Peter Cornelius
pcornelius7
CS7641

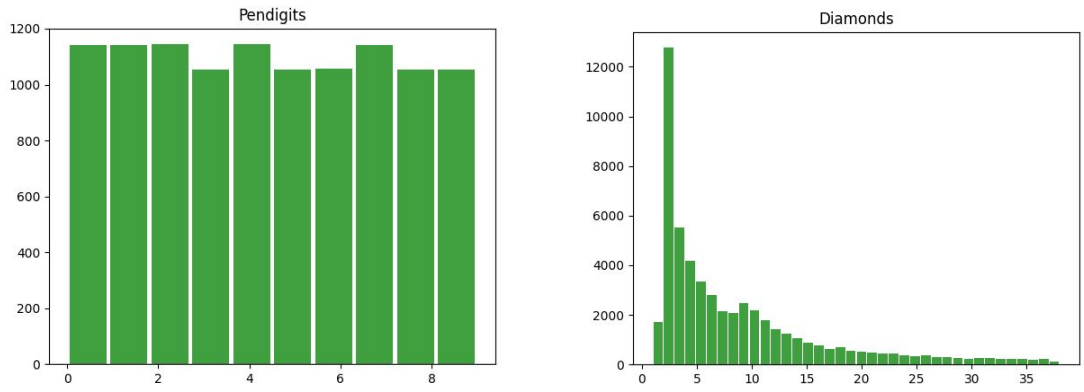
Supervised Learning Assignment 1

Datasets

Two datasets were selected for this assignment based largely on how they performed with decision trees, which was the first algorithm attempted. I chose one dataset that seemed to work easily and one that did not, hoping this would be an indicator of how different they were.

Pendigits is a collection of coordinates captured while volunteers wrote the digits “0” through “9” on a tablet. There are 16 features for the X and Y coordinates captured over approximately 1 second. Something potentially interesting about this dataset is that the order of these features is significant. Each pair of columns is an X, Y coordinate and the order of the pairs is sequential in time. All features have the same range of values because they are restricted to the positions within the test area of the tablet, so no normalization is required. The dataset is fairly balanced and had accuracy scores in the mid 90s with a standard decision tree.

Diamonds dataset contains descriptive characteristics of diamonds and the price they were sold for. This data set required some pre-processing because a number of the features used strings as category descriptors. For example, “Ideal” or “Good” to describe the cut. All of these columns also had a qualitative meaning so I did not need to add extra columns with one hot encoding. Instead I assigned meaningful numeric values, giving the “Ideal” cut the highest value. I also converted the continuous “price” label into buckets to frame my experiments around classification instead of regression. This was done by grouping price in $500 ranges and assigning increasing integers, 1-38, for the categories. Understandably, this dataset is not balanced, with far fewer expensive diamonds than there are inexpensive



Distribution of categories to samples in the Pendigits and CCDefaults datasets

Decision Trees

I chose to implement my decision tree experiment with scikit-learn to leverage the built in grid search and model selection tools and help perform my analysis. The first pass was to create a "dumb" tree with a max_depth of 5, which was chosen because it was larger than 1 but small enough to run quickly. The score from this was considered a benchmark that I hoped to improve on through hyper-parameter tuning and pruning. The random state was locked at "42" for the usual reasons.

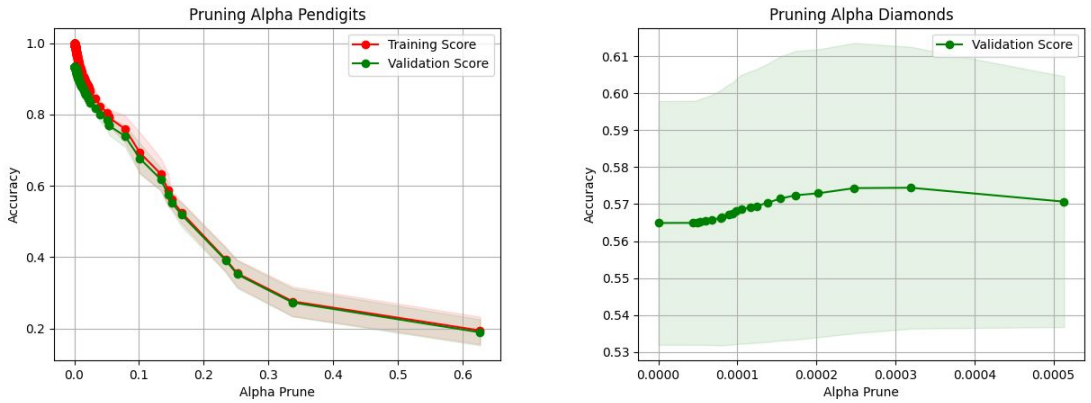
Dataset	Initial Score	Final Score	Final Parameters
Pendigits	0.8231	0.9654	'criterion': 'entropy'' 'max_depth': 15 'ccp_alpha': 0.0
Diamonds	0.4676	0.6113	'criterion': 'entropy' 'max_depth': 24 'ccp_alpha': 0.0003194796193862388

