Supervised Learning

Introduction

For my two datasets, I have chosen one provided by the popular python machine learning library scikit-learn and another downloaded from the UCI Machine Learning Repository. The first dataset contains a collection of handwritten digits between zero and ten where the goal will be to correctly identify which digit has been written. The second dataset used contains a description of a variety of characteristics of wine and the wine's associated quality. The wine quality has been put into two bins, good and bad, in order to simplify the problem at hand to one of classification.

Five different learning algorithms were applied to these two datasets, which were Decision Trees, Neural Networks, Boosting, Support Vector Machines, and *k*-Nearest Neighbors. The following analysis delves into methodology used for each algorithm in order to try to maximize the performance without overfitting to the available training data.

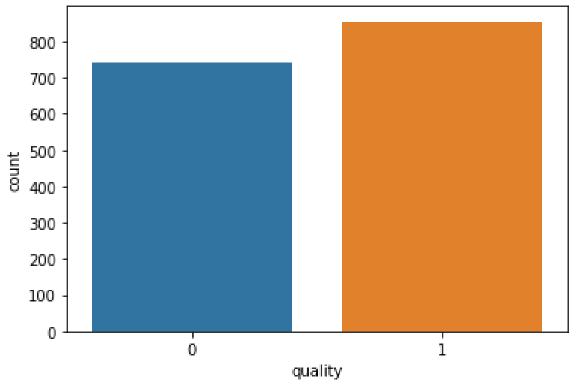


Figure 1: Distribution of wine quality, 0 being bad and 1 good

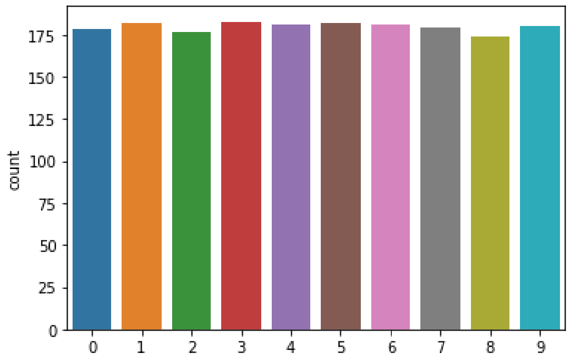


Figure 2: Distribution of handwritten numbers in digits dataset

Wine Quality

Decision Trees

For the wine quality dataset, a grid search was first performed on the *max_depth* hyperparameter for which the optimal result returned was 7. A validation curve was then created by varying the max depth of the decision tree and seeing which value performed the best on the validation set. The optimal value returned in this case was 18, but as seen by the curve in Figure 3. a max depth of 7 produces almost the exact same accuracy but will not overfit the training data as much. Although the training score continues to go up with more tree depth, this is to be expected as the tree can traverse to encompass all training examples, but in a way that it would be overfitting.

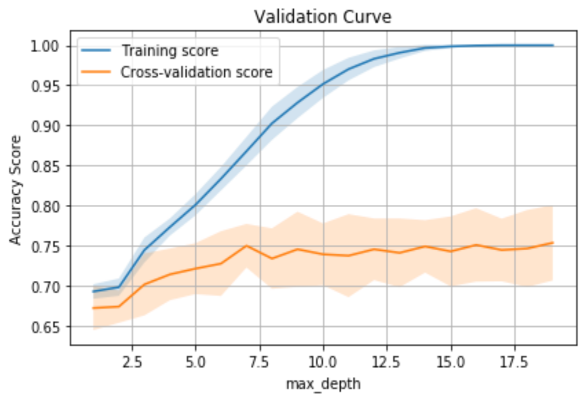


Figure 3: Validation curve for wine quality confirming an optimal max depth of 7

In addition to tuning the max depth of the tree, post pruning was also attempted by using cost complexity pruning and attempting the choose the right *ccp_alpha* value for the dataset. By choosing a larger alpha value, a decision tree can be created that is not quite so prone to overfitting. An alpha value of 0.015 was chosen, which showed the best results as seen in Figure 4. As can be seen from the learning curve in Figure 6, this model has much lower variance than the model created from just varying the max depth of the tree which can be seen in Figure 5.

