Project 1

Classification of Handwritten Digits Using Different Classification Techniques

Submitted To,

Professor Birsen Sirkeci

Assistant Professor

Electrical Engineering Department

San Jose State University

By,

Kiran Pallappa Venkatarathna

011445525

Contents

Chapter 1

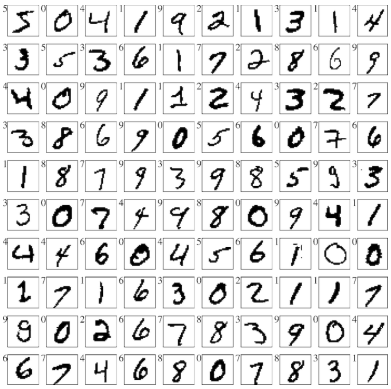
MNIST Data Set

1.1 Introduction [1]

MNIST is dataset consisting of handwritten digits images. This dataset is used widely for the as standard dataset for testing the algorithms that in meant for classification of the handwritten digit images [2]. MNIST is the abbreviation for Mixed National Institute of Standards and Technology database. This dataset is having been divided into two parts training samples and testing samples. Training samples were extracted from the American Bureau of Census employees and testing samples were extracted from the students of American High School.

MNIST dataset is formed from the fusion of NIST special databases three and one which contains handwritten images in the binary format. This fusion was necessary because the data collected from the American Bureau of Census considered as the training set was cleaner as compared to the data collected from the students of American High School. This dataset plays a key role in the training of the algorithm. Hence the fusion of data was necessary.

In the MNIST dataset 60,000 samples are considered for training the algorithm and 10,000 samples are considered for testing of the algorithm. Originally, the images were black and white and were normalized to fit into image size of 20x20. Because of this the present images are enhanced with grey levels due to the anti-aliasing techniques and images are fitted to 28x28.



The first 100 samples of MNIST dataset [3]

Chapter 2

Linear Classifier Model

2.1 Introduction [4]

A linear classifier performs the data classification based on linear combination of features which are considered as the input data to the model. These features are typically the characteristics of the data which are provided in the form of vector called the feature vector. Linear classifier works well with those data that has many features as it takes less time train and apply providing the accuracy that are comparable to non-linear classification models.

If the input feature to the model is a real vector $\vec{x}$, then the mathematical equation to the classifier is given as,

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_j\right)$$

where,
$\vec{w}$ = weights,
f = function converting dot products of two vector into required outcome.

2.2 Training and Fitting to MNIST Dataset

In this section, the linear classifier model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the linear classifier model, validating and testing.

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
from sklearn.model_selection import cross_val_score
from sklearn import model_selection
from sklearn.linear_model import SGDClassifier
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the linear model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

  The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

- **Step4**: Selection of best *n_iter* value with cross validation

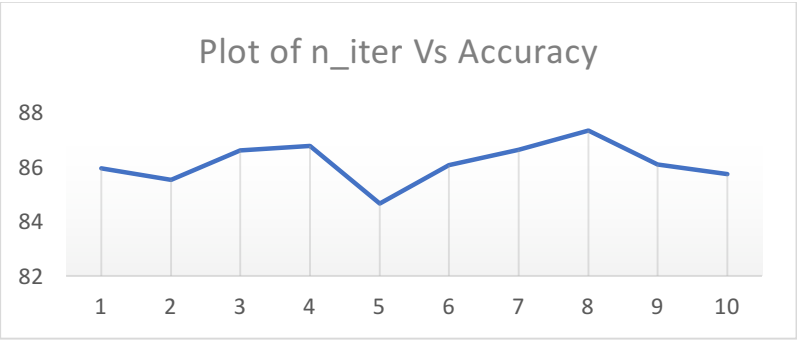
  To perform the cross-validation **cross_val_score**, which is simple helper function to perform cross-validation is used. This function is called on the dataset as well as the estimator.

  Using cross_val_score, the accuracy of the linear classifier model is estimated over the dataset by splitting it, fitting the model and computing the score for three consecutive times for different values of n_iter ranging from 1 to 11.

```
#Building the Linear classifier model by changing the n_iter as the hyperparameter
# Used the cros_val_score cross validation method
# For every n_iter values accuracy of all the 3 folds and their average is calculated
```

```
for i in range(1,11):
    clf = SGDClassifier(random_state = 42, n_iter=i)
    train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
    print("Iterations No: " +str(i))
    print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
    print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
    print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
    print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

  During each n_iter values, the accuracy of all the three folds and average of them is calculated as show below.
  Analysing and choosing the best n_iter to reduce the training error.



  It is seen that at n_iter equal to 8, the average accuracy is more. Considering this n_iter for further training and testing

- **Step5**: Performing the accuracy of the model over training data

  By choosing the n_iter = 8, the model is tested for its accuracy over the training data.

```
#Choosing the suitable n_iter values
#Performing the accuracy test on the training data
```

```
clf = SGDClassifier(random_state =42, n_iter=8)
from sklearn.model_selection import cross_val_predict
clf = clf.fit(X_train,y_train)
y_train_pred = cross_val_predict(clf, X_train, y_train, cv=5)
accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy: %.2f%%" % (accuracy*100))
```

```
Training Accuracy: 87.31%
```

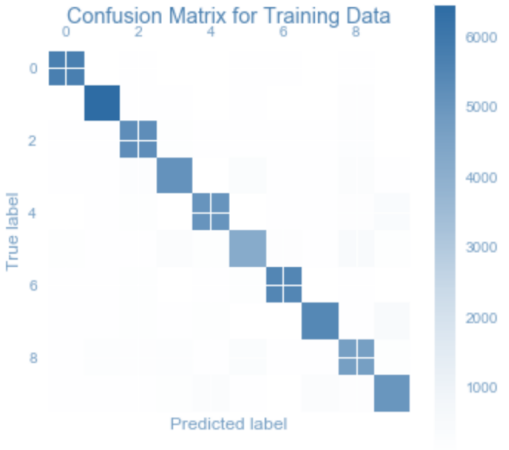
  It is observed that the model's accuracy over the training set using the **cross_val_predict** has been found to be 87.31%.

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the confusion matrix for the training set
```

```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

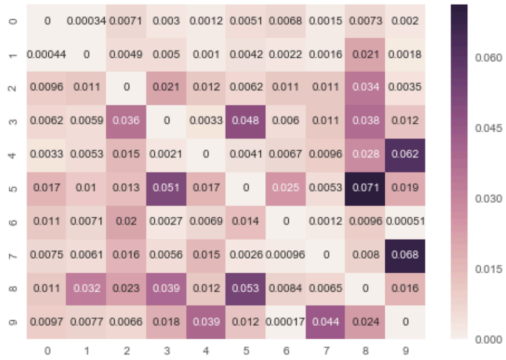


- **Step7**: Error confusion matrix for the training set

  Error confusion matrix for the training set model is plotted to analyze the misclassification of values by the linear classifier model.

```
#Priting the error values of the confusion matrix
```

```
import seaborn as sns
val_row= train_conf_mat.sum(axis=1, keepdims=True)
train_err_conf_mat = train_conf_mat / val_row
np.fill_diagonal(train_err_conf_mat, 0)
ax = sns.heatmap(train_err_conf_mat,annot=True)
```



- **Step8**: Fitting the model and testing

  The model is fitted and tested for its accuracy. The accuracy is found to be 84.04 %.

```
#Chose the suitable n_iter and performing the fit
#Testing the model
#Calculating the accuracy of the test data
```

```
#clf = SGDClassifier(random_state =42, n_iter=8)
#clf.fit(X_train, y_train)
y_pred_test = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print("Test Data Accuracy: %.2f%%" % (accuracy*100))
```

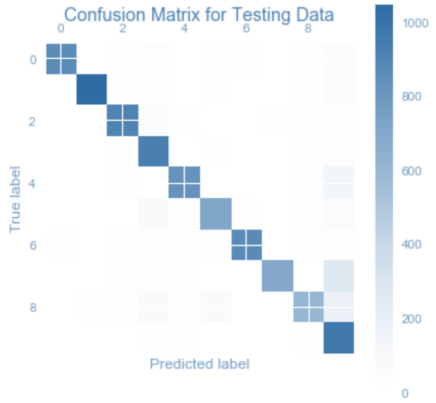
```
Test Data Accuracy: 84.04%
```

- **Step9**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Priting the error values of the confusion matrix
```

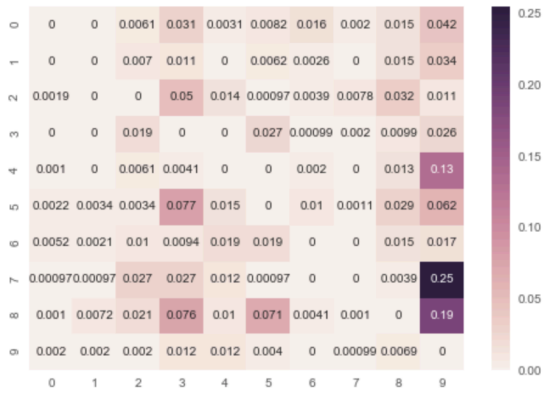
```
test_conf_mat = confusion_matrix(y_test,y_pred_test)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step10**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the linear classifier model.

