Comparative Evaluation of Classification Algorithms for Handwritten Digit Recognition

Sangeeta Kumari (kumari.14@osu.edu) Bratati Das (das.200@osu.edu)

Abstract

Although there are many methods used for object recognition, we always struggle with questions like which model would be best for solving a problem and how much training data would be enough for a good accuracy without getting into the problem of overfitting. The main motivation behind this work was to study the relation between training data size and different models, the limitations of different learning methods and ways of improving an unsupervised algorithm so that we can get rid of the expensive task of labeling data in supervised learning. Since digit recognition is the most common studies problem, we chose this for our evaluation.

1 Introduction

Handwritten digit recognition is a classic benchmark test for computer vision algorithms and has a wide range of applications in signature verification, bank-check processing, postal-address verification etc. Although supervised learning algorithms like support vector machines (SVM) and neural networks produce higher accuracy in classifying the digits, they are both more expensive; supervised training requires large training datasets to be manually labelled and neural nets has a very high latency. On the contrary, unsupervised learning algorithms have the potential of having better training efficiency and can be generalized to more complicated handwriting recognition problems like recognizing handwritten alphabet or Chinese characters.

In this project, we use the MNIST dataset, consisting of 60000 training images and 10000 testing images, to compare the performances of supervised and unsupervised algorithms. Each image in the set is a 28 x 28 matrix of pixels in grayscale ranging from 0 to 255 and labelled with the digit it represents. Our first approach is to implement discriminative models using both supervised and unsupervised learning. For supervised learning, we use Error-Correcting Output Codes (ECOC) multiclass model which basically is an ensemble of support vector machines and a multilayer neural network and for unsupervised, we implement an improvised version of k-means clustering algorithm. Despite the fact that discriminative models do not need to model the distribution of the observed variables, they cannot generally express more complex relationships between the observed and target variables which is why our second approach is an implementation of a generative model, for which we choose Deep Belief Network (DBN). We find that generative models outperformed the discriminative ones giving an accuracy as high as 99.98%.

2 Discriminative Models

2.1 Supervised Learning Algorithm - Error-Correction Output Code Multiclass Model

ECOC is an ensemble method designed for multi-class classification problem. In multi-class classification problem, the task is to decide one label from k > 2 possible choices. In digit recognition task, we need to map each handwritten digit to one of k = 10 classes. ECOC is a meta method which combines many binary classifiers in order to solve the multi-class problem. MATLAB's *fitcecoc()* function is used to train the multi-class model using support vector machine binary learners.

2.1.1 Feature Extraction

Histogram of Oriented Gradient (HOG) was first proposed by Dalal and Triggs [2] for human body detection but it is now widely used for pattern recognition. It counts occurrences of gradient orientation in localized portions of the image. Since Dalal and Triggs point out that image pre-processing provides little impact on performance, we skip this step and directly compute gradient values. For this, the image is divided into small square cells (we used 4×4) and based on the gradient values, they are binned in 9 histogram channels, thus producing a feature vector of 576 features which is what the SVM is trained with. To account for changes in illumination and contrast, the gradient strengths is also locally normalized.

2.1.2 Classification

Class	Code Word														
	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
2	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
3	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
4	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
5	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
6	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
8	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
9	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

Figure 1: A 15-bit error-correcting output code for a ten-class problem

Figure 1 shows a 15-bit error-correcting output code for the digit classification (ten-class) problem. Each class is assigned a unique binary string of length 15, called a codeword. During training, one binary classifier is learned for each column. For example, for the first column, we build a binary classifier to separate {0, 2, 4, 6, 8} from {1, 3, 5, 7, 9}. Thus 15 binary classifiers are trained in this way. To classify a new data point x, all 15 binary classifiers are evaluated to obtain a 15-bit string. Finally, we choose the class whose codeword is closest to x's output string as the predicted label.

