**CS 178 Project (Group 3; Kaggle team name: the boosters; Private Leaderboard rank #10)**

The first learner we tried was a random forest using the decision tree classifier from the mltools library. Initially, we used all features and trained it by attempting to tune the parameters, but we could not get better results than the results from homework 4 with this method. We then started using feature selection to use only the most informative features (the method of feature selection used will be described in a later paragraph), and ended up creating a **128 size ensemble random forest** using 13 features per bootstrap with a max depth of 10, a min parent of 64, and a min leaf of 4. Through rigorous testing, we found that these parameter settings gave us the best results when the ensemble was averaged at the end. The random forest gave us good results, but not good enough individually to achieve an outstanding score.

The second model we trained was an ensemble of linear regressors from the mltools library, using selected features. We created an **800 size ensemble of gradient boosted linear regressors** with a single normalization parameter of 5.2. This value was found by incrementing with a step size of 0.1 and optimizing over the AUC score. The boosted linear regressors produced sub-par scores individually, but when stacked with other models improved the average AUC.

The third model we trained was a gradient boosted ensemble of decision trees, again using feature selection to feed it only the best features. Through some trial and error, we ended up using a **128 size ensemble of gradient boosted trees** using a sigmoid loss function with a max depth of 3 per tree. Similar to the random forests, this model gave us good results, and we were happy to find that both ensembles of decision trees performed admirably.

The last model was a **simple single-layered neural network** using the mltools library with 1000 hidden nodes and a random initialization in order to avoid local optima. We started with a simple setting of the parameters as a baseline, and moved forward with testing other settings, however we found that the simpler the parameter settings the better the performance was in this case. Even with the best tuned settings we could find, this model did not individually perform as well as we thought it would. However, it ended up improving our performance within the final stack.

The final predictor we used is a **stack of the four separate learners described above** combined through a linear combination (linear classifier). Feature selection was used in every model in order to only use the features that would give the model the most information gain (45 features used). In the final stack, we used a linear classifier with a small regularization value (1e-10) in order to achieve a final output prediction.

We decided to use a stack of four learners to train this model primarily because of trial and error. We attempted each model individually, and although some of them individually had good results on the test data, we found that they were much more accurate when combined. Because of this, we turned our attention towards fine tuning the individual models and stacking them to achieve a diversely voted vector of predictions with high accuracy.

The biggest improvement in our model’s accuracy came from the use of feature selection. We found that when we used feature selection as opposed to using every single feature, our kaggle score jumped by ~0.005, and our leaderboard placement rose by about 70 places. Feature selection was performed by identifying and removing outlier features that adversely impacted the AUC score. These outliers were found by removing individual features and noting the largest resulting change to the AUC. We then applied feature selection to every model in our ensemble.

At the end, we were able to achieve a good result that we are proud of, with a private Kaggle score of ~0.755 and achieving 10th place. We attribute our success to two primary reasons: 1. The use of stacking in order to get a diverse group of learners, and, 2. The use of feature selection to remove features that did not have a strong learning power associated to them. The best improvements to our final model were made through the implementation of these two techniques. Additionally, we found that individually, **decision trees** (specifically random forests) worked better than any other learner that we tried to implement. The biggest disappointments to our group were **neural networks and linear regression**, as we were not able to achieve an AUC greater than ~0.67 for either of these models, no matter which parameter settings we tried. This may be due to the data not being linearly separable. As for our neural network, we believe it was getting stuck in a local optima, rather than overfitting, as shown by both the training and validation AUC being the same. Though, even after increasing our step size to try and break out of the local optima, we were unsuccessful in significantly improving its AUC.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Training Error** | **Validation Error** | **Public Kaggle Score** |
| Gradient boosted trees w/ feature selection | ~0.20 | ~0.20 | ~0.735 |
| Random forest w/ feature selection | ~0.18 | ~0.20 | ~0.74 |
| Gradient boosted linear regressors w/ feature selection | ~0.23 | ~0.23 | ~0.67 |
| Simple neural network w/ feature selection | ~0.23 | ~0.23 | ~0.66 |
| Final stack of four learners (random forest, gradient boosted linear regressors, gradient boosted trees, simple neural net) w/ feature selection | ~0.17 | ~0.20 | ~0.748 |