Supervised Learning Analysis

By Josh Adams

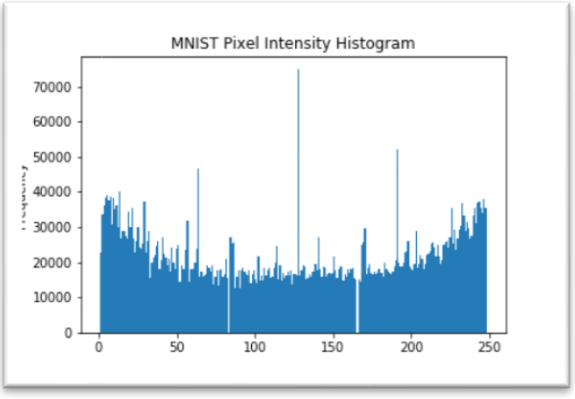
CS-7641 Machine Learning

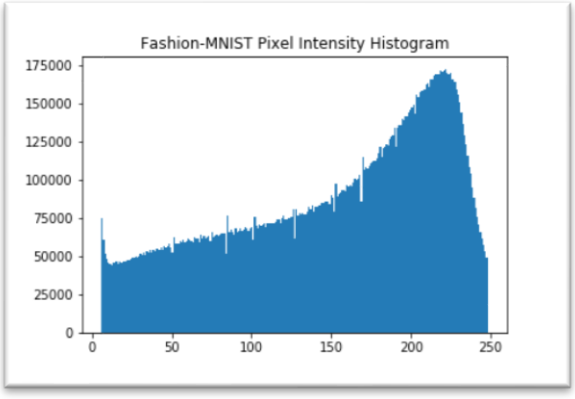
Classification Problems

MNIST dataset is a size-normalized and centered subset of the NIST dataset. MNIST was developed by Yann LeCun (Courant Institute, NYU), Corinna Cortes (Google Labs, New York), and Christopher J.C. Burges (Microsoft Research, Redmond). The dataset contains 30,000 hand-written digits which were collected by the Census Bureau, and 30,000 hand-written digits from high-school students which makes up the 60,000 patterns in the training set. The testing set contains 5,000 patterns from both the Census Bureau and high-school student sets. This is one of the most heavily used and tested datasets available. The MNIST dataset partially owes its popularity to the ease of exploration and understanding of the models trained using the dataset.

Fashion MNIST is meant to serve as a more challenging alternative to the MNIST dataset. The original MNIST dataset consists of very small variation between images, apart from the shapes depicted. The fashion MNIST dataset contains many more possible shapes along with added complexity from having various shading within the image. I was not able to achieve as accurate classifiers using the Fashion datasets as on the MNIST dataset.

I chose these datasets for multiple reasons. The first is that I want to understand why my models behave the way they do. One way to try and accomplish that is to use a simpler dataset, where variations in the model may be traced to the artifacts which produced them. I was not able to dive into these datasets that deeply, but I do hope to gain a true understanding of them over the semester.



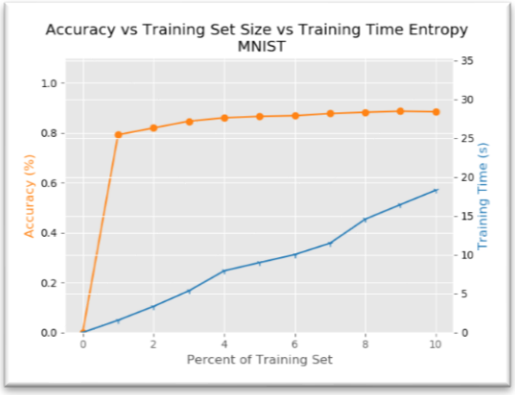


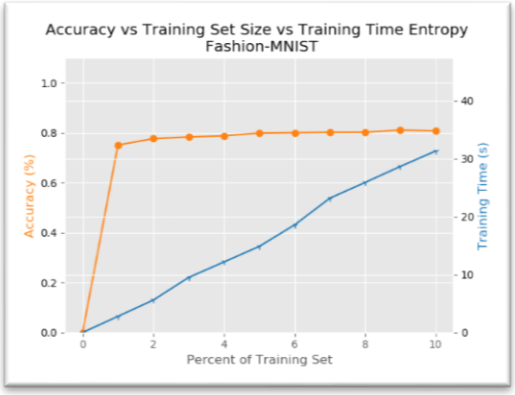
These are two histograms showing the distribution of pixel intensities in both training sets used. I removed some outliers, such as the number 0 and some near 255, to make the graphs more readable.