```
row_sums = conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
```



2.3 Observations Summary

- The model was trained and built using the sklearn library
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- number of iterations(**n_iter**) was changed and at each run the accuracy of all the folds and its average are being calculated.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 3

KNN Classifier Model

3.1 Introduction [5]

KNN termed as k-nearest-neighbor the simple machine learning model used for classification as well as regression. This is an instance based learning where the function itself is approximated locally and delayed until classification. In the feature space, present in the multi-dimension, the vectors are considered as the examples to train the KNN model. During the training phase of the model, the algorithm stores the feature vectors of the training examples as well the label's class. It uses Euclidean distance measurement technique for classification involving the continuous set of variables, overlap distance measurement for variables which are discreet.

3.2 Training and Fitting to MNIST Dataset

In this section, the KNN classifier model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the KNN classifier model, validating and testing

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the KNN model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

  The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

- **Step4**: Selection of n_neighbor value with cross validation

  To perform the cross-validation **cross_val_score**, which is simple helper function to perform cross-validation is used. This function is called on the dataset as well as the estimator.

  Using cross_val_score, the accuracy of the KNN classifier model is estimated over the dataset by splitting it, fitting the model and computing the score for three consecutive times for different values of n_neighbor (*1, 3 and 5(default)).*

```
#Building the KNN classifier model by changing the no of neighbours as the hyperparameter
# Used the cros_val_score cross validation method
# For different values of n_neighbours(1, 2 and default(5))
# Accuracy of all the 3 folds and their average is calculated

#KNN classifier model with default n_neighbor value

from sklearn.model_selection import cross_val_score
clf = KNeighborsClassifier(n_jobs=10, weights='distance')
train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
print("Default Neighbours")
print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
Default Neighbours
1st fold accuracy- 97.01%
2nd fold accuracy- 96.96%
3rd fold accuracy- 96.96%
Average fold accuracy- 96.97%
```

```
#KNN classifier model with 2 as n_neighbor value

from sklearn.model_selection import cross_val_score
clf = KNeighborsClassifier(n_jobs=10, weights='distance', n_neighbors = 2)
train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
print("No of Neighbours: 2")
print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

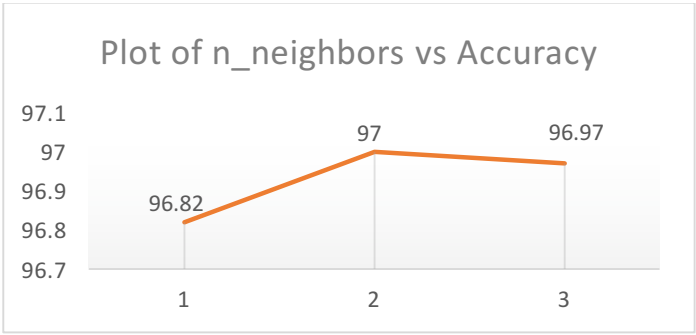
```
No of Neighbours: 2
1st fold accuracy- 96.96%
2nd fold accuracy- 96.77%
3rd fold accuracy- 96.74%
Average fold accuracy- 96.82%
```

```
#KNN classifier model with 3 as n_neighbor value

from sklearn.model_selection import cross_val_score
clf = KNeighborsClassifier(n_jobs=10, weights='distance', n_neighbors = 3)
train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
print("No of Neighbours: 3")
print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
No of Neighbours: 3
1st fold accuracy- 97.11%
2nd fold accuracy- 96.98%
3rd fold accuracy- 96.90%
Average fold accuracy- 97.00%
```

After analysis, it is found that with the increase in the n_neighbors the accuracy keeps increasing. It is also seen that the accuracy percentage after default n_neighbor value it increases in slow amount.



It is seen that at n_neighbor equal to 3 or above, the average accuracy is nearly the same. Considering this n_ neighbor for further training and testing.

- **Step5**: Performing the accuracy of the model over training data

  By choosing the n_neighbors = 5, as the model gives better results as compared to other test cases. Using this value, the model is tested for its accuracy over the training data.

```
#Choosing default (5) as the n_neighbors value
#Performing the accuracy test on the training data
```

```
clf = KNeighborsClassifier(n_jobs=10, weights='distance')
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(clf, X_train, y_train, cv=3)
accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy: %.2f%%" % (accuracy*100))
```

```
Training Accuracy: 96.97%
```

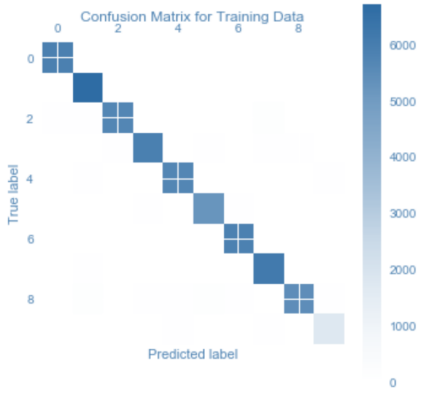
  It is observed that the model's accuracy over the training set using the **cross_val_predict** has been found to be 96.97%.

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the KNN classifier model.

```
#Printing the confusion matrix for the training set
```

```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step7**: Error confusion matrix for the training set

  Error confusion matrix for the training set model is plotted to analyze the misclassification of values by the KNN classifier model.

```
#Priting the error values of the confusion matrix
```

```
import seaborn as sns
val_row= train_conf_mat.sum(axis=1, keepdims=True)
train_err_conf_mat = train_conf_mat / val_row
np.fill_diagonal(train_err_conf_mat, 0)
ax = sns.heatmap(train_err_conf_mat,annot=True)
```

- **Step8**: Fitting the model and testing

  The model is fitted and tested for its accuracy. The accuracy is found to be 95.03 %.

```
#Chose the suitable n_neighbor and performing the fit
#Testing the model
#Calculating the accuracy of the test data
```

```
clf = KNeighborsClassifier(n_jobs=10, weights='distance')
clf.fit(X_train, y_train)
y_pred_test = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print("Test Data Accuracy: %.2f%%" % (accuracy*100))
```

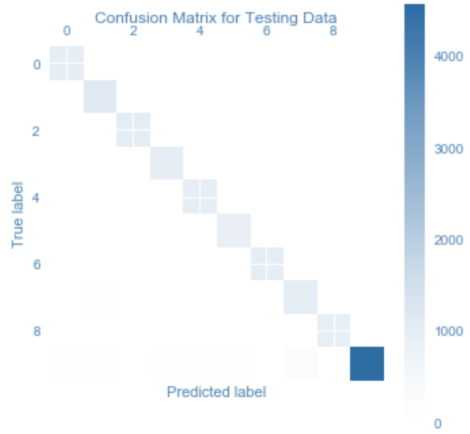
```
Test Data Accuracy: 95.03%
```

- **Step9**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the KNN classifier model.

```
#Priting the error values of the confusion matrix
```

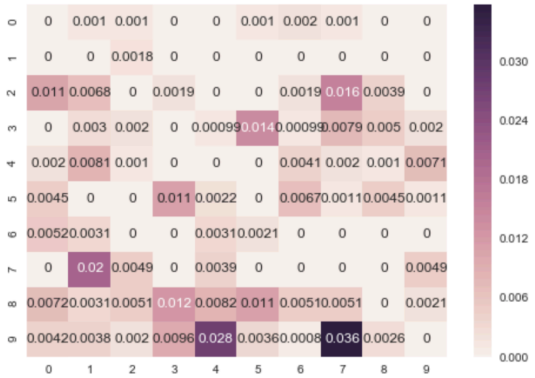
```
test_conf_mat = confusion_matrix(y_test,y_pred_test)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step10**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the KNN classifier model.

```
row_sums = conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
```



3.3 Observations Summary

- The model was trained and built using the sklearn library
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- number of neighbors(**n_neighbors**) also called as the k-value was changed and at each run the accuracy of all the folds and its average are being calculated.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 4

Radial Basis Function Neural Network

4.1 Introduction [6]

Radial Basis Function(RBF) is network come under the family of artificial neural network. RBF network encompasses the radial basis function as their activation function for neurons. This network has totally three layers involved: input layer, hidden layer and the output layer. The hidden layer neurons are equipped with non-linear RBF functions. The output layer performs the scalar function of the data input provided at the input layer of the network. It is given as,

$$\varphi(\mathbf{x}) = \sum_{i=1}^{N} a_i \rho(||\mathbf{x} - \mathbf{c}_i||)$$

where, N = no of neurons present in the hidden layer

c_i = center vector for neuron *i*

a_i = neuron weight

4.2 Training and Fitting to MNIST Dataset

In this section, the RBF network model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the RBF network model, validating and testing

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the linear model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

