CS7641 Assignment 1: Exploring Techniques in Supervised Learning

Boyuan Liu
bryanliu@gatech.edu

1. Introduction

Machine learning has become increasingly popular in the past decade. Many previously unachievable tasks were made possible with machine learning techniques. In this assignment, I explored five supervised learning algorithms: decision tree, gradient boosting, K-nearest neighbors(KNN), support vector machine(SVM), and multi-layer perception(MLP). I designed two experiments and trained these models using the UCI Credit Approval Dataset and the Handwritten Digits Dataset [1]. I explored the hyper parameter space to find the optimal hyper parameters for each of the models and compared their performance in various aspects such as accuracy, scalability, training time, etc. The final models were chosen after carefully balancing these metrics.

2. Experiment 1 - Credit Approval Prediction

The dataset I used in this experiment was the credit approval dataset from the UCI machine learning repository. This dataset has 690 entries in total. It is an interesting dataset because it has 15 feature columns including gender, age, debt, marital status, etc, some have much more influence on the target than the others. The target column contains a boolean value of whether the applicant was approved or not. 55.5% of the applicants were approved and 44.5% of the applicants were denied. Therefore it is a fairly balanced dataset.

2.1 Dataset Cleaning

There were 0.61% of total values missing in this dataset. The missing values were represented with '?'. I used pandas' 'ffill' method to fill categorical missing values from the previous row. The numerical missing values were replaced with the average value of the corresponding column. Since the categorical values in the dataset can not be trained with directly, I used sci-kit learn's OrdinalEncoder to convert them into numeric values. I then normalized the numeric values to a range between 0 and 1 to reduce the impact of large numbers in the training set. To better understand the correlations between the features in this dataset, I plotted a heat map of correlations in Figure 1 below.

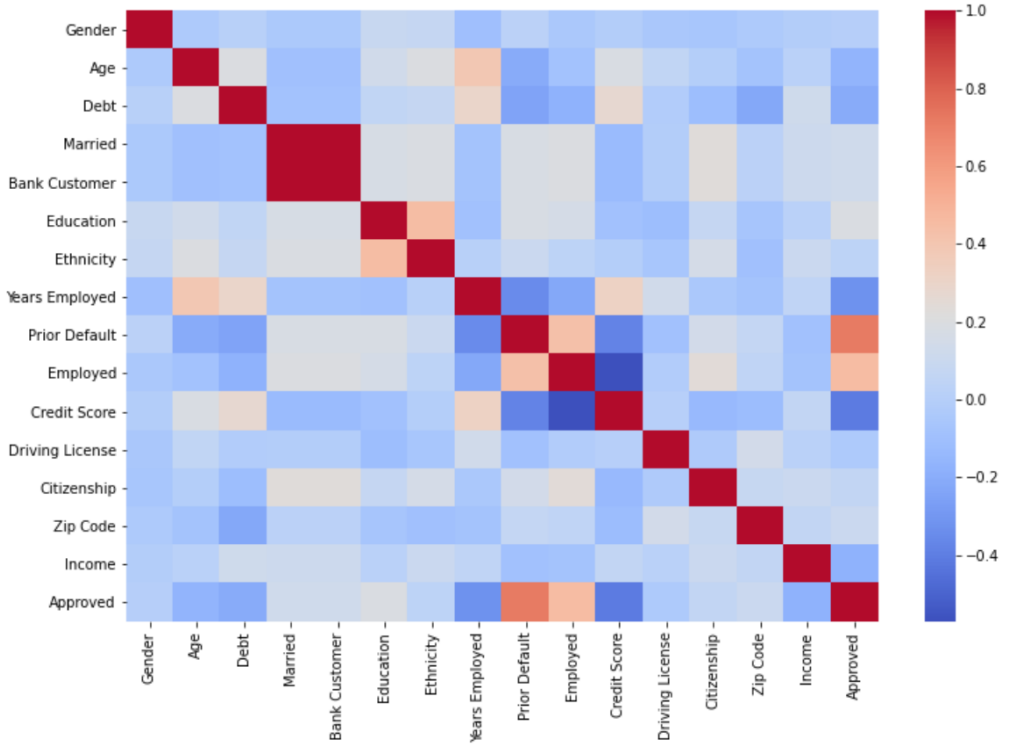


Figure 1 - Correlations between features in the credit approval dataset.

As shown in Figure 1, 'Married', 'Bank Customer', 'Education', 'Citizenship', and 'Zip Code' features have very weak correlations to whether the applicant was approved. Therefore I created a second training set with these features removed. I trained the machine learning models on both of the training sets to see which one gives a better overall result.

2.2. Experiment methods and results

I conducted an experiment for each of the five supervised learning algorithms: decision tree, gradient boosting, K-nearest neighbors(KNN), support vector machine(SVM), and multi-layer perception(MLP). The main purpose of these experiments is to find out how the parameters impact the performance of the model and what is the set of optimal parameters for this dataset. These experiments vary due to the difference in nature of the machine learning models.

