explore how much data was missing and figure out what would be the best way to handle it.

We also would have like to have tried other advanced gradient boosting models in addition to XGBoost to see if they might perform better on our dataset. In particular, we might have tried to create some models using LightGBM, a very similar model to XGBoost that grows trees leaf-wise instead of level-wise, and CatBoost, a gradient boosting model built specifically for categorical models.

One of the main obstacles that we faced while working on this project was time. Since we had a bunch of different ideas we wanted to try, we had to make the trade off between taking time to explore a bunch of different ideas vs diving into a single idea. This meant that some of the ideas that we wanted to try, such as imputation and other gradient boosting models, didn't get to be explored in order to have time to optimize the models that we did use.

6 Extra Credit [5 points]

For our grid search, we did use parallelization (although probably not in the most optimal way). When we ran our grid search code (which only searched for a single or pair of parameters rather than all parameters at once), we used 4 different threads as well as we ran multiple instances of the code at the same time for different parameters. This helped speed the process up, but we definitely could have utilized our resources even better to speed up our parameter search.

We would have been interested in trying to run our other models on a subset of features chosen from a feature selection technique (for exmaple, taking the 5 or 10 most important features from our XGBoost model). This would have been interesting to see if the linear and logistic models could perform as well as our random forest and XGBoost models if the task is just better suited towards a tree-based model.