Dataset Description:

For each algorithm, I used the digits dataset from scikit-learn, which contained 1797 samples. The dataset represents written digits as an array of size 64, where the values in the array range from 0 to 16 (light to dark), and each value represents a pixel of an 8x8 image of the written digit. Each pixel is feature, so in total there are 64 features. There are 10 classes, which are the base-10 digits. Additionally, there is an array that has the correct label for each datum, which is the target array.

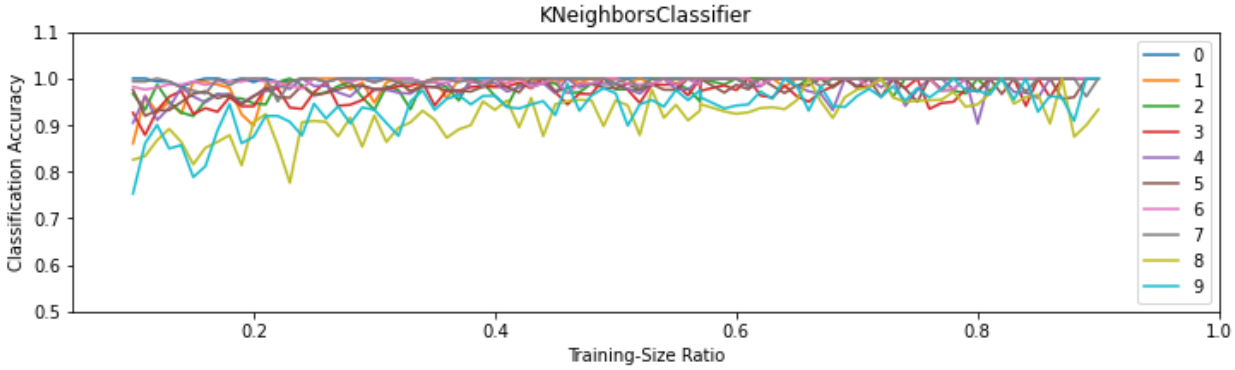
Classifiers:

For the classifiers, I followed the book and tested each against differing training-testing ratios. The training size starts at 10% of the data, and increments by 1% each test up to 90% of the data.

K-Nearest Neighbors:

The main way this method of classification works is by calculating the Euclidian distance between a point and all the points around it (up to k points), and after that is done find the class of points with the greatest frequency, class will then be assigned to the point. In context of the digits dataset, the method would take the pixel values and compare them with the training data, and based off of the results it would assign it to a digit (i.e., a new image has pixel values that match most with pixel values corresponding to the number 5, so the new image is classified as a 5).

Accuracy results:



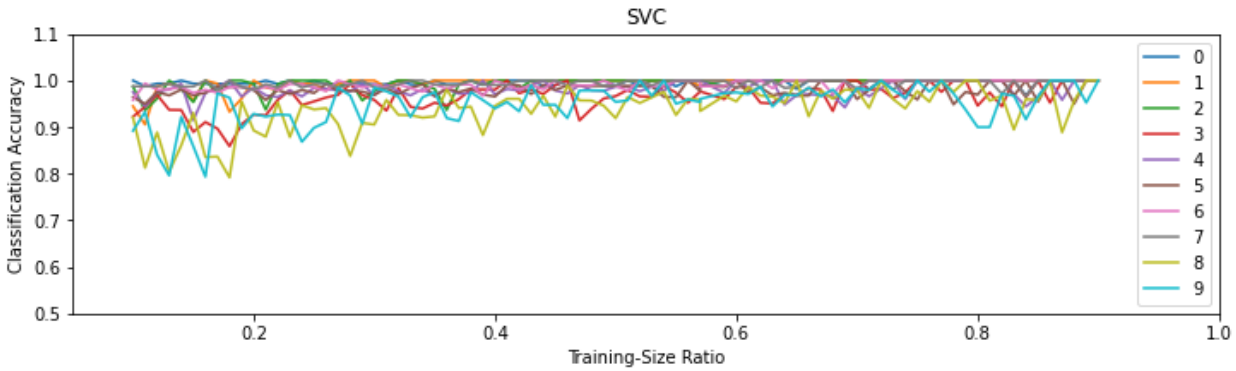
The above graph displays the classification accuracy of each digit using different training sizes. Looking at the accuracy graph for digit 9, we can see that with a lower amount of training data, the accuracy was lower when compared to when we had more training data, or even a moderate amount of training data. The 10% training data had a prediction accuracy of about 75%, whereas the 90% training data had an accuracy of 100%. This graph shows that with more training data, the K-Nearest Neighbors becomes more accurate, however, I have observed in other tests where the greater the percentage of training data (>80%) the less accurate the prediction. In this graph, that result can be seen from the digit 8 results, where there is greater accuracy from 60% - 80%.

SVC:

The way support vector classification (SVC) works is by partitioning an n-dimensional space with a hyperplane(s). SVC splits up this n-d space such that the distance margin is maximized. Effectively, it is

like drawing a line between two group to separate them, and the line is aligned in such a way that the distance from the nearest of each group is maximized. For this dataset, it will be similar how k-neighbors functioned, except instead of finding distances, it will check which hyperspace the pixel values are in and assign it to the class of that hyperspace.

