# CSE574 Assignment 3 Report

# CLASSIFICATION

**Tenzin Norden**
**UB person# 50096989**
**November 21, 2018**

## 1 Project Description

In this project, we have 0 to 9 handwritten digit dataset from MNIST and USPS. We have to train a machine learning model to classify input samples data to digits 0 to 9. A total of 10 classes makes this problem a multiclass classification problem. Four classification machine learning models namely multiclass logistic regression, neural network, support vector machine (svm) and random forest classifiers are implemented in this project to classify the handwritten digits into 10 classes from 0 to 9. This problem is a supervised machine learning.

## 2 Dataset

### 2.1 MNIST Data

The MNIST (Modified National Institute of Standards and Technology) database is a commonly used large database of handwritten digits from 0 to 9 for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

The MNIST database contains 60,000 training images and 10,000 testing images. Each of the input data sample from MNIST have 784 features and corresponds to one label or class (one number from 0 to 9). The original black and white images from MNIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The images were normalized to fit into the center of a 28x28 pixel field.

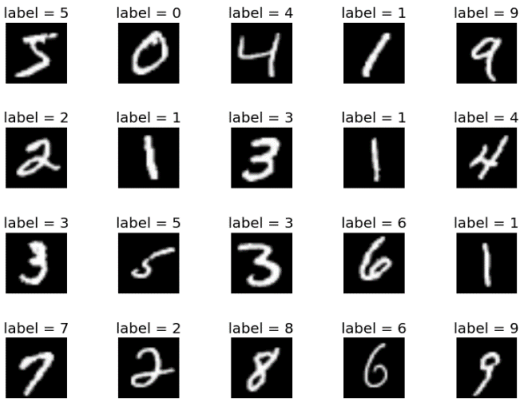


Figure 1 Example of MNIST images. Each image is a 28x28 pixel. https://corochann.com/mnist-dataset-introduction-1138.html

### 2.1 USPS Data

A second set of data, USPS handwritten digit dataset is used as a testing data to test whether the classifiers models implemented could generalize to a new population of data. Example of the digits are given in figure 2. Each digit has a 2000 samples available for testing. The images are segmented and scanned at a resolution of 100 ppi and cropped. They are then resized to a 28x28 pixels size images like the MNIST digit images. USPS data will be then used to test the trained model and the results are compared with that of the MNIST test data.

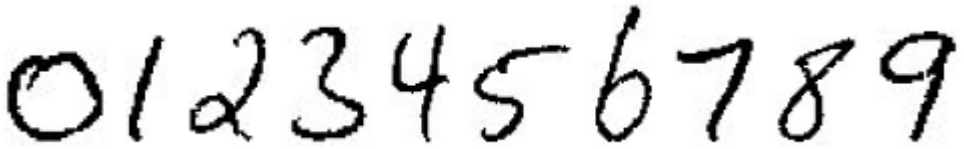

Figure 2. Examples of USPS images for each of the digits.

### 3.1 Model 1: Softmax Regression / Multiclass Logistic Regression

Unlike Logistic Regression which is an algorithm for performing binary classification which used sigmoid function. But for problems with $\boldsymbol{K}$ classes ($\boldsymbol{K}$>2), a multiclass classification approach Logistic regression or softmax regression can be used. The sigmoid logistic function is replaced by softmax function, hence the name softmax regression.

In multiclass logistic regression, the output layer is forced to be discrete probability distribution over the $\boldsymbol{K}$ classes. The output vector **y** should contain (a) non-negative values and (b) all the values should sum up to 1 to be a valid probability distribution. All of this is achieved by a ***softmax*** function.

$$P\left(y = i|z^{(i)}\right) = softmax(z) = \frac{e^{z}}{\sum_{i=1}^{k} e^{z_i}}$$

where $\boldsymbol{k}$ is number of classes and the input vector **z** is defined as

$$\mathbf{z} = w_0 x_0 + w_1 x_1 + \cdots w_m x_m = \mathbf{w}^T.\mathbf{x}$$

Here w is the weight vector, x is the feature vector of 1 training sample, m is the number of features and $w_0$ is the bias term. The softmax function will compute the probability $P\left(y = i|z^{(i)}\right)$ for each class label (i=1,2…k). The denominator term in the softmax function is the normalization term which ensures that the probabilities of the classes sum up to 1. The output probabilities are converted into class labels by taking the maximum probability index position of output vector. The output class labels are then compared with the index of the one-hot encoding of the target class labels to find the performance of the model.

In this model, the class labels are first encoded into one-hot vector. For example, class 0 is encoded to [1, 0, 0, …… 0], class 1 = [0, 1, 0, 0, …0] and so on. The respective class corresponds to 1 respective index in the vector and rest as zeros.

The weight matrix has a shape of mxk (m=number of features, k = number of

classes) and it is initialized randomly. In this model, **w** is initialized to a matrix of zeros.

