

Classification Datasets:

PenDigits Dataset:

The *PenDigits* dataset was created by collecting 250 samples from 44 different writers. Each writer wrote the digits zero through nine (in random order for random multiple of times for each digit until 250 samples were collected) and those images were scanned and encoded to convert image data into multivariate data. The overall goal of the algorithms will be to classify these encodings into one of the ten digits (zero through nine). The dataset contains 10992 instances with 16 attributes for each instance.

The *PenDigits* dataset is very interesting as image classification, specifically the classification of text, is widespread and in high demand in today's age. As numbers are often used as identifiers, counters, and metrics of measurement, it is crucial for machine learning to solve the task of successful and accurate classification. From hospitals to construction, handling certain tasks would be much easier if machines that had the ability to scan and read numbers were deployable. The dataset itself is very interesting with its large number of instances, but relatively small number of attributes per instance. As a result, I predict that very few of the algorithms we are assigned to analyze will face problems in overfitting, as the attribute space is much smaller than the instance space. From past experience in working with some of these algorithms, I predict that the neural net and KNN algorithms will perform the best than the other algorithms (neural nets because of the ability to handle complex data and KNN due the sheer volume of instances).

Vehicle Dataset:

The *Vehicle* dataset was created by taking 360° images of different types of vehicles and then extracting the silhouettes of the vehicles from each image. In other words, this dataset focuses on 2D shape and edge detection extracted from 3D objects. There are 846 instances with 18 attribute per instance. Examples of attributes include: compactness, circularity, radius ratio, and skewness about major and minor axes. The overall goal of the algorithms will be to classify each instance into one of four classes ("opel", "saab", "van", and "bus").

I chose the *Vehicle* dataset as it was also a dataset created using images and wanted to see the difference between this dataset and the *PenDigits* dataset (specifically the different types of problems we run into when working with primarily edge/shape detection versus whole image analysis). The dataset in itself is very practical as its successful implementation in traffic cameras would allow for classification of different types of vehicles and perhaps using that information for various tasks (i.e. police investigative work, traffic management, or logistical analysis of driving patterns). As the number of instances for this dataset is much smaller than the *PenDigits*

dataset, I predict overfitting will be a bigger concern for most of the algorithms we are going to use for analysis. Additionally, the authors of the dataset noted on the similarities between the “opel” and “saab” silhouette images so I predict those two will be hardest to tease apart and contribute the most to overall testing error. Unlike the *PenDigits* dataset, I have little experience with edge detection problems and would predict that neural nets would perform the best (simply from the complex nature of the data), but I doubt that regular neural nets will be able to completely resolve the differences between the classes (maybe deep or more complex neural nets).

Decision Tree Analysis:

| Input | | Output | | | | |
|---------|------------------|-----------|---------------|------------------|---------------|--------------|
| Pruned? | Confidence Level | Tree Size | Total Time(s) | CV Train Correct | Train Correct | Test Correct |
| Yes | 0.01 | 299 | 0.15 | 96.0651% | 98.7945% | 95.5434% |
| Yes | 0.05 | 321 | 0.15 | 96.2356% | 99.0219% | 95.6799% |
| Yes | 0.1 | 343 | 0.15 | 96.3153% | 99.1925% | 95.8163% |
| Yes | 0.25 | 367 | 0.15 | 96.3607% | 99.363% | 96.0891% |
| Yes | 0.5 | 377 | 0.15 | 96.3721% | 99.42% | 96.1801% |
| No | - | 395 | 0.13 | 96.3607% | 99.4541% | 96.0891% |

Table 1.1 – Decision Tree Training, Cross-Validation, and Testing Accuracies with various confidence levels for the *PenDigits* Dataset

| Input | | Output | | | | |
|---------|------------------|-----------|---------------|------------------|---------------|--------------|
| Pruned? | Confidence Level | Tree Size | Total Time(s) | CV Train Correct | Train Correct | Test Correct |
| Yes | 0.01 | 63 | 0.01 | 71.0059% | 83.432% | 70% |
| Yes | 0.05 | 95 | 0.01 | 71.8935% | 88.6095% | 69.4118% |
| Yes | 0.1 | 109 | 0.01 | 71.1538% | 90.2367% | 67.6471% |
| Yes | 0.25 | 127 | 0.01 | 71.3018% | 91.568% | 68.2353% |
| Yes | 0.5 | 137 | 0.01 | 71.5976% | 92.3077% | 68.2353% |
| No | - | 147 | 0.01 | 71.7456% | 92.6036% | 67.6471% |

Table 1.2 – Decision Tree Training, Cross-Validation, and Testing Accuracies with various confidence levels for the *Vehicle* Dataset

Consistent throughout both datasets, increased confidence level for pruning resulted in larger, but more accurate trees with respect to training, cross-validation, and testing accuracies. The only deviation from this general trend occurred in the *Vehicle* dataset, whose testing accuracy decreased as the confidence level increased (Table 1.2 – Test Correct). One plausible explanation for this trend could be poor generalization caused by overfitting the training dataset. As the tree for the *Vehicle* dataset grew larger, decreased pruning left more leaf nodes with specific outliers that were native to the training set and not the testing set. Comparing pruning and not-pruning, we see that the training set accuracy was the highest when the tree was not pruned, but as a result of overfitting, unpruned trees also had the largest gap between training

and testing accuracies. Cross-validation proved to very extremely useful as the resulting accuracies were very close to the testing accuracies, therefore offering a reliable proxy to approximate the power of decision tree learning for both the *Vehicle* and *PenDigits* datasets.

```

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  i  j  <-- classified as
215  1  0  0  0  0  0  0  1  1 | a = 0
  0 216 15  4  0  1  0  2  0  1 | b = 1
  0  6 231  0  0  0  0  1  0  0 | c = 2
  0  2  1 188  0  2  0  0  0  2 | d = 3
  1  3  1  0 226  0  0  2  0  2 | e = 4
  0  2  0  5  0 200  0  1  0  5 | f = 5
  1  0  0  0  0  2 207  2  0  0 | g = 6
  0  2  0  1  0  0  1 227  2  1 | h = 7
  5  1  0  0  0  1  1  3 197  1 | i = 8
  0  2  0  0  1  6  0  3  0 194 | j = 9

