Analysis of Handwritten Digit Recognition

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

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 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.

Algorithms

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 tend 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

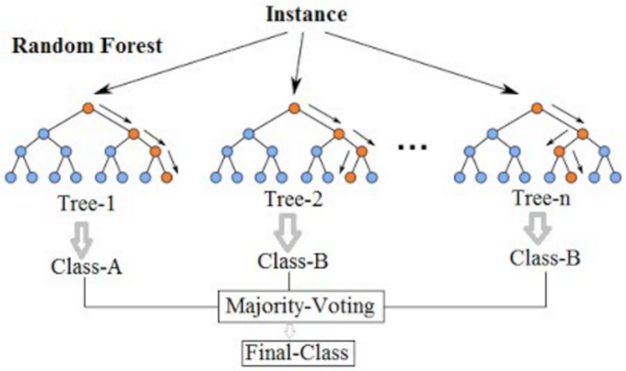

Figure 1: Simplified View of Random Forest

Random forest Pseudocode

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

```
function RandomForest(S , F)
    H ← ∅
    for i ∈ 1, . . . , B do
        S (i) ← A bootstrap sample from S
        hi ← RandomizedTreeLearn(S (i) , F)
        H ← H ∪ {hi}
    end for
    return H
end function
function RandomizedTreeLearn(S , F)
    At each node:
        f ← very small subset of F
        Split on best feature in f
    return The learned tree
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

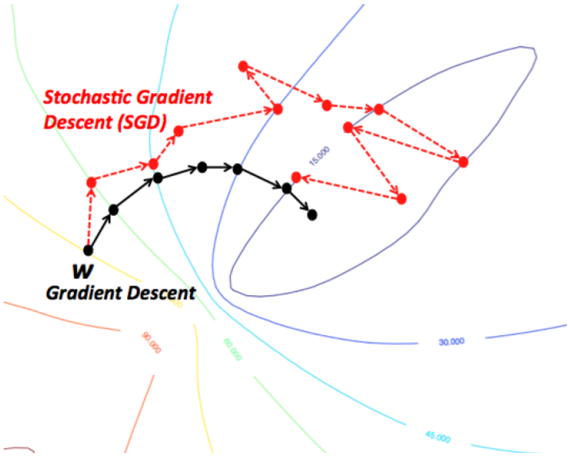

Figure 2:SGD VS GD (Source: https://wikidocs.net/3413)

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* $Q(w)=\frac{1}{2}\sum(target-output)^2$ and the cost gradient can be written as $-\eta\nabla Q_i(w)=-\eta\frac{dQ}{dw}$. 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 η
- Repeat until an approximate minimum is obtained:
    - Randomly shuffle examples in the training set
    - For i=1,2,…….,n. do:
        - $w=w-\eta\nabla Q_i(w)$

  Here $Q_i(w)$ *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

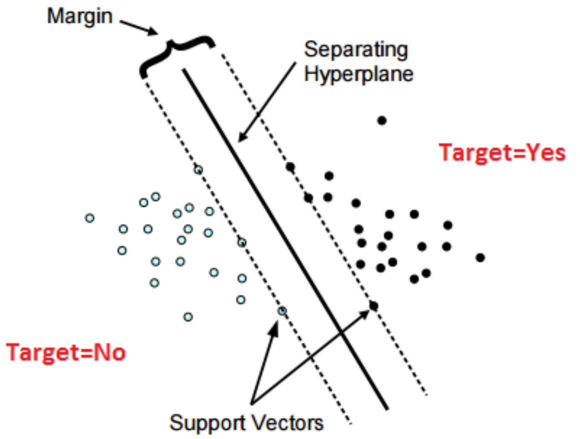

Figure 3: Support Vector Machine (Source: dni-institute.in)

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 (x_1,y_1)…….(x_n,y_n) where y can 1 or -1 and each x is a p dimensional real vector. Any Hyperplane can be written as

$$\vec{w}.\vec{x} - b = 0$$

Here, $\vec{w}$ is the normal vector to the Hyperplane.

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

$$\vec{w}.\vec{x} - b = 1$$
$$\vec{w}.\vec{x} - b = -1$$

Distance between these hyperplane is $\frac{2}{w}$, hence inorder to maximize the distance $\vec{w}$.

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

Artificial Neural Networks (ANN) are one of the most effective solutions when it comes to learning problems. ANN's are inspired by the human brain. Each node imitates a neuron. The inner working of a node could either be simulated by a perceptron or a sigmoid function. Perceptron is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time.