2.2.1 Decision Tree

The decision tree algorithm classifies the data points by directing them through a series of branch nodes with 'questions' and returns a classification when they reach a leaf node. The order of 'questions' is selected based on their properties such as Gini impurity,

entropy, or information gain. I used the Gini impurity in this experiment. The features with lower Gini impurities were selected first since they could better classify the data points.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

The decision tree classifier can easily overfit if there is no pruning mechanism or limitation on the maximum tree depth. I used the minimal cost-complexity pruning method [2] to avoid overfitting. The cost-complexity is calculated using the equation $R_\alpha(T) = R(T) + \alpha|T|$, where α is the complexity parameter, $|T|$ is the number of leaf nodes, and $R(T)$ is the total error rate of the leaf nodes. To find the best complexity parameter α for this dataset, I used sci-kit learn's DecisionTreeClassifier and its 'cost_complexity_pruning _path' method. I first calculated the cost complexity pruning path to find all the possible α and how they affect the total impurity, the tree depths, and total number of nodes.

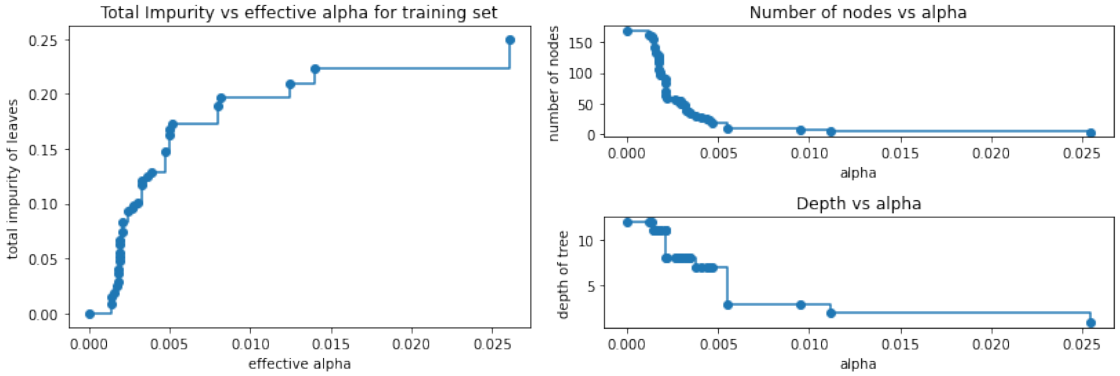


Figure 2 - Total impurity, the tree depths and total number of nodes v.s. complexity parameter α.

As shown in Figure 2, as complexity parameter α increases, the total leaf impurity increases and the classifier complexity decreases. I then trained the classifier with all the possible α and compared their accuracy. The results are shown in Figure 3 below. The best complexity parameter α for this classifier is 0.0055 because it has the highest testing accuracy and does not make the classifier oversimplified.

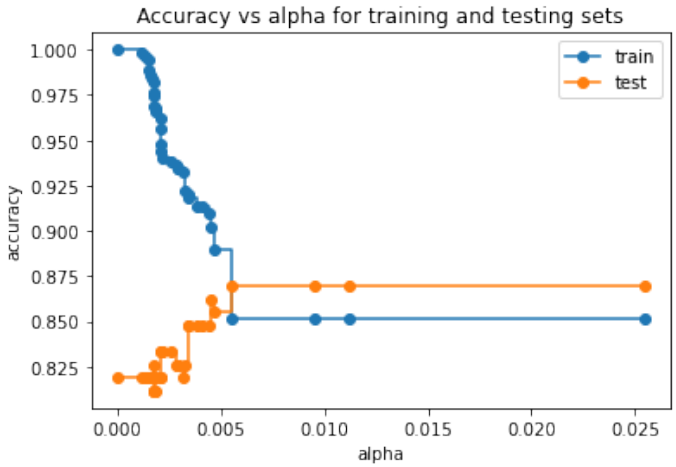
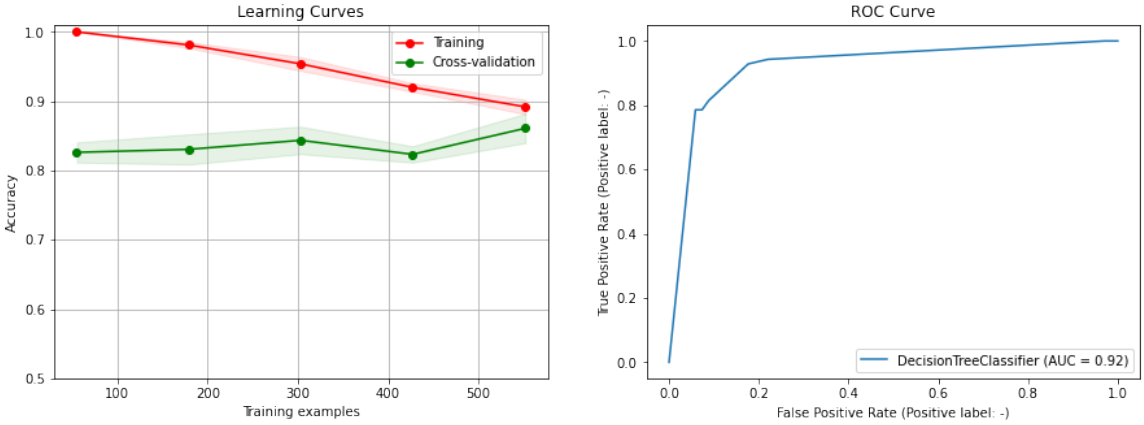


Figure 3 - Accuracy vs complexity parameter α for training and testing sets.

