Project 3: Classification

Arinjoy Bhattacharya
Department of Physics
University at Buffalo, SUNY
Buffalo, NY 14214
Person # 50168806
arinjoyb@buffalo.edu

Abstract

Classification is an important tool for machine learning. It is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known. Through this project, various methods of classification are tested with the help on MNIST and USPS dataset.

1 Introduction

Logistic regression is the easiest form of classification among all the other classifiers. But it doesn't yield so much accuracy. Also, "no free lunch theorem" says that if a model is trained with some classifier is doesn't work good with another one. These will be verified in this project.

2 Model

2.1 Model: Dataset

The task is to recognize a 28×28 grayscale handwritten digit image and identify it as a digit among 0, 1, 2, ..., 9. The data is extracted from two data-sets.:

- MNIST Dataset: The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

The database contains 60,000 training images and 10,000 testing images. The original black and white (bi-level) images from MNIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

- USPS Dataset: This dataset is only used for testing purposes for all the models. Training is done only with MNIST.

Each digit has 2000 samples available for testing. These are segmented images scanned at a resolution of 100ppi and cropped.

2.2 Model: Partitioning the dataset

Like in any other Machine Learning problem, we implement the method of training, validation and testing. So we partition the dataset into three parts:

- 80% training
- 10% validation
- 10% testing

These three datasets are distinct and don't overlap with each other.

2.3 Model: Multiclass Logistic regression model

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

2.3.1 Model: Softmax function definition

For 10 classes, soft-max function is used where input is $\phi = [\phi_1 \ldots, \phi_M]^T$.
The multiclass logistic regression model could be represented in the form,

$$p(C_k|x) = y_k(x) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

where the activation a_k are given by $a_k = w^T x + b_k$.

2.3.2 Model: Cross-entropy error function definition

To generate the probability function, the weights from each cases are taken and feed it into the softmax function to generate the probability class. But as can be observed, the Mean square error function can't be used here since the probability function is nonlinear. Hence, a better cost function needs to be implemented.

The likelihood of observations here is given by:

$$p(T|w_1, \ldots\ldots, w_{10}) = \prod_{n=1}^{N} \prod_{k=1}^{10} y_{nk}^{t_{nk}}$$

So, our error function is generated by the negative log of the likelihood function

$$E(w_1, \ldots\ldots, w_{10}) = -\ln p(T|w_1, \ldots\ldots, w_{10}) = \sum_{n=1}^{N} \sum_{k=1}^{N} y_{nk}^{t_{nk}}$$

The gradient of the error determines how the weights are updated for the model.

$$\nabla_{w_j} E(w_1, \ldots\ldots, w_k) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\phi(x_n)$$

Updating the weights,

$$w^{\tau+1} = w^{\tau} - \eta \nabla E_n$$

2.3.3 Model: One-hot vector notation

The target vectors are converted to one-hot vector notation (binary value of dimension 10) so as to implement multiclass logistic regression.

2.4 Model: Neural Networks

Neural networks are trained iteratively using optimization techniques like gradient descent. After each cycle of training, an error metric is calculated based on the difference between prediction and target. The derivatives of this error metric are calculated and propagated back through the network using a technique called backpropagation. Each neuron's coefficients (weights) are then adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold.

2.4.1 Model: Neurons

The inputs to each node are combined using a weighted linear combination. The result is then modified by a nonlinear function before being output. Mathematically, it can be shown by the equation.

$$z_j = b_j + \sum_{i=1}^{n} w_{i,j} x_i$$

, where $w_{i,j}$ represents the weight vector, x_i represent the input vector from each neuron and z_j gives the values in the hidden layer.

2.4.2 Model: Layers

There are mostly speaking three layers in the network, the input layer, the hidden layer and the output layer. The provided code has two hidden layers, so essentially speaking there are total of four layers. In the hidden layer, this is then modified using a nonlinear function such as a relu,

$$f(x) = x^+ = \max(0, x)$$

to give the input for the next layer. This tends to reduce the effect of extreme input values, thus making the network somewhat potent to arbitrary values. The parameters $b_1, b_2, b_3, w_{1,2}$ ... are "learned" by training the data. The values of the weights are often confined to prevent them from blowing up. The parameter that restricts the weights in the code is known as the 'learning rate', which varies from 0.1 to 1.

2.4.3 Model: Weights

The weights take arbitrary values to commence with, and these are then updated utilizing the observed data. Thereupon, there's a part of unpredictability within the predictions made by a neural network. Therefore, the network is typically trained many times considering totally different random beginning points, and also the results area unit averaged. The number of hidden layers, and the number of neurons in each hidden layer, must be specified in advance.

2.4.4 Model: Activation functions

Activation functions takes the input from the hidden layer and modify the values to tune the model. Activation functions give neural networks to model complex non-linear relationships. By modifying inputs with non-linear functions neural networks can model highly complex relationships between features. Popular activation functions include relu and leaky relu.

2.4.5 Model: Loss functions

After the hidden layer, a loss function is implemented to check how well the model performs for a particular set of parameters. In the provided code, it is a cross entropy function whose slope gives us how the model actually is performing. So, depending on the performance of the model, the parameters are modified by the loss function.

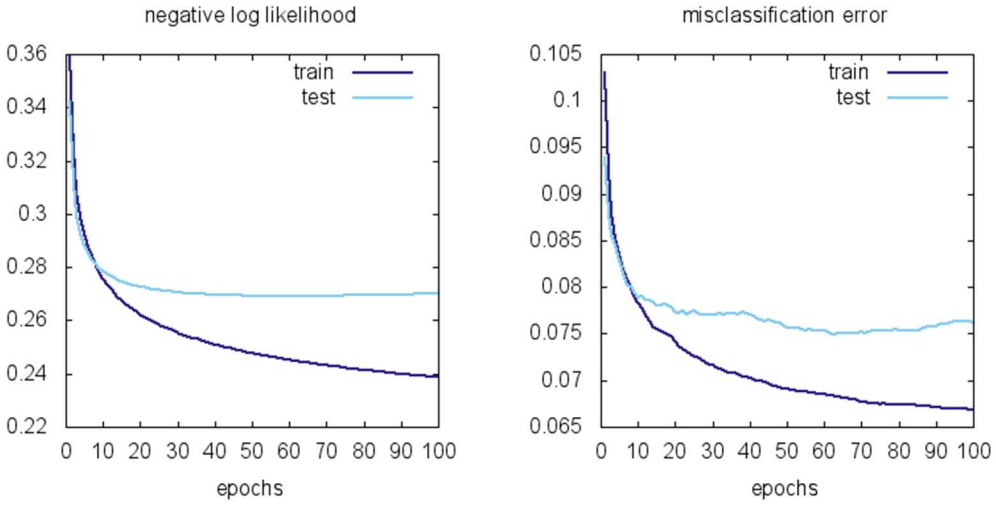

Figure 1: Range of possible loss values for cross entropy for a particular set of epochs.

2.4.6 Model: Optimization Methods

Optimization algorithms are used to back process the model and reduce the error in accuracy rate of the model. The method used in the provided code is gradient descent. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

$$\hat{C} = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2$$

2.4.7 Model: Forward and Backward Propagation

Essentially speaking, the two methods of propagation are complimentary to one another. The forward propagation involves calculating weights and multiplying them with the input, and finally using the activation function to pass it to the output layer. In case of backpropagation each weight in the network are adjusted in proportion to how much it contributes to overall error. The error function if differentiated with respect to particular hyperparameters and equated to zero to get our best probable accuracy.

2.5 Model: Support Vector Machine (SVM)

Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, when a labeled training data (supervised learning) is provided, the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

2.5.1 Model: Tuning parameters: Kernel functions

For SVM there are four sets of Kernel functions which are being tuned: “radial basis functions”, “polynomial of degrees 3,4,5”, “sigmoid” and “linear”.
Each of these functions have a different method of creating the hyperplane for the classification.

2.5.2 Model: Tuning parameters: c

This parameter “c” acts like the regularizer in linear regression. It serves as a tradeoff

between classification error and learning rate.
Depending on the value of c, we can estimate the decision boundary or what is called before as “hyperplane”. It also determines whether an SVM is soft margin or hard margin. In SVM our goal is to maximize the margin width which is inversely proportional to square of weights, so the goal changes to minimize sum of square of weights. With regularization, the formula is

$$\sum_i w_i^2 + c \sum_i \epsilon^2$$

For large c, the target point is very close to the decision boundary and for small c, they are far apart.

2.5.3 Model: Tuning parameters: Gamma (γ)

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible separation line are considered in calculation for the separation line, whereas high gamma means the points close to plausible line are considered in calculation. Mathematically speaking, it is the inverse of variance.

$$\phi(x) = \exp(-\gamma\ (v_1 - v_2)^2)$$

where $\phi(x)$ is the radial basis function and v_1 and v_2 are two points of consideration.

2.6 Model: Random Forest Classifier

Like the name suggest, this classifier creates a forest and makes it somehow random. The “forest” it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems.

2.6.1 Model: Tuning parameters: n_estimators

In easy terms, it just determines the number of trees in the forest. Thus, ideally higher numbers should give better performance and prevent overfitting too.

2.6.2 Model: Tuning parameters: criterion

It can be tuned to either “Gini” or “entropy”.
The mean decrease in Gini coefficient is a measure of how each variable contributes to the homogeneity of the nodes and leaves in the resulting random forest.
The Gini and Entropy functions are given as

$$Gini(t) = 1 - \sum_j [p(j|t)^2]\,, where\ the\ quantity\ in\ parantheses\ is\ probability$$

$$Entropy = -\sum_j [p(j|t)] \log(p(j|t))$$

2.6.3 Model: Tuning parameters: max_depth

This determines the max depth till which the decision trees can go up to. For example, is the max depth is 5, then each branches of the decision tree will split three times to determine its children nodes.