```

Table 2.1 – Confusion Matrix for *Vehicle* Testing Set with Pruned C = 0.01

Looking at the above confusion matrix, the decision tree, for the most part, was very successful in classifying most of the examples to the appropriate class. However, there were several classes the tree performed poorly on. Specifically, the decision tree had the most trouble correctly classifying digits one, five, and eight. One plausible explanation could be the lack of enough training examples to tease apart the relationship between confusing digit pairs (such as the difference between the digits one and two, difference between digit five and digits three and nine, and difference between digit eight and digit zero).

```

=== Confusion Matrix ===

  a  b  c  d  <-- classified as
22 21  0  2 | a = opel
13 21  1  8 | b = saab
  0  2 41  2 | c = bus
  0  2  0 35 | d = van

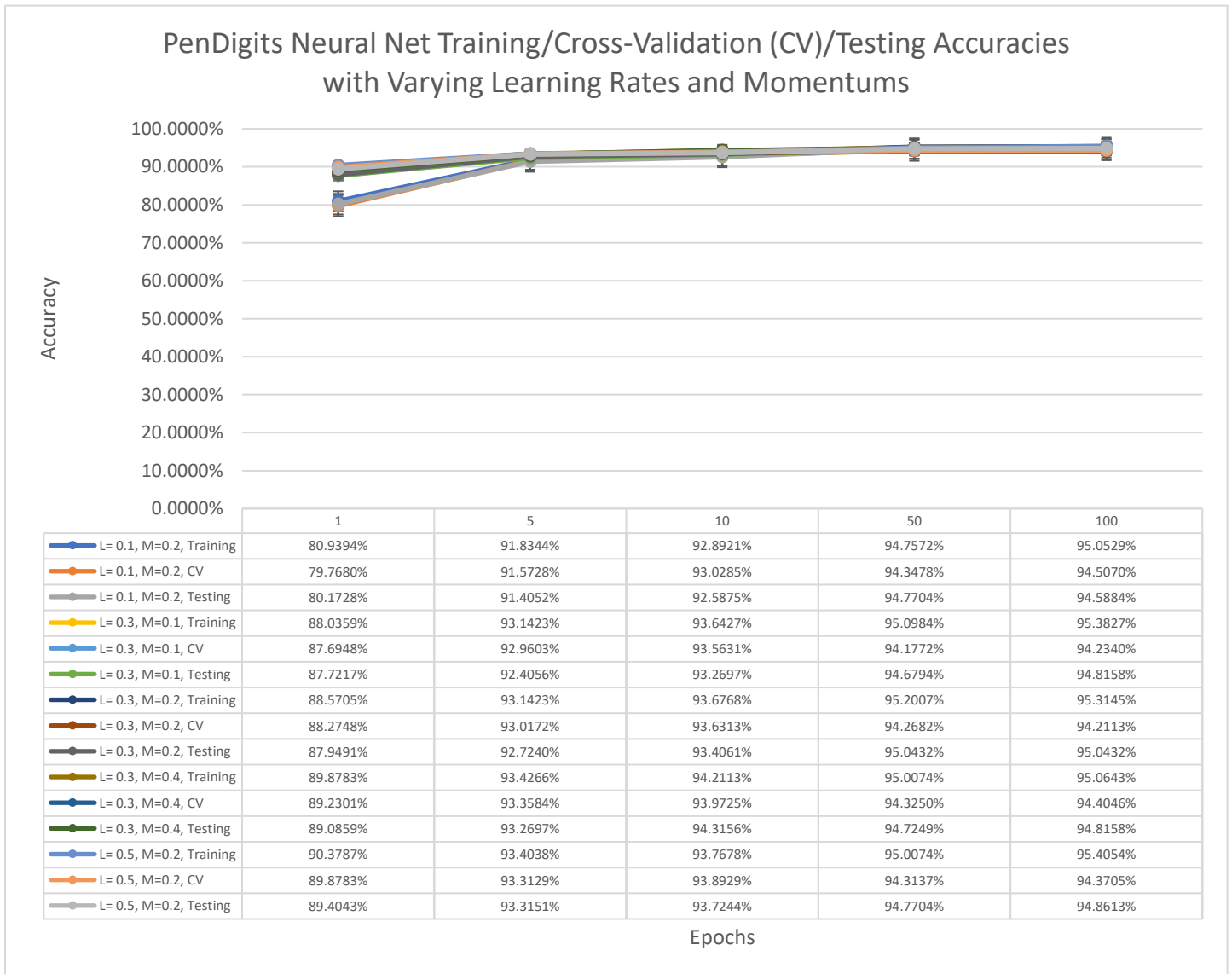
```

Table 2.2 – Confusion Matrix for *Vehicle* Testing Set with Pruned C = 0.01

The *Vehicle* dataset, on the other hand, had a poor classification rate compared to the *PenDigits* dataset. Looking at the confusion matrix above, we see that the most errors occurred in the classification of the “opel” and “saab” vehicles. This was discussed in the dataset introduction as these two vehicles had very similar geometries and there simply were not enough instances or enough attributes per instance for the decision tree to tease apart the differences between those classes. The “bus” and “van” classes, however, were very successfully classified with minor errors compared to the other vehicles.