After determining the complexity parameter for pruning, the classifier was trained with five folds cross-validation. The learning curves and classier performance were shown in Figure 4. The classifier had an average training accuracy of 89.2% and an average testing accuracy of 86.1% after training. The classifier had a testing accuracy of 82.0% after training on 100 samples. The testing accuracy slightly increases after training on all samples and the training accuracy decreases as the model became less overfitted. The ROC curve was fairly symmetrical, which means the false positives and false negatives were distributed evenly. In the scalability plot, the training time grew linearly with the number of training samples. This shows the good scalability of the model. The classifier performance curve grew slightly with training time. The general trend aligns well with the learning curve.



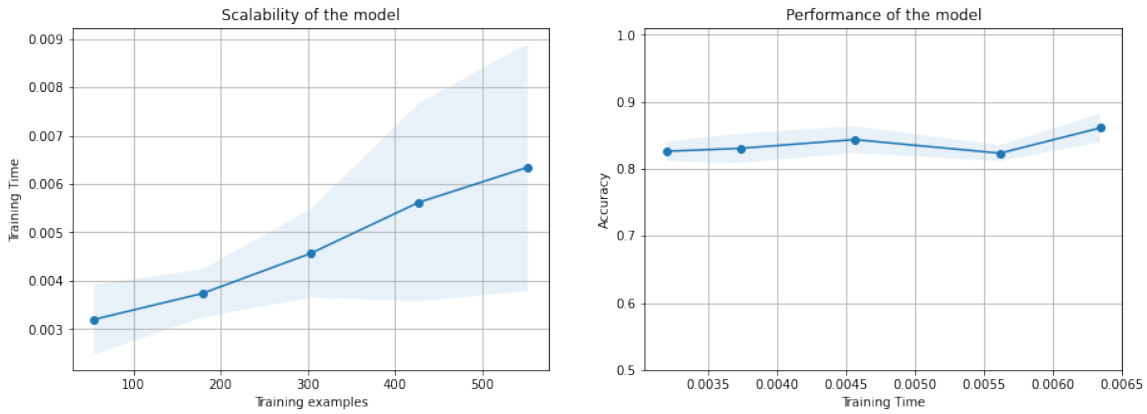


Figure 4 - Learning curves and classier performance of the Decision Tree classifier.

2.2.2 Adaptive Boosting

The Adaptive Boosting(AdaBoost) algorithm combines many weak learners to make classifications. The weak learners are often made of one root node and two leaf nodes, or 'stumps'. Each feature in the training set are given equal weights initially. The first weak learner is the feature that has the lowest Gini impurity, the same as regular decision trees. The following weak learners are chosen based on either the weighted Gini indexes. Each weak learner has a different 'amount of say'. The 'amount of say' is determined by the following equation.

$$\alpha_t = \frac{1}{2} ln \frac{(1 - TotalError)}{TotalError}$$

The total error is the sum of weights associated with the incorrectly classified examples. The feature weights will then be updated based on the amount of say. The incorrectly classified samples will have their weights increased. This will make the following weak learners focus more on the misclassified sample. AdaBoost decision trees can use more aggressive pruning compared to regular decision trees. I used a decision tree classifier as the base model and explored all the complexity parameter α for the AdaBoost classifier. As shown in Figure 5 below, the best complexity parameter α is 0.0029. I also experimented with the learning rate and the total number of estimators AdaBoost classifier. I chose 100 estimators and a 0.8 learning rate as the final number after balancing performance and training time.

