Project Group 70
Kaggle Team Name: uh
Leaderboard Position (as of Dec. 7): 103
Members:
Serena Tse -- sctse -- 21907178
Jenny Thu Nguyen -- thupn -- 66341423
Tammy Thai -- thait5 -- 46194673

CS178 Project Report

Model	Training AUC	Validation AUC	Leaderboard
kNN	0.6555342754688316	0.6273204833788633	0.53071
Random Forest	0.9476643563277867	0.7429059165436094	0.73029
Neural Network	0.7298712148520866	0.7100584793200146	0.69612
Ensemble	0.9058238372066639	0.7404703414892575	0.73214

K-Nearest Neighbor Classifier

We trained our k-nearest neighbor classifier using KNeighborsClassifier from sklearn.neighbors. This classifier was trained using 100 neighbors because we tried different k-amounts of neighbors and k=100 gave us the best result.

Because k-nearest neighbor does not perform well with high dimensions, we tried to select the k best features to use as our input, rather than all the features. To find the best k features we used SelectKBest from sklearn.feature_selection. Although we tried different values of the best features, the result seemed to perform worse than if we used all the features as the input, so we decided to use the raw inputs.

Random Forest

We trained our random forests with RandomForestClassifier from sklearn.ensemble. We decided to give the raw inputs as the data because random forest uses a bunch of weak learners to become a stronger learner. Since the random forest is an ensemble itself, it takes the average of all the nodes so even if one were an outlier it would not affect the result much because there are other data that can help to make sure the learner is not skewed towards underfitting.

We decided to set the max number of features to 106 because it gave us the highest validation AUC after testing and finding the maximum out of all 107 features using 1000 trees. At first, we chose to do 100 trees in the forest but then we decided to try different numbers of trees to see if it would make our results better, as we added more trees, our results turned out to be a lot better. It seemed like the pattern was that if

we had more trees, the result was better, so we decided to use 1000 trees because adding more did not do much to increase the AUC for our validation data and it took a long time to run the learner. We also decided to set the max depth to 10 after testing different depths with 1000 trees and a max of 106 features.

Neural Network Classifier

For our neural network classifier, we used a multi-layer perception that uses backpropagation to train with MLPClassifier from sklearn.neural_network. We trained our neural network using the all the features. We first tried training with alpha = 1e-5, 5 hidden layers, and default max iteration of 200 from sklearn's example on multi-layer perceptrons. This did pretty well (0.668 validation AUC) so we decided to try changing the number of hidden layers and number of iterations. After changing the parameters around, we decided on alpha = 0.0001, 2 hidden layers, and a max iteration of 256. The best solver for weight optimization we chose was 'lbfgs' which is part of the quasi-Newton method family.

Overall Ensemble

We chose to use a weighted voting using sklearn.ensemble.VotingClassifier. We chose to use this method to get our overall ensemble because our k-nearest neighbor classifier did not perform very well on the actual data. Since we could weigh the classifiers, we decided to make our random forest and neural network classifiers more heavily weighed so that our results would not be dragged down by the model that did not perform well.

Conclusion

The model that worked the best for this data is the random forest. We think that the random forest did particularly well because random forests, in general, do well. We hypothesize this is because we have such a large number of trees, 1000, that they average out any poorly performing trees. Another model that worked pretty well was the neural network. We think this worked because neural networks are able to better fit complex data, such as the one we were given with 107 features. On the other hand, our k-nearest neighbor classifier performed pretty poorly. We think this is because the data had so many features, making it too complex for k-nearest neighbor to accurately predict values. We also could not come up with a way to improve our k-nearest neighbor to do better. Overall, our ensemble did pretty well because we lowered the weight on our k-nearest neighbor and increased the weight on random forest, which performed well.