Neural Net Analysis:

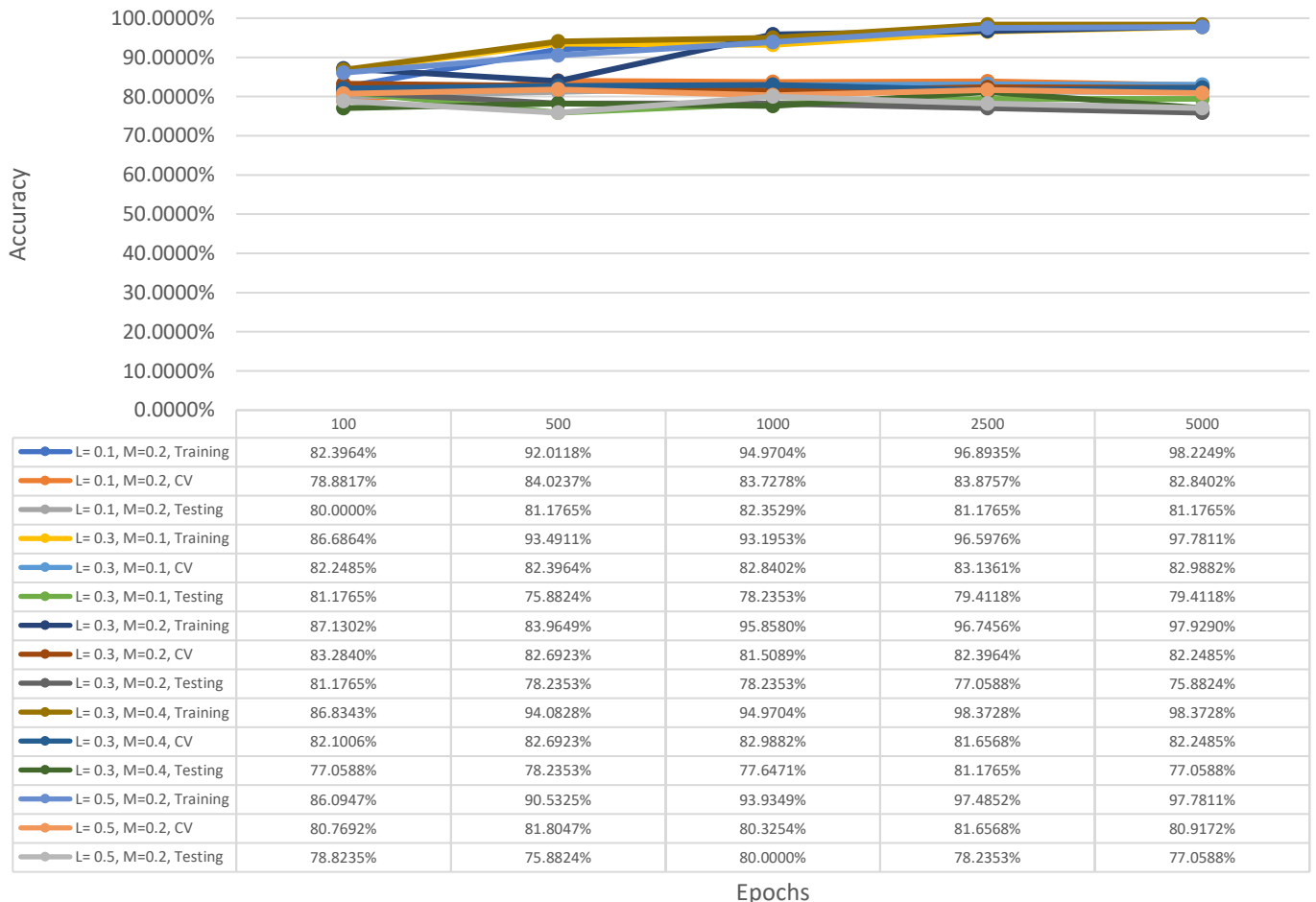
Overall, I ran five experiments, each with different combinations of learning rates ($L = 0.1, 0.3, \text{ and } 0.5$) and momentums ($M = 0.1, 0.2, \text{ and } 0.4$) in order to see how different hyperparameter values would influence the algorithm's accuracy on both the *Vehicle* and *PenDigits* datasets. Furthermore, as the two datasets had drastically different number of instances, in order to run the algorithms in a reasonable time frame, I used different values for epochs for each dataset. For the *PenDigits* dataset, I used small epoch values (1, 5, 10, 50, and 100) as it had a large number of instances. For the *Vehicle* dataset, I used larger epoch values (100, 500, 1000, 2500, and 5000) as it had a small number of instances. Below I plot training, cross-validation, and testing accuracies for each experiment.



Note: For the above learning curve, and all other learning curves in this analysis, I kept the range of y-values (accuracy) from 0% to 100% as instructed by the professor. If one is to zoom in, the various trends discussed below are more apparent and easy to visualize.

Overall, the neural net algorithm performed very well in classifying *PenDigits* dataset, with most of the accuracies ranging from 80% to 95%. As the number of epochs increased, the training, cross-validation, and testing accuracies also increased. The only exception to this trend was the testing accuracy decrease from 50 epochs to 100 epochs for $L = 0.1$ and $M = 0.2$. A plausible explanation for this outlier would be overfitting the training dataset. In general, however, overfitting was not a concern for the neural net algorithm as the difference between the training and testing accuracies was very minimal. Consequently, we also see very little difference between the cross-validation accuracies and the training accuracies (unlike when we used the Decision Tree) because cross-validation did not reduce overfitting error (as there was minimal overfitting). In general, as we increased the learning rate (from 0.1 to 0.3 to 0.5) and momentums (from 0.1 to 0.2 to 0.4), the accuracy of the overall algorithm increased. One explanation for this is the lack of noise in the dataset. As learning rate and momentum determine how quickly and by how much the weights change, having a steeper gradient descent allowed us to reach the local minima just as quickly as a slower gradient descent. Furthermore, as the number of instances was very large, each individual instance only contributed a small amount to the weight change.