Decision Tree Benchmark and Final Accuracy Scores

Tree Depth

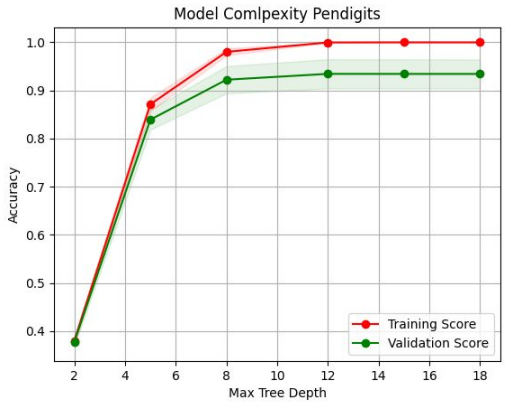
I was surprised to find that post pruning did nothing to improve my tree performance on the either data set at first. After inspecting the trees I found that I could improve my results for the diamond dataset by relaxing the "max_depth" parameter, allowing more overfitting in my base tree. I then used the built in cost complexity pruning from scikit-learn to remove nodes, increasing the impurity of leaves and reducing the overfitting. Using this technique I was able to improve the accuracy on the validation set to ~0.58 and on the test set to approximately 0.61.

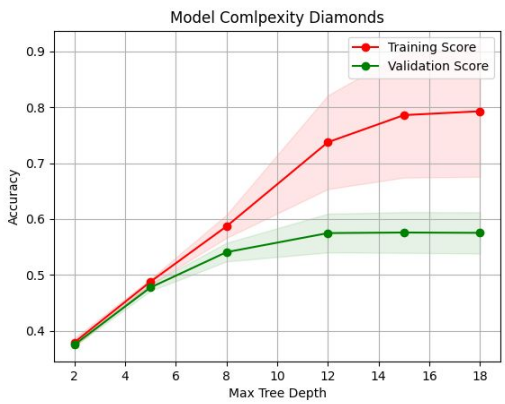
Interestingly this did not help the Pendigits dataset. I think that because this dataset has minimal noise, even if we overfit the training set it will be a close representation of the validation and test sets.



Tree Pruning effects on Accuracy

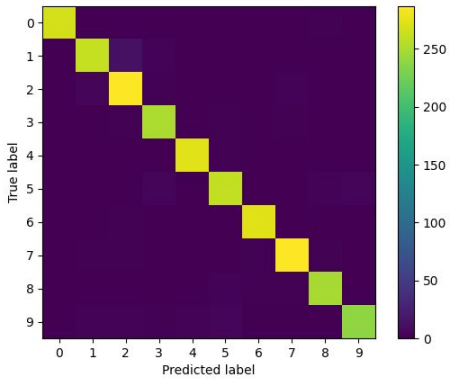
In the Pendigits graph you can see that both the validation and training scores fell immediately with any pruning but with the Diamonds dataset I was able to see some improvement.