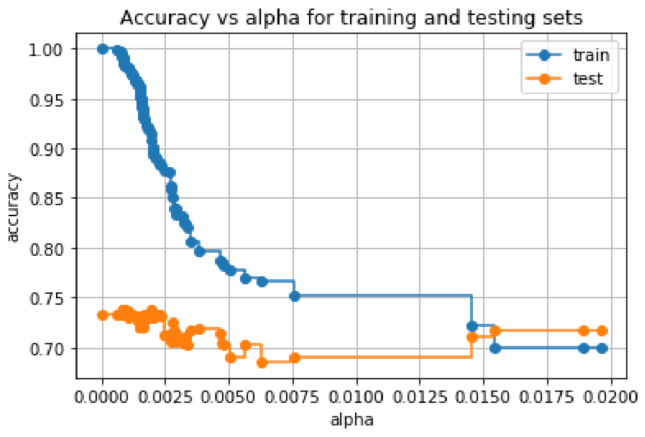


Figure 4: Cost complexity pruning of the wine quality set

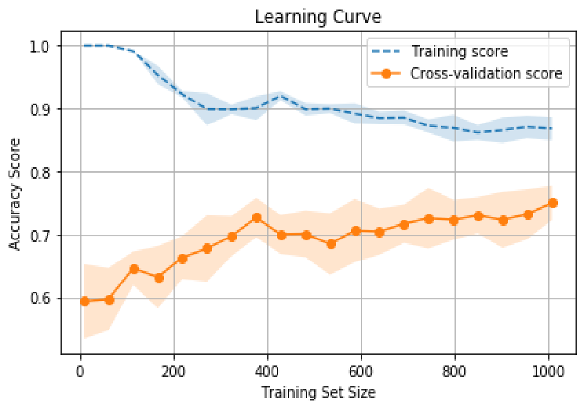


Figure 5: Learning curve for a decision tree just setting max depth

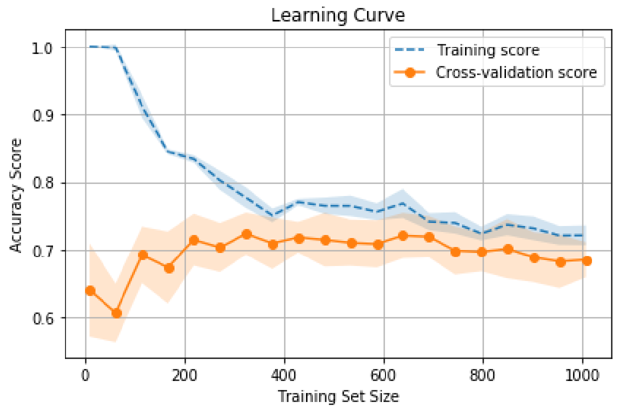


Figure 6: Learning curve for a decision tree setting ccp_alpha

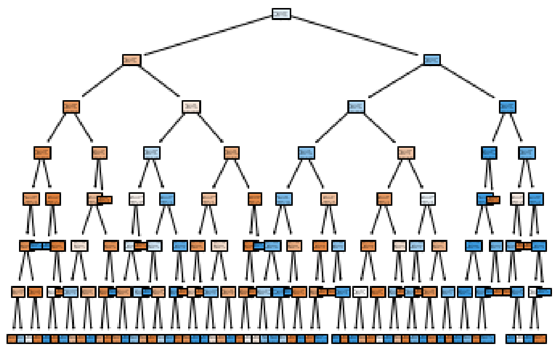


Figure 7: Decision Tree diagram after setting max depth to 7

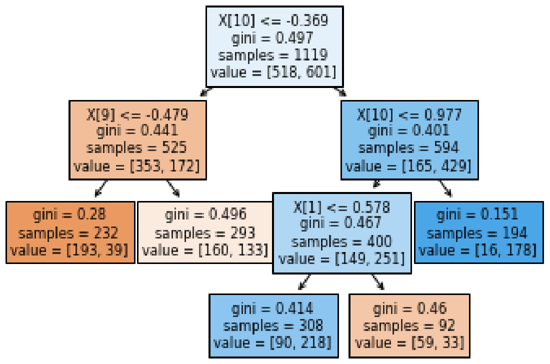


Figure 8: Decision Tree diagram after cost complexity pruning

The result of the cost complexity pruning can clearly be seen in Figures 7 and 8 above, where the model is simplified down to a depth of 3. This greatly minimizes any possible overfitting, while maintaining very similar accuracy on the test set.

Neural Networks

For neural networks, a grid search was first done on various parameters, including trying different learning rates, solvers, and different size hidden layers. The time to train this algorithm increased from just a second or two up to 28 seconds. The best parameters found during the grid search involved using a constant grid search with a L-BFGS solver. Setting these two inputs, different size hidden layers were tested, and the following learning curve and confusion matrix can be seen from the most successful choice.

