Project 3: Handwritten Digit Recognition

Nishi Mehta
Graduate student, Computer Science
#Person 50291671
nishimeh@buffalo.edu

Abstract

This project aims to solve a classification problem. The classification problem is that of handwritten digits. To solve this problem, we test the data on multiple classifiers and then implement an ensemble classifier using bagging and majority voting. Performance of all the classifiers is evaluated individually and compared with the ensemble classifier. We take the MNIST public dataset and train and test on it. The classifiers used to solve the problem are Logistic Regression using backpropagation, Support Vector Machine, Neural networks and Random Forest Classifiers.

1 Introduction

With handwritten digit recognition being an established and significant problem, there has been a great deal of research work that has been undertaken in this area. It is not a trivial task because of the big variation that exists in the writing styles that have been found in the available data. Therefore both, the features and the classifier need to be efficient.
Handwritten digit recognition remains a vital area because of its huge number of practical applications, as well as the important financial implications. It is promising in a wide range of application domains, including online handwriting recognition on computer tablets; zip code recognition to help sort posted mail, as well as the verification of signatures on cheques in order to thwart any attempts at bank fraud, etc. Handwritten digit recognition is also widely used in a number of academic institutions to process their examination papers.

2 Datasets

2.1 MNIST Dataset

For both training and testing of our classifiers, we will use the MNIST dataset. The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

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 dataset could be downloaded from here:

http://yann.lecun.com/exdb/mnist/

The original black and white (bilevel) 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.

Figure 1 Sample Images from the MNIST Dataset

2.2 USPS Dataset

We use USPS handwritten digit as another testing data for this project to test whether your models could be generalized to a new population of data.

As we train on the MNIST and then test on the USPS dataset, we resize all the images in the USPS dataset to 28*28 pixels so that all the images have the same number of features.

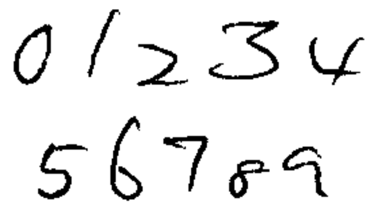

Figure 2 USPS Dataset Sample Images

As mentioned above, the handwritten digit recognition has important applications in the field of postcard zip code reading. It can be very useful if an efficient algorithm can be built to do so.

2.3 Training and Testing Data split

We split the USPS data into Training, Validation and Testing data in the ratio of 5:1:1 respectively. As there are 70,000 samples, we take 50,000 samples for training, 10,000 samples for validation and testing data.

We use the 19999 samples of the USPS dataset entirely for testing and evaluate whether our models can work on a new population of data.

Each image has 784 features, both the MNIST and the USPS dataset.

3 Logistic Regression

Logistic regression is a discriminative probabilistic statistical classification model that can be used to predict the probability of occurrence of an event. In this case there are multiple events. Hence, we use the Multinomial Logistic Regression approach. We use multinomial when the dependent variable is categorical. In this case, we need to classify the data into 10 classes that is categories. For example, the event that a given image is a '0'. We calculate probabilities of all the classes and then output the class with the maximum probability.

3.1 Genesis Equation and Computational Graph

The genesis equation is given by:

$$p(C_k|X) = y_k(X) = \frac{e^{a_k}}{\sum_j e^{a_j}}$$

Where $a_k = {W_k}^T X + b_k$

Hence, the class with the maximum probability is chosen for a given input X.

