LOGBOOK FOR CP60032E

Natural Language Interface

Module leader: Massoud Zolgharni

Pouya Shahverdi Moghaddam
ID: 21245645

Table of Contents

Week 8 ..... *2*
Task 1 2
Task 2 2
Task 3 3
Week 9 ..... *6*
Task 1 6
Task 2 6
Task 3 7
Week 10 ..... *9*
Task 1 9
Task 2 10
Task 3 12
Week 10 ..... *14*
Task 1 14
Task 2 15
Task 3 16
Week 12 ..... *18*
Task *18*
Appendix 25

Week 8

Task 1

Explain how can we make sure the decision trees used in Random Forest Classifier are decorrelated?
classifier.

The classifier works by trying to reduce issues with correlation by feature selection in terms of subsample of the feature space at each split.
The classifiers aim is to make the trees de-correlated and to build large numbers of trees that are also de-correlated with the trees being cut for splitting the node.

With random forest there will be some trees that are identical because of the nature of the classifier with the nature in which trees can be repeated more than once.

Task 2

For the example of Iris dataset, compute the accuracy of your developed Random Forest Classifier. Tip: you can use the confusion matrix.

To achieve the correct accuracy, we will use the confusion matrix code from bellow. This will produce a table showing the accuracy.

```
1. print(pd.crosstab(test['species'], preds, rownames=['Actual
   Species'], colnames=['Predicted Species']))
```

Predicted Species	setosa	versicolor	virginica
Actual Species			
setosa	13	0	0
versicolor	0	5	2
virginica	0	0	12

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

$$Accuracy = \frac{13+5+12}{13+5+12+2} = \frac{30}{32} = 0.9375$$

```
cm=np.array(cnf_matrix)

def accuracy(confusion_matrix):

    diagonal_sum = confusion_matrix.trace()
    print('sum of diagonal elements=',diagonal_sum)
    sum_of_all_elements = confusion_matrix.sum()
    print('sum_of_all_elements = ',sum_of_all_elements)
    ACC=diagonal_sum/sum_of_all_elements
    print ('Accuracy= ',ACC)
    return ACC
accuracy(cm)
```

```
sum of diagonal elements= 30
sum_of_all_elements =  32
Accuracy=  0.9375
```

Task 3

Try Random Forest Classifier for the Iris dataset example, and when creating decision trees from the bootstrapped dataset, use different number of features:

- a subset of 2 features
- a subset of 3 features

And compare the accuracy of your Random Forest Classifier. Which one is better? What is the largest number of features you may have? Tip: you can specify the number of features using max_features parameter in RandomForestClassifier.

By using the max_features parameter we can specify the number of feature and then work out the accuracy by using the formula below.

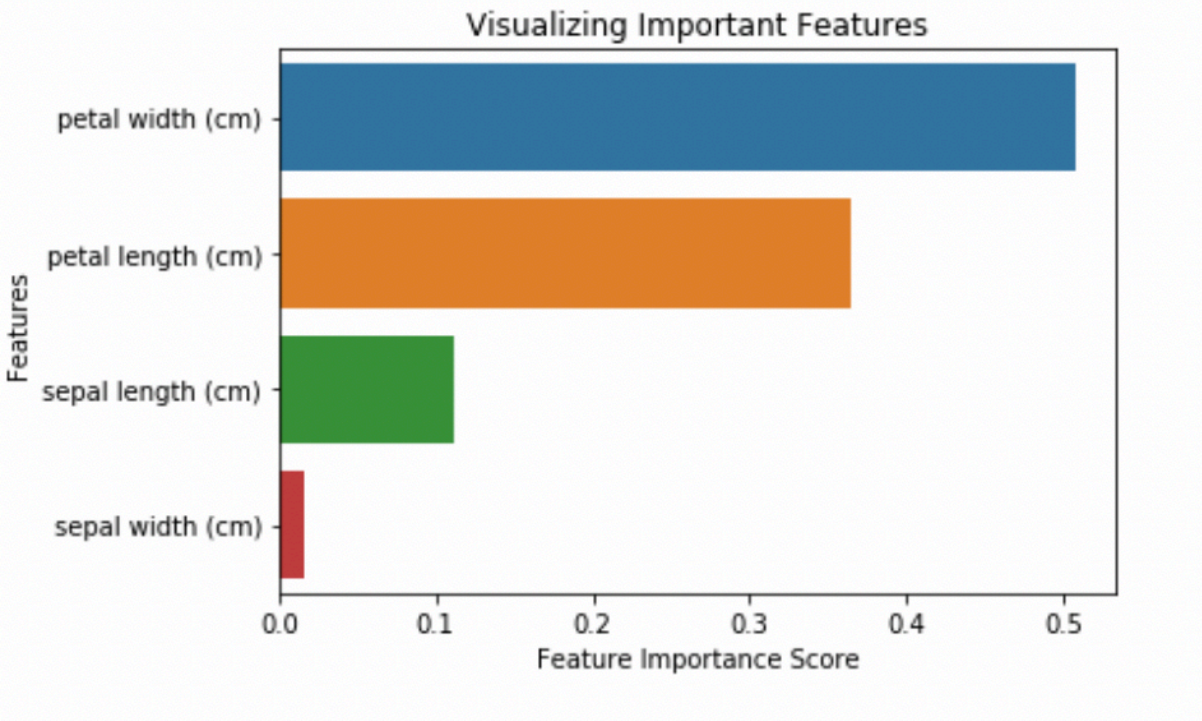
$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

```
clf = RandomForestClassifier(max_features=2, random_state=0)
```

Subset of 2 features :

Predicted Species	setosa	versicolor	virginica
Actual Species			
setosa	13	0	0
versicolor	0	5	2
virginica	0	0	12

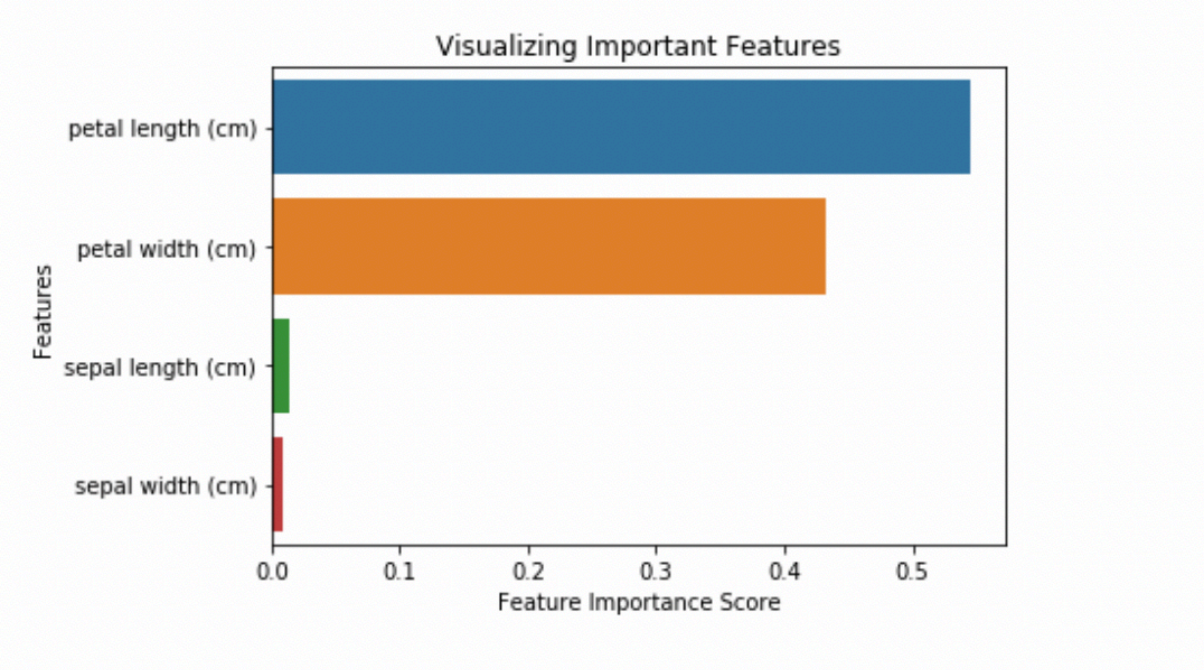
$$Accuracy = \frac{13+5+12}{13+5+12+2} = \frac{30}{32} = = 0.9375$$