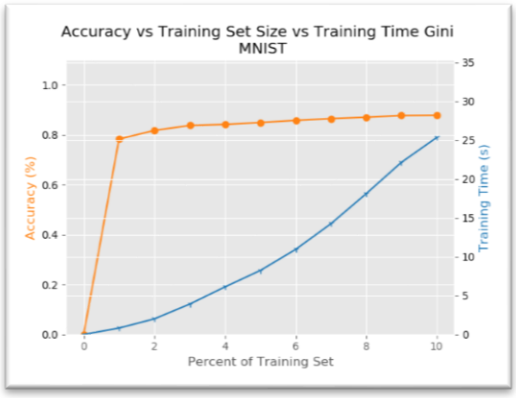
Decision Trees with Pruning

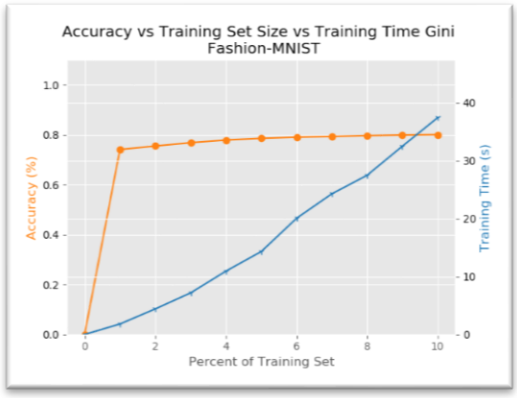
With decision tree you have a couple of choices when it comes to the way you split your data at each branch. I chose to work with Gini impurity and Entropy. Gini impurity is a measure of how often a randomly chosen element from a set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. Entropy which is treated as information gain when used in the construction of the decision tree classifier. Information gain is used to decide which feature to split on that will produce the purest child nodes. Using Gridsearch and cross validation set to 5, I tested using both Gini and entropy on various parameters. I found that the max depth for the trees could be set to some value between 20 and 30 without noticeable accuracy loss. I also tested the minimum number of samples required to split an internal node and the minimum number of samples needed to be a leaf node. I found that increasing the number for either of these parameters above 15 significantly impacted accuracy of the model. I ended up using a 4 for the minimum samples required to split and 4 for the minimum number of samples to be considered a leaf. I would consider this to be a form of pruning because restricting a node from being created has the same impact as just deleting a node.

I did implement an explicit method of pruning the decision tree developed by David Dale (March 2018, Stack overflow). This method uses the root node of the tree and traverses down the tree, explicitly removing node which do not meet a passed in threshold. When a node is removed, subsequently all connected sub-nodes are also removed. I made some modifications to David's method to remove nodes who had less than the threshold number of children. My datasets made an extraordinarily wide decision tree and even after pruning was still too large to include a picture of it. The decision tree images will be included in the container folders listed in the sources

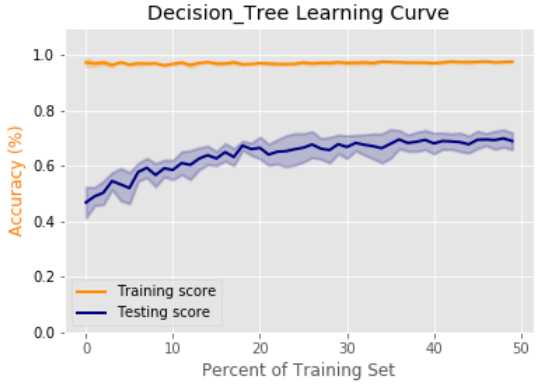


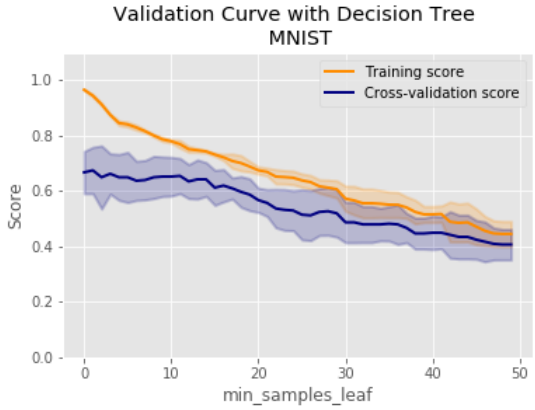






These charts show the two different splitting criteria trained against the two datasets. We have their accuracy in orange and time to train in blue, all as a function of percentage of the total dataset. It clearly shows that increasing the dataset size did not impact the overall accuracy very much.