$$\widehat{y_i}|X_i = argmax_k(p(C_k|X))$$

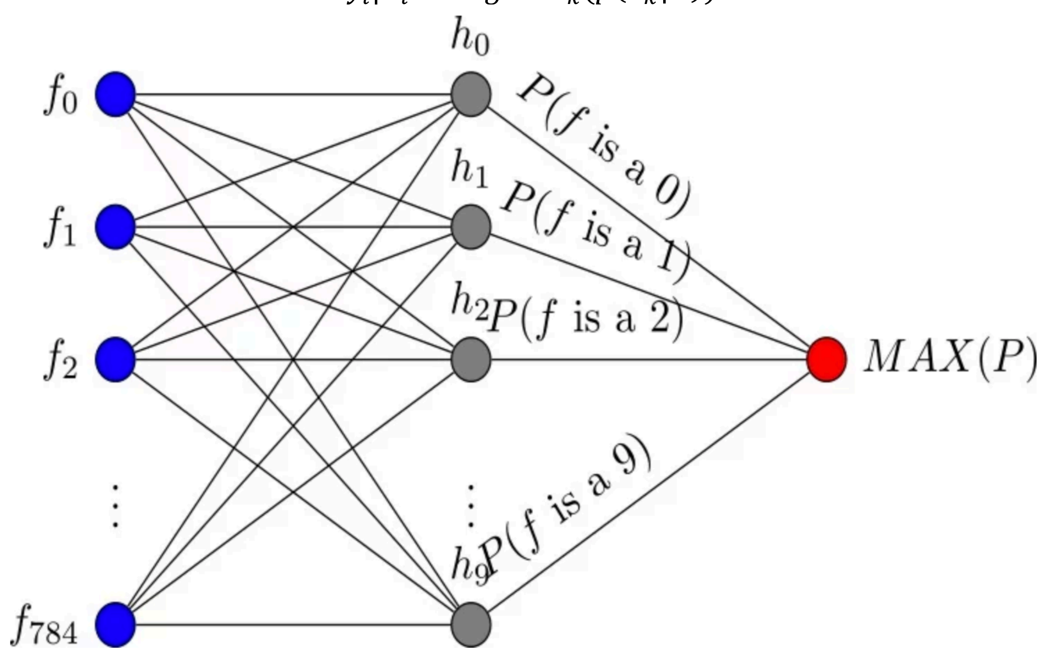

Figure 3 Computational graph for Multinomial Logistic Regression

The above computational graph represents how 784 features are mapped to 10 classes. Each edge has a particular weight which is learned over many iterations.

3.2 Loss function

We use the cross-entropy loss function for our multinomial logistic regression model.

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. A perfect model would have a log loss of 0.

$$E(X) = L = -\sum_{k=1}^{K} t_k ln y_k$$

Where $y_k = y_k(X)$ and t_k represents the actual output value.

3.3 Computation of weights

In order to optimize the logistic regression model, we need to use gradient descent approach. We take gradient of the loss function with respect to the weight and change weights according to the gradient value to reach a minima.

The gradient of the error function would be,

$$\Delta_{w_j} E(X) = (y_j - t_j)X$$

$$w_j^{t+1} = w_j^t - \eta \, \Delta_{w_j} E(X)$$

Hence, we update the weights iteratively to reach an optimal solution.

3.4 Batch Gradient Descent

To perform Logistic Regression, we use batch Gradient Descent for faster computations using matrix multiplications. Using this the speed the computation, increases greatly. We run the algorithm for multiple epochs on the dataset.

3.5 Performance

The performance of the datasets on the logistic regression model is given below:

Dataset	Log Loss Training	Log Loss Validation	Log Loss Testing	Accuracy
MNIST	0.50747	0.6171	0.63616	0.9012
USPS	-	-	3.325	0.3501

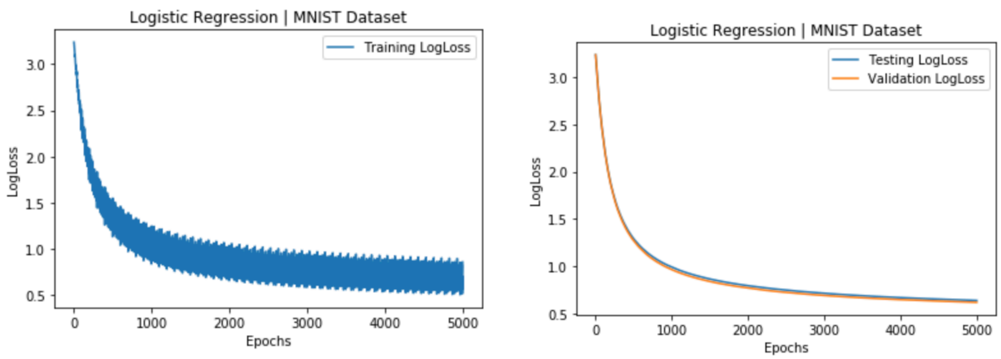

Figure 4 Negative Log Loss Plot for Training, Validation and Testing USPS Data

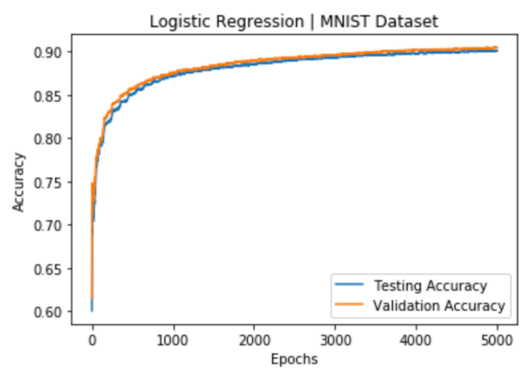

Figure 5 Accuracy Plot for USPS Data

Through this we can observe that after a certain amount of epochs we get around 90% accuracy for the MNIST dataset. However, the USPS dataset gives only 35% accurate results which is not a good result.