3 features:

Predicted Species	setosa	versicolor	virginica
Actual Species			
setosa	13	0	0
versicolor	0	6	1
virginica	0	0	12

$$Accuracy = \frac{13+6+12}{13+6+12+1} = \frac{31}{32} = = 0.96875$$



From the above calculations of 2 and 3 features we can see that the accuracy of feature 3 is higher at 0.968 compared to “2 features” which is calculated at 0.937.

With regards to the maximum features, we can change the features number below, however anything above 4 features will result in an error. This is due to the dataset having only four columns of features in the code. This is shown in the graph and code snippet below.

```
# Create a list of the feature column's names
features = df.columns[:4]

# View features
features
```

4 features:

Predicted Species	setosa	versicolor	virginica
Actual Species			
setosa	13	0	0
versicolor	0	5	2
virginica	0	0	12

Week 9

Task 1

Explain the time complexity of k-nearest neighbours (k-NN) algorithm for image classification?

We have to consider the limitation of KNN when used for image classification and this is due to the slow nature of the algorithm when the algorithm is at training phase. At this stage the process can be slow and can increase in time as the training dataset increases in size. It's simply not a very good algorithm for image classification due to the large dimensional data.

In order to determine who are the nearest neighbours you need to take the testing set and compare them one by one to the training sets however this method is slow and it involves n times d (i.e O (nd)) comparisons, where d is dimensionality and n is the number of instances, instead of comparing the all the training examples we should instead try to reduce d using dimensionality reduction or reduce n. With the reduction of n we end up identifying and guessing the potential nearest neighbours m where m is bigger than k but smaller than n i.e m<<n

In order to find the nearest potential we can use K-Decision tree that works on low dimensional and real valued data

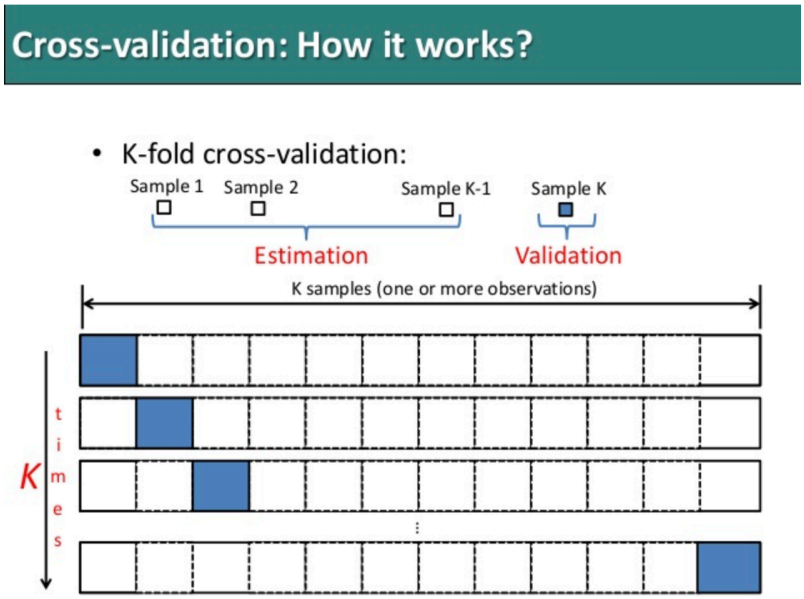
Task 2

Describe one method for setting Hyperparameters in k-NN classififcation method.

A method we can use for setting hyperparameter in KNN is k-Fold Cross validation, this is a resampling method used to data and it involves using a parameter called "k", this is unrelated to the K used in KNN. The k is the number of subsets that the data is divided into.

This method involves with dividing the training dataset randomly into "k groups" or fold into approximately equal sets.

The first fold of data is used as a validation set with k subset to train the model into the remaining k-1 folds. The process is then repeated k times with a different group of folds used as a validation set with the end result involving in K estimates of the test error that are averaged out.



