**Project Report**

|  |  |  |  |
| --- | --- | --- | --- |
| **Models** | Neural Network | Random Forest | Stacked |
| **AUC** | 0.92 | 0.965 | 0.98 |

For the neural net implementation, we decided to train it using all of the features. Separating the data based on discrete features or binary features does not provide any improvement when training the neural net. We trained the neural net using the mltools package provided to us by the professor. We had it learn on our training data and checked our performance with our validation data. Some key parameters that were set was that we only used one hidden layer which had 100 nodes within that hidden layer. We did various testing and found that more nodes did not give us better performance and more layers usually worsened our performance. Then when setting the activation feature, we decided to use a logistic function as that one provides the smoothest curve. The tank function since it as just a scaled sigmoid was able to run faster and found better results on the training data but did poorly on the validation data. Some of the last hyper parameters we set was the stoptol and stepsize. We set these in a way to take smaller steps and not stop immediately. We wanted the neural net to run many iterations and not stop early.

We chose to also include a random forest that was constructed with thirty decision trees. This inherently reduces how bias our predictions are on the validation data and test data and computational cost. Each individual tree was initialized with parameters in which were optimized to keep from overfitting the training data. The process we used to find the best value for each parameter was by computing and plotting the training and validation error rates of a chosen range of values. From there we used our own intuition to determine what value provided the best decision tree model based on the error plots. Bootstrapping was utilized in order to choose the random sample to be learned from each tree. A random sample size of the number of columns in the training data set with replacement for was pulled for each decision tree to be trained on. We made sure the number of times we bootstrapped the data was sufficient enough to ensure that the whole population of the training data was covered.

To further improve our random forest, we tried first by removing any individual tree that produced an AUC score less than the mean AUC score of all of the thirty decision trees. This helped filter out any abnormally bad decision tree learners in our ensemble which may throw our predictions off. We also tried implementing a function in order to extract any outliers using their IQR score in the training data which may be affecting our predictions. The effect of this was very minimal due to the knowledge that random forests are not heavily impacted by outliers. This is due to there being multiple predictions made by multiple decision trees all trained on different randomly sampled training data. After all of our attempts in improving our random forest, we received an AUC score of 0.70. This score shows our models performance is decent. After being content with the AUC score we received, we found the mean predictions of all the decision trees for the testing and validation predictions.

In order to export our final predictions to Kaggle easier, we made a function to easily convert the predictions of each model to a csv file. Each individual model’s test and validation predictions were stored in separate csv files. After we collected each model’s predictions for the test and validation data set, we horizontally stacked them together. With the horizontally stacked validation predictions we trained a linear classifier in order to make predictions based off our horizontally stacked test predictions. We picked a linear classifier since we found utilizing nonlinear classifiers did not significantly improve our results with the test predictions. A slight change in performance between a linear and nonlinear classifier shows us that the data set is linearly separable. So, with this known we chose to stick with a linear classifier in order to lower overall model complexity.

As discussed, we chose to investigate a neural network model, random forest decision tree model, and simple linear classifier. We expected our neural network to perform better, though its performance could be due to the length of time the algorithm was run. However, since we made the model run on smaller steps, this would have taken a long time to find a better model. In contrast, the random forest model was not overfitting, which could explain its higher AUC score. Therefore, for the set of data provided, the random forest model we finetuned was the best individual model. However, after combining the models, our stacked model of all three algorithms proved to perform better than the random forest model.