# Report On : Project - 3 Classification

**Neeraj Ajit Abhyankar** **UB Person No. : 50290958** **UBID : nabyank**

**Objective :**

This project focuses on the task of classification of different methods.

The aim of this project is to implement and evaluate classification algorithms to classify hand written images of digits into 0,1, 2,…. 9 by training with the MNIST & USPS dataset.

**Task :**

To implement different methods for classification :

1. Logistic regression using backpropagtion.
2. Multilayer Perceptron Neural Network.
3. Random Forest package.
4. SVM package.

**Overview of Data and Pre-Processing :**

**For MNIST :**

1) Each image in the dataset is handwritten grayscale image of size 28x28 with source http://yann.lecun.com/exdb/mnist/

2) The Preprocessing of the MNIST data includes opening the file using qzip, and using pickle serialize the MNIST dataset into Training, Validation and Target sets.

**For USPS :**

1) Images in USPS data are of different sizes and the pixels are represented by combination of RGB colours.
2) The preprocessing of the USPS data includes *resizing of images into 28x28* and conversion of each image into grayscale was done before USPS data was used for testing.

**Algorithms for Different Classifier Methods :**

**Neural Network :**

**1) Import Required Libraries**

**2) Take the pre-processed data for Training, Testing and Validation**

**3) Add dense layer-1 for neural network model with sigmoid function for activation.**

**4) Add dense layer-2 for neural network model with softmax function for activation.**

**5) Compile all the models and Print Accuracy of each data.**

**SVM :**

**1) Take the pre-processed data for Training, Testing and Validation.**

**2) Give the required values for kernel, C, gamma and max iterations to the classifier model.**

**3) Run the SVM Model and get the accuracy using score function.**

**4) Generate the Confusion matrix for each Training, Testing and Validation and USPS Data.**

**Random Forest :**

**1) Take the pre-processed data for Training, Testing and Validation.**

**2) Give the required values for n-estimators to the classifier model.**

**3) Run the RF Model and get the accuracy using score function.**

**4) Generate the Confusion matrix for each Training, Testing and Validation and USPS Data.**

**Logistic Regression :**

**1) Take the pre-processed data for Training, Testing and Validation.**

**2) define the required functions for logistic regression such as softmax, Loss_Function, Predictions, onehot vector converter, accuracy.**

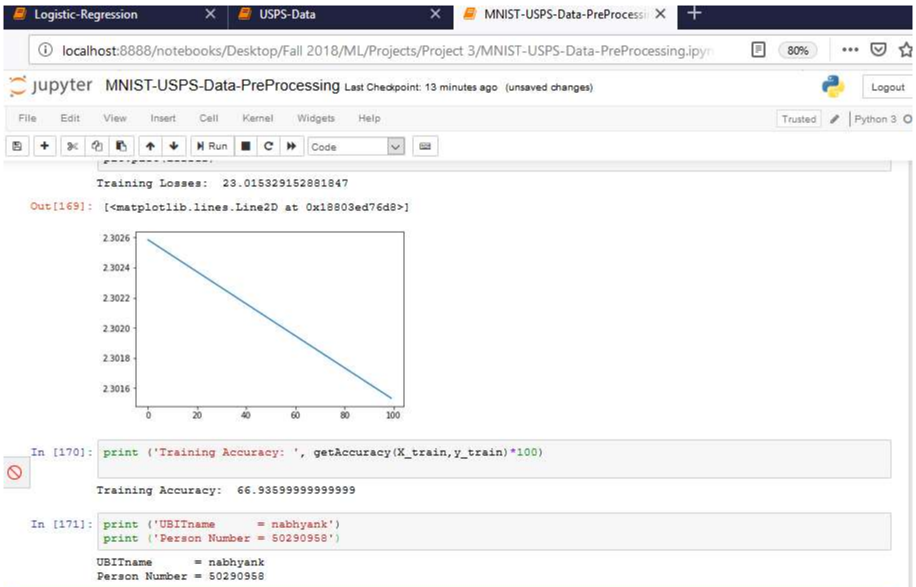
**3) Run the regression function with the required parameter values of lambda, max iterations, learning rate.**

