Athreya Anand
CS 4641
25 September 2018

Supervised Learning Deep Dive

Introduction

This project has analyzed five uniquely different supervised learning algorithms: Decision Trees, Neural Networks, Boosting, Support Vector Machines and k-Nearest Neighbors. In this paper, I will be looking into the results of running these algorithms with different parameters on the following datasets. By the end of this paper we will be able to determine why a certain algorithm is the most effective for a certain dataset and why its classified as the "best."

Dataset Summary

1. Letter Recognition: This csv consists of 20,000 instances of typed and differently distorted letters from 20 distinct fonts. Since the set is based on English, there are 26 plausible labels/classifications. Each letter shows up around 750 times in the set; thus, the data can be considered a close-to balanced distribution. The sixteen features are as follows: width of character box, height of the letter, the number of pixels comprised and the other thirteen describing the edges of letters and their positions. The dataset is all discrete values and consists of no continuous ones.

   I personally liked this dataset due to its implications in optical character recognition. There are tons of systems and tools for the task at hand and I thought it would be immensely interesting to see how such classifications are made at a surface level using five unique supervised learning algorithms.

2. Fashion Recognition: This dataset is a composition of Zolando's (a fashion magazine) article images. The set is comprised of 10,000 instances and is a close-to-perfect balanced distribution. Each instance is a 28x28 grayscale image, associated with a label from 10 unique classes. Thus, every image consists of 784 pixels in which each pixel is associated to a single pixel value 0 to 255 (lower is lighter while greater is darker). With the 784 pixels and 1 label, the dataset consists of 785 columns. The labels are the following from 0-9: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

   I chose this dataset mainly for two reasons. One was just the immense amount of features going into the classification; compared to the letter dataset, the fashion dataset has 4900% more features per item. Second was the field of optical image recognition; I chose one OCR (optical character recognition) with the letter dataset and one OIR (optical image recognition) with this fashion dataset. The idea that a computer/algorithm can identify different types of clothing accurately was baffling and led me to immediately choose this as one of my primary datasets.

Decision Trees

Decision trees are one of the popular algorithms and use simple decisions inferred from the dataset in order to construct a classification tree. To implement this and examine the results I used python's sklearn library

Letter Recognition

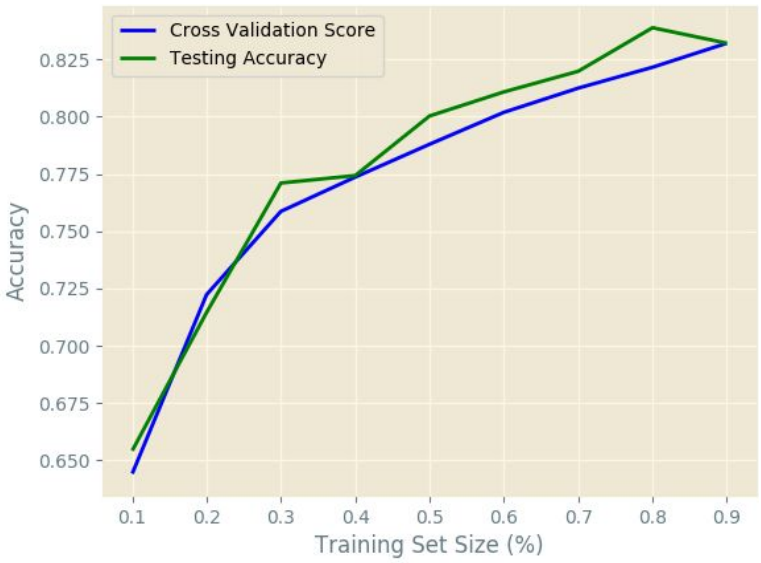

Figure 1: *Complexity graph showing the effectiveness of decision tree with increasing training set size (Character Recognition)*

As seen through figure 1, as the training size increases and grows through 100 percent, the accuracy of the decision tree also increases (testing and cross validation score). This clearly shows that decision trees perform immensely better the more data they are trained on/learn from. In the graph, we can also identify that the cross validation score and testing scores are very similar and both trend upwards with an increase in training set size; this matching pattern shows that the data/labels are evenly distributed across the set and validates are claim that a somewhat equivalent amount of unique labels exist. This tree has been *post-pruned* which will be discussed after looking at our fashion decision tree.

