Team: DKT
Daniel Seo, 8028887
Kenneth Yam, 61791166
Tony Tong, 68916837

CS178 Project Report

Model	Training Score	Validation Score	Validation AUC
Neural Network (MLP)	.84627	.70616	.69601
Random Forests	.86165	.72320	.75531
Extra Tree	.81041	.72315	.74939
Gradient Boost	.89011	.72905	.75725
AdaBoost	.71035	.70430	.69372
k-Nearest Neighbors	.66450	.63870	.63060
Final Stacked Model .77265(Kaggle)	.88330	.73170	.76461

One of the models our group trained is neural network classifier (nnet). We rescaled the data and trained the model using 80 percent of given data. The learning algorithm we used is MLPClassifier provided by scikit. With activation function set as rectified linear unit function and solver function set as “adam” which the nnet converges faster on large dataset, we tried to optimize the learner using three parameters: number of hidden layers, hidden nodes per layer, and max iterations. Initially, we trained with one hidden layer and 100 max iterations. We varied the number of hidden nodes from 100 to 1000. The average validation AUC was .6903 while the training AUC increased from .7056 to .7180. We suspected underfitting since training AUC was close to validation AUC, so we trained more models using 2 and 3 hidden layers. Although the average training AUC increased to .94, the max validation AUC remained around .70. We increased max iterations but it made the model overfit. To get the best out of the nnet model, we averaged the predictions of trained nnet models and used it to improve the ensemble performance.

The next model we tested is random forest classifier from scikit (RF). We trained the model using 80 percent of given raw data. There are two major hyperparameters to tune: n_estimators and min_sample_leaf, once these two are fixed, the others such as max_depth can also be determined. We tried 670 combinations of the two hyperparameters to find out the best. To accelerate this process we first vary both hyperparameters and narrowed down the range for one and then the other, like Figure 1 and 2 show. The final settings that give the best validation AUC is n_estimators=1160, min_samples_leaf= 5. We found that the more estimators or the fewer min_samples_leaf is, the more complex the model is and it is more likely to overfit. After RF, we tested a similar tree classifier: extremely randomized trees (ET). The overall tuning

process is very similar to that of RF. In the end, it gives slightly worse score than the RF, may because that ET tends to behave badly in high dimensional data-sets. One thing to note that is ET is twice as fast as RF since ET does not calculate the optimal cut of the tree.

Next, we tested boosted learners: Gradient boosting (GB) and AdaBoost. We trained the model using 80 percent of given raw data. There are three hyperparameters to tune: n_estimators, learning_rate and max_depth. We used similar searching technique as above and collected 184 combinations of the hyperparameters. Initially we searched n_estimators and max_depth while fixing learning_rate to be either 0.5/1. We found the when max_depth is greater than 10, the model is significantly overfitted, as Figure 4 shows, and very slow to train. So we then focus on searching models with max_depth less than 10. Then we found that the number of estimators does not have a strong correlation with the AUC score as Figure 3 shows. On the other hand, the learning rate has a strong influence of the score, 0.5 is doing consistently better than 1, shown in Figure 5. However, despite the fact that learning_rate lower than 0.5 did give better performance, we did not test it systematically due to time constraint. Instead, we used around 10 learning rates below 0.5 and found 0.3 gives the best performance. The learning rate actually acts like regularization and nicely balanced out the complex model. Then we tried Adaboost, which did not give promising score, probably because it is sensitive to the outliers as it tries to give them larger weight during learning.

Another model tested was the k-NN classifier from mltools. Initially, we used 10 percent of the training data due to this classifier being very slow when given a large number of X values. We decided to start with selecting two features. We first decided the best k for the data to be compared to. We found the best k to be 101. Afterwards, we tested each combination. The best pair only had an AUC of .6368. After comparing results with the validation data's AUC of .6317, we believed that the data is fitted relatively well, but the score was lower than we preferred. We increased the feature relations to three, which made the combination of features and time to train increase. The best 3-group's AUC was slightly better with .6434 but the validation data had .6306, meaning the data was starting to overfit. Increasing the X data would solve the problem, but take much more time. Compared with the other models that scored much higher, we decided that we will use the other classifier models.

Finally, we chose to use a simple weighted average method to get our ensemble model since it is easy to tune. We firstly locked down a range of weight for each model according to their score (higher score gets more weight) and tried 162 combinations to get the best one as Figure 6 shows.