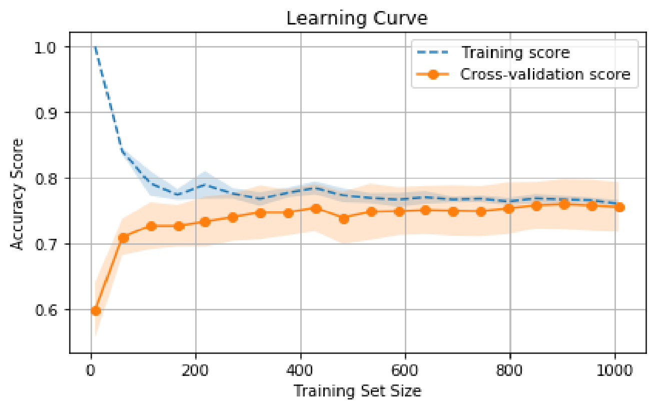


Figure 9: Learning curve for Neural Network on wine quality dataset

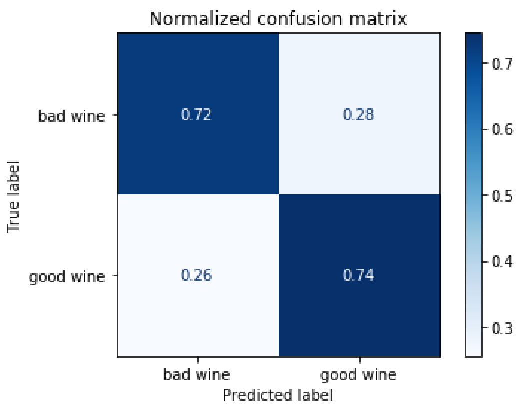


Figure 10: Confusion Matrix for Neural Network

As can be seen from the learning curve, this model has an even lower variance than the Decision Tree algorithm with better performance. If given more time, more combinations of hidden layers could be tested to see if the test set accuracy could be increased.

Boosting

Boosting was used next where the weak learner used was a Decision Tree. After deciding to use Decision Trees, different parameters were used for max depth while doing a grid search in order to try to determine what the optimal number of estimators would be. It was determined that a max depth of 1 was optimal, but the grid search and validation curve returned differing numbers on the optimal number of estimators. After trying out many different parameters it turns out that the best number of estimators returned by grid search, 37, performed the best.

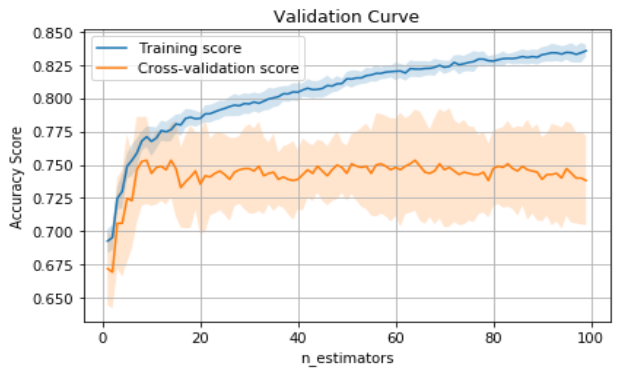


Figure 11: Validation curve for boosting when varying number of estimators

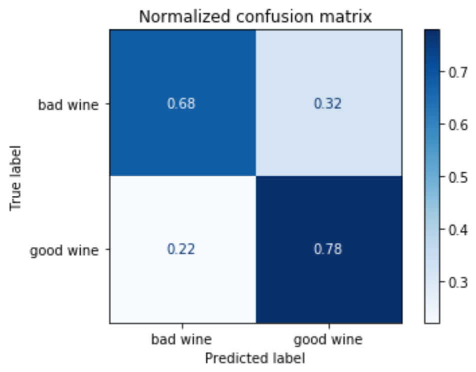


Figure 12: Confusion matrix for the boosting algorithm with 37 estimators

The learning curve from boosting seems to show a large amount of bias and variance when comparing to the previous algorithms. In this case, it seems that the algorithm would've had aided from a larger dataset, as it can be seen from the learning curve that the cross-validation score was continuing to improve with more and more examples to learn from.

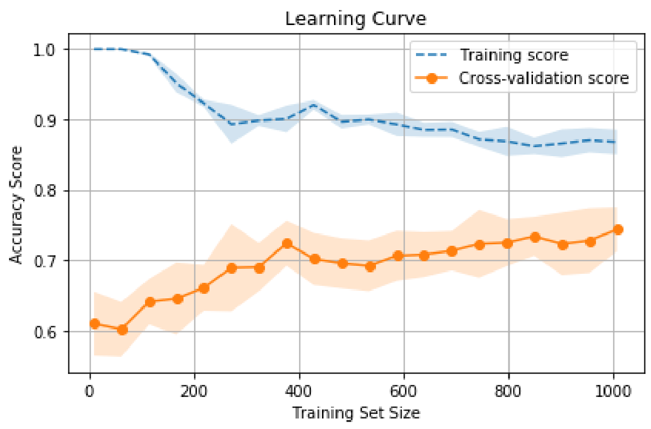


Figure 13: Boosting learning curve for wine quality

Support Vector Machines

For Support Vector Machines (SVM), we again start with an exhaustive grid search in order to try and find the optimal parameters. The hyperparameters in play for this algorithm includes C, gamma, and the type of kernel used. For this dataset, we used two different kernels, *linear* and *rbf*. This algorithm by far took the longest, using up to 1000 seconds in order to perform the grid search for optimal hyperparameters. Once done, the grid search returned that the best parameters were 16 for C, 0.1 for gamma, and a *rbf* kernel. From there, a validation curve was run on C and gamma, varying one parameter while keeping the other constant, as seen below.