**4) Plot the Loss and Print the Accuracy of the Model.**

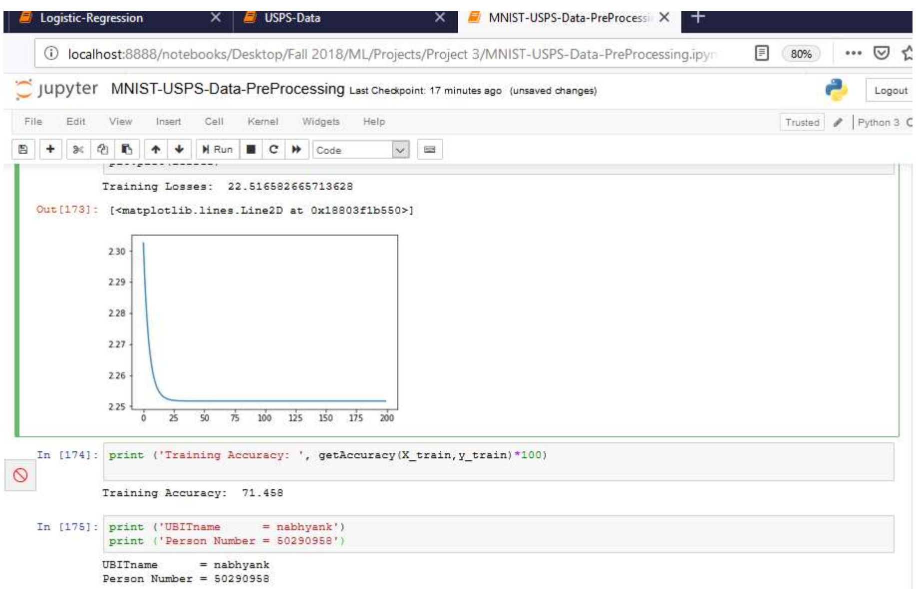
**Output For Logistic Regression :**

**For MNIST Data :**

**Parameter Set 1 : Lambda = 1 ; Max Iterations = 100 ; Learning Rate = 1e-5**

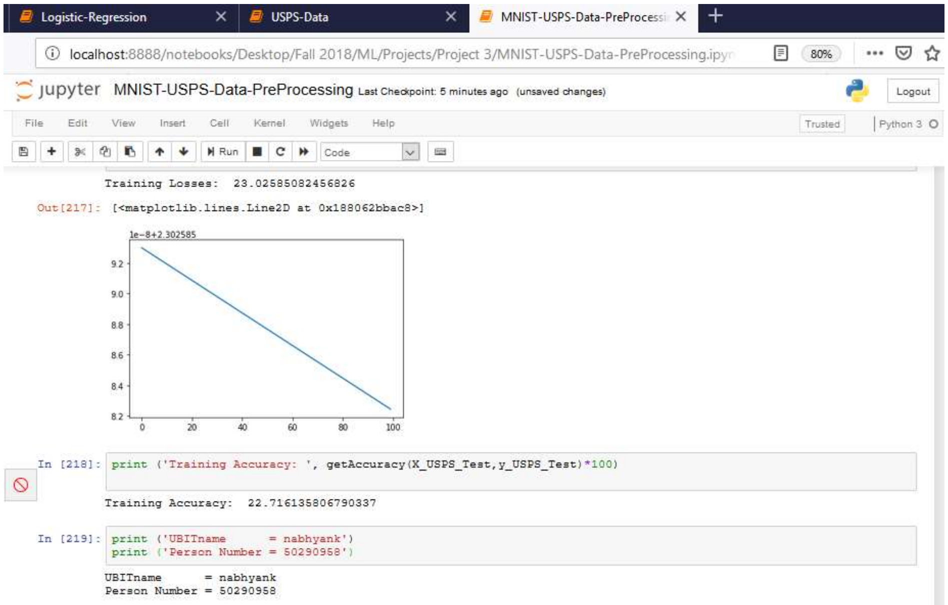



**Parameter Set 2 : Lambda = 10 ; Max Iterations = 200 ; Learning Rate = 1e-2**

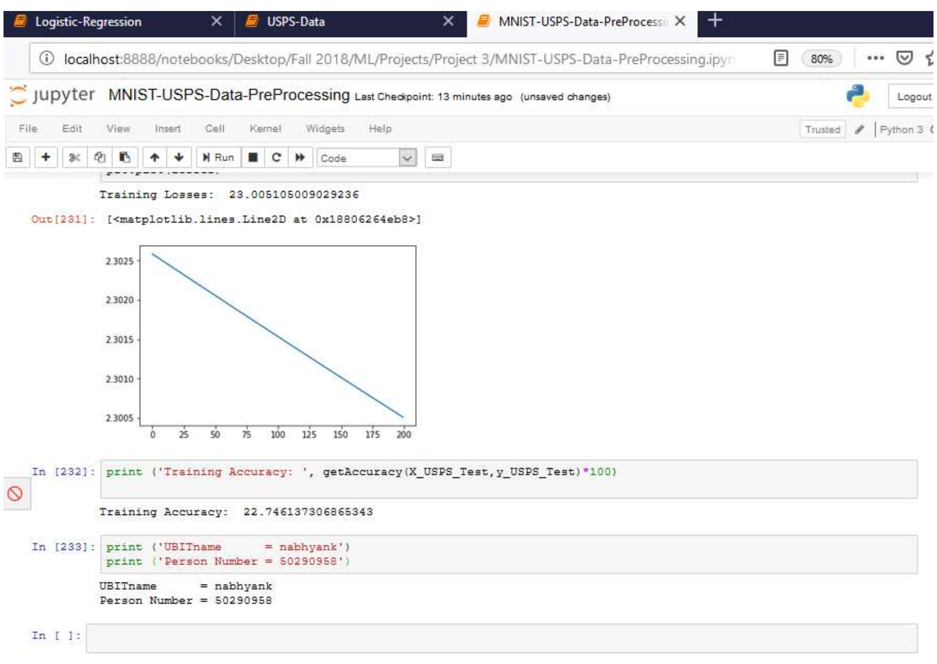



**For USPS Data :**

**Parameter Set 1 : Lambda = 5 ; Max Iterations = 100 ; Learning Rate = 1e-10**




**Parameter Set 2 : Lambda = 10 ; Max Iterations = 200 ; Learning Rate = 1e-5**

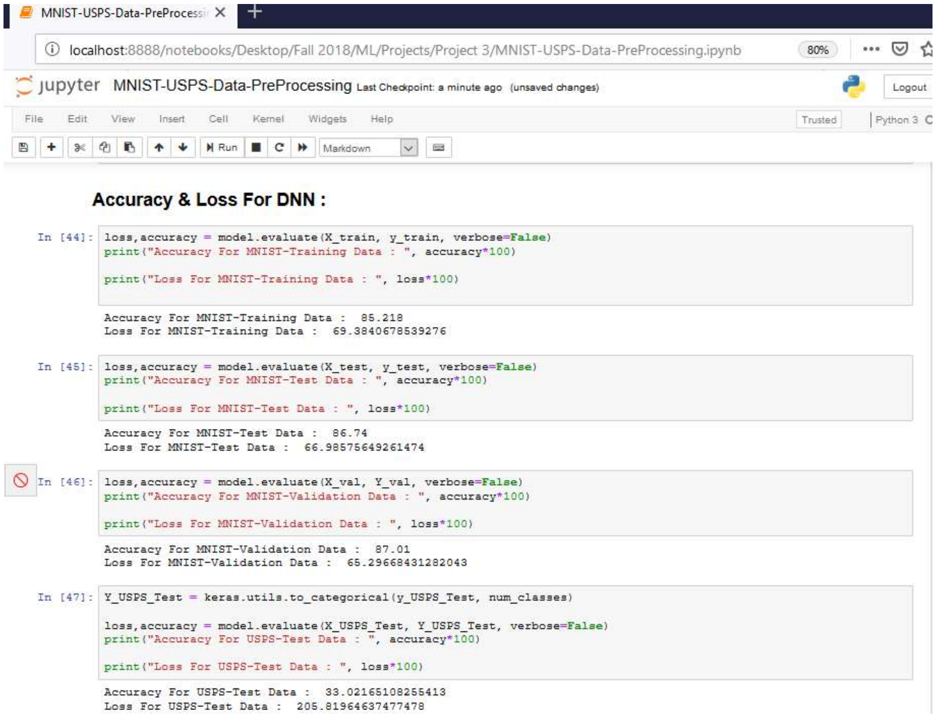



| Parameter Set | MNIST Data Accuracy | USPS Data Accuracy |
|---|---|---|
| 1 | 66.9 | 22.71 |
| 2 | 71.45 | 22.74 |

**Output For DNN:**

**For MNIST Data :**

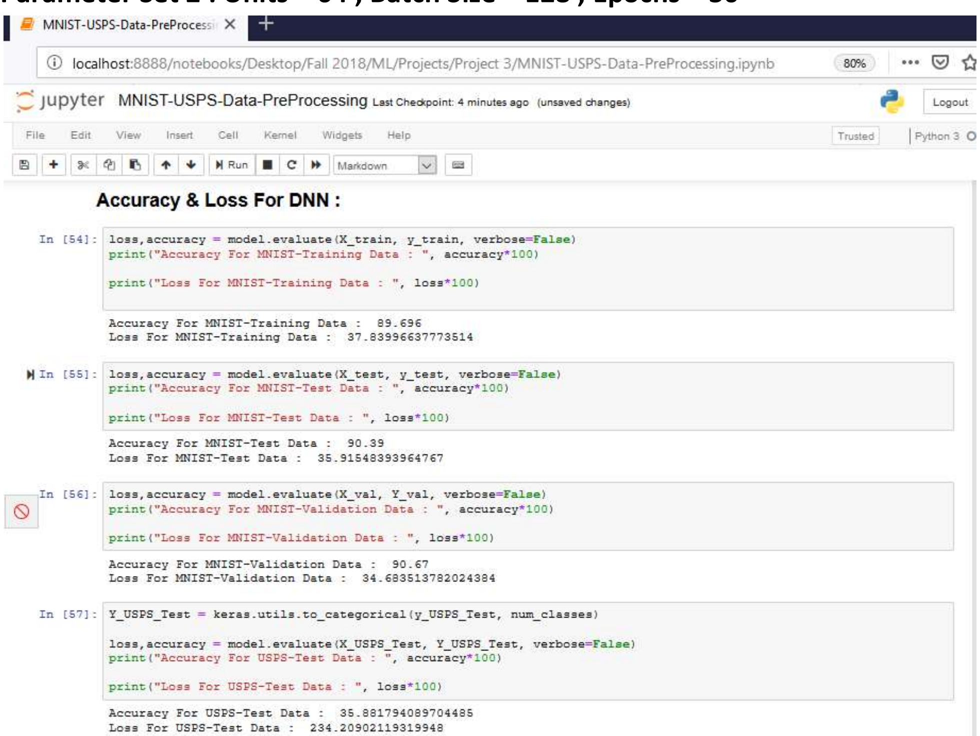
**Parameter Set 1 : Units – 32 ; Batch Size – 128 ; Epochs – 10**



Accuracy & Loss For DNN :

```
In [44]: loss,accuracy = model.evaluate(X_train, y_train, verbose=False)
         print("Accuracy For MNIST-Training Data : ", accuracy*100)

         print("Loss For MNIST-Training Data : ", loss*100)

Accuracy For MNIST-Training Data :  85.218
Loss For MNIST-Training Data :  69.3840678539276

In [45]: loss,accuracy = model.evaluate(X_test, y_test, verbose=False)
         print("Accuracy For MNIST-Test Data : ", accuracy*100)

         print("Loss For MNIST-Test Data : ", loss*100)

Accuracy For MNIST-Test Data :  86.74
Loss For MNIST-Test Data :  66.98575649261474

In [46]: loss,accuracy = model.evaluate(X_val, Y_val, verbose=False)
         print("Accuracy For MNIST-Validation Data : ", accuracy*100)

         print("Loss For MNIST-Validation Data : ", loss*100)

Accuracy For MNIST-Validation Data :  87.01
Loss For MNIST-Validation Data :  65.29668431282043

In [47]: Y_USPS_Test = keras.utils.to_categorical(y_USPS_Test, num_classes)

         loss,accuracy = model.evaluate(X_USPS_Test, Y_USPS_Test, verbose=False)
         print("Accuracy For USPS-Test Data : ", accuracy*100)

         print("Loss For USPS-Test Data : ", loss*100)

Accuracy For USPS-Test Data :  33.02165108255413
Loss For USPS-Test Data :  205.81964637477478
```