2.7 Model: Ensemble Classifier

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).
Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type leading to homogeneous ensembles. There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to heterogeneous ensembles. In order for ensemble methods to be more accurate than any of its individual members the base learners have to be as accurate as possible and as diverse as possible.

2.7.1 Model: Bagging

Bagging stands for bootstrap aggregation. One way to reduce the variance of an estimate is to average together multiple estimates. For example, we can train M different trees f_m on different subsets of the data (chosen randomly with replacement) and compute the ensemble.

$$f(x) = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$$

Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses majority voting for classification and averaging for regression

2.7.2 Model: Boosting

Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners (models that are only slightly better than random guessing, such as small decision trees) to weighted versions of the data, where more weight is given to examples that were mis-classified by earlier rounds. The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction. The principal difference between boosting and the committee methods such as bagging is that base learners are trained in sequence on a weighted version of the data.

2.7.3 Model: Stacking

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on complete training set then the meta-model is trained on the outputs of base level model as features. The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous.

3 Experiment

For the experiment, the logistic regression model was trained with the various hyperparameters and are sequentially explained with plots below. Similarly, the neural network, SVM and Random forest was also trained as given below. Ensemble classification was also implemented and shown. Confusion matrices for each of them are also illustrated below.

3.1 Experiment: Logistic Regression: different values of learning rate.

In case of Logistic, we measure accuracy to check whether our model is working or not.
From the data which we have, and the results which we obtained we can easily say a logistic regression model is better suited to fit our data than linear regressionActually, it is basically like a bell-shaped. Trying out various learning rates, we basically find out a local maximum value of learning rate where the accuracy is maximum. In our case, it's around a value of 1.

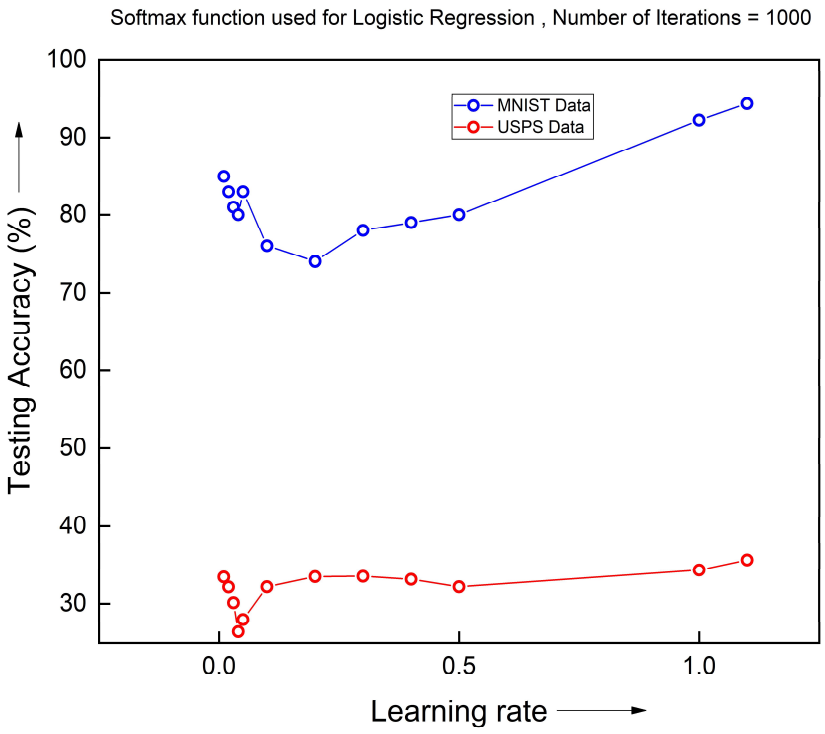

Figure 2: Testing Accuracy Vs different values of learning rate (η)

3.2 Experiment: Logistic Regression: number of iterations.

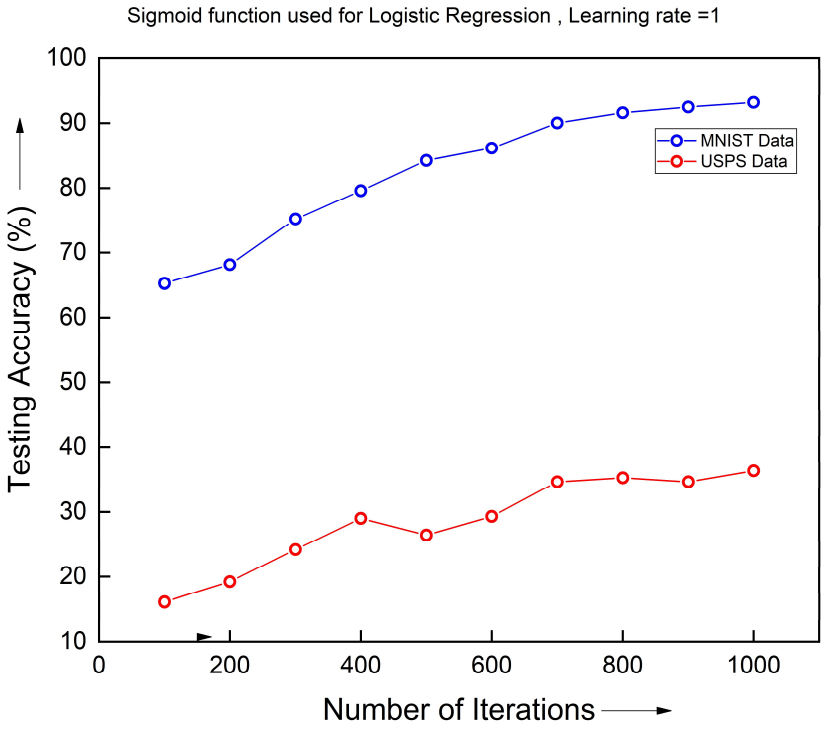

Figure 3: Testing Accuracy Vs different values of iterations

The number of iterations is directly proportional to the accuracy.

3.3 Experiment: Logistic Regression: confusion matrices.

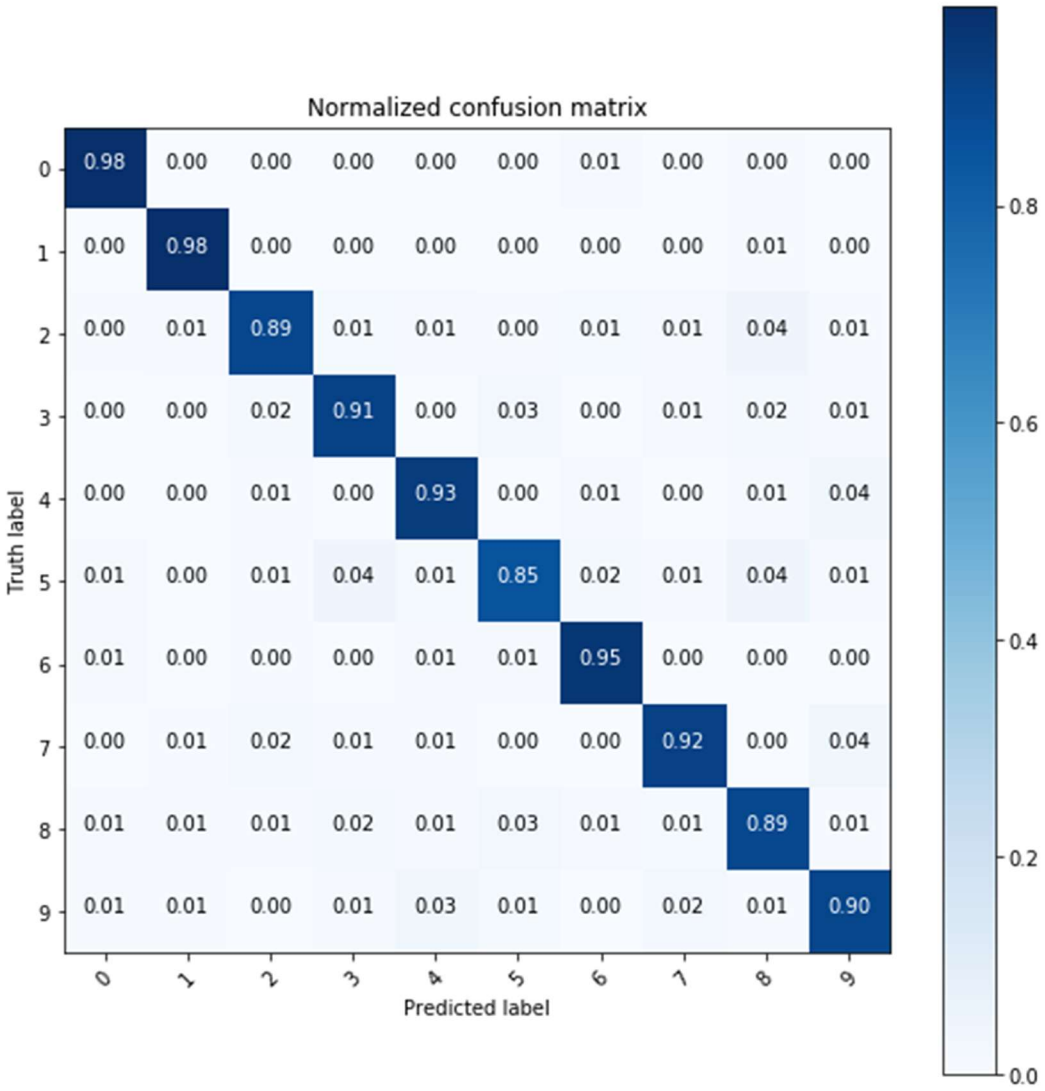

Figure 4: MNIST data confusion matrix for logistic regression.

