Assignment 1 Analysis

Introduction

For this assignment, we will be using two different datasets with different sizes and numbers of attributes: **spam_base.csv** and **magic_gamma_telescope.csv**, and both datasets fall under our binary classification problems. For each dataset, we pick 30% for testing at random to avoid data bias, and the 70% remaining is for training. We also train each algorithm with and without cross validation (CV) to see the effect of overfitting or underfitting. For each algorithm, we will measure how well they perform by calculating the confusion matrix, accuracy, Receiver operator characteristic (ROC) curve, and Area under the curve (AUC) value. We also record how long each algorithm takes to perform training and classification.

Definitions for the metrics we are using:
- Confusion matrix or an error matrix is a 2x2 table that reports the number of false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN).
- Accuracy is the percent of correct predictions that the classifier gets.
    Accuracy = (TP + FP) / # Total predictions
- Error is the percent of misclassification
    Error = 1 - Accuracy
- Receiver operator characteristic is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold varies. A good classifier would fall into the top left corner of the graph with true positive rate close to 1 and false positive rate close to 0
- Area under the curve characterizes the performance of a classification model. The closer to 1 it is, the better the performance of the classifier.

Experiment and Analysis

1. **Spam Base dataset**:
   This medium-sized dataset contains 4601 records and 56 continuous features. The class label represents spam or not spam email, and the features represent the frequency of certain words contained in the emails. The reason this dataset is interesting is that it has quite a lot of features and a reasonable amount of instances which can bring out the different characteristics of each algorithm.

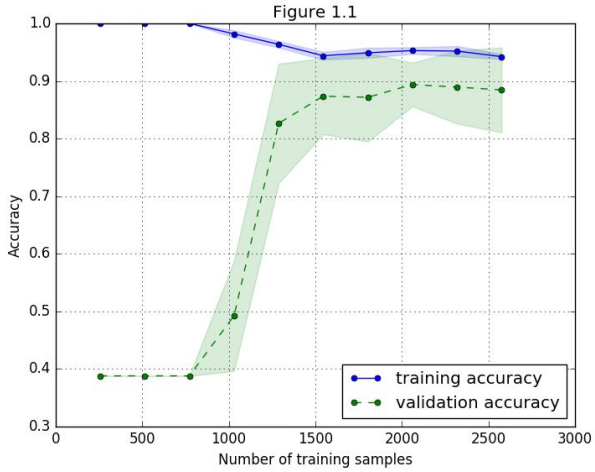
   a. Decision Tree with pruning Analysis:
   In this section, we will be using the extended version of the DecisionTreeClassifier algorithm provided by sklearn framework to run the analysis, and information gain is used as a splitting criterion at each node. After running the initial training on the default model provided by the framework, we obtain a decent result for our model:

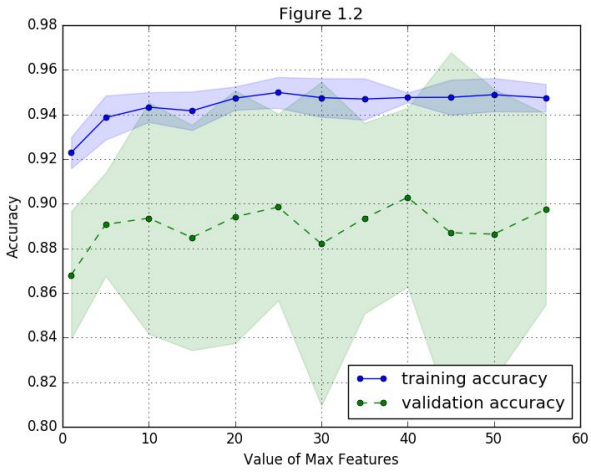
   Training Accuracy: 0.95995, Testing Accuracy: 0.90797

   The next step is to determine how data size can affect the model's performance. Figure 1.1 shows that the model becomes more accurate when the size of our training data increases. The gap between training and validation curves is closer which indicates a good bias and variance tradeoff; however, the overfitting issue still exists, bringing us to next step which is cross-validation.

   We want to perform k-folds cross-validation technique to tune our model. Our training data is divided into five folds with replacement, then the model trains on k-1 folds and validates on the remaining one. The process repeats until each fold is used for validation. Since this dataset has quite a lot of features, we want to see if limiting the

number of features that the classifier uses for training has an effect on its performance by running cross-validation on the **max_features** parameter. The optimal value for this dataset is **40** (Figure 1.2) out of 56.

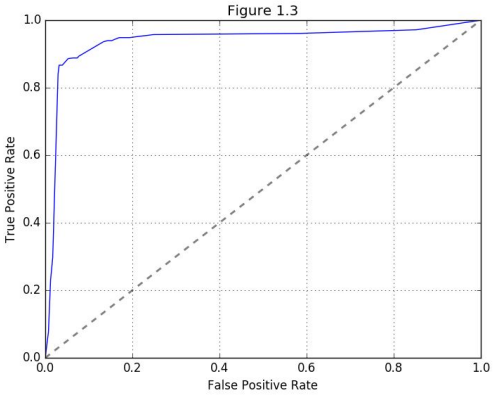




With the tuned parameter, we want to perform the final measurement see the improvement. Here is the result:

CV Testing Accuracy: 0.92246

We will graph ROC curve and calculate the AUC value for our decision tree model for comparison purpose (Figure 1.3). The model has a decent curve as it is very close to the top left corner and an AUC value of **0.93856.**



b. K-Nearest Neighbors Analysis:

We will be using the K nearest neighbors implementation provided by Scikit Learn framework. The analysis will be similar to decision tree which shows performance of the classifier, its learning curve, and tuning the k value using cross-validation. Since KNN is a lazy algorithm that tends to delay learning until the prediction process, we will measure the training time and prediction time (in millisecond) to verify this behavior.