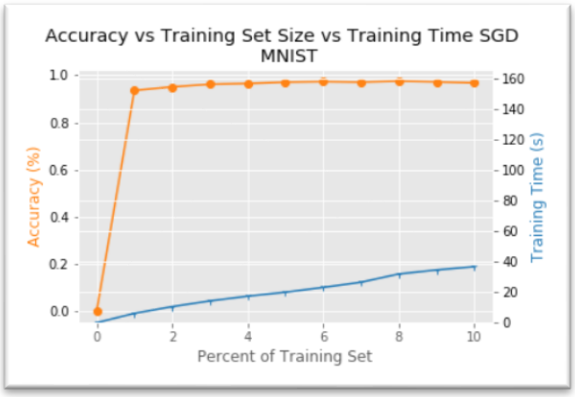


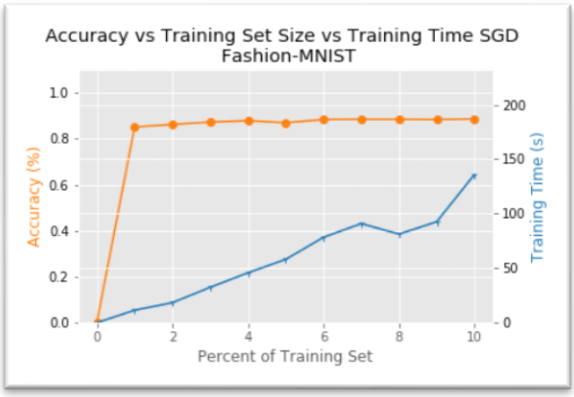
The learning and validation curves were generated using around 2000 samples for training, 1000 for testing and about 500 for validation. The learning curve has an upward trend so I would expect, if I added more examples for training, I would have been able to increase the testing score. I am not sure what exactly went wrong with the validation set but I was expecting to have low bias and high variance but the validation curve would indicate there is low variance. I am also not sure how to interperet the validation graph, having a downward trend.

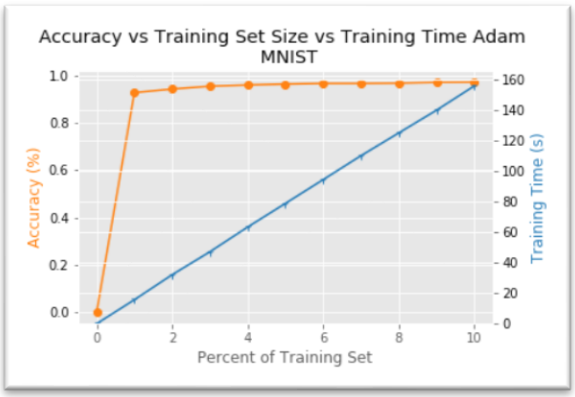
Neural Networks

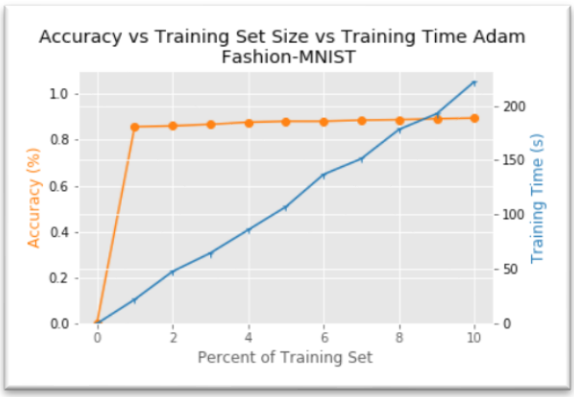
I found neural networks to be the best in terms of accuracy and the trade-off of training time. I used two different forms of optimizer, one being standard stochastic gradient descent and the other being referred to as ‘Adam’ which was proposed by Diederik Kingma and Jimmy Ba. Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS), consistently under-performed on the two datasets, subsequently I focused my attention on the two forms of stochastic gradient descent. I trained my neural networks with various sized subsets of each dataset. I used Scikit-Learns gridsearch to test various