2.2 Multi-Layer Neural Network

In problems like computer vision, the number of features is very large. For example, a 28 x 28 pixel greyscale image has 784 pixels and supervised algorithms like linear or logistic regression becomes computationally expensive. Hence, we choose to implement a simple 3-layer neural network. A neural

network [6] is a computational approach which is based on a large collection of neural units, loosely modelling the way a biological brain would solve problems. The neuron is a computational unit that takes as input the training examples and outputs a labels of the data points based on an activation function. The activation function we use is the sigmoid function, f(z) = 1/(1+exp(-z)).

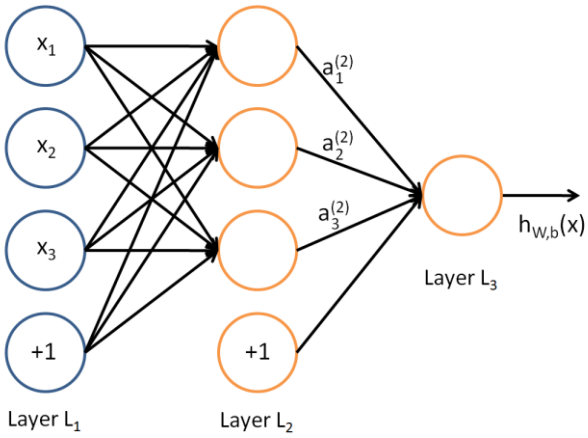

Figure 2: A simple neural network

A neural network is put together by combining neurons in multiple layers along with respective weights, which are calculated in every iteration of the learning algorithm. Figure 2 depicts a simple neural network that we implement on the MNIST database. The leftmost layer (L_1) is called the input layer, the middle layer (L_2) is called the hidden layer and the rightmost layer (L_3) is called the output layer. This implementation of neural network gives an accuracy of 83% (Table 1). More complex neural networks can be designed by varying the activation function, the number of hidden layers and the weights assigned to the controlling features.

2.3 Unsupervised Learning Algorithm - k-means++

2.3.1 Pre-processing the Data

We centralize the images by centering the images at the origin but since the MNIST dataset is already well centered, we do not see any significant change in accuracy. Euclidean distance function is used to find the distance between the data points to cluster and classify the digits.

2.3.2 Feature Selection

Principal Component Analysis (PCA) on the data set follows the above step. The principal component coefficients, also called loadings, are calculated using singular value decomposition (SVD) algorithm. This analysis selects the k-dimensional linear subspace spanned by the 28 x 28 dimensional inputs with the largest correlation to each other. We find an improvement of about 2% in the accuracy of the k-means classification when principal component analysis is performed on the dataset (Table 2).

2.3.3 Improved Initialization

The k-means clustering algorithm is highly susceptible to the initialization of the centroids of the clusters and random initialization of centroids does not always produce optimum classification results. Thus, we implement the improved initialization technique from k-means++ algorithm. The first cluster centroid is chosen uniformly at random from the data points being clustered, after which each subsequent cluster center is chosen from the remaining data points by taking into consideration the probability of it being proportional to its squared distance from the point's closest existing cluster center, i.e., for each data point x, D(x), distance between x and the nearest center that has already been chosen, is computed. The new cluster centroid is calculated using a weighted probability distribution where a point x is chosen with the probability proportional to $(D(x))^2$. We do this until k cluster centroids are chosen, after which we run the usual k-means clustering algorithm. The improvised initialization produced an increase of 3% in the accuracy of the clustering algorithm as seen in Table 2.

2.3.4 K-means++ Clustering algorithm

After the improved initialization, we perform the k-means clustering algorithm. For each iteration of the clustering algorithm, the sum of Euclidean distances squared from their respective centroids is calculated. After each iteration of the converge loop, we choose at random a training example from each cluster as the new centroids and repeat the clustering algorithm to check if the function calculated above is minimized. The program terminates when the choosing of new centroids does not improve the function in the consecutive trials.

With the improved feature set and initialization, we receive an accuracy of 60% in the clustering algorithm, which is an improvement of 5% from the simple k-means algorithm with random centroid initialization and without feature set selection. However, we were not able to produce better results than the supervised algorithms discussed in sections 2.1 and 3.1.