Model Accuracy vs Tree Depth

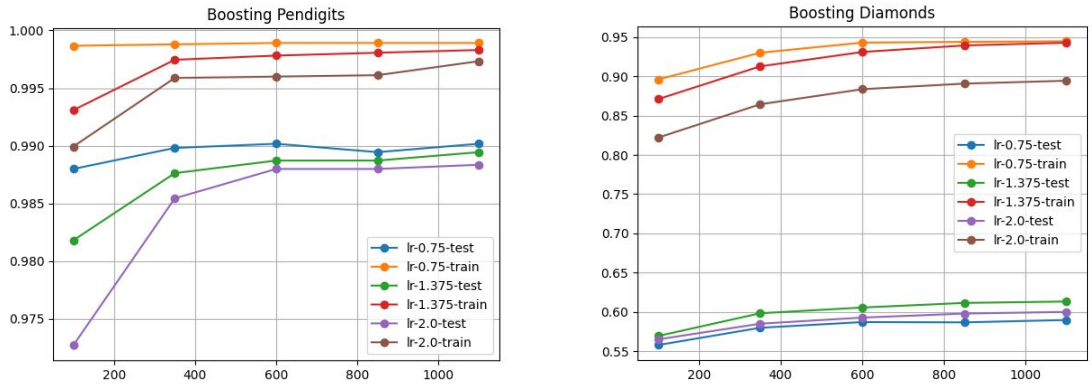
I also ran experiments without pruning but just increasing the tree depth and capturing the related scores. I found that after a certain point the “max_depth” setting no longer had an impact because the tree was as complete as it could be. Both of these data sets seemet to plateau in both training and validation quickly without dropping in accuracy. I expected to eventually overfit my training data to the point that the validation set would lose accuracy but that did not happen. With the pendigits dataset I thought it might just be a very complete and consistent dataset but the Diamond dataset does not score well as it is, telling me there is plenty of noise to overfit too. I suspect there is something in the sklearn implementation that I did not de-tune enough to overfit.



Pendigits Final Confusion matrix

Boosting

My initial attempts at AdaBoost were disappointing. I began with the same trees that I had tuned above and used an ensemble of up to 10 of them. For both datasets they performed noticeably worse, dropping my accuracy scores to ~94% for pendigits and ~52% for diamonds. To address this weakened the base classifier by decreasing the tree depth to be smaller than optimal. I then ran an experiment where I tried several learning rates and dramatically increased the number of base classifiers used from 10 to several hundred.



Boosting impact on accuracy (y axis) with varying learning rate and classifier count (x axis)

While the improvements were not quite as dramatic as I had hoped they were noticeable. With the best combinations found the diamond dataset accuracy improved very slightly but the pendigit accuracy against the test set approached 99%.

Dataset	Settings	Test Accuracy	Training Accuracy
Pendigits	LR: 0.75 Count: 600	0.9902	0.9989
Diamonds	LR: 1.375 Count: 850	0.6114	0.9394

Boosting Final Accuracy Scores

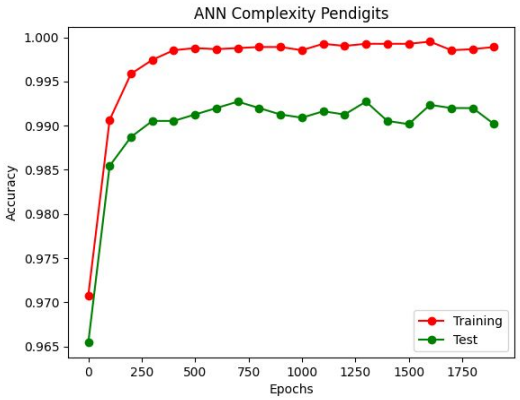
I suspect that with further experiments and tuning these scores could be improved. As expected, the test scores did not drop dramatically with the addition of more weak classifiers though the rate of increase was getting quite small.

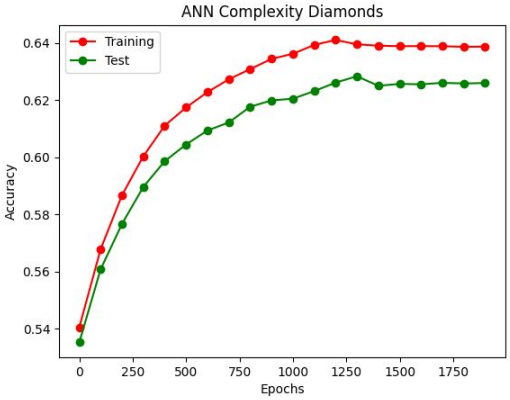
I also think that the number of categories may make boosting more challenging. It seems that if very simple “stump” trees are used the number would need to be in some way a function of the number of categories.

ANN

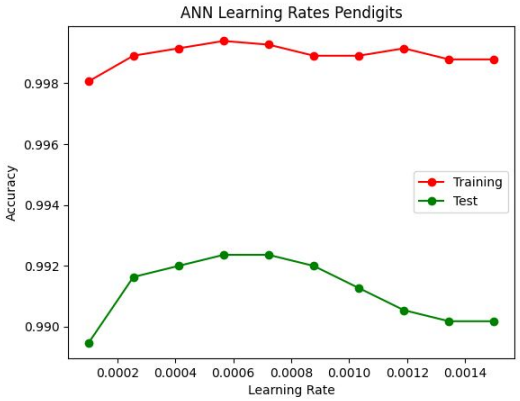
With Artificial Neural Networks I was able to explore the impacts of extending the number of training epochs, different activation functions, and changes to the nodes within the hidden layer. I constrained the hidden layer changes to a single hidden layer after some initial experiments showed no accuracy improvement from random additional layers but the time to train was significantly longer.

