* **ratio of training data vs accuracy**
* **test train ratio**
* **which algorithm outperformed**
* **train times?**
* **what was the general trend**
* **same on any other dataset?**

**Analysis**

We would be comparing various algorithms by training their models on the MNIST dataset. The metric to compare these algorithms would be their accuracies in predicting the test data.

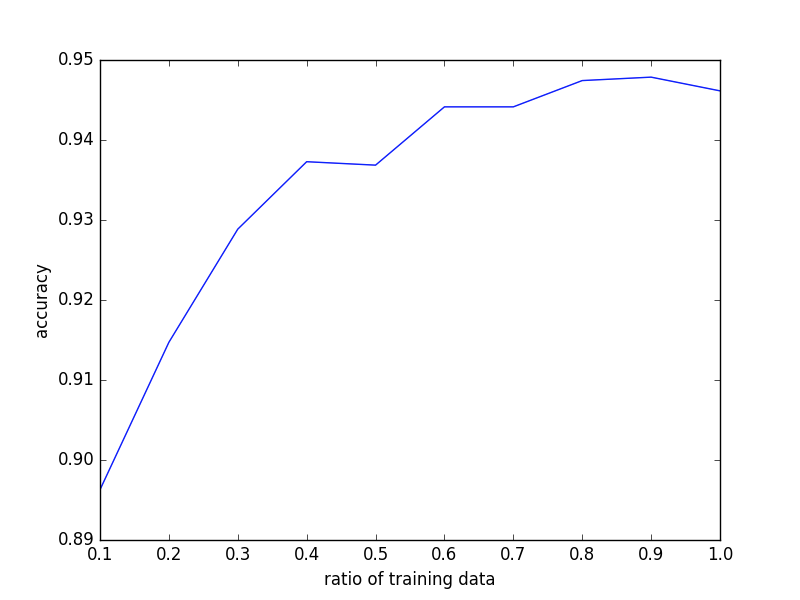
Different algorithms learn at a different rate. To analyse this trend, we would be testing these algorithms with varying amounts of training data. Our training data would range from 10% to 100% of the available training data with increments of 10%. To visualize this better, we would plot the accuracies of the model for each increment of the training data.

**Comparison of Implemented Algorithms**

For our testing purposes, we compared and analysed four main algorithms:

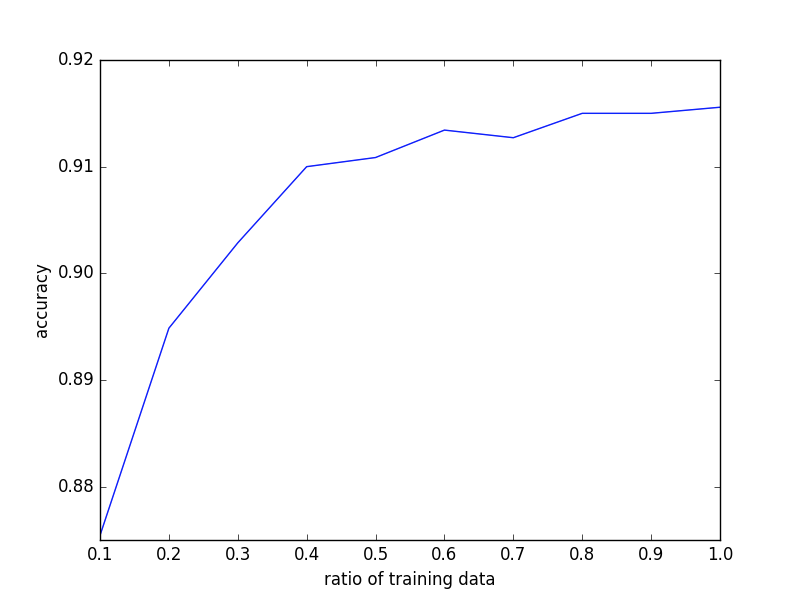
**Random Forest Algorithm:**

Random Forest algorithm is versatile and require very little modifications. The run time of random forest algorithm was the lowest among all the implemented algorithms. The algorithm took \_\_\_\_ seconds to train on 63000 training entries. In general, the rate of change of accuracy was proportional to the change in training data. The change of this rate decreased with increase in the training data. Mathematically, dA / dD = +ve while d2A / dD2 = -ve. Random forest was able to give an accuracy of \_\_\_ when trained using 100% of  the training dataset.