Fashion Recognition

Figure 2: *Complexity graph showing the effectiveness of decision tree with increasing training set size (Fashion Recognition)*

Figure two shows a similar graph to figure one; however, we can notice how the testing and cross validation accuracy do not go as far up as they do in figure 1. Contrasting the two previous figures, we can infer or make a solid hypothesis that decision trees are much more effective with datasets that consist of fewer features rather than more. We can see that the fashion sets' 784 features decreased the accuracy of its decision tree and that the 16 features of the letter recognition set made it much more reliable and consistent.

Post Pruning

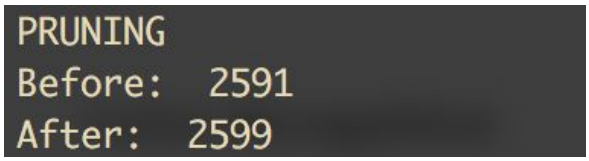

Figure 3: *Picture of the console after tree pruning has taken place (Letter Recognition)*

In order to prune my algorithm I chose to post prune my tree; to do this, I traversed the tree and removed all children of nodes with a minimum class count less than 5. By post pruning and removing excess children, we are able to get more accuracy from our decision tree and thus be able to better predict other similar datasets.

Boosting

To implement and analyze boosting I used python's Adaboost algorithm; i tested both the number of estimators and the accuracy versus the training size for both data sets. Let's dive into it.

Letter Recognition

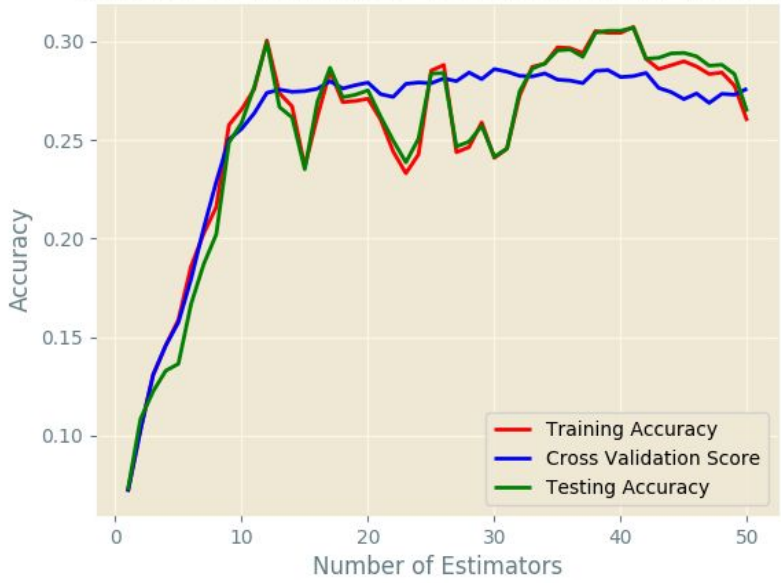

Figure 4: *Variable number of estimators used in boosting to test for accuracy (Letter Recognition)*

In figure four, I try to show the effect of having little versus a lot of estimators in my boosting algorithm. The cross validation score indicates that the accuracy plateaus around 12 estimators; chances are if we went past 50 estimators, we would see a significant drop in the cross validation score indicating overfitting and an abundant noise/inaccuracy in our boosting algorithm. Another aspect we can take away from the graph is the low accuracy even at the highest point. We can see that, even after it plateaus, the accuracy barely hits in between 25 and 30 percent clearly indicating that boosting might not be the best algorithm for this dataset.

Figure 5: *Boosting effectiveness learning complexity graph (Letter Recognition)*

Figure 5 has a very interesting suggestion that the training size does not have a necessarily precise correlation to the accuracy of the boosting algorithm. There is a somewhat general increase if we look at the point of 0.1 training size and 0.9; however, there is much more variance if you compare this graph to the decision tree graphs previously described. We can also see, like previously stated, our maximum accuracy is very poor and peaks at 27.5 percent.