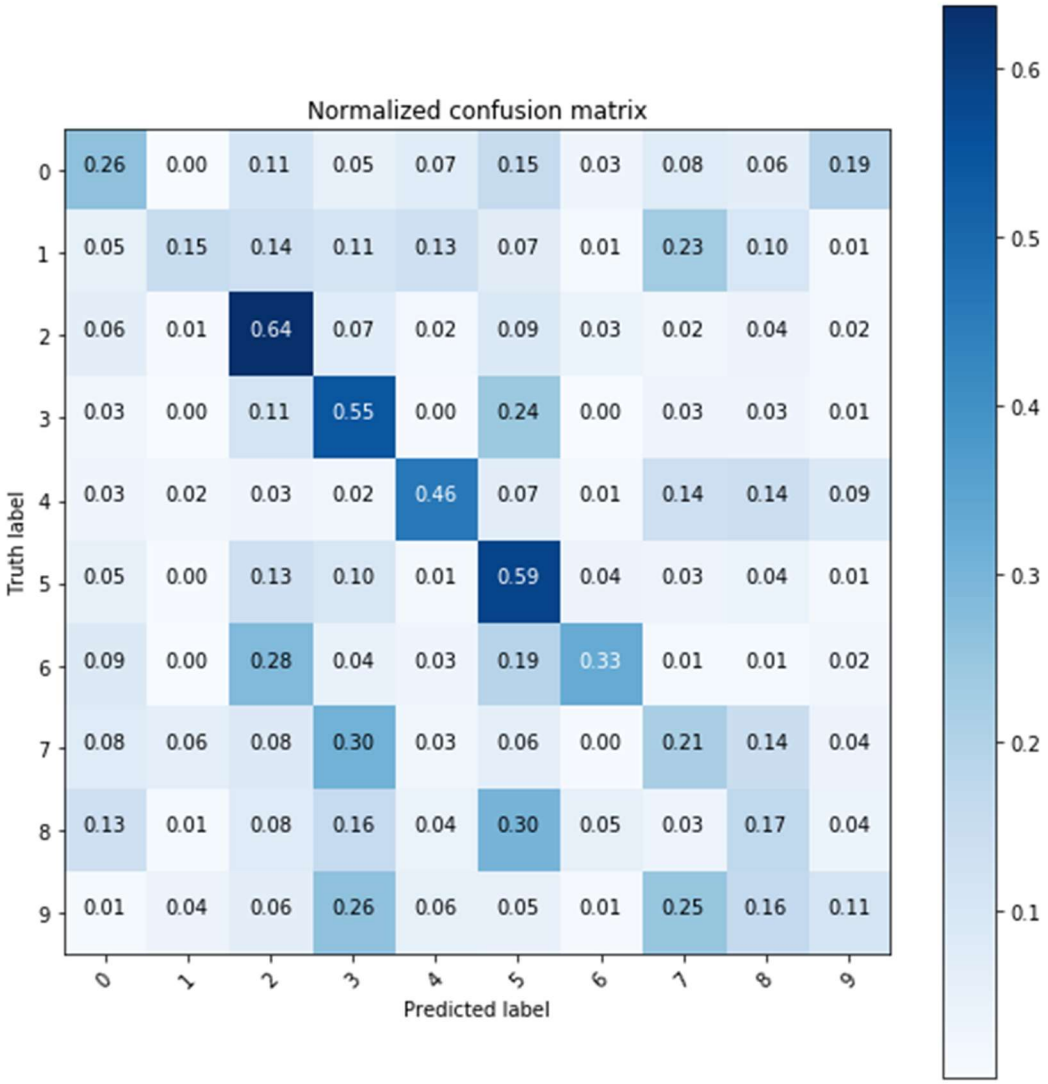

Figure 5: USPS data confusion matrix for logistic regression.

The heat map clearly illustrates the results of "no free lunch theorem". Since the model was trained on MNIST, a good accuracy heat map was obtained for it but unfortunately for USPS, it wasn't performing too good.

3.4 Experiment: Neural Networks: number of epochs.

The number of epochs is directly proportional to the accuracy. The common reason might be because with more epochs, the model can be trained better and hence prevents overfitting or underfitting.

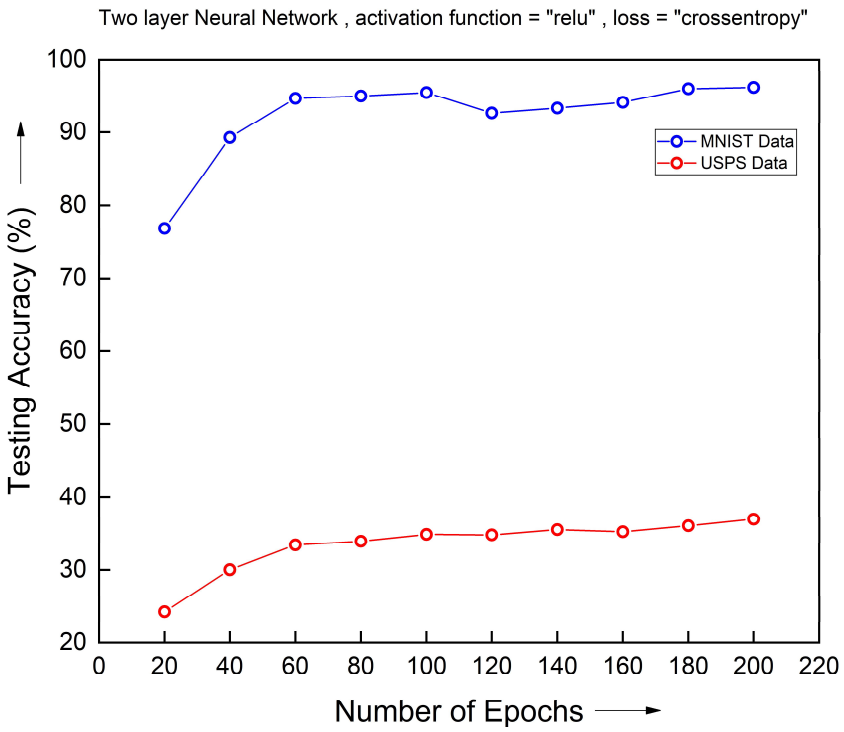

Figure 6: Testing Accuracy Vs different values of epochs

3.5 Experiment: Neural Networks: number of neurons.

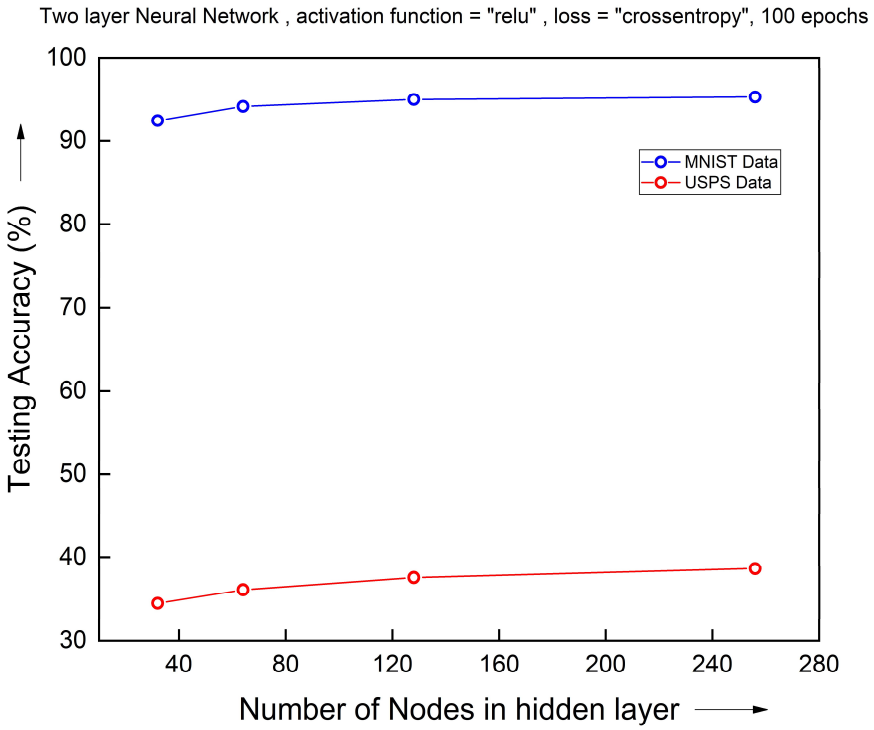

Figure 7: Testing Accuracy Vs different values of nodes in hidden layer

The number of nodes is directly proportional to the accuracy. This might be because of the fact that it passes through the activation function multiple times thereby training better.