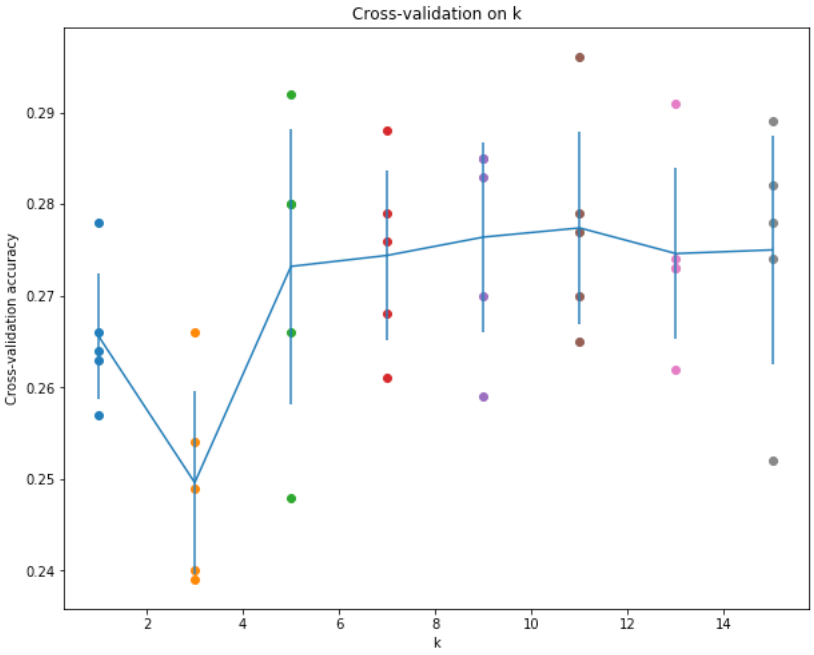
Task 3

Based on the cross-validation results in the Introduction exercises above, choose the best value for k in k-NN. Then retrain the image classifier using all CIFAR10 training data, and test it on the test data.

```
k = 1, accuracy = 0.263000
k = 1, accuracy = 0.257000
k = 1, accuracy = 0.264000
k = 1, accuracy = 0.278000
k = 1, accuracy = 0.266000
k = 3, accuracy = 0.239000
k = 3, accuracy = 0.249000
k = 3, accuracy = 0.240000
k = 3, accuracy = 0.266000
k = 3, accuracy = 0.254000
k = 5, accuracy = 0.248000
k = 5, accuracy = 0.266000
k = 5, accuracy = 0.280000
k = 5, accuracy = 0.292000
k = 5, accuracy = 0.280000
k = 7, accuracy = 0.261000
k = 7, accuracy = 0.279000
k = 7, accuracy = 0.268000
k = 7, accuracy = 0.288000
k = 7, accuracy = 0.276000
k = 9, accuracy = 0.259000
k = 9, accuracy = 0.283000
k = 9, accuracy = 0.270000
k = 9, accuracy = 0.285000
k = 9, accuracy = 0.285000
k = 11, accuracy = 0.265000
k = 11, accuracy = 0.296000
k = 11, accuracy = 0.277000
k = 11, accuracy = 0.279000
k = 11, accuracy = 0.270000
k = 13, accuracy = 0.262000
k = 13, accuracy = 0.291000
k = 13, accuracy = 0.273000
k = 13, accuracy = 0.274000
k = 13, accuracy = 0.273000
k = 15, accuracy = 0.252000
k = 15, accuracy = 0.289000
k = 15, accuracy = 0.278000
k = 15, accuracy = 0.282000
k = 15, accuracy = 0.274000
```

Looking at the results from above we can see that the highest accuracy recorded is 0.296000 and this is the K value of 11.

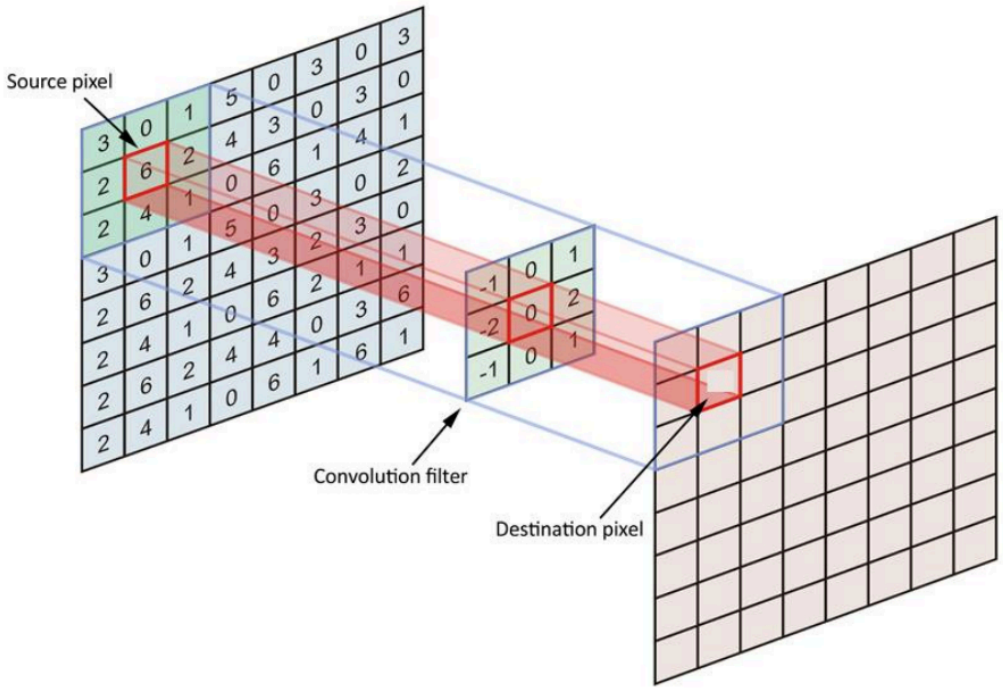
This result can be cross validated by the graph below. The graph below showing that K11 also being the best value.



Week 10

Task 1

For the convolution operation given below, compute the destination pixel value in the output image by applying the 3×3 filter to the source (input) image. Provide the details of your calculations.



From the image above we can see that the centre element of the kernel (filter) is place over the source pixel and then replaced with a weighted sum of itself and nearby pixel.

3	0	1
2	6	2
2	4	1

Source pixel

x

-1	0	1
-2	0	2
-1	0	1

Convolution filter

$$Denstination\ Pixel = \Big(\big(3 \times (-1)\big) + \big(2 \times (-2)\big) + \big(2 \times (-1)\big)\Big) + ((0 + \big((0 \times 0) + (6 \times 0) + (4 \times 0)\big) + \big((1 \times 1) + (2 \times 2) + (1 \times 1)\big) = -3$$

Task 2

What is a convolutional neural network? Simply describe your understanding of CNNs (maximum of one page, including figures, if any).

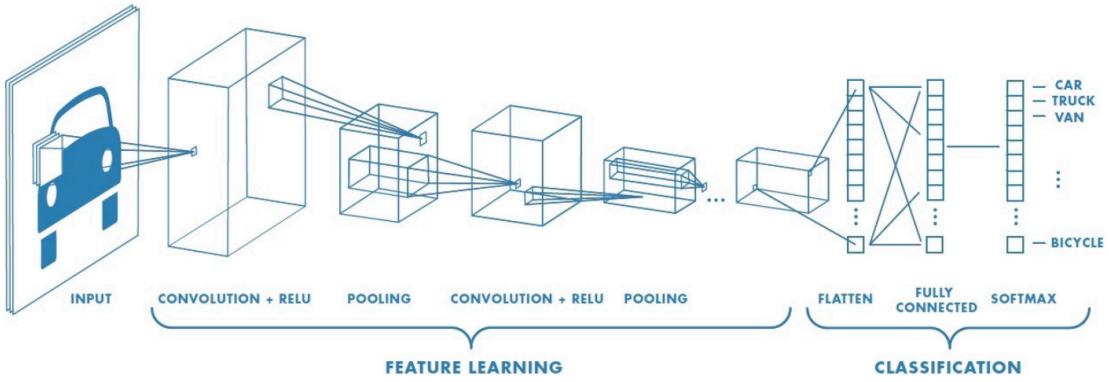
Convolutional Neural Network or as it known as CNN is a type of neural network that is widely used in image recognition and processing due to the abilities of recognizing patterns and it's specifically designed to process pixel data. Convolutional Neural Networks contain layers, an input layer and an output layer, with each layer. They perceive images as three-dimensional objects instead of the way humans see images.

For the CNN to detect patterns in images, each neuron in the convolutional layers is connected to a small portion of pixels within the image, this is the specific layer to focus in a specific layer with layers evolving to detect patterns that are more complex. A first layer can be to detect low level patterns like the edge of a image whereas deeper layers can detect complexity such as facial features and signs.

Each convolutional layer will contain specific filters/kernals with specific matrix for detections. For example a "basic" convolutional layer that has limited capabilities for edge detections can be a small 3 x3 matrix with each box containing a numbers with the number representing a pattern in the image.

