Analysis of Handwritten Digits Recognitions using SVM and ANN

Amritansh Tripathi

Northeastern University

tripathi.a@husky.neu.edu

**Bharat Vaidhyanathan**

Northeastern University

vaidhyanathan.b@husky.neu.edu

Abstract

Efficient and effective learning of handwritten digits has been an intensive area of research. The paper aims to analyze some of the studied algorithms using MNIST dataset. We separate our analysis in two parts, the first part will show comparison of accuracies using Support Vector Machine, Artificial Neural Network, Random Forest and Stochastic Gradient Descent while the second part will aim at analyzing the effect of feature descriptor on the accuracies of the applied algorithms. Our analysis on feature extraction will be based on histogram of oriented gradients (HOG) feature descriptor which is widely used in computer vision and image processing. The main observations in HOG feature descriptor analysis will be improved accuracies in all four algorithms and faster runtime in SVM and ANN due to reduction in number of features.

Introduction*[[1]](#footnote-1)*

Handwritten Digit Recognition has been subject of intensive research in the past decades. Initially, researchers were limited with memory and processing power but with advancement of technology it has greatly helped in the research of this field. Handwritten digits are a common part of everyday life. One of the trivial uses is in the US Postal Department which requires digitization of zip codes. The biggest challenge has been huge variation in writing styles of different people which creates a very noisy data to give results with 100 percent accuracy.

Though, there are many algorithms that are being used in study of this field but few have outperformed. Supported vector Machines, Artificial neural Networks, K-Nearest Neighbor and Random Forest have particularly given really good accuracies.

The testing samples can be generated offline by scanning the text images written by hand or it can be generated optically using pen based computer screen which is formally known as an online method.

Random Forest Algorithm

Random forest is a method for classification task that creates multitude of decision trees while training and the class which is most occurring of all the classes is the output. Trees which have large depth have a tendency to produce output which overfit. Random forest helps in averaging multiple decision trees which are trained on different parts of same dataset. This helps in reducing the high variance in output which is generally produced in decision trees. Some features of random forest are as follows:

* Excellent Accuracy
* Efficient on large datasets
* Estimates important variable in a classification
* Handle thousands of input variable without variable deletion

**Random forest Pseudocode**

A training set S := (x1, y1), . . . ,(xn, yn), features F, and number of trees in forest B.

1 function RandomForest(S , F)

2 H ← ∅

3 for i ∈ 1, . . . , B do

4 S (i) ← A bootstrap sample from S

5 hi ← RandomizedTreeLearn(S (i) , F)

6 H ← H ∪ {hi}

7 end for

8 return H

9 end function

10 function RandomizedTreeLearn(S , F)

11 At each node:

12 f ← very small subset of F

13 Split on best feature in f

14 return The learned tree

15 end function

We select an ith sample S(i) using which we learn a decision tree. The algorithm is modified such that, at each node of the tree we randomly select some subset features. The node is then split on best feature f where f<F. Since the number of features are reduced, it drastically increases the speed of computation.

**Stochastic Gradient Descent**

Stochastic gradient descent is a stochastic approximation of the Gradient Descent method to identify the global minimum by iterations. It is popularly used for large scale datasets. In Gradient Descent, the cost gradient is calculated based on the complete training dataset after each pass. *The cost function Q(w) can be written as Q(w)=*and the cost gradient can be written as *-ɳ∇Q­i(w)=- ɳ.* When there are large datasets, GD is very costly since we run through all samples in the training dataset to do a single update of the value. In GD, the weights are updated slower and it takes longer to converge to a global minimum cost due to its asymptotic rate of convergence. In stochastic Gradient Descent(SGD) however weights are computed using single sample hence is significantly faster and starts improving itself from the first sample.

**Stochastic Gradient Descent Pseudocode**

* Select an initial vector of parameters w and learning rate *ɳ*
* Repeat until an approximate minimum is obtained:
  + Randomly shuffle examples in the training set
  + For i=1,2,…….,*n*. do:
    - *w=w- ɳ∇Q­i(w)*

|  |  |
| --- | --- |
|  |  |

Here *Q­i(w) is the value of loss function*

**Support Vector Machine**

Support Vector Machines are supervised learning methods which are used for classification and regression analysis. They construct a single or a set of hyperplanes which can be used for classification. A data point in SVM is considered as a p-dimensional vector which can be separated using (p-1) dimensional hyperplane. Hyperplane which has maximum margin between two classes is generally chosen. If a set of data points are not linearly separable then they are mapped into higher dimensions which makes separation easier. This is known a kernel trick.

**Linear SVM**

Consider n points training dataset (x1,y1)…….(xn,yn) where y can 1 or -1 and each x is a p dimensional real vector. Any Hyperplane can be written as

Here, is the normal vector to the Hyperplane.