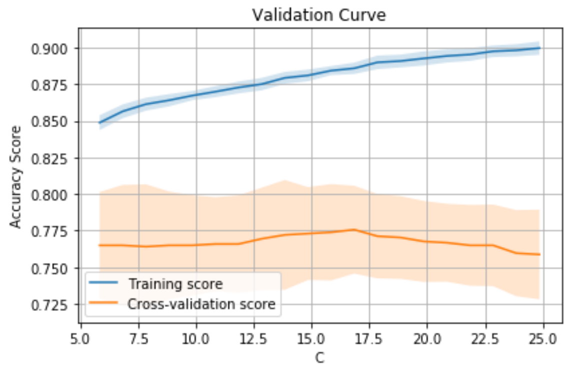


Figure 14: Validation curve for SVM varying C

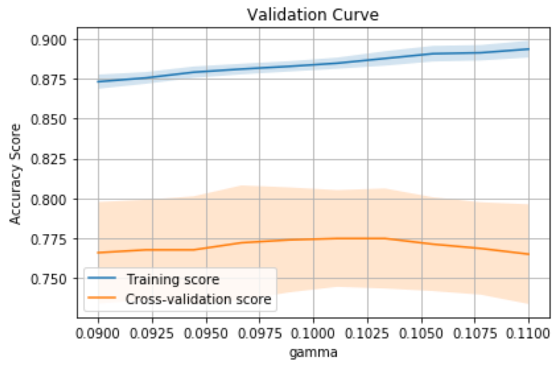


Figure 15: Validation curve for SVM varying gamma

As can be seen by the figures above, the cross validation score does not seem to change much, but it turns out a little more optimal value of C is around 17, while it can be seen for gamma that 0.1 is indeed the optimal value.

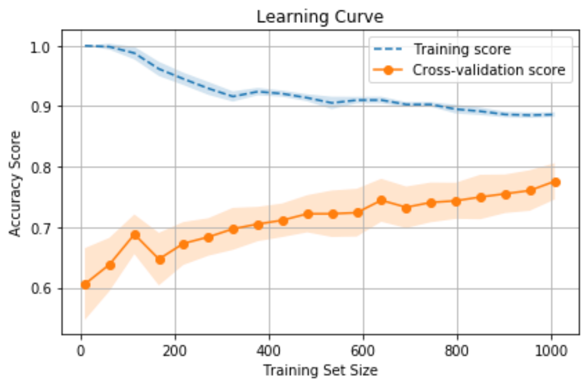


Figure 16: Learning Curve for SVM for wine quality dataset

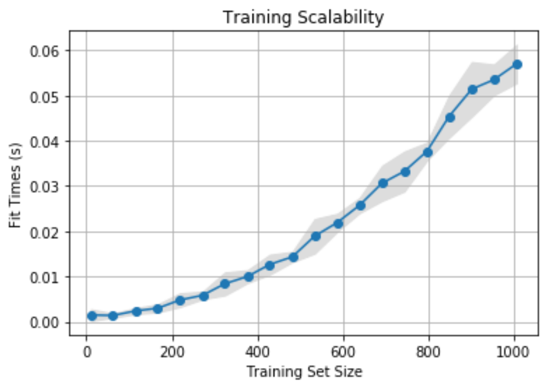


Figure 17: Training Scalability for SVM

As can be seen from the learning curve, it looks like the results may improve if there were more data available for the algorithm to keep improving on and become more complex. With the data available, there seems to be a high variance in the results like the boosting algorithm. As can be seen from Figure 17, the amount of time needed to fit the model increases exponentially with time, which would mean that any added data to learn from would come at a higher cost than what has already been trained on.

k-Nearest Neighbors

For the *k*-Nearest Neighbors algorithm, a grid search was run in order to determine what the optimal number of neighbors would be. The grid search returned that just using 1 neighbor would be optimal, although this may lead to solution with low bias and high variance. In order to see what the effect of using a different number of neighbors, a validation curve was created by varying the performance using different numbers for *k*.

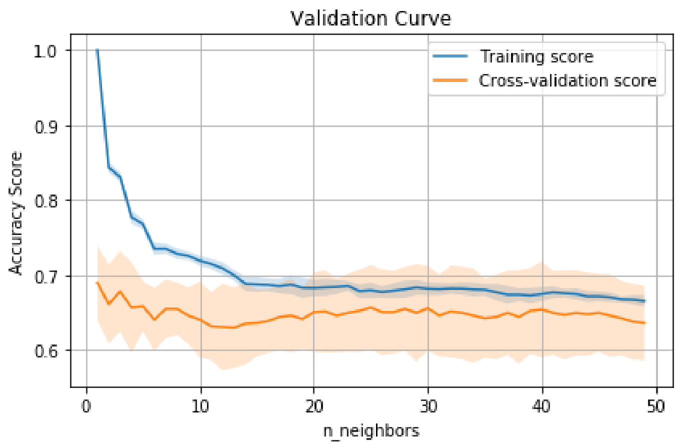


Figure 18: Validation curve for wine quality dataset varying number of nearest neighbors for KNN

As can be seen from Figure 18, the highest validation score does indeed come from only having 1 neighbor, but the results seem to be fairly equal for higher numbers of *k*, which would aid in balancing out the variance and bias in order to have a fairly generalized model.