We start out by applying the learning algorithm with k = 5 and obtaining the following result:

Training Time: 4ms, Prediction Time: 36ms, Testing Accuracy: 0.78551

We notice that the performance of the classifier is undesirable and we need to determine the causes using learning curve. Figure 1.4 shows us a consistent pattern with decision tree where the accuracy is increasing as we feed more data. However, the gap between training and validation curves is larger due to an issue with overfitting. We tune the K value for KNN to determine if cross-validation can improve the performance, otherwise, we might need to collect more training data.

The next step we need to do is to experiment with different k values to determine the best fit for our dataset using cross validation with five folds (Figure 1.5). We can see that the classifier has lots of overfitting with k = 1, and the accuracy begins to improve as we increase the number of k. The optimal k value for this dataset appears to be at **3**, and overfitting occurs again as we increase k passing the optimal value.

Figure 1.4

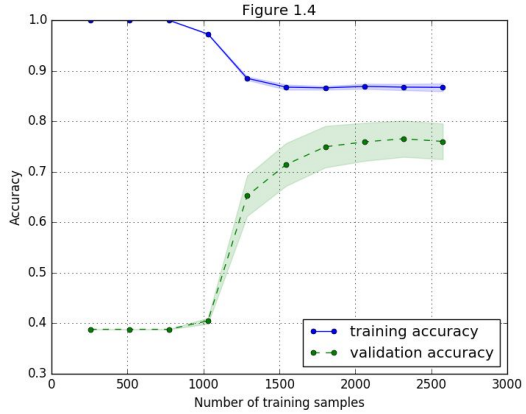
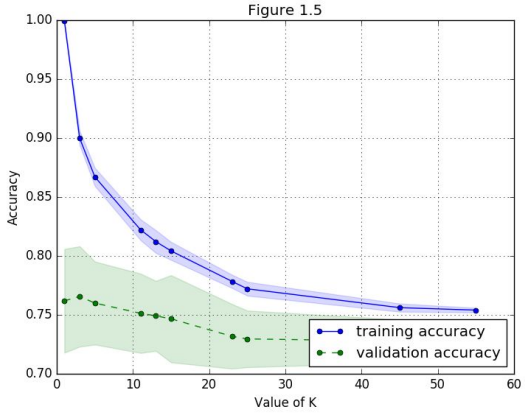


Figure 1.5

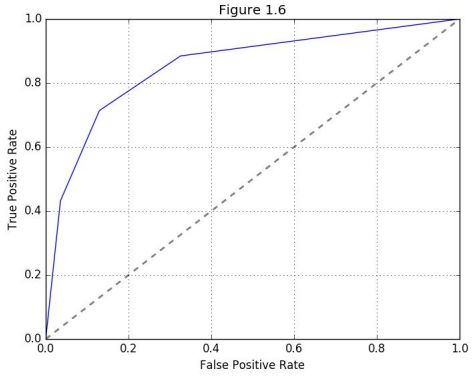


With the new k, we want to run the classifier against our test data to verify the improvement.

Testing Accuracy: 0.80652, Training fit time: 4ms, Prediction time: 29ms

Finally, we want to graph ROC curve and compute the AUC value to compare with other algorithms (Figure 1.6).

Figure 1.6



AUC: 0.85367

From the measurement, we can see that Decision Tree performs better than KNN for the given dataset.

c. **Neural Networks Analysis:**

In this section, we will perform the analysis using Multi-layer perceptron algorithm that trains using Backpropagation provided by Scikit-learn framework. There are quite a few parameters that we can tune for the MLP classifier, but for simplicity, we will focus on the number of iterations until convergence, number of neurons per layer.

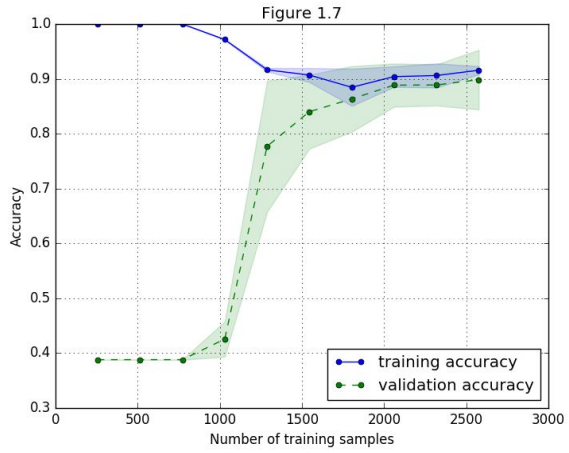
We use the recommended values by the library for the remaining parameters such as **stochastic gradient-based optimizer** for weight optimization and **rectified linear unit function** for activation function, and the tolerance value for weight update of **1e-6** to see the effect of training iteration, and we will fix the number of hidden layers to 2. Running the initial training and testing gives us the following result:

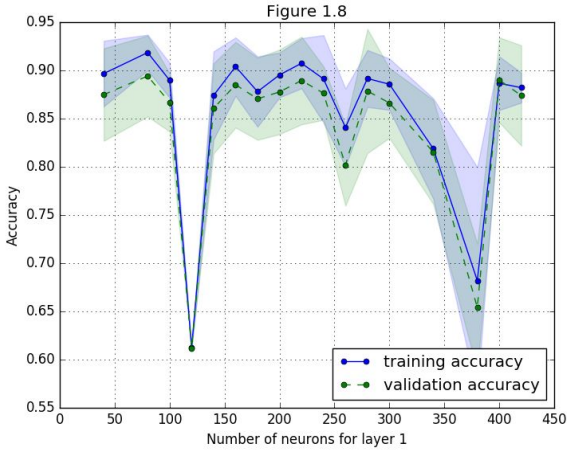
Training fit time: 139ms, Testing predict time: 1ms, Testing Accuracy: 0.83986

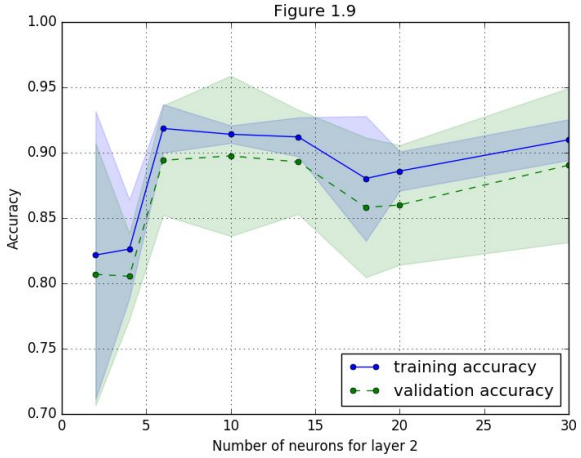
The performance of our MLP classifier is slightly low in comparison to Decision Tree but better than KNN. The learning curve will help us determine how well the model scales with data. The result indicates that our model has good trade-off between bias and variance, and it can converge to a higher accuracy given more data (Figure 1.7). Also, Neural Networks algorithm spends most of its time in the training phase for weights tuning and it takes constant time to predict new examples. This behavior reflects the runtime of roughly 140ms for training and 1ms for predicting new values.

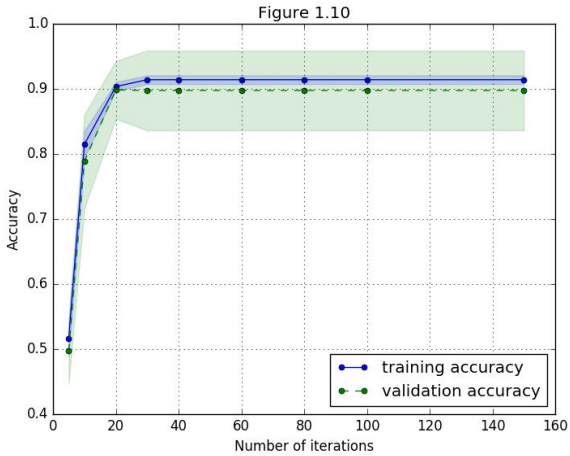
We then run cross validation to optimize the classifier parameters by tuning the number of neurons for each layer. The optimal values for this data set appear to be **80** neurons for the first layer and **10** neurons for the second (Figures 1.8 and 1.9).

The last parameter we are looking at is the number of iteration we want to train the neural networks. From the graph below (Figure 1.10), the classifier accuracy improves as we run more and more iterations until it converges around **30**.





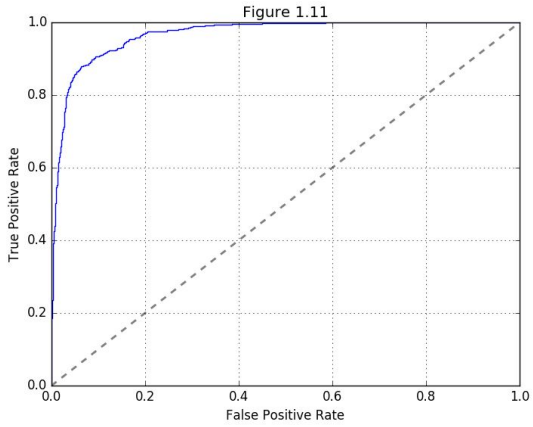




Following through with the above cross validation steps, we obtain the optimal parameters for this dataset. The last thing we need to do is to run our model against the test dataset to verify that its performance is indeed better.

Training fit time: 509ms, Testing predict time: 1ms, Testing Accuracy: 0.9058

We now need to graph the ROC and calculate the AUC values to compare with decision tree and KNN algorithms. As a result, decision tree and Neural networks both perform quite well on this dataset in comparison to KNN. Given the learning curve shape and the AUC value, Neural Networks can potentially outperform Decision Tree given more training data.