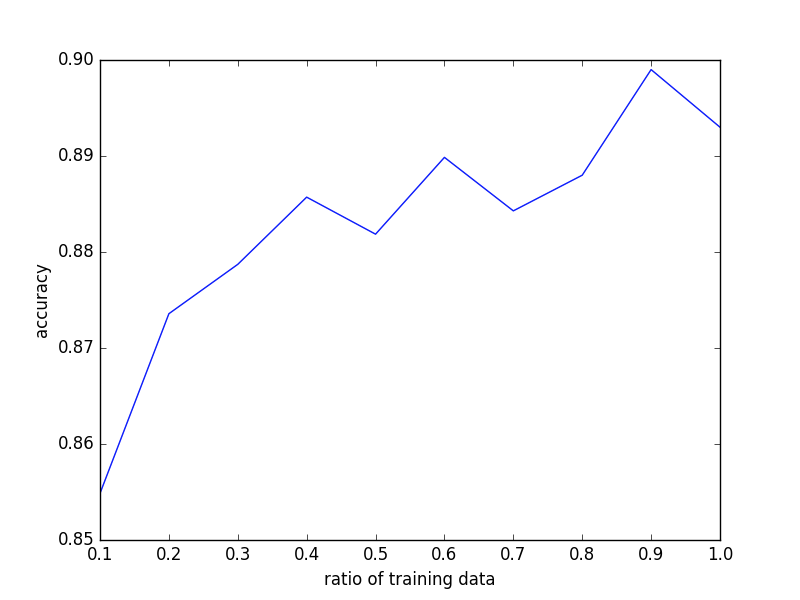
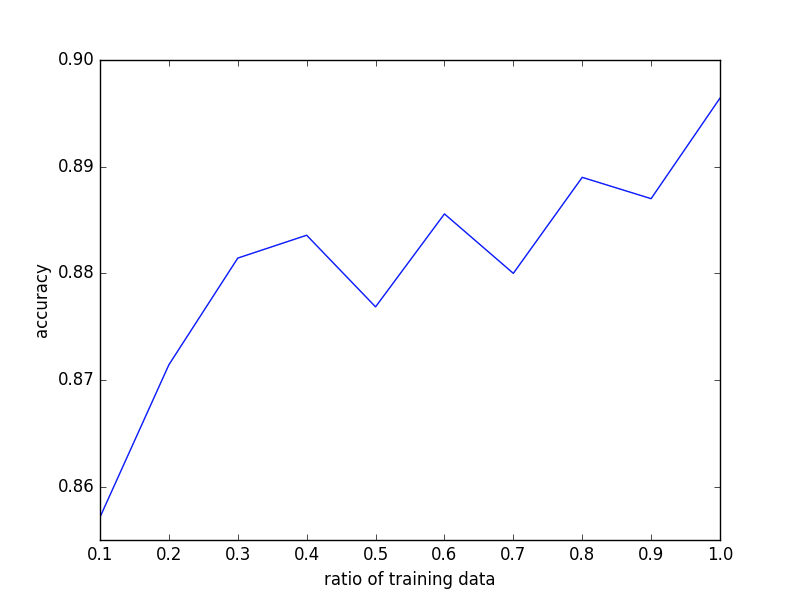
**Support Vector Machine**