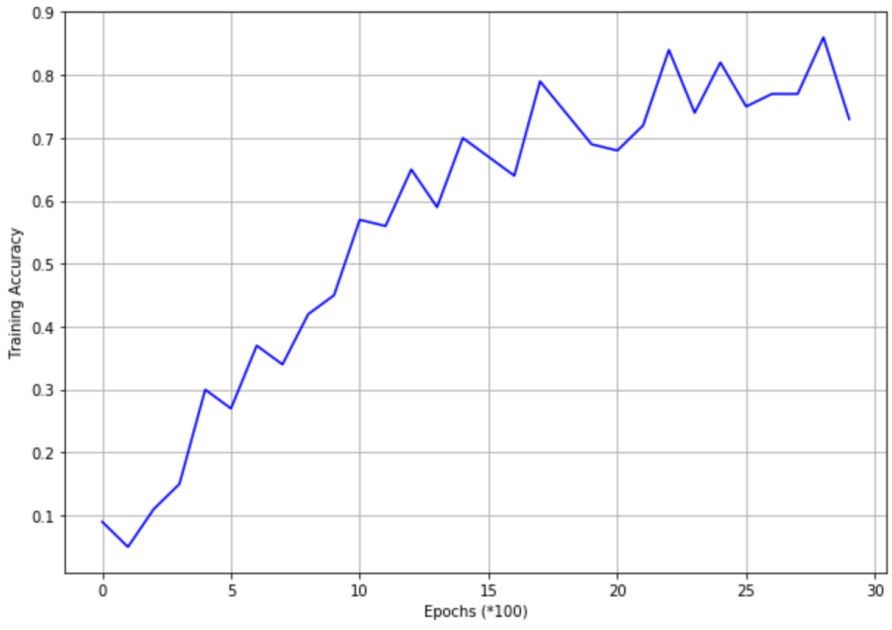
```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

- **Step4**: Importing the tensor flow and building the model

  To train the RBF neural model, RBF.py code [7] was used. In this training, RBFs hyper parameter- epochs i.e the maximum no of iteration. During each iteration values, the training accuracy is calculated as follows.

```
Epoch 0, training accuracy 0.09
Epoch 100, training accuracy 0.05
Epoch 200, training accuracy 0.11
Epoch 300, training accuracy 0.15
Epoch 400, training accuracy 0.3
Epoch 500, training accuracy 0.27
Epoch 600, training accuracy 0.37
Epoch 700, training accuracy 0.34
Epoch 800, training accuracy 0.42
Epoch 900, training accuracy 0.45
Epoch 1000, training accuracy 0.57
Epoch 1100, training accuracy 0.56
Epoch 1200, training accuracy 0.65
Epoch 1300, training accuracy 0.59
Epoch 1400, training accuracy 0.7
Epoch 1500, training accuracy 0.67
Epoch 1600, training accuracy 0.64
Epoch 1700, training accuracy 0.79
Epoch 1800, training accuracy 0.74
Epoch 1900, training accuracy 0.69
Epoch 2000, training accuracy 0.68
Epoch 2100, training accuracy 0.72
Epoch 2200, training accuracy 0.84
Epoch 2300, training accuracy 0.74
Epoch 2400, training accuracy 0.82
Epoch 2500, training accuracy 0.75
Epoch 2600, training accuracy 0.77
Epoch 2700, training accuracy 0.77
Epoch 2800, training accuracy 0.86
Epoch 2900, training accuracy 0.73
```

A plot of epochs vs training accuracy is plotted as shown below.



- **Step5**: Estimating the training accuracy of the model over testing data

  Once the model was trained using the best epoch = 3000, the model was tested and testing accuracy was found to be 81.71%

```
Test accuracy 0.8171
```

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the RBF neural network model.

```
#Printing the confusion matrix for the training set
```

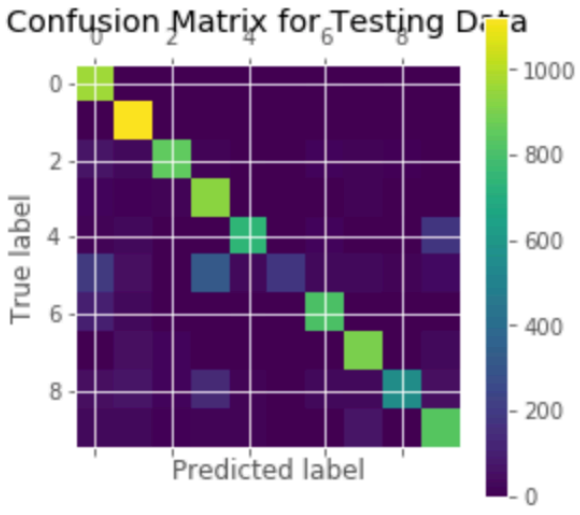
```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

- **Step7**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the RBF neural network model.

```
y_pred = prediction.eval(feed_dict={x: X_test,  keep_prob: 1.0})
```

```
test_conf_mat = confusion_matrix(y_test,y_pred)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step8**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the RBF neural network model.

```
import seaborn as sns
row_sums = test_conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = test_conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
plt.show()
```

4.3 Observations Summary

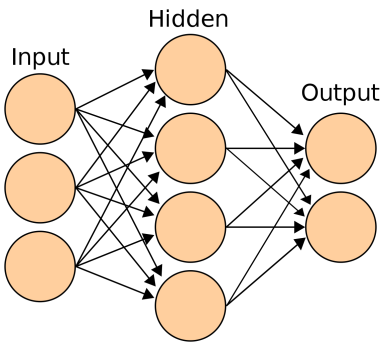
- The model was trained and built using the standard code that was available online.
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- maximum number of iterations(**max_iter**) was changed and at each run the accuracy of all the folds and its average are being calculated.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 5

One Hidden Layer Fully Connected Multilayer Neural Network

5.1 Introduction

Any neural network involving a hidden layer is considered as a multi-layer perceptron. In a one hidden fully connected layer a network involves one input layer, one hidden layer and one output layer with each neuron connected to every other neuron in the network. Each neuron can have sigmoid, tanh or RELU as the activation function. Each neuron takes the input and performs a nonlinear combination of the input and then passed on to the next layer where the non-linear activation functions is applied before passing it to the output layer. The following figure depicts the multi-layer neural network.



Multilayer Neural Network with Single Hidden Layer [8]

5.2 Training and Fitting to MNIST Dataset

In this section, one hidden layer fully connected multilayer neural network model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the one hidden layer fully connected multilayer neural network model network model, validating and testing.

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the linear model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

  The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

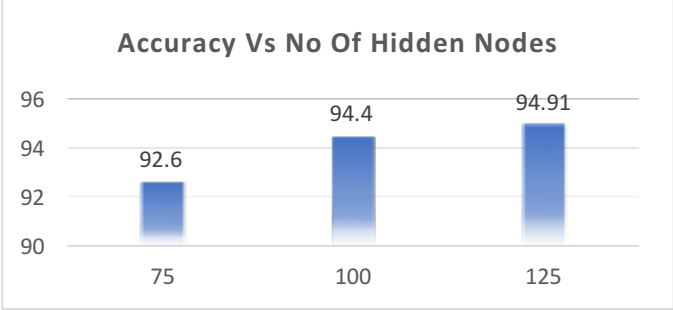
  Using cross_val_score, the accuracy of the one hidden layer neural network classifier model is estimated over the dataset by splitting it, fitting the model and computing the score for three consecutive times (as three folds are created) for different no of neurons in the hidden layer.

- **Step4**: Selection of no of neurons in the hidden layer with cross validation

  To perform the cross-validation **cross_val_score**, simple helper function to perform cross-validation, is used. This function is called on the dataset as well as the estimator.

```
for i in [75, 100, 125]:
    from sklearn.model_selection import cross_val_score
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(i, ), random_state=1)
    train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
    print("Hidden Layer Nodes: " +str(i))
    print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
    print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
    print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
    print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
Hidden Layer Nodes: 75
1st fold accuracy- 91.72%
2nd fold accuracy- 92.89%
3rd fold accuracy- 93.19%
Average fold accuracy- 92.60%
Hidden Layer Nodes: 100
1st fold accuracy- 94.27%
2nd fold accuracy- 94.52%
3rd fold accuracy- 94.42%
Average fold accuracy- 94.40%
Hidden Layer Nodes: 125
1st fold accuracy- 94.75%
2nd fold accuracy- 95.04%
3rd fold accuracy- 94.93%
Average fold accuracy- 94.91%
```



It is found that for the hidden layer having 125 neurons the training accuracy is found to be 94.91%.

- **Step5**: Performing the accuracy of the model over training data

  Choosing the no hidden neurons to be 125, as the model gives better results as compared to other test cases. Using this value, the model is tested for its accuracy over the training data.