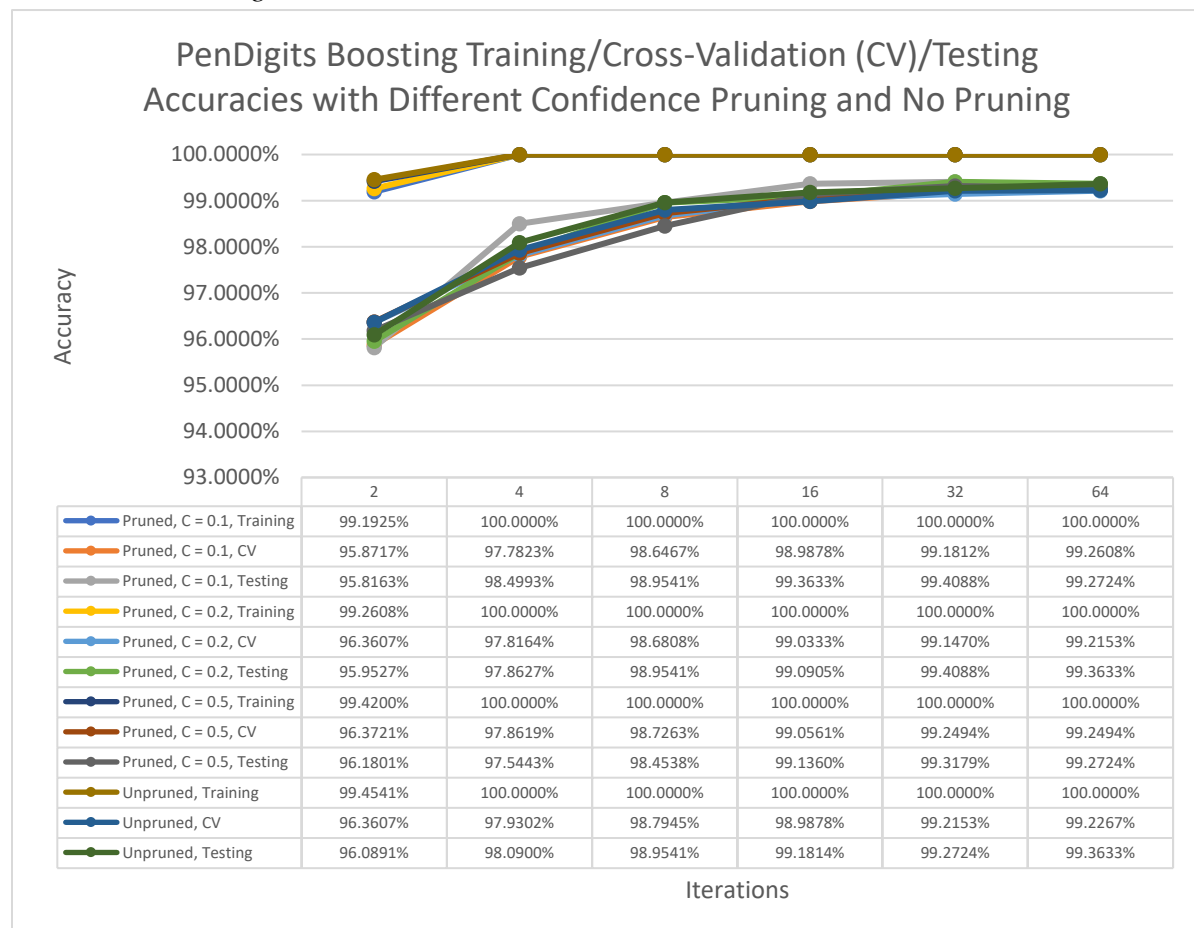
Vehicle Neural Net Training/Cross-Validation (CV)/Testing Accuracies with Different Learning Rates and Momentums



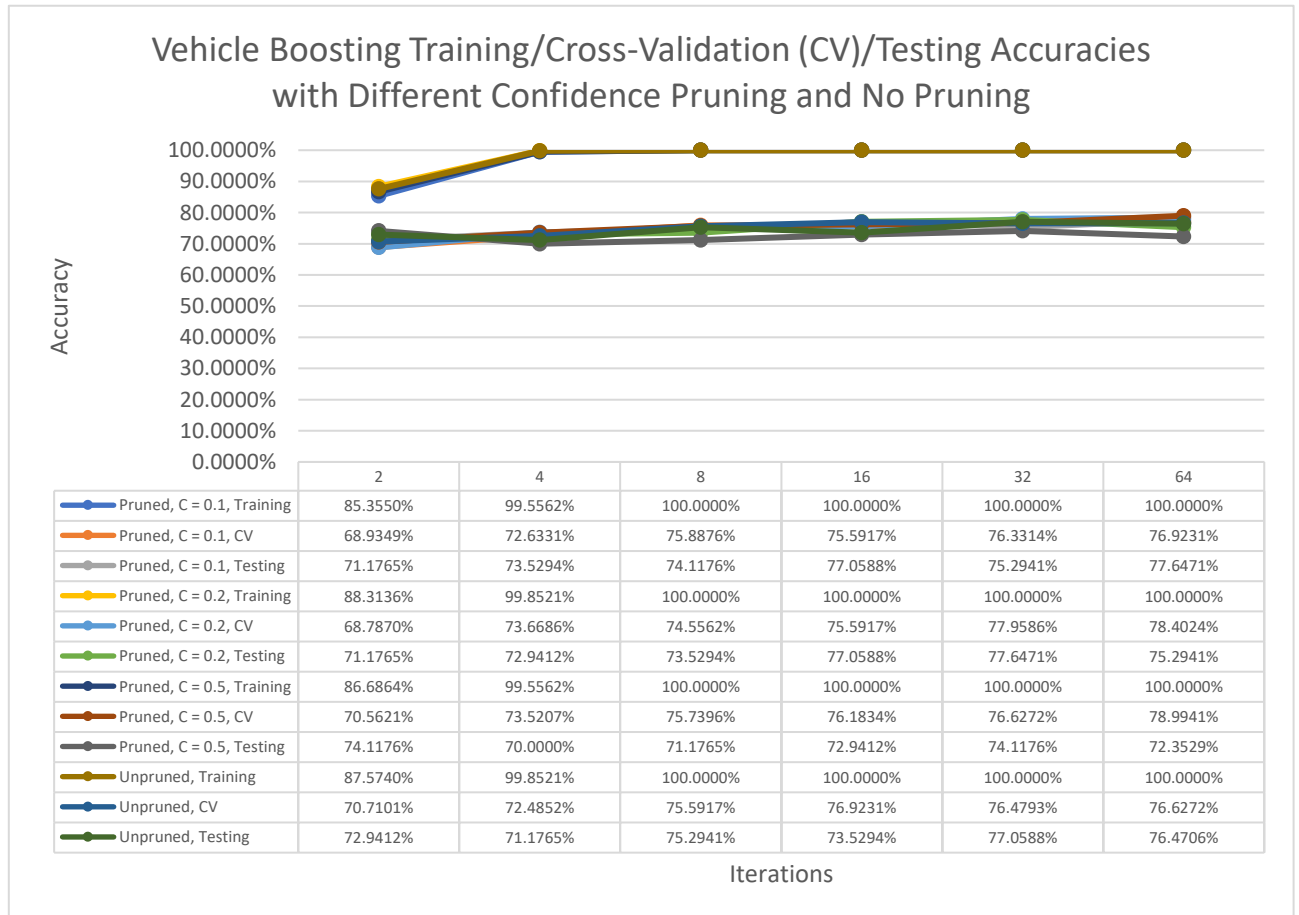
Likewise, the neural net algorithm performed well on the *Vehicle* dataset as well. One key difference is the decrease in testing accuracies as the number of epochs increased (mostly from 500 epochs onwards). What this tells us is that overfitting may have occurred in the training dataset and causing problems in classification during the testing phase. One plausible explanation for overfitting is the lack of enough instances in the *Vehicle* dataset than compared to the *PenDigits* dataset. Another explanation is the larger number of epochs compared to the *PenDigits* dataset. Nevertheless, from the graph we see a clear distinction between the training accuracies and both the cross-validation and testing accuracies (large gap between the sets of lines, clearer if zoomed in). Similar to the decision tree, the cross-validation accuracies were much better predictors for testing accuracy than compared to the training accuracies. The role of learning rate and momentum played slightly different roles in the *Vehicle* dataset. As the learning rate changed from 0.1 to 0.5 (with the momentum kept at 0.2), the testing accuracy actually declined from the $L = 0.3$ to the $L = 0.5$ transition. We see a similar trend when focusing on momentum, where the testing accuracy declines as well from $M = 0.1$ to $M = 0.4$ (keeping the learning rate at 0.3). Once again, this can be attributed to low number of instances and overfitting as rapid changes during gradient descent could cause the algorithm to get stuck in a suboptimal local minima. Compared to the decision tree, the neural net approach worked better for the *Vehicle* dataset.