AUC: 0.96788

d. Boosting Analysis:

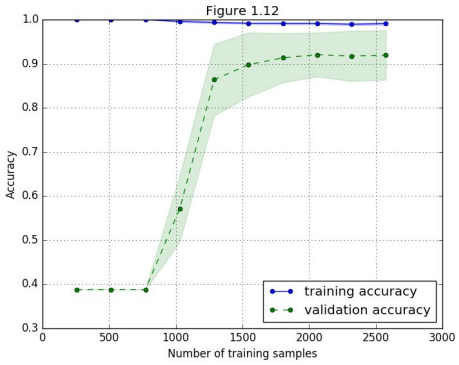
In this section, we will use Adaptive Boosting algorithm provided by Scikit-learn framework with Decision Tree to demonstrate the effectiveness of Boosting. We will select the best model of decision tree we derived from the first experiment to be the base estimator for our boosting algorithm. For parameter tuning, we will look at n_estimators, the max number of estimators at which boosting is terminated, and the learning rate, the rate that each classifier is shrunk by.

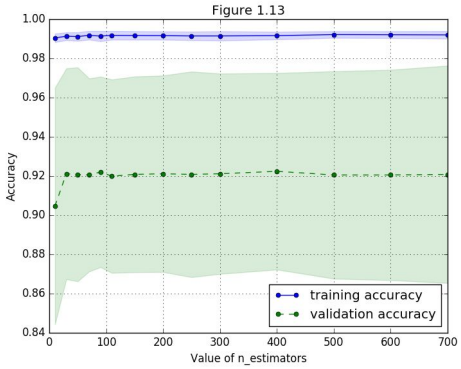
First, we select the default parameters for AdaBoost. Then, we begin training and testing on the dataset and obtain the following result:

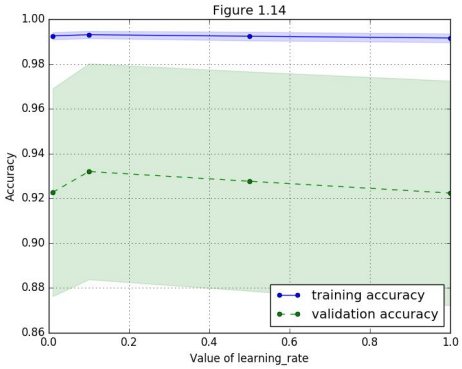
Training time: 1750ms, Training Accuracy: 0.9907, Testing Accuracy: 0.93406

Although the training accuracy is almost perfect, there remains a small difference between training and testing accuracy that indicates overfitting. We need to graph the learning curve and do cross-validation to further tune the model and prevent overfitting. The learning curve (Figure 1.12) shows that the performance of the model is desirable, and the gap between training and validation curves align with the training and testing accuracy results above.

We counter overfitting by tuning n_estimators and learning_rate with cross-validation and obtain the optimal values of **400** for n_estimators and **0.1** for learning_rate (Figures 1.13 and 1.14).



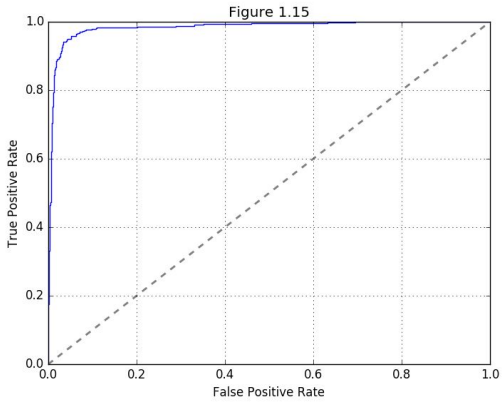




We now perform the final evaluation on the model with the well tuned parameters. Thus, the result increases as expected:

Training time: 17216ms, Testing Accuracy: 0.9529

With ROC curve and the AUC value of **0.98446**, AdaBoost is by far the best model for this dataset.

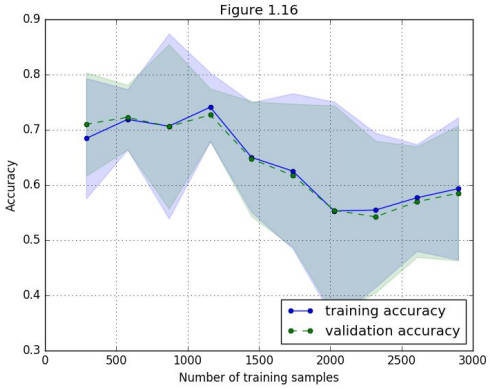


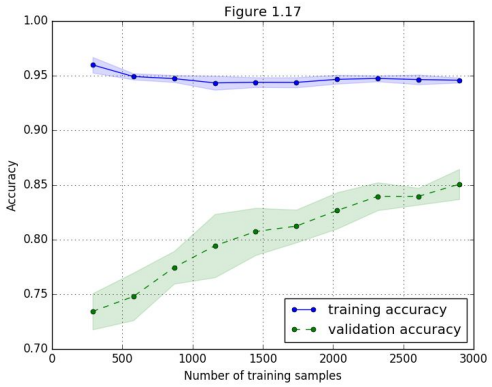
e. Support Vector Machines Analysis:

For this analysis, we analyze the performance of Support Vector Machine on this dataset using the implementation provided by Scikit-learn framework. We will test with two different kernels, **radial basis function (RBF)** and **linear function**, and demonstrate the effect of using different kernel functions on the dataset.