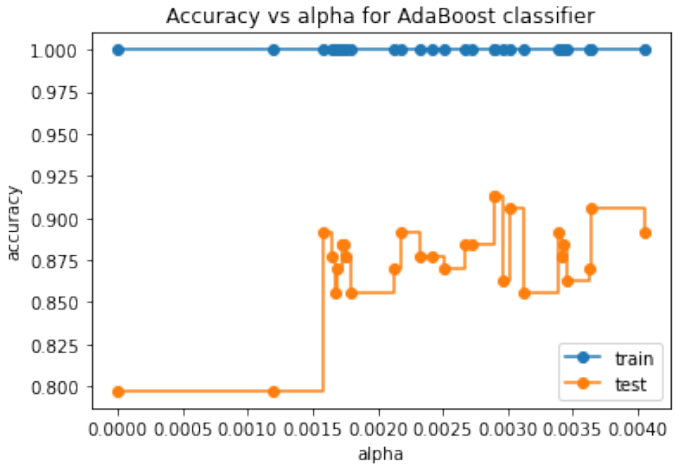
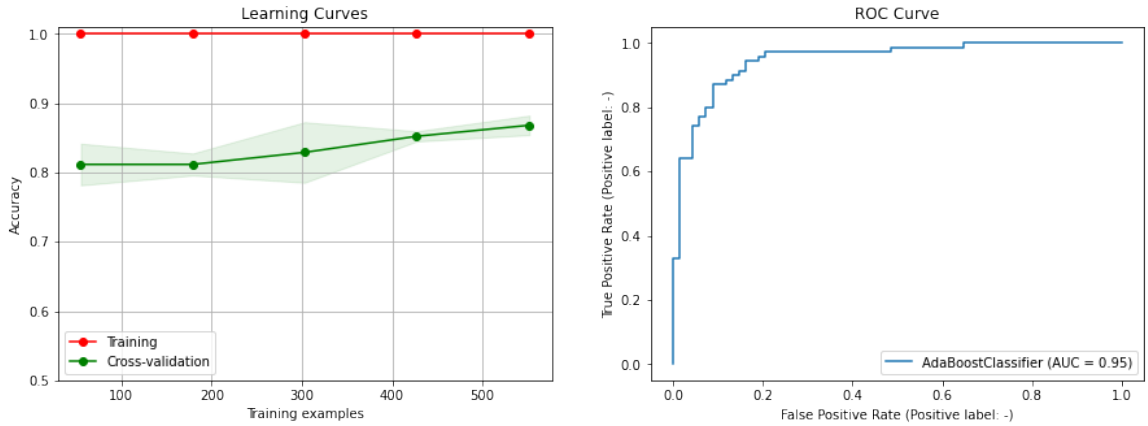


Figure 5 - Accuracy vs complexity parameter α for the AdaBoost classifier.

The results of cross-validation evaluation of the AdaBoost classifier is shown in Figure 6 below. The AdaBoost classifier achieved a testing accuracy of 86.8%, slightly higher than the decision tree classifier's 86.1% accuracy. The training accuracy, however, was a steady 1.0 throughout the training process, which is a clear sign of overfitting. The AdaBoost classifier has an AUC of 95.0%, larger than the decision tree classifier's 92.0%, and the false positives and false negatives were distributed evenly. The AdaBoost classifier took significantly longer to train compared to the decision tree classifier because it contains 100 weak learners. It has clear signs of overfitting and performs only slightly more accurately than the decision tree classifier. Therefore the decision tree classifier is preferred over the AdaBoost classifier.



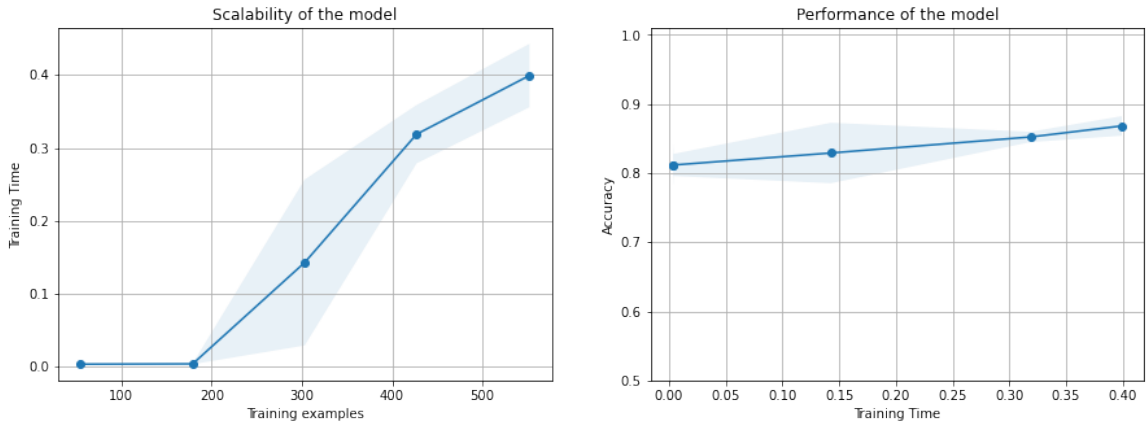


Figure 6 - Learning curves and classier performance of the AdaBoost classifier.

2.2.3 K-Nearest Neighbors

The K-Nearest Neighbors algorithm is a simple but effective method to classify data. It uses the following equation to measure the data point's distance to its surrounding neighbors and classifies the data point to its nearest cluster.

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

I used sci-kit learn's KNeighborsClassifier to implement this classifying algorithm and explored how the number of neighbors affects the classifier's accuracy. As shown in Figure 7, the training accuracy decreases as the number of neighbors increases. The classifier has the highest testing accuracy with 9-nearest neighbors.

