**Introduction**

In the project, our goal is to build a classifier to predict two classes of quantum physics data under several performance metrics, including accuracy, area under ROC curve (auc), cross-entropy, and SLAC Q-Score. The training data consists of 50,000 entries, each with 78 features. The tools we used in this project include Matlab, Weka, and Python (especially the scikit-learn module). The best model we learned is a Gradient Boosting tree with 750 iterations, and the average rank on the KDDCUP website we got is 4.75, with accuracy = 73.23%, AUC = 0.8278, cross-entropy = 0.7168, and SLAC Q-score = 0.3276.

**Model selection**

Based on the models we tried in the first part, we decide to use Gradient Boosting tree classifier as our optimized learner. We used 5-fold cross-validation to test the training and validation error, and the select the best parameters. The final parameters we chose is a Gradient Boosting tree with 750 iteration and maximum depth 3, and the best validation accuracy is 0.7335.

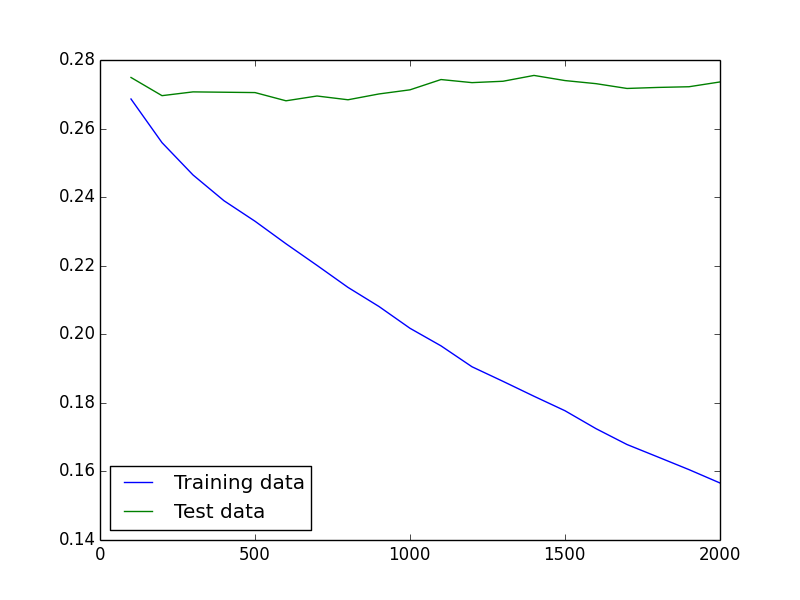


Figure. Error rates versus iterations

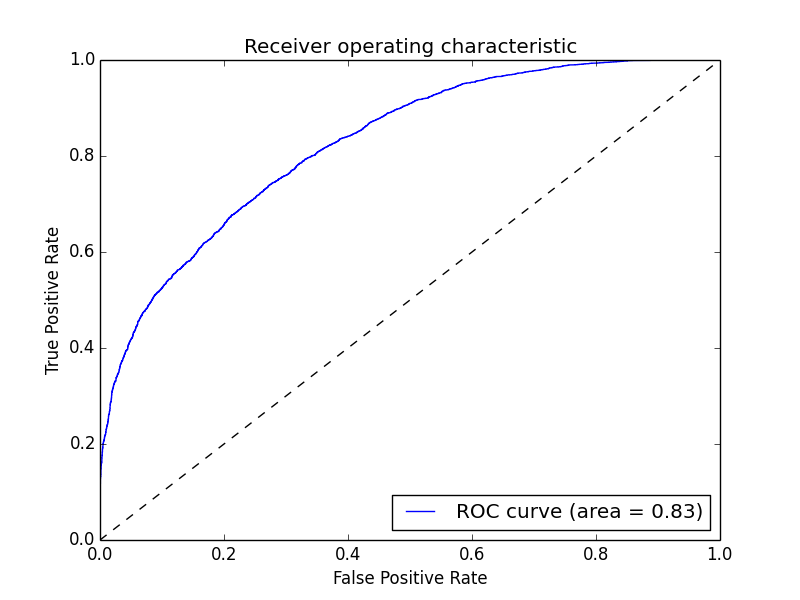


Figure. ROC curve of Gradient Boosting tree with 750 iterations

**Data preprocessing**

1. Rescaling

We tried (1) rescale the data to zero mean, and (2) normalize the data, assuming the values have a normal distribution. Normalization does not give us a better performance, so we simply chose the simple zero mean rescaling.