The GB and RF classifiers worked particularly well may because they work well on raw data without rescaling. On the other hand, GB works better than random forest may because GB with a low learning rate is more robust to overfitting. Also since GB tries to reduce the bias as it moves closer to the target, it outperform RF which tries to reduce the variance on this particular dataset where some of the features have high variance. MLP classifier (nnet) was one of the models that worked poorly. The overall performance of our neural network model was not as good as we wanted. With combination of three different hyperparameters to choose from, the training process for each neural network model was quite slow. Therefore, we hypothesize that we potentially missed some combination that could boost the performance greatly. Also because we used raw data to train the model, we suspect that outliers lowered the overall performance. As for KNN, we suspect that the low performance is from lack of good feature selection algorithm.

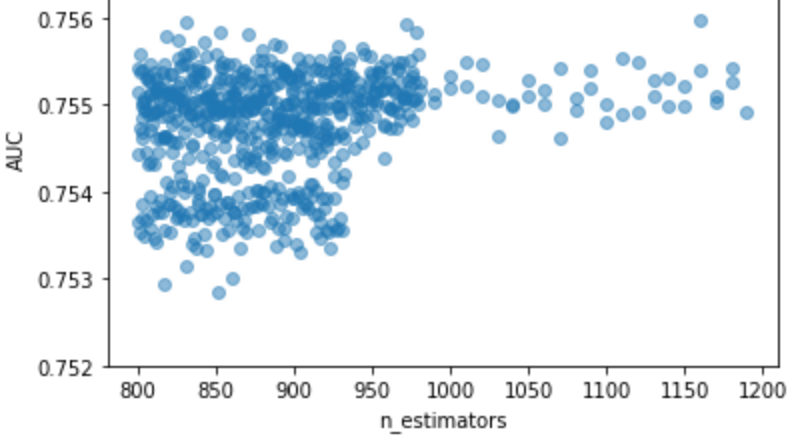
Figure 1 (AUC vs n_estimators for RF)
AUC
n_estimators

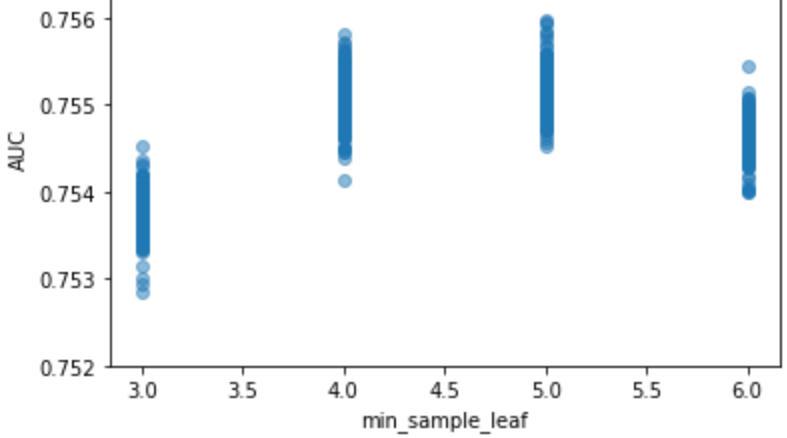
Figure 2 (AUC vs min_sample_leaf for RF)
AUC
min_sample_leaf

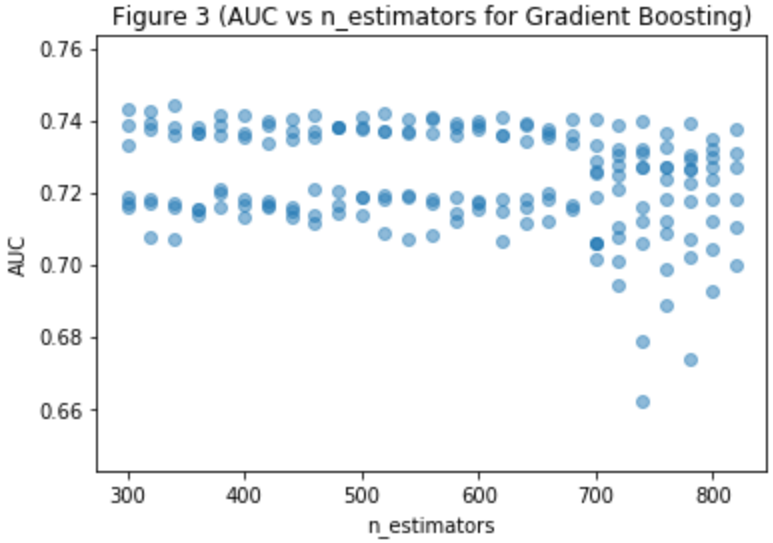
Figure 3 (AUC vs n_estimators for Gradient Boosting)
AUC
n_estimators