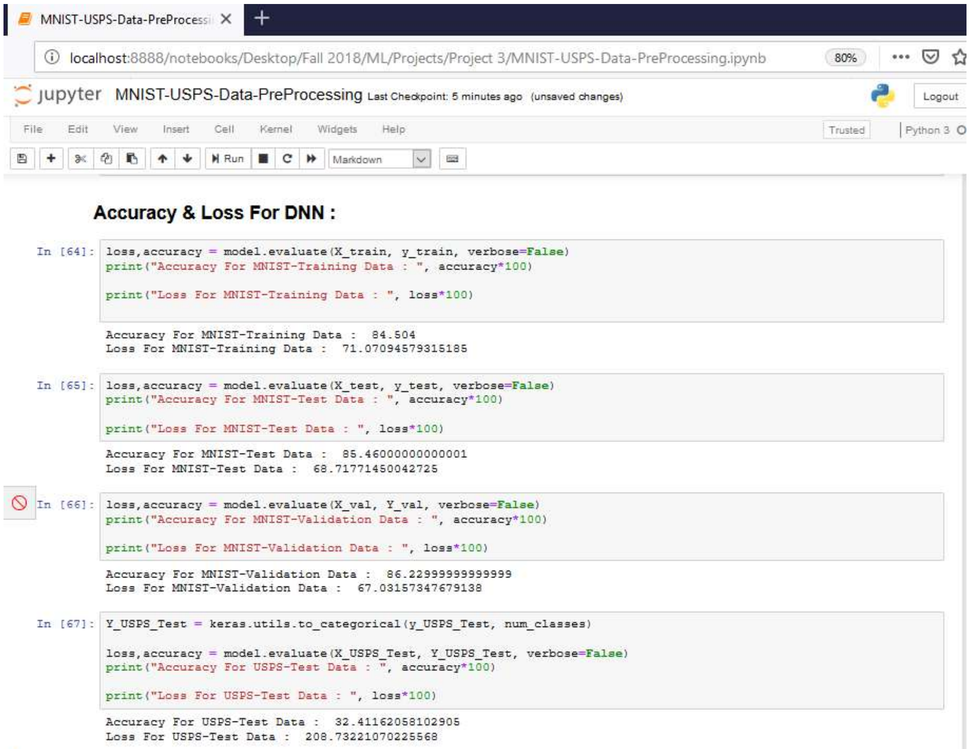
**Parameter Set 2 : Units – 64 ; Batch Size – 128 ; Epochs – 30**



Accuracy & Loss For DNN :

```
In [54]: loss,accuracy = model.evaluate(X_train, y_train, verbose=False)
         print("Accuracy For MNIST-Training Data : ", accuracy*100)

         print("Loss For MNIST-Training Data : ", loss*100)

Accuracy For MNIST-Training Data :  89.696
Loss For MNIST-Training Data :  37.83996637773514

In [55]: loss,accuracy = model.evaluate(X_test, y_test, verbose=False)
         print("Accuracy For MNIST-Test Data : ", accuracy*100)

         print("Loss For MNIST-Test Data : ", loss*100)

Accuracy For MNIST-Test Data :  90.39
Loss For MNIST-Test Data :  35.91548393964767

In [56]: loss,accuracy = model.evaluate(X_val, Y_val, verbose=False)
         print("Accuracy For MNIST-Validation Data : ", accuracy*100)

         print("Loss For MNIST-Validation Data : ", loss*100)

Accuracy For MNIST-Validation Data :  90.67
Loss For MNIST-Validation Data :  34.683513782024384

In [57]: Y_USPS_Test = keras.utils.to_categorical(y_USPS_Test, num_classes)

         loss,accuracy = model.evaluate(X_USPS_Test, Y_USPS_Test, verbose=False)
         print("Accuracy For USPS-Test Data : ", accuracy*100)

         print("Loss For USPS-Test Data : ", loss*100)

Accuracy For USPS-Test Data :  35.881794089704485
Loss For USPS-Test Data :  234.20902119319948
```

**Parameter Set 3 : Units – 32 ; Batch Size – 256 ; Epochs – 20**

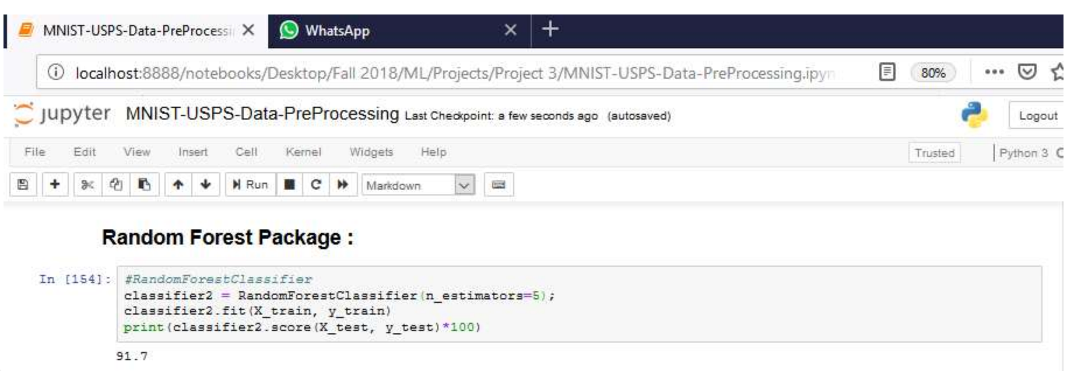



```
In [64]: loss,accuracy = model.evaluate(X_train, y_train, verbose=False)
         print("Accuracy For MNIST-Training Data : ", accuracy*100)

         print("Loss For MNIST-Training Data : ", loss*100)

Accuracy For MNIST-Training Data :  84.504
Loss For MNIST-Training Data :  71.07094579315185

In [65]: loss,accuracy = model.evaluate(X_test, y_test, verbose=False)
         print("Accuracy For MNIST-Test Data : ", accuracy*100)

         print("Loss For MNIST-Test Data : ", loss*100)

Accuracy For MNIST-Test Data :  85.46000000000001
Loss For MNIST-Test Data :  68.71771450042725

In [66]: loss,accuracy = model.evaluate(X_val, Y_val, verbose=False)
         print("Accuracy For MNIST-Validation Data : ", accuracy*100)

         print("Loss For MNIST-Validation Data : ", loss*100)

Accuracy For MNIST-Validation Data :  86.22999999999999
Loss For MNIST-Validation Data :  67.03157347679138

In [67]: Y_USPS_Test = keras.utils.to_categorical(y_USPS_Test, num_classes)

         loss,accuracy = model.evaluate(X_USPS_Test, Y_USPS_Test, verbose=False)
         print("Accuracy For USPS-Test Data : ", accuracy*100)

         print("Loss For USPS-Test Data : ", loss*100)

Accuracy For USPS-Test Data :  32.41162058102905
Loss For USPS-Test Data :  208.73221070225568
```

| Parameter Set | Training Accuracy | Testing Accuracy | Validation Accuracy | USPS Accuracy |
|---|---|---|---|---|
| 1 | 85.21 | 86.74 | 87.01 | 33.02 |
| 2 | 89.69 | 90.39 | 90.67 | 35.88 |
| 3 | 85.50 | 85.46 | 86.22 | 32.41 |

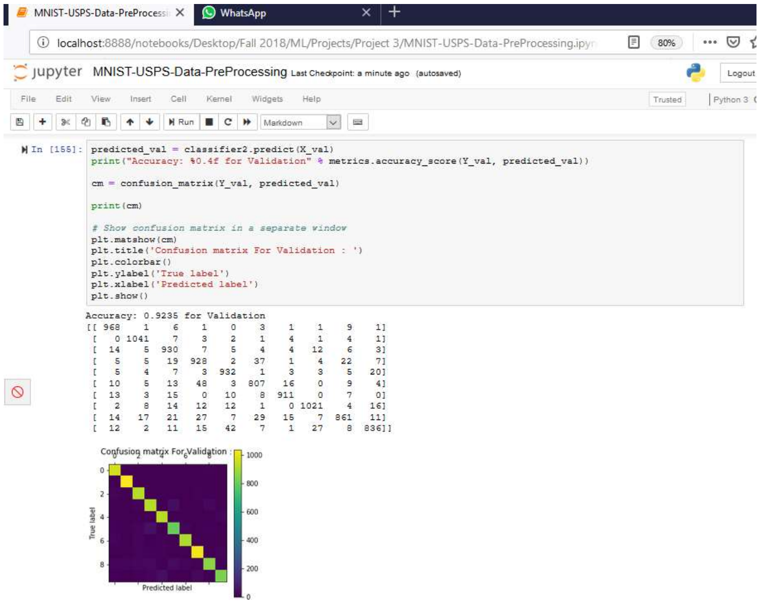
**Output For Random Forest and Confusion Matrix for MNIST & USPS Data :**

**Total Accuracy for Random Forest with n_estimators = 5 : 91.7**




```
In [154]: #RandomForestClassifier
          classifier2 = RandomForestClassifier(n_estimators=5);
          classifier2.fit(X_train, y_train)
          print(classifier2.score(X_test, y_test)*100)

91.7
```