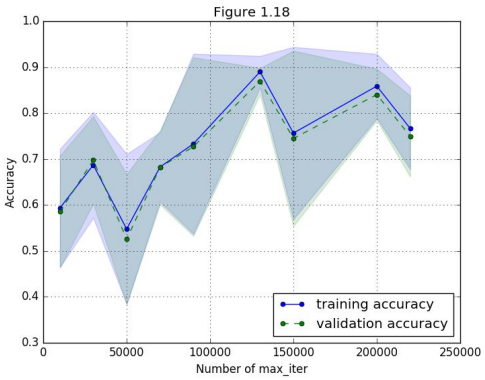
Starting with training and testing both models on the dataset and see the performance, we then graph the learning curve for them. The linear model seems to underfit the training data, while RBF model overfits the training data with very high variance (Figures 1.16 and 1.17).

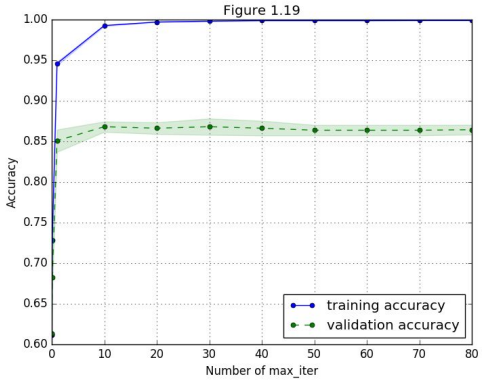
Linear Training time: 1731ms, Linear Testing Accuracy: 0.64058
RBF Training time: 5975ms, RBF Testing Accuracy: 0.82681





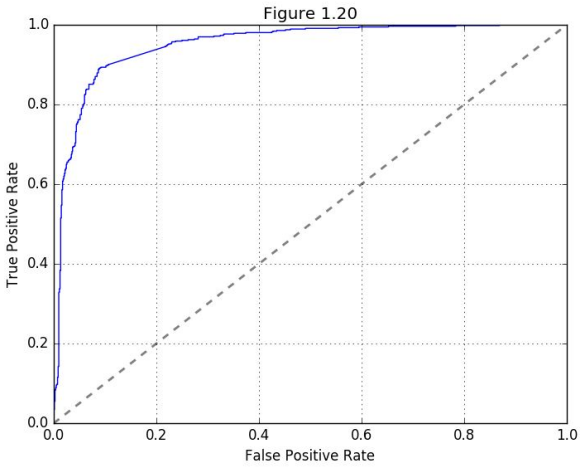
We will tune each model to determine the best fit for this dataset. For linear model, we will use different number of max_iter for termination, and the optimal value appears to be **130,000** (Figure 1.18). For RBF, we use the value of C (penalty parameter for error term) and the optimal value is **30.**

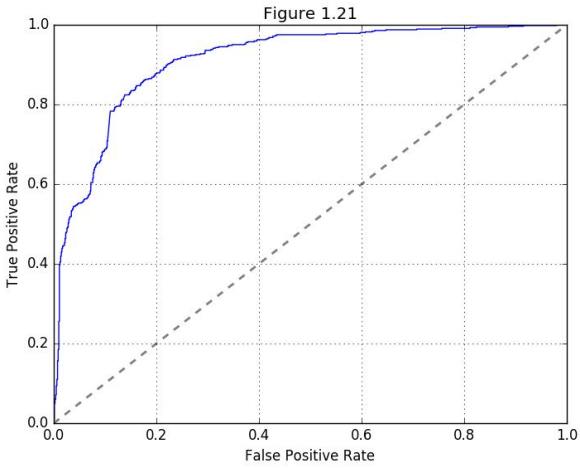




Finally, we want to show the ROC curves and AUC values to compare both models. Given the results, we can see that the linear model has better performance with faster training time.

Linear AUC: 0.95108, RBF AUC: 0.91124





We now run the final result on the linear model with the tuned parameter to see the result

CV Testing Accuracy: 0.89275, Training time: 8785ms

In conclusion, we successfully observe the performance difference on each algorithm given all the experiments. Although they all suffer from overfitting issues, we are able to draw out the pros and cons from each algorithm. **Boosting** comes out in the lead with the highest result with long training time. **Decision Tree** comes in second for performance and a more decent training time even though the learning curve shows us the model will not scale with more data. **Neural Networks** is the next in line given the accurate performance and consistent improvements with more training data. **KNN** is trailing behind due to the "curse of dimensionality" issue in this dataset. Lastly, despite a decent performance, **SVM** cannot scale with large dataset, and it tends to take a longer time to train. Personally, I would prefer using Neural Networks for dataset as it can be tuned with more training data.