In order to identify the maximum-margin hyperplane, two hyperplane with maximum distance is identified which can separate two classes of data. Equations describing these hyperplanes are

Distance between these hyperplane is , hence inorder to maximize the distance .

**Nonlinear SVM**

Data sets that are not linearly separable we try to separate them by using kernel trick that converts them into higher dimensions as shown in Figure 1. It is comparatively easy to separate nonlinear data in higher dimensions.

**Artificial Neural Network**

**Dataset**

To test and train our model MNIST (Mixed National Institute of Standards and Technology database)

Dataset is being used. MNIST dataset contains 70000 images of numbers ranging from 0 to 9. It was developed by Yann LeCun, Corinna Cortes and Christopher Burges for evaluating machine learning models. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. For our implementation, the MNIST dataset was split into training dataset and testing dataset. 63000 entries were used for training dataset and the remaining 7000 entries were used for testing dataset

The data set is preprocessed using sklearn’s MinMaxScaler function in the preprocessing module which scales down the features to a range of values between 0 and 1. The scaling helps in creating a very small standard deviation between features and existing zero entries in a sparse data.

The formula used to scale the values is:

Figure :Analysis of Random Forest Algorithm

Where min and max are the range of feature values and X\_scaled is the new feature value generate from the function.

**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.

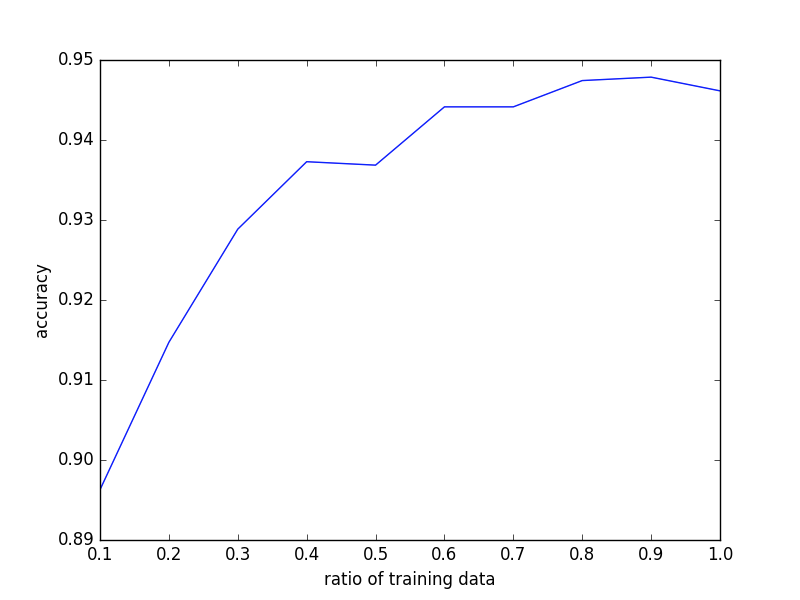
Different algorithms learn at a different rate. To analyze this trend, we would be testing these algorithms with varying amounts of training data. Our training data would range from 10% to 100% of the available training data with increments of 10%. To visualize this better, we would plot the accuracies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

**Comparison of Implemented Algorithms**

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

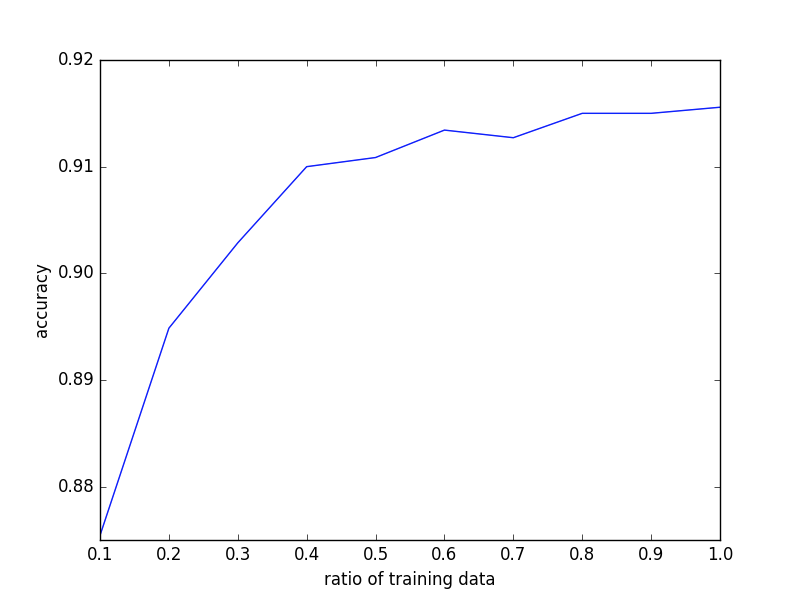
**Random Forest Algorithm:**

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. We have used the RandomForestClassifier module from sklearn’s (Scikit Learn) ensemble library to implement Random Forest in our analysis. 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 94.77% when trained using 100% of the training dataset.