The model is trained via optimization algorithm gradient descent. The loss function is minimized to train the model. The loss function can be found from the log likelihood of observations defined as

$$p(T \mid \boldsymbol{w}_1,..,\boldsymbol{w}_{10}) = \prod_{n=1}^{N}\prod_{k=1}^{10} p(C_k \mid \boldsymbol{\phi}_n)^{t_{n,k}} = \prod_{n=1}^{N}\prod_{k=1}^{10} y_{nk}^{t_{nk}}$$

where n is the number of samples and k classes. And $\boldsymbol{y}_{nk}$ is the output probability matrix after going through softmax function. Loss function can then be defined as the negative log-likelihood as follows

$$E(\boldsymbol{w}_1,...,\boldsymbol{w}_{10}) = -\ln p(T \mid \boldsymbol{w}_1,..,\boldsymbol{w}_{10}) = -\sum_{n=1}^{N}\sum_{k=1}^{10} t_{nk} \ln y_{nk}$$

here T is the target matrix of shape nxk, where each row is one-hot encoding of the corresponding label. This loss function is known as cross-entropy error for multi-class.

Finally, the weight is optimized with gradient descent algorithm. The gradient of the loss function with respect to parameter $w_j$ as

$$\nabla_{w_j} E(w_1, \dots w_{10}) = -1/N \sum_{n=1}^{N} (y_{nj} - t_{nj})\, x_n$$

The gradient is then use to update the weights with back propagation of the gradient:

$$w_j^{new} = w_j^{old} - \eta \nabla_{w_j} E(w_1, \dots w_{10})$$

where $\eta$ is learning rate, the rate of the descent. The model is run over many iterations to find the optimum weight which gives the best performance. I used stochastic gradient descent algorithm to train the model using batches of samples over several epochs, where weight is updated after every batches.

### 3.2 Model 2: Neural Network

Neural network classification algorithm is implemented as the second machine learning model to classify the handwritten digits from 0 to 9.

The input to the model is features from each sample data which corresponds to a particular class label and have 10 output probabilities corresponding to each label.

The model is set up in **keras** framework. Tensorflow placeholder for input variable with width 784 (equal to number of features of an input sample) and output variable with width 10 (10 classes) is created. I implemented 2 hidden layers neural network in this project. Number of neurons in the hidden layer one is set to 256 and hidden layer two is set to 32. The number of neurons is a hyperparameter which can be changed to optimize the model performance. An example of the schematic for two layers deep neural network with one hidden layer and one output layer is shown in figure 1.

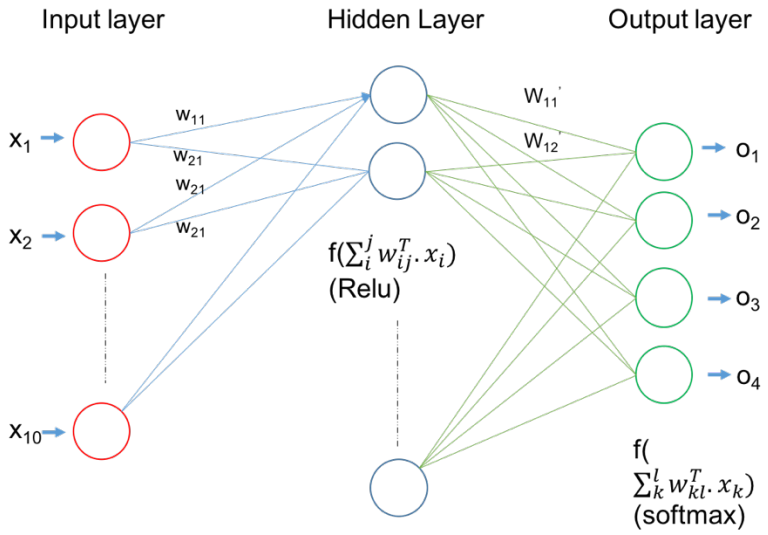


Figure 3. Example of two layers deep neural network with one hidden layer and 10 input neurons. Relu Activation function used at hidden layer and softmax activation is used at the output layer

The weights for the input, hidden and output neurons are randomly initialized to a normal distribution. *In the hidden layer, an activation function is applied on the sum of the dot products of input vectors (x) and their corresponding weights ($w^T$), which gives the output of that layer.* An activation function is used to make neural network non-linear and limits the output signal to a finite value, for eg, between 0 and 1.

$$\mathrm{Relu(f)} = (\sum_{i}^{j} w_{ij}^{T}.x_i, 0)$$

In this model, **relu (rectified linear unit)** activation function is used in the hidden layer which is just f(x) = max(0, x). Relu limits the output to be either 0 or x and it can only be used in hidden layers of the neural network.

The output of the hidden layer after applying relu function is set as input for the output layer of our neural network model. Once again, the input to this layer is multiplied with a set of random weights and their sum is fed into **softmax** activation which computes probabilities for different output classes.

The output of the output layer is then compared with the actual known output. The error is computed using cross entropy error function as we have classification problem. Steps so far can be called forward propagation.