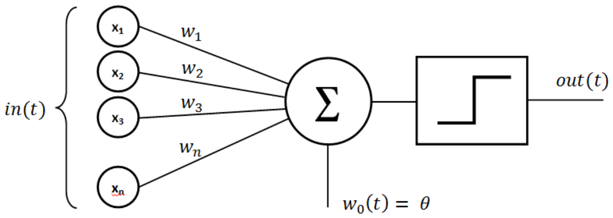

Figure 4:Perceptron (Source : github.com/cdipaolo/goml)

Sigmoid functions on the other hand simulate the exact functioning of perceptrons with one difference that they operate and output on a continuous spectrum. This helps in avoiding random jumps which is observed in the perceptron. Every ANN includes an input, hidden and output layer.

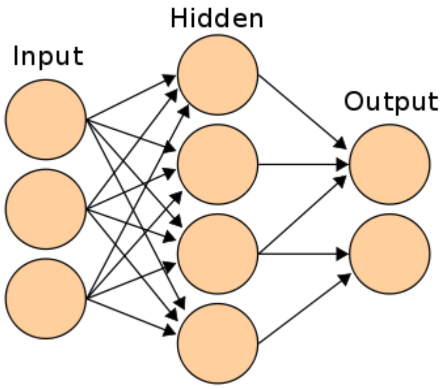

Figure 5: Artificial Neural Net (Source: colah.gihub.io)

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:

$$X\_std = ((X - X.min\ (axis = 0)\ ))/\ (X.max\ (axis = 0) - X.min\ (axis = 0)\ )$$

$$X\_scaled\ = X\_std\ /\ (max\ - min)\ +\ min$$

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 d^2A / dD^2 = -ve. Random forest was able to give an accuracy of 94.77% when trained using 100% of the training dataset.

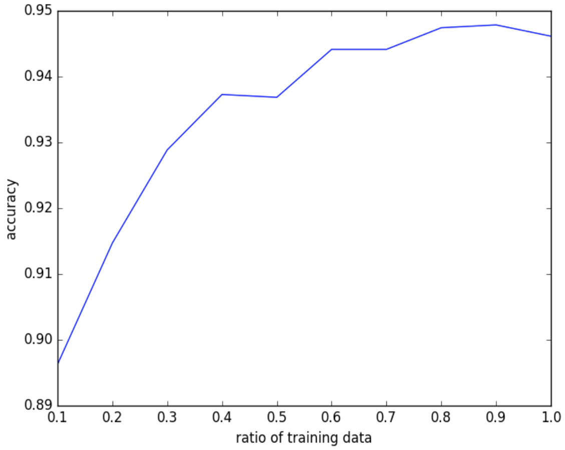

Figure 6: Analysis of Random Forest Algorithm

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 d^2A / dD^2 = -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.

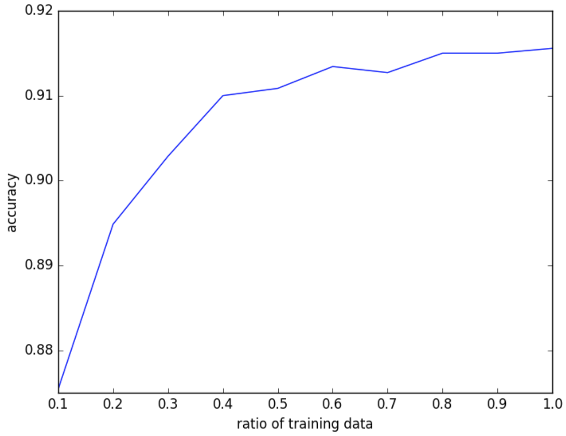

Figure 7: Analysis of Support Vector Machine

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.

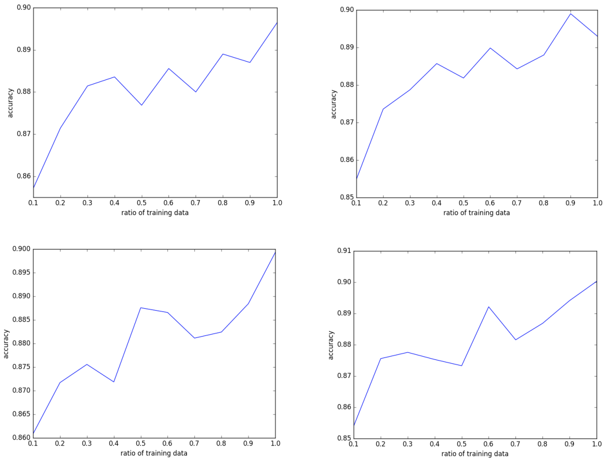

Figure 8: Analysis of Stochastic Gradient Descent Algorithm

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%.