3.5 Experiment: Neural Networks: activation function.

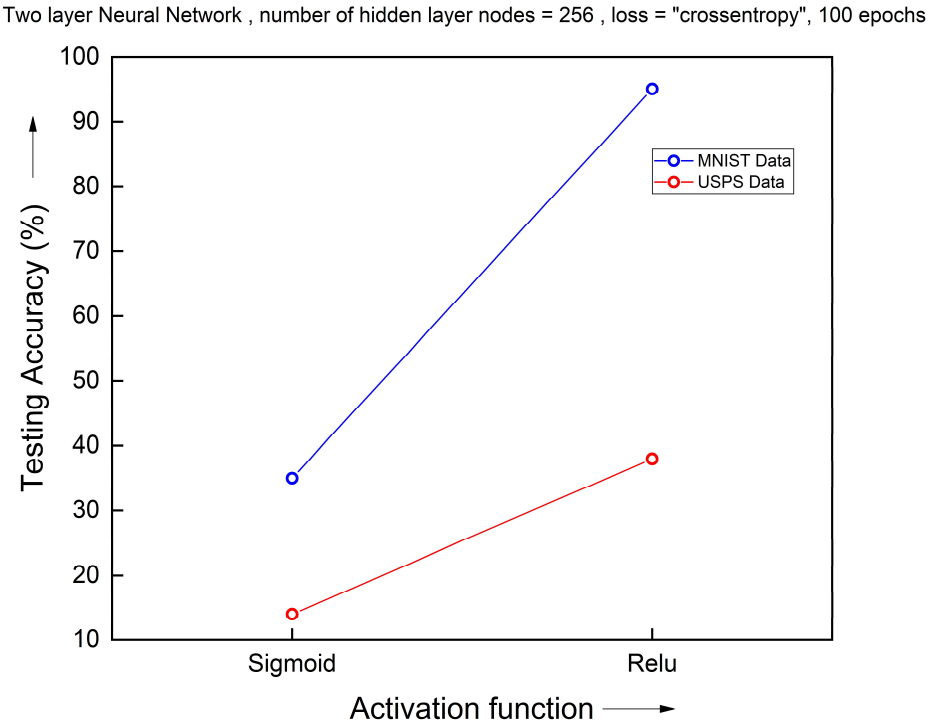

Figure 8: Testing Accuracy Vs different activation functions

The network performs way better with relu than sigmoid.

3.6 Experiment: Neural Networks: Confusion matrix.

The heat map clearly illustrates the results of “no free lunch theorem”. Since the model was trained on MNIST, a good accuracy heat map was obtained for it but unfortunately for USPS, it wasn’t performing too good.

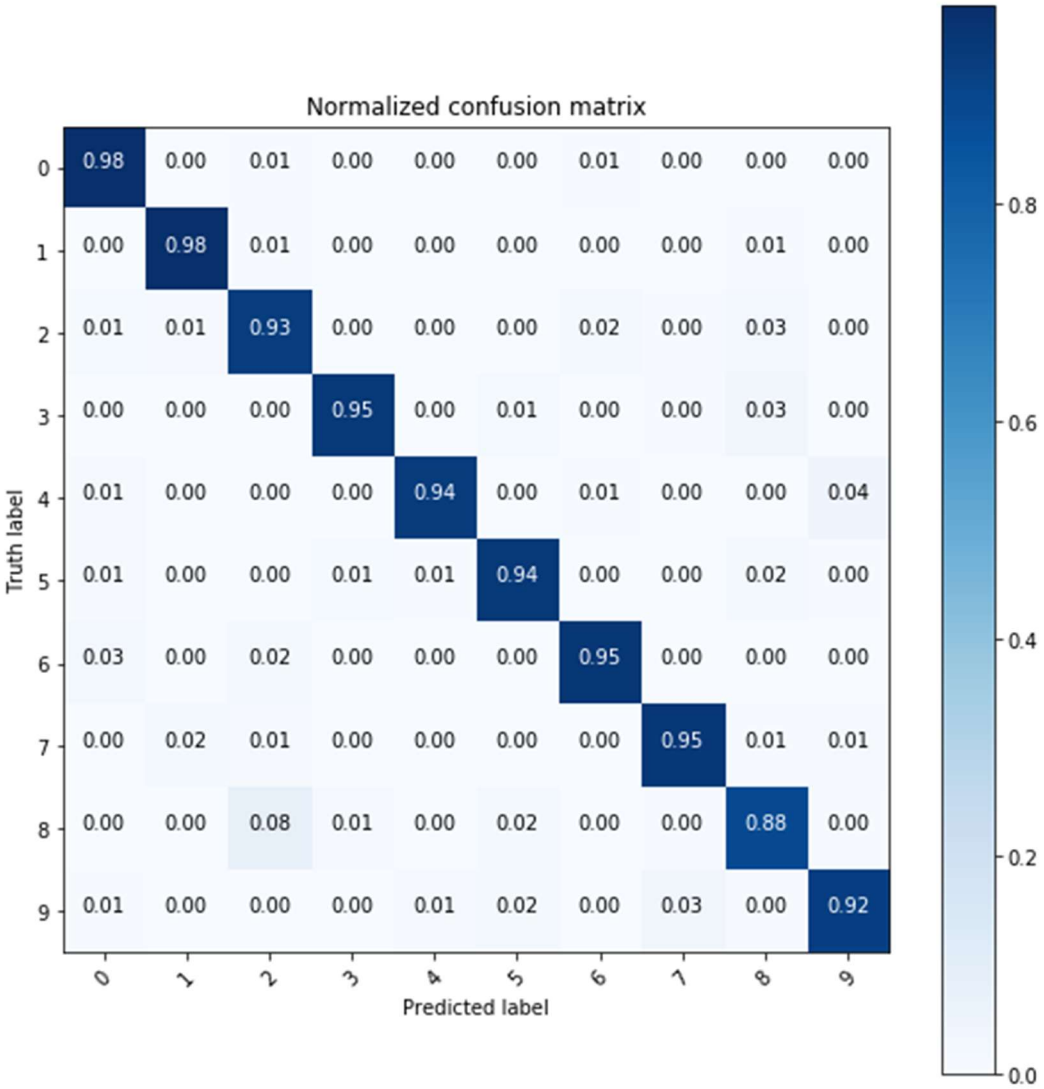

Figure 9: MNIST data confusion matrix for neural network.

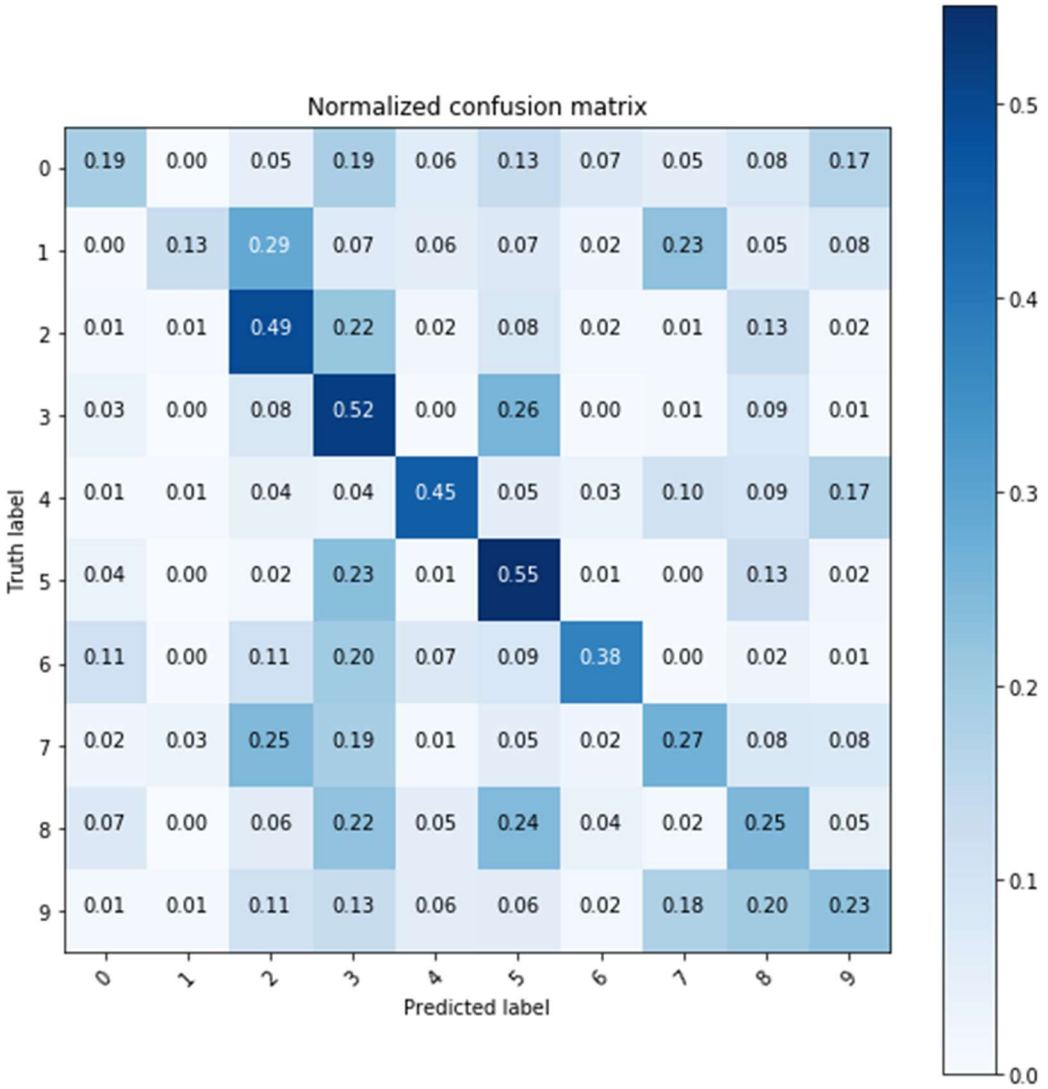

Figure 10: USPS data confusion matrix for neural network

3.7 Experiment: SVM: kernel function.

Linear kernel is way better than any other kernel followed by polynomial of increasing order and then is sigmoid and then radial basis funcitons.