```
#Choosing  as the no of neurons in the hidden layer
#Performing the accuracy test on the training data
```

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(125, ), random_state=42)
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(clf, X_train, y_train, cv=5)
accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy: %.2f%%" % (accuracy*100))
```

```
Training Accuracy: 94.69%
```

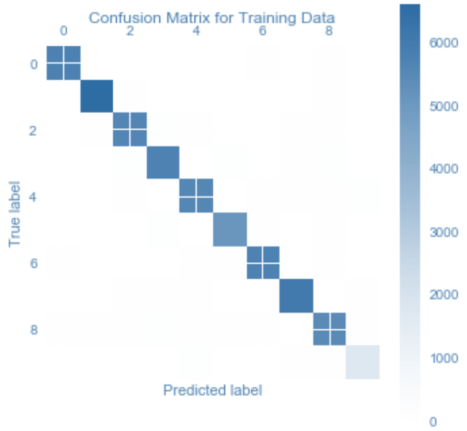
  It is observed that the model's accuracy over the training set using the **cross_val_predict** has been found to be 94.69%.

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the confusion matrix for the training set
```

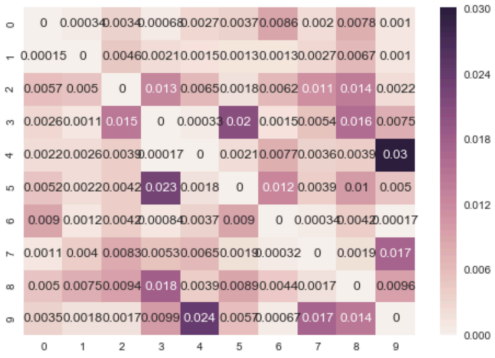
```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step7**: Error confusion matrix for the training set

  Error confusion matrix for the training set model is plotted to analyze the misclassification of values by the linear classifier model.

```
import seaborn as sns
val_row= train_conf_mat.sum(axis=1, keepdims=True)
train_err_conf_mat = train_conf_mat / val_row
np.fill_diagonal(train_err_conf_mat, 0)
ax = sns.heatmap(train_err_conf_mat,annot=True)
plt.show()
```



- **Step8**: Fitting the model and testing

  The model is fitted and tested for its accuracy. The accuracy is found to be 91.70 %.

```
#Chose no of neurons in the hidden layer and performing the fit
#Testing the model
#Calculating the accuracy of the test data
```

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(125, ), random_state=1)
clf.fit(X_train, y_train)
y_pred_test = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print("Test Data Accuracy: %.2f%%" % (accuracy*100))
```

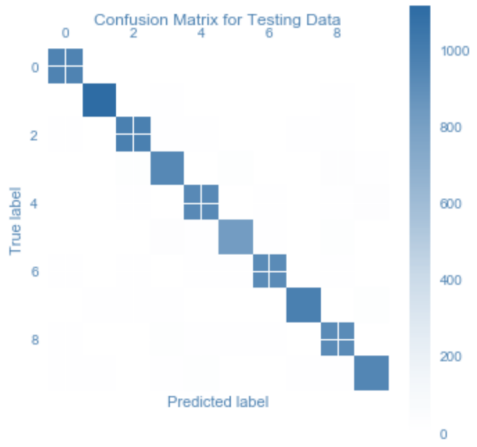
```
Test Data Accuracy: 95.12%
```

- **Step9**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the  confusion matrix
```

```
test_conf_mat = confusion_matrix(y_test,y_pred_test)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

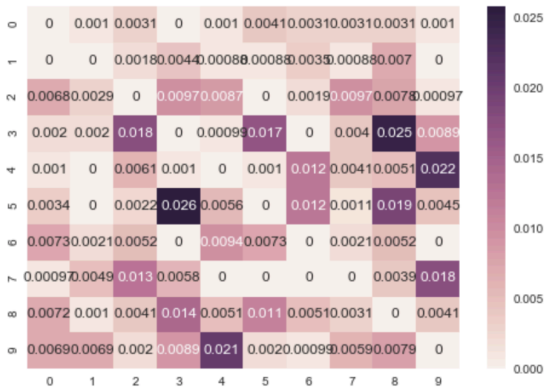


- **Step10**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the linear classifier model.

```
#Priting the error values of the confusion matrix
```

```
row_sums = test_conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = test_conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
plt.show()
```



5.3 Observation Summary

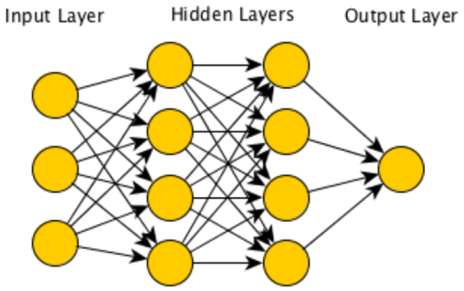
- The model was trained and built using the sklearn library
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- no of nodes in the hidden layer was changed and at each run the accuracy of all the folds and its average are being calculated.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 6

Two Hidden Layer Fully Connected Multilayer Neural Network

6.1 Introduction

Any neural network involving a hidden layer is considered as a multi-layer perceptron. In a two hidden fully connected layer a network involves one input layer, two hidden layers and one output layer with each neuron connected to every other neuron in the network. Each neuron can have sigmoid, tanh or RELU as the activation function. Each neuron takes the input and performs a nonlinear combination of the input and then passed on to the next layer where the non-linear activation functions is applied before passing it to the output layer. The following figure depicts the multi-layer neural network.



Multilayer Neural Network with Two Hidden Layer [9]

6.2 Training and Fitting to MNIST Dataset

In this section, one hidden layer fully connected multilayer neural network model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the one hidden layer fully connected multilayer neural network model network model, validating and testing.

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the linear model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

  The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

  Using cross_val_score, the accuracy of the one hidden layer neural network classifier model is estimated over the dataset by splitting it, fitting the model and computing the score for three consecutive times (as three folds are created) for different no of neurons in the hidden layer.

- **Step4**: Selection of no of neurons in the hidden layers with cross validation

  To perform the cross-validation **cross_val_score**, simple helper function to perform cross-validation, is used. This function is called on the dataset as well as the estimator.

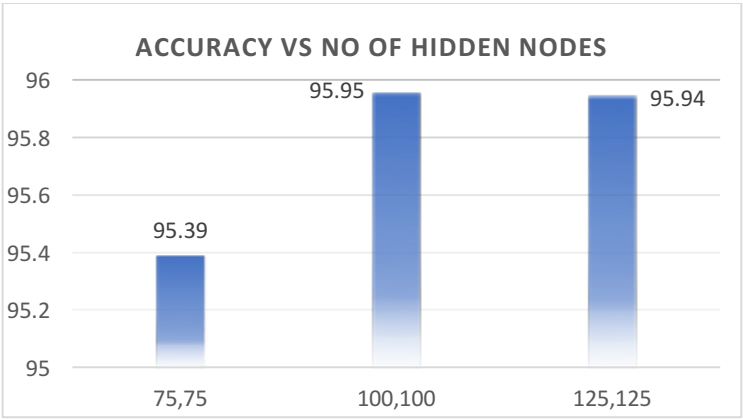
```
# Building the One Hidden Layer Neural Network model by changing the no of hidden layer neurons
# Used the cros_val_score cross validation method
# For different values of hidden layer neurons
# Accuracy of all the 3 folds and their average is calculated
```

```
for i in [75, 100, 125]:
    from sklearn.model_selection import cross_val_score
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(i, i), random_state=1)
    train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
    print("1st Hidden Layer Nodes: " +str(i))
    print("2nd Hidden Layer Nodes: " +str(i))
    print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
    print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
    print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
    print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
1st Hidden Layer Nodes: 75
2nd Hidden Layer Nodes: 75
1st fold accuracy- 95.46%
2nd fold accuracy- 95.27%
3rd fold accuracy- 95.45%
Average fold accuracy- 95.39%
```

```
1st Hidden Layer Nodes: 100
2nd Hidden Layer Nodes: 100
1st fold accuracy- 96.14%
2nd fold accuracy- 95.73%
3rd fold accuracy- 95.97%
Average fold accuracy- 95.95%
```

```
1st Hidden Layer Nodes: 125
2nd Hidden Layer Nodes: 125
1st fold accuracy- 95.99%
2nd fold accuracy- 95.89%
3rd fold accuracy- 95.95%
Average fold accuracy- 95.94%
```



It is found that for the hidden layer having 125 neurons in H1 and H2, the training accuracy is found to be 95.94%.

- **Step5**: Performing the accuracy of the model over training data

  Choosing the no hidden neurons to be 125 in H1 and H2, as the model gives better results as compared to other test cases. Using this value, the model is tested for its accuracy over the training data.

```
#Choosing  as the no of neurons in the hidden layer
#Performing the accuracy test on the training data
```

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(125, 125), random_state=1)
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(clf, X_train, y_train, cv=3)
accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy: %.2f%%" % (accuracy*100))
```

```
Training Accuracy: 95.94%
```

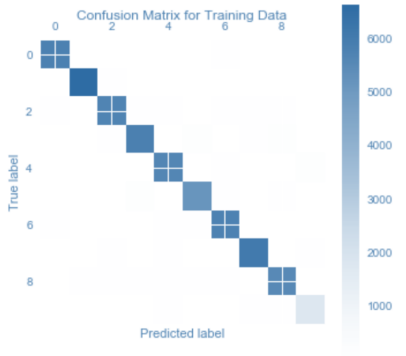
  It is observed that the model's accuracy over the training set using the **cross_val_predict** has been found to be 95.94%.