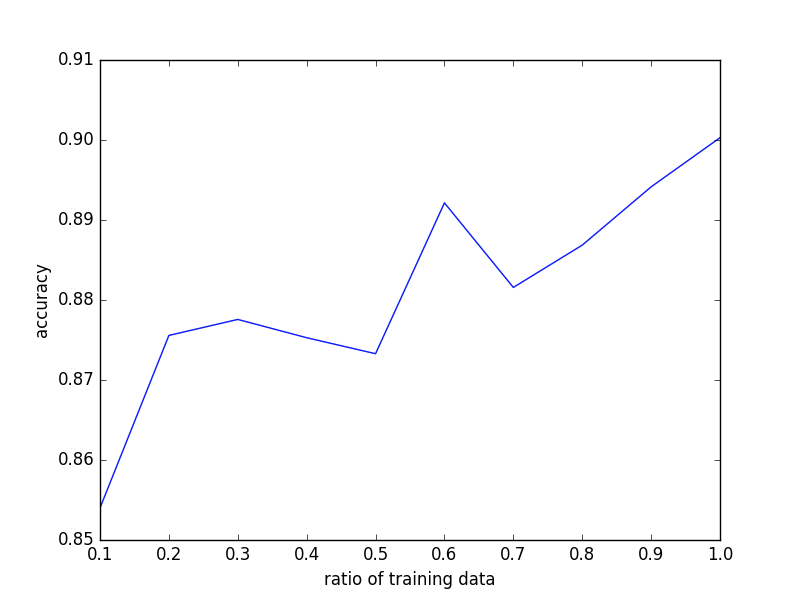
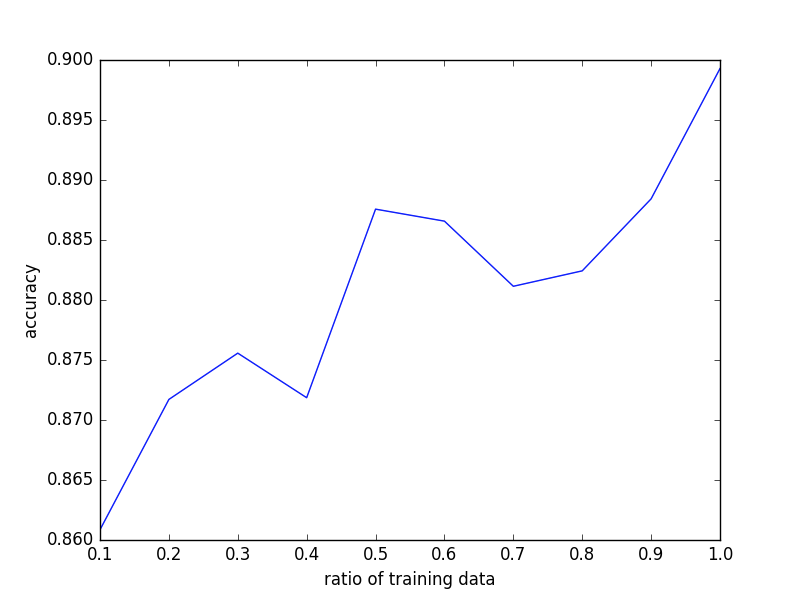
Support Vector Machine is a non stochastic algorithm which has comparatively slower run time as compared to Random Forest . In our implementation, we have used the Linear SVM Classifiers due to their low time and space complexity when to other Non Linear SVM Classifiers. Similar to Random Forest, the rate of change of accuracy was proportional to the change in training data. The change of this rate decreased with increase in the training data. Mathematically, dA / dD = +ve while d2A / dD2 = -ve. Support Vector Machine was able to give an accuracy of\_\_\_ when trained using 100% of  the training dataset. The algorithm took \_\_\_ seconds to train on 63000 training entries.



**Stochastic Gradient Descent**

The next algorithm that we implemented was stochastic gradient descent. It is widely popular for solving large scale learning problems and is known to work efficiently. Stochastic Gradient Descent was able to give an accuracy of\_\_\_ when trained using 100% of  the training dataset. The algorithm took \_\_\_ seconds to train on 63000 training entries. On observing the graph, the algorithm did not show a smooth graph like the previous two. Instead, the accuracy of the model fluctuated with changes in the ratio of the training data. In spite of this fluctuation, the accuracy managed to increase with increase in training data. Due to the stochastic nature of the algorithm, we observed different graphs on different instances of testing the model.





**Artificial Neural Networks**

Neural Nets have been in trend lately. Most of the algorithms for solving learning problems employ some form of neural nets. We used a 3 layer Neural Net with an input, hidden and output layer. The input layer consisted of 784 nodes (28\*28), one for each feature (x,y). The hidden layer consisted of 300 nodes. Thus the input layer mapped 784 features to this 300 nodes. The output layer consisted of 10 nodes, one for each digit. Thus, a prediction of [0000010000] would mean a 5. The runtime of the neural net model was comparatively high but it attained unmatched accuracy. The neural net took \_\_ seconds to train on 63000 entries. It predicted the test data with an accuracy of \_\_ %.