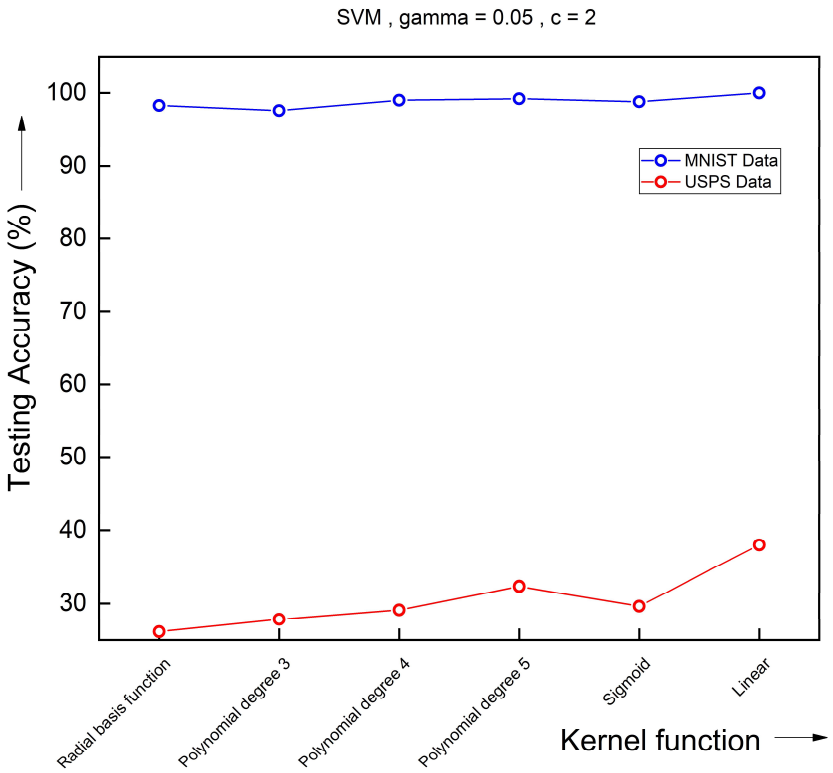

Figure 11: Testing Accuracy Vs different kernel functions

3.8 Experiment: SVM: parameter "c".

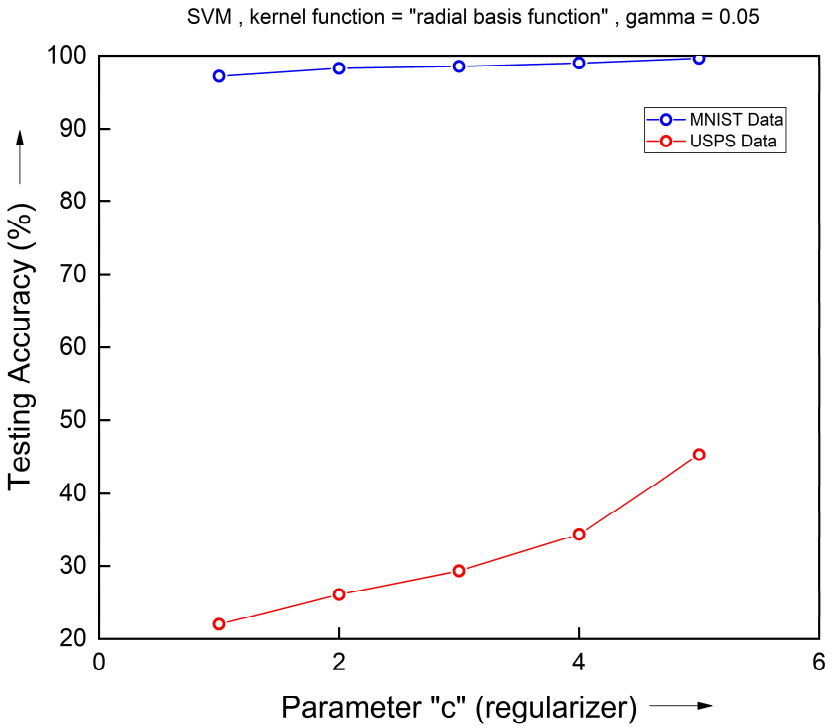

Figure 12: Testing Accuracy Vs different parameter “c” values

The value of c determines the decision boundary. So, higher c means closer to the decision boundary and hence can incorporate less points so more accuracy.

3.9 Experiment: SVM: parameter “γ”.

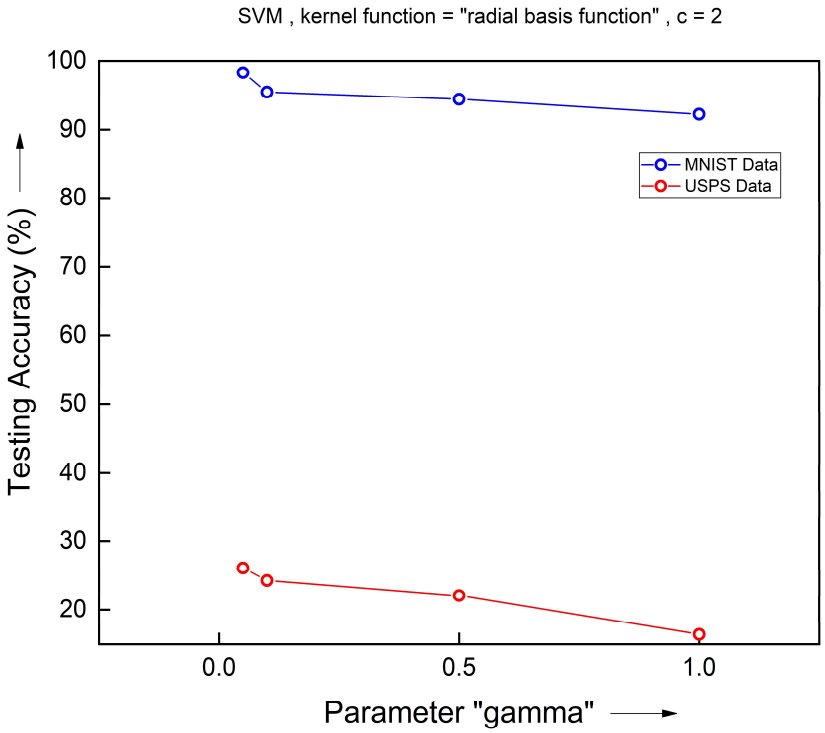

Figure 13: Testing Accuracy Vs different parameter “γ” values

The value of γ determines the spread of the function. Large γ means more constricted and hence lesser accuracy.

3.10 Experiment: SVM: Confusion matrix.

The heat map clearly illustrates the results of “no free lunch theorem”. Since the model was trained on MNIST, a good accuracy heat map was obtained for it but unfortunately for USPS, it wasn’t performing too good.