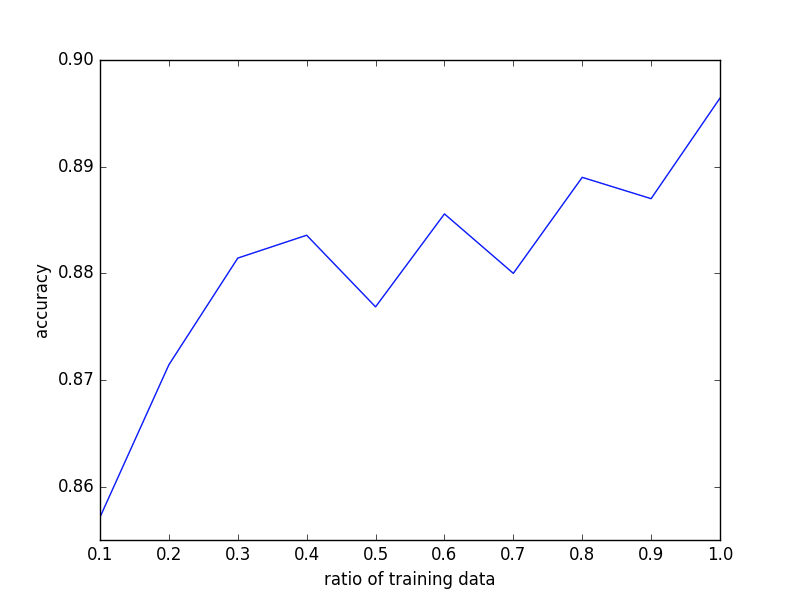
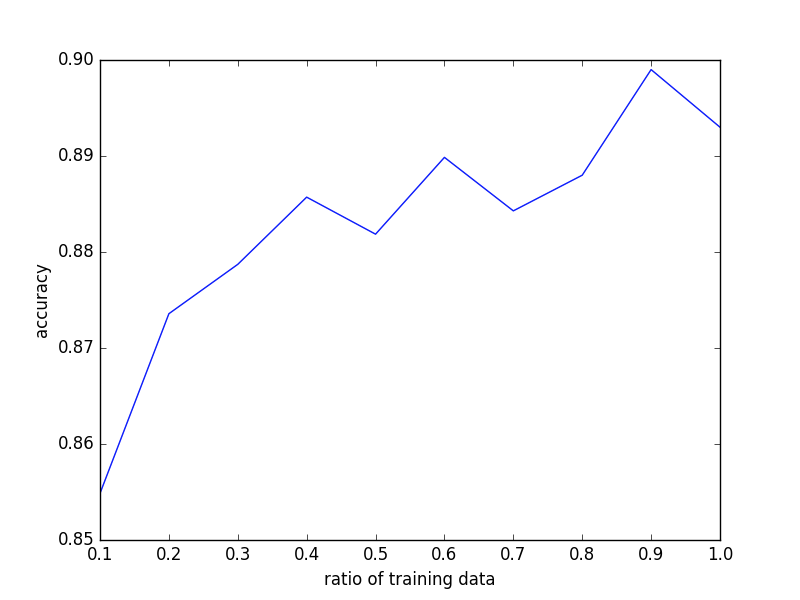
**Support Vector Machine**