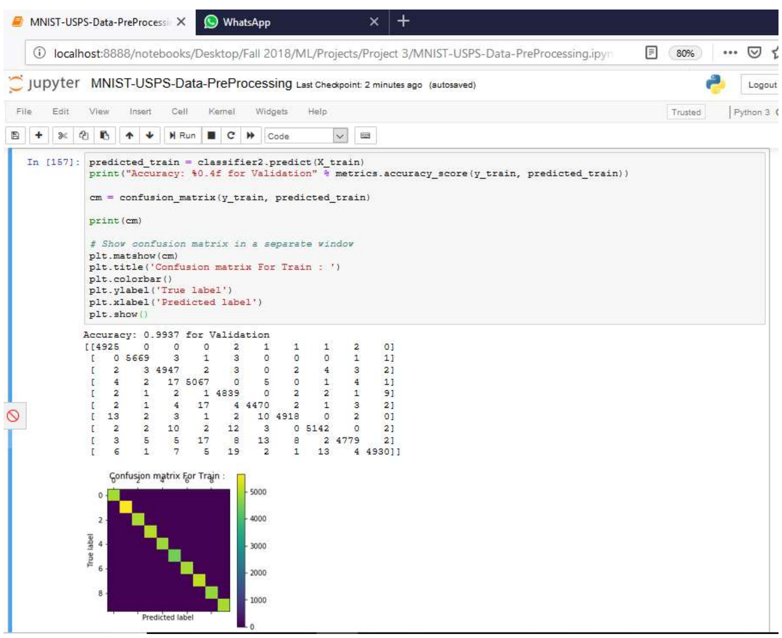
## Confusion Matrix for Validation MNIST Data :



```
predicted_val = classifier2.predict(X_val)
print("Accuracy: %0.4f for Validation" % metrics.accuracy_score(Y_val, predicted_val))

cm = confusion_matrix(Y_val, predicted_val)

print(cm)

# Show confusion matrix in a separate window
plt.matshow(cm)
plt.title('Confusion matrix For Validation : ')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Accuracy: 0.9235 for Validation
[[ 968    1    6    1    0    3    1    1    9    1]
 [   0 1041    7    3    2    1    4    1    4    1]
 [  14    5  930    7    5    4    4   12    6    3]
 [   5    5   19  928    2   37    1    4   22    7]
 [   5    4    7    3  932    1    3    3    5   20]
 [  10    5   13   48    3  807   16    0    9    4]
 [  13    3   15    0   10    8  911    0    7    0]
 [   2    8   14   12   12    1    0 1021    4   16]
 [  14   17   21   27    7   29   15    7  861   11]
 [  12    2   11   15   42    7    1   27    8  836]]
```

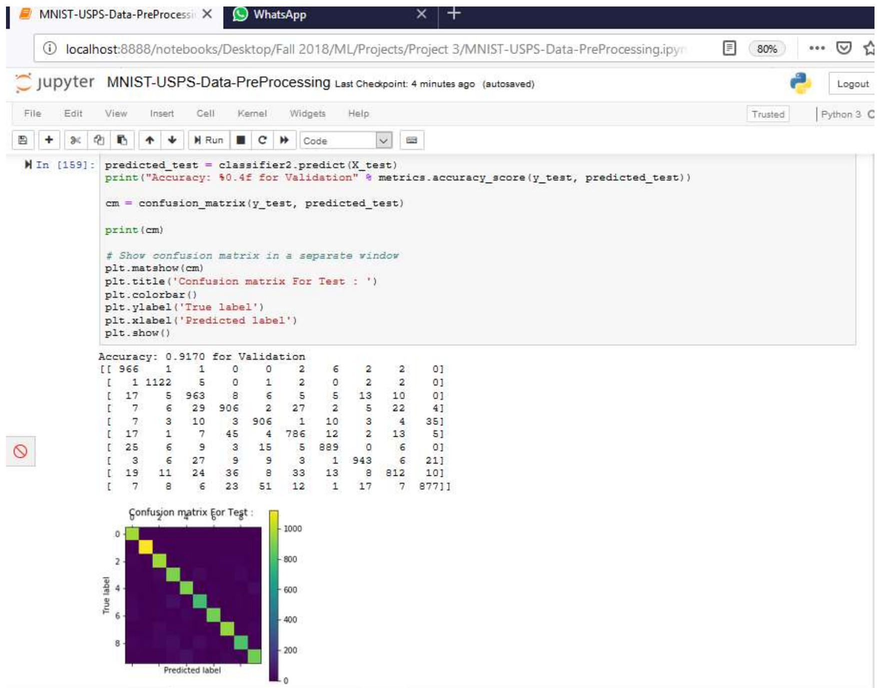
## Confusion Matrix for Training MNIST Data :



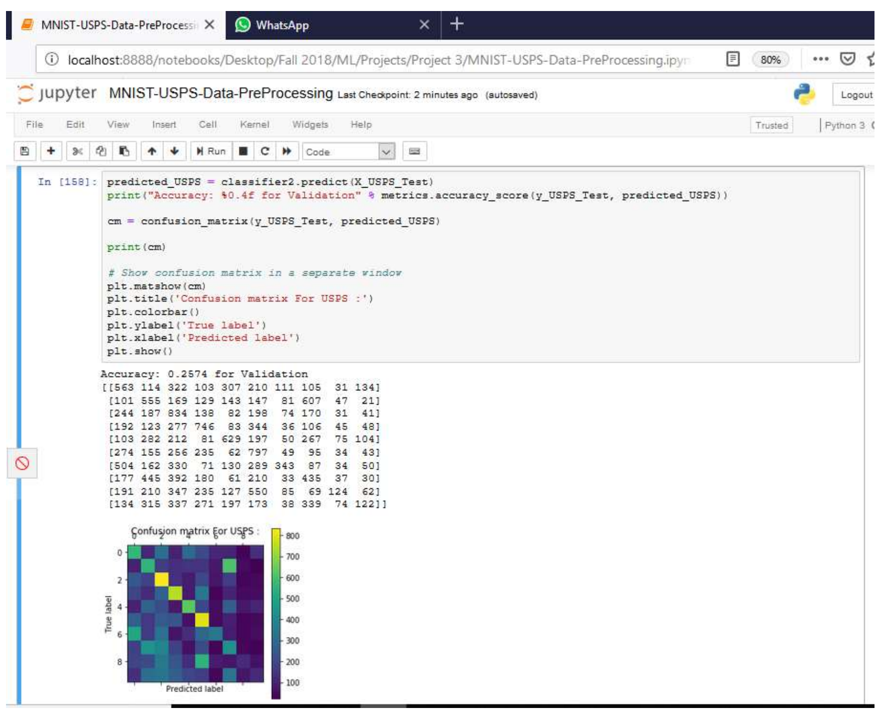
```
predicted_train = classifier2.predict(X_train)
print("Accuracy: %0.4f for Validation" % metrics.accuracy_score(y_train, predicted_train))

cm = confusion_matrix(y_train, predicted_train)

print(cm)

# Show confusion matrix in a separate window
plt.matshow(cm)
plt.title('Confusion matrix For Train : ')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Accuracy: 0.9937 for Validation
[[4925    0    0    0    2    1    1    1    2    0]
 [   0 5669    3    1    3    0    0    0    1    1]
 [   2    3 4947    2    3    0    2    4    3    2]
 [   4    2   17 5067    0    5    0    1    4    1]
 [   2    1    2    1 4839    0    2    2    1    9]
 [   2    1    4   17    4 4470    2    1    3    2]
 [  13    2    3    1    2   10 4918    0    2    0]
 [   2    2   10    2   12    3    0 5142    0    2]
 [   3    5    5   17    8   13    8    2 4779    2]
 [   6    1    7    5   19    2    1   13    4 4930]]
```

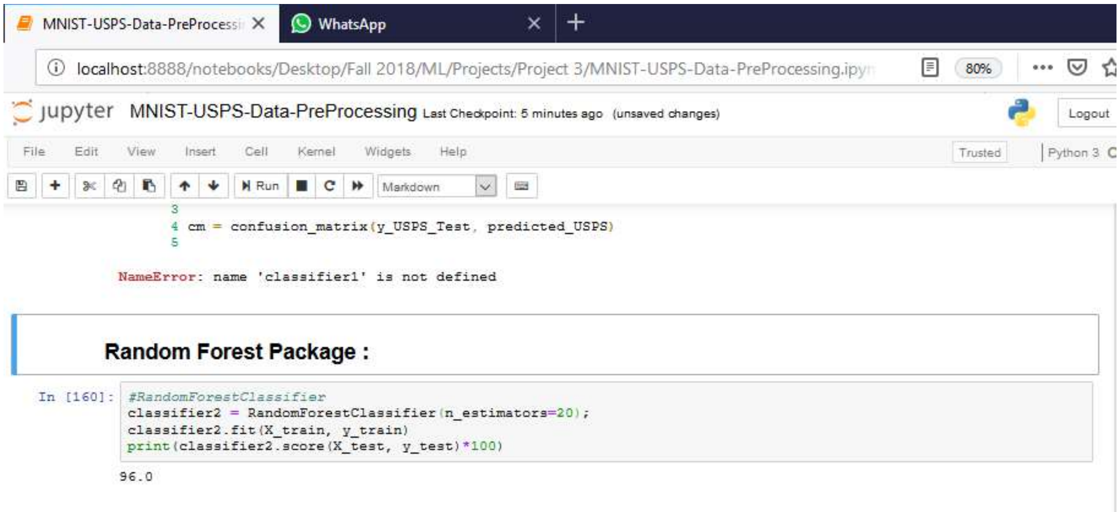
## Confusion Matrix for Testing MNIST Data :