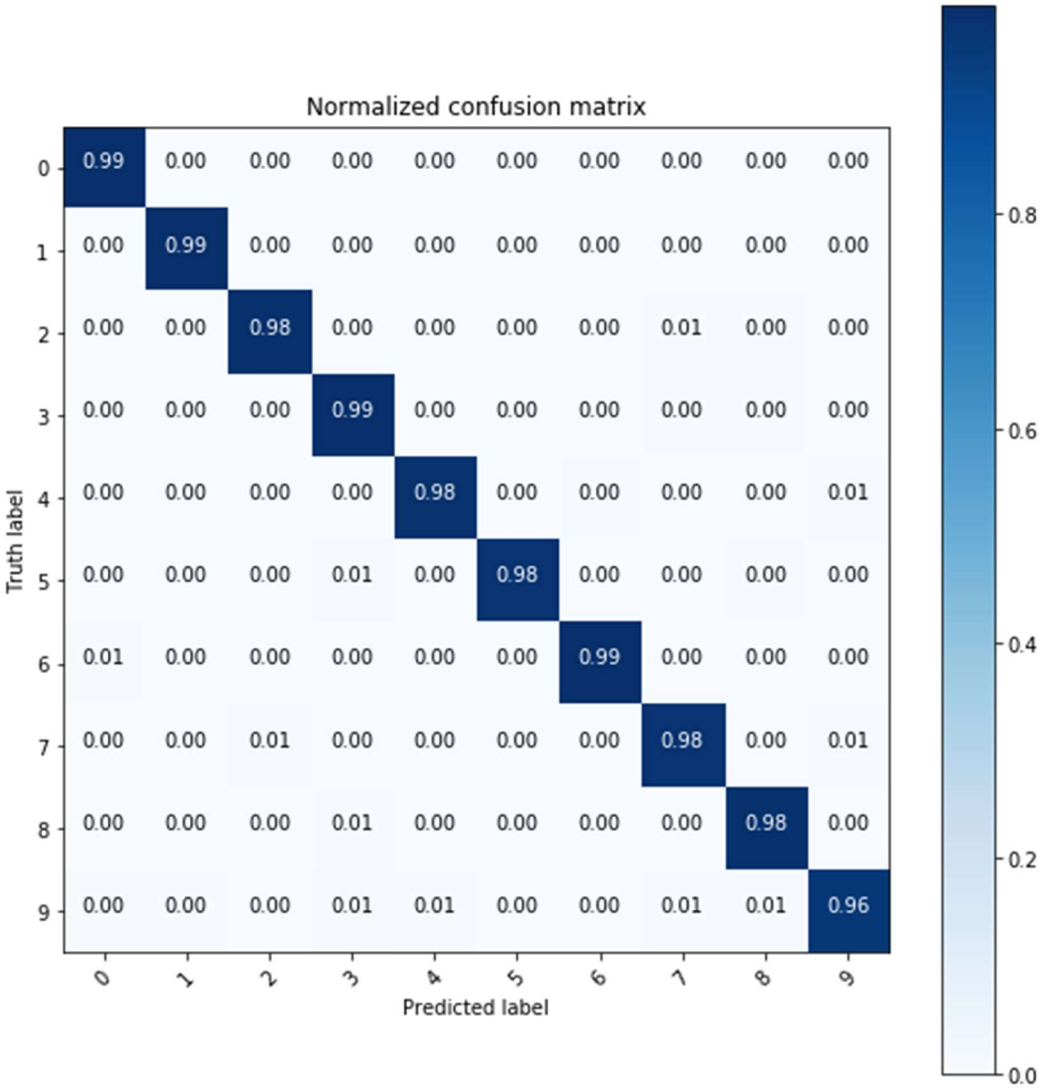

Figure 14: MNIST data confusion matrix for SVM

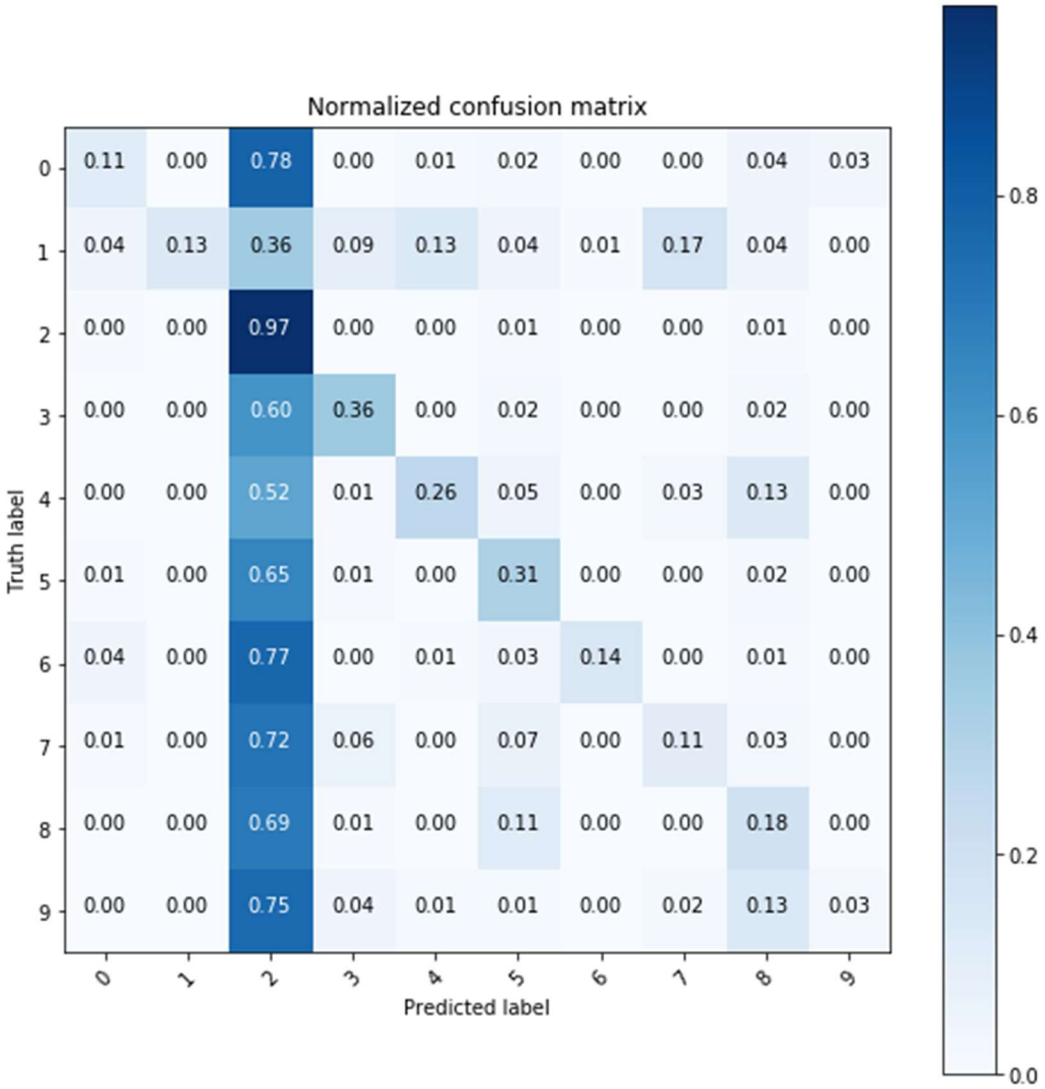

Figure 15: USPS data confusion matrix for SVM

3.11 Experiment: Random forest: parameter "n_estimators".

The value of n_estimators give the number of trees. So, more trees mean more classification means more accuracy.

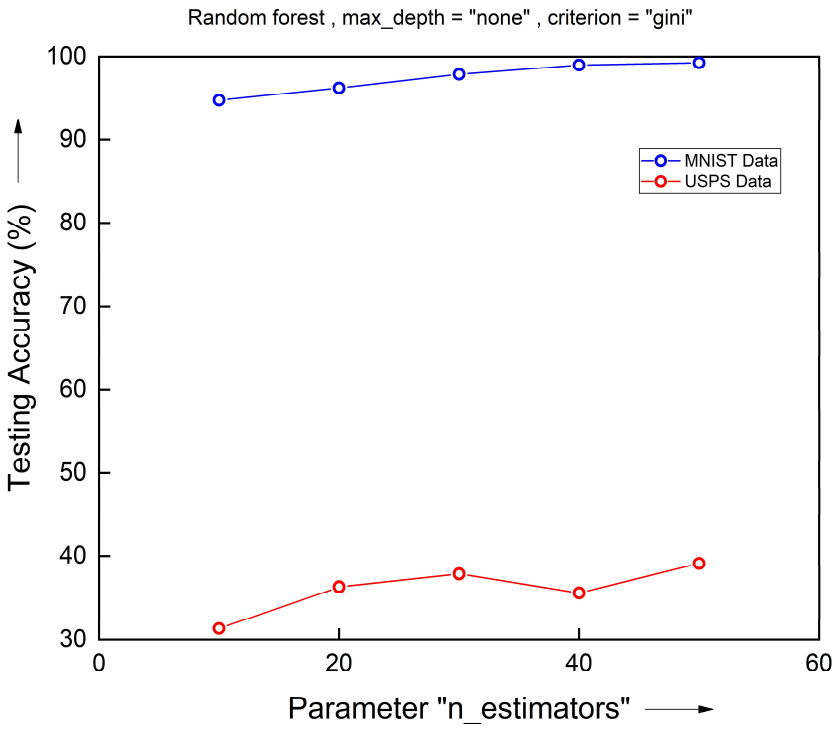

Figure 16: Testing Accuracy Vs different parameter “n_estimatros” values

3.12 Experiment: Random Forest: parameter “criterion”.

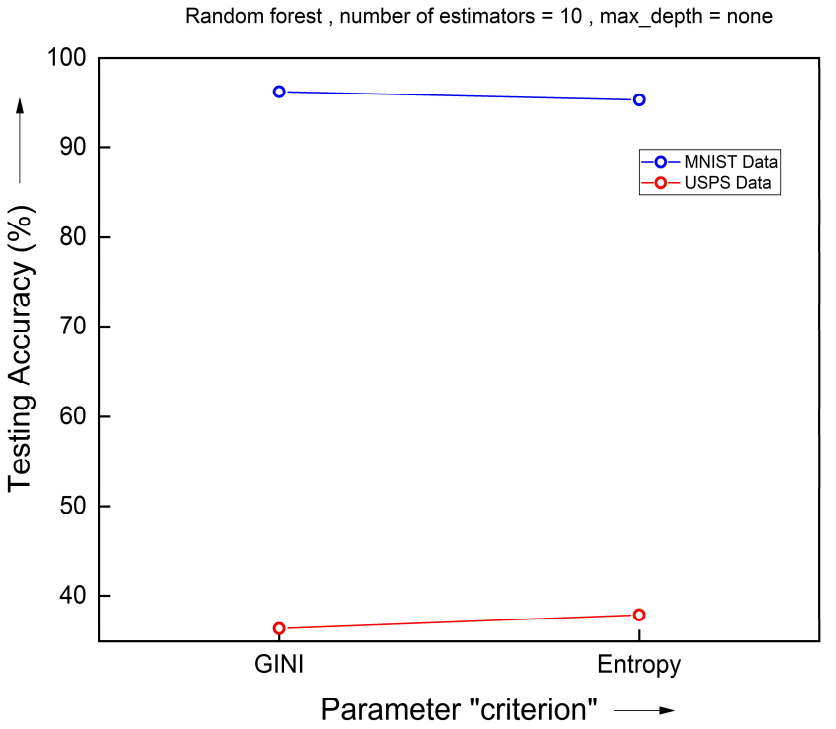

Figure 17: Testing Accuracy Vs different criterion parameter

The network performs way better with GINI than entropy.

