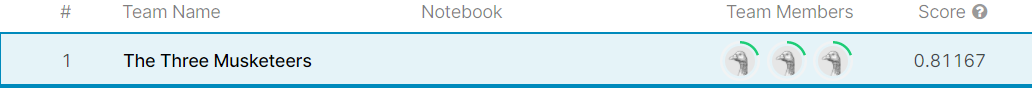
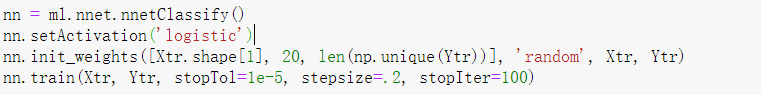
|  |  |  |  |
| --- | --- | --- | --- |
| Model name | Training AUC score | Validation AUC score | Kaggle AUC score |
| Neural Network | 0.64942 | 0.65256 |  |
| Gradient Boosting | 0.78415 | 0.78219 | 0.77650 |
| XG Boosting | 0.79976 | 0.79924 | 0.80003 |
| Ada Boosting | 0.69534 | 0.69214 |  |
| Random Forest | 0.79232 | 0.79226 | 0.79923 |
| Bagging | 0.80132 | 0.80100 | 0.80140 |
| Weighted average | 0.81134 | 0.81082 | 0.81167 |



Besides python libraries we used in class, we also import sklearn library.

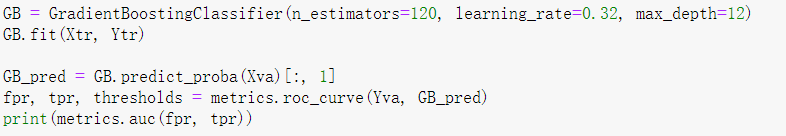
We use sklearn.preprocessing to handle the original data and use sklearn.metrics to calculate AUC scores. In every model, we use transformed Xtr and Xva with Ytr and Yva shown in the screenshot to train the model and test their performance.

Neural Network:



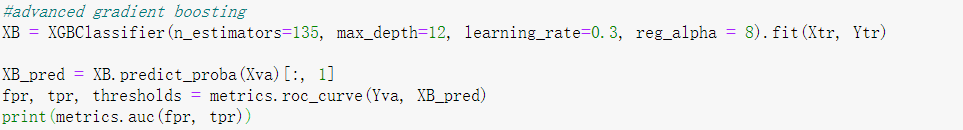
We use the MLTools library to train our neutral network classifier. We initialize its input and output layers using Xtr and Ytr’s shape and initialize all values all random small values. We set the activation function to be the logistic function provided. We tried those hyperparameters based on previous experience, but its validation AUC score is relatively low. So we didn’t submit it on Kaggle and go for other methods.

Gradient Boosting:



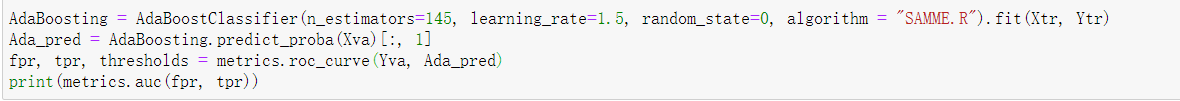
We use sklearn’s GradientBoostingClassifier. It builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. N\_estimators is the number of boosting stages to perform. We learn that gradient boosting is fairly robust to over-fitting, so a large number usually results in better performance. We choose some values above 100, and it makes little difference when n\_estimator is larger than 120. So, we decide to use it for a relatively faster training speed. Learning rate shrinks the contribution of each tree by its value. We use cross-validation to test several combinations of n\_estimator and learning rate to check the model’s MSE and validation auc score. We found that it behaves its best when n\_estimator = 120 and learning\_rate = 0.32. We also use a cross validation to find the best max\_depth using the idea of binary search from range 10 to 50 (it allows us to find the best value faster).