The different layers in convolutional Neural Network are as follows:

Convolutional Layer, Pooling Layer, and Fully-Connected Layer and a RELU layer that can come between the Conv and Pool layer.



.

Convolution Layer

This is the first layer and is used to extract features from the input image. This layer learns the image by the pixels using a small square input. It is a mathematical operation that takes two inputs such as image matrix and filter or kernel.

Non Linearity (ReLU)

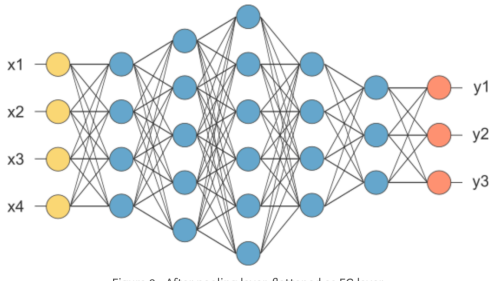
ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

Pooling layer

At this layer a reduction of parameters when the images are too large and cannot be processed. Spital pooling which is also called subsampling, this is the method that reduced the dimensionality of each map but also holding as much information as possible. There are different types of pooling such as max pooling, average pooling and sum pooling

Fully Connected Layer

At this layer the vector that has been flattened by out matric will feed into a connected layer like neural network like the image below. The feature map matrix that has been converted as vector x1, x2 etc with the fully connected layers, we have a combined the features together and created a model with the end result being a activation function such as softmax or sigmoid to classify the outputs



Task 3

What is the best validation accuracy you achieved in the Introduction section? How many neurons would you consider for your hidden layer? And what is your test accuracy?

```
hidden_size = [80, 100, 120]
learning_rate = [1e-4, 1e-3]
reg = [0.2, 0.4]
best_acc = -1

log = {}

for hs in hidden_size:
    for lr in learning_rate:
        for r in reg:

            # Set up the network
            net = TwoLayerNet(input_size, hs, num_classes)

            # Train the network
            stats = net.train(X_train, y_train, X_val, y_val,
                        num_iters=1000, batch_size=200,
                        learning_rate=lr, learning_rate_decay=0.95,
                        reg=r, verbose=False)

            acc = stats['val_acc_history'][-1]
            log[(hs, lr, r)] = acc

            # Print Log
            print('for hs: %e, lr: %e and r: %e, valid accuracy is: %f'
                    % (hs, lr, r, acc))

            if acc > best_acc:
                best_net = net
                best_acc = acc
                HS = hs

print('Best Networks has a Validation Accuracy of: %f' % best_acc)
print('Best Networks has %s neurons in the hidden layer ' % HS)


# visualize the weights of the best network
show_net_weights(best_net)


test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

```
for hs: 1.200000e+02, lr: 1.000000e-03 and r: 4.000000e-01, valid accuracy is:
0.476000
Best Networks has a Validation Accuracy of: 0.504000
Best Networks has 120 neurons in the hidden layer
```

Best validation accuracy is 0.50400
Best network has 120 neurons in the hidden layer.

Test accuracy : 0.483

Week 10

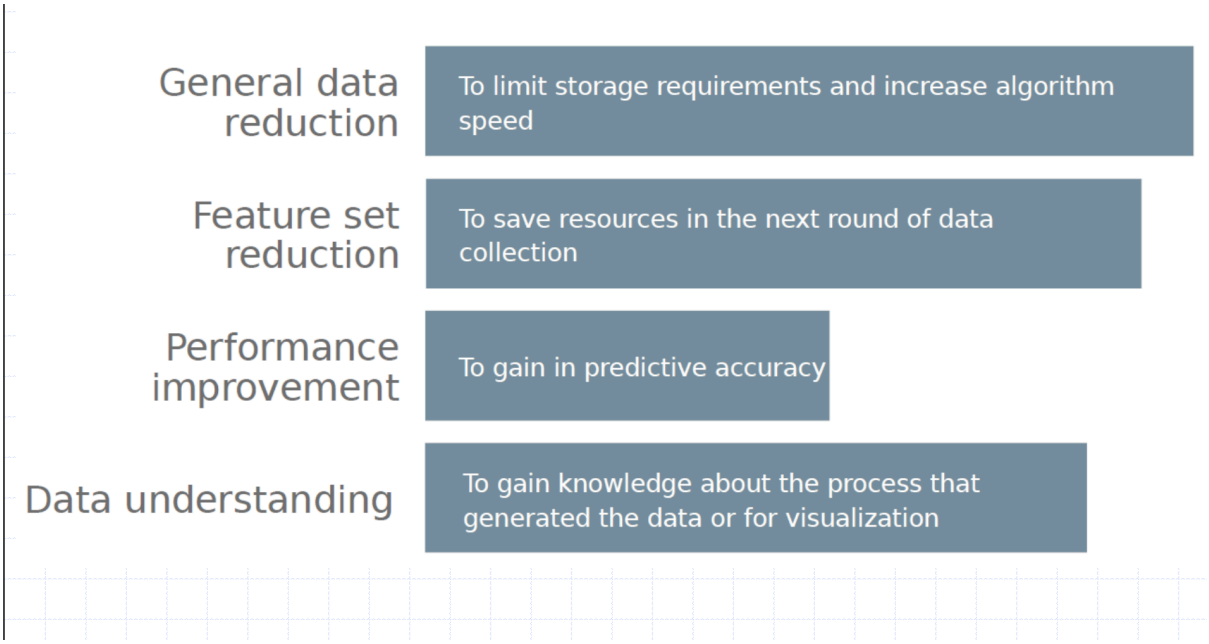
Task 1

Explain the Dimensionality Reduction and why we need it. Describe two methods to achieve Dimensionality Reduction?

This is a technique used in Machine learning to ensure that an algorithm works efficiently. It's all well and good for a particular algorithm to process data and make predictions but it also becomes useless if it takes hours, days or weeks and with large computing powers. This is where Dimensionality Reduction comes into play.

This is used to reduced large sets of variables in order for the algorithm to use the important features of the data and avoids using the unnecessary features that can slow down the process. The techniques used for this is called feature selection and feature extraction.

Dimensionality Reduction is needed because some algorithms simply do not perform well with large dataset and this helps in terms of space used to store the data. It can also help with spotting patterns more easily as it allows clearer visualization as the unrelated data is reduced thus creating a clearer pattern. There is also the situation of curse of dimensionality, this is where an increase in the number of dimensions of a dataset which means there are more entries in the vector of features that represent each observation in the corresponding Euclidean space. This is avoided with Dimensionality Reduction.



There are many methods that can be used to achieve Dimensionality Reduction but we the two I have chosen to explain are Principal Component Analysis (PCA) and Missing Value Ratio.

Principal Component Analysis (PCA)

This is one of the most widely used methods, this is a technique used to help extract new set of variables from an existing large set of variables. This new variable is referred to as Principal Components with the first one being known as a first principal component with a maximum variance in dataset. The second principal component will explain the remaining variance in the dataset and uncorrelated to the first principal component with the third principal component explaining the variance that has not been explained by the first two principal components. This can continue….

Missing Value Ratio

This is quite a straight forward method. There may be times where we are checking the data before building a model, we discover some anomalies within the dataset with some values missing. Normally one or two of these anomalies won't affect the outcome but if there is a high number of missing values in the dataset the chances of that variable being useful is low so we can set a certain threshold value in the code that will remove that variable all together.

Task 2

Describe your understanding of Principal Component Analysis.

This was briefly explained in Task 1 however I will go in more detail how with the understanding of Principal Component Analysis.

Principal Component Analysis is a method used in Dimensionality Reduction to help simplify the complexity in high dimensional data whilst retaining accuracy in trends and patterns within the data, especially with data that contains multiple features.

Principal Component Analysis is an unsupervised learning method, this means that it can find patterns and trends without reference to prior knowledge about the sample data.

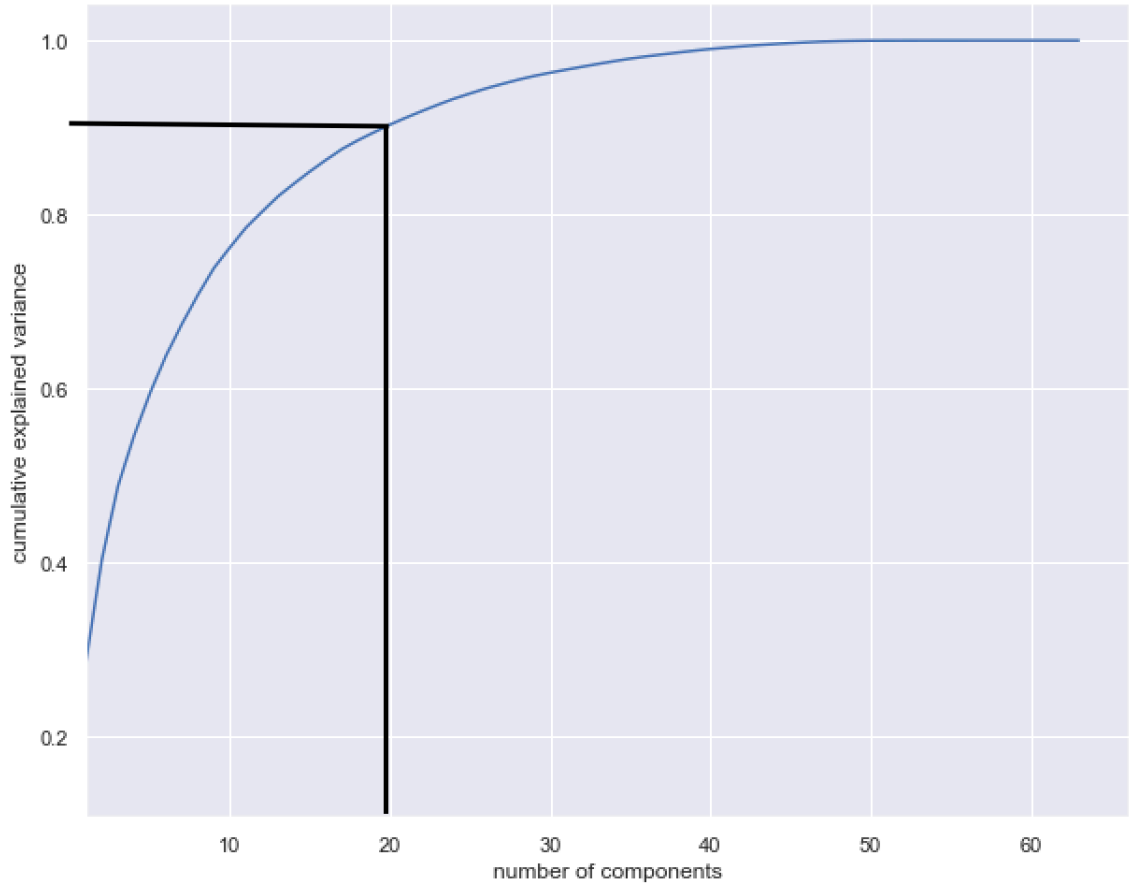
Task 3

How many principal components did you need to retain approximately 90% of the variance in the Introduction section for the problem of Hand-written digits? Justify your answer.

The function below is implemented in order to determine the number of components.