3 Generative Models

3.1 Deep Belief Network (DBN)

A successful learning algorithm for deep networks is one that discovers a meaningful and possibly complex hidden representation of the data at its top hidden layer. However, learning such non-linear representations is a hard problem. A solution, proposed by Hinton [3], is based on the learning algorithm of the restricted Boltzmann machine (RBM) [4], a generative model that uses a layer of binary variables to explain its input data. Hinton argues that this representation can be improved by giving it as input to another RBM, whose posterior over its hidden layer will then provide a more complex representation of the input.

A DBN is a multi-layer generative model, with each layer pair forming a RBM. In a RBM, each unit is a stochastic binary neuron, and the probability to turn on is given by a sigmoid function applied to the weighted sum of its inputs. We have a visible layer of RBM consisting of 784 (=28x28 pixels) units which is a mere copy of the data vector and a bias. Besides, the visible layer has undirected connections with the hidden layer (800 units in our case). In order to perform classification of hand-written images, another layer is added at the end of the last hidden layer with corresponding weights. Thus, the last layer has 10 units, each unit corresponding to a class (corresponding to digits 0-9).

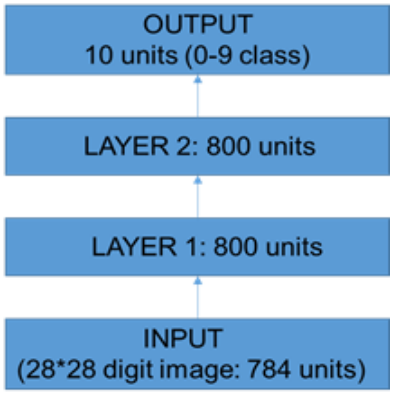

Figure 3: 784-800-800-10 architecture for DBN

3.1.1 Pre-training

Each layer is pre-trained with an unsupervised learning algorithm, learning a nonlinear transformation of its input (the output of the previous layer) that captures the main variations in its input. This unsupervised pre-training sets the stage for a final training phase where the deep architecture is fine-tuned with respect to a supervised training criterion with gradient-based optimization.

3.1.2 Training with Contrastive Divergence

For training RBM, we first randomly initialize the units and parameters. Then, there are two phases in Contrastive Divergence (CD) algorithm - positive and negative. During the positive phase, the binary states of the hidden units are determined by calculating the probabilities of weights and visible units. Since it is increasing the probability of training data, it is called positive phase. On the other hand, the negative phase decreases the probability of samples generated by the model. A complete positive negative phase is considered as one epoch and the error between generated samples by the model and actual data vector is calculated at the end of the iteration. Finally, weights are updated by taking the derivative of the probability of visible units with respect to weights, which is the expectation of the difference between positive phase contribution and negative phase contribution.

Training the DBN is achieved by greedy learning that trains one RBM at a time and continues until the last RBM. This simple greedy learning algorithm works because training RBM using CD algorithm for each layer looks for the local optimum and the next stacked RBM layer takes those optimally trained values and again looks for the local optimum. At the end of this procedure, it is likely to get the global optimum as each layer consistently trained to get the optimum value. Greedy learning algorithm for is implemented by simply initializing parameters with previously obtained values. Then, it calls Contrastive Algorithm to train next RBM hidden layer.

3.1.3 Fine-Tuning and Classification

Pre-training introduces a useful prior to the *supervised fine-tuning* training procedure. We use backpropagation for fine-tuning, whose error derivative functions help to update weights for future prediction. Classification is performed by obtaining the highest probability of one unit in the last layer.

4 Experimental Results

4.1 ECOC Model

Since the data used to train the classifier are HOG feature vectors extracted from the training images, it is important to make sure the HOG feature vector encodes the right amount of information about the object. It should neither be too less that we lose necessary shape information nor too high that we unnecessarily increase the feature vector dimensionality. We compare the performance of the classifier using cell size of [8 8] and [4 4] and find that the accuracy increases in the latter case. We also study the effects with increase in training data size and we notice performance improvement with increase in the size. The result can be found in Figure 4 and 5.