Results

Table 1: Performance of all algorithms on the wine quality dataset

	Null Accuracy	Decision Tree	Neural Network	Boosting	SVM	*k*NN
Time to train (s)	0	2.37	28.63	2.73	958.43	0.63
True Negative (%)	0	77	72	68	70	65
True Positive (%)	53.47	66	74	78	78	69

As can be seen from Table 1, every algorithm performed better than the null accuracy for both labeling true negative examples as well as true positive. Although the neural network did take an order of magnitude longer to train than algorithms such as the decision trees or boosting, the results are a good mix of having a high precision on both the positive and negative test cases available. In order to try to speed up the neural network if more data was available, a larger learning rate could be selected going forward to try and minimize the downside of having a larger dataset available to train on. The SVM algorithm as performed very well, with the highest accuracy among all algorithms, and also the highest training time by far.

Digits

Decision Trees

For the digits database, grid search was again used to determine the optimal max depth of the decision tree and returned a depth of 10. A validation curve was then run to confirm this and showed that indeed after about 10 levels the cross-validation score seems to level out, as seen below. For this dataset, by performing cost complexity pruning, a model was chosen that ended up being overly simplistic and did not have as good of results as simply setting the max depth of the tree.

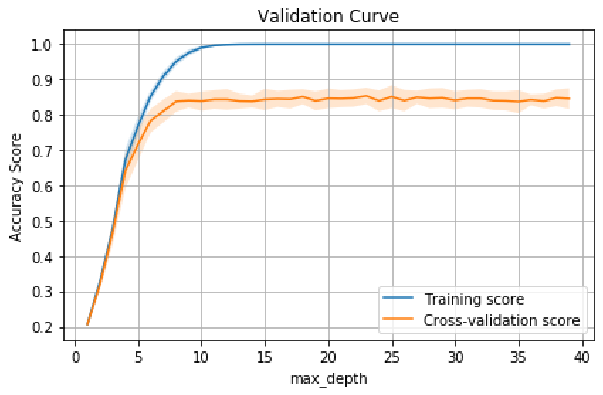


Figure 19: Validation curve for decision tree algorithm on digits dataset

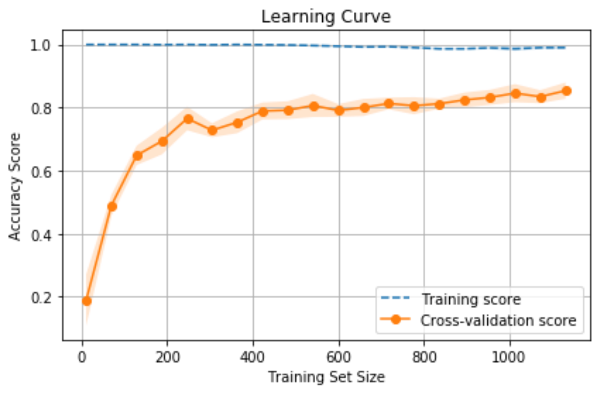


Figure 20: Learning curve for decision tree on digits dataset

Neural Networks

For the neural network algorithm, the grid search run returned that a constant learning rate with the L-BFGS solver would produce the best results. The learning curve produced from using these hyperparameters below shows a model that seems to be performing very well with low variance. Somehow the training set is still able to do very well with a very small training set size, although it takes quite a bit larger training set size in order to get the cross-validation score where it needs to be.

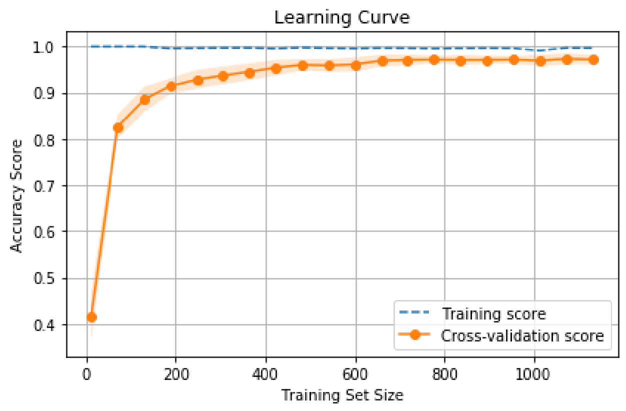


Figure 21: Learning curve for neural network on digits dataset

The activation function for the hidden layer used in this case was a rectified linear unit (ReLU). Different combinations of hidden layers were tested, but 2 layers were settled on. If given more time, it would be interesting to come up with a better way to nail down the optimal number of layers and nodes used. It would certainly increase the amount of training time necessary, but the performance might go up with a better combination.