3.13 Experiment: Random Forest: parameter "max depth".

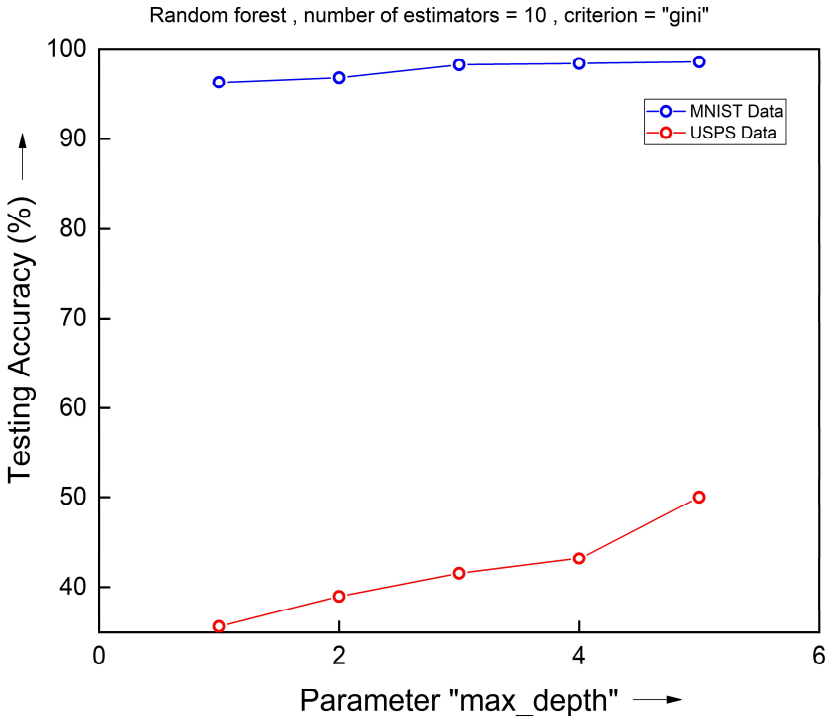

Figure 18: Testing Accuracy Vs different "max depth" parameter values

With more max depth of random forest there is more accuracy since there is more control over the complexity of the decision trees and hence less overfitting.

3.14 Experiment: Random forest: Confusion matrix.

The heat map clearly illustrates the results of "no free lunch theorem". Since the model was trained on MNIST, a good accuracy heat map was obtained for it but unfortunately for USPS, it wasn't performing too good.

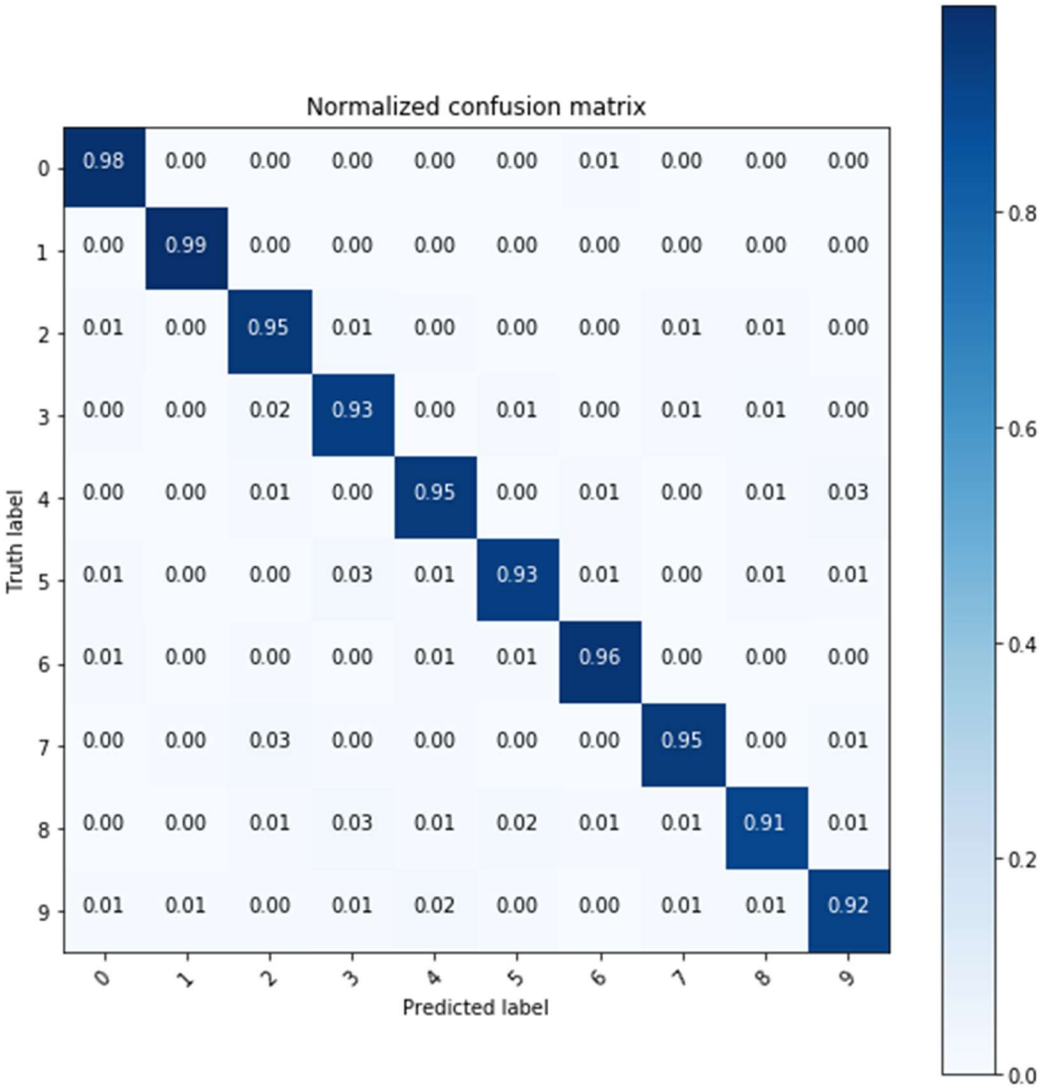

Figure 19: MNIST data confusion matrix for Random Forest

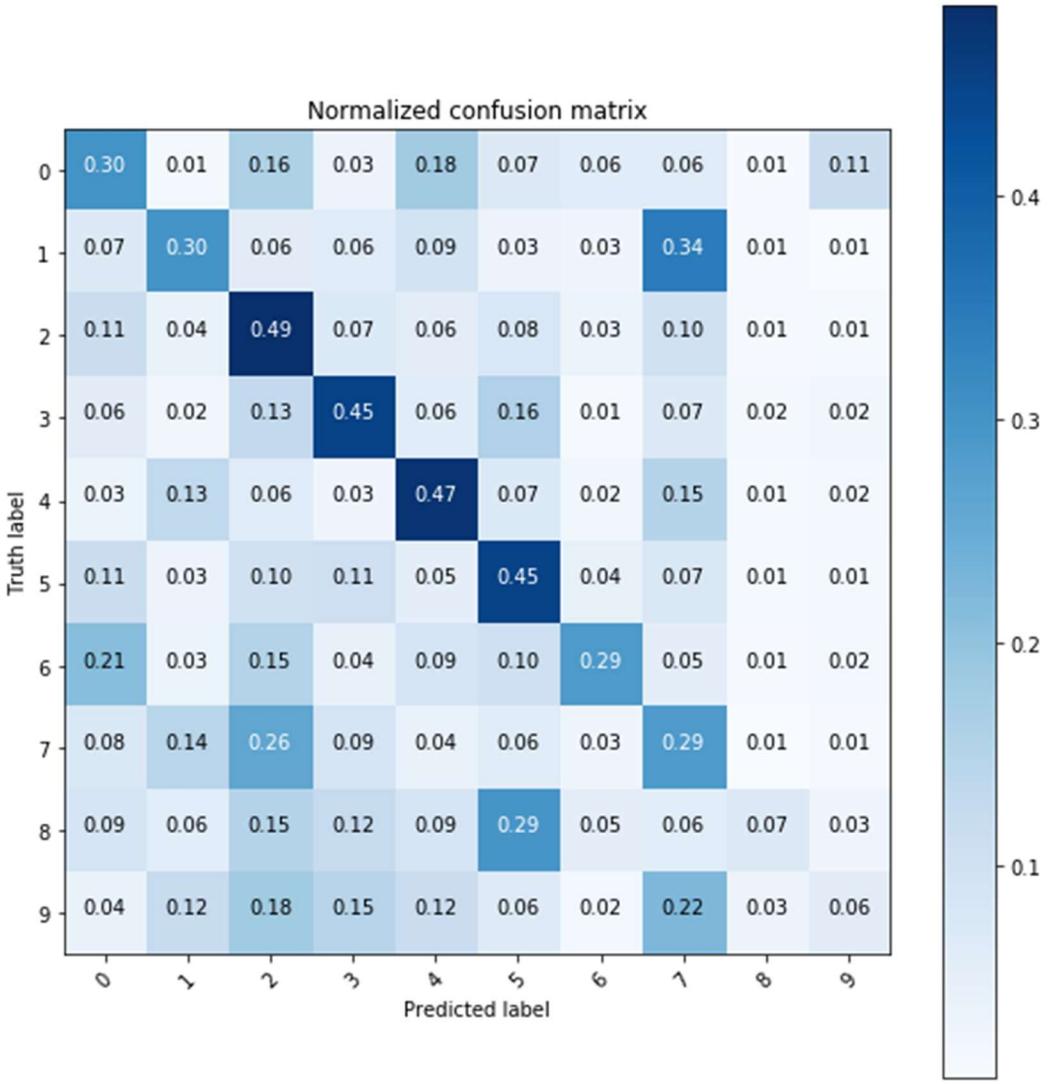

Figure 20: USPS data confusion matrix for Random Forest

3.15 Experiment: Bagging: Confusion matrix.

The heat map clearly illustrates the results of “no free lunch theorem”. Since the model was trained on MNIST, a good accuracy heat map was obtained for it but unfortunately for USPS, it wasn’t performing too good.