Boosting Analysis:

Overall, I ran four experiments, each with the same “sub”-classifier (pruned decision tree with different confidence levels and unpruned decision tree), but with different number iterations to see how changing that hyperparameter would influence the algorithm’s accuracy on both the *Vehicle* and *PenDigits* datasets.



With respect to the *PenDigits* dataset, the AdaBoostM1 (specific boosting algorithm) performed very well. It was able to completely learn the entire training dataset by four iterations, while also having very high cross-validation and testing accuracies throughout all of the iteration values. As the training accuracies were close to the testing accuracies, overfitting did not play a huge role in this dataset. We can still see its impacts when looking at decreases in testing accuracies from 32 iterations to 64 iterations, but the decrease is very small. Pruning with different confidence levels or not pruning at all did not play that big of a role as the testing accuracies for each iteration are very close to each other. By averaging out and weighting the individual classifiers, boosting was able to better control how tree size impacted overall testing accuracy (where outliers did not have that much power as they did during the decision tree analysis). Boosting also performed similar to the Neural Net algorithm with respect to accuracy, with the exception of small impacts of overfitting being present in boosting. But as a result and similar to the decision tree analysis, cross-validation allowed for better prediction of the testing accuracies.

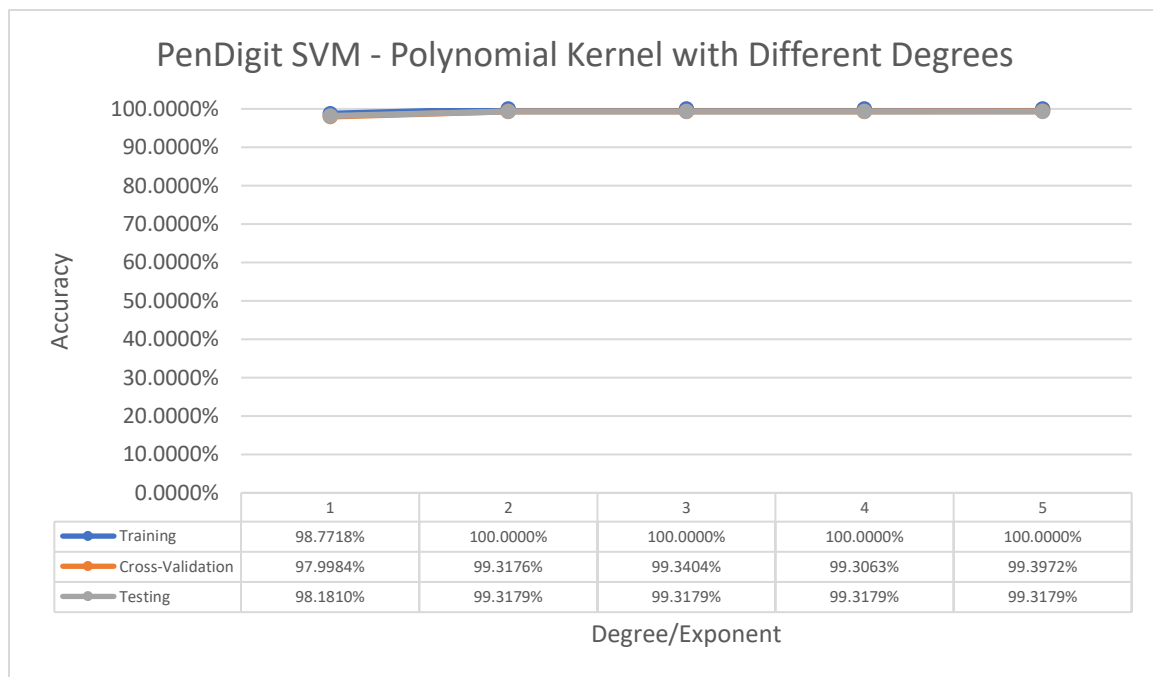


With respect to the *Vehicle* dataset, the AdaBoostM1 algorithm was not as powerful in terms of predictive accuracy compared to when it was used with the *PenDigits* dataset. Increasing the number of iterations did not seem to have a large impact on the overall cross-