<u>Fashion Recognition:</u>

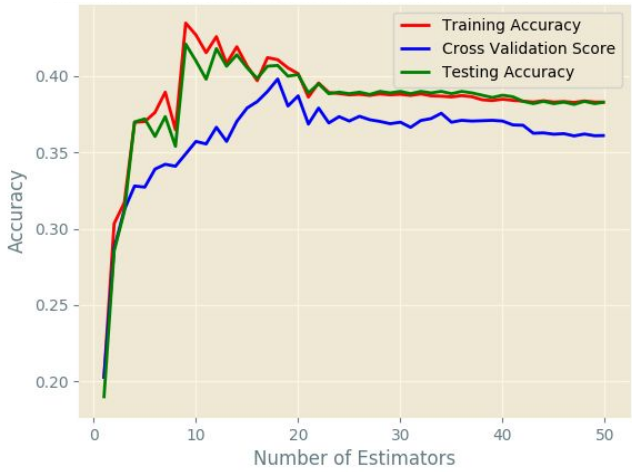

Figure 6: *Variable number of estimators used in boosting to test for accuracy (Fashion Recognition)*

Figure 6 is identical to Figure 4 in that it shows the accuracy depending on the number of estimators contained within the adaboost algorithm. It is also similar in the sense that the maximum accuracy is immensely poor. Although fashion accuracy is a little bit higher than letter at around 40 percent at its peak, it is still significantly poorer than the decision tree algorithm we previously visited.

Overall, we can infer from the two datasets that adaboost is not effective at all with datasets with a somewhat significant amount of features. If we had another dataset that was more binary or only had a select few features, it would result in a much more accurate adaboost algorithm.

Neural Networks

For neural networks, I again use python's sklearn algorithm; using this, I ran tests on the amount of hidden layers, the number of neurons per layer and the accuracy on different training sizes.

Letter Recognition:

Figure 7: *Complexity graph exploring amount of hidden layers (Letter Recognition)*

Figure 7 show us the effect of having different numbers of hidden layers in our neural network; we can clearly see that two or three layers are our most optimal. Similarly, the more neurons we have, the more accurate our algorithm becomes. The above graph for layers was tested with 32 neurons; however, since we know that having more neurons will give us more accuracy I did not try to make a graph to show that proportion. It is safe to assume that the more neurons we have, the better our algorithm functions. However, based on figure 7 we can tell that having more hidden layers than needed is not a good idea; the relation between the number of neurons and accuracy is not even close to the relation between the number of layers and accuracy.

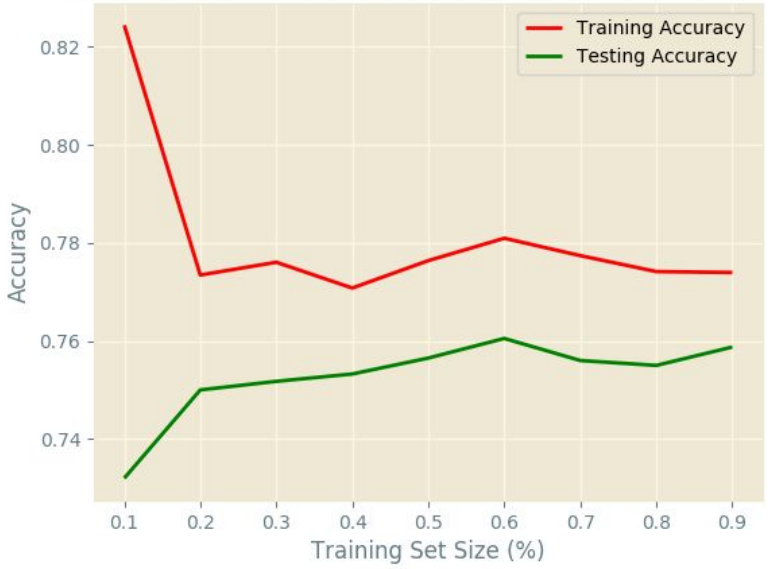

Figure 8: *Learning curve complexity graph with 0 hidden layers(Letter Recognition)*

After playing around with the number of layers, I found that 0 hidden layers gave me the best accuracy when it came to the neural network algorithm. As Figure 7 suggests, the more layers we add, the more inaccurate our predictions become. From the same graph we can see that it decreased accuracy from almost 85 percent to just 50 percent. Based off of figure 8 we can also infer that smaller training sizes result in significant underfitting and that larger training sets slowly fit the sets more and more accurately and allow the model to perform better.