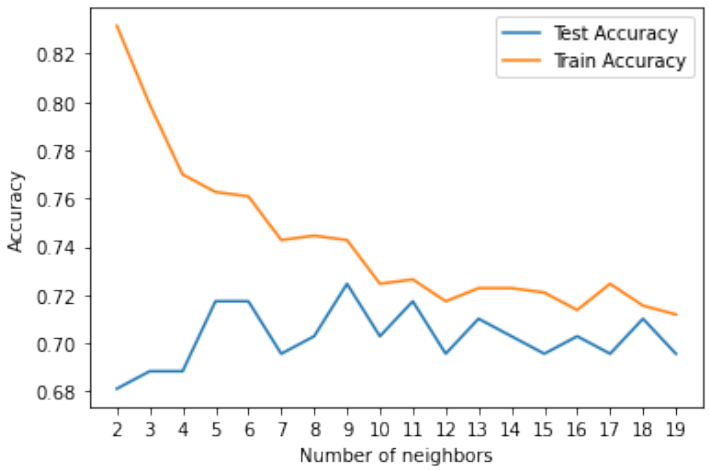


Figure 7 - Accuracy v.s. number of neighbors.

After determining the number of neighbors for the KNN classifier, I examined its performance using the same cross-validation method as the previous classifiers. The KNN classifier did not perform as well as the DT and AdaBoost classifiers. It only had a training accuracy of 74.4% and a testing accuracy of 69.7%. Its area under the curve was only 76.0%. Based on its ROC curve, the KNN classifier's predictions were slightly unbalanced. The biggest advantage of the KNN classifier is that it does not require training. It completed the cross-validation in 0.0036 seconds, much faster than the DT and AdaBoost classifier.

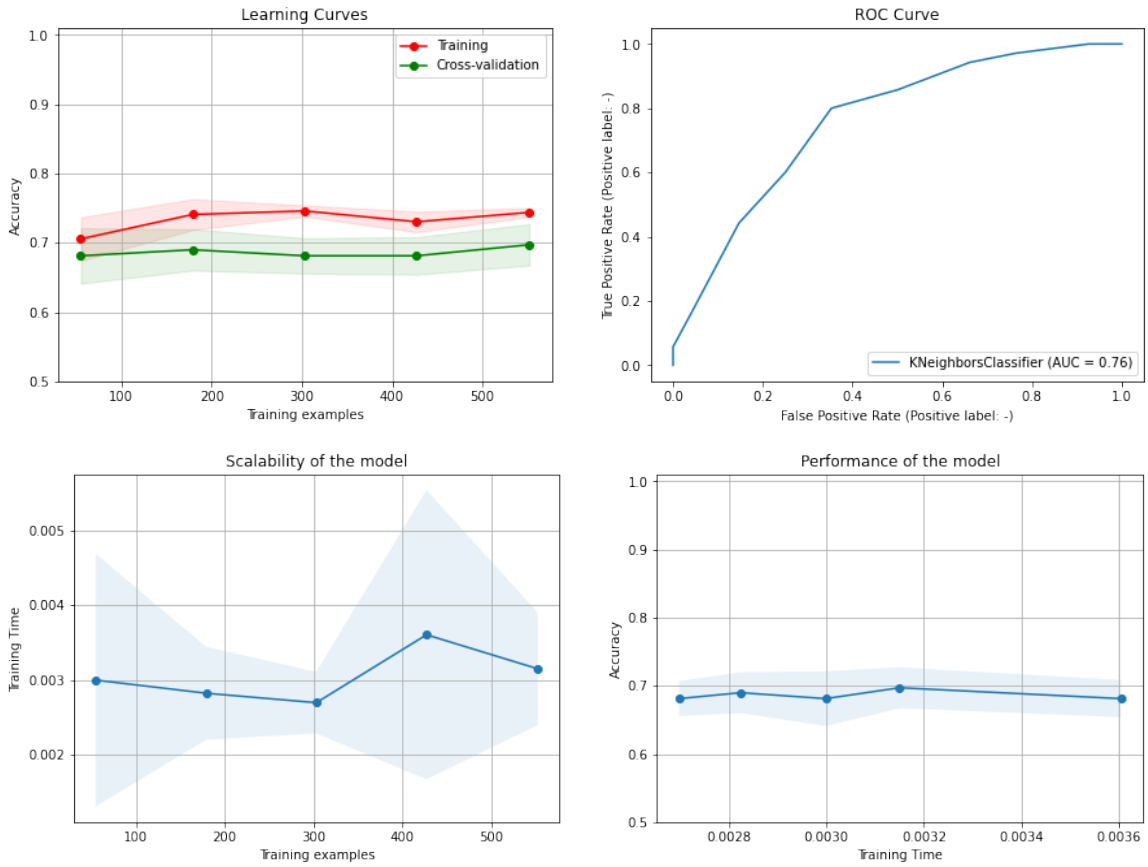


Figure 8 - Learning curves and classier performance of the KNN classifier.