**Backpropagation** is then performed using stochastic **Gradient Descent** optimizer to minimize the error. It propagates backwards in the network to compute gradient of error function with respect to weights and then optimize weights using Gradient Descent optimization to reduce error. Gradient descent is then used to update and optimize weights until a global minimum of the error function is reached. Keras framework does all of these automatically. **Our model trains via backpropagation.**

To train the model, we run the model certain epochs, where one **epoch** is equivalent to one forward and one backward propagation of all the training data. In this model, we use the **batch size**, the number of training data in one forward and backward propagation, as 128. The output is then classified into 10 classes after all the iterations or training. Epoch and batches are also hyperparameters which is tuned to get the optimal performance of the model.

After batch and epoch training, the testing is done for the testing dataset from MNIST and USPS.

The output label is then compared with the known label of the testing data and the accuracy is computed.

### 3.3 Model 3: Support Vector Machine (SVM)

A support Vector Machine is a discriminative classifier defined by a separating hyperplane. With given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. In other words, the objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N-the number of features) that distinctly classifies the data points.

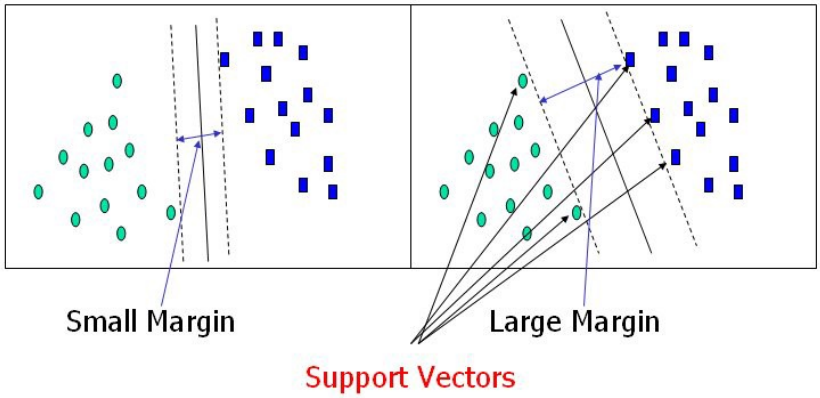


Figure 4. possible hyperplanes classifier and support vectors. (https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47)

The hyperplane with the maximum margin (the maximum distance between data points of both classes) is chosen such that the margin distance provides a reinforcement for the classification of future data with more confidence.

Hyperplane are decision boundaries which helps classify the data points and its dimension depends upon the number of features. So with an input feature of 784, the hyperplane is difficult to imagine in this project. Data points falling on either side of the hyperplane can be attributed to different classes.

The support vectors are data points that are closer to the hyperplane which influence the position and orientation of the hyperplane as shown in figure 4. The margin of the classifier is maximized using these support vectors.

The loss function used to maximize the margin between the data points and the hyperplane is hinge loss function.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

For the implementation of SVM algorithm, we use scikit-learn package SVC classifier. The hyperparameters such as C, kernels such as rbf (radial basis functions), linear, poly and sigmoid, gamma were tuned to optimized the model and get best accuracy.

The kernel parameter is used when SVM model is learning the hyperplane. The problem is transformed using some linear algebra and the choice of kernel is used in

doing so.

Regularization parameter C is the penalty for the error term. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. On the other hand, a very small value of C will cause the optimizer to look for a larger margin separating hyperplane, even if that hyperplane misclassifies more points.

The gamma parameter defines how far the influence of a single training example reaches. Low gamma considers points far away from the possible hyperplane in calculation for the optimized hyperplane. High gamma value means points closer to the plausible hyperplane are considered in calculation.

### 3.4 Model 4: Random Forest

Random Forest algorithm can be used for both classification and regression task. Random Forest algorithm builds multiple decision trees by picking sub-sample sets randomly and merges them together to get a more accurate and stable prediction as seen in figure 5.

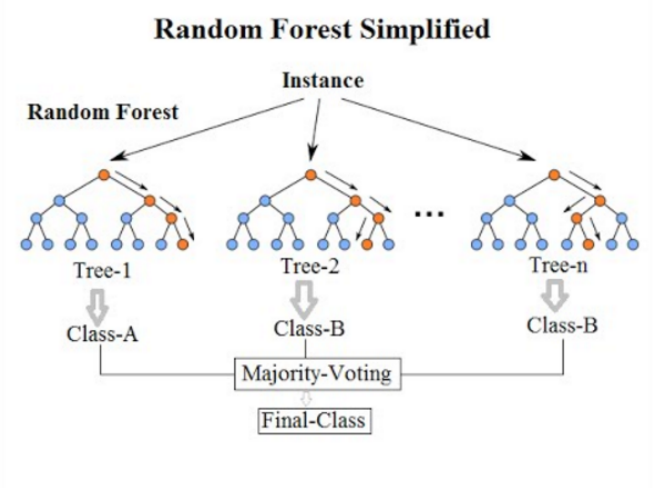


Figure 5. random forest model. (https://en.wikipedia.org/wiki/Random_forest

We use scikit-learn package to implement this model by tuning various hyperparameters.