Confusion Matrix for MNIST Data:

```
[[ 956    0   11    5    2   15   16    3    9   13]
 [   0 1102    7    1    6    6    3   20   10    8]
 [   3    2  888   18    5    6    6   29   10    6]
 [   3    4   19  898    0   43    2    4   29   11]
 [   0    1   15    1  901   15   13   11    8   43]
 [   2    2    0   32    0  726   16    0   26   16]
 [   8    4   17    6   10   17  897    0   12    0]
 [   1    0   21   15    2   10    1  918   13   24]
 [   7   20   45   22    8   43    4    4  840    6]
 [   0    0    9   12   48   11    0   39   17  882]]
```

Confusion Matrix for USPS Data:

```
[[ 601  234  219  108   65  182  380  198  226   52]
 [   4  298   25    3   86   20   13  213   30  188]
 [ 375  126 1176  121   36  214  346  318  146  164]
 [  56  350  138 1259   62  184  106  450  208  470]
 [ 255  286   67   21 1028   45  105   74  127  155]
 [ 111   52   75  236  120 1031  218   78  573   84]
 [ 104   41   93   31   41  126  698   35  119   15]
 [  42  299   93   58  128   72   25  300   43  365]
 [ 147  296   90  103  292   89   75  287  444  339]
 [ 305   18   23   60  142   37   34   47   84  168]]
```

3.6 Strength and Weakness

Logistic Regression attempts to predict the output using independent variables. If any of the independent variables are inaccurate or if the variables are dependent on each other then, the logistic regression model may not perform well. This is because the model then over weights those features which are not independent. It is also vulnerable to overfitting and may not give the best results due to that.

It is a simple model which is easy to implement and understand. In this case, however the pixel values may not be independent of each other which is why it takes a large number of epochs for the model to give considerable results.

4 Neural Network

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example.

4.1 Deep Neural Network

4.1.1 Hyperparameters

We discuss some hyperparameters for the neural network and how they affect the performance of the model.

4.1.1.1 Learning Rate

The learning rate determines how gradually or quickly the weight of the neural network are updated. A high learning rate might not be able to finetune the network, however a low learning rate might become very slow for the parameters to adapt.

4.1.1.2 Dropout Rate

The dropout rate is defined as the fraction rate of input that is made 0 after each training iteration. This is done to prevent overfitting of data.

4.1.1.3 Hidden Layers and number of nodes

When a neural network has too few hidden neurons (<64), the network does not have the capacity to learn enough. On increasing the number of nodes to 128 the accuracy increases greatly. However, afterwards even if we increase the nodes by a value of 512, the accuracy increases by only 2%. Hence, we can say that after a point the hidden layer of nodes do not make much difference to the accuracy.

4.1.1.4 Epochs

One epoch is when an entire dataset is passed forward and backward through the neural network only once. As the number of epochs increases, more the number of times the weights are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve.

4.1.1.5 Activation Function

An activation function is the output of that node, given an input or set of inputs.The output is usually between [-1,1], depending on the activation function. It decides if the information at the node is relevant or should be ignored.
Relu function: $A(x) = \max(0,x)$
As we can observe, the relu function works best on this data. The reason is that all the activation functions: tanh, sigmoid and others still keep the data sparse. However, relu does not activate around half of the nodes because of its nature.

4.1.1.6 Loss Function

The loss function is used to measure the inconsistency between actual and predicted output. It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function.

4.1.1.7 Hyperparameters used

To train the model, we used the following hyperparameters and found that this combination gives the best output.

Hyperparameter	Value
Optimizer	sgd
Epochs	100
Number of nodes	32
Activation function	softmax
Loss function	Categorical_Crossentropy

4.1.2 Performance

The performance of the deep neural network on the datasets is given below.