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the confusion matrix for the training set
```

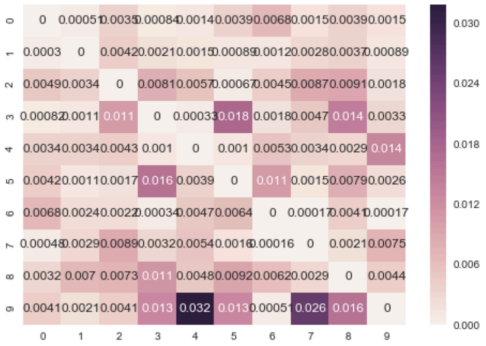
```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step7**: Error confusion matrix for the training set

  Error confusion matrix for the training set model is plotted to analyze the misclassification of values by the linear classifier model.

```
import seaborn as sns
val_row= train_conf_mat.sum(axis=1, keepdims=True)
train_err_conf_mat = train_conf_mat / val_row
np.fill_diagonal(train_err_conf_mat, 0)
ax = sns.heatmap(train_err_conf_mat,annot=True)
plt.show()
```



- **Step8**: Fitting the model and testing

  The model is fitted and tested for its accuracy. The accuracy is found to be 94.40 %.

```
#Chose no of neurons in the hidden layer and performing the fit
#Testing the model
#Calculating the accuracy of the test data
```

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(125,125), random_state=1)
clf.fit(X_train, y_train)
y_pred_test = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print("Test Data Accuracy: %.2f%%" % (accuracy*100))
```

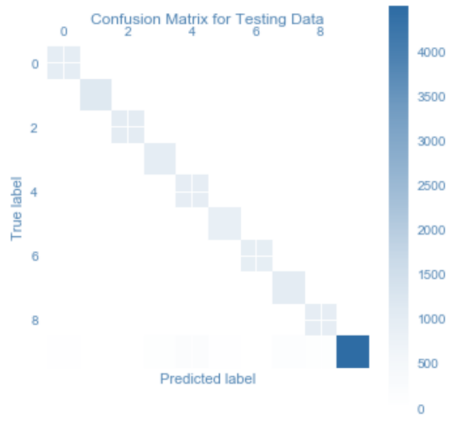
```
Test Data Accuracy: 94.40%
```

- **Step9**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the  confusion matrix
```

```
test_conf_mat = confusion_matrix(y_test,y_pred_test)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step10**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the linear classifier model.

```
#Priting the error values of the confusion matrix
```

```
row_sums = test_conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = test_conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
plt.show()
```

6.3 Observation Summary

- The model was trained and built using the sklearn library
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- no of nodes in the hidden layer 1 and hidden layer 2 was changed and at each run the accuracy of all the folds and its average are being calculated.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 7

Naiver Bayes Classifier

7.1 Introduction [10]

Naïve Bayes is one of the simple algorithms used for data classification technique. The input features provided to the model are the vector which are the problem instances. To these problem instances the model assign a class labels which are drawn from the finite set of labels. These classifiers are extensively ascendable as of which an involves parameters that are linear to the variables. Using the closed-form expression maximum-likelihood training can be carried out rather than the use of iterative approximation that consumes huge time as when in conjunction with other classifiers.

7.2 Training and Fitting to MNIST Dataset

In this section, Naïve Bayes model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the Naïve Bayes model network model, validating and testing.

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the linear model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

  The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

Using cross_val_score, the accuracy of the Naïve Bayes classifier model is estimated over the dataset by splitting it, fitting the model and computing the score for three consecutive times (as three folds are created) for different no of neurons in the hidden layer.

- **Step4**: Selection of event model with cross validation

  To perform the cross-validation **cross_val_score**, simple helper function to perform cross-validation, is used. This function is called on the dataset as well as the estimator.

  In here different event models such as Gaussian, Binomial and Multinomial Naïve Bayes are trained and accuracy of each model is evaluated. It is found the Binomial accuracy was more when compared to the other event model.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
clf = MultinomialNB()
train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
print("Naive Bayes type: MultinomialNB")
print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

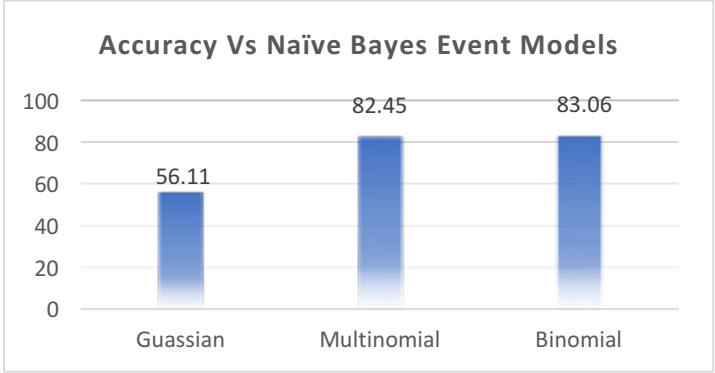
```
Naive Bayes type: MultinomialNB
1st fold accuracy- 82.58%
2nd fold accuracy- 82.53%
3rd fold accuracy- 82.23%
Average fold accuracy- 82.45%
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
clf = GaussianNB()
train_acc = cross_val_score(clf, X_train, y_train, cv=5, scoring="accuracy")
print("Naive Bayes type: GaussianNB")
print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
Naive Bayes type: GaussianNB
1st fold accuracy- 55.61%
2nd fold accuracy- 56.22%
3rd fold accuracy- 56.51%
Average fold accuracy- 56.11%
```

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import cross_val_score
clf = BernoulliNB(alpha=0.01)
train_acc = cross_val_score(clf, X_train, y_train, cv=5, scoring="accuracy")
print("Naive Bayes type: BernoulliNB")
print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
Naive Bayes type: BernoulliNB
1st fold accuracy- 82.87%
2nd fold accuracy- 83.10%
3rd fold accuracy- 83.21%
Average fold accuracy- 83.06%
```



The Binomial event model's accuracy is found to be 83.06 %.

- **Step5**: Performing the accuracy of the model over training data

  Choosing Binomial event model of Naïve Bayes, as the model gives better results as compared to other test cases. Using this value, the model is tested for its accuracy over the training data.

```
#Choosing  BernoulliNB as the event model for this data
#Performing the accuracy test on the training data
```

```
clf = BernoulliNB()
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(clf, X_train, y_train, cv=10)
accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy: %.2f%%" % (accuracy*100))
```

```
Training Accuracy: 83.08%
```

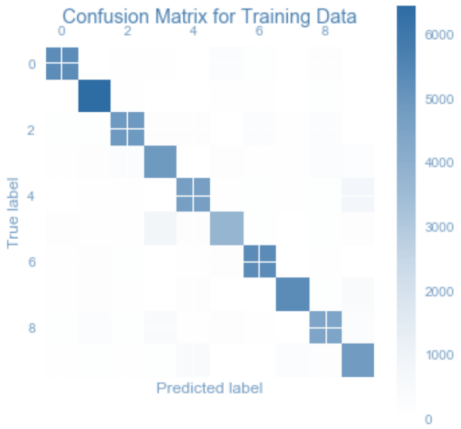
  It is observed that the model's accuracy over the training set using the **cross_val_predict** has been found to be 83.08%.

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the Naïve Bayes classifier model.

```
#Printing the confusion matrix for the training set
```

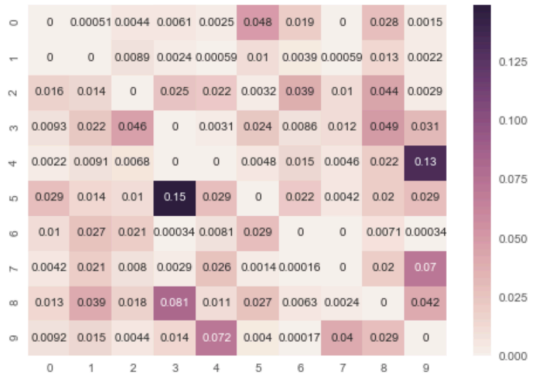
```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step7**: Error confusion matrix for the training set

  Error confusion matrix for the training set model is plotted to analyze the misclassification of values by the Naïve Bayes classifier model.

```
import seaborn as sns
val_row= train_conf_mat.sum(axis=1, keepdims=True)
train_err_conf_mat = train_conf_mat / val_row
np.fill_diagonal(train_err_conf_mat, 0)
ax = sns.heatmap(train_err_conf_mat,annot=True)
plt.show()
```



- **Step8**: Fitting the model and testing

  The model is fitted and tested for its accuracy. The accuracy is found to be 84.13 %.