2.2.4 Support Vector Machine

The foundation of a Support Vector Machine(SVM) is a Support Vector Classifier(SVC). The SVC uses cross-validation to find the hyperplane that maximizes the soft margin to separate two data clusters. For the data points that are not separable by a simple hyperplane, SVMs use kernel functions, such as a polynomial kernel, to systematically find SVCs in higher dimensions. Instead of converting the entire dataset to a higher

dimension, The kernel functions use the dot product to calculate which saves computation power. In this experiment, I explored four kernels, linear, polynomial, rbf, and sigmoid, and the different degrees of the polynomial kernel. As shown in Figure 9, the polynomial kernel has the best performance with degree one polynomial. Of all the four kernels experimented, the linear kernel has the highest testing score of 87.0%. Therefore I chose the linear kernel for the SVM classifier.

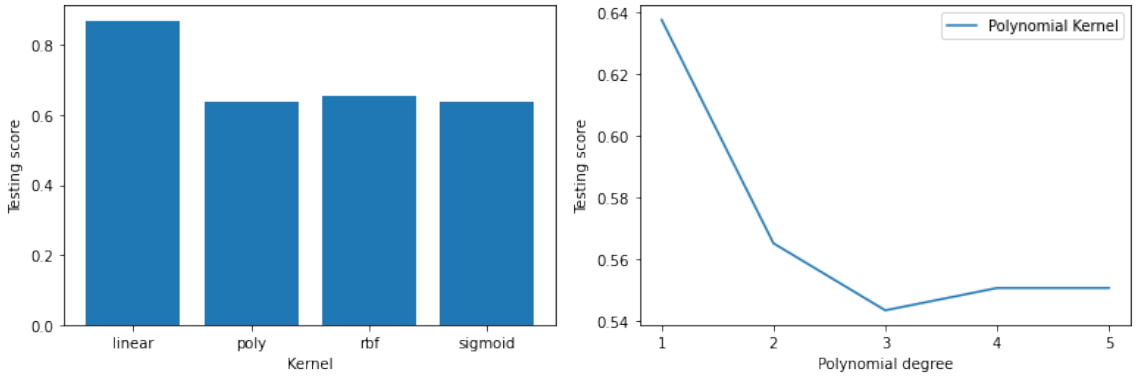
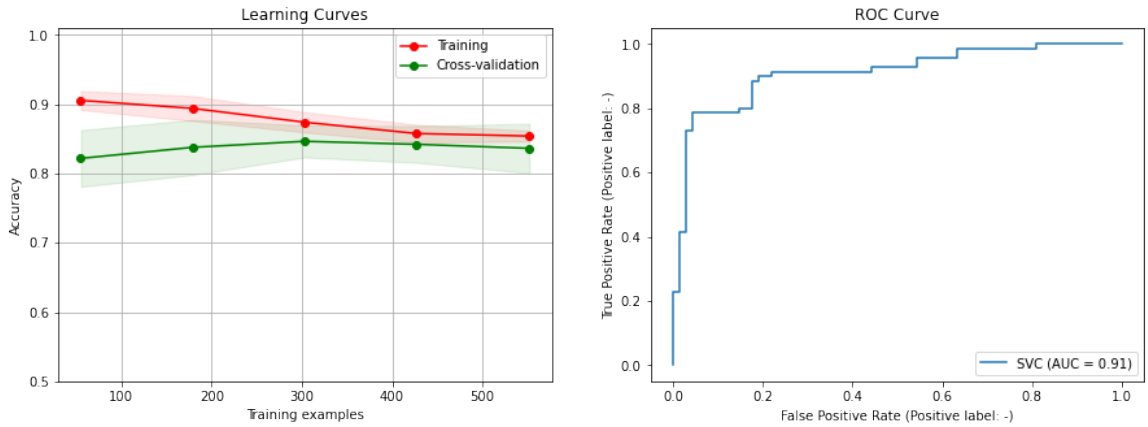


Figure 9 - Testing scores of four SVM kernels.

The cross-validation results in Figure 10 show that the SVM classifier achieved a training accuracy of 85.4% and a testing accuracy of 83.6%. The learning curves show that the training accuracy decreased with the training process, and the testing accuracy slightly increased. Although the increase in sample size did not significantly improve the classifier accuracy, the classifier became less overfit. The SVM classifier did not perform as well as the DT or AdaBoost classifier and took significantly longer to train, over 200 seconds on average. Therefore it is not the ideal classifier for this dataset.