The n_estimators hyperparameter is just the number of trees the algorithm is tasks to build before taking the maximum voting or taking average of predictions. In general, more trees in the forest increases the performance and will give a better prediction but it slows down the computation time.

The max_depth hyperparameter represents the depth of each tree in the forest. The deeper the tree, the more splits it has and it captures more information about the data. We fit each decision tree with depths ranging from 1 to 32 and plot the training and test errors.

Criterion parameter can be either ‘gini’ for the Gini impurity or ‘entropy’ for the information gain.

## 4 Experiment

I have experimented with the learning program by changing learning rate from 0.1 to 1, activation functions and optimizers to find the combination which gives the best performance. Hyperparameters such as number of epochs (5000), batch size (128) and error function (cross entropy) is kept constant for all experiments. I have also changed number of hidden layers in the network to find the best performance.

Since the weights are randomly initialized, accuracies are different every run. Therefore, all the accuracies plotted are averaged over five individual accuracy.

### 4.1 Softmax Regression

#### a. Learning rate vs Accuracy

In the softmax regression model, we train the model using SGD optimizer with 250 epochs and 100 batchs. The regularization coefficient is kept very minimal at 0.0001. The experiment is done by tuning learning rate measuring accuracy as shown in figure 6. Learning rate is tuned from 0.001 to 0.2.

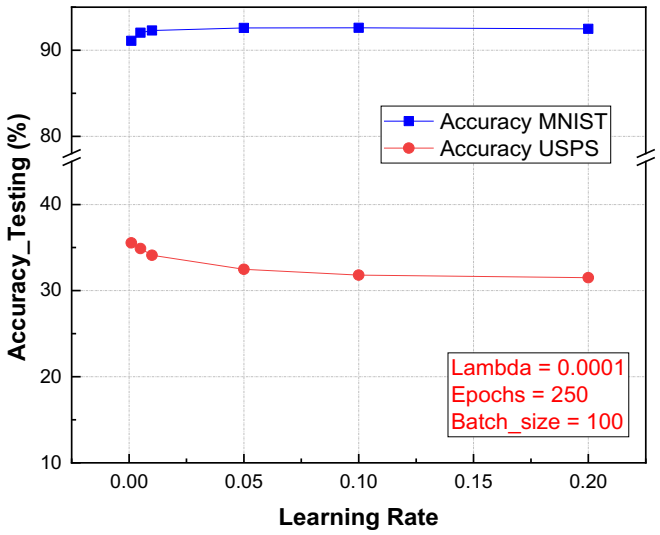


Figure 6. Accuracy vs Learning rate

From figure 6, the accuracy for the MNIST testing data is far better as compared to testing on USPS testing data. Accuracy from MNIST testing data is above 90% for all the tuned values learning rates and is lowest at 0.001, possibly due to overfitting. However, accuracy for USPS dataset is below 35% for all the learning rates.

As the model is trained on the MNIST training data set, the performance is better while testing on the MNIST testing data unlike the USPS testing data because of the so called 'no free lunch theorem'. Which means a training a model on one dataset will not perform well on testing on other dataset even though the two datasets are similar.

The results I have obtained proves no free lunch theorem.

### b. Error vs epoch

Error as a function of epochs is plotted for softmax regression model MNIST training dataset. After every epoch, the weight is updated and the model is better optimized by minimizing the error and increasing accuracy as can be seen in the figure 7 below.

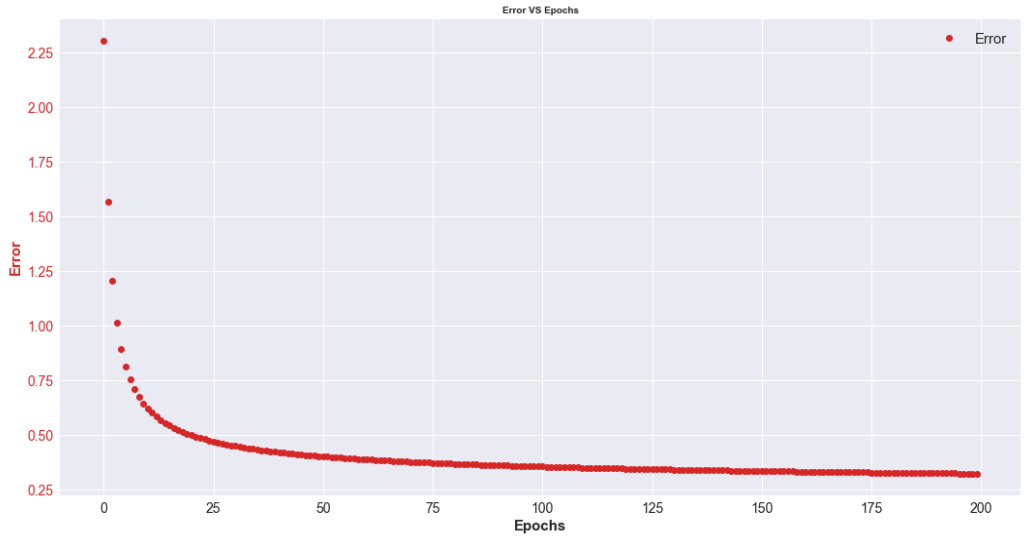


Figure 7. Error vs epoch

### c. Confusion matrix

Confusion matrix formed with the predicted output classes by the model and the actual target values with labels from 0 to 10. The following confusion matrix is constructed from MNIST testing dataset and its normalized out of 100.