Accuracy results:

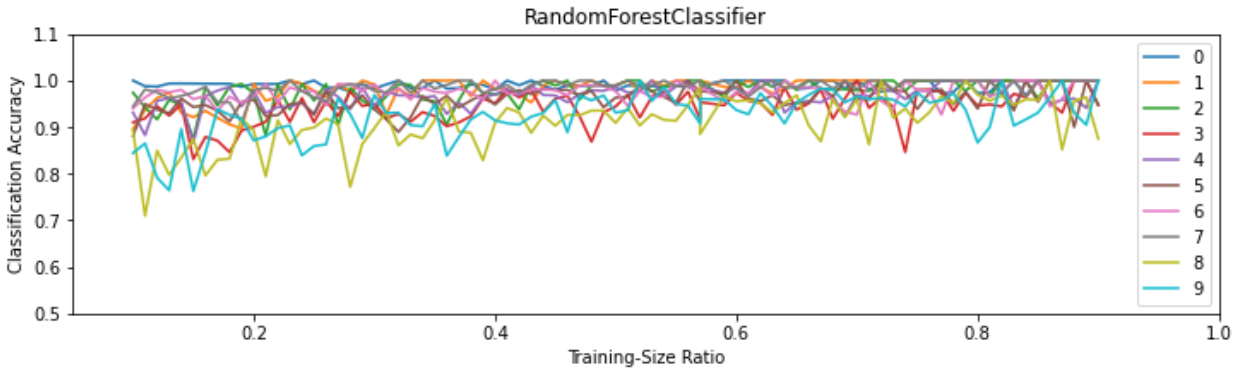


From the graph above, we can see the accuracy results are similar to that of k-neighbors. The lower the percentage of training data, the lower the accuracy. However, while there is some observed decrease in accuracy with a higher percentage of training data, overall, as training data size increased, the greater the accuracy, where at 90% every test was correctly identified. Another reason for this could also be attributed to a decreased test data size at 90%, where in the lower training data sizes, there is an increased amount of testing data compared to 90%. So, the model has less data to go off of, and has more data that it needs to get correct to have a higher accuracy (i.e., at 10% training, there is 90% testing, and vice versa at 90% training).

Random Forest:

A random forest classifier works by using decision trees. Like the name suggests, the classifier uses a number of uncorrelated decision trees to predict the answer. In the case of this dataset, the random forest would run the image data through the decision trees and each tree would give an answer, and the majority would win. So, if I had five trees and the trees answered 5, 4, 6, 5, 5, then the random forest model would predict a 5 as the answer for the input.

Accuracy results:



From the results in this graph, we can see that they were the most inaccurate when compared to the other two classifiers. Where, the low was ~70% accuracy for digit 8 at 10% training data, and a lot of the digits weren't guessed with 100% accuracy for a multitude of tests, whereas in the others, in many of the tests many of the digits were correctly guessed with 100% accuracy, or there was a higher concentration of results around 100% accuracy compared to random forest. I believe, like the other classifiers, and as I mentioned in SVC, random forest suffers from finding a balance between training and testing sizes, however, it isn't as apparent and may not be as influential in this case since the accuracies tended to fluctuate a lot as training size increased. When looking at the first two classifiers, we can see this general trend from low to higher as training size increased, but with random forest that trend was flattened a bit due to the fluctuations in the data.

Clustering:

K-Means:

K-Means clustering works by using k centroids and creating k clusters based around these centroids. It holds similar ideas to how k-neighbors works, but with this we randomly choose starting points for the centroids, and then calculate the Euclidean distance between the centroids and the data points. The data points are then assigned to a cluster based on the closest centroid to it. Then, for each cluster, the mean of the points in the cluster is calculated and new centroid is created with that mean value, and then the process is repeated until there are no changing values or until some threshold. For this dataset it works by clustering similar pixel data together, somewhat like SVC does by partitioning a space, so when an image is tested, if it has similar pixel data compared to a certain cluster, then it will be classified as that cluster's class. This also adds the need to normalize the data before/after predictions, so that the data is interpretable to us (e.g., say 5 was used as the cluster class for 0, then change cluster class to 0, or predictions for cluster 5 to 0).

Confusion Matrix:

		Predicted	→									Accuracy
		0	1	2	3	4	5	6	7	8	9	
Actual	0	177	0	0	0	1	0	0	0	0	0	99.44%
↓	1	0	55	24	1	0	1	2	0	99	0	30.22%
	2	1	2	148	13	0	0	0	3	8	2	83.62%
	3	0	0	1	157	0	2	0	7	7	9	85.79%
	4	0	7	0	0	162	0	0	9	3	0	89.50%
	5	0	0	0	1	2	136	1	0	0	42	74.73%
	6	1	1	0	0	0	0	177	0	2	0	97.79%
	7	0	2	0	0	0	1	0	174	2	0	97.21%
	8	0	6	3	4	0	4	2	5	102	48	58.62%
	9	0	20	0	7	0	6	0	7	2	138	76.67%

The confusion matrix details the accuracies of the k-means method. We can readily see that, for predicting 0, the model correctly guessed 177/178 times, as displayed by the 177 at (0,0) where there is an additional 1 at (0,4). In the right-most column, we can see the accuracies for this method. Overall, the accuracy is good, however, for a number like 1, it easily got confused with the number 8 using this

method. A reason to explain this discrepancy is that the pixel pattern of an 8x8 representation of 1 and 8 resemble each other, thus the model grouped them together, or close to one another. A similar issue with a similar explanation can be seen with number 9, where 5 and 8 were confused for 9 around 40-50 times each.