Dataset	Accuracy	Loss
MNIST	0.9284	0.259
USPS	0.384	2.54

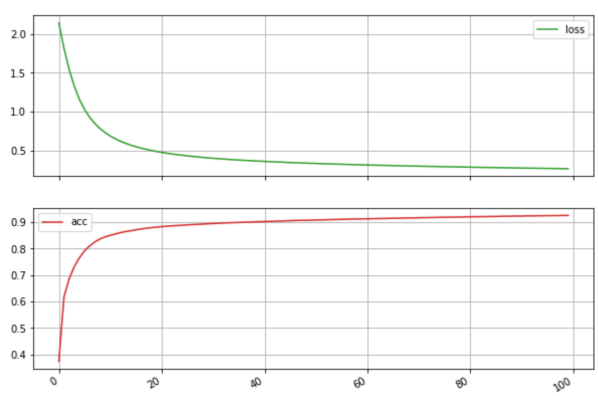

Figure 6 Accuracy and Loss for USPS Dataset for Deep Neural Network

Confusion Matrix for MNIST Data:

```
[[ 958    0    2    1    0    4   10    1    4    0]
 [   0 1110    3    2    0    1    4    2   13    0]
 [  13    6  932    8   10    1   13   12   33    4]
 [   3    1   17  931    0   23    2   11   18    4]
 [   1    2    5    1  922    0   12    2    6   31]
 [   8    3    4   39    5  785   15    4   23    6]
 [  10    3    4    0   13   14  910    1    3    0]
 [   4    9   27    6    8    0    0  949    2   23]
 [   6    7    6   20    8   18   10   13  881    5]
 [  14    7    1   12   36    7    1   21    4  906]]
```

Confusion Matrix for USPS Data:

```
Deep Neural Network | USPS
Accuracy USPS: 0.384869243453232
Loss USPS: 2.546163725920919
[[ 530    2  256   78  242  192   63   74  132  431]
 [ 130  365  203  237  144  116   23  597  148   37]
 [ 181   18 1285  105   57  156   58   69   52   18]
 [  66    4  145 1358   11  267   10   67   45   27]
 [  24   50   43   36 1088  157   23  177  231  171]
 [ 110   10  202  179   39 1232   84   65   60   19]
 [ 304    7  386   58   87  297  774   15   23   49]
 [  94  174  285  518   40  195   14  450  164   66]
 [ 177   24  165  231  105  651  106   59  407   75]
 [  27  130  122  500  141   96    6  471  299  208]]
```

4.1.3 Strength and Weakness

The disadvantage of using the neural network is that it is a black box and one can't figure out how it came up with the output. Hence, it is difficult to tweak the hyperparameters as we don't know which ones are important.
The neural network requires a large amount of data to make the right prediction. Hence, because the neural network has not trained on the USPS dataset, it can't make accurate predictions and returns a low accuracy of 38%.
As we have 50,000 images to train on, neural networks return a good result on the MNIST test data.
Neural Networks do return good results but are computationally expensive.

4.2 Convolutional Neural Networks

Convolutional Neural Networks are very similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. Softmax) on the last (fully-connected) layer.
CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network.

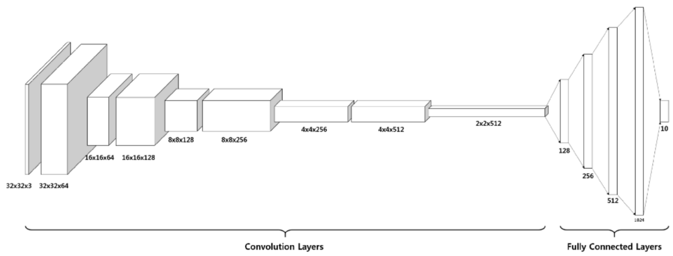

Figure 7 Example of convolutional network architecture

For our dataset, we use the following architecture:

In → [Conv2D → relu]*2 → MaxPool2D → Dropout(0.25) → Flatten → Dense → relu → Dropout (0.5) → Softmax→ Out

4.2.1 Performance

Using the above configuration, we achieve very good results for the CNN model. Below are the results.