Fashion Recognition:

Figure 9: *Complexity graph exploring amount of hidden layers (Fashion Recognition)*

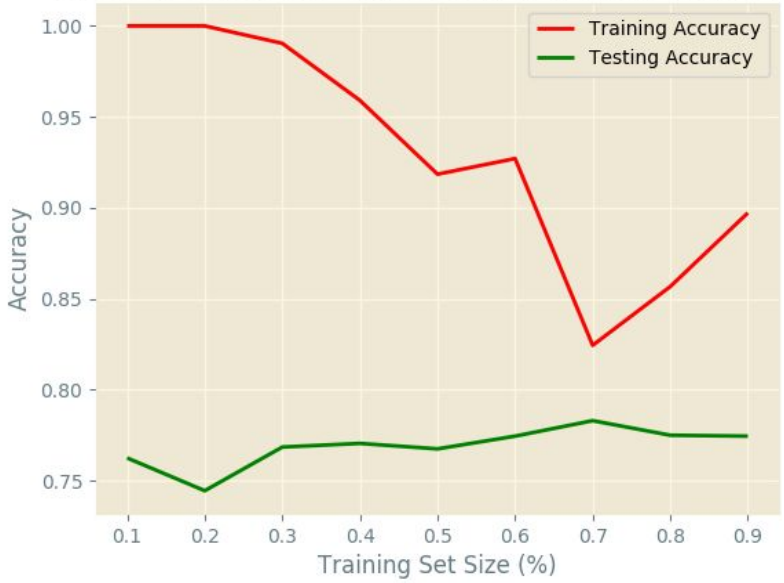

Figure 10: *Learning graph complexity curve with 4 layers (Fashion Recognition)*

Figures 9 and 10 are very similar to 7 and 8 yet have some differences. One major one is the plateau in performance with the increased amount of layers in fashion recognition versus the decrease in letter recognition. The overall performance for fashion was also much higher than it was for letter recognition when it came to the usage of a neural network in general. This means that with significant number of features, a neural network proves to be quite efficient compared to a decision tree and especially boosting.

However, one major drawback is clock runtime. The letter recognition runthrough took me around 15 minutes while the fashion recognition took me a mind boggling 1.5 hours. The previous two algorithms run significantly quicker than a neural network with the fashion dataset but do result in some cases much lower accuracy.

Support Vector Machine

To implement sklearn's support vector machine algorithm I had to first explore the different kernels I could possibly use my svm with before testing the accuracy and training set size. Let's take a look at what my thought process was.