2. Missing data

In the training data, there are some features with missing values. To deal with this problem, we tried (1) using the data directly, pretending there are no missing values, (2) ignoring the features with missing values, and (3) imputing the missing values with the mean of other values of the same feature. It turns out that (3) gives us the best performance (accuracy and AUC), while (1) and (2) are much the same.

|  |  |  |  |
| --- | --- | --- | --- |
| Metric\Method | (1) All features | (2) Ignoring missing | (3) Imputing |
| Training accuracy | 0.7823 | 0.7835 | 0.7839 |
| Validation accuracy | 0.7295 | 0.7321 | 0.7335 |
| AUC | 0.8261 | 0.8266 | 0.8267 |

Table. Different missing value preprocessing, using a Gradient Boosting tree with 750 iterations

**Feature selection (need PCA plot!)**

We also tried PCA to see if dimensionality reduction helps. However, we found that in the PCA, with more features, the validation error tends to be smaller, and we concluded that there is no need to do a PCA before learning the classifier.

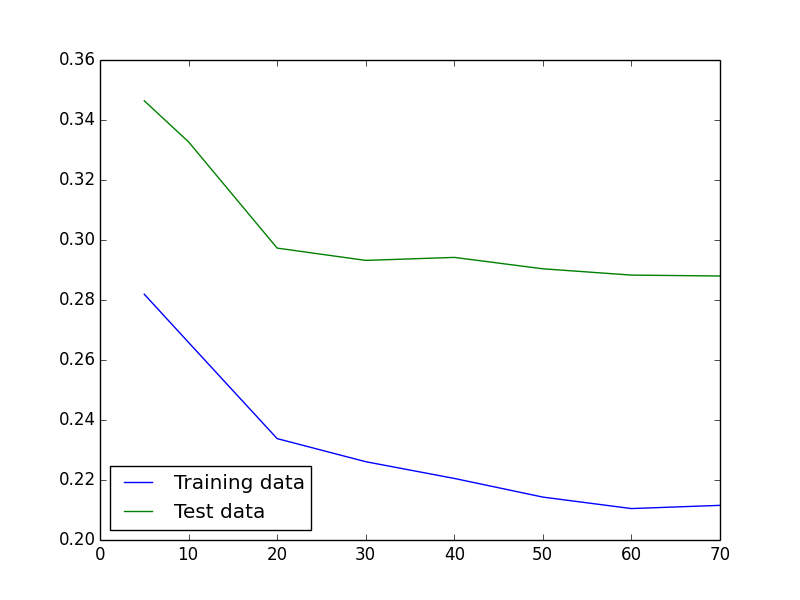


Figure. Error rate versus number of principle components used in the model

**AUC (Area under ROC curve) optimization**

In addition to optimizing the overall prediction accuracy, we also try to optimize other performance metrics, including AUC. We noticed that using “hard” scores (that is, only 0/1 as predictions) will give us bad AUC, but using “soft”, un-thresholded scores (that is, probability estimates as predictions) can achieve better AUC. In Python’s scikit-learn module, after we train the classifiers, we can get the probability estimates easily, and we used the estimates as our predictions. Before using soft scores, we get AUC around 0.732, but we improved it to 0.8267 with soft scores.

**Cross-entropy and SLAC Q-score optimization**

Although we did not explicitly optimize these two metrics, we found that after optimizing AUC, these two metrics became much better.

**Stacking**

Based on the models we believe as good, we also tried to construct a classifier using weighted vote of multiple classifiers (stacking). The way we trained the classifier is to use perceptron with several classifiers (SVM, Random forests, AdaBoost, Gradient boosting) as input, and generate a weighted, soft (un-thresholded) probability estimate as the output. The best validation accuracy we got is about 72%, which is still worse than the accuracy (73.35%) we got when only using a single gradient boosting tree classifier.