```
#Chose BernoulliNB as the event model and performing the fit
#Testing the model
#Calculating the accuracy of the test data
```

```
clf = BernoulliNB()
clf.fit(X_train, y_train)
y_pred_test = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print("Test Data Accuracy: %.2f%%" % (accuracy*100))
```

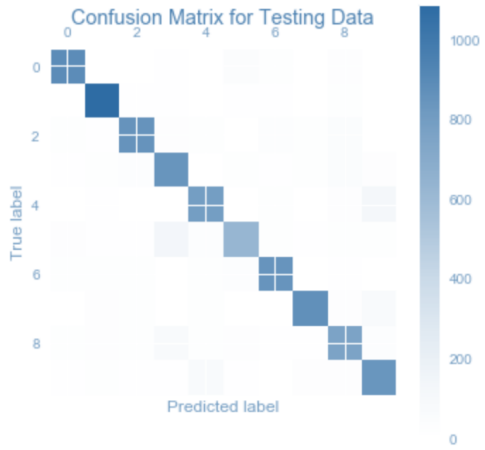
```
Test Data Accuracy: 84.13%
```

- **Step9**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the  confusion matrix
```

```
test_conf_mat = confusion_matrix(y_test,y_pred_test)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

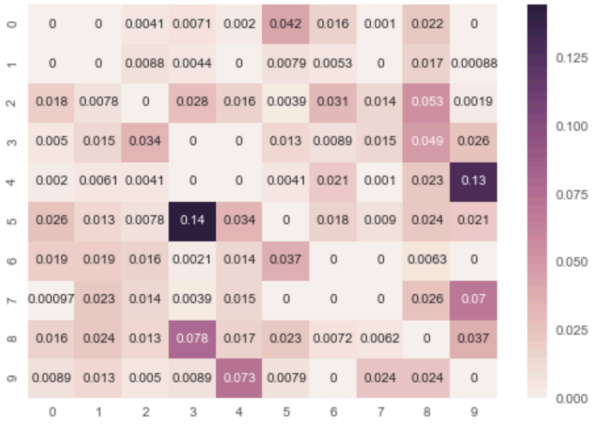


- **Step10**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the linear classifier model.

```
#Priting the error values of the confusion matrix
```

```
row_sums = test_conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = test_conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
plt.show()
```



7.3 Observation Summary

- The model was trained and built using the sklearn library
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- **event models such as Bernoulli's, Multinomial and Gaussian** were fitted for the training data. Accuracy of each model is estimated and it was found that Bernoulli's performance was good.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 8

Random Forest Classifier

8.1 Introduction [11]

Random Forest classification algorithm is called as the ensemble algorithm- algorithms that combines two or more different algorithms for the purpose of classification. These are the algorithms that whenever an input is provided randomly a tree is selected with sample replacement. In doing so almost one third of the data is left during the sample replacement. The left-out data is used to get an impartial data as the trees are added to the forest. Importance of the variable can also be extracted. The proximities for each pair of cases are estimated after the construction of each tree and all data are run down. Proximity is incremented by 1 when 2 cases take the same terminal node. Once the run is completing the proximities are normalized by dividing with the number of trees.

8.2 Training and Fitting to MNIST Dataset

In this section, Random Forest model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the Random Forest network model, validating and testing.

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the linear model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

  The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

Using cross_val_score, the accuracy of the Random Forest classifier model is estimated over the dataset by splitting it, fitting the model and computing the score for three consecutive times (as three folds are created) for different no of neurons in the hidden layer.

- **Step4**: Selection of event model with cross validation

  To perform the cross-validation **cross_val_score**, simple helper function to perform cross-validation, is used. This function is called on the dataset as well as the estimator.

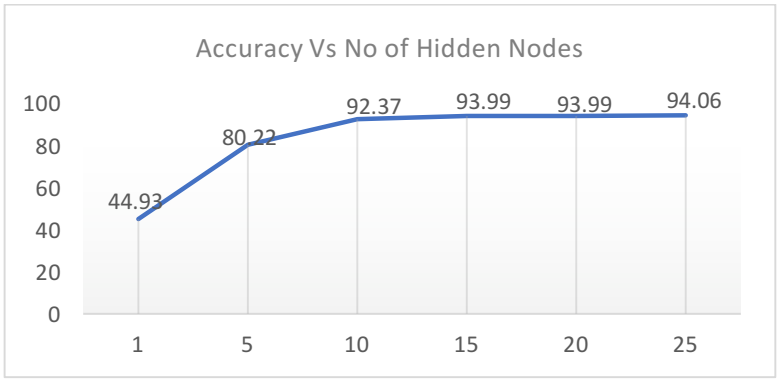
  In here different for different values of max_depth of the tree, model is trained and accuracy of each model is evaluated.

```
# Building the Random Forest model by changing the max_depth
# Used the cros_val_score cross validation method
# For different event model
# Accuracy of all the 3 folds and their average is calculated
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
for i in (1,5,10,15,20,25):
    clf = RandomForestClassifier(max_depth=i, random_state=0)
    train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
    print("For max_depth= " +str(i))
    print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
    print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
    print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
    print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
For max_depth= 1
1st fold accuracy- 46.02%
2nd fold accuracy- 45.46%
3rd fold accuracy- 43.32%
Average fold accuracy- 44.93%
For max_depth= 5
1st fold accuracy- 80.31%
2nd fold accuracy- 79.37%
3rd fold accuracy- 80.98%
Average fold accuracy- 80.22%
For max_depth= 10
1st fold accuracy- 92.41%
2nd fold accuracy- 92.35%
3rd fold accuracy- 92.36%
Average fold accuracy- 92.37%
```

```
For max_depth= 15
1st fold accuracy- 94.14%
2nd fold accuracy- 93.96%
3rd fold accuracy- 93.86%
Average fold accuracy- 93.99%
For max_depth= 20
1st fold accuracy- 93.92%
2nd fold accuracy- 93.86%
3rd fold accuracy- 94.18%
Average fold accuracy- 93.99%
For max_depth= 25
1st fold accuracy- 94.31%
2nd fold accuracy- 93.94%
3rd fold accuracy- 93.92%
Average fold accuracy- 94.06%
```



It is found the for max_depth = 25 the accuracy started to saturate.

- **Step5**: Performing the accuracy of the model over training data

Choosing max_depth = 25, as the model gives better results as compared to other test cases. Using this value, the model is tested for its accuracy over the training data.

```
#Choosing max_depth = 25
#Performing the accuracy test on the training data
```

```
clf =  RandomForestClassifier(max_depth=25, random_state=0)
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(clf, X_train, y_train, cv=10)
accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy: %.2f%%" % (accuracy*100))
```

```
Training Accuracy: 94.70%
```

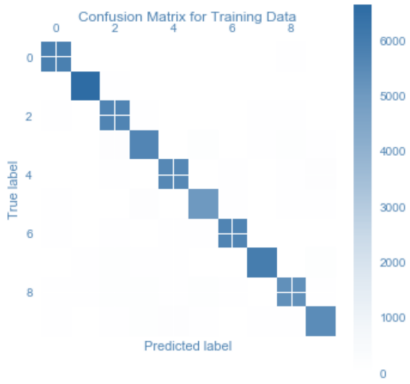
It is observed that the model's accuracy over the training set using the **cross_val_predict** has been found to be 94.70 %.

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the Random Forest classifier model.

```
#Printing the confusion matrix for the training set
```

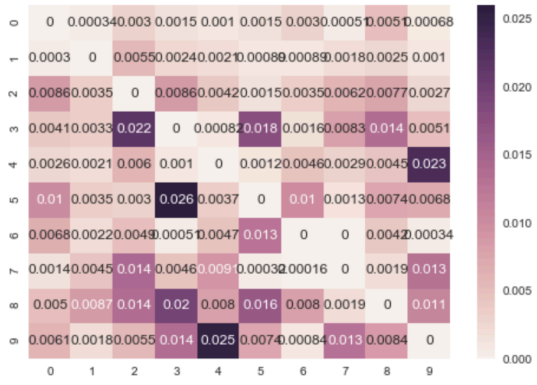
```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step7**: Error confusion matrix for the training set

  Error confusion matrix for the training set model is plotted to analyze the misclassification of values by the Random Forest classifier model.

```
import seaborn as sns
val_row= train_conf_mat.sum(axis=1, keepdims=True)
train_err_conf_mat = train_conf_mat / val_row
np.fill_diagonal(train_err_conf_mat, 0)
ax = sns.heatmap(train_err_conf_mat,annot=True)
plt.show()
```



- **Step8**: Fitting the model and testing

The model is fitted and tested for its accuracy. The accuracy is found to be 94.71 %.

```
#Chose max_depth = 25
#Testing the model
#Calculating the accuracy of the test data
```

```
clf =  RandomForestClassifier(max_depth=25, random_state=0)
clf.fit(X_train, y_train)
y_pred_test = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print("Test Data Accuracy: %.2f%%" % (accuracy*100))
```

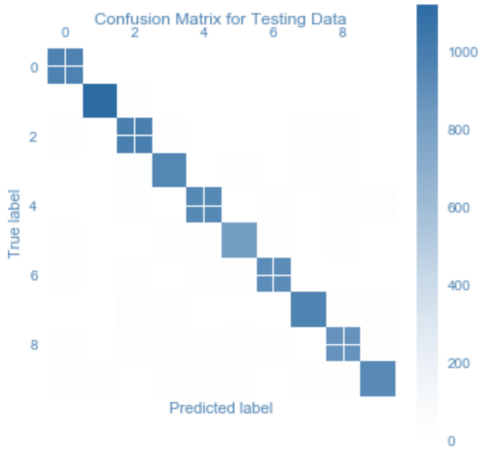
```
Test Data Accuracy: 94.71%
```

- **Step9**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the  confusion matrix
```