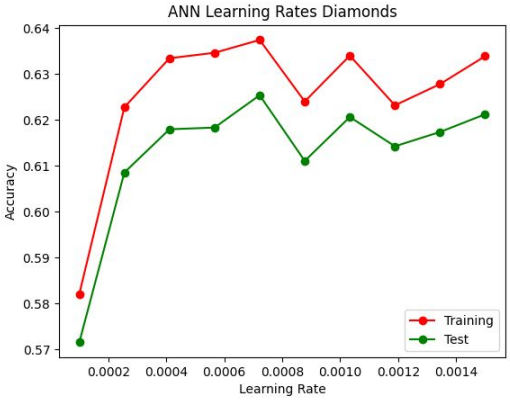
Both datasets reached a point of diminishing improvement from repeated training epochs but without much overfitting. The pendigits test scores were a bit unstable but the variation was less than 1%. The diamonds dataset seemed to converge quickly and not overfit.



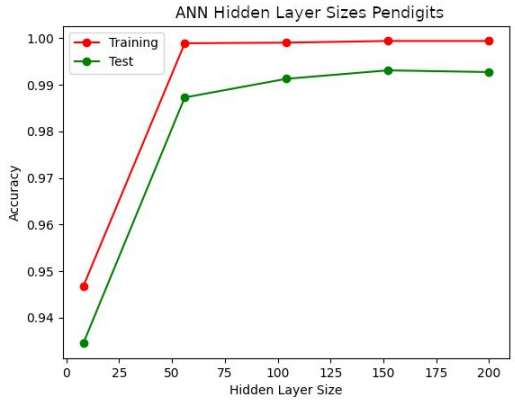


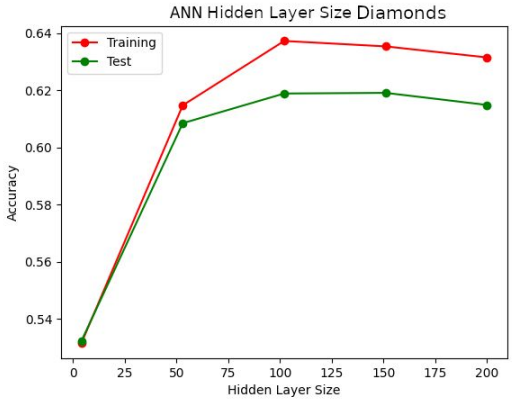
I also tried a range of values for the learning rate but did not see much reliable change there except the obvious issue of values that were too small. I was wondering if the complexity of these functions was producing local minima that an increased learning rate would improve but that did not seem to be the case.



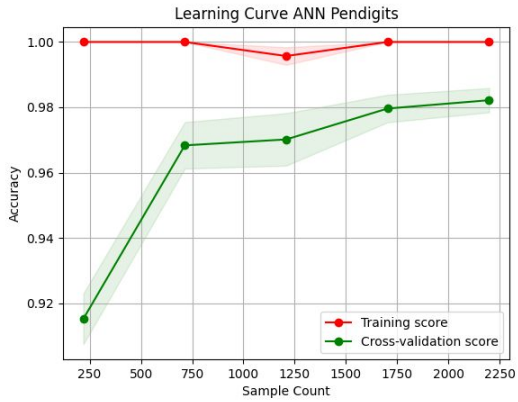


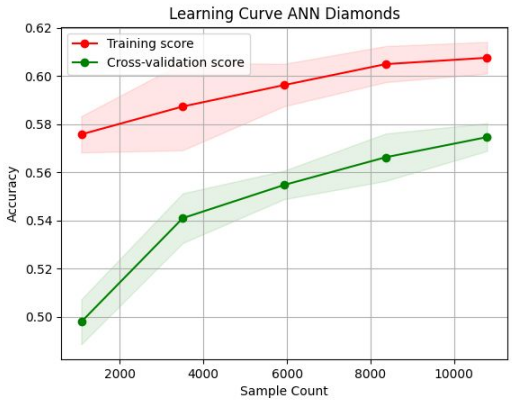
Finally, I explored changes to the hidden layers. I expected for larger hidden layers to improve accuracy because it would allow for more nuanced weighting of input combinations. However, it seemed that, for these datasets, it made very little difference as long as it was “big enough”. Adding additional layers also did not seem to make any improvement.