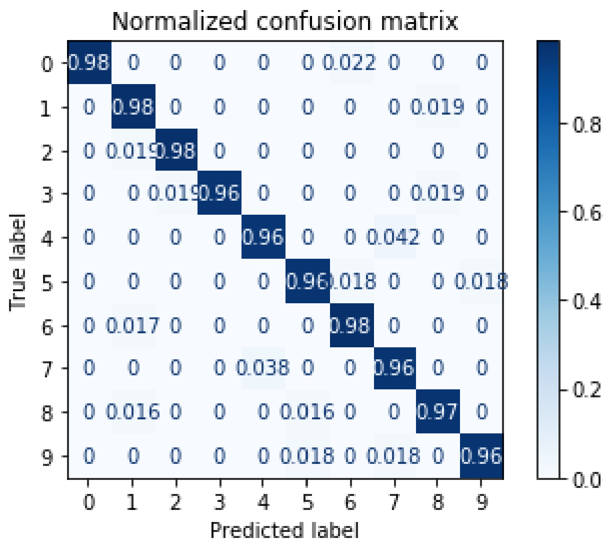


Figure 22: Confusion matrix for neural network on digits dataset

As can be seen from the confusion matrix, the model seems to perform very well on the test set, with around 96-98% accuracy for each digit.

Boosting

For boosting, decision trees with a max depth of 1 were again settled on as the weak learner. A grid search was then run in order to try and hunt down the right number of estimators to use. After running the grid search and creating a validation curve as seen below, the methods didn't seem to agree on the correct number of estimators, with the grid search returning 47 and the validation curve pointing to only using 5.

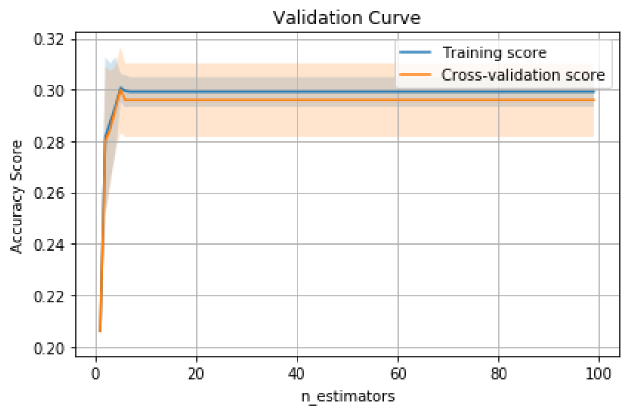


Figure 23: Validation curve for boosting algorithm varying the number of estimators

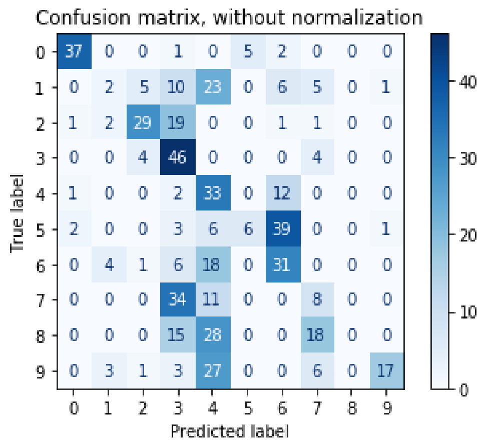


Figure 24: Confusion matrix for max depth =1, 5 estimators

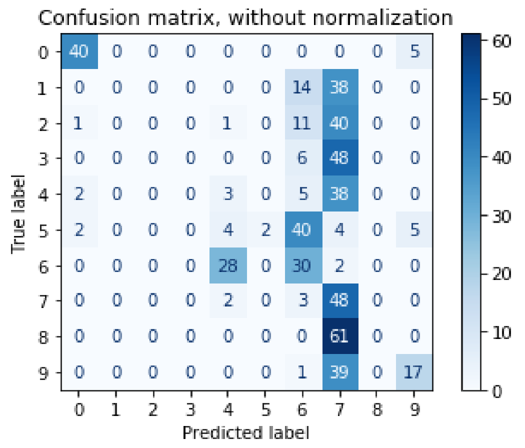


Figure 25: Confusion matrix for max depth=1, 37 estimators

After seeing these poor results for both 5 and 37 estimators, I went back and added depth to my Decision Tree being used as a weak learner. After increasing the max depth to 3 and re-running the grid search for a different number of estimators, the performance quickly got better as can be seen on the right. My next step would be to possibly change up the type of weak learner used in order to try and increase while performance while not overfitting the training data.