Dataset	Accuracy	Loss
MNIST	0.9917	0.027
USPS	0.60	2.64

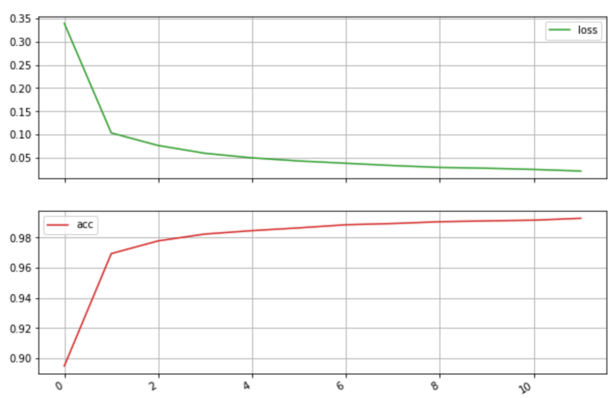

Figure 8 Accuracy and Loss for USPS Dataset for Convolutional Neural Network

Confusion Matrix for MNIST Data:

```
[[ 977    1    0    0    0    0    1    1    0    0]
 [   0 1132    1    0    0    0    1    0    1    0]
 [   1    1 1023    0    2    0    0    3    1    1]
 [   0    0    3  999    0    3    0    0    2    3]
 [   0    0    0    0  979    0    0    0    0    3]
 [   1    0    0    4    0  879    5    2    0    1]
 [   3    2    0    0    2    1  949    0    1    0]
 [   0    2    7    1    0    0    0 1016    0    2]
 [   4    0    2    0    0    0    0    0  965    3]
 [   0    0    0    1    7    0    0    3    0  998]]
```

Confusion Matrix for USPS Data:

```
Convolutional Neural Network | USPS
Accuracy USPS: 0.6004300214876633
Loss USPS: 2.6429732614987804
[[ 869    1  202   15  193   11   55    7  260  387]
 [   9  755  583   65  143   20   39  358   20    8]
 [  32    5 1802   46   11   51   18   14   14    6]
 [   2    1   96 1666    7  206    5    6   10    1]
 [   1   32   61    5 1347   12   58  199  265   20]
 [  13    0   60   48   13 1774    7   26   38   21]
 [  47   11  114    4   61   26 1710    1   20    6]
 [  11   77 1033  133   17   10   55  575   87    2]
 [  32    7  147  175   44  421  101   33 1006   34]
 [   1   26  386  157  140   13    3  476  294  504]]
```

4.2.2 Strength and Weakness

The convolutional neural network returns the best results on this dataset. It is due to the fact that the convolutional networks are built assuming that the input is in the form of images. It returns a high accuracy of 99.17% on the USPS test dataset. It also returns a decent 60% accuracy on the USPS dataset which has not even been trained on.

The only disadvantages of CNN are that it is computationally expensive as a single epoch takes atleast 1 minute to run on a general configuration computer and that it is a black box.

5 Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples.

5.1 Kernel

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.
The linear kernel separates the data by using a linear hyperplane between the samples of the data.
The polynomial or the RBF kernel is used when the data is not linearly separable.

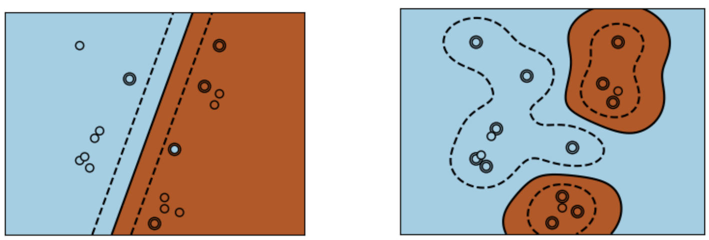

Figure 9 Linear and RBF kernel

5.2 Gamma

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 seperation line are considered in calculation for the separation line. Whereas high gamma means the points close to plausible line are considered in calculation.

5.3 Performance

We evaluate the SVM model using various hyperparameters and we compute the ideal hyperparameters for the same.

5.3.1 Linear Kernel

We evaluate the performance of SVM using the linear kernel and keeping all other parameter values as default.

```
SVM Linear Kernel | MNIST
Confusion matrix:
[[ 959    0    5    2    2    4    7    0    1    0]
 [   0 1121    3    3    0    1    2    1    4    0]
 [   6    8  968    9    3    2   11   10   13    2]
 [   5    2   17  944    4   13    1    8   13    3]
 [   2    1   10    1  943    0    4    2    2   17]
 [  13    4    2   39    5  792    9    1   22    5]
 [  10    3   11    1    5   14  911    2    1    0]
 [   1    8   20   10    6    1    0  961    3   18]
 [   8    4    9   25   11   27    6    5  871    8]
 [   7    6    2   13   32    4    0   18    7  920]]
Accuracy: 0.939
```

```
SVM Linear Kernel | USPS
Confusion matrix:
[[ 348    0  476  152  222  345   74  172   10  201]
 [  60  303  534  275  230  172   17  351   37   21]
 [ 139   63 1293  115   33  221   55   45   21   14]
 [  56   58  341  898    8  520    9   45   48   17]
 [  24   24  221   82  800  215   10  464   82   78]
 [  47   25  652  240   41  876   30   35   41   13]
 [ 146   19  903   55   86  264  462   38    2   25]
 [  19   74  201  706   54  294   12  522   84   34]
 [ 100   16  298  449  126  692   82   58  160   19]
 [  18   38  204  588  142  104    8  580  155  163]]
Accuracy: 0.2912645632281614
```