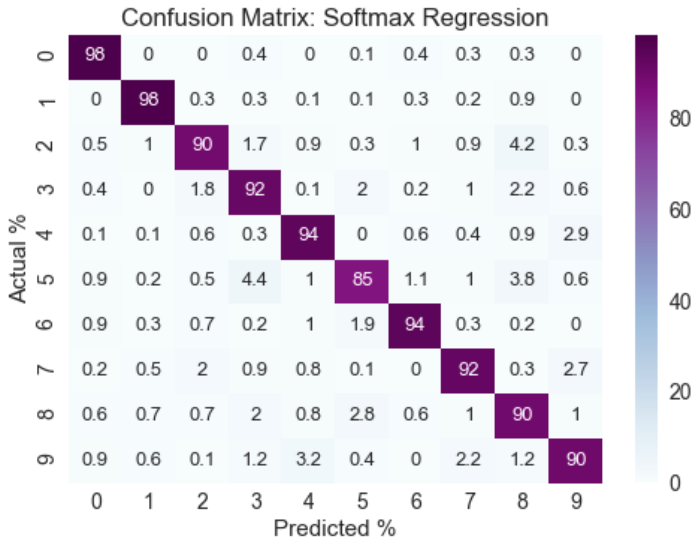


Figure 8 confusion matrix for softmax regression

From the confusion matrix as shown in figure 7, the softmax regression model performs worst in classifying number 5, where it predicted only 85% of the time correctly as compared to other number with better accuracies. Number 5 is mostly predicted as 3 or 8. Number such as 0, 1, 4, 6 are predicted relatively better as

compared to numbers 2, 3, 5, 7, 8 and 9. The total maximum accuracy obtained on the MNIST testing data with tuning hyperparameters to optimal values is 92.49%.

### 4.2 Neural Network

Keras framework is used to implement the neural network model. A two hidden layer neural network model is used to train the MNIST training data. The input neuron is equal to the number of features 784 and output is set to 10 (10 classes). The number neurons in the first hidden layer is set to be 256 and second hidden layer as 32 which gives a good performance. Relu activation function is used in the first and second layers and the final activation used at the output layer is softmax with categorical cross entropy loss function. SGD optimizer is used to train the model through backpropagation.

#### a. Accuracy vs epochs

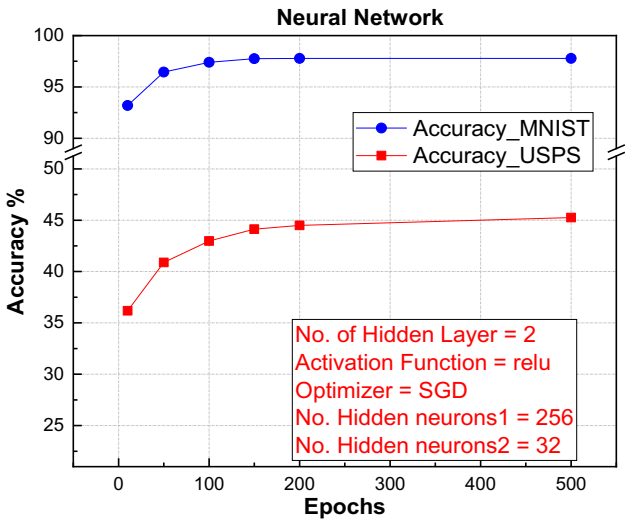


Figure 9. accuracy vs epoch

Accuracy of the MNIST testing data set is plotted for different number of epochs keeping the batch size and other parameters fixed at optimal values. The performance of the model is excellent with accuracies form the MNIST testing data reaching a maximum of 97.8%. The accuracy increases with increasing epochs as expected and it saturates after 150 epochs. Again, the model performs bad when testing on USPS data.

#### b. Confusion matrix

Confusion matrix for neural network classifier is shown in figure 10. The confusion matrix is formed from MNIST testing data prediction and real target values. Neural network model predicts all the number very accurately. With the worst classification with number 9 where 9 is classified as 7 for 1% of the dataset which is not bad at all.