```
pca = PCA().fit(digits.data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
left, right = plt.xlim() # return the current xlim
plt.xlim((1, right)) # set the xlim to left, right
plt.show()
```

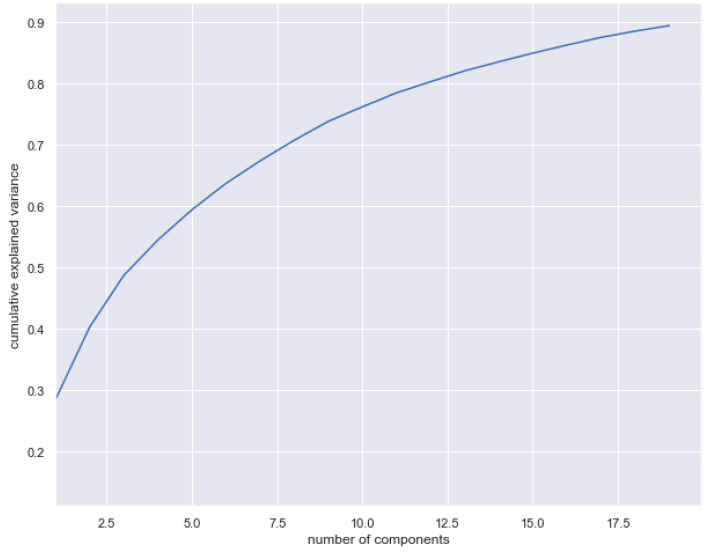
The graph below is produced, from this graph we can determine that 90% of variance will be around 20/21 components.

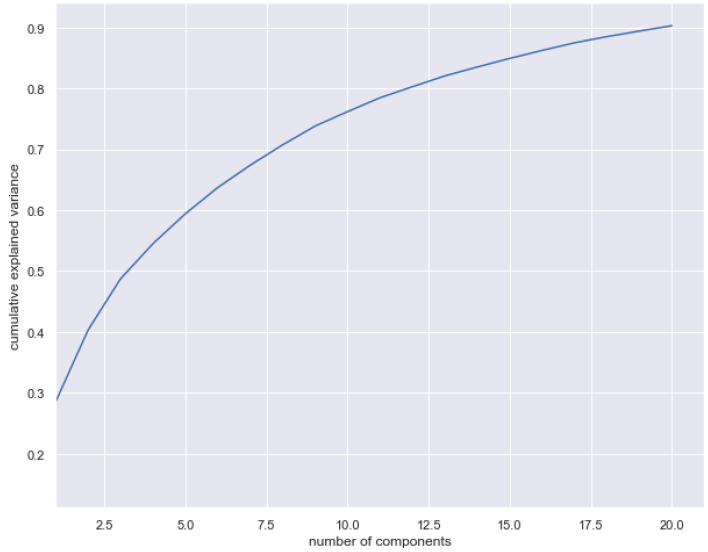


Number of components = 20