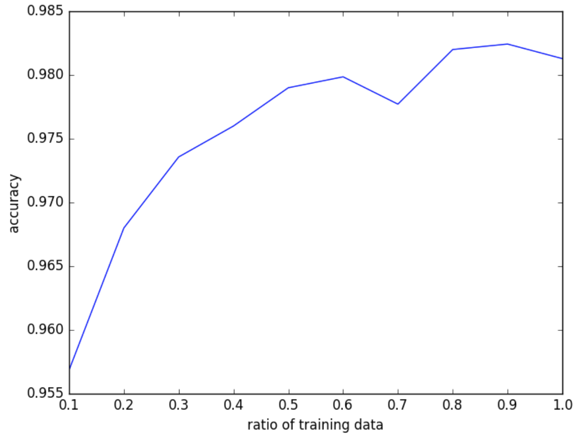

Figure 9: Analysis of Artificial Neural Network

Summary of comparisons

The experiments on RF, SGD,SVM and ANN show that artificial neural network gave the best accuracy of 98.13% when the model is trained with 100% training dataset. ANN is followed by RF, SVM and SGD in decreasing order of accuracies. In general, we can observe a similar pattern in the graphs of RF, SVM and SGD while the graph of SGD varied with increase in training data entries due to the stochastic nature of the algorithm.

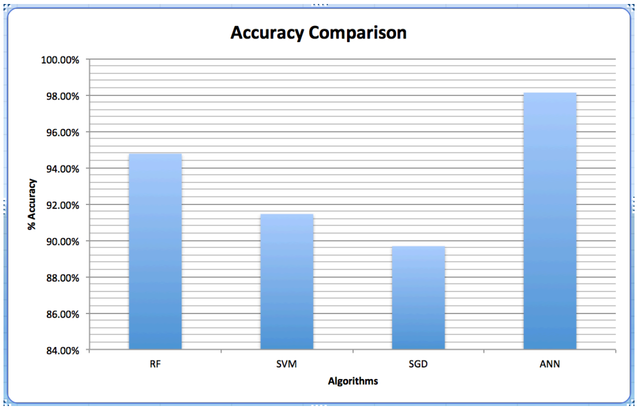

Figure 9: Accuracy Comparison of RF, SGD, SVM, ANN

Improvement

The results obtained previously show an accurate representation of the algorithms and their accuracies. One major thing to note is that all of the algorithms were implemented with the standard features, i.e. one feature for every pixel: 0 if its off and 1 if its on. Hence this would result in a total of 784 pixels. The effectiveness of any learning algorithm depends a lot on the features chosen. Hence, as a method to improve the accuracy in test data prediction, the features must be improved. One such rich feature set in handwritten digit recognition is Histogram of Oriented Gradients (HOG).

Histogram of Oriented Gradients

Histogram of Oriented Gradients or HOG was first developed in 2005 by researchers Navneet Dalal and Bill Triggs. They used this approach in pedestrian detection in security cameras. Later applications of this approach was found in human face detection.

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

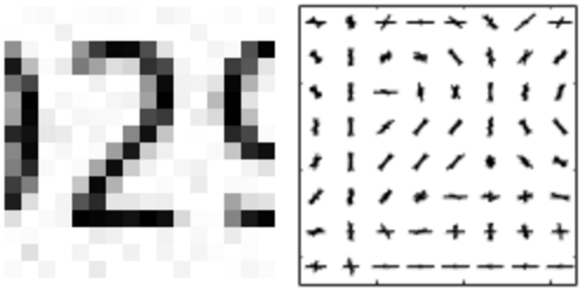

Figure 10: An example of the image compared with its hog feature image.

Since our digit recognition implementation consisted of preprocessing, the preprocessing required for implementing the histogram of oriented gradients was satisfied. Next step was extraction of the HOG features. Here, for every 28 X 28 digit image, it first had compiled into a feature array of 28 X 28 dimension. Then this feature array would be transformed such that each each cell would give out an orientation that it finds. Every pixel in that cell would vote for the orientation and then the cell would combine all these votes to give out a single orientation. Combining all the features would form the feature set, i.e. the new features which would be used for training and testing.

We have used Scikit Image's hog module for our analysis. We tried out the HOG implementation by varying its parameters, but only one configuration was optimal. In our implementation of HOG, we have used the optimal configuration for our tests. Our HOG features vary over 9 orientations. We are using 16 pixels per cell and 4 cells per block.

