RepMLA_Etivity_2_Ciaran_Cox_19170157

August 6, 2021

#**Artificial Intelligence - MSc** CS6501 - MACHINE LEARNING AND APPLICATIONS #**Business Analytics - MSc** ET5003 - MACHINE LEARNING APPLICATIONS ##***Annual Repeat*** ###Instructor: Enrique Naredo

###RepMLA_Etivity-2

1 Etivity-2

1.1 Introduction

[1]:
```
#@title Current Date
Today = '2021-08-04' #@param {type:"date"}
```

[2]:
```
#@markdown ---
#@markdown ### Enter your details here:
Student_ID = "19170157" #@param {type:"string"}
Student_full_name = "Ciaran Cox" #@param {type:"string"}
#@markdown ---
```

[3]:
```
#@title Notebook information
Notebook_type = 'Example' #@param ["Example", "Lab", "Practice", "Etivity",␣
 ↪"Assignment", "Exam"]
Version = 'Final' #@param ["Draft", "Final"] {type:"raw"}
Submission = True #@param {type:"boolean"}
```

- The goal of this E-tivity is to implement a Logistic regression and Multilayer perceptron algorithims and use them to solve MNIST classification.
- Logistic Regression is a binary classification model that uses the sigmoid probability function to classify an input feature as belonging to one class or another.
- Multi-Layer Perceptrons are built from layers of neurons. The weights and biases assigned to these neurons can be used to build a robust predictive model.

1.2 Dataset

- The MNIST dataset contains a set of handwriten digits from 0 - 9. The dataset can be found as part of the keras package of datasets and can be loaded using the following line of code:

[4]:
```
# import MNIST dataset from keras
from keras.datasets import mnist
```

- Each of the training images will later be reshaped into row vectors.
- The MNIST dataset contains training images and labels and test images and labels. Due to this the training and testing data can be easily into train and test sets.

1.3 Method

[5]:
```
# load mnist dataset
data = mnist.load_data()
```

[6]:
```
# Split data into training and testing data
(X_train, y_train), (X_test, y_test) = data
```

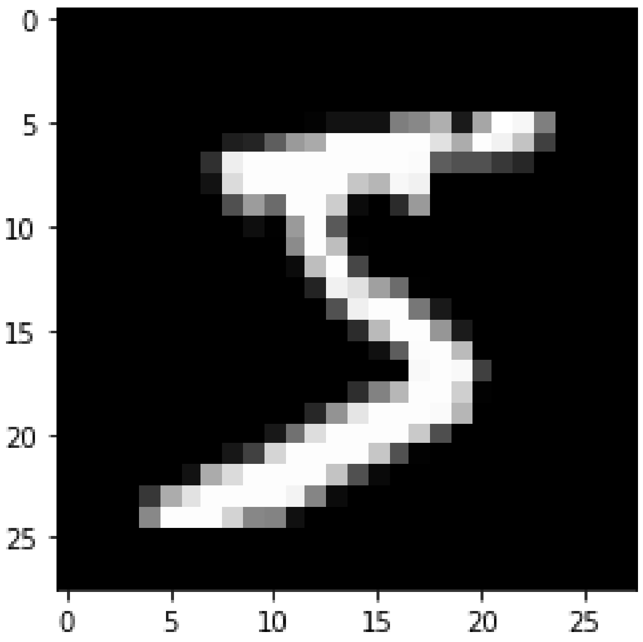
- X_train and X_test are the sets of training and testing images.
- There are 60000 image in the training set and 10000 in the testing set.
- Each image is a 28 x 28 pixel greyscale image, with each pixel having a value between 0 and 254.
- The y_train and y_test are the labels for each image, an integer between 0 and 9. We will use these to tell the perceptron which digit each image represents.
- There are 60000 labels in y_train and 10000 labels in y_test, matching the size of each X sets.

[7]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