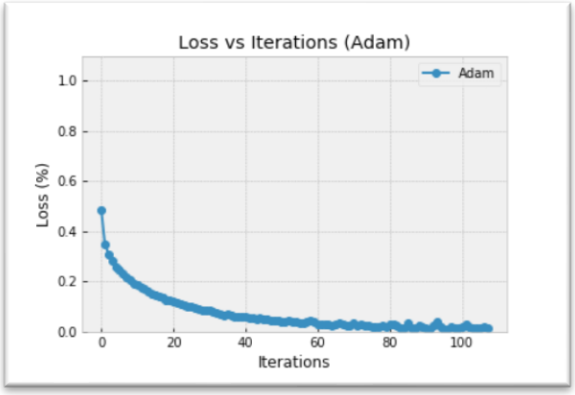
parameters for the neural networks and found that having more than one hidden layer was not needed for these datasets. The number neurons were also considered and increasing past 100 would increase training time, while increasing complexity in the network, which in many cases would reduce the overall accuracy of the model. As more neurons were added the training accuracy would slowly decrease, while simultaneously decreasing testing accuracy. The cause of the accuracy loss in the testing set was that the greater the number of neurons in the hidden layer the more capable the neural network was at over-fitting to the training set. I found it interesting that the training accuracy also began to fall albeit not as dramatically as the testing accuracy.

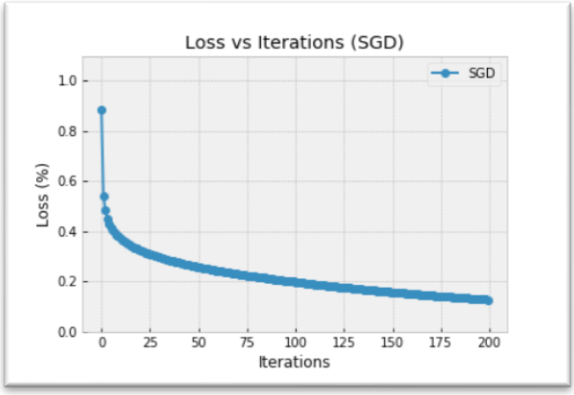










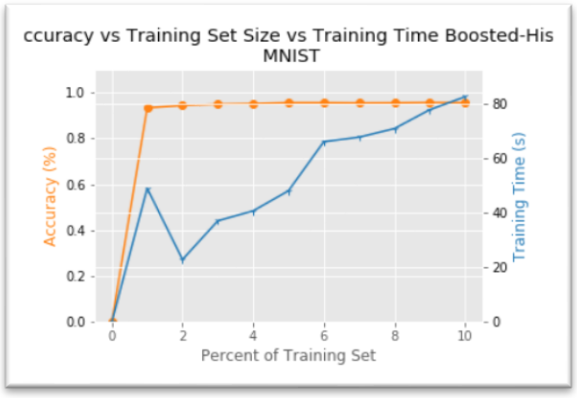


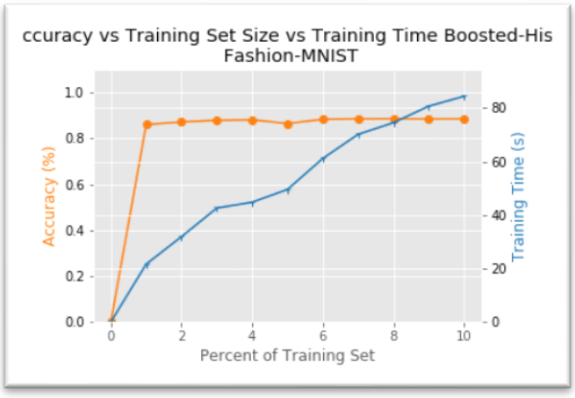
These are graphs of the average accuracy as a function of training set size, as well as training time as a function of the training set size. Both optimizers have dramatic diminishing returns once the data size was above 6,000 examples or 10% of the data. After that threshold, the gain in accuracy was heavily weighted by the time spent training. Depending on the task, the miniscule increase in accuracy may be worth the time spent training. It also is apparent the accuracy difference between the two datasets. The MNIST set stagnating near 97% accuracy while the Fashion-MNIST was in the upper eighties to low nineties.