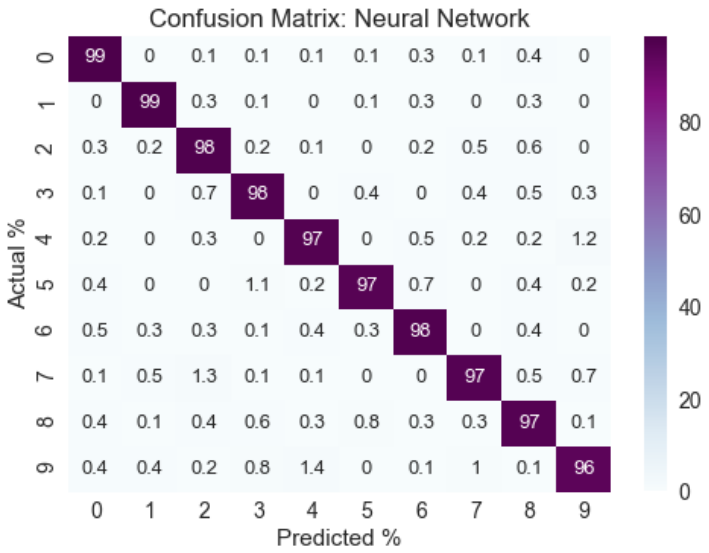


Figure 10 confusion matrix from MNIST testing data: neural network

### 4.3 Support Vector Machine

SVM is implemented using scikit learn package and various hyper-parameters are tuned to find the accuracy on MNIST dataset.

#### a. kernels vs accuracy

In this experiment, various kernels are changed to find the optimal kernel which gives best accuracy and takes the least computation time. Other parameter kept fixed at C=2 and gamma = 0.05 for all kernels.

Table 1. SVM kernels and accuracy

| Kernel | rbf | Sigmoid | Linear | Poly (deg = 3) | Poly (deg = 3) | Poly (deg = 3) |
|---|---|---|---|---|---|---|
| Accuracy (%) | 97.94 | 33.34 | 93 | 97.78 | 96.65 | 95.55 |

From table 1, we can see that 'rbf' kernel gives the best accuracy, which is a bit higher than degree 3 'poly' kernel. However, the computation time of rbf is much longer than 'poly'. Therefore, 'poly' degree 3 kernel is used for other experiments in SVM.

#### b. C, gamma vs accuracy

In figure 11a, I plot accuracy vs C. C is a penalty for the error term and is a regularization term in SVM. For different values of C from 0.5 to 15, the accuracy change is very minor. No free lunch theorem is again proved as models performs badly on USPS testing data but gives very high accuracy around 98% for MNIST test data.

In figure 11b, the accuracy vs gamma parameter is shown. gamma is tuned from values 0.01 to 1. Again, accuracy is lower for small gamma values but the accuracy remains constant for gamma values above 0.05 at about 98%.

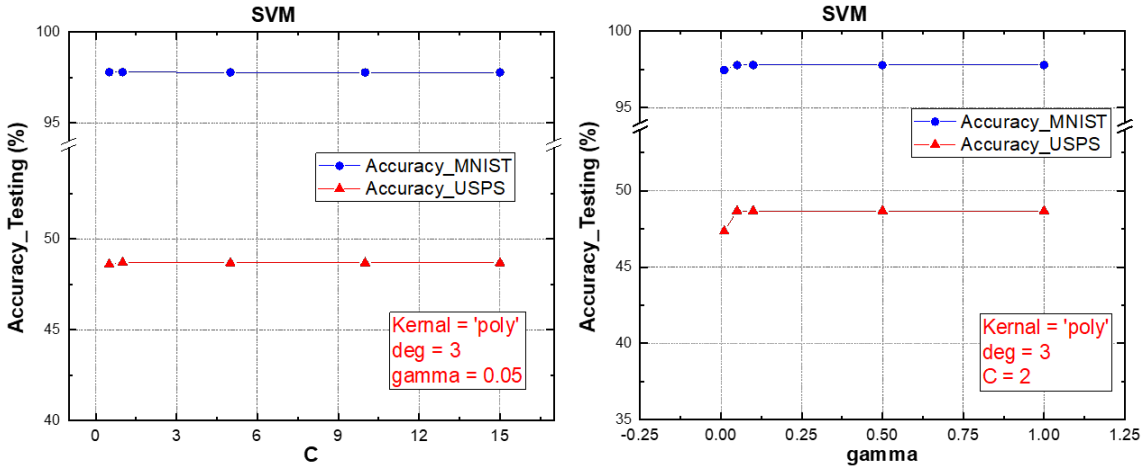


Figure 11. a). Accuracy vs C (left) and b) accuracy vs gamma (right).