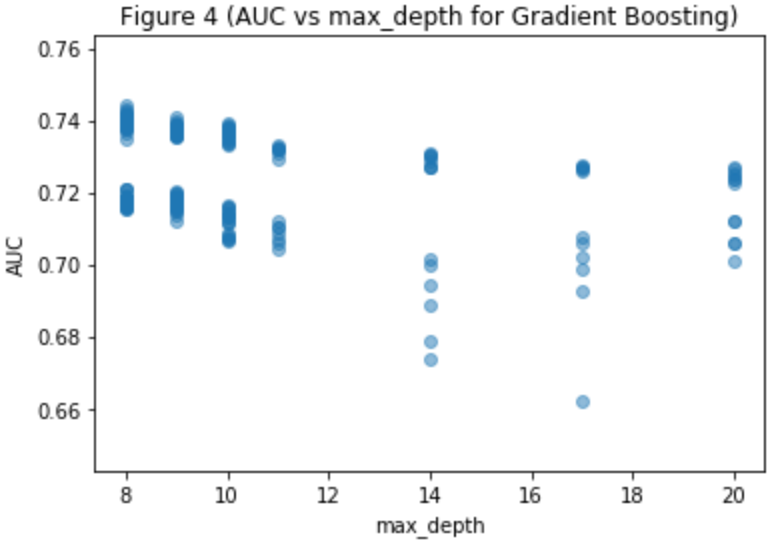
Figure 4 (AUC vs max_depth for Gradient Boosting)
AUC
max_depth

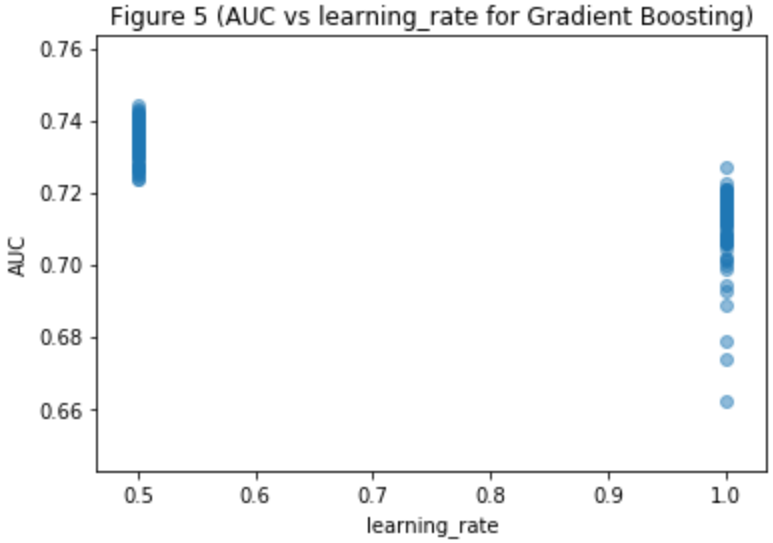
Figure 5 (AUC vs learning_rate for Gradient Boosting)
AUC
learning_rate

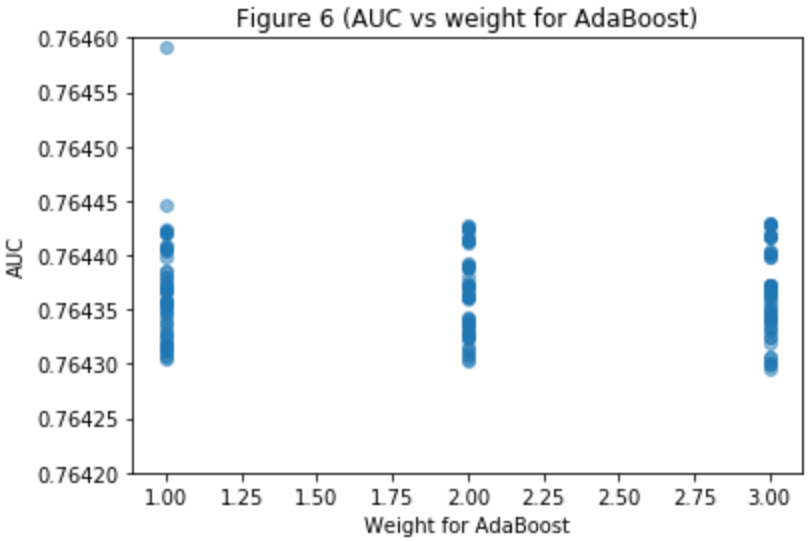
Figure 6 (AUC vs weight for AdaBoost)
AUC
Weight for AdaBoost