This chart shows a reduction in the testing error and this would indicate low bias and low variance as the two scores were closing their gap.

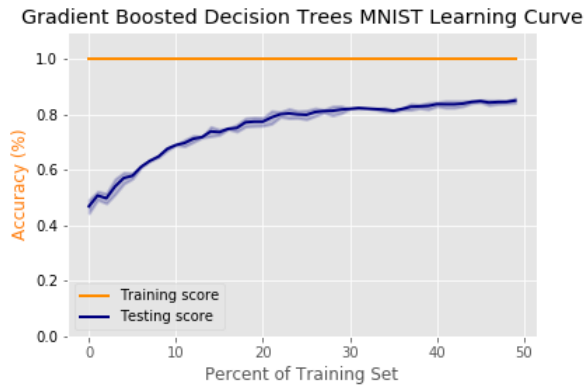
Boosted Decision Tree with Pruning

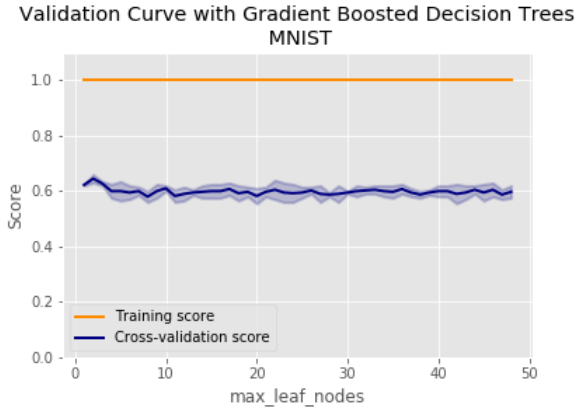
Starting with AdaBoost and GradientBoost, I used Gridsearch with various parameters for the number of estimators, the learning rate and the max depth. I quickly found that my tree depths could be limited to around 20 to 25, without sacrificing noticeable accuracy. The training process of the boosted decision trees was the most time consuming. Through grid search I found that having the number of estimators set to 100 and the max depth to 25, would overall be the best for accuracy on the two datasets. I was not able to test this classifier on the 10 different subsets of each dataset, as I did with the other classifiers. The reason was that for each increase in subset size, the time to train was almost exponentially increasing. To a point where it was estimated to take days for the classifier to train on one subset. I was not able to use the same pruning function I used for the standard decision tree, so I implemented pruning using minimal cost-complexity set to 0.01. This value is used within the classifier to remove nodes from the tree. Starting at nodes with the lowest cost-complexity measure, and continually pruning until the pruned trees minimal cost-complexity is above that passed in threshold.





I found an alternative to GradientBoost called Histogram-based Gradient Boosting, which was drastically faster than GradientBoost. I was not able to find a way to prune that tree explicitly but as stated earlier I limited the max depth and the minimum number of samples for something to be come a leaf. This is a form of pruning as it does not create the leaf unless that threshold is met.