The histogram of oriented gradients feature set has often been coupled with support vector machines to improve their performance. In this analysis, we would try to improve our previously implemented algorithms using HOG.

Analysis using HOG Features

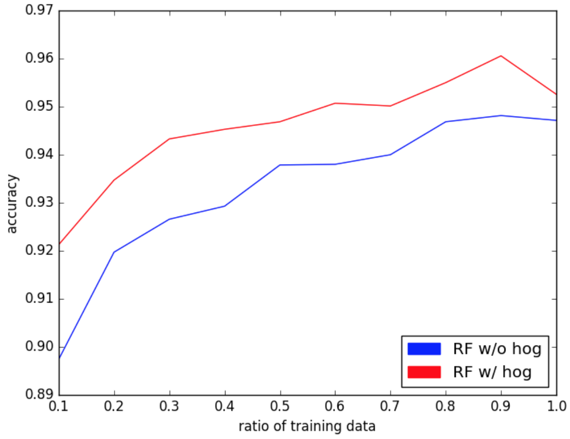

Figure 11: Analysis of Random Forest Using Hog

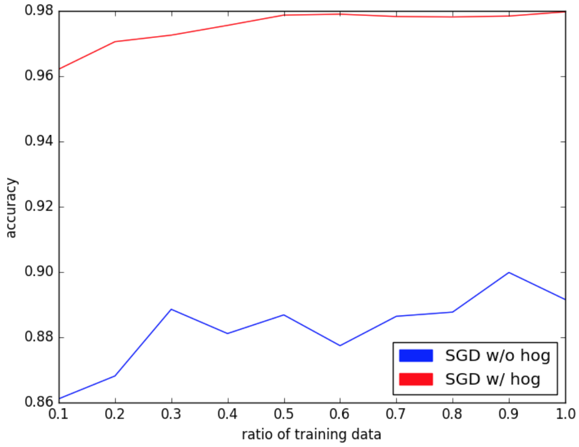

Figure 12: Analysis of Stochastic Gradient Descent Using Hog

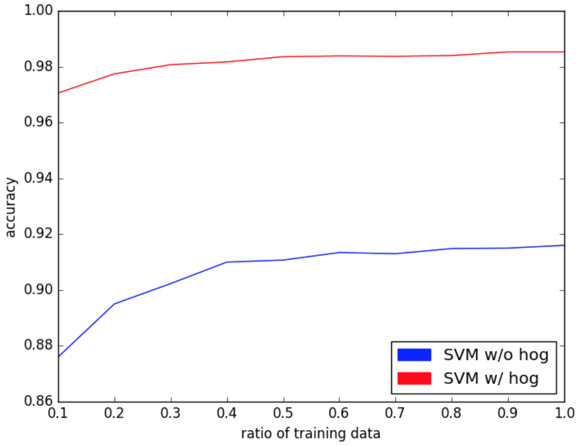

Figure 13: Analysis of Support Vector Machine Using Hog

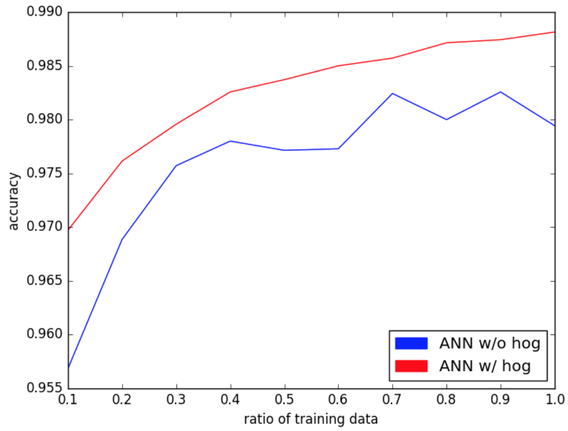

Figure 14: Analysis of Artificial Neural Network Using Hog

We could observe from these graphs that the use of histogram of oriented gradients feature set has boosted the performance of the algorithms. The algorithms now could predict the test data values at a much better accuracy. The HOG approach worked for all the 4 previously implemented algorithms.