```
pca = PCA(20).fit(digits.data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
left, right = plt.xlim() # return the current xlim
plt.xlim((1, right)) # set the xlim to left, right
plt.show()
```

Using the apprimation from the graph we can change the maximim number of componets in order go get a more accurate graph.





We can see from the two above figures that the 20 components is required for 90% variance.

Week 12

Task

For this week, you are required to use Machine Learning to solve the problem of "Recognizing handwritten digits". The MNIST database of handwritten digits with their labels, available from this page (http://yann.lecun.com/exdb/mnist/), has a training set of 60,000 examples, and a test set of 10,000 examples. A range of machine learning classifiers, inspired by the various models and categories explored within the module and beyond (i.e. from reading and literature) is available to you. You must pick one classifier (e.g., SVM, K-NN, Neural Networks, etc.) and investigate its performance. This should include measures/metrics such as TP/FP rates, Precision-Recall, or other relevant metrics. For this task, you are free to train any classifier you find in Python. You are free to implement your own algorithm(s) instead of only using libraries. You will need to clearly mention your resources, and acknowledge appropriately. You will need to prepare a technical report indicating, discussing and justifying all the steps and strategies you followed to demonstrate your understanding of Machine Learning techniques. Note that your source code should be included as an appendix within your submitted logbook. Please do not submit your source code as supporting material.

To solve this particular problem, I have chosen to use the Random Forest Classifier. This method was discussed in week 9 but the classifier works by using a large collection of de-correlated decision trees and then the training data forms into a matrix as an input and with the those matrix more new matrix are created with random elements. With the created random elements a corresponding decision tree is created. This is created for the classification of the testing data.

The next stage involves the test data. Once the test data is inputted the decision trees will then classify the input test data with the predication occurring for the class to which the input will end up going into.

A step by step guide of how this has been achieved will be discussed below.

```
import sys
import numpy as np
import pickle
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from MNIST_Dataset_Loader.mnist_loader import MNIST
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

In order to start the program, we need to important libraries.

```
old_stdout = sys.stdout
log_file = open("summary.log","w")
sys.stdout = log_file


print('\nLoading MNIST Data...')
# data = MNIST('./python-mnist/data/')

data = MNIST('./MNIST_Dataset_Loader/dataset/')

print('\nLoading Training Data...')
img_train, labels_train = data.load_training()
train_img = np.array(img_train)
train_labels = np.array(labels_train)

print('\nLoading Testing Data...')
img_test, labels_test = data.load_testing()
test_img = np.array(img_test)
test_labels = np.array(labels_test)


#Features
X = train_img

#Labels
y = train_labels

print('\nPreparing Classifier Training and Validation Data...')
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,y,test_size=0.1)
```

Data is retrieved from the MNIST dataset folder with fetch four variables, two are for the labels and two are for the images.

```
print('\nRandom Forest Classifier with n_estimators = 100, n_jobs = 10')
print('\nPickling the Classifier for Future Use...')
clf = RandomForestClassifier(n_estimators=100, n_jobs=10)
clf.fit(X_train,y_train)

with open('MNIST_RFC.pickle','wb') as f:
    pickle.dump(clf, f)

pickle_in = open('MNIST_RFC.pickle','rb')
clf = pickle.load(pickle_in)

print('\nCalculating Accuracy of trained Classifier...')
confidence = clf.score(X_test,y_test)

print('\nMaking Predictions on Validation Data...')
y_pred = clf.predict(X_test)

print('\nCalculating Accuracy of Predictions...')
accuracy = accuracy_score(y_test, y_pred)

print('\nCreating Confusion Matrix...')
conf_mat = confusion_matrix(y_test,y_pred)

print('\nRFC Trained Classifier Confidence: ',confidence)
print('\nPredicted Values: ',y_pred)
print('\nAccuracy of Classifier on Validation Image Data: ',accuracy)
print('\nConfusion Matrix: \n',conf_mat)
```

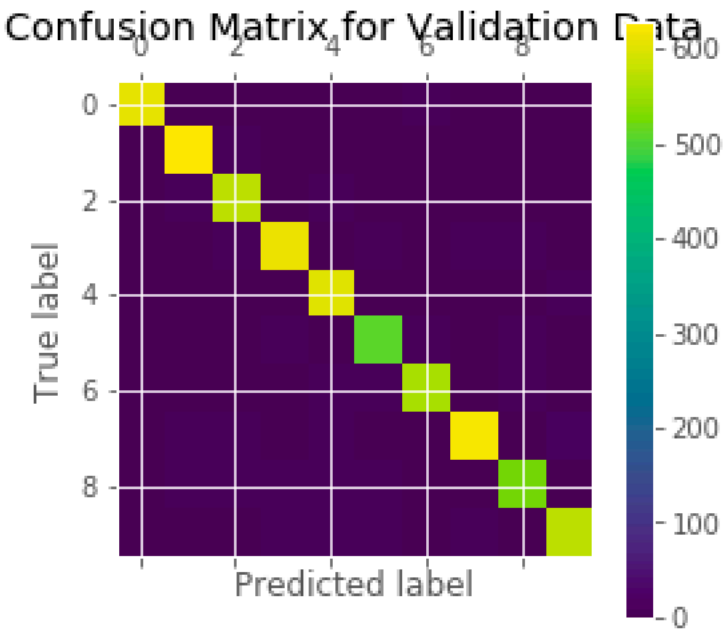
This stage of the program the classifier will be set up and fetch the dataset from the MNIST folder. This stage the accuracy of the program is also calculated as well as the confusion matrix

With the accuracy being calculated at 0.9701666666666666

	0	1	2	3	4	5	6	7	8	9
0	608	1	0	0	0	1	3	0	1	1
1	0	630	3	0	1	0	1	1	2	0
2	2	3	576	2	6	1	1	1	0	0
3	0	0	7	611	0	8	1	3	4	2
4	1	2	0	1	605	0	1	2	1	7
5	2	0	0	6	1	508	3	1	4	2
6	1	2	0	0	3	5	562	0	3	0
7	0	3	4	0	3	0	1	622	1	10
8	1	4	5	5	5	7	1	1	524	1
9	0	1	0	3	7	4	0	6	2	575