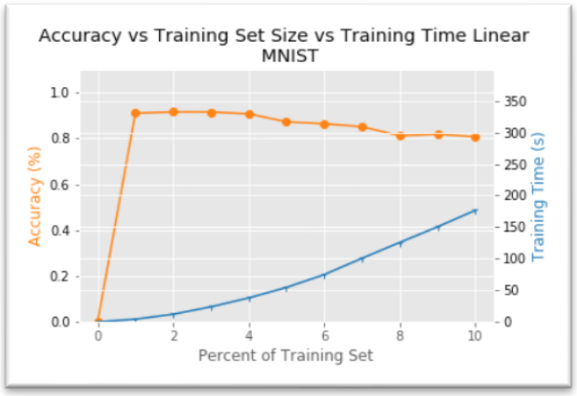


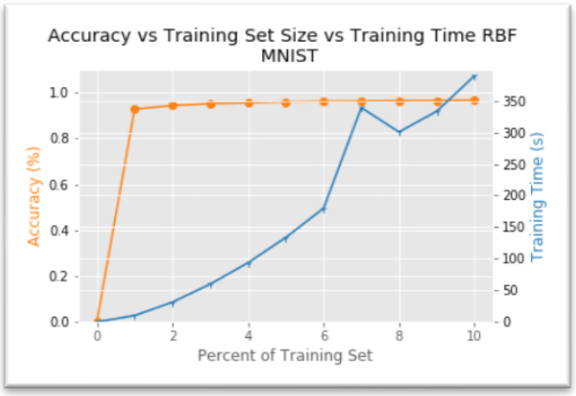


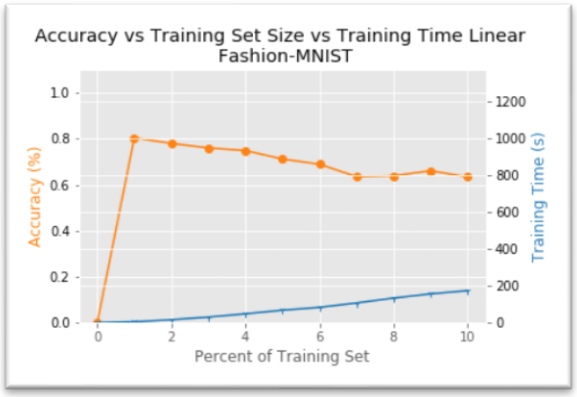
The difference between the training and testing score was very close, I believe this model has high variance and low bias because the training error was very low. I believe if I had trained with more data, this model could have been made better.

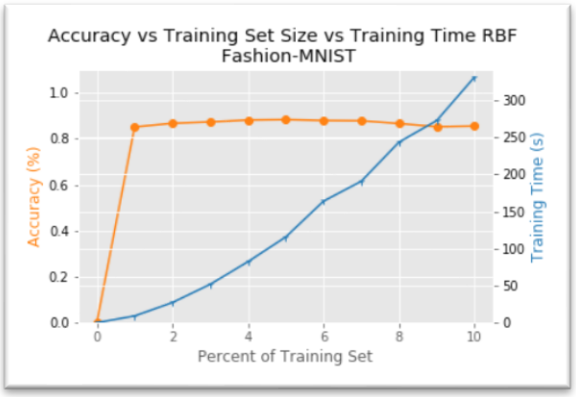
Support Vector Machines (SVM)

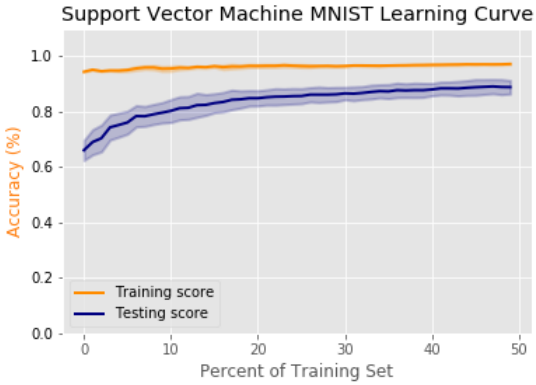
I used three different kernels for my SVM setup, polynomial, radial basis function, and linear. Using Gridsearch, iteratively create various models using the different kernels and other parameters such as the modifying the gamma. I was not able to find any significant differences when changing the gamma. I ended up only focusing on the kernels as to try and reduce the required time for the Gridsearch process. I was able to achieve moderate success with the SVM in terms of accuracy. I found the SVM training quickly and the process to predict values slow.