XG Boosting:



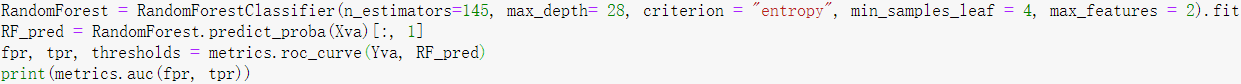
XG boosting is an advanced gradient boosting method from xgboost library. Its hyperparameters are akin to gradient boosting’s. But for this classifier, we find that, via cross validation, it achieves its best performance when n\_estimators = 135 and learning rate = 0.3. The reg\_alpha is the alpha value of L1 regularization. We test the classifier’s performance with a range of reg\_alpha from 1 to 25, and we find that it produces its best AUC score when equals to 8.

Ada Boosting:



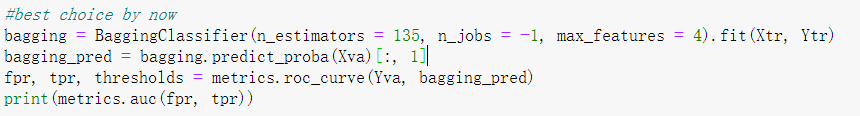
We use sklearn’s AdaBoostingClassifier. Its hyperparameters is akin to gradient boosting. We use cross validation to find its best values. The “SAMME.R” algorithm here can achieve a relatively lower test error in a faster way. However, its AUC validation score is too low, so we give up this model.

Random Forest:



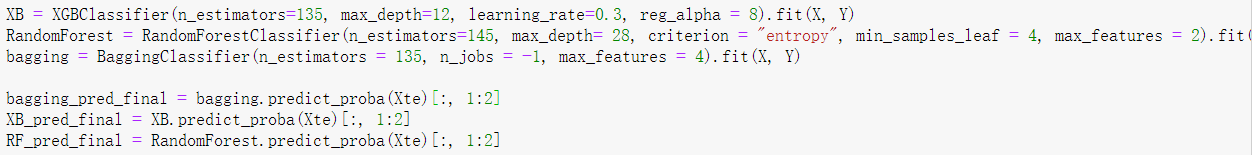
We use the sklearn.RandomForestClassifier. It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. N\_estimators is the trees’ number. Min\_sample\_leaf is the minimum number of samples required to be at a leaf node. We use the cross validation to find that it achieves the best performance when n\_estimators = 145, max\_depth = 28, and min\_sample\_leaf = 4. Criterion is the function to measure the information gain. We find that using entropy can give a higher score. Also, when we set max\_features to 2 instead of a default of sqrt(n features), the score increases a little.

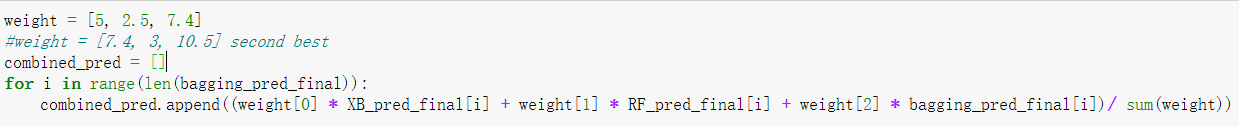
Bagging:



We use the sklearn.BaggingClassifier. It’s an ensemble meta estimator that fits base classifiers each on a random subset of the original dataset, and then aggregate each prediction to form a final prediction. N\_jobs indicates number of jobs to run in parallel. We set it to -1 which means to use all processors. Max\_features is the number of features to draw from X to train each base estimator. We find that it achieves the best performance via cross validation when max\_features = 4.

In the end, based on the AUC score, we choose to use a weighted average prediction via three models: XG boosting, random forest, and bagging.





Based on their Kaggle AUC score performance, we give each of them a weight and average the prediction. We give higher weights to bagging and XG boosting because they have a relatively higher AUC scores in previous individually trained model. For Random Forest, we slightly weaken its effect on the final prediction. By doing this, we thought that we could further alleviate errors. As the result shows, it indeed improves our score to 0.81 compared with 0.80.

In a nutshell, two ensemble methods, XG boosting and bagging, work particularly well for this data. It probably benefited from gradient boosting’s optimization method and bagging’s features of an improved meta estimator. They both can avoid overfitting to some degree, which also could be a key reason when processing a relatively large dataset. The neural network works poorly for this dataset. The main reason probably is that it’s trapped in local optima. Besides, the activation function may not be appropriate for this dataset.