The confusion matrix above which is available in the variable explorer shows there are some incorrectly classified digits but most of the numbers have been classified correctly but at one instance there was a error of 10 times, this being at value of 7 with prediction of 9.

```
# Plot Confusion Matrix Data as a Matrix
plt.matshow(conf_mat)
plt.title('Confusion Matrix for Validation Data')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

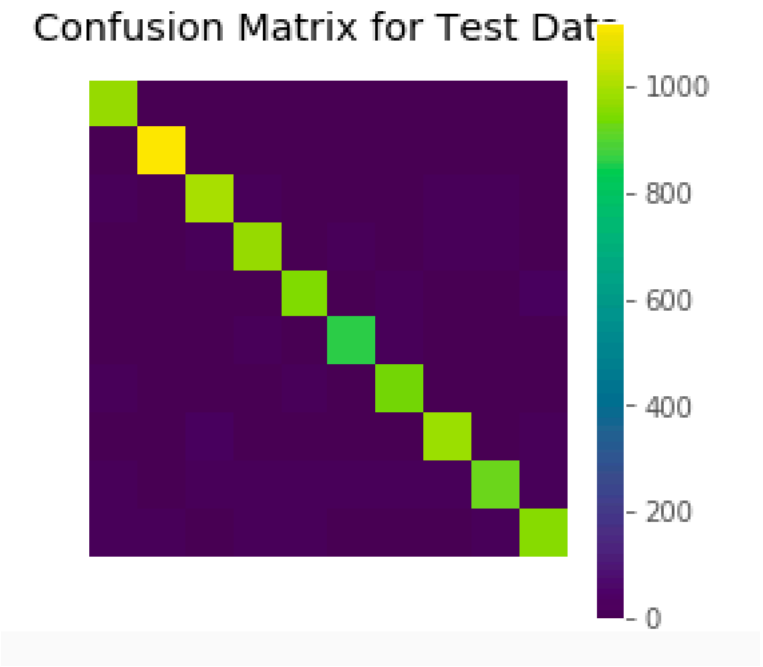


In order to get a better representation of the confusion matrix matlot library was used in order to plot the matrix. This shows the exact same at the array or “classic” matrix but you can say it comes across more professional.

```
print('\nMaking Predictions on Test Input Images...')
test_labels_pred = clf.predict(test_img)

print('\nCalculating Accuracy of Trained Classifier on Test Data... ')
acc = accuracy_score(test_labels,test_labels_pred)

print('\n Creating Confusion Matrix for Test Data...')
conf_mat_test = confusion_matrix(test_labels,test_labels_pred)

print('\nPredicted Labels for Test Images: ',test_labels_pred)
print('\nAccuracy of Classifier on Test Images: ',acc)
print('\nConfusion Matrix for Test Data: \n',conf_mat_test)
```

	0	1	2	3	4	5	6	7	8	9
0	972	0	3	0	0	1	1	1	2	0
1	0	1123	3	3	0	2	2	0	1	1
2	6	0	999	7	2	0	3	8	7	0
3	0	0	8	975	0	5	0	10	9	3
4	0	0	1	0	951	0	6	0	3	21
5	3	1	1	11	4	859	5	3	4	1
6	7	3	1	0	7	3	936	0	1	0
7	2	2	20	1	1	0	0	987	2	13
8	6	0	5	6	5	5	7	6	924	10
9	6	5	2	12	12	2	1	4	7	958

We carry out the same method for the test images to see if the classification was correct and accurate. We can see from the confusion matrix that it's not prefect with the value of 7 miscalculating the predicated number of 2 "20" times



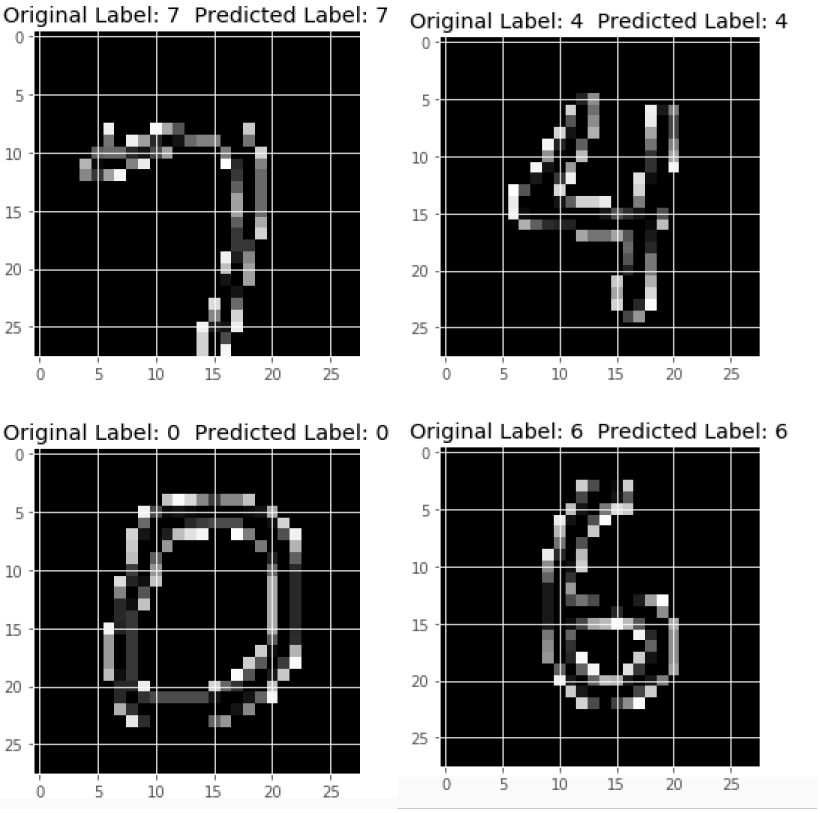
Again the matplotlib was used to construct a confusion matric test diagram for a more professional look.

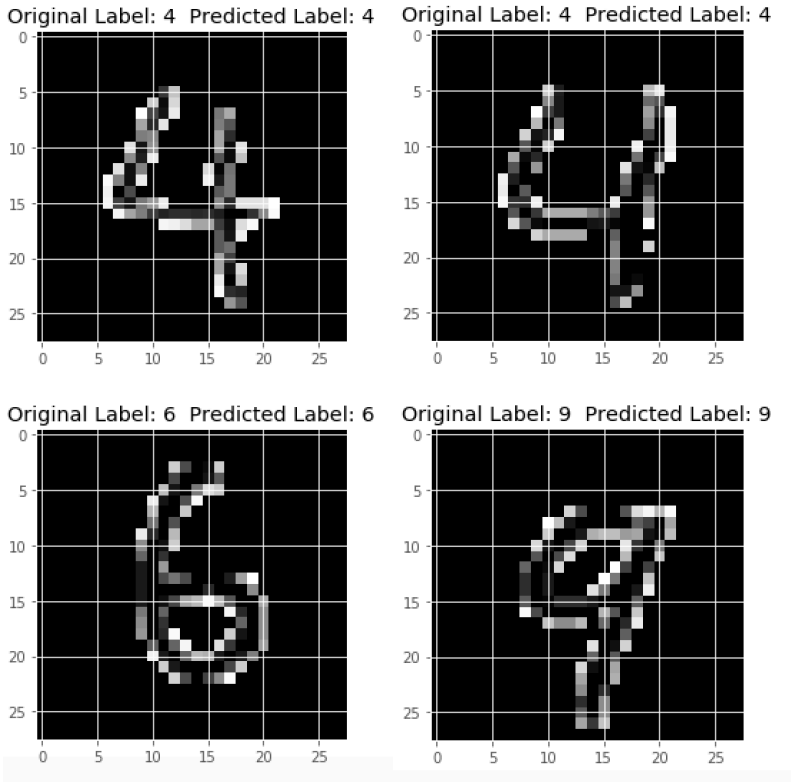
```
print('\nAccuracy of Classifier on Test Images: ',acc)
```

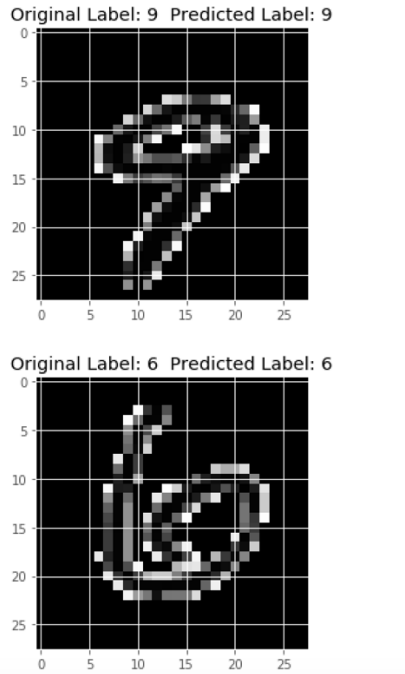
We have also calculated the classifier for the test images with the result being: 0.9684. this is close to the validation set of 0.9701666666666666 proving a positive classification.