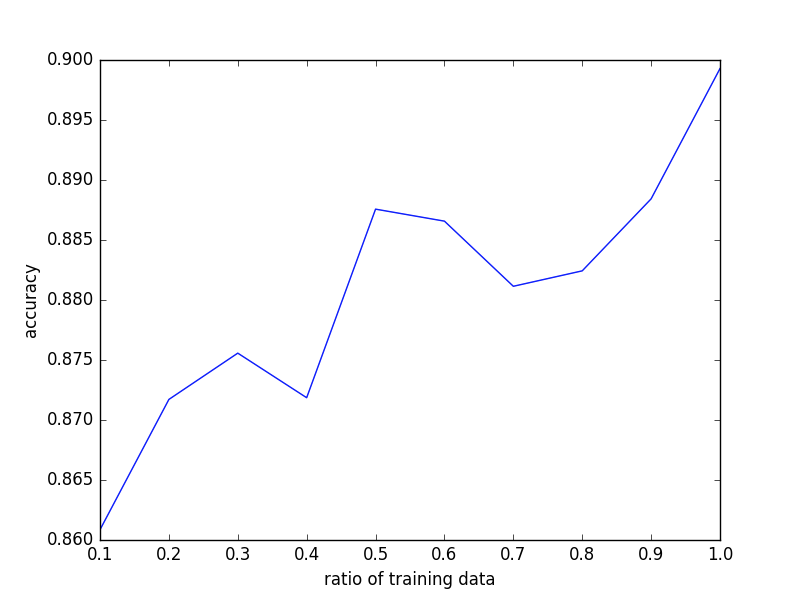
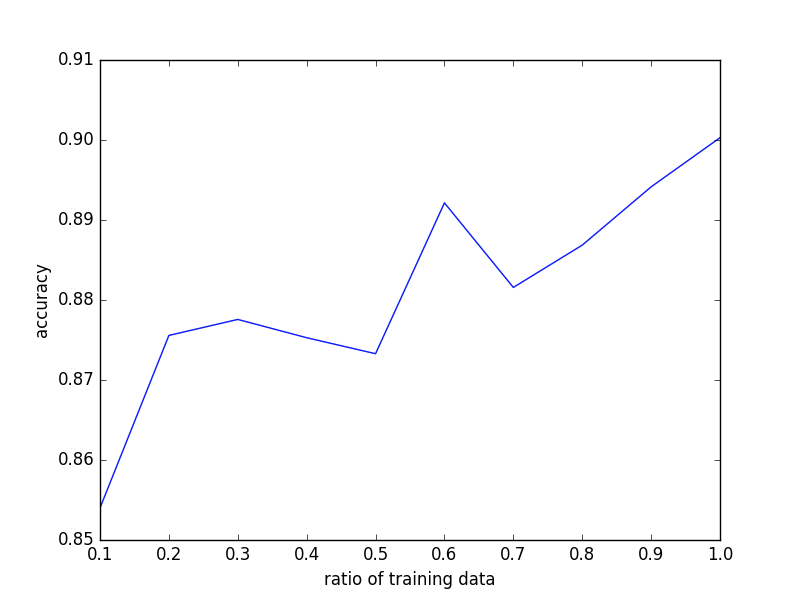
Support Vector Machine is a non-stochastic algorithm which has comparatively slower run time as compared to Random Forest . We have used the LinearSVC module from sklearn’s (Scikit Learn) SVM library to implement Support Vector Machine in our analysis. In our implementation, we have used the Linear SVM 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 91.45% when trained using 100% of  the training dataset. The algorithm took \_\_\_ seconds to train on 63000 training entries.



**Stochastic Gradient Descent**

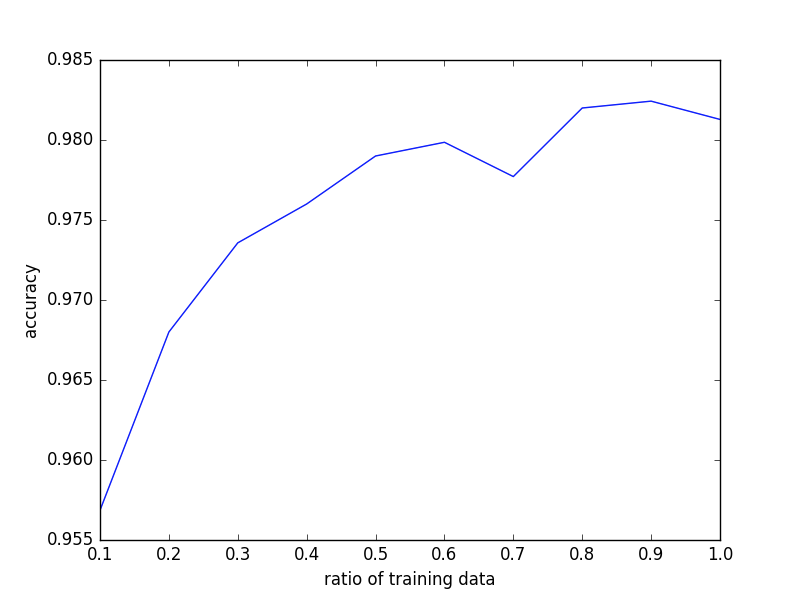
The next algorithm that we implemented was stochastic gradient descent. It is widely popular for solving large scale learning problems and is known to work efficiently. We have used the SGDClassifier module from sklearn’s (Scikit Learn) linear\_model library to implement Stochastic Gradient Descent in our analysis. Stochastic Gradient Descent was able to give an accuracy of 89.68% 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. 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 have used the DBN (Deep Belief Network) module from the nolearn library to implement Neural Nets in our analysis. 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 98.13%.



All fonts must be embedded for your paper to be published. No exceptions.

References

Engelmore, R., and Morgan, A. eds. 1986. *Blackboard Sys­tems.* Reading, Mass.: Addison-Wesley.

*(If you use EndNote and it complains "object has been deleted" when you try to generate references, temporarily delete the heading with the copyright notice attached as a footnote, and reinsert it after EndNote finishes.)*

1. Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved. [↑](#footnote-ref-1)