## c. Confusion matrix

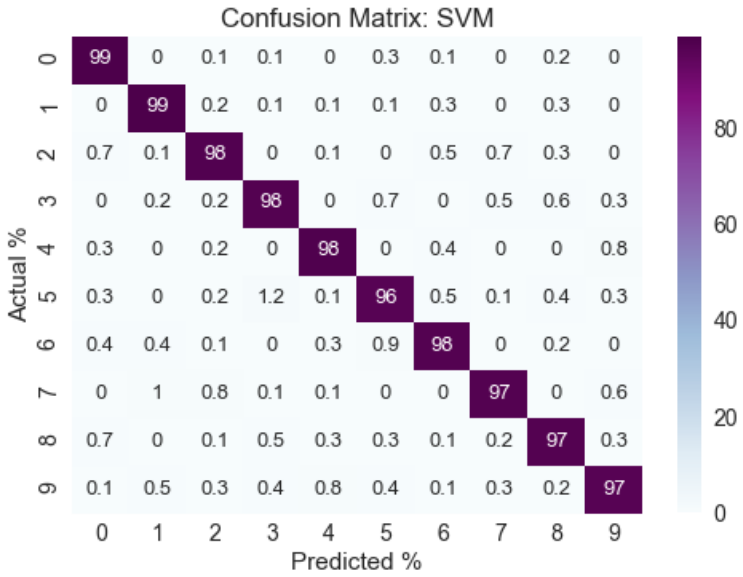


Figure 12. Confusion matrix for svm

Confusion matrix is plotted in figure 12 for SVM model on MNIST testing predicted and target vector. SVM performance is very good for all the numbers. Its worst prediction is on number 5, which was predicted incorrectly as 3 for 1.2% of the data.

### 4.4 Random Forest

Here, parameters such as n_estimator, max_depth is tuned for two different criterions, gini and entropy. Scikit-learn package is again used to implement this model. Since no free lunch is proved from the above three models by testing on USPS dataset, only MNIST test data is used in random forest to test the model accuracy. A high accuracy of 97% is achieved for this model.

#### a. n_estimators vs accuracy

In figure 13a, n_estimators is tuned from 10 to 100 and the measured accuracies are plotted for 'gini' and 'entropy' criterions. Since n_estimators are number of forest, one would expect that high n_estimator (more decision trees) values corresponds to higher accuracy and better performance of the model. This is exactly what I observed, where the accuracy increases with increasing n_estimators number and accuracy saturates over higher values above 50. There is no big difference in the performance for the two criterions.

#### b. max_depth vs accuracy

In figure 13b, max_depth is tuned from 1 to 20. Accuracy is increasing dramatically with the increasing max_depth value and saturates after 15. This is expected as max_depth parameter changes the depth of each tree which directly correlates to the model performance. This is no much difference in the performance for the two criterions.

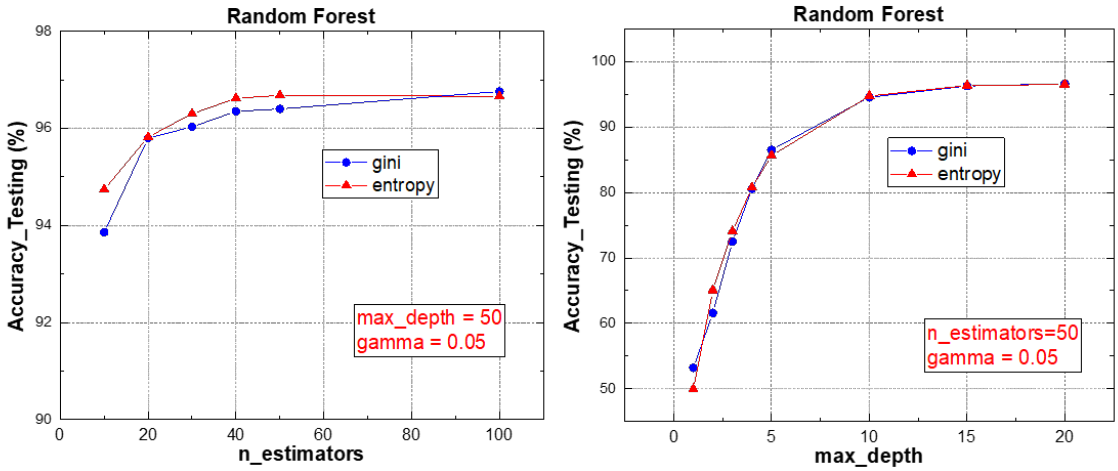


Figure 13. a) n_estimators vs accuracy for MNIST testing data (left), b) max_depth vs accuracy.

#### c. Confusion matrix

Confusion matrix for random forest model is shown in figure 14. This model performs worst with classification of number 8 and 9 as compared with other numbers. Number 9 is classified as number 4 for 1.4 % of the total data.