Learning curve for the neural networks seemed to be trending towards a proper fit though the compute times made it difficult to get enough datapoints. However, I do think it’s clear that the training and test curves are not diverging and the gap in accuracy is being reduced.

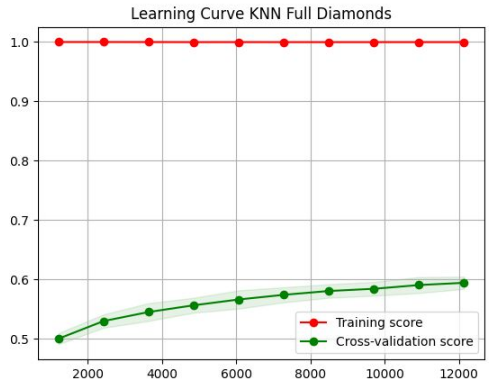




KNN

Exploring the KNN algorithm uncovered a few unexpected optimizations that might have helped other algorithms as well. Like the other algorithms the default version had disappointing performance. The diamonds dataset was below 53% though the pendigits seemed to immediately hit 99%. I explored several of the hyperparameters, including the distance metric which provided a boost for the diamond dataset. This dataset is mostly integer values so I selected some metrics designed for integer based data and got a bump to 62%.

But looking at the learning curve it seemed to me that the training set was perfect but the training score was not quite hitting its limit. This smelled a bit like The Curse of Dimensionality where the data I had was not sufficient for the number of columns. Even though the diamond dataset doesn’t have as excessive number I looked at



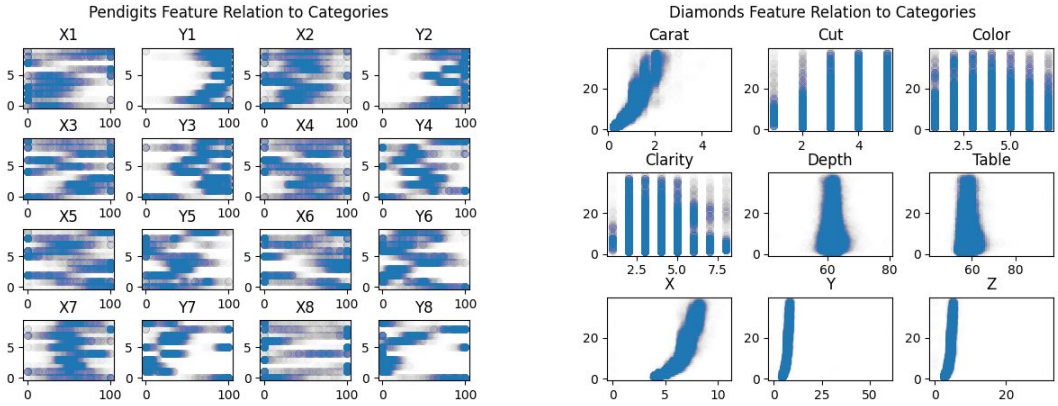
what happened when some were removed and did see a modest improvement:

Pendigits Accuracy Scores		Diamond Accuracy Scores	
All Columns:	0.9909024745269287	All Columns:	0.6253615127919911
-X1:	0.990174672489083	-carat:	0.6065257693733779
-Y1:	0.9879912663755459	-cut:	0.6327030033370411
-X2:	0.9883551673944687	-color :	0.5174638487208009
-Y2:	0.9894468704512372	-clarity:	0.46955876900259547
-X3:	0.9894468704512372	-depth:	0.6271412680756396
-Y3:	0.9879912663755459	-table:	0.6298850574712643
-X4:	0.9890829694323144	-x:	0.6240266963292548
-Y4:	0.9894468704512372	-y:	0.6236559139784946
-X5:	0.992721979621543	-z:	0.624842417500927
-Y5:	0.9887190684133915		
-X6:	0.9898107714701602		
-Y6:	0.9905385735080058		
-X7:	0.9909024745269287		
-Y7:	0.9898107714701602		
-X8:	0.9879912663755459		
-Y8:	0.9905385735080058		

Impact of removing individual columns before KNN training

Dataset	Initial Score	Improved Metric	Removed Columns
Pendigits	0.5279	0.6254	0.6341
Diamonds	0.9909	0.9909	0.9927