```
# Show the Test Images with Original and Predicted Labels
a = np.random.randint(1,30,10)
for i in a:
    two_d = (np.reshape(test_img[i], (28, 28)) * 255).astype(np.uint8)
    plt.title('Original Label: {0}  Predicted Label: {1}'.format(test_labels[i],test_labels_pred[i]))
    plt.imshow(two_d, interpolation='nearest',cmap='gray')
    plt.show()
```

To display examples of the digits and show examples of classification set.


Original Label: 7 Predicted Label: 7
Original Label: 4 Predicted Label: 4
Original Label: 0 Predicted Label: 0
Original Label: 6 Predicted Label: 6


Original Label: 4 Predicted Label: 4
Original Label: 4 Predicted Label: 4
Original Label: 6 Predicted Label: 6
Original Label: 9 Predicted Label: 9



As you can see the examples above prove the classification has been successful in predicated. the numbers with the original label and the predicted label on the side

The algorithm essentially works by the test data being loaded. This is the MNIST data. Once the data has been loaded up the data will get labelled correctly in the categories of testing images and training with the use of cross validation used to divide the training data in order to train the classifier. This is an important step as the MNIST data is quite large. Then the classifier will start the training using the algorithm we have chosen for this which is the Random Forrest. As random forest works by using decision trees the algorithm needs to find the best number of splits for efficiency. Once the value that has used the random forest classification to be recognised has been matched with the training label it will then try to get a accuracy from the trained classifier with the same trained classifier being used again on the testing data. Once the results have been calculated we then have a confusion matrix printed, accuracy calculated and images printed out to show the findings.

Appendix

```
1. # Random Forest Classifier
2.
3. import sys
4. import numpy as np
5. import pickle
6. from sklearn import model_selection
7. from sklearn.ensemble import RandomForestClassifier
8. from sklearn.metrics import accuracy_score, confusion_matrix
```

```
9. from MNIST_Dataset_Loader.mnist_loader import MNIST
10.  import matplotlib.pyplot as plt
11.  from matplotlib import style
12.  style.use('ggplot')
13.
14.
15.
16.  old_stdout = sys.stdout
17.  log_file = open("summary.log","w")
18.  sys.stdout = log_file
19.
20.
21.  print('\nLoading MNIST Data...')
22.  # data = MNIST('./python-mnist/data/')
23.
24.  data = MNIST('./MNIST_Dataset_Loader/dataset/')
25.
26.  print('\nLoading Training Data...')
27.  img_train, labels_train = data.load_training()
28.  train_img = np.array(img_train)
29.  train_labels = np.array(labels_train)
30.
31.  print('\nLoading Testing Data...')
32.  img_test, labels_test = data.load_testing()
33.  test_img = np.array(img_test)
34.  test_labels = np.array(labels_test)
35.
36.
37.  #Features
38.  X = train_img
39.
40.  #Labels
41.  y = train_labels
42.
43.  print('\nPreparing Classifier Training and Validation Data...')
44.  X_train, X_test, y_train, y_test =
   model_selection.train_test_split(X,y,test_size=0.1)
45.
46.
47.  print('\nRandom Forest Classifier with n_estimators = 100, n_jobs =
   10')
48.  print('\nPickling the Classifier for Future Use...')
49.  clf = RandomForestClassifier(n_estimators=100, n_jobs=10)
50.  clf.fit(X_train,y_train)
51.
52.  with open('MNIST_RFC.pickle','wb') as f:
53.          pickle.dump(clf, f)
54.
55.  pickle_in = open('MNIST_RFC.pickle','rb')
56.  clf = pickle.load(pickle_in)
57.
58.  print('\nCalculating Accuracy of trained Classifier...')
59.  confidence = clf.score(X_test,y_test)
60.
61.  print('\nMaking Predictions on Validation Data...')
62.  y_pred = clf.predict(X_test)
63.
64.  print('\nCalculating Accuracy of Predictions...')
65.  accuracy = accuracy_score(y_test, y_pred)
66.
67.  print('\nCreating Confusion Matrix...')
```

```
68.  conf_mat = confusion_matrix(y_test,y_pred)
69.
70.  print('\nRFC Trained Classifier Confidence: ',confidence)
71.  print('\nPredicted Values: ',y_pred)
72.  print('\nAccuracy of Classifier on Validation Image Data:
   ',accuracy)
73.  print('\nConfusion Matrix: \n',conf_mat)
74.
75.
76.  # Plot Confusion Matrix Data as a Matrix
77.  plt.matshow(conf_mat)
78.  plt.title('Confusion Matrix for Validation Data')
79.  plt.colorbar()
80.  plt.ylabel('True label')
81.  plt.xlabel('Predicted label')
82.  plt.show()
83.
84.
85.  print('\nMaking Predictions on Test Input Images...')
86.  test_labels_pred = clf.predict(test_img)
87.
88.  print('\nCalculating Accuracy of Trained Classifier on Test Data...
   ')
89.  acc = accuracy_score(test_labels,test_labels_pred)
90.
91.  print('\n Creating Confusion Matrix for Test Data...')
92.  conf_mat_test = confusion_matrix(test_labels,test_labels_pred)
93.
94.  print('\nPredicted Labels for Test Images: ',test_labels_pred)
95.  print('\nAccuracy of Classifier on Test Images: ',acc)
96.  print('\nConfusion Matrix for Test Data: \n',conf_mat_test)
97.
98.  # Plot Confusion Matrix for Test Data
99.  plt.matshow(conf_mat_test)
100. plt.title('Confusion Matrix for Test Data')
101. plt.colorbar()
102. plt.ylabel('True label')
103. plt.xlabel('Predicted label')
104. plt.axis('off')
105. plt.show()
106.
107. sys.stdout = old_stdout
108. log_file.close()
109.
110.
111. # Show the Test Images with Original and Predicted Labels
112. a = np.random.randint(1,30,10)
113. for i in a:
114.         two_d = (np.reshape(test_img[i], (28, 28)) *
   255).astype(np.uint8)
115.         plt.title('Original Label: {0}  Predicted Label:
   {1}'.format(test_labels[i],test_labels_pred[i]))
116.         plt.imshow(two_d, interpolation='nearest',cmap='gray')
117.         plt.show()
```