5.3.2 RBF Kernel

We evaluate the performance of SVM using the RBF kernel and keeping all other parameter values as default.

```
SVM | RBF Kernel | Gamma = Default | MNIST
Confusion matrix:
[[ 967    0    1    0    0    5    4    1    2    0]
 [   0 1120    2    3    0    1    3    1    5    0]
 [   9    1  962    7   10    1   13   11   16    2]
 [   1    1   14  950    1   17    1   10   11    4]
 [   1    1    7    0  937    0    7    2    2   25]
 [   7    4    5   33    7  808   11    2   10    5]
 [  10    3    4    1    5   10  924    0    1    0]
 [   2   13   22    5    7    1    0  954    4   20]
 [   4    6    6   14    8   24   10    8  891    3]
 [  10    6    0   12   33    5    1   14    6  922]]
Accuracy: 0.9435
```

```
SVM | RBF Kernel | Gamma = Default | USPS
Confusion matrix:
[[ 573    2  428   19  285  248   73   44    6  322]
 [ 110  429  285  137  273  180   46  501   22   17]
 [ 128   18 1402   59   39  198   61   57   23   14]
 [  76    3  186 1123   11  483    5   70   27   16]
 [  18   67   91   14 1167  267   22  194   69   91]
 [ 108   17  257  102   25 1367   60   43   15    6]
 [ 197    7  489   24   98  394  748   13    7   23]
 [  50  225  457  265   57  416   15  452   41   22]
 [  73   25  209  193   87 1006   95   41  244   27]
 [  26  166  228  278  213  165    8  499  214  203]]
Accuracy: 0.38541927096354817
```

5.3.2 RBF Kernel, Gamma = 0.01

The highest accuracy is obtained when we use the RBF kernel and Gamma = 0.01.

```
SVM | MNIST
Confusion matrix:
[[ 973    0    2    1    0    1    1    0    2    0]
 [   0 1128    3    0    0    1    1    1    1    0]
 [   3    1 1016    0    1    0    1    7    3    0]
 [   0    0    1  993    0    3    0    4    6    3]
 [   1    0    3    0  967    0    2    0    0    9]
 [   3    0    0    9    1  870    4    0    3    2]
 [   5    2    1    0    2    4  943    0    1    0]
 [   0    7    8    2    0    0    0 1004    0    7]
 [   3    0    2    4    4    2    1    2  952    4]
 [   3    4    1    7   10    2    0    4    1  977]]
Accuracy: 0.9823
```

An accuracy of 98.23% is obtained in this case.

```
SVM | USPS
Confusion matrix:
[[ 652    1  485   40  128  291   39   93    0  271]
 [ 105  517  345  116   99  128   17  647   15   11]
 [  55   11 1681   56    9  128   18   36    4    1]
 [  22    3  234 1236    0  473    0   29    0    3]
 [  17   72  214   22  909  282    6  422   14   42]
 [  59    6  293   45    8 1544   14   25    5    1]
 [ 169    6  738   27   35  202  799    8    0   16]
 [  45  183  452  395   20  288    2  610    2    3]
 [  81    5  321  313   33 1031   32   40  137    7]
 [   9   82  328  383  113  144    2  690   89  160]]
Accuracy: 0.4122706135306765
```

Dataset	Gamma	Kernel	Accuracy
MNIST	Default	linear	0.939
USPS	Default	linear	0.291
MNIST	Default	RBF	0.9435
USPS	Default	RBF	0.385
MNIST	0.01	RBF	0.9823
USPS	0.01	RBF	0.4122

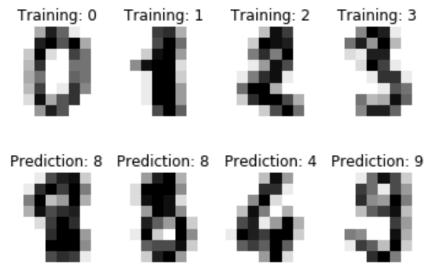

Figure 10 SVM Training and Testing

5.4 Strength and Weakness

SVM tries to maximize the margin between the different classes of data. It can work very well with high dimensions. In this case, we have 784 features and it gives around 98% accuracy which is very ideal.

