1 Introduction [20 points]

Group members: Jacob Snyder, Chad Thut, Alex Janosi

Team Name: Rohan, Joe, and Netra

Placement on competition 1 private leaderboard: 54

Placement on competition 2 private leaderboard: 42

AUC score on competition 1 private leaderboard: 0.78315

AUC score on competition 2 private leaderboard: 0.77523

Division of labor: Equal across all aspects

2 Overview [20 points]

The first model we tried was a simple Decision Tree Classifier. Basic training and validation was used, until we deemed that Decision Trees were not a robust enough model class to fit this data set.

We then implemented a Random Forest Regressor, using the regression values as an approximation of the probability that a point equals 1. We tweaked parameters using cross validation.

We ultimately settled on the use of a Random Forest Classifier, extensively optimizing parameters using both cross validation and out of bag error.

We implemented Adaptive Boosting for the Random Forest Classifier, but we found using in sample error and one test submission that the classifier without Adaptive Boosting performed better.

We used some unique data processing techniques for all models. First, we removed all dimensions of the data that was the same for every voter (roughly 40 columns of data). We assigned binary (1 or 0) values to every dimension of the data which could be represented as binary (explained more later); this changed our performance very little but decreased computation time. The implementation of AdaBoost was also unique.

Our work timeline was cut short. We only had two solid days of full collaborative work. Before we began working collaboratively, however, we each spent time examining the data, doing basic data processing, and attempting basic models. The first half was spent brainstorming optimal models and testing parameters. The second half was spent trying new approaches and fine tuning the best models we had.

3 Approach [20 points]

We first attempted a simple Decision Tree Classifier because we thought it would be ideal to start simpler. This involved using sklearn to attempt different parameters. Although this method was simple to use, the simplicity did not allow it to accurately fit the complex data set.

We then moved to a Random Forest Regressor. This model was more complex than the Decision Tree, but we realized that it was not the correct model we wanted. The Regressor gave a decimal prediction, but we needed to train the data more accurately. Since the "target" column was binary, we decided to switch to a Random Forest Classifier. This gave a more accurate fit. We used the Out-of-Bag Error to decided which model was optimal. We used this because the oob_score predicts how well the model will fit to unseen data, which would increase our chances with the private leader board. After choosing a model based on oob_score, we tested the model using cross validation. Overall, the classifier became our best model. We believe this model was the best design, but it was very computationally expensive. Testing the models was very time consuming, and we ultimately did not get our best model. We attempted to use a Random Grid Search for the best parameters (number of trees, tree depth, etc.), but we ran out of time before this could finish processing.

We looked into different ways to preprocess the data in order to improve performance. However, there wasn't much that could be done in terms of preprocessing since we were planning on using a Random Forest (preprocessing isn't useful for random forests in most cases). We ended up writing a short function that would change binary columns with possible values a and b to columns with only 1's and 0's. That is, every a is changed to 1 and every b is changed to 0. This in theory simplifies the weights needed to account for that column.

We also tried an Adaboost implementation, based off of the Adaboost implementation we'd made in the previous set. We decided to try this because boosting seemed like a good idea to try on the data given. Upon plugging in our best parameters for the RandomForestClassifier, our Adaboost model performed well but not as well as the RandomForestClassifier model. However, we didn't have time to tune the parameters specifically for the Adaboost implementation so it's possible that given more time the Adaboost implementation would have performed better.

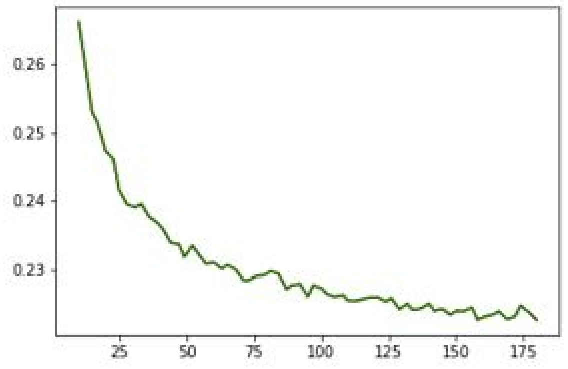
4 Model Selection [20 points]

Our optimization objectives were to test different parameters (max_features, min_samples_leaf, n_estimators) and to minimize out-of-bag error and validation error by varying that parameter. Through this method we could decide on a somewhat ”optimal” version of each model.