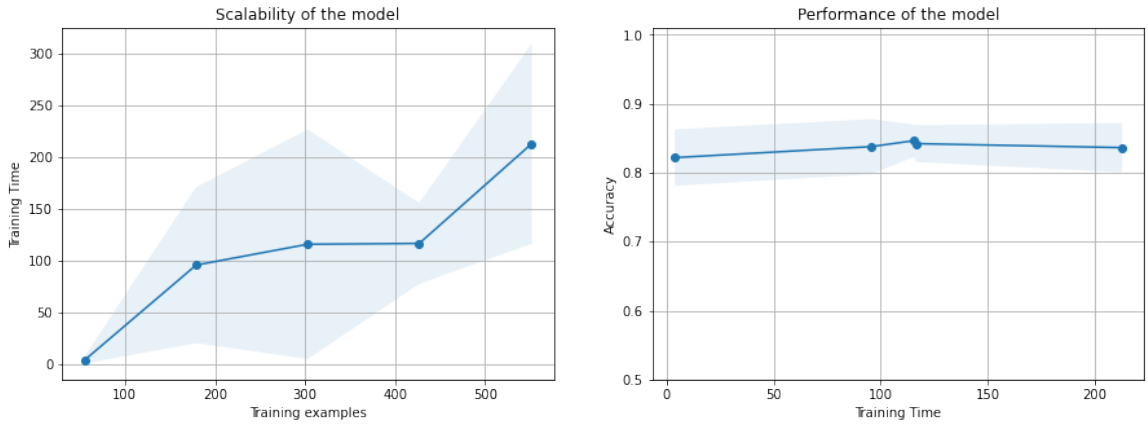


Figure 10 - Learning curves and classier performance of the SVM classifier.

2.2.5 Multi-Layer Perceptron

MLP is a type of neural network that consists of an input layer, multiple hidden layers, and an output layer. Each layer contains multiple neurons or nodes, and each of them has its corresponding weights and activation function. Activation functions provide nonlinearities for the neural networks. Common activation functions include Relu, sigmoid, tanh, etc. The weights and biases in the neural network usually start with random values. The neural network has training data pass through during forward pass and calculate the loss at the end using loss functions such as cross-entropy loss. After the loss is calculated, the neural network then performs backpropagation to update the weights and biases within the layers using an optimizer such as stochastic gradient descent or Adam. MLP classifiers have many hyperparameters that can be adjusted such as layer size, learning rate, optimizer, activation function, etc. Therefore, I used the grid search method to systematically find the optimal parameters for the MLP classifier. Figure 11 shows the loss curve of the top-performing MLP classifiers from the grid search. The optimal classifier converged after 210 iterations and reached a testing accuracy of 87.7%, which is the highest among all the classifiers in this experiment.

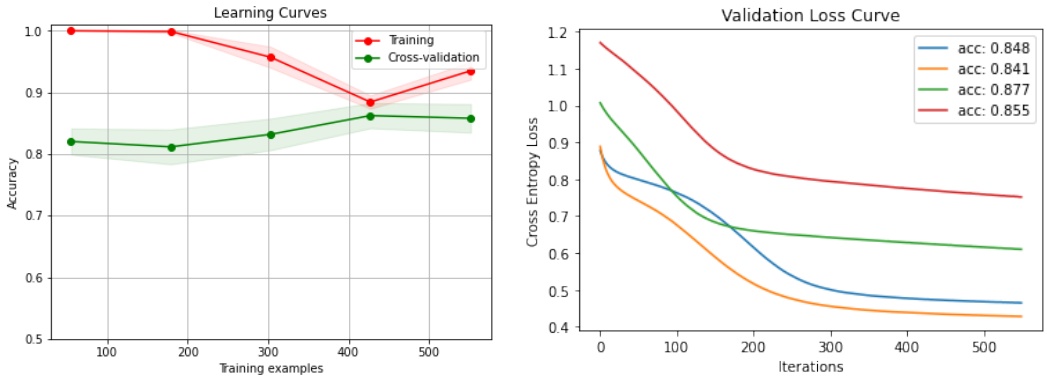
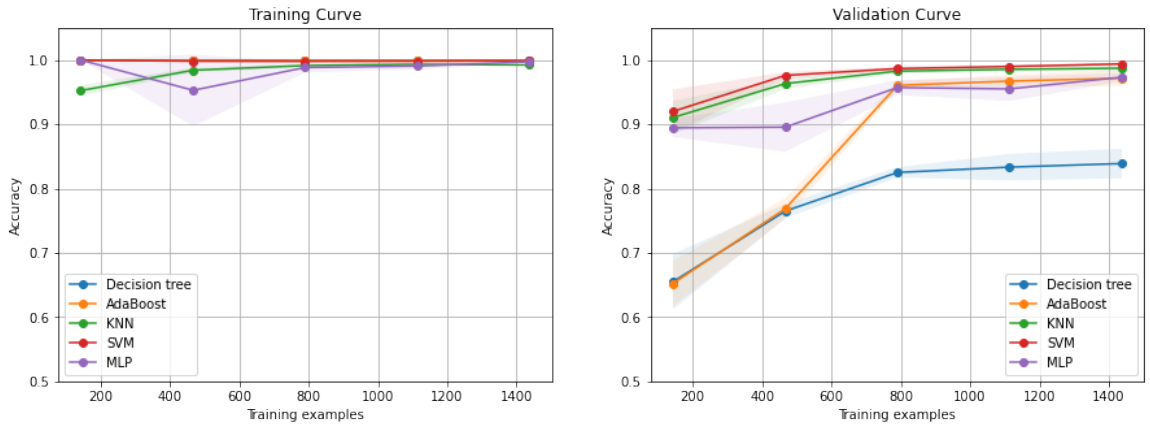


Figure 11 - Cross entropy loss curves of four classifiers from the grid search.