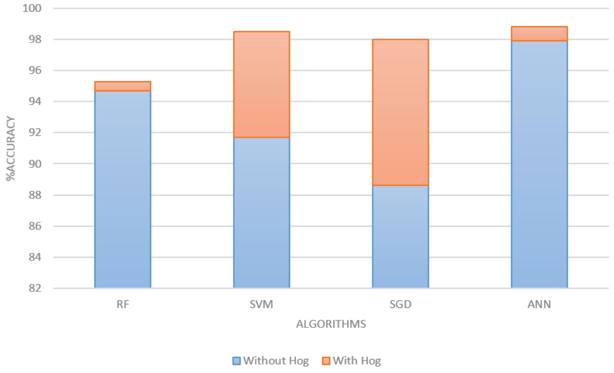

Figure 15: Comparison of Improvement Using Hog

The use of HOG features improved the runtime for the SVM and ANN implementations, while remained the same for Random Forest and SGD. Below is the table for the same:

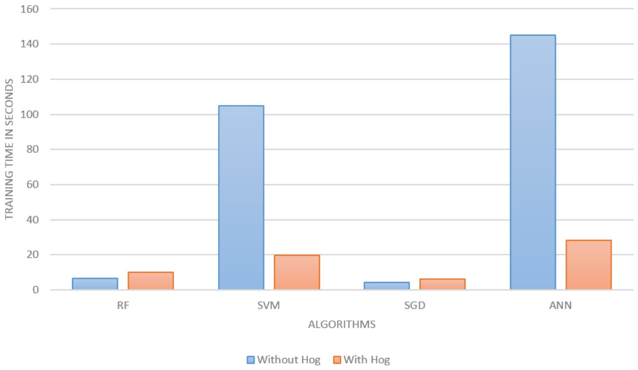

Figure 16: Training Time Comparison

Conclusion

Experimenting with different algorithms has shown that each algorithm can produce different accuracies for a particular Dataset. We saw that Artificial Neural Networks performed the best among all four algorithms in terms of accuracy and hence is considered state of the art. One important take away from these experiments is the importance of feature selection in a machine learning problem. Proper feature selection can help in reducing the training time and improves the accuracy by preventing overfitting of the data. It was observed in general that implementing Hog feature descriptor increased the accuracies of RF, SVM, SGD and ANN. Stochastic Gradient Descent showed a comparatively large increase in accuracy of about 10.6% when hog feature descriptor was implemented.

Algorithm	Accuracy without Hog	Accuracy with Hog
Random Forest	94.7%	95.3%
Stochastic Gradient Descent	88.6%	98.0%
Support Vector Machine	91.7%	98.5%
Artificial Neural Network	97.9%	98.8%

Algorithm	Training Time without HOG (in seconds)	Training Time with HOG (in seconds)
Random Forest	6.51	10.08
Stochastic Gradient Descent	4.12	6.27
Support Vector Machine	105	19.88
Artificial Neural Network	145	28.35

Table 1: Comparison of Accuracies and Training Time of RF, SGD, SVM, ANN

References

1. LeCun.Y., (1991). Reading Handwritten Digit : A Zip Code Recognition. [online] Available at: http://yann.lecun.com/exdb/publis/pdf/matan-92.pdf .
2. LeCun, Y., Jackel, L., Bottou, L., Brunot, L., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., Simard,P., and Vapnik,V. (n.d.). Comparison of Learning Algorithms for Handwriting Digit Recognition. [online] Available at: http://yann.lecun.com/exdb/publis/pdf/lecun-95b.pdf .
3. Stat.berkeley.edu. (n.d.). Random forests - classification description. [online] Available at: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.
4. Pages.cs.wisc.edu.(n.d). [online] Available at: http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf
5. En.wikipedia.org. (n.d.). Support vector machine. [online] Available at: https://en.wikipedia.org/wiki/Support_vector_machine#frbanner3 .
6. Sunaresan, V. and Lin, J. (n.d.). Recognizing Handwritten Digits and Characters. [online] Available at: http://cs231n.stanford.edu/reports/vishnu_final.pdf.
7. Scikit-learn.org. (n.d.). sklearn.preprocessing.MinMaxScaler — scikit-learn 0.18.1 documentation. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.
8. Ufldl.stanford.edu. (n.d.). Unsupervised Feature Learning and Deep Learning Tutorial. [online] Available at: http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/.
9. En.wikipedia.org. (n.d.). Artificial neural network. [online] Available at: https://en.wikipedia.org/wiki/Artificial_neural_network.
10. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition.
11. Gupta, J., Chanda, B. (2012) Novel Methods for Slope and Slant Correction of Off-line Handwritten Text Word. Third International Conference on Emerging Applications of Information Technology.
12. R. Babu,U., Venkateswarlu, Y. and Chintha, A. K. (2014), "Handwritten Digit Recognition Using K-Nearest Neighbour Classifier," World Congress on Computing and Communication Technologies, Trichirappalli.
13. AL-Mansoori, S.(2015). Intelligent Handwritten Digit Recognition using Artificial Neural Network[online] Available at: http://www.ijera.com/papers/Vol5_issue5/Part%20-%203/H505034651.pdf.