The weakness is that SVM requires high level of computation which is why it takes the most time to run out of all the classifiers. It also requires a lot of memory when running.

6 Random Forest Classifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

It is a kind of bagging technique used in ensemble classifier. It separates a number of features and builds decision trees individually. Then it takes an average of the outputs of all the decision trees and outputs the final result.

We can define the number of trees in the forest by a parameter called 'n_estimators'. In our case we use 10 trees to determine the result.

6.1 Decision Trees

We discuss the most basic decision tree algorithm, ID3(Iterative Dichotomiser 3).

Let the entropy be defined as:

$$S = -\sum_{i=1}^{N} p_i log p_i$$

Where p_i is the probability.

And let Information gain be defined as,

$$IG = E_0 - \sum \frac{N_i}{N} S_i$$

Where N_i = No. of ith elements in group and N = Total no. of elements in group.

We calculate the information gain of each feature. We select the feature with the highest gain as the root node and keep dividing the tree until the point where we reach to a decision.

6.2 Performance

We use a forest of 10 decision trees to classify our data.

Dataset	Accuracy
MNIST	0.9445
USPS	0.307

Confusion Matrix:

```
Random Forest | MNIST
Confusion matrix:
[[ 968    0    1    2    0    3    5    1    0    0]
 [    0 1118    3    3    2    0    4    1    4    0]
 [   10    1  978    6    6    1    7   12   10    1]
 [    1    2   17  944    0   20    1    8   11    6]
 [    3    2    1    1  932    1   11    4    3   24]
 [    8    2    2   32    6  819    6    1    9    7]
 [    7    3    1    1    7   12  924    0    3    0]
 [    2    8   22    7    6    0    0  962    3   18]
 [    7    3   16   21    9   14    8    7  878   11]
 [    6    7    6   14   31    9    2    6    6  922]]
Accuracy: 0.9445
```

```
Random Forest | USPS
Confusion matrix:
[[640  44 272 109 369 210  77  95  14 170]
 [ 62 637 169  83 116  96  32 767  22  16]
 [209 131 957 173  67 194  40 185  24  19]
 [ 98  74 224 927 104 367  18 140  15  33]
 [ 47 273 110  91 874 160  33 330  33  49]
 [253  67 219 258  79 911  46 110  26  31]
 [434 109 311 112 164 272 468  97  13  20]
 [ 67 431 415 263  93 172  37 499   9  14]
 [150 130 252 249 165 684  96  97 130  47]
 [ 75 301 300 297 249 133  24 439  82 100]]
Accuracy: 0.3071653582679134
```

6.3 Strength and Weakness

The random forest classifier doesn't need to know the type and nature of the data. It works on independent and dependent features. It is easier to understand and we can understand how we reach to a certain output. We can also predict which features are the most important in order to classify the data. It is computationally faster than the other classifiers.

The only major drawback is that it is possible for the random forest to take up a lot of memory which in turn leads to slower evaluation.

7 Ensemble Classifier

Ensemble Learning is a process using which multiple classifiers are strategically constructed to solve a particular problem.
In our case we use bagging ensemble classifier with majority voting.

7.1 Bagging

Bagging is one of the Ensemble construction techniques which is also known as Bootstrap Aggregation. Bootstrap establishes the foundation of Bagging technique. ‘n’ samples are selected randomly and are feeded to different classifiers. Samples from the training set are randomly selected and certain classifiers are trained using these samples.

We implement majority voting which means that we select the output which is given by most classifiers. This means that each classifier will predict one value and the value which is output by the greatest number of classifiers is the output.

7.2 Boosting

Boosting is another technique. The algorithm works by training a model with the entire training set, and subsequent models are constructed by fitting the residual error values of the initial model. In this way, boosting attempts to give higher weight to those observations that were poorly estimated by the previous model. Once the sequence of the models is created the predictions made by models are weighted by their accuracy scores and the results are combined to create a final estimation.