<u>Letter Recognition:</u>

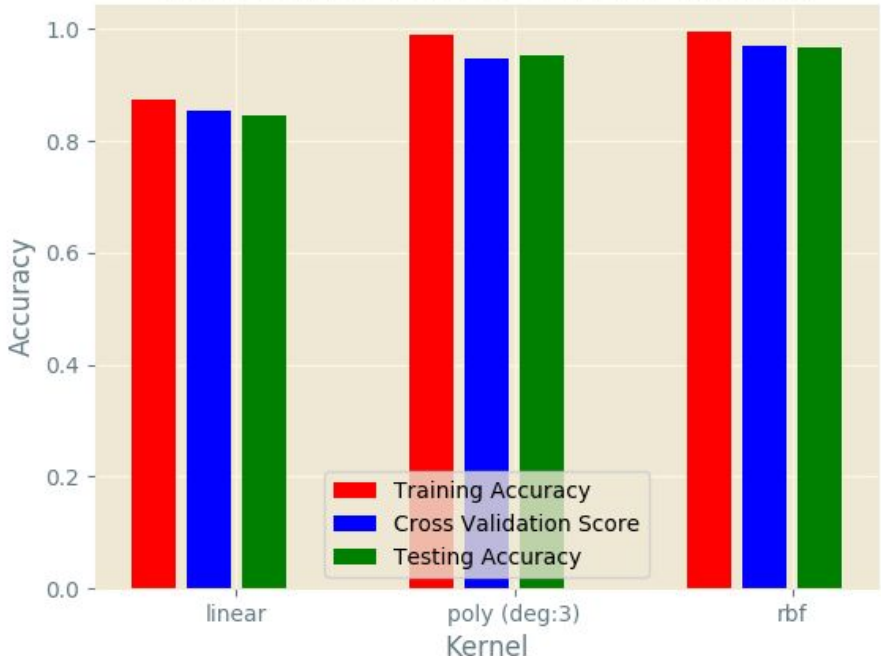

Figure 11: *The complexity chart to explore different kernels for svm (Letter Recognition)*

Figure 11 illustrates a bar chart consisting of the performance of three different kernels working on the letter recognition data set: linear, poly, and rbf. While working through the three, I noticed that linear took significantly less time than both poly and rbf; however, I also noticed that the more complex the algorithm the more accurate the results became. The RBF kernel performed with a 97 percent accuracy while linear was only around 85 percent.

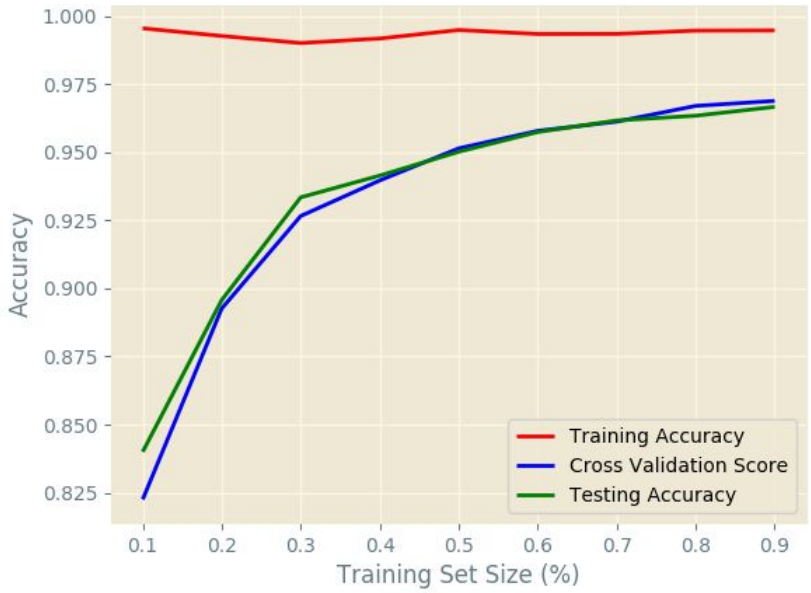

Figure 12: *Learning curve complexity graph for SVM using RBF (Letter Recognition)*

Figure 12 shows the training accuracy curve using the RBF kernel; The level of accuracy and perfect increasing pattern of the prediction is truly amazing. However, although the cv scores and testing accuracies are slowly increasing, it looks like it will not ever reach the training accuracy indicating a

small/decreasing amount of overfitting. With this we can come to the conclusion that the kernel overcomplicates the function and that a simpler one might have performed better in this instance.

Figure 13/14: *Kernel and Learning complexity curves for SVM (Fashion Recognition)*

Both figures 13 and 14 show a similar pattern between letter and fashion recognition: that training accuracy line is extremely high and consistent. Although our general accuracy is somewhat high, we can still be somewhat overfitting the data due to the discrepancy between training and testing accuracy. A unique element about Figure 13, nevertheless, is its extremely poor performance in rbf. This shows that rbf overcomplicated the algorithm and significantly decreased the overall performance. Thus, the most complicated kernel isn't always the best/most accurate for the dataset at hand. We can also make the case that too many features practically guarantees that rbf will not be an efficient kernel to be used within the svm algorithm.