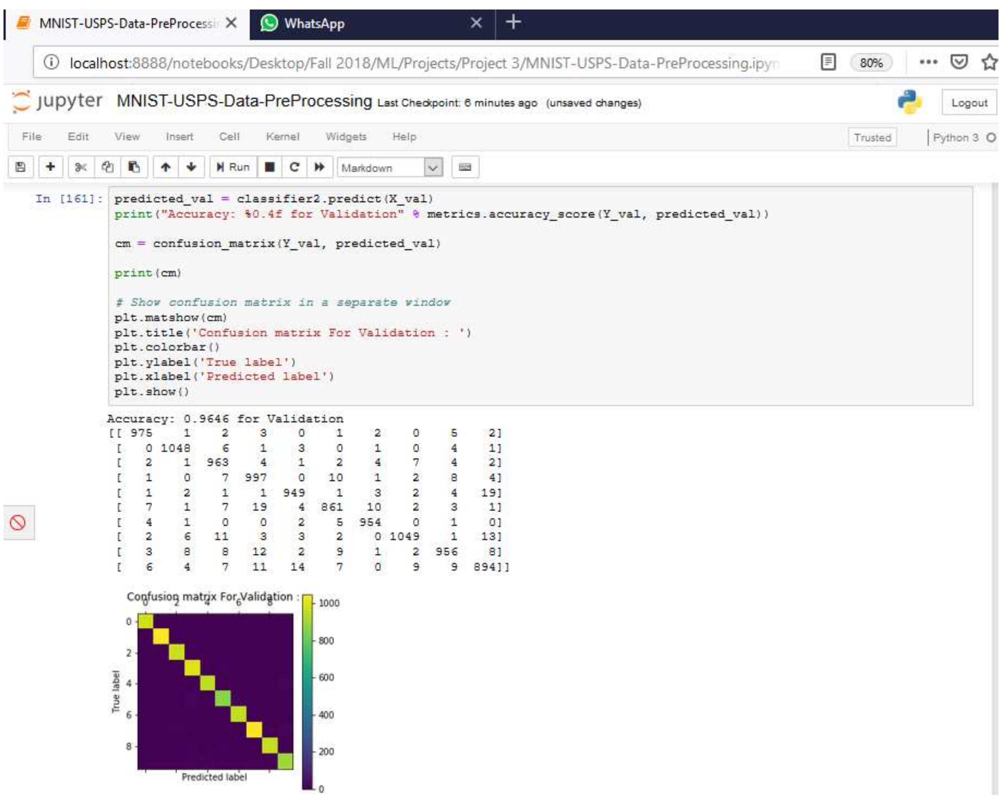

## Confusion Matrix for Testing USPS Data :



**Total Accuracy for Random Forest with n_estimators = 10 : 96.0**




**Confusion Matrix for Validation MNIST Data :**



## Confusion Matrix for Training MNIST Data :

MNIST-USPS-Data-PreProcessing | WhatsApp

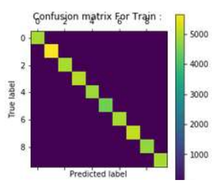
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 3/MNIST-USPS-Data-PreProcessing.ipynb

jupyter MNIST-USPS-Data-PreProcessing Last Checkpoint: 8 minutes ago (autosaved)

In [157]:

```
predicted_train = classifier2.predict(X_train)
print("Accuracy: %0.4f for Validation" % metrics.accuracy_score(y_train, predicted_train))

cm = confusion_matrix(y_train, predicted_train)

print(cm)

# Show confusion matrix in a separate window
plt.matshow(cm)
plt.title('Confusion matrix For Train : ')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Accuracy: 0.9937 for Validation
[[4925    0    0    0    2    1    1    1    2    0]
 [   0 5669    3    1    3    0    0    0    1    1]
 [   2    3 4947    2    3    0    2    4    3    2]
 [   4    2   17 5067    0    5    0    1    4    1]
 [   2    1    2    1 4839    0    2    2    1    9]
 [   2    1    4   17    4 4470    2    1    3    2]
 [  13    2    3    1    2   10 4918    0    2    0]
 [   2    2   10    2   12    3    0 5142    0    2]
 [   3    5    5   17    8   13    8    2 4779    2]
 [   6    1    7    5   19    2    1   13    4 4930]]
```




## Confusion Matrix for Testing MNIST Data :

MNIST-USPS-Data-PreProcessing | WhatsApp

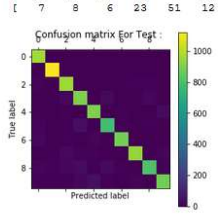
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 3/MNIST-USPS-Data-PreProcessing.ipynb

jupyter MNIST-USPS-Data-PreProcessing Last Checkpoint: 6 minutes ago (unsaved changes)

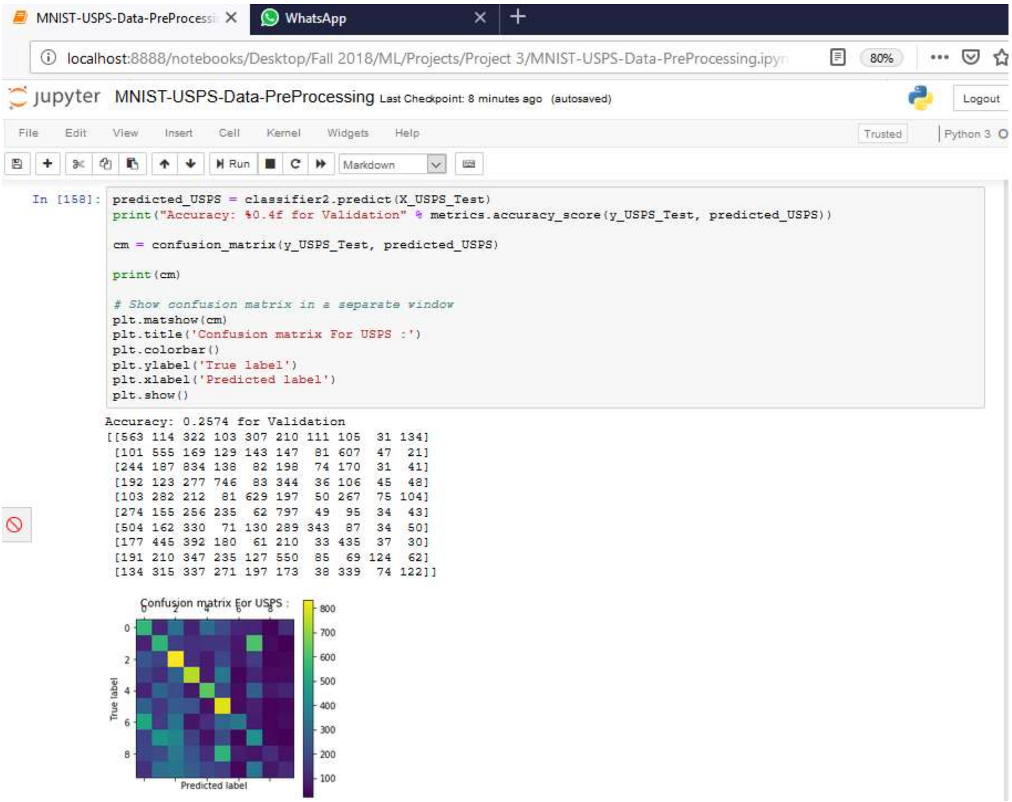
In [159]:

```
predicted_test = classifier2.predict(X_test)
print("Accuracy: %0.4f for Validation" % metrics.accuracy_score(y_test, predicted_test))

cm = confusion_matrix(y_test, predicted_test)

print(cm)

# Show confusion matrix in a separate window
plt.matshow(cm)
plt.title('Confusion matrix For Test : ')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Accuracy: 0.9170 for Validation
[[ 966    1    1    0    0    2    6    2    2    0]
 [   1 1122    5    0    1    2    0    2    2    0]
 [  17    5  963    8    6    5    5   13   10    0]
 [   7    6   29  906    2   27    2    5   22    4]
 [   7    3   10    3  906    1   10    3    4   35]
 [  17    1    7   45    4  786   12    2   13    5]
 [  25    6    9    3   15    5  889    0    6    0]
 [   3    6   27    9    9    3    1  943    6   21]
 [  19   11   24   36    8   33   13    8  812   10]
 [   7    8    6   23   51   12    1   17    7  877]]
```