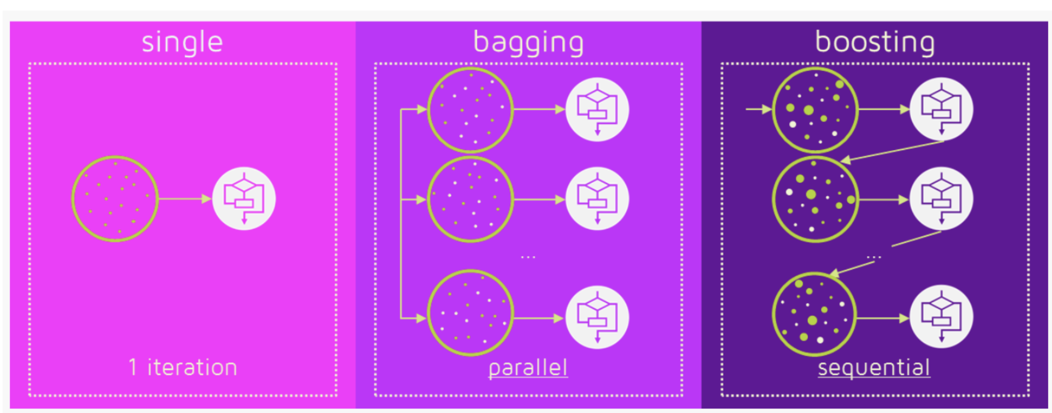

Figure 11 Bagging and Boosting

7.3 Performance

Classifier	Accuracy MNIST	Accuracy USPS
Logistic Regression	0.9012	0.3501
Deep Neural Network	0.9284	0.384
Convolutional Neural Network	0.9917	0.60
Linear SVM	0.939	0.291
RBF SVM	0.9823	0.4122
Random Forest	0.9445	0.307
Ensemble	**0.9591**	**0.416**

We can see that the performance of the ensemble learning is better than most classifiers in the case of both MNIST and USPS dataset.

Confusion Matrix for the ensemble:
MNIST Dataset:

```
Confusion matrix:
[[ 972    0    0    1    0    2    3    1    1    0]
 [    0 1120    2    2    0    1    4    1    5    0]
 [    9    1  987    4    4    0    8   10    8    1]
 [    0    0   10  971    0   10    1    7    8    3]
 [    1    0    4    0  955    0    6    0    2   14]
 [    8    2    1   23    4  830    9    1    9    5]
 [    8    3    3    1    4    9  929    0    1    0]
 [    3    9   22    4    4    0    0  970    2   14]
 [    6    3    3   14    6   15    8    5  911    3]
 [   10    6    1   12   18    4    1    9    2  946]]
Accuracy: 0.9591
```

USPS Dataset:

```
Confusion matrix:
[[ 655    2  419   44  262  192   53   41   28  304]
 [ 134  479  316  177  175  117   18  543   32    9]
 [ 136   17 1532   66   32  118   38   38   15    7]
 [  64    3  181 1359    4  315    3   45   17    9]
 [  21   78   95   26 1172  182   13  224  122   67]
 [ 107   12  263  116   23 1384   34   41   17    3]
 [ 269    8  521   36   78  255  797   11    5   20]
 [  80  215  446  413   43  245   13  474   59   12]
 [ 120   24  225  252   87  854   75   39  301   23]
 [  30  157  235  419  151  102    5  502  222  177]]
Accuracy: 0.4165208260413021
```

8 Conclusion

Through this project, we tried to solve the handwritten digit classification problem using the MNIST and USPS dataset. We used various types of classifiers and evaluated the models on MNIST and USPS datasets. We then built an ensemble of all classifiers using bagging and majority voting. We observed the pros and cons of all the classifiers and how they can be useful. We achieved the best result using Convolutional Neural Networks.

8.1 No Free Lunch Theorem

The “No Free Lunch” theorem states that there is no one model that works best for every problem. The assumptions of a great model for one problem may not hold for another problem, so it is common in machine learning to try multiple models and find one that works best for a particular problem.
We evaluated the model on the USPS dataset and did not achieve good results in most cases. This is because the ground truth label for the USPS data is different than that of MNIST data. Hence our results also support the “No Free Lunch” theorem.

References

[1] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html
[2] https://medium.com/@pushkarmandot/build-your-first-deep-learning-neural-network-model-using-keras-in-python-a90b5864116d
[3] https://docs.scipy.org/doc/numpy-1.15.1/index.html
[4] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[5] https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
[6] https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html
[7] https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d
[8] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.score
[9] https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html
[10] https://www.datacamp.com/community/tutorials/ensemble-learning-python
[11] https://chemicalstatistician.wordpress.com/2014/01/24/machine-learning-lesson-of-the-day-the-no-free-lunch-theorem/
[12] https://www.codeproject.com/Articles/821347/MultiClass-Logistic-Classifier-in-Python
[13] https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py
[14] https://www.pugetsystems.com/labs/hpc/Machine-Learning-and-Data-Science-Multinomial-Multiclass-Logistic-Regression-1007/