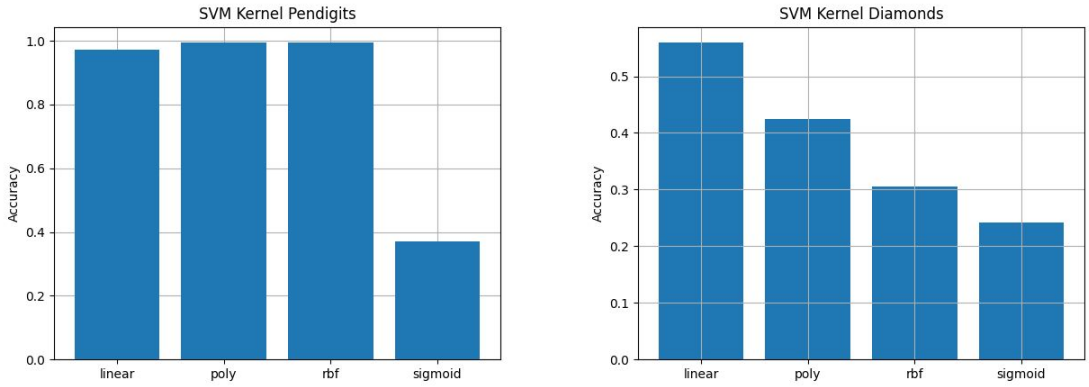
Something surprising was that the columns removed were not the ones I had initially guessed. For the diamonds dataset I assumed the “carat”, “cut”, and “clarity” would be the most significant columns but it turned out that removing the “cut” column from the data before training gave a slight improvement. Also, for Pendigits, I would have expected all the columns to be important because they are X,Y pixel locations. However, oddly, removing the 5th X column also gave a slight improvement in the accuracy on the test set. This led to some investigation and visualization of the relationship of the individual features to the labels.



Relation of Features to Categories

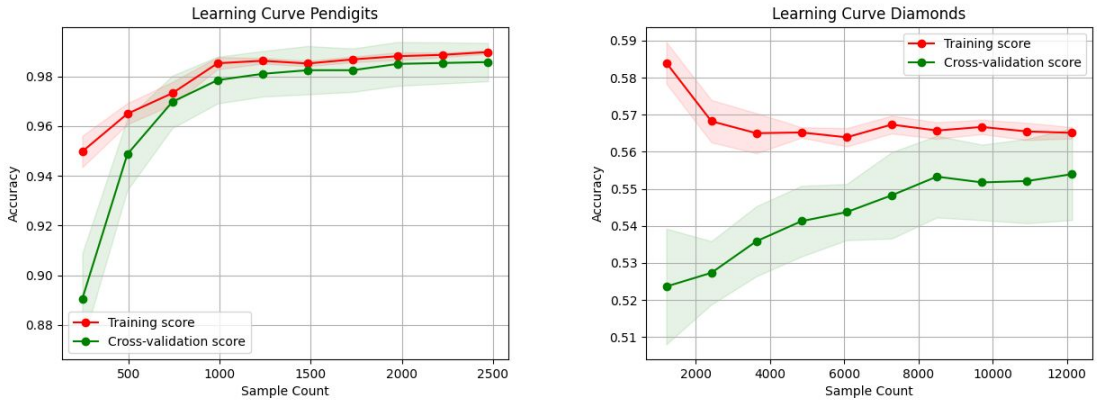
SVM

I used the SVC (support vector classifier) from the SVM module in scikit-learn. I was able to try several kernels on each dataset to find the one that best fit. For Pendigits it did not seem to make much of a difference, probably because the data is not too noisy and is very well balanced so overfitting and underrepresenting the data is not much of an issue. However, the diamond dataset clearly struggled with kernels other than the “linear”,



Accuracy for Different Kernels

Learning curve for the Pendigits data set seems to show low variance, because the validation and training curves are converging, but also low bias because the training scores are staying high. In contrast the Diamond dataset is struggling with both higher variance and bias, as the training scores fall and validation scores are not able to converge.



References

Machine Learning by Tom Mitchell - http://www.cs.cmu.edu/~tom/mlbook.html

Code was re-used freely from various scikit-learn examples found at
https://scikit-learn.org/stable/auto_examples/index.html

MatplotLib code was often copied and modified from
https://matplotlib.org/stable/tutorials/introductory/sample_plots.html

I found the following post on interpreting learning curves helpful
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/