K-Nearest Neighbors

Knn is an algorithm that tries to classify the instance by comparing it to its k nearest neighbors and identifying if it belongs to a specific cluster or not. The first step is to find the optimal amount of neighbors to compare and then see the accuracy with different training set sizes.

<u>Letter Recognition</u>

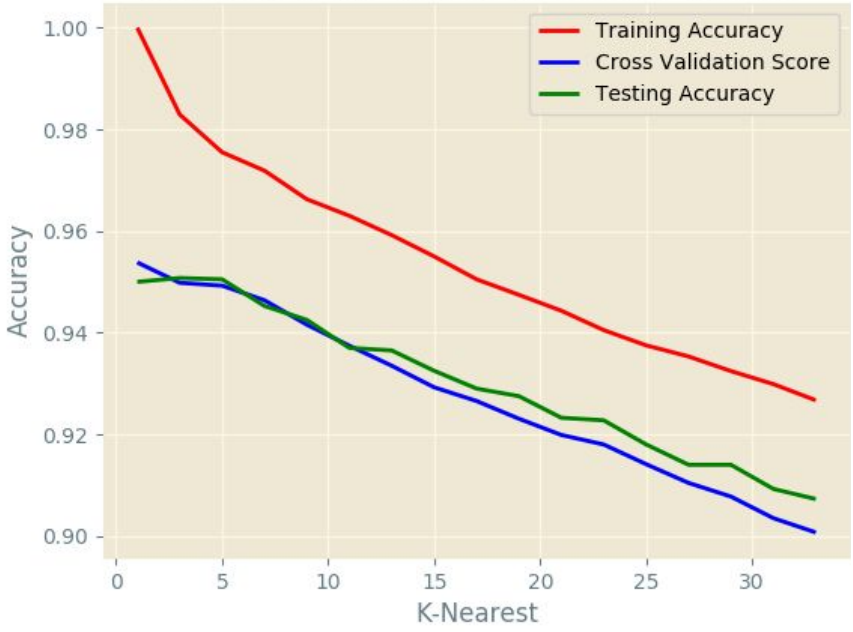

Figure 15: *Complexity graph for amount of k (Letter Recognition)*

Running the algorithm on values between 1 and 35, I got the graph of figure 15 in which we can see that the algorithm performs much better on lower k values. On these lower k values, the model performs much better at 95 percent compared to just 90 on the far end. However, there again is a small amount of overfitting since the training and testing accuracies to have a significant discrepancy between them. We can also see that the ideal Ks is somewhere between 1 and 5. The accuracy decreases towards the end since the algorithm is now merely merging different clusters and resulting in larger more inaccurate clusters.

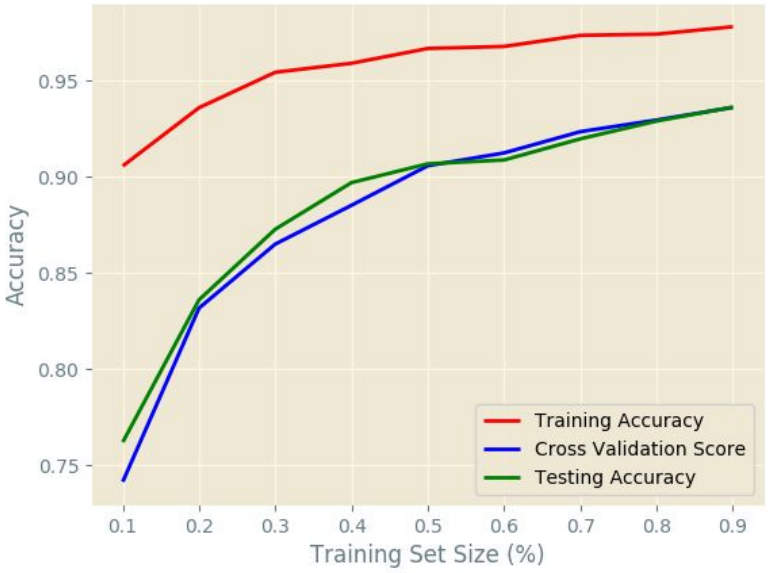

Figure 16: *Learning curve complexity graph (Letter Recognition)*

Looking at figure 16, we can infer that as the training size increases, so does the accuracy of the model. However, we can also see that the training accuracy is always higher than the testing accuracy of the data. This could be because the knn algorithm stores all the given data to compare the new inputs and

predictions it needs to make. Also, we can tell there is not much bias since the testing and cross validation scores are very similar in their curves. Overall, we see that the entire model performs around 93% accuracy, showing that with 3 neighbors is perfectly optimal for the set.