3. Experiment 2 - Recognizing handwritten digits

In this experiment, I decided to test the same classifiers with the completely different Handwritten Digits Dataset from the UCI Machine learning Repository. This is an interesting dataset for this experiment because decision trees or boosting are best for numerical or categorical data. It would be interesting to see how they perform on images. Instead of having 15 categories of personal data like the credit approval dataset, the samples from the handwritten digits dataset are 8×8 pixel greyscale images. The images were flattened into 64x1 arrays before training. I imported the dataset from sci-kit learn's dataset package. Therefore it did not require cleaning. The dataset is well balanced. There are 1797 images of handwritten digits in this dataset. The lowest number of handwritten digits is 8 with 174 samples and the highest number of digits is 3 with 183 samples.

I used the same methods as experiment one to find the optimal classifiers for this dataset. The results of this experiment are also plotted using the same method, shown in Figure 11 below. Among the five classifiers, KNN, AdaBoost, and SVM all achieved a very high testing accuracy of over 97%. The decision tree classifier had a training accuracy of 100% and the lowest accuracy of 84.9%. Therefore it was clearly overfitted. The high accuracy of KNN and AdaBoost classifiers was out of my expectation. I used the same grid search method to find the best hyperparameters for the MLP classifier. The Loss curves of the top performing MLP classifiers are shown in Figure 11. The best performing MLP classifier achieved a testing accuracy of 91.7%. In terms of scalability and performance, the AdaBoost classifier took significantly longer to train than the other classifiers. This is because it contained 100 weak learners in the classifier. The KNN classifier was the fastest among them because it did not require training due to its algorithm. The SVM classifier had the highest accuracy of 99.4% and showed good scalability and training time. Therefore it is the ideal classifier for this dataset.



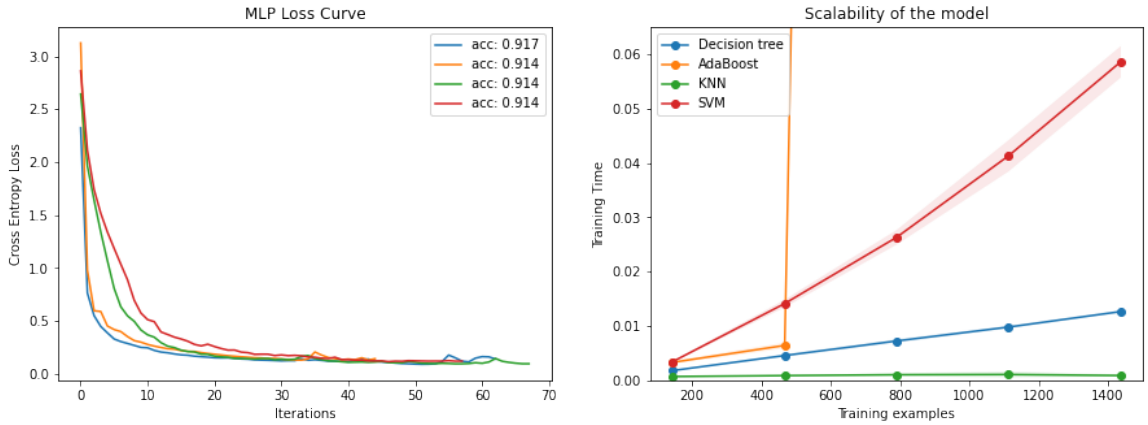


Figure 11 - Classifier accuracy and performance from the handwritten digits experiment.

4. Conclusion

In this project, I explored five classifiers, DT, AdaBoost, KNN, SVM, and MLP, in two experiments. In the credit approval prediction experiment, the MLP classifier reached the highest testing accuracy of 87.7%, while the KNN classifier struggled the most with only a 69.7% accuracy. In the handwritten digits recognition experiment, AdaBoost(97.2%), KNN(98.7%), SVM(99.4%), and MLP(97.3%) all reached over 97% testing accuracy, while the DT classifier struggled with an overfitting problem and had the lowest 83.9% accuracy. The final models are the MLP classifier for credit approval and the SVM classifier for handwritten digits recognition after balancing accuracy, training time, and overfitting. The credit approval prediction experiment still has room for improvement. The current dataset only has 690 entries. Collecting more data would help reduce overfitting and also allow deeper and larger neural networks. The handwritten digits recognition experiment could be further extended to recognize handwritten letters. It would be interesting to explore how the classifiers perform on the more complex handwritten letters.

Reference

[1] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.