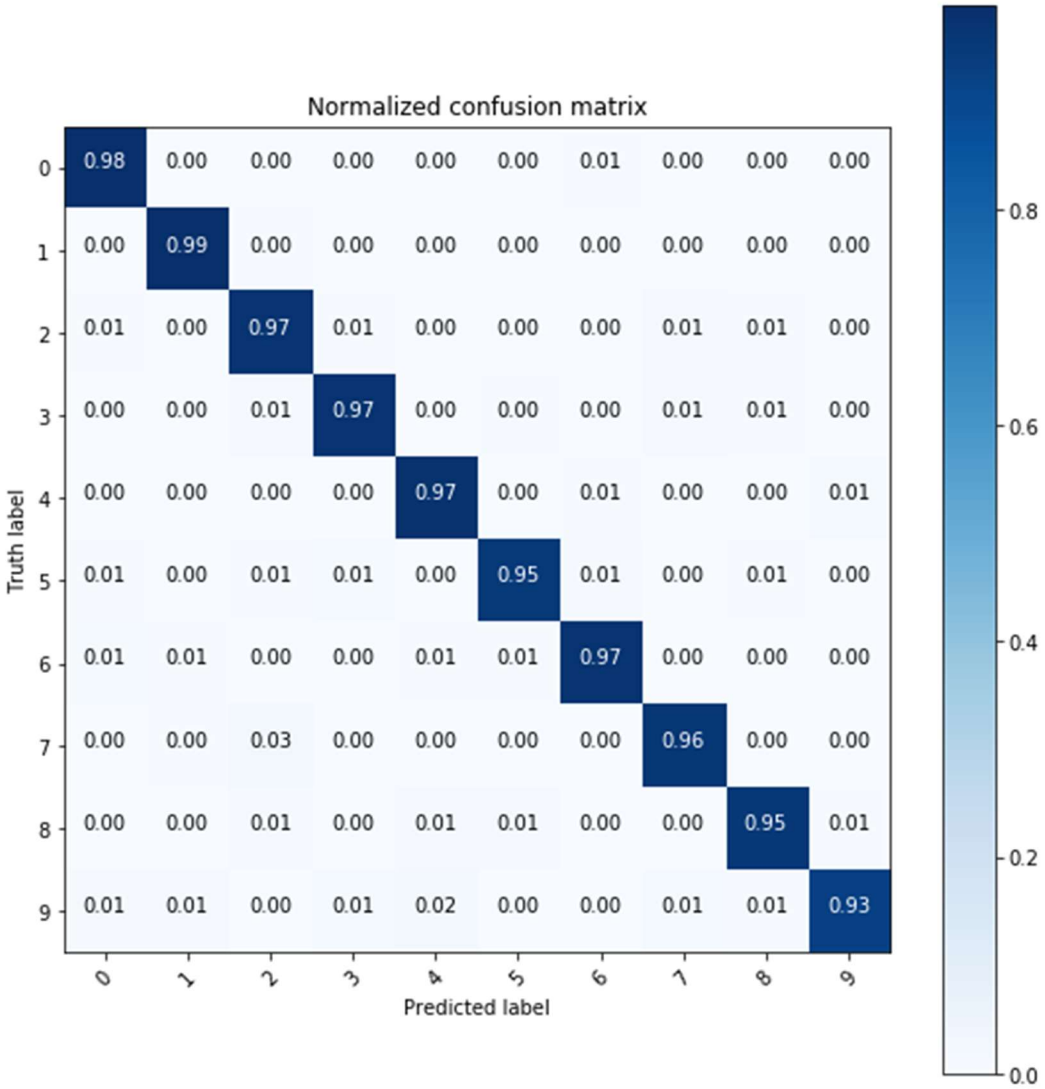

Figure 21: MNIST data confusion matrix for Bagging

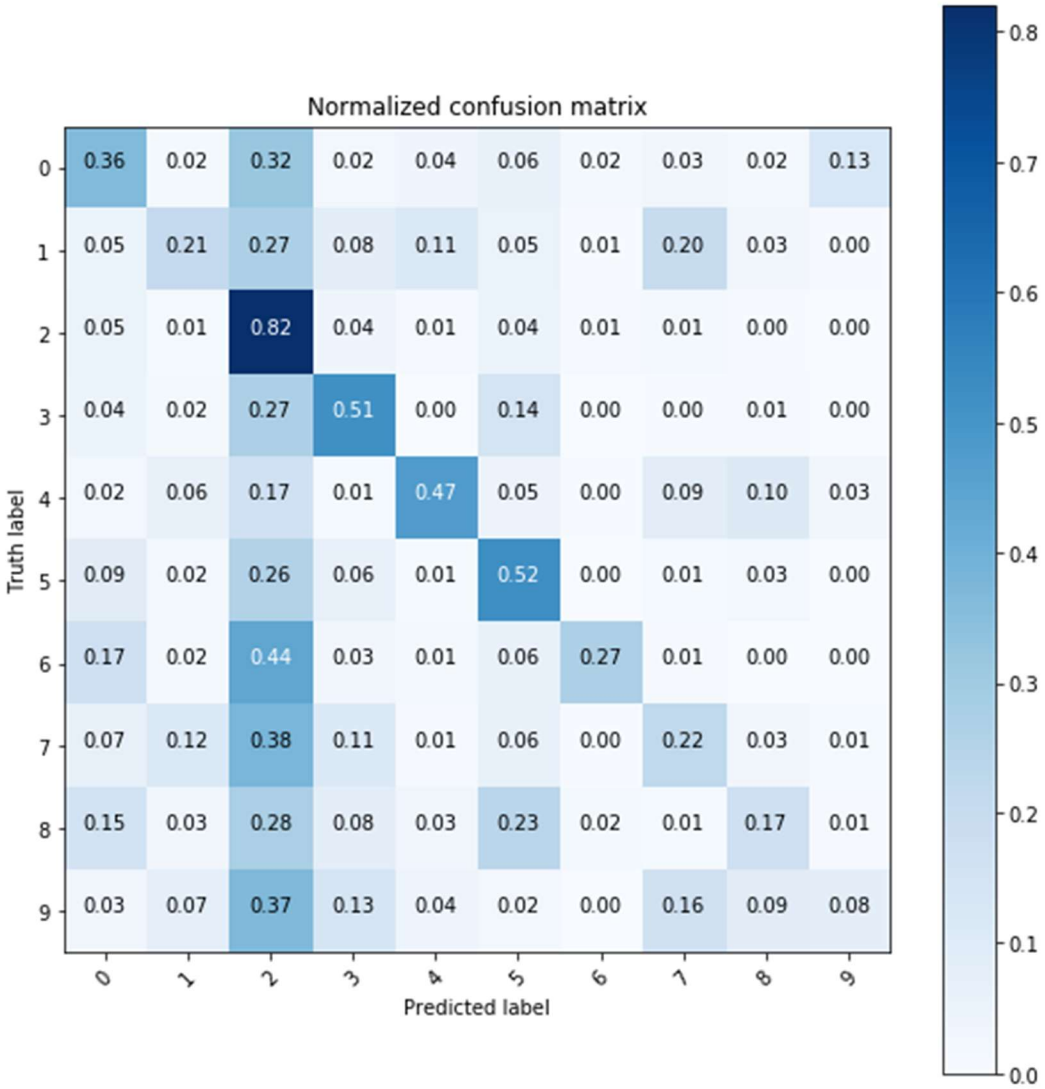

Figure 22: USPS data confusion matrix for Bagging

3.16 Experiment: Boosting and Stacking.

Boosting is implemented with base classifier as Random Forest and the performance was stellar. MNIST gave 96.44% and USPS gave 86.5%

Stacking of "K-Nearest Neighbours", "Random Forest", "Naïve Bayesian" classifiers with logistic regression as the meta classifier was implemented by the accuracy wasn't too good.

3.17 Experiment: the perfect parameter

For the highest accuracy, the following parameters seemed proper for Logistic Regression:

- Learning rate of 1.
- Number of iterations in the SGD calculation = 2000.
- Softmax for updating the weights.

For the highest accuracy, the following parameters seemed proper for Neural Networks:

- 256 or more nodes in each layer.
- Number of epochs = 5000.

For the highest accuracy, the following parameters seemed proper for SVM:

- Parameter “c” to be 5 or higher.
- Parameter “γ” to be 0.01 or lower.
- Linear kernel function.

For the highest accuracy, the following parameters seemed proper for Random forest:

- Number of trees to be 50 or more.
- Parameter “criterion” to be GINI.
- Parameter “max depth” to be 5 or more.

4 Conclusion

Average accuracy for MNIST was around 92-99% but that for USPS was 35-40%. Hence the “no free lunch theorem” was verified.

Acknowledgments

I thank the professor Dr. Srihari and the TAs for helping me complete this report.

References

[1] CM Bishop, TM Mitchell. (2014) *Pattern Recognition and Machine Learning*, Springer.