We scored our models based on the out-of-bag error and validation error metrics. We decided to use out-of-bag because it's a good predictor of out-of-sample error. We decided to use validation error because it's also a good predictor of out-of-sample error. Training error was taken into account in the sense that we made sure it wasn't unreasonably low, but we did not use training error as a primary metric for deciding on the optimal model (to avoid overfitting).

The models that scored the best were all Random Forests. Our best attempt was a RandomForestClassifier, using the optimal parameters found using the method described above. Our values for the parameters were n_estimators = 220, min_samples_leaf = 5, max_features = 50.

We used cross validation as our main method of validation. We also used MSE for simple validation testing, since the competition used AUC. We used the oob_score to initially choose our parameters. For example, one test was checking the score of basic RandomForestClassifiers with different number of trees. This is a generated graph:



This graph shows that, as the number of trees increases from 10 to 180, the oob_error decreases, which means that the model predicts unknown data better. We then picked a model that was somewhere in the right range for an individual test while compensating for other features. For example, based on the graph above, we would want more true, but this does not take into account the max depth of each tree, the minimum number of features considered for each split, etc. Once we picked a model, we would test it using cross validation on the training and testing data give for 2008. The error shown would tell us how to tweak our model, and we would run through this process again. Most importantly, we used these tests to give immediate feedback on our simpler tests, since the run time for our code increase immensely when we increased the parameters (for example, number of trees = 500).

5 Conclusion [20 points]

To calculate the features that had the largest influence on the prediction target, we normalized all data to be from 0 to 1 in their respective data dimensions, and then used the Random Forest Regressor's feature_importances to see which features had the largest influence on the prediction target.

We found that the most important features form the data were (followed by their respective importance):

$$\begin{bmatrix} PWLGWGT & PEEDUCA & HUFAMINC & PEAGE & PERRP \\ QSTNUM & HETENURE & GESTCEN & HWHHWGT & GESTFIPS \end{bmatrix}$$

$$\begin{bmatrix} 0.01995164 & 0.17642643 & 0.03370712 & 0.08862158 & 0.02240348 \\ 0.03656398 & 0.04913355 & 0.02505403 & 0.02009684 & 0.02429566] \end{bmatrix}$$

We believe that the Kaggle competition metric used AUC to make sure that our output was as accurate as possible. Since our output was a probability, the AUC method made the error testing more accurate, since we had to be close to a decimal. In binary classification, we would just have to check if the result was 0/1 for the error testing, but this method does a better job of checking the legitimacy of our results. The AUC method is calculated by taking the ROC curve, which tests the true positive and false positive rates, so the ROC curve takes into account the features of the model given. We do not believe that a better metric exists for binary classification. Most other methods of checking do not consider the false positive rates, so the models would be rewarded for something that is not accurately done. The simplicity of other methods is why we think the AUC method is ideal.

We found out that the parameter checking method we described in the Model Selection section could be paralleled through a random grid search. Instead of running a single feature test on a single computer, We would have had one computer run a full grid search on all of the parameters. Even having each person testing a single feature would have made the process more parallel. Unfortunately, we found these processes out too late, and these methods are extremely time consuming. This will be the most important feature we learned moving forward with the next project.

From this project, we learned how to find optimal parameter values to maximize a given output. We also learned how to compare models to one another and decide which one should be used. We learned how to do normalization from scratch (not using a package) and how to do some basic preprocessing of the data from scratch.

Overall, from this project we learned the dynamics of constructing and testing an optimal model for a machine learning task. In previous sets we had never had to think about what kind of model or what kind of preprocessing would be ideal for the given data set, so we had to decide based on the properties of the data what model we should use. We also learned how building a model on a timeline works start to finish. The time constraint was definitely a big factor for our group, having that as a parameter was something we'd never experienced before.

Primarily, we could have started some processes earlier. As our first real project with machine learning, we believe that we were naive with the time needed to fully train a good model. We ran into many issues with tests taking hours at a time, and we started processes before discussing with each other, which ended up wasting more time. For example, two members created tests that conflicted with each other, and ultimately, wasted an hour. The biggest major change would be creating a random grid search as soon as the project comes out. This process can take up to three days but gives amazing results.