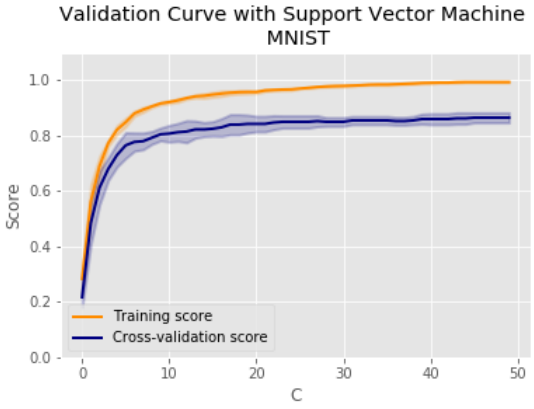








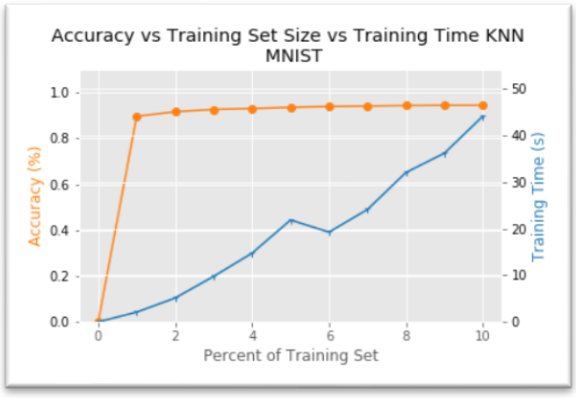


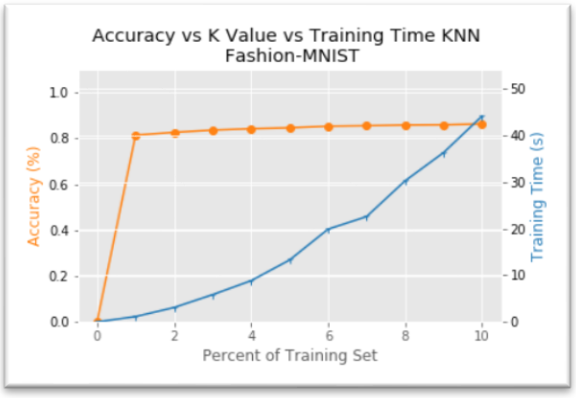


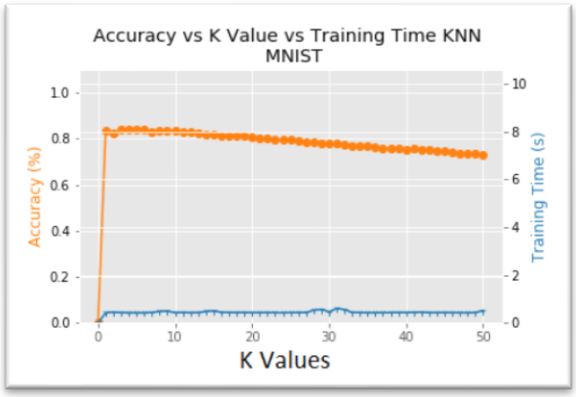
The SVM classifier has low variance and low bias as it was converging towards the training score.

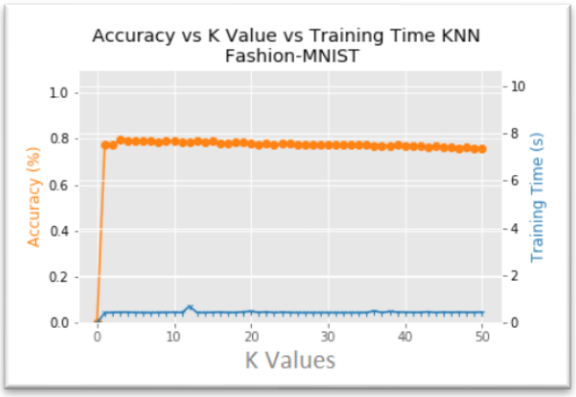
K-Nearest Neighbors (KNN)

KNN was one of the quickest algorithms to train over all but by far the most time consuming to perform predictions. I did not realize until exploring the algorithm that it must keep a copy of the entire dataset in order to provide predictions. This is a major drawback of this algorithm and more than likely, one of the reasons it was so incredibly slow with making predictions.