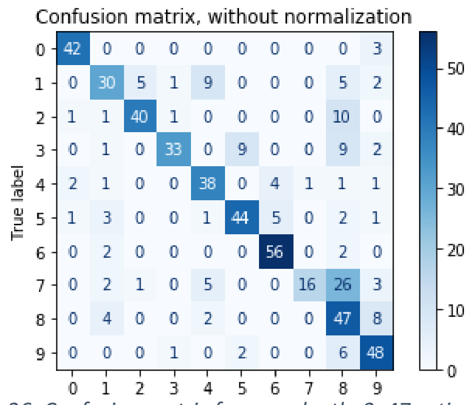


Figure 26: Confusion matrix for max depth=3, 47 estimators

Support Vector Machines

For the SVM algorithm, grid search was again first used to try to find the optimal combination of C, gamma, and kernel. Given the results, two validation curves were then run to vary the hyperparameters over a finer space to see if the region of the values chosen was in an area of high influence. As can be seen by the two validation curves below for C and gamma, the cross-validation scores calculated were not too sensitive to the hyperparameter values chosen.

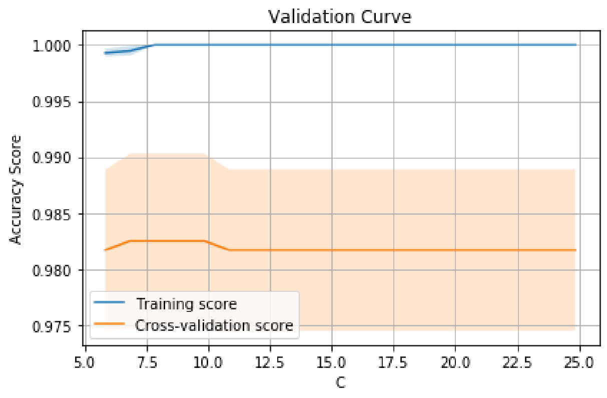


Figure 27: Validation curve for SVM varying C for the digits dataset

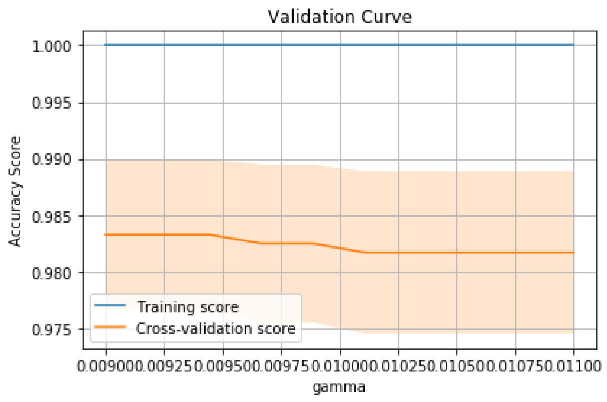


Figure 28: Validation curve for SVM varying gamma for the digits dataset

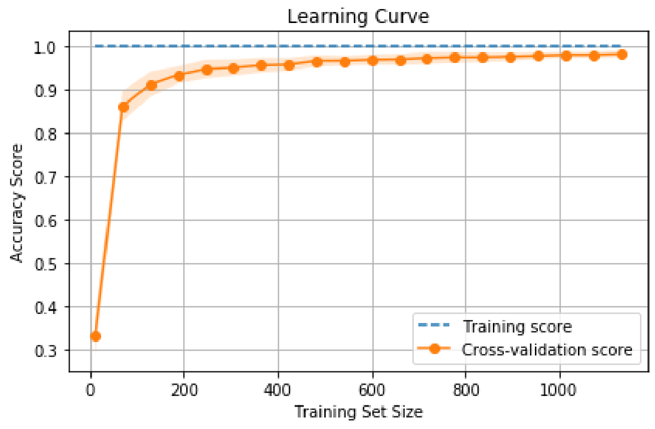
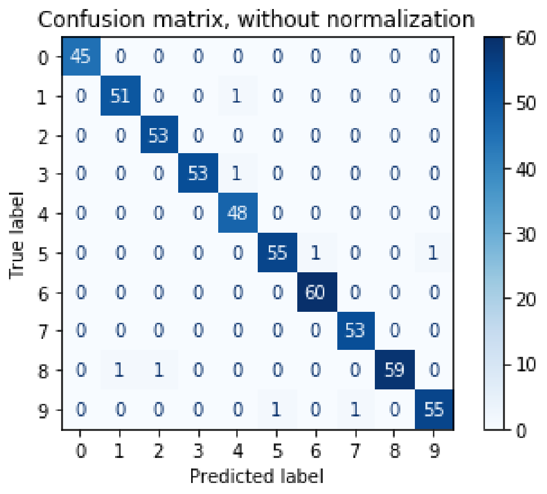


Figure 29: Learning curve for SVM on digits dataset



	0	1	2	3	4	5	6	7	8	9
0	45	0	0	0	0	0	0	0	0	0
1	0	51	0	0	1	0	0	0	0	0
2	0	0	53	0	0	0	0	0	0	0
3	0	0	0	53	1	0	0	0	0	0
4	0	0	0	0	48	0	0	0	0	0
5	0	0	0	0	0	55	1	0	0	1
6	0	0	0	0	0	0	60	0	0	0
7	0	0	0	0	0	0	0	53	0	0
8	0	1	1	0	0	0	0	0	59	0
9	0	0	0	0	0	1	0	1	0	55

