Anh Hoang Le - 22874310
Pratyush Muthukumar - 66495041
Junhan Ouyang - 53766701

CS 178 Project Report

For this project, we implemented all models using a combination of the scikit-learn and xgboost Python packages. Our best **public leaderboard AUC score of 0.80420** and **private leaderboard AUC score of 0.80319** came from a stacked ensemble model with three base learners: MLP, XGBoost, and KNN. At the end of the competition, we finished at **5th place** in both the public leaderboard and private leaderboard (**Team Name: Pratyush Junhan Anh**).

Model Type	Training AUC	Validation AUC	Public Leaderboard AUC
K-Nearest Neighbor	0.68234	0.68844	0.68059
Multilayer Perceptron	0.70284	0.69212	0.68161
XGBoost	0.75149	0.74643	0.75343
Random Forest	0.71254	0.70712	0.69947
Stacked Ensemble (MLP, XGBoost, and KNN)	**0.82923**	**0.81358**	**0.80420**

K-Nearest Neighbor. We provided 150,000 randomly shuffled samples of the 14 raw input features as training data to the model, and used the remaining 50,000 samples for validation. We experimented with normalizing the training data, but we found that normalizing the data did not improve the accuracy. We implemented the model with the *KNeighborsClassifier* model with K=250. We found the optimal K value using scikit-learn's *GridSearchCV*, which tests various hyperparameters of an estimator using cross-validation. Of the various K we tested (K = 100, 250, 350, 500, 1000), we found that K = 250 provided the highest accuracy. We found that uniform point weights performed better than distance-based weights which aren't effective in high dimensions. We also used a more efficient algorithm to find nearest neighbors for high-dimensional data called BallTree, provided by scikit-learn.

Multilayer Perceptron. For this model, we normalized the input data to have mean 0 and variance 1 as data normalization for multilayer perceptrons tends to improve accuracies. Similarly to the KNN model, we used 150,000 samples for training and 50,000 samples for validation. We again used *GridSearchCV* to determine the optimal number of hidden layers and number of nodes per layer. Our best performing implementation used the scikit-learn *MLPClassifier* model with 5 hidden layers of sizes [50, 50, 25, 25, 10] with ReLU activation functions trained on the Adam optimizer using an initial learning rate of 0.001 for 200 iterations.

XGBoost. For this approach, we provided the raw input data to the model as we found that data normalization did not improve the accuracy of this model. We split the dataset into training and validation as we have done so prior and utilized *GridSearchCV* to find the best hyperparameters. Our best XGBoost model was implemented with the *XGBClassifier* with 1000 boosting rounds, maximum depth of 8, a learning rate of 0.01, and a GBTree booster.

Random Forest. We used the raw data as input to this model, with the same 150k-50k training-validation split. We evaluated variations of the scikit-learn *RandomForestClassifier* with the following number of trees: [50, 100, 300, 500, 1000]. The model performed poorly for 50, 100, and 300 trees as the complexity of the model was too low and thus was underfitting the training data. Then after 300 trees, the validation scores remained steady but the training scores continued to improve. Therefore, we decided that 300 trees was the optimal amount, as the classifier started to overfit the training data for larger amounts of trees. Our best implementation using the *RandomForestClassifier* had 300 trees with a maximum tree depth of 15 where each tree in the random forest was implemented with the *DecisionTreeClassifier* from scikit-learn.

Stacked Ensemble. Our final prediction model was a stacked ensemble model with three base learners: MLP, XGBoost, and KNN. The reason we chose this combination of base learners is because we wanted to find models to best capture all of the feature relationships and variances of the data. We used the KNN base learner to capture similarities between training data; the MLP base learner to capture multi feature relationships that are difficult for tree models to capture; and the XGBoost base learner to capture complex nonlinear relationships within features. For all of the base learners, we first experimented with using the best performing set of hyperparameters we found when training their respective standalone models.

To combine the base learners, we first tried using a voting ensemble learning model to combine these base learners, but we found that the overall model complexity of a voting ensemble classifier with either soft or hard voting was not sufficient and thus performed worse than a stacking ensemble. Instead, we used the scikit-learn *StackingClassifier* ensemble model to use our base learners' learned representations as input to a Logistic Regression meta-classifier.

We trained the stacked ensemble learner on normalized input data and got a validation AUC score of 0.76643 and a public leaderboard AUC score of 0.75664. We decided to apply another round of *GridSearchCV* on the stacked ensemble learner to improve the results, as we believed that the hyperparameters for the base learners in their standalone variants may not be the best when used in an ensemble model. Running *GridSearchCV* on our stacked learner showed that the optimal MLP model had 6 layers with size [60, 60, 30, 30, 15, 5] and the optimal Random Forest model had 250 trees. This new version of the Stacked Ensemble was our best performance in the competition overall with a training AUC score of 0.82923, a public leaderboard AUC score of 0.80420, and a private leaderboard AUC score of 0.80319.

Conclusion. In conclusion, for a standalone model, XGBoost was the best performing as it had the ability to form complex nonlinear relationships within features. For the same reason, MLP and KNN classifiers weren't as effective by themselves. However, when we put all three models together using an ensemble learner, we can reach a much higher accuracy because we can capture richer relationships and variances among features. Finally, we believe the stacking ensemble learner could improve with more information about the input features, as we can apply complex feature engineering and encoding to improve the accuracy even further.