2. Magic Gamma Telescope dataset:

This dataset contains 13314 samples and 10 continuous attributes. The features are different measurement values from the Gamma Imaging telescope and the binary classes represent the actual signal or background from the image. It is considered interesting as it has more training samples but comes with a common problem in Machine Learning which is unbalanced dataset (2:1 ratio). Through the experiments, we will learn how each algorithm scale with large dataset and how robust they can handle this problem.

a. Decision Tree Analysis:

Similar to the previous dataset, we will begin with training and testing to see how the algorithm can handle a large dataset:

Training time: 767ms, Training Accuracy: 0.89474, Testing Accuracy: 0.84251

The learning curve is graphed to demonstrate how well the classifier models the target hypothesis. The graph shows that the model has high variance which indicates overfitting (Figure 2.1). Through cross validation, we determine that the optimal value for max_features is **8** (Figure 2.2).

Figure 2.1

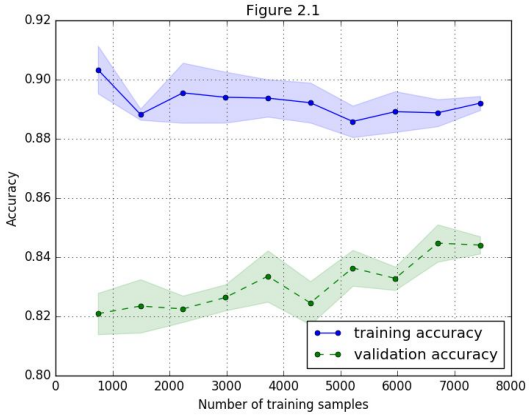
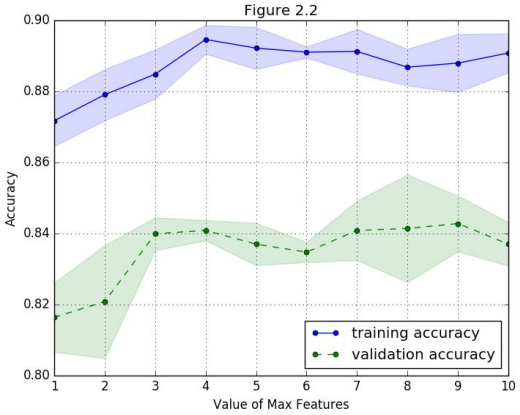


Figure 2.2



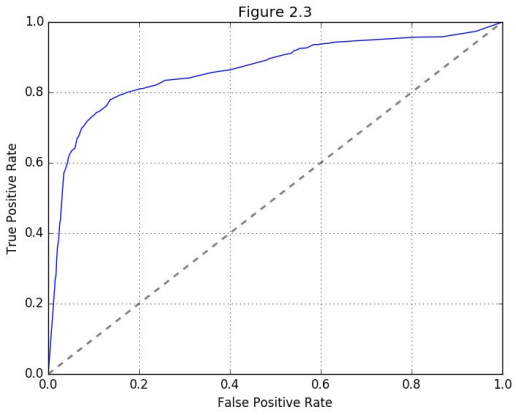
We will now use the tuned parameter and run the final test for performance. The result does not improve much which may suggest that we need to collect more data to balance out the dataset.

Training time: 507ms, Testing Accuracy: 0.84577

The last step in the process will be showing the ROC curve and calculate the AUC value for this model.

AUC: 0.86347

Figure 2.3

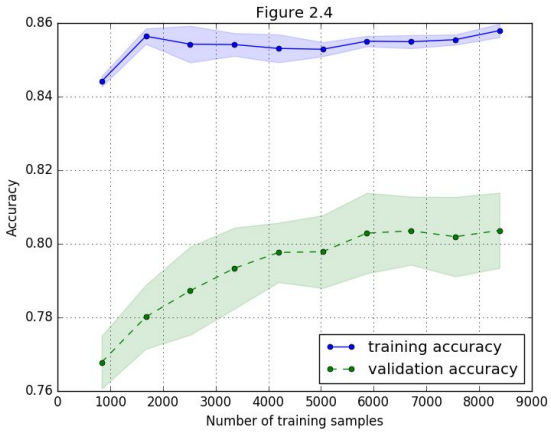


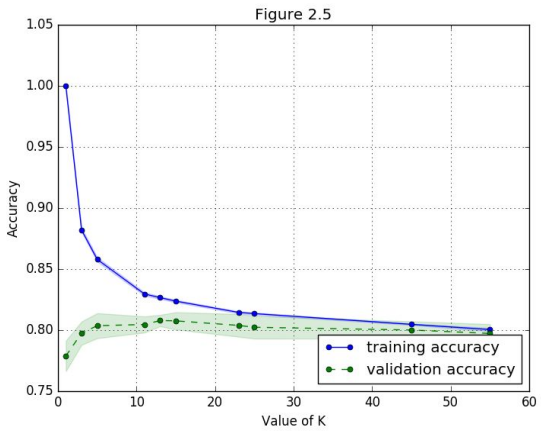
b. K Nearest Neighbor Analysis:

Following the same steps, we perform the initial step for KNN and acquire the following result:

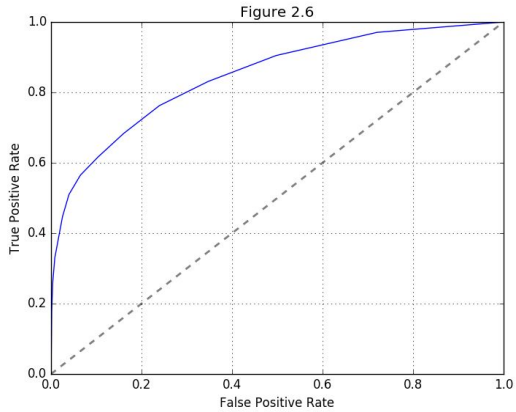
Testing Accuracy: **0.79845**

KNN also suffers from bad performance due to unbalanced dataset. The learning curve tells the same story as our decision tree model, but the score is lower which indicates that KNN is more sensitive to unbalanced dataset. When tuning the value of k to improve the performance, we achieve the optimal value of **15** (Figure 2.5)





For comparison, we graph the ROC curve and calculate the AUC value. The result is very close to our decision tree model with the AUC value of **0.84825**

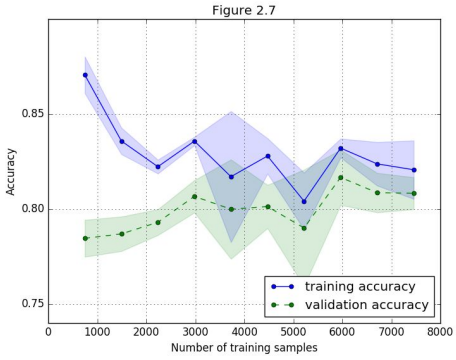


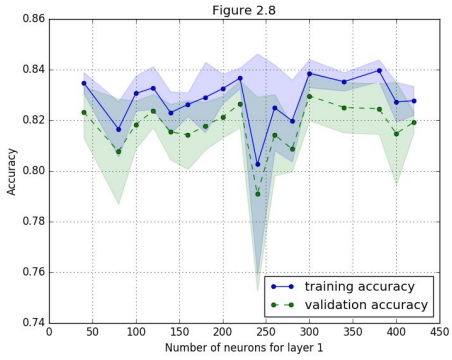
c. Neural Networks Analysis:

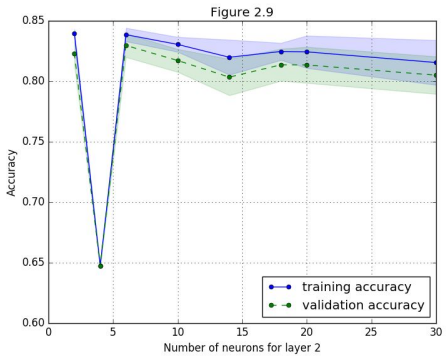
Repeating the steps above for our neural networks classifier, and we have the following results:

Training time: 782ms, Testing Accuracy: 0.81773

We notice that Neural Networks also have poor performance, and it is likely due to the effect of unbalanced dataset. The learning curve (Figure 2.7) shows that the model has low variance but cannot perfectly capture the target hypothesis for this dataset. Through cross-validation, we obtain the optimal values for the parameters **60** iterations, **300** neurons for first layer, and **6** neurons for second layer (figure 2.8 and 2.9).







Even with all the tunings, our Neural networks cannot seem to overcome the unbalanced issue. The final scores are close to Decision Tree:

Testing Accuracy: 0.82098, AUC: 0.87305

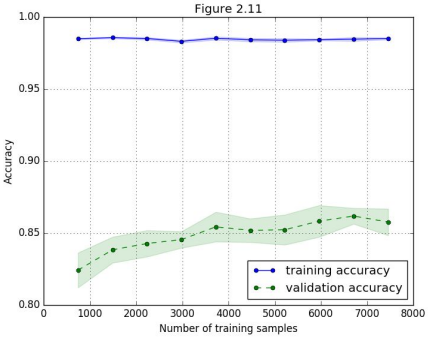
d. Boosting Analysis:

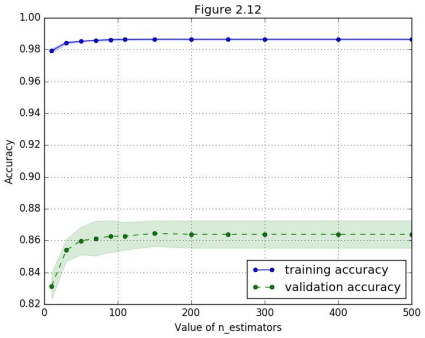
Our boosting model appears to perform better with much less issue with overfitting, although the testing score is still lower compared to the first dataset.

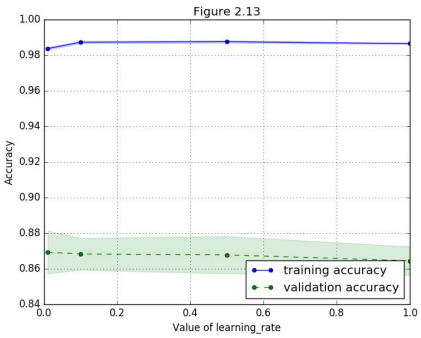
Training time: 31876ms, Testing Accuracy: 0.85228

So far, boosting has the highest score in term of performance. Figure 2.11 shows that the training accuracy is very high with high variances. Using hyperparameters tuning, we are able to improve the performance of boosting with the values of **150** for n_estimators and **0.1** for learning_rate.