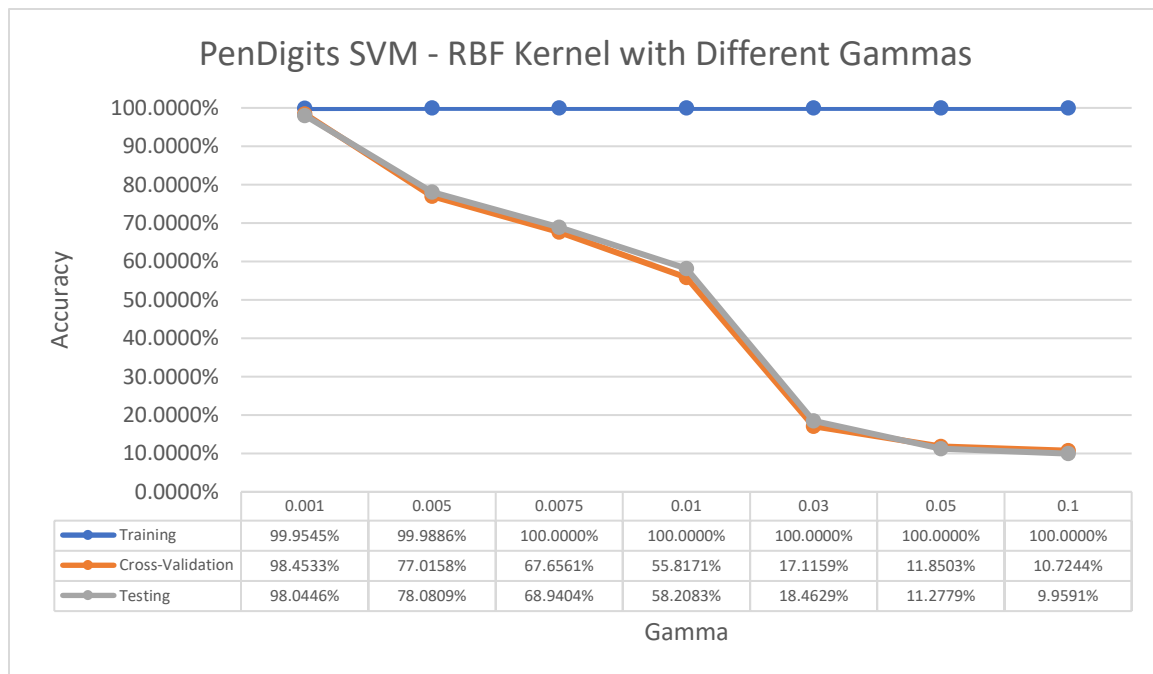
validation accuracies and testing accuracies. We see a clear distinction between the training accuracy and both the cross-validation and testing accuracies. This is most certainly due to overfitting of the training data, regardless of confidence level pruning, no pruning, or number of iterations. This is a bit surprising as the trend mirrors what we observed during the decision tree analysis for the *Vehicle* dataset. This adds evidence to overfitting explanation presented during the decision tree analysis, where lack of instance data and lack of enough attributes is causing a disparity between training and testing accuracies. Compared to the previous two algorithms, boosting performed similar to decision trees in terms of accuracy and worse compared to the neural net implementation.

Support Vector Machine Analysis:

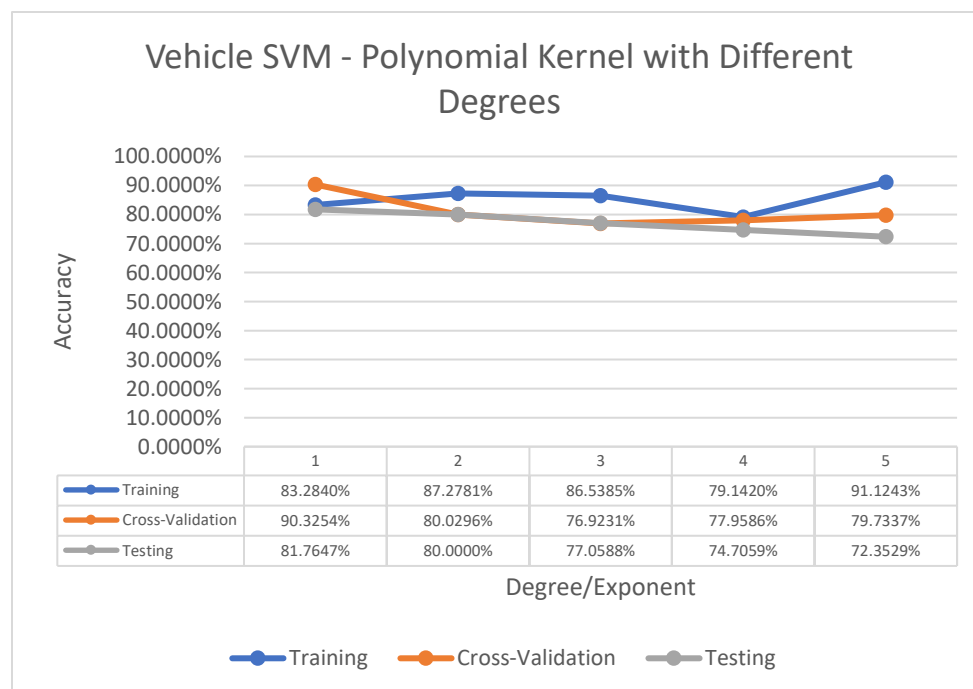
Overall, I ran three experiments: one with a polynomial kernel with varying degrees (exponents = 1, 2, 3, 4, and 5) and one with an RBF kernel with varying values for gamma (0.001, 0.005, 0.0075, 0.01, 0.03, 0.05, and 0.1). I tried to use larger values for gamma (10, 30, 50, and 100) with both datasets, but that resulted in really poor results with accuracies less than 10% so I did not include those in my analysis.