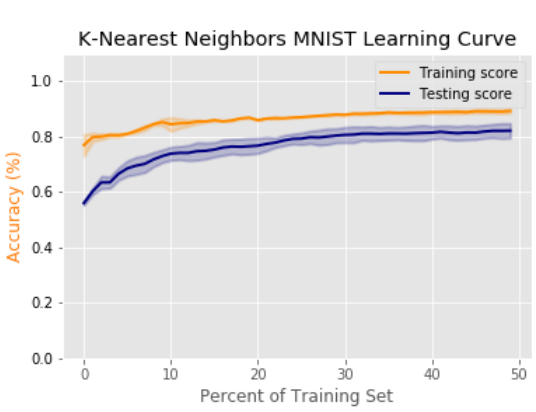








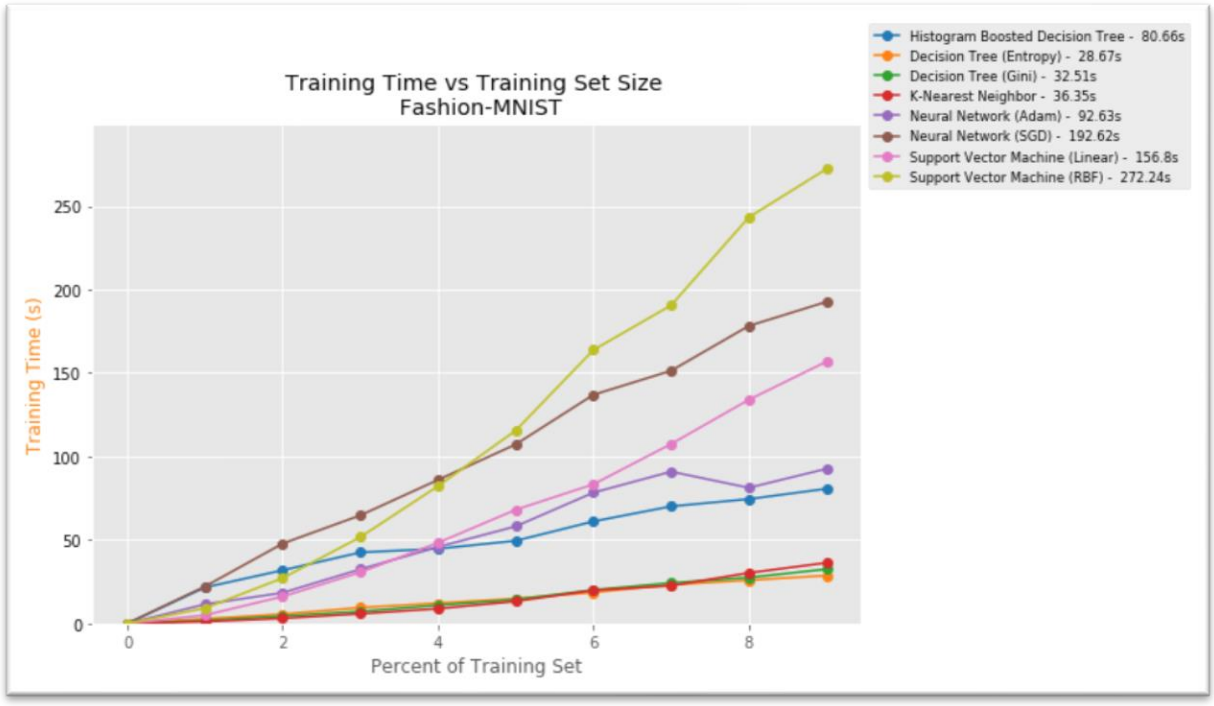
Just like with many of the previous classifiers, you do not gain much when increasing the size of the training sets, other than extended training times. This KNN classifier has a low bias and low variance, as the scores between the both the training and testing converge to be very close to one another.



This KNN classifier ended up having high bias and low variance. It also shows that I did not need to add any more to my training set.

Classifier Comparison

When testing these classifiers on the same datasets, allows for easier comparison between measurements. Without going through this process, I would not have known how different the various algorithms perform. The neural network performed overall the best as it had one of the highest accuracies and one of the lowest training times.



The graphs show how much the training time varies between the algorithms on the different datasets. All the algorithms performed relatively well but each comes with its own set of challenges such as having fast training times but extremely lengthy prediction time.

Sources

- **Files**

https://www.dropbox.com/sh/q4f7ptc6t4bb1ol/AAD3t1L8yTE3g_CPC6cH6Ajga?dl=0

- **Decision Tree Pruning**

https://stackoverflow.com/questions/49428469/pruning-decision-trees

by David Dale

- **Decision Tree**

https://en.wikipedia.org/wiki/Decision_tree_learning

- **MNIST Dataset**

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.

- **Fashion MNIST Dataset**

– https://github.com/zalandoresearch/fashion-mnist

Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf. arXiv:1708.07747

- Understanding Decision Trees for Classification

https://towardsdatascience.com/understanding-decision-trees-for-classification-python-9663d683c952 by Michael Galarnyk

- **Image Extract**

https://gist.github.com/ceykmc/c6f3d27bb0b406e91c27 username:ceykmc

- **Libraries**
    - Scikit-Learn
    - Matplotlib
    - Numpy
    - Pandas
    - OpenCv
    - PyDotPlus
    - IPython.display