Figure 17/18: *Complexity graph for the amount of neighbors and learning curve (Fashion Recognition)*

Figure 17/18 reiterate our points claimed with figures 15 and 16 and show a very similar pattern. We can tell that the algorithm performs the best with very few neighbors; however, we can also see that the data is again overfit by identifying the general difference between train and test data. We did show that this is most likely due to the fact that the knn algorithm stores all data to create labeled clusters and thus generally has a much more accurate result when testing the train data.

Conclusion

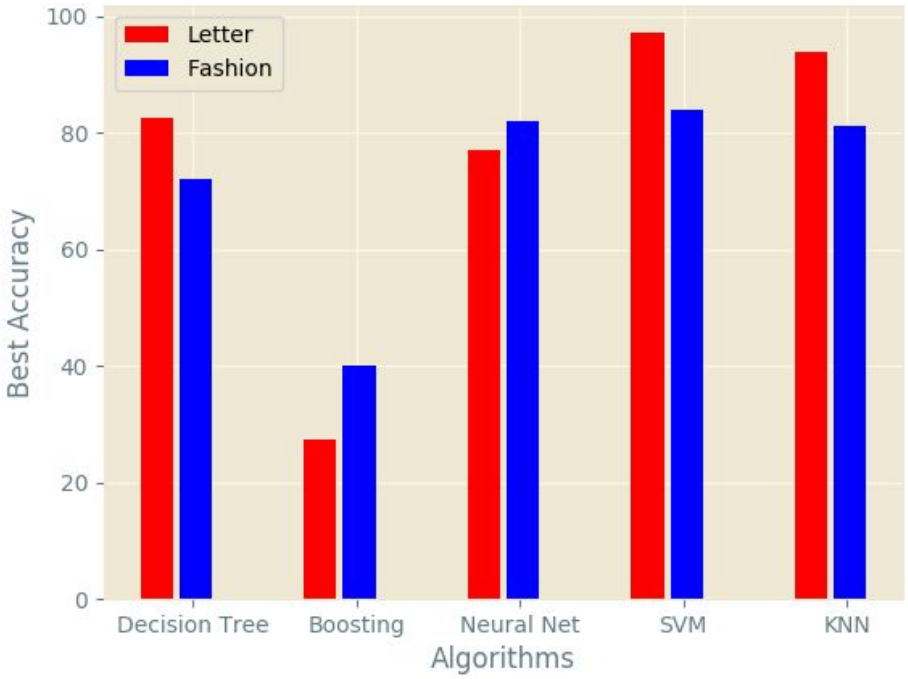

Figure 19: *A comparative bar graph of all tested/analyzed algorithms*

With our final look, figure 18 shows the overall performance on every algorithm we have taken a look at; the chart was created considering only the best accuracies with the highest training sizes. It can be

seen that both datasets have a somewhat similar performance across all algorithms with no more than a 10 percent difference occurring between the two. Letter recognition did outperform fashion in three of the five algorithms; however, this can be attributed to the fact that the letter dataset has significantly less features but more labels and that the fashion one has 4000+% more features and less than half the labels of letter. Generally the SVM algorithm performed the best for the letter dataset around 97 percent while Neural Networks, SVM and KNN performed around the same with the fashion set around 83 percent. The most unsuccessful algorithm for both sets was boositing since the kernels I used didn't maximize the amount of accuracy and tended to overfit the data. Time wise, neural nets took the longest with SVM and KNN coming in second and third place while boosting and decision trees came out on the faster end. In general, although the fashion dataset was very interesting it took nearly a dozen hours to run through all algorithms and tweak details within the code. Overall, I was able to classify the best supervised learning algorithm(s) for each dataset and compare them to one another to see nuances and discrpancies.