```
test_conf_mat = confusion_matrix(y_test,y_pred_test)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

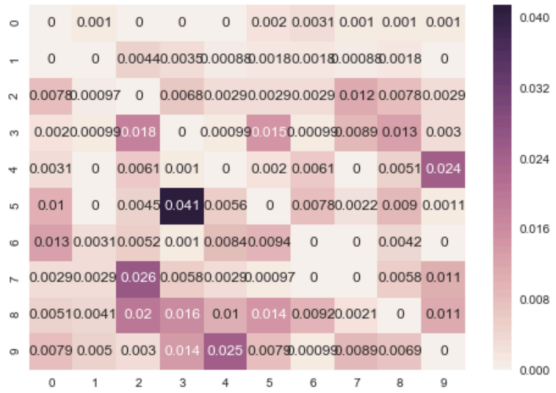


- **Step10**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the Random Forest classifier model.

```
#Priting the error values of the confusion matrix
```

```
row_sums = test_conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = test_conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
plt.show()
```



8.3 Observation Summary

- The model was trained and built using the sklearn library
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- **maximum depth of the tree (max_depth)** were varied for the training data and accuracy of each depth is estimated.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 9

Decision Tree Classifier

9.1 Introduction [12]

Decision Tree classification algorithm belongs to the family of supervised learning. This algorithm can also be used for both classification and regression problem solving. The algorithm works by building a decision tree and answers and questions to the input received to the trees. From the root node, the test data is passed to the bottom of the tree structure and at each instant the node makes a decision of yes or no and then takes the necessary actions and forwards the data to the bottom. Once it reaches the end of the node the traversed with decision will be evaluated and a value will be predicted.

9.2 Training and Fitting to MNIST Dataset

In this section, Decision Tree model has been trained and tested to the MNIST dataset. The following screenshots describes the process of training the Decision Tree model, validating and testing.

- **Step1**: Importing the required libraries

  Initially, the required libraries to train, test, plot, creating confusion matrices are all extracted.

```
#importing the required libraries
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

- **Step2**: Extracting the MNIST dataset

  The require dataset-MNIST, is extracted with which the linear model is trained and tested.

```
#Importing the MNIST data set
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
X, y = mnist["data"], mnist["target"]
```

- **Step3**: Splitting the dataset into training and testing set

  The data has been split into training and testing data. The data is divided into proper proportion to minimize the training error as compared to test error.

```
#Splitting the dataset into training set and testing set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000]
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

Using cross_val_score, the accuracy of the Decision Tree classifier model is estimated over the dataset by splitting it, fitting the model and computing the score for three consecutive times (as three folds are created) for different no of neurons in the hidden layer.

- **Step4**: Selection of event model with cross validation

  To perform the cross-validation **cross_val_score**, simple helper function to perform cross-validation, is used. This function is called on the dataset as well as the estimator.

  In here different for different values of max_depth of the tree, model is trained and accuracy of each model is evaluated.

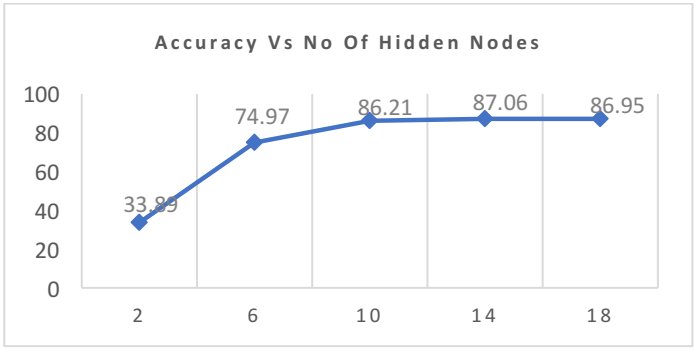
```
from sklearn import tree

for i in range(2, 20, 4):
    clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth = i)
    from sklearn.model_selection import cross_val_score
    train_acc = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
    print("Max Depth -: " +str(i))
    print("1st fold accuracy- %.2f%%" %(100*train_acc[0]))
    print("2nd fold accuracy- %.2f%%" %(100*train_acc[1]))
    print("3rd fold accuracy- %.2f%%" %(100*train_acc[2]))
    print("Average fold accuracy- %.2f%%" %((100*train_acc[2] + 100*train_acc[1] + 100*train_acc[0])/3))
```

```
Max Depth -: 2
1st fold accuracy- 32.58%
2nd fold accuracy- 35.83%
3rd fold accuracy- 33.25%
Average fold accuracy- 33.89%
Max Depth -: 6
1st fold accuracy- 75.58%
2nd fold accuracy- 74.77%
3rd fold accuracy- 74.55%
Average fold accuracy- 74.97%
```

```
Max Depth -: 10
1st fold accuracy- 86.31%
2nd fold accuracy- 86.79%
3rd fold accuracy- 85.53%
Average fold accuracy- 86.21%
Max Depth -: 14
1st fold accuracy- 87.14%
2nd fold accuracy- 87.18%
3rd fold accuracy- 86.87%
Average fold accuracy- 87.06%
```

```
Max Depth -: 18
1st fold accuracy- 86.92%
2nd fold accuracy- 87.21%
3rd fold accuracy- 86.73%
Average fold accuracy- 86.95%
```



It is found the for max_depth = 25 the accuracy started to saturate hence choosing the default max_depth value.

- **Step5**: Performing the accuracy of the model over training data

  Choosing default max_depth value as the model gives better results as compared to other test cases. Using this value, the model is tested for its accuracy over the training data.

```
clf = tree.DecisionTreeClassifier(criterion='entropy',random_state=42)
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(clf, X_train, y_train, cv=5)
accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy: %.2f%%" % (accuracy*100))
```

```
Training Accuracy: 87.61%
```

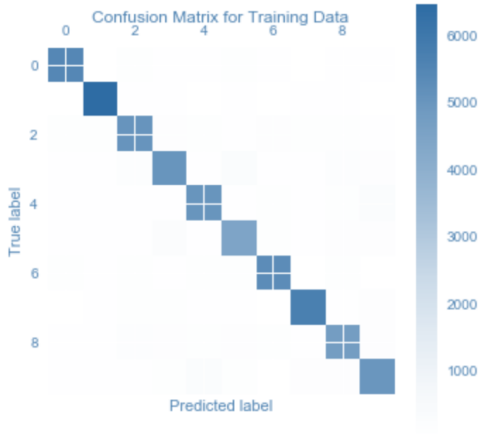
It is observed that the model's accuracy over the training set using the **cross_val_predict** has been found to be 87.61 %.

- **Step6**: Confusion matrix for the training set

  Confusion matrix for the training set model is plotted to analyze the false and true prediction of values by the Decision Tree classifier model.

```
#Printing the confusion matrix for the training set
```

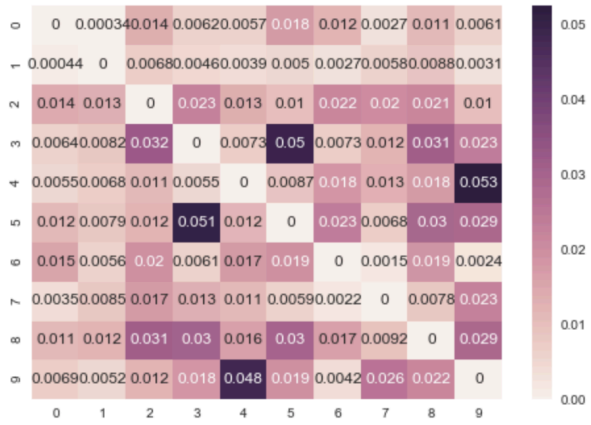
```
train_conf_mat = confusion_matrix(y_train,y_train_pred)
plt.matshow(train_conf_mat)
plt.title('Confusion Matrix for Training Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



- **Step7**: Error confusion matrix for the training set

  Error confusion matrix for the training set model is plotted to analyze the misclassification of values by the Random Forest classifier model.

```
import seaborn as sns
val_row= train_conf_mat.sum(axis=1, keepdims=True)
train_err_conf_mat = train_conf_mat / val_row
np.fill_diagonal(train_err_conf_mat, 0)
ax = sns.heatmap(train_err_conf_mat,annot=True)
plt.show()
```



- **Step8**: Fitting the model and testing

  The model is fitted and tested for its accuracy. The accuracy is found to be 88.55 %.