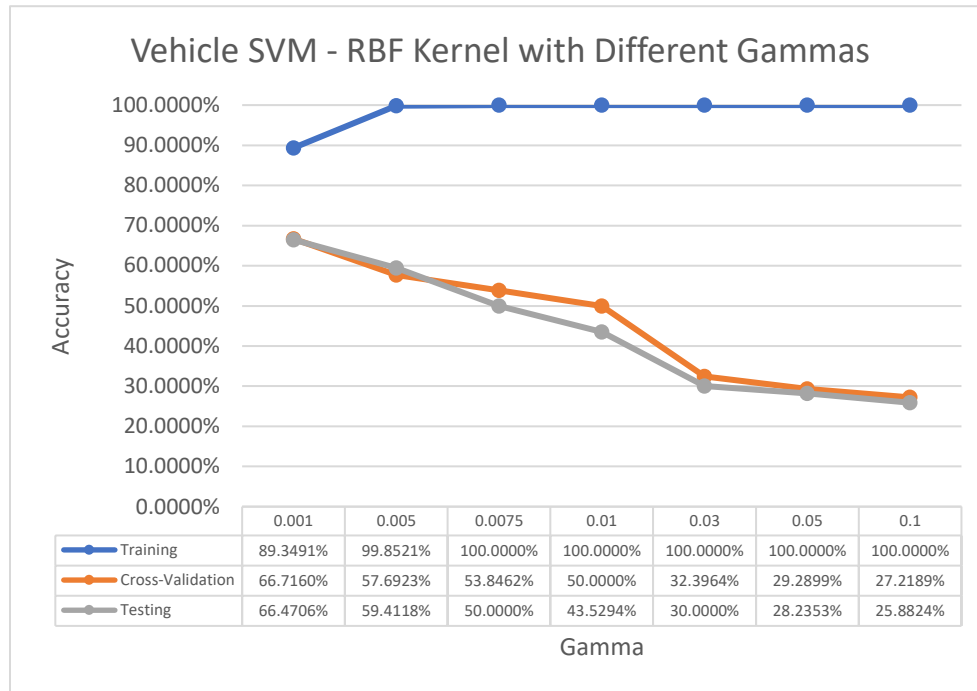


Note: Combined analysis for both kernels for the *PenDigits* dataset on the next page. Could not fit both graphs side by side without interfering with graph dimensions and overall appearance



Breaking down the SVM algorithm, the polynomial kernel performed much better than the RBF kernel with respect to the *PenDigits* dataset. The polynomial kernel was able to completely learn the training dataset starting at degree two and we observe very little differences between training, cross-validation and testing accuracies. Similar to the other algorithms discussed earlier, the *PenDigits* dataset seems to be very well protected against overfitting, at least with a polynomial kernel. With a RBF kernel, on the other hand, the cross-validation and testing accuracies plummet as we increase gamma values whereas the training accuracy becomes fixed at 100%. Overfitting is playing a small role in this drop in accuracy along with the use of suboptimal hyperparameter values for gamma. In short, the RBF kernel performed much worse than any of the previous algorithms. However, the polynomial kernel performed on the same par as the boosting and neural net algorithms for the *PenDigits* dataset.



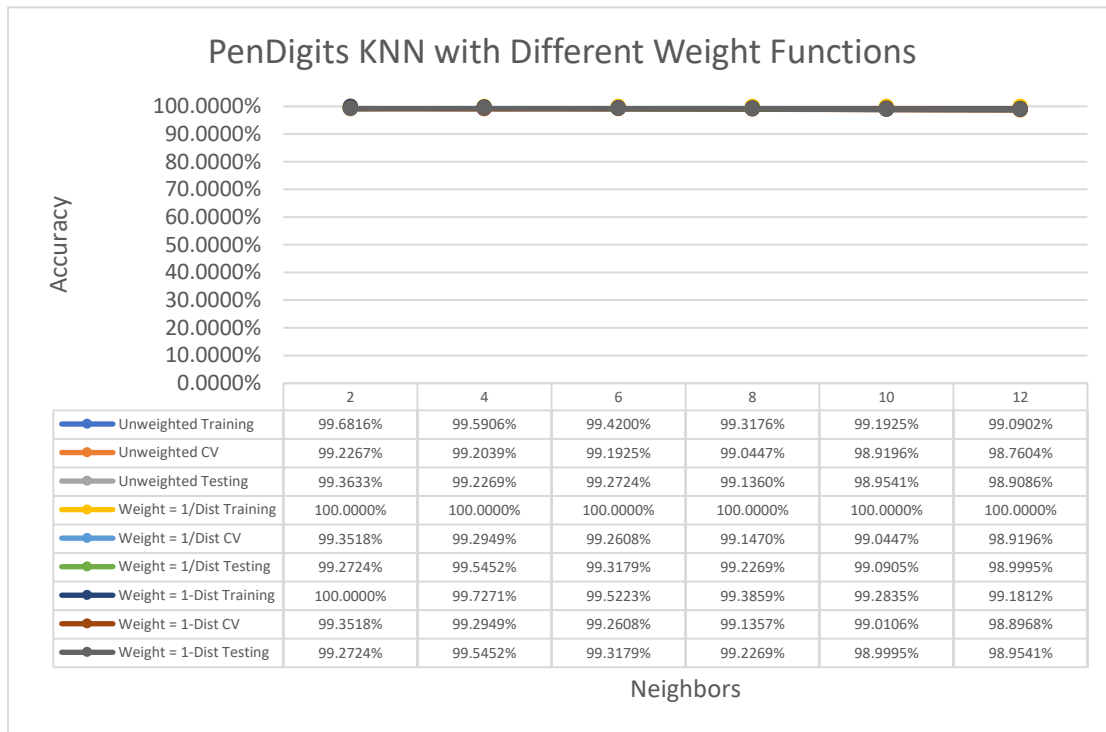


For the *Vehicle* dataset, the polynomial kernel performed very well, with high training, cross-validation, and testing accuracies. Overfitting was still a problem with the *Vehicle* dataset, as testing accuracy decreases with increased degree values, but the difference between training and testing accuracies was much less than compared to the other algorithms used earlier (decision tree, neural nets, and boosting). The RBF kernel performance was very poor, similar to that of the *PenDigits* dataset. Combined with the overfitting problem present in the dataset along with the suboptimal gamma values, it is not that farfetched to see why we got such poor performance. Overall, SVM analysis on the *Vehicle* dataset performed better than boosting and decision trees if we use the polynomial kernel. With respect to neural nets, the SVM algorithm performed similarly, but with a longer algorithm runtime.

