Canvas project group: Group 61
Kaggle team name: Dunno Name
Member 1: Tao Wang, 76254921
Member 2: Jiliang Ni, 81224742

Part 1

Models	Training AUC	Validation AUC	Leader Board AUC
Linear Regression	0.751729920672	0.71702103368	0.72
Random Forest	0.9514777898889919	0.7863389729786047	0.7835
Gradient Boost	0.9764663562182002	0.7995	0.794
Ada Boosting	0.9889990307412349	0.7949058343945232	0.77
Blending Models	0.9772733380984986	0.7958278932110232	0.792

Part 2

Linear Models:

For inputs, we used non-linear feature expansion to normalize the data input with degrees from 1 to 4. Then, we changed the linear regression to polynomial. And we also used all features of data input. And we shuffled the data and used K-fold cross validation with k = 4 to split the data into 4 folds. Each time, we picked a fold as validation data, and trained other data to find optimal model.

We trained this model by using PolynomialFeatures from sklearn.preprocessing to change data to polynomial and using LinearRegression to fit the model.

For choosing parameters, we changed degree of PolynomialFeatures from 1 to 4. And we used a for loop to get training AUC score and validation AUC score. After comparing the results, we find the best value for degree is 4. And the score results is training 0.751729920672, validtaion 0.71702103368 and Kaggle 0.72.

Random Forests:

For inputs, we normalized input data by using model_selection.train_test_split method from sklearn, which will split the data into training data and validation data suitably. This method will also use a random argument 'random_state' to do similar thing like shuffle, and we assign random_state = 42 here. And we use all features of training data to train RandomForestClassifier.

We trained this model by using RandomForestClassifier from sklearn.ensemble. In fact, we tried the algorithm in the lecture slides, but it ran really slowly, so we choose to use sklearn library's random forest. Also, we used a method called predict_prob which is similar to predictsoft method in mltools library. After using predict_prob, the AUC score on Kaggle increased from 0.65 to 0.78, which is a huge improvement.

For choosing parameters, we used GridSearchCV method to check the performance with different parameters. It will return a gridsearch.best_score_ for different parameters, showing the performance. Finally, we found n_estimators=1000, min_samples_leaf=4,max_depth= 25, oob_score=True are the best values for these parameters. Also, we used a for-loop to check AUC score under many parameters combinations. And we also found these values will lead a highest validation AUC score without overfitting. And the training AUC is 0.9514777898889919, Validation AUC is 0.7863389729786047 and kaggle score 0.7835.

Boosted Learners(Gradient Boost):

For inputs, we normalized input data by using model_selection.train_test_split method from sklearn, which will split the data into training data and validation data suitably. This method will also use a random argument 'random_state' to do similar thing like shuffle, and we assign random_state = 42 here. And we use all features of training data to train GradientBoostingClassifier.

We trained this model by using GradientBoostingClassifier from sklearn.ensemble. We initially set n_estimators as 500, then it will do boosting process 500 times. The gradient descent procedure will decrease loss.

For choosing parameters, we used GridSearchCV from sklearn to look for best values for each parameter. Because using for-loop to find best values are really slow, almost several hours, we used GridSearchCV. Then we found when n_estimators increase, the performance will increase. Until 1000, the performance is almost unchanged. Using GridSearchCV, we also found the optimal values for parameters, like n_estimators=1000, max_depth=12, min_samples_split=12, min_samples_leaf=5. And the training AUC is 0.9764663562182002, validation AUC is 0.7995 and Kaggle AUC score is 0.794. Finally, with best values of parameters, we can get best validation AUC score without overfitting.

Boosted Learners(Ada Boosting):

For inputs, we normalized input data by using model_selection.train_test_split method from sklearn, which will split the data into training data and validation data suitably. This method will also use a random argument 'random_state' to do similar thing like shuffle, and we assign random_state = 42 here. And we use all features of training data to train AdaBoostClassifier.

We trained this model by using AdaBoostClassifier from sklearn.ensemble. And we used decision tree as base model. For choosing parameters, we used GridSearchCV from sklearn to look for best values for each parameter. Because using for-loop to find best values is really slow, we first set learning_rate as 1.1 and called GridSearchCV to get gridsearch.best_score_. But the score is not really good. Then we put a list of values into GridSearchCV and finally we found when learning_rate = 0.001, the gridsearch.best_score is the highest. Using GridSearchCV, we also found the optimal values for parameters, like n_estimators = 800. And the training AUC is 0.9889990307412349, validation AUC is 0.7949058343945232. Finally, with best values of parameters, we can get best validation AUC score without overfitting.

Part 3

Overall Ensembles:

We used VotingClassifier to combine our first three models with highest Kaggle AUC score to our ensemble model. And we pick Random Forest, Gradient Boost and Ada Boosting models, because these three models have highest Kaggle AUC scores. And we set the parameter voting in VotingClassifier as 'soft', because it will cause the classifier to predict by argmax of sums of predicted probabilities, which may improve the performance. Also, we set the parameter weights in VotingClassifier as [1:2:1], because the Gradient Boosting model has the highest Kaggle AUC score from all models, then we should let it has more important role in the VotingClassifier. Finally, the blending model gets better performance than Random Forest and Ada Boosting models and it is close to Gradient Boost model.

Part 4

Conclusion:

According to the Kaggle AUC scores, we finally found that Linear Regression works most poorly in these four models. And the Gradient Boost model works best. And the ensemble model also works well and is close to the best model Gradient Boost.

For the reason that Linear Regression works poorly here, I think it is because we have few input features but large amount of training data, so the dimension is not enough to make the data separable. And the linear model will not perform well on this non-separable data set.

For the reason that Gradient Boost works well here, I think it is because it trains learners sequentially and focus later predictions on correcting previous errors. Then it converts many weak learners to a complex predictor. And it generally has good performance.

For the reason that Ensemble model works well here, I think it is because it will combine good predictions from all models and give a higher weight to the better model. Then it generally will have a better performance.

What's more, when we want to find optimal parameter values, using GridSearchCV really saves time and it can help us find optimal values quickly.