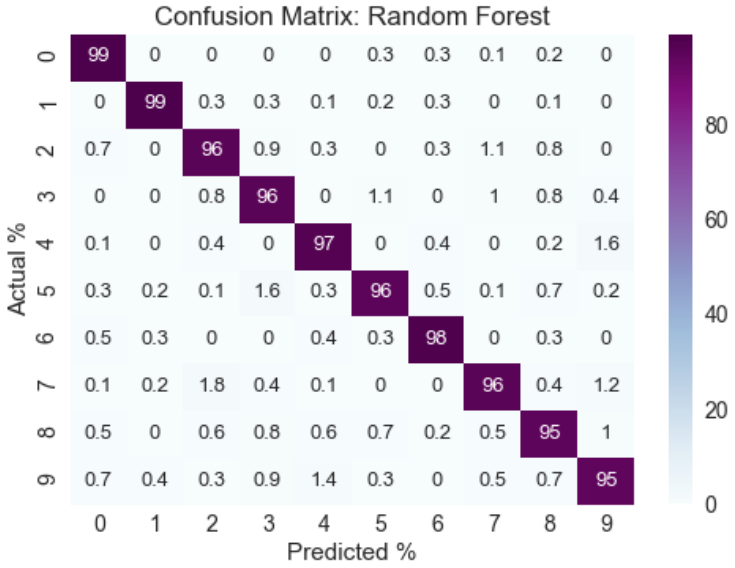


Figure 14. Random forest confusion matrix

### 4.5 Bootstrap aggregation: Bagging

Bagging is implemented for the four classifier models. For bagging, subsamples of data used for training each model are randomly created from the MNIST training dataset. Size of each subsample of is set as 60% of the original training data, and sample data are randomly picked from the original data with replacement. The boosting algorithm is written from scratch.

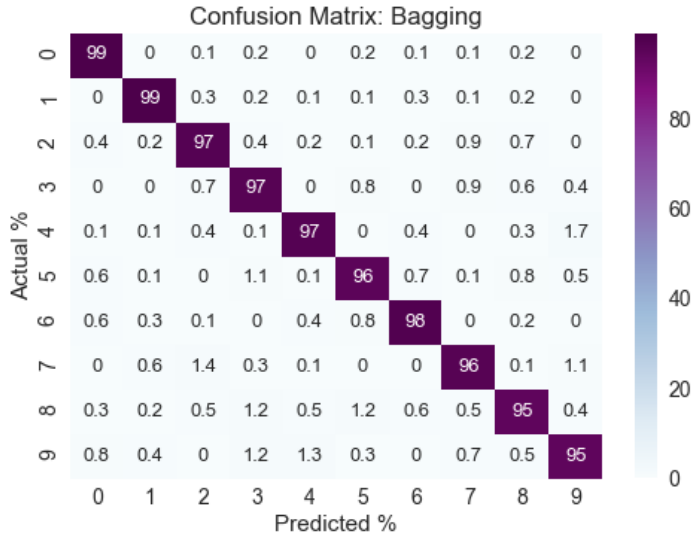


Figure 15. confusion matrix from bagging algorithm

After creating the subsample of data for each model, they are used to train all models independently and their predicted output values are compared. Majority vote is used to compare and give the final output. For example, for class number 1, if 3 models predict as 1 and one model predict as 7, the majority vote decides the output as 1 instead of 7.

This improves the overall performance. The accuracy of Bagging algorithm is about 96.78%. Confusion matrix out of bagging algorithm is plotted in figure 15.

### 4.6 Boosting Algorithm

As an additional experiment, boosting algorithm is tested on random forest classifier. Scikit-learn package AdaBoostClassifier is used for boosting. The prediction from boosting is about 96.88 %, not much different from just random forest model accuracy. From the confusion matrix plot, the algorithm has more difficulty classifying number 9 and 5 compared to others. Number 9 is classified as 3 and 4 more often than others.

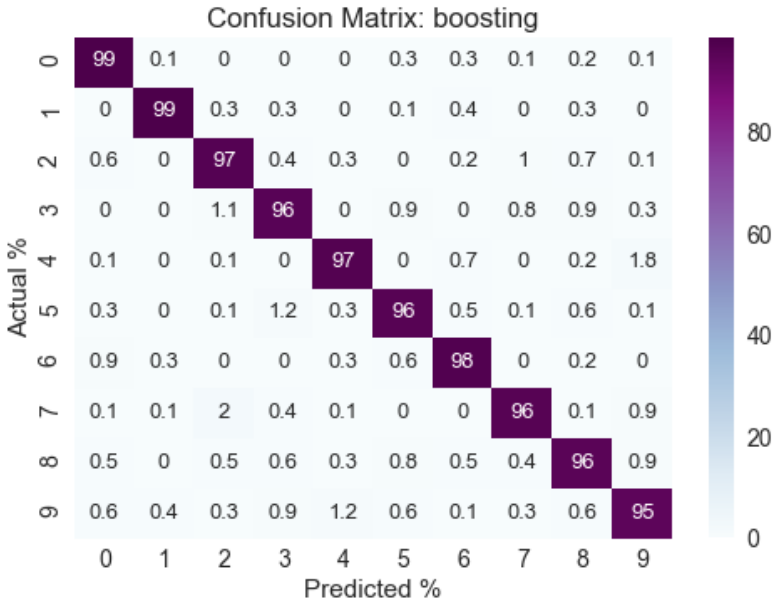


Figure 16. confusion matrix for boosting algorith

## 5 Conclusion

To conclude, four different machine learning models are used to classify MNIST and USPS handwritten digit from 0 to 9. The models were trained on MNIST training dataset and tested on MNIST testing data as well as USPS dataset. No free lunch was proved as performance of the models trained on MNIST training data is very poor when tested on USPS dataset. Additional algorithms such as bagging and boosting is also implemented.