Figure 30: Confusion matrix for SVM on digits dataset

As can be seen from the learning curve figure, the model was able to successfully learn from the training set given and shows a very low variance. This came through when applied to the testing set and the SVM algorithm performed with 99% accuracy. This can be seen in the confusion matrix figure shown above, where there are only a small number of mislabeled digits scattered about, with no more than one wrong label in any single category. This shows that the model has not been overfit in some small fashion that would cause the results to be skewed or allow a certain pattern in the handwritten digit to cause a lack of precision.

k-Nearest Neighbors

As with the wine quality dataset, the *k* Nearest Neighbor model comes down to deciding how many neighbors to use in the model. After running grid search and a validation curve on the model, the two methods agreed that using 3 neighbors would be optimal.

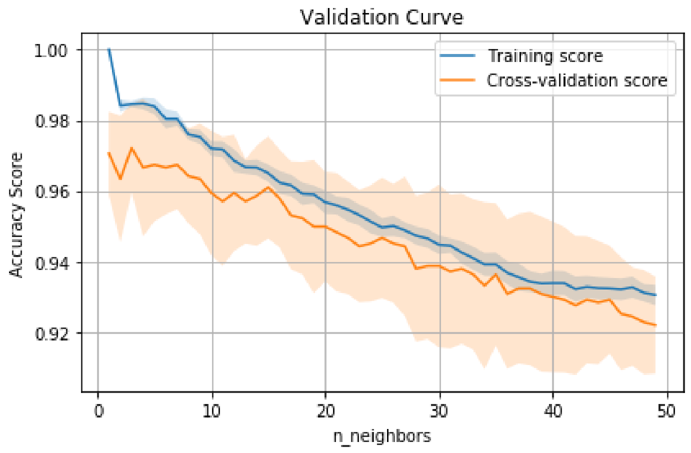


Figure 31: Validation curve for kNN varying the number of neighbors used for the digits dataset

Taking 3 neighbors, the model can be seen to do very well on all the training data, but not quite so much on the cross-validation data. One good aspect of this model is that the training scalability scales linearly with the size of the dataset as can be seen below.

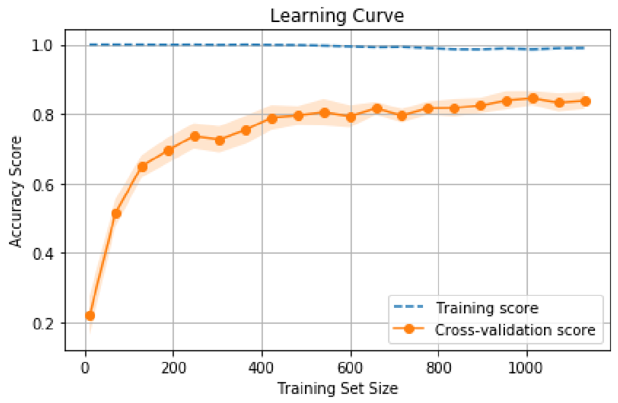


Figure 32: Learning curve for kNN for digits dataset

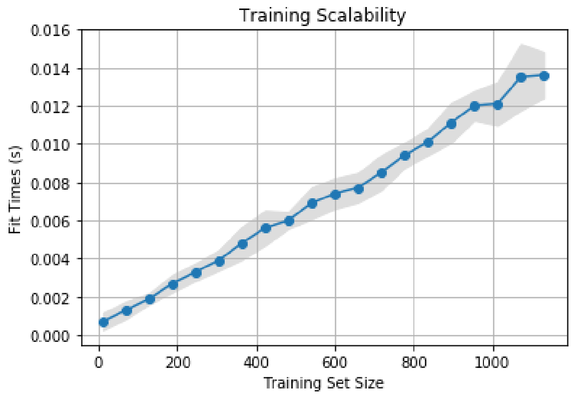


Figure 33: Scalability of kNN for digits dataset

This did result in performance that was better than would be indicated from the learning curve, coming in with an accuracy of 97% when ran against the test set.

Results

Table 2: Performance of all algorithms on the digits dataset

	Null Accuracy	**Decision Tree**	**Neural Network**	**Boosting**	**SVM**	***k*NN**
Time to train (s)	0	4.45	32.94	7.20	5.35	2.82
Accuracy (%)	10	85	97	78	99	97

When looking at the results, it seems that the SVM was the best algorithm for the digits dataset. While only taking a mere 5 seconds to train and resulting in a 99% accuracy, it is the clear winner. If there was a case where the training data was much bigger than what was used here, a case could be made for using a *k*-Nearest Neighbor where the time train would be cut in half for almost as good of performance.