**CS 178 Kaggle Project Report**

**Kishan Rajasekhar (57609613)**

**Jason Kandu (23434725)**

**Kaggle Team Name: Isaiah Thomas #MVP**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Training**  **ROC-AUC** | **Validation**  **ROC-AUC** | **Kaggle Score**  **(Private Leaderboard)** |
| 1) KNN | 0.867312308946 | 0.706247494793 | 0.70773 |
| 2) Random Forests | 0.968757280831 | 0.719818134936 | .72541 |
| 3) Adaboost | 0.724944697866 | 0.699405989396 | 0.65361 |
| Blended Combination  (Voting Classifier) | 0.959052296333 | 0.741687620014 | 0.74339 |

\* We split the data with 80% used as training data (80,000 values) and 20% used as validation data (20,000 values)

**1) K-Nearest Neighbor**

We trained this model using the KNeighborsClassifier from the sklearn.neighbors module. As the instructions suggested, we performed a feature selection process to overcome the issues that came with high data dimension. Because there were 14 features, we used the sklearn.model\_selection module in order to cross validate the scores. By reshaping individual features from the X\_train data and measuring that against the Y\_train data, we took the mean of those scores, thus finding the accuracy for all 14 features. After ranking the features from best to worst, we took the top 10 features and used them for our KNeighborsClassifier. Initially, we reduced the large training & test data, but after running tests we realized that the original data sizes had the best performance. After further research, we discovered this because the KNeighborsClassifier was implemented with KD trees, which produces good results on high dimensional data space. However, as a consequence, it consumed a large amount of memory usage because the KNeighborsClassifier had to load the whole dataset into its’ memory.

For the parameter settings, we tested each setting extensively and found that most of the default settings produced the best results. We kept the default settings of 5 nearest neighbors, uniform weights for each point, and 30 leaves. However, we changed the distance to be Euclidean and the algorithm to KD trees.

**2) Random Forests**

We used the RandomForestClassifier from the sklearn module. We didn’t scale the data or added extra features, since Random Forests have their own method of reducing data and preventing overfitting.

We initialized our learner with our training data (80% of the data from the training file). We only allowed 10 features (out of 14) to be used for each tree. By allowing only a subset of features, we had a diverse set of learners with different results. This prevents the learner from being biased towards one particular model. We also enabled bootstrapping, which creates a random subset of data for each learner. This is used to prevent overfitting, since each learner will have different data.

We chose a subset of 10 features because adding more features would reduce the diversity of the learners; the learners would have too many similar features, which defeats the purpose of using a random forest. We also did not want our learner to lack enough features because there might be a possibility that each learner in the random forest might underfit the data. Therefore, after testing extensively, we decided that 10 was a good amount.

**3) Adaptive Boosting**

Boosting is a technique which starts of with a weak learner and trains it to predict the data better. We wanted to use a boosted learner because each learner aims to fix the errors of the previous learner, instead of randomly using another learner function. We thought it may yield better results than our other learners. We used adaptive boosting since we are doing binary classification, which means the output is either 0 or 1.

We used the AdaBoostClassifier from sklearn library. The base learner, or the weak learner, was a decision tree with a depth of 3. We tested out the auc score for each depth of the decision tree. As the depth increased, the auc score increased as well. However, since adaptive boosting expects a weak learner to start off with, we limited the depth to 3. We decided to use a decision tree because it is the most common learner used with adaboost.

**Overall Ensemble**

We combined our individual models using the VotingClassifier module from sklearn. We decided to use VotingClassifier because it computes a weighted average from the class probabilities of the three learners. Since our three learners use different types of classifiers (KNN, Random Forest, and AdaBoost), the class probabilities have a lot of variance, so we thought it was best to use a weighted average. Another reason we decided to take the average is because we wanted to retain the exact probabilities instead of losing information. For example, if the prediction for a point was 1, we wanted to know if the probability was 0.99 or 0.51, so we could get a better understanding of how strong the prediction was.   
 We specified the voting to be soft in order to predict class labels based on the sums of the predicted probabilities. We thought this would be more appropriate for our ensemble since we had well-calibrated classifiers. We assigned specific weights to each classifier so they could be multiplied by their respective classifier’s prediction probabilities. Finally, we took the average of all the predictions and picked the class label with the highest average probability.