```
#Chose default max_depth
#Testing the model
#Calculating the accuracy of the test data
```

```
clf = tree.DecisionTreeClassifier(criterion='entropy', random_state=42)
clf.fit(X_train, y_train)
y_pred_test = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print("Test Data Accuracy: %.2f%%" % (accuracy*100))
```

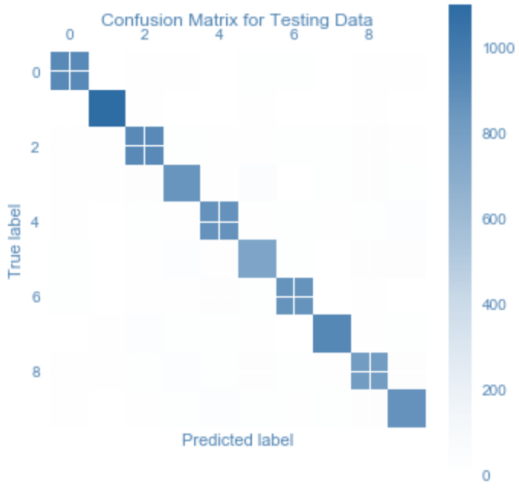
```
Test Data Accuracy: 88.55%
```

- **Step9**: Confusion matrix for the testing set

  Confusion matrix for the testing set model is plotted to analyze the false and true prediction of values by the linear classifier model.

```
#Printing the  confusion matrix
```

```
test_conf_mat = confusion_matrix(y_test,y_pred_test)
plt.matshow(test_conf_mat)
plt.title('Confusion Matrix for Testing Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

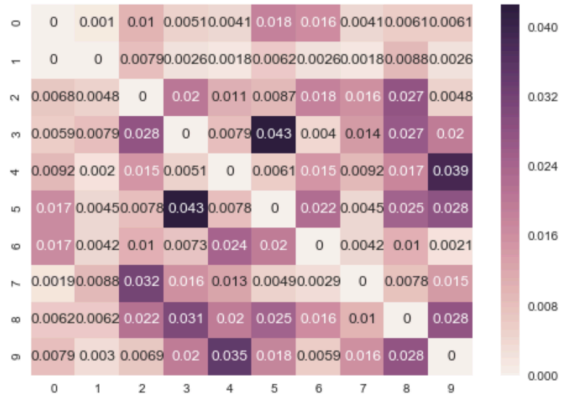


- **Step10**: Error confusion matrix for the testing set

  Error confusion matrix for the testing set model is plotted to analyze the misclassification of values by the Random Forest classifier model.

```
#Priting the error values of the confusion matrix
```

```
row_sums = test_conf_mat.sum(axis=1, keepdims=True)
norm_conf_mx = test_conf_mat / row_sums
np.fill_diagonal(norm_conf_mx, 0)
ax = sns.heatmap(norm_conf_mx,annot=True)
plt.show()
```



8.3 Observation Summary

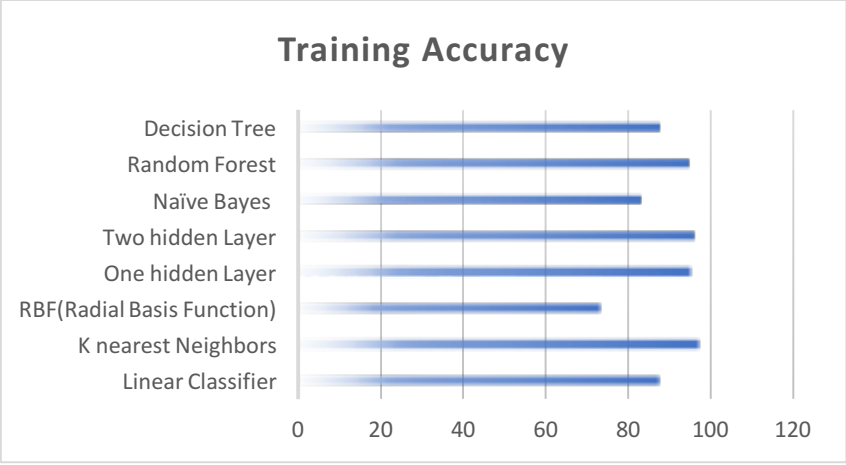
- The model was trained and built using the sklearn library
- During the training, **cross_val_score** cross-validation method was used.
- During each folds of the dataset, hyper parameter- **maximum depth of the tree (max_depth)** were varied for the training data and accuracy of each depth is estimated.
- Confusion matrix and error matrix for both training and testing datasets are included.

Chapter 10

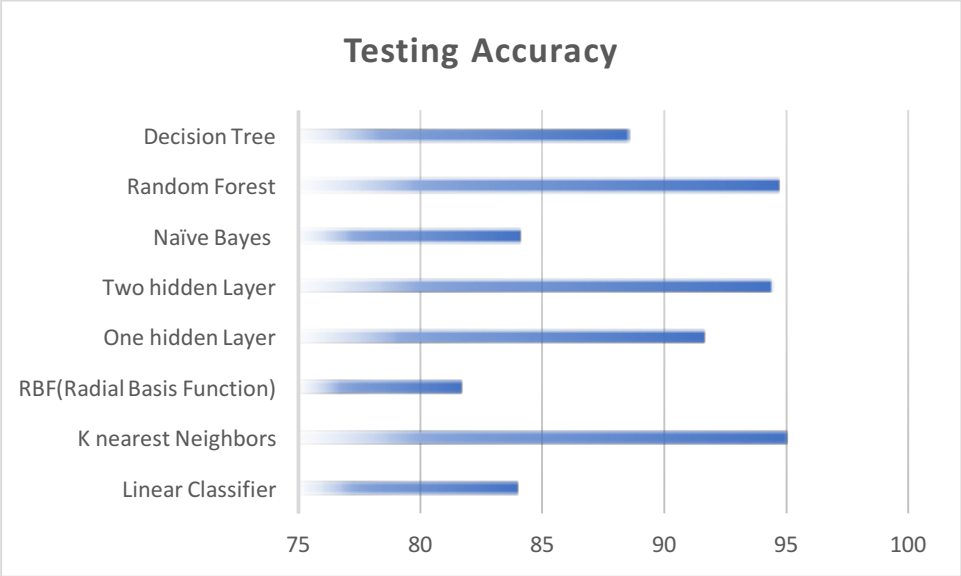
Classification Model Comparison

Using the MNIST dataset, the classification models such as Linear Classifier Model, KNN Classification Model, Radial Basis Neural Network, Single and Double Hidden Layer Network, Naïve Bayes, Random Forest and Decision Tree Classifiers were trained. During the training phase, hyper parameters were varied and their accuracy was estimated. Obtaining the best value of the hyper-parameter the model was tested using test data and its accuracy is calculated. In each classification techniques cross_val_score cross-validation technique is used.

Classifiers	Cross Validation	Hyperparameter	Training accuracy	Testing accuracy
Linear Classifier	cross_val_score	Number of iterations	87.31	84.04
K nearest Neighbors		Nighbor Values	96.97	95.03
RBF(Radial Basis Function)		Maximum iterations	73	81.71
One hidden Layer	cross_val_score	Number of hidden nodes	95.01	91.7
Two hidden Layer		Number of hidden nodes(H1 & H2)	95.94	94.4
Naïve Bayes		Event models	83.08	84.13
Random Forest		Depth of the tree	94.7	94.71
Decision Tree		Depth of decision Tree	87.61	88.55



From the above chart, it is observed that the best training accuracy has been achieved by KNN classifier.



From the above chart, it is observed that the best testing accuracy has been achieved by KNN classifier

From the above chart and table, it can be said that KNN classifier performs well on MNIST dataset. The performance of other classifiers was also found to be good.

Reference:

[1]. Fatahi, M. (n.d.). MNIST handwritten digits Description and using. Retrieved …. November 8, 2-17

[2] M. Wu and Z. Zhang, "Handwritten Digit Classification using the MNIST Data Set," 2010.

[3] Decoste, D., & Cristianini, N. (2002, August). Figure 3. The first 100 MNIST training images, with class labels. . Retrieved November 8, 2017, from https://www.researchgate.net/figure/2924845_fig1_Figure-3-The-first-100-MNIST-training-images-with-class-labels

[4] Linear classifier. (2017, October 07). Retrieved November 13, 2017, from https://en.wikipedia.org/wiki/Linear_classifier

[5] K-nearest neighbors algorithm. (2017, November 10). Retrieved November 13, 2017, from https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[6] Radial basis function network. (2017, October 19). Retrieved November 13, 2017, from https://en.wikipedia.org/wiki/Radial_basis_function_network

[7]https://github.com/nfmcclure/tensorflow_cookbook/blob/master/04_Support_Vector_Machines/04_Working_with_Kernels

[8] Intelichencia artificial. (2017, October 27). Retrieved November 13, 2017, from https://an.wikipedia.org/wiki/Intelichencia_artificial

[9] Edvinsson, J. (n.d.). A Neural Network Primer. Retrieved November 13, 2017, from https://technology.condenast.com/story/a-neural-network-primer

[10] Naive Bayes classifier. (2017, November 10). Retrieved November 13, 2017, from https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[12] "Random Forests Leo Breiman and Adele Cutler." Random forests - classification description. Accessed November 13, 2017. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[12] (n.d.). Retrieved November 13, 2017, from http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/lguo/decisionTree.html