Project Group 7 on Canvas
Kaggle Team Name: Houhong He
Members: Angelo Zhu 59714552 Boaz Chum 13869832 Houhong He 56553097
Leaderboard position: 95
Score of stacked learners 0.74625

Model Performance			
	Training Performance	Validation Performance	Kaggle Score
Neural Network	0.74	0.65	0.72
Gradient Boosting Classifier	0.688	0.653	0.740
Extra Random Trees	0.628	0.611	0.69
Ensemble/Stacked	0.73	0.72	0.746

Neural Network

For one of our classifiers, we decided to do a neural network. It was trained with the raw input of the data. Before we did any model training, we transformed the data using a standard scaler to make sure no one feature dominates another just because of a high variance. There was a lot of trial and error with the hyper parameters like the epochs and batch_size. We decided to settle the epoch at 100 because that was when there was a lot of diminishing returns on the training error and validation error, our training accuracy peaks about 80% and barely increases. There was also similar reasoning to the batch size. One thing that we noticed was that our model was overfitting by quite a bit in the beginning, like reaching 90% accuracy on test data but only 60% on validation data. We then decided to add the Dropout which results in a random chance of neurons being deleted to reduce overfitting. We also added kernel regularizers so that the model would penalize the weight matrices of the nodes. This helped bring the validation data accuracy to 65%. Finally, we submitted this individual model and got a 72% on the Kaggle test data.

Gradient Boosting Classifier

We trained a gradient boosting classifier using the sklearn library. When looking at the gradient boosting classifier, our main focus was figuring out the optimal number of decision trees to pick to refrain from overfitting our data. With a 25/75 training and validation split on the raw input, we plotted the training and validation errors for boosters in range(10, 200, 10). Observing that validation error rate was lowest around the 20 to 40 range, we then plotted the error rates for those trees and noticed that 30 trees had the lowest error rate. Furthermore, we plotted the ROC_AUC score for trees 20 to 40 and realized that 30 has the highest score. Thus, we concluded that 30 was the best value for our "n_estimators" parameter as it had low potential for overfitting while performing well.

We also explored the "max_depth" parameter with the intention of preventing overfitting. However, as we played with depth values around 3, ROC_AUC scores were lowering and error rates were increasing. We ended up sticking with the default max depth of 3, assuming that either our data set or n_estimators was no big enough for us to need to fix a larger max_depth to reduce overfitting.

Extremely Randomized Trees

Random forests are created from randomly sampling the training set, which is a set of decision trees. Selecting a random subset of features, and the algorithm finds the optimal split to create the root node. Here, we use the Extra Trees Classifier or Extremely Randomized Trees. This time, all the data in the training set is used and instead of finding an optimal split, we choose the split of a certain feature randomly. Since splits are chosen at random, it is much faster to compute than Random Forest, however it may be slightly less optimal. We used the scikit learn library to train my model on the data, using ExtraTreesClassifier().

The optimal parameters for this set of data was n_estimators = 80, max_depth = 40, and min_samples_split = 40. The curve for finding the optimal number of estimators grows very quickly until around 20 estimators and then slows down but still increases steadily. I chose 80 because that gave me the highest ROC_AUC score. For max depth of each tree, any value greater than 40 gives approximately the same score. The parameter for minimum samples to split by is much more varied, with peaks at 15~20 and at 40, anything higher gradually decreases the score.

Ensemble

For the ensemble, we decided to stack all the classifiers together where we let all of them vote. However, we realized that the gradient boosting classifier, was outperforming the other two by a large margin, 2-3% on the kaggle performance. Because of this, we decided to give more weight to the gradient boosting classifier where we have 10 gradient boosting classifiers, 5 neural networks, and 5 extremely randomized forests. This had a better kaggle performance than the other ensemble by approximately 1%.

Conclusion

We noticed that our gradient boosting classifier was the dominant force in predicting our because we were able to tune the parameters to reduce overfitting. This worked well, but we could have experimented with removing noisy data which may make this classifier fit even better. Neural networks was not as effective as the decision trees. We think this may be due to the large amount of features and not enough data engineering on the categorical and binary features. Extremely randomized trees were also not as effective as gradient boosting which is why we had the 10:5:5 ratio where we had the most effective classifier having the highest weight.