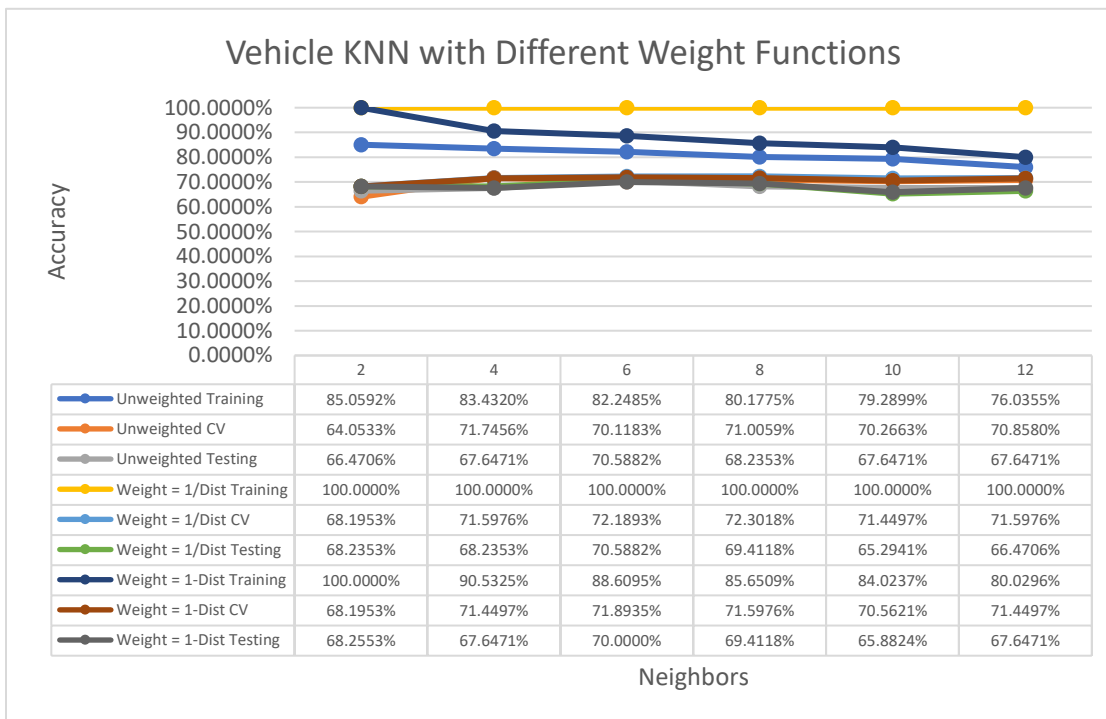
K-Nearest Neighbors Analysis:

Overall, I ran three experiments for each of the two datasets, each with a different neighbor weight function (no weights, weight = $1 / \text{dist}$, and weight = $1 - \text{dist}$) and with varied number of nearest neighbors (2, 4, 6, 8, 10, and 12).

Referring to the learning curves on the next page, the KNN algorithm performed very well on the *PenDigits* dataset. Comparing the three different weighting metrics, the $1 / \text{distance}$ weight function performed the best, followed by the $1 - \text{distance}$, and then the unweighted metric. There was very little difference training and testing accuracies for small number of neighbors, but that increased as the number of neighbors increased. This may be due to error brought in by farther away neighbors in the instance-attribute space. Compared to the other algorithms, KNN performed better than neural nets, decision trees, and boosting, and roughly on the same par as the SVM using the polynomial kernel.



Note: Analysis for the *PenDigits* KNN algorithm is on the previous page.



Overall, the KNN algorithm performed decent on the *Vehicle* dataset. Comparing the three different weighting metrics, the 1 / distance weight function performed the best, followed by the 1 – distance, and then the unweighted metric. Similar to the other algorithms with respect to the *Vehicle* dataset, there were differences between the training accuracy and both the cross-validation and testing accuracy. Overall, the testing accuracy decreased regardless of the

weighting metric. This may be due to error brought in by farther away neighbors in the instance-attribute space. One interesting trend present in the learning curve above is higher testing accuracy when using six neighbors. This may just be due to the instances available in the dataset, but it allows for a baseline for further optimization if one was to use KNN in a real-world setting. The KNN algorithm did not perform as well as the polynomial SVM or the neural net algorithm, but it was on par with the boosting and decision tree implementations.

Conclusion:

With respect to the *PenDigits* dataset, the KNN, Neural Net, and polynomial kernel SVM algorithms performed better than the Decision Tree and Boosting algorithms (though both decision tree and boosting did a pretty good job at successful classification). The large number of instances and relatively small number of attributes per instance played a huge role in preventing overfitting and thus resulted in similar training, cross-validation, and testing accuracies.

With respect to the *Vehicle* dataset, the Neural Net and polynomial kernel SVM algorithms performed better than the KNN, Decision Tree, and Boosting algorithms. The small number of instances and also small number of attributes per instance allowed for overfitting to play a huge role resulting in poor generalization between training and testing. However, cross-validation accuracies were often very similar to testing accuracies and provided a better estimator of algorithm performance than the training set accuracies.

Going back to my initial predictions, I was correct in stating that the neural net algorithm would perform the best with both datasets, primarily because of its ability to create new representation of the complex data in each of the hidden layers until it is able to effectively classify instances into appropriate classes. However, I was also impressed by the performance of the KNN algorithm (as it is thought to be a “lazy” algorithm) and the Boosting algorithm (as it is able to take sub-“classifiers” and apply weights to create a better approximator). Overall, I enjoyed implementing and analyzing these algorithms and was able to get a deeper understanding of how different hyperparameters influenced different algorithms.