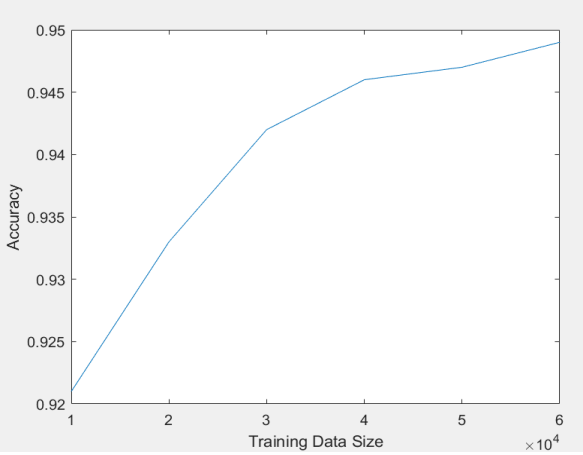

Figure 4: Accuracy of ECOC with cell size [8 8]

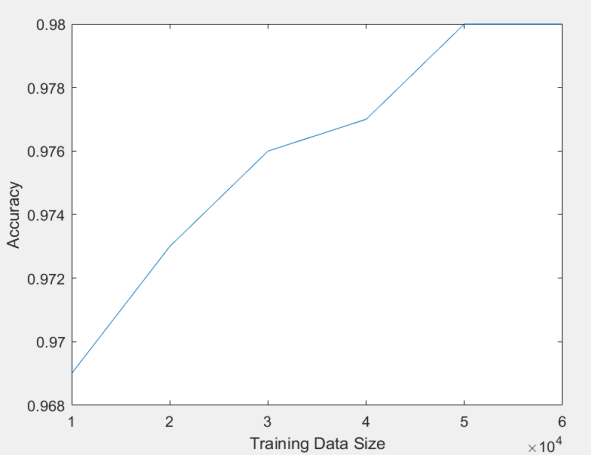

Figure 5: Accuracy of ECOC with cell size [4 4]

4.2 Multi-layer Neural Network

Type of Algorithm	Training Set Accuracy	Test Set Accuracy
Single Layer Neural Network	82.74	82.54
Multilayer Neural Network	82.94	82.8

Table 1: Accuracy results (in %) for neural networks

4.3 K-means++

Clustering Specifics	Training Set Accuracy	Test Set Accuracy
K-means	57.75	58.5
K-means with image centralization	58.19	58.59
PCA + k-means	58.2	58.61
PCA + k-means++	59.49	59.93

Table 2: Accuracy results (in %) for different versions of k-means clustering

4.4 Deep Belief Network

Training Dataset Size	Training Set Accuracy	Test Set Accuracy
10000	97.99	97.3
30000	98.99	97.8
60000	98.99	98.3

Table 3: Accuracy results (in %) for different training data size in DBN model

6 Discussion and Future Work

Although unsupervised learning is not able to give an accuracy as high as their supervised counterparts, we do see that it can be improved with slight modifications. We also see that generative models give the maximum accuracy in all the methods evaluated and also have the ability to learn low-level features without requiring feedback from the label and they can learn many more parameters than discriminative models without overfitting.

Although we use supervised learning for training the DBN, we would like to implement it in an unsupervised manner so that we can work with unlabeled data. Also, we would like to make the learning process faster by using efficient GPU implementations so as to make such models feasible for real-time applications.

References

[1] Thomas G. Dietterich and Ghulum Bakiri. (1995) Solving multiclass learning problems via error-correcting output codes. arXiv preprint cs/9501101.

[2] N. Dalal and B. Triggs. (CVPR, 2005) Histograms of oriented gradients for human detection.

[3] Hinton, G. E, Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural Computation.

[4] Hinton, G. E. (2010). A Practical Guide to Training Restricted Boltzmann Machines. Department of Computer Science; University of Toronto.

[5] David Arthur and Sergei Vassilvitskii. kmeans++: The Advantages of Careful Seeding. SODA '07. http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

[6] Stanford Unsupervised Feature Learning and Deep Learning Tutorial. http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/