Training time: 98327ms, Testing Accuracy: 0.86430







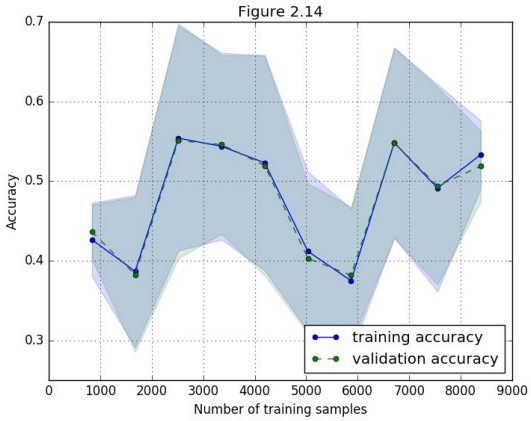
As expected, the AUC value of **0.91226** sets boosting in the lead for this dataset.

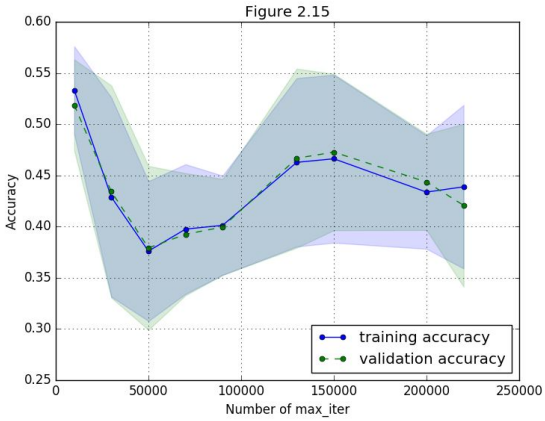
e. Support Vector Machine Analysis:

Lastly, we will perform the analysis with the SVM algorithm. Surprisingly, the linear SVM model completely fails the performance test with a very low score.

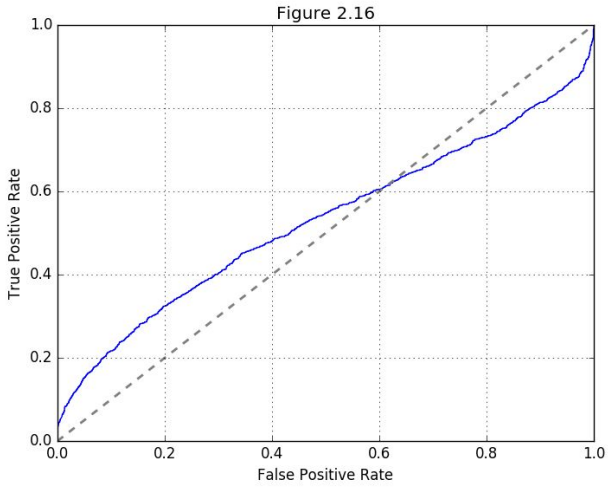
Training time: 4666ms, Testing Accuracy: 0.35729

The linear model in Figure 2.14 has very high bias with training data and fails to capture the target hypothesis. This emphasizes that SVM is very sensitive to unbalanced dataset thus cannot perform well. We try moving on to tuning the number of max_iter in hope of improving the performance. However, there is no value that seems to significantly improve the performance.



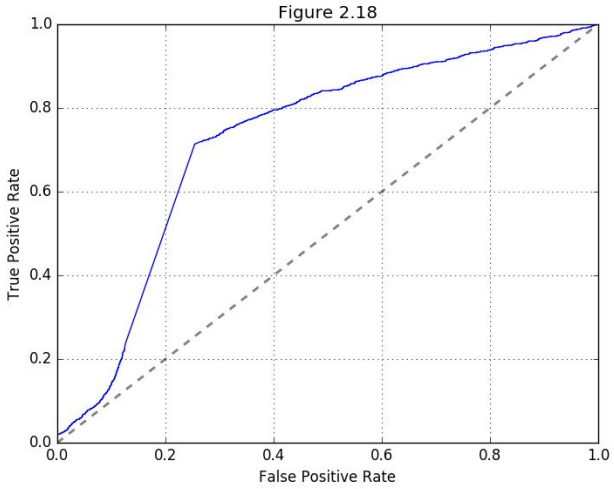


The ROC curve and the AUC value of **0.52896** shows that the linear model does no better than random chance.



We now analyze our SVM model with the **radial basis function kernel** to determine if using a different kernel will have an impact on our dataset. Together with tuning, we can improve the performance of our SVM model, but the result is much worse compared to the other algorithms.

Training fit time: 168439ms, Testing Accuracy: 0.65623, AUC: 0.7261



In conclusion, all algorithms suffer from the unbalanced effect of the dataset. Boosting and Decision Tree are the two algorithms that can perform relatively well, while SVM can only perform slightly better than chance. There are several steps that we can take to counter the problem, which are collecting more training data to balance the dataset, choosing different algorithms (Boosting for this dataset), or changing the performance metric like precision, recall, or Kappa (accuracy normalized by the imbalance of classes). Overall, I would choose Boosting for this dataset despite the long training time.

Summary

From these experiments, we gained a thorough understanding of each algorithm, their performances, accuracy, and runtime tradeoff using datasets that vary in size and number features. We tacked unbalanced dataset problems, and we also learned how cross-validation could help tune and improve the performance. Most importantly, we have devised a process to measure and determine the best algorithm for a given dataset.