Team 26

Team name: http://bit.ly/2FPnPV7

Member 1: Zhaomin Li, 11671379

Member 2: Yufei Wang, 13911282

Models	Training/Validating AUC	LeaderBoard AUC
Linear Model	Training: 0.74004 Validating: 0.71839	0.72318
Random Forest	Training: 0.96983 Validating: 0.78383	0.78810
Gradient Boosting	Training: 0.97974 Validating: 0.78481	0.79583
Blending Models	Training: 0.97661 Validating: 0.79450	0.80090

Linear model:

We use non-linear feature expansion to adapt the original input to degrees from 1-4. Therefore, we transformed the linear regression to polynomial and fitted the model with a lot of data points.

We first shuffled the raw input, and we use K-folders(4) cross-validation. We split the data into four folds and use one for validation. We repeat the test-validation for four times to find the best model and avoid overfitting. We use the *PolynomialFeatures* from sklearn to transform input into polynomial and use *LinearRegression* to fit the regression model.

The major parameter is the degree of the polynomial. We used for-loop to adapt the inputs with different degrees and use the test/validation error-rate/MSE curves to find the best degree of all. We selected degree 4.

Random Forest:

We randomly split the input data into training and validating data (0.75: 0.25). And we would bootstrap the training data due to the concept of Random Forest.

We use the *RandomForestClassifier* in the sklearn library. We set the number of the trees(n_estimators) to 1000, so it generates 1000 trees from random vectors with the same probability distribution and uses the majority voting to decide the prediction. For testing, we calculate the probability of predicting one rather than the hard prediction.

First, we set the max tree depth to 20, the minimal size of a splittable parent to 10, and the minimal size of leaf to 4. These parameters are used to limit the size of the trees to yield a better prediction while avoiding overfitting. We use for-loop to find a proper value for each parameter and evaluate with error loss of training and validating data. Second, we set the oob_score to true so that we can use out-of-bag samples to evaluate the accuracy of the generated forest.

Gradient Boosting:

We randomly split the input data into training and validating data (0.75: 0.25). We use the training data to fit the regression tree

We use the *GradientBoostingClassifier* in the sklearn library. We set the number of boosting stages(n_estimators) to 1000, so it performs the boosting process for 1000 times. Gradient boosting is fairly robust to overfitting so that we choose 1000 as stage number to yield a better performance. We use the deviance loss function(loss = 'deviance') to optimize MSE since we have the binomial prediction and we want probabilistic outputs.

For the other parameters, we set the max tree depth to 12, the minimal size of a splittable parent to 12, and the minimal size of leaf to 5. We use for-loop to find each proper parameter and evaluate with the error rate of both training and validating data. Setting these parameters can limit the number of nodes in the tree, preventing the overfitting and tuning the performance. The shrinking rate is 0.1 which is good enough for each step.

Overall Ensembles:

We use majority voting to build our ensemble model and we combine the predictions from Random Forest and Gradient Boosting(the two best predictions) by using the soft voting (*VotingClassifier* with vote = 'soft'). In this way, we can predict the class of results based on the argmax of the sums of the predicted probabilities, without generating a hard prediction which may lower the AUC performance.

Based on the majority voting rule, if the two models have the same prediction on one data point, then the final prediction would be consistent with the two models. We assigned a higher weight to the Gradient Boosting prediction(1.5: 1) than the Random Forest prediction since the Boosting has a better prediction performance. Therefore, the boosting model would play a bigger but not deterministic role when the two predictions are contradictory.

Conclusion:

According to the leaderboard performance, the random forest, the gradient boosting, and the ensemble of the two models work pretty well. For the **random forest model**, the randomization can reduce computation time when we have a huge dataset. Fitting multiple trees can result in less variance because we have better chance to avoid the bad classifier. And by averaging the numerous trees, we effectively reduce in overfitting. For the **gradient boosting**, it starts from a fairly good prediction and keeps reducing the MSE after each boosting. The robustness of the Gradient Boosting also guarantees its performance. Furthermore, the **ensemble** model combines the good predictions from two models above and assign the better model with a higher weight. That is the reason why the ensemble model has the best prediction. The **linear model** performs poorly. There are many data points, a number of features (14) in X, and only binary predicting results. so it's hard to fit all the data points into space with a proper degree and low MSE. And it is really time expensive.