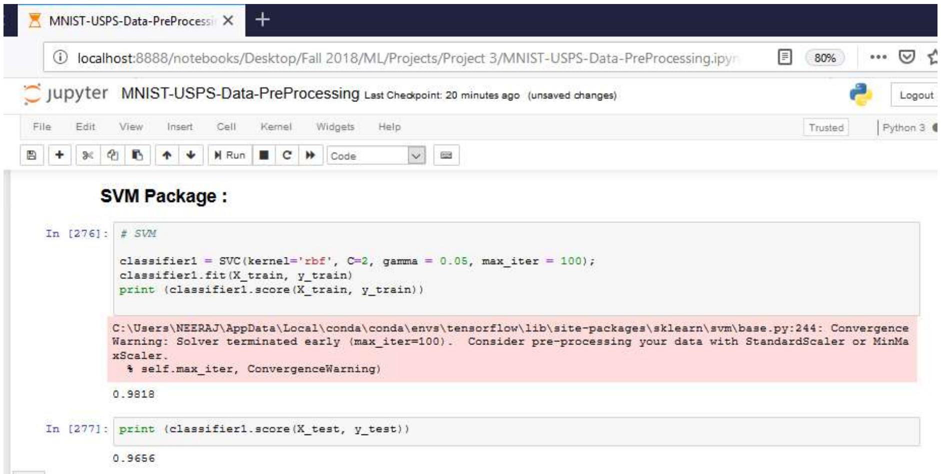
**Confusion Matrix for Testing USPS Data :**



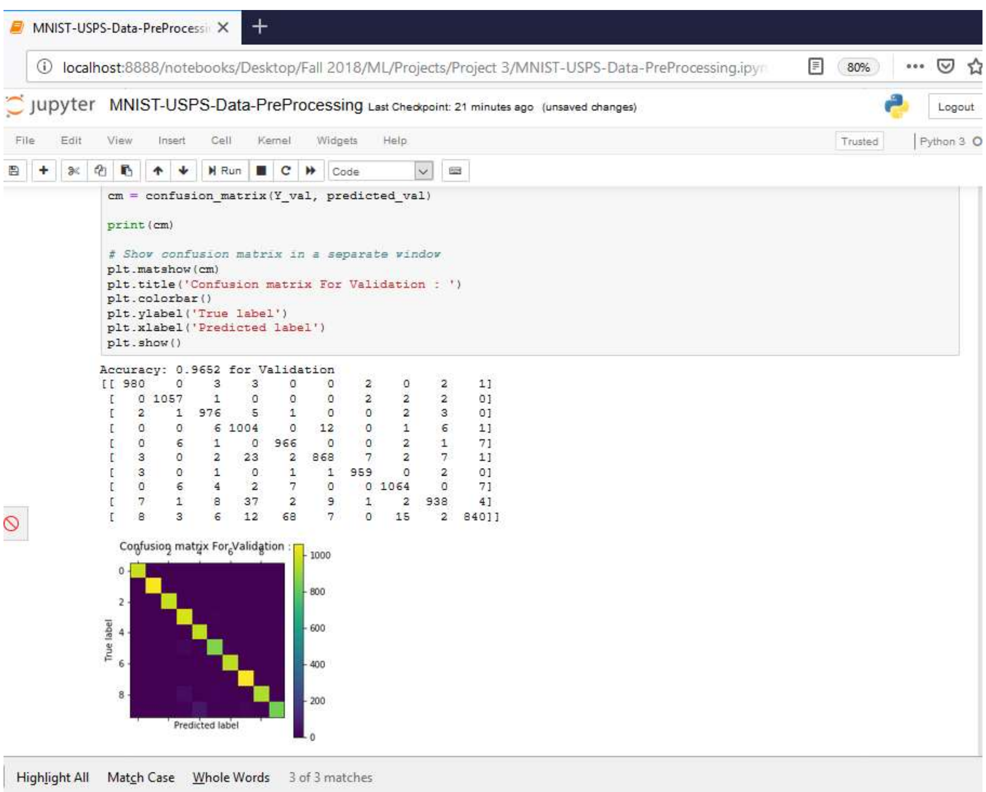

| Parameter Set | MNIST Validation Accuracy | MNIST Testing Accuracy | MNIST Training Accuracy | USPS Testing Accuracy |
|---|---|---|---|---|
| **1** | **92.35** | **99.37** | **91.70** | **25.74** |
| **2** | **96.46** | **99.37** | **91.70** | **25.74** |

**Output For SVM :**

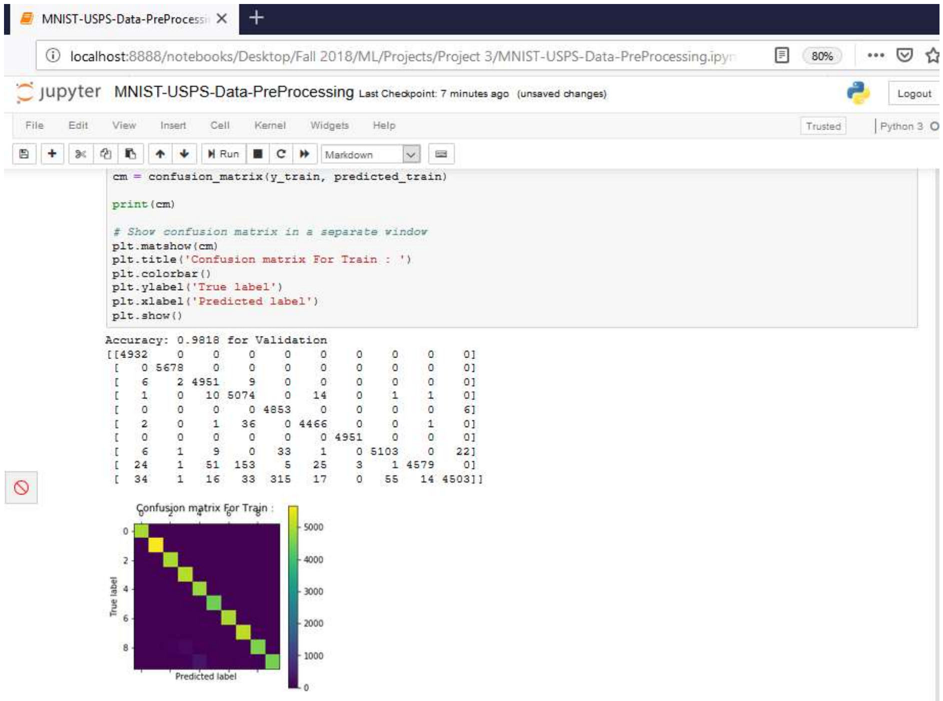
**Total Accuracy for Random Forest with C=2 ; gamma=0.05 ; max_iter = 100**




**Confusion Matrix for Validation MNIST Data :**



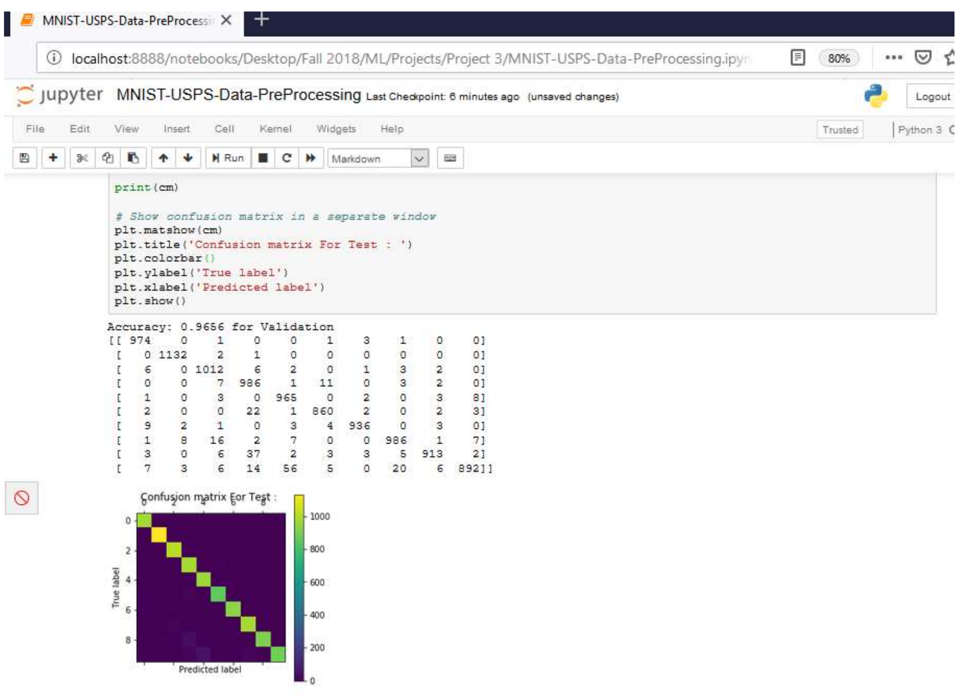
## Confusion Matrix for Training MNIST Data :




```
cm = confusion_matrix(y_train, predicted_train)

print(cm)

# Show confusion matrix in a separate window
plt.matshow(cm)
plt.title('Confusion matrix For Train : ')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Accuracy: 0.9818 for Validation
[[4932    0    0    0    0    0    0    0    0    0]
 [   0 5678    0    0    0    0    0    0    0    0]
 [   6    2 4951    9    0    0    0    0    0    0]
 [   1    0   10 5074    0   14    0    1    1    0]
 [   0    0    0    0 4853    0    0    0    0    6]
 [   2    0    1   36    0 4466    0    0    1    0]
 [   0    0    0    0    0    0 4951    0    0    0]
 [   6    1    9    0   33    1    0 5103    0   22]
 [  24    1   51  153    5   25    3    1 4579    0]
 [  34    1   16   33  315   17    0   55   14 4503]]
```

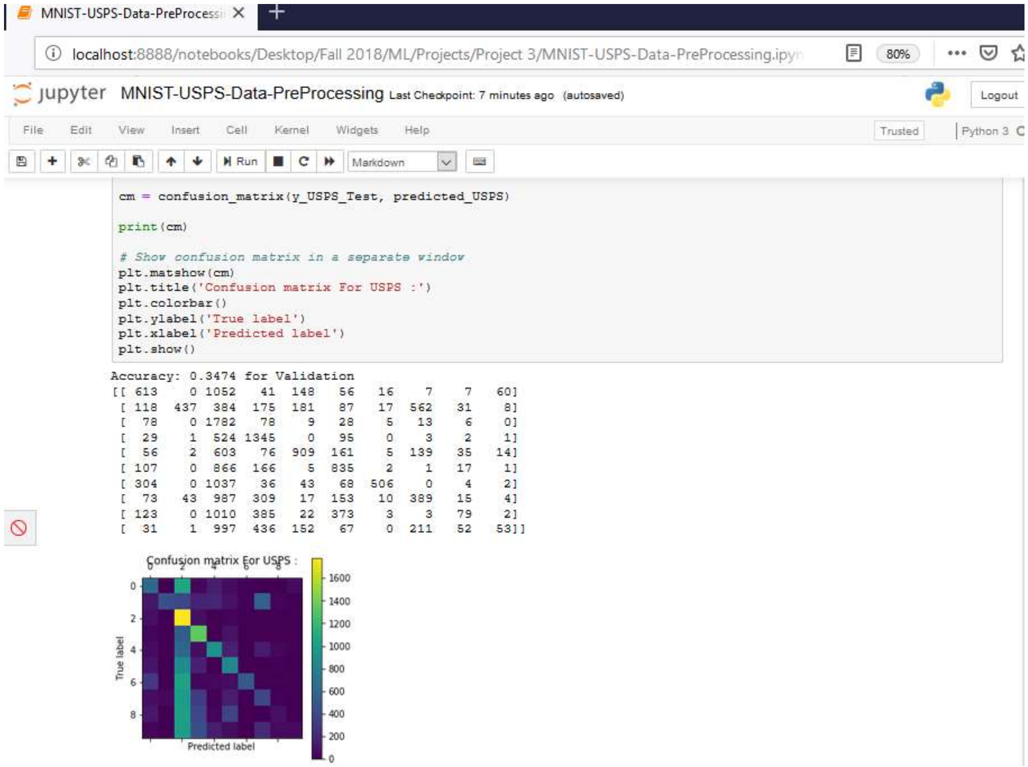
## Confusion Matrix for Testing MNIST Data :



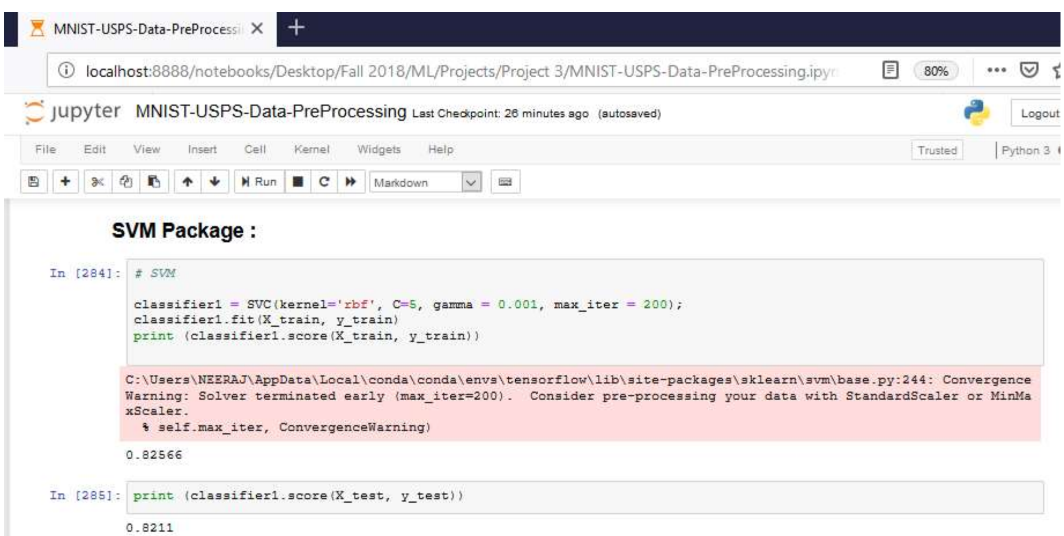

```
print(cm)

# Show confusion matrix in a separate window
plt.matshow(cm)
plt.title('Confusion matrix For Test : ')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Accuracy: 0.9656 for Validation
[[ 974    0    1    0    0    1    3    1    0    0]
 [   0 1132    2    1    0    0    0    0    0    0]
 [   6    0 1012    6    2    0    1    3    2    0]
 [   0    0    7  986    1   11    0    3    2    0]
 [   1    0    3    0  965    0    2    0    3    8]
 [   2    0    0   22    1  860    2    0    2    3]
 [   9    2    1    0    3    4  936    0    3    0]
 [   1    8   16    2    7    0    0  986    1    7]
 [   3    0    6   37    2    3    3    5  913    2]
 [   7    3    6   14   56    5    0   20    6  892]]
```

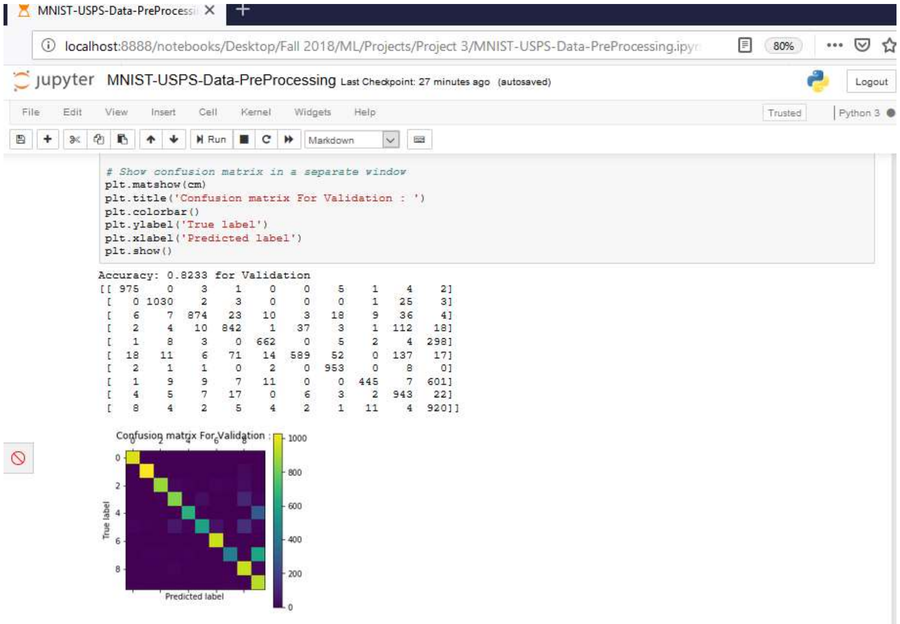
**Confusion Matrix for Testing USPS Data :**



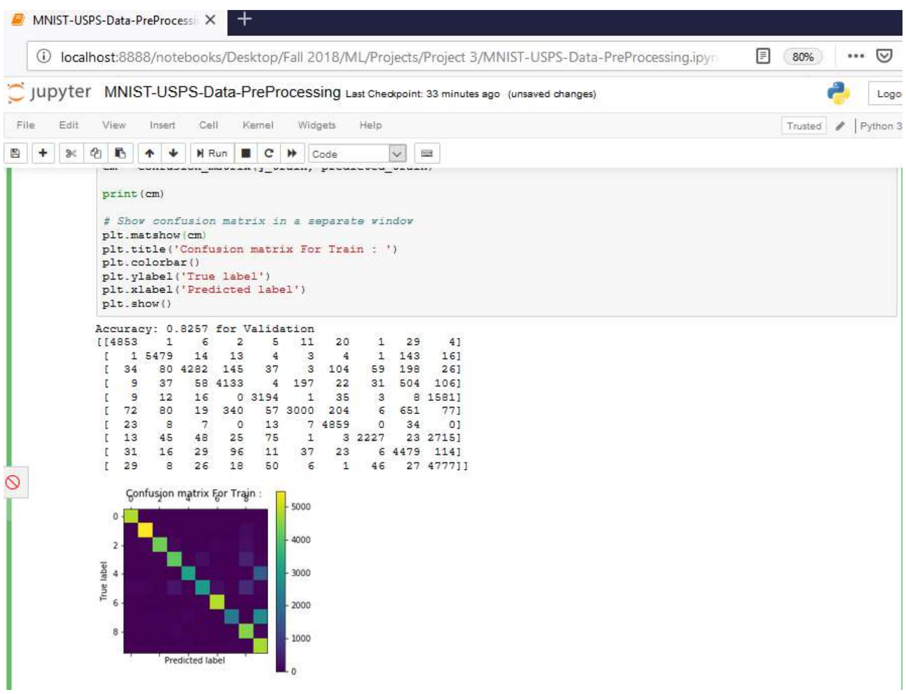

**Total Accuracy for Random Forest with C=5 ; gamma=0.001 ; max_iter = 200**



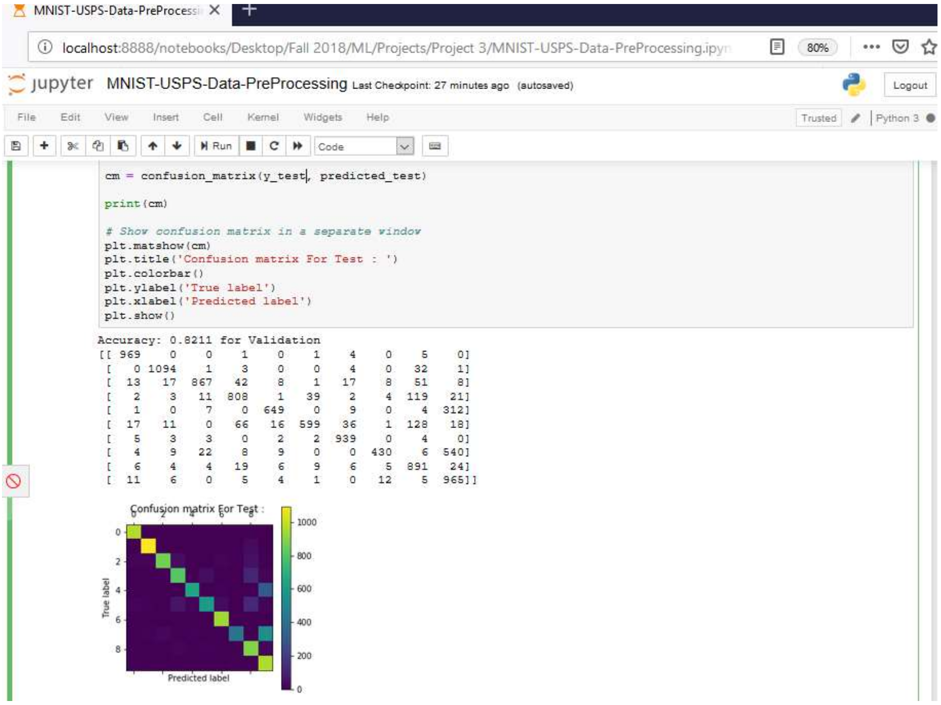
**Confusion Matrix for Validation MNIST Data :**



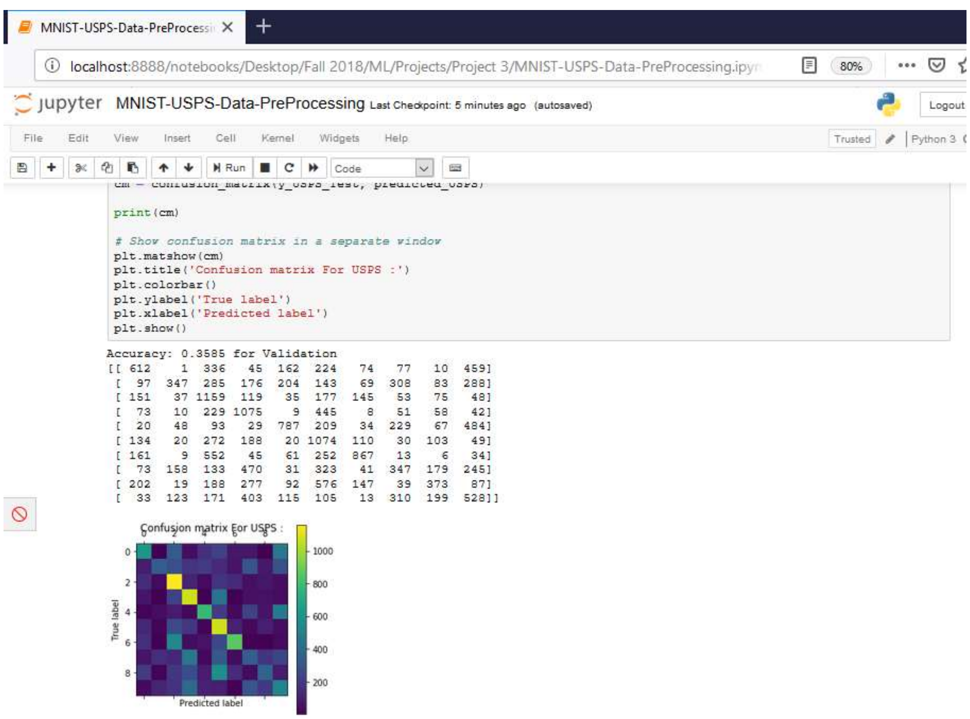

**Confusion Matrix for Training MNIST Data :**



**Confusion Matrix for Testing MNIST Data :**




**Confusion Matrix for Testing USPS Data :**



| Parameter Set | MNIST Validation Accuracy | MNIST Testing Accuracy | MNIST Training Accuracy | USPS Testing Accuracy |
|---|---|---|---|---|
| **1** | **96.52** | **96.56** | **98.18** | **34.74** |
| **2** | **82.33** | **82.11** | **82.57** | **35.85** |

## Questions :

**1. We test the MNIST trained models on two different test sets: the test set from MNIST and a test set from the USPS data set. Do your results support the "No Free Lunch" theorem?**

**A—**

a) "No-Free Lunch Theorem" basically means that you get "No-Free Learning Output" by only examining the training sets. This means that we cannot get an output for a model just by examining the training data set. In our case of classification models, we had four models for classification with no model having zero error. All models had some error and when the other model was trained over these the margin of error grew exponentially; Say we had a 72% accuracy for MNIST data for model Logistic Regression but had a significantly lower value accuracy of 23% for USPS data.

b) This was because the models we trained were for the MNIST data and not for the USPS data, the models could relate and classify the MNIST data set with better accuracy than the USPS data. Since the USPS data set did not belong to the MNIST data set and was different to the models that were trained on MNIST data, the result was a higher margin for error in the USPS Accuracy.

c) Thus, after thoroughly analyzing the Models it is concluded that the results support the "NO-FREE-LUNCH" theorem.

**2. Observe the confusion matrix of each classifier and describe the relative strengths / weaknesses of each classifier. Which classifier has the overall best performance?**

**A—**

1) **Logistic Regression Model :** This model trained faster than other models but had poor accuracy compared to other models both in MNIST and USPS data sets.

2) **Deep Learning Neural Network Model :** This Model trained a bit slower compared to the Logistic Regression Model, it also had an accuracy as high as 90% for MNIST and 35% for USPS. But this model did not show the accuracy rate as high as the Random Forest Model

3) **Random Forest Model :** This model trained as fast as the DNN model and also showed a higher accuracy rate of 96% for MNIST data but failed to show the increase in accuracy for the USPS data with 25.74%.

**4) Support Vector Machine Model :** This model was the slowest to run compared to all model and took most of the time to process data and generate the output. But SVM Model showed the highest accuracy for both MNIST data 98.18% and USPS Data 34.74%.

5) The SVM & Random Forest Classifiers have a really competitive performance with both the Accuracy and Loss parameters. But the SVM Model is the most efficient classifier with the overall best performance.

**3) Combine the results of the individual classifiers using a classifier combination method such as majority voting. Is the overall combined performance better than that of any individual classifier?**

**A—**

1) In the Majority Voting type classifier system, a number of classifiers are assembled in one single file. This model contains all the training data of all the learning models used as classifiers.

2) The training data of these models is then used to take a hard or soft vote on the model which is to be selected for use. This is done by Majority Voting; in this the model which gives the highest accuracy is voted by using the training data of all the models.

3) Say for simplicity we have four classifiers, and the overall optimum performance model is Random Forest Model. Now, for majority voting all the training data sets will be compared with each other and then a vote will be passed to verify the model selected is really of optimum performance.

4) Yes, the overall combined performance of the classifier is optimum compared to individual performance of different classifiers. This is due to the Majority Voting, as it selects only the algorithm which can perform optimally for the given data.

## References :


http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/
https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932
https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
https://blog.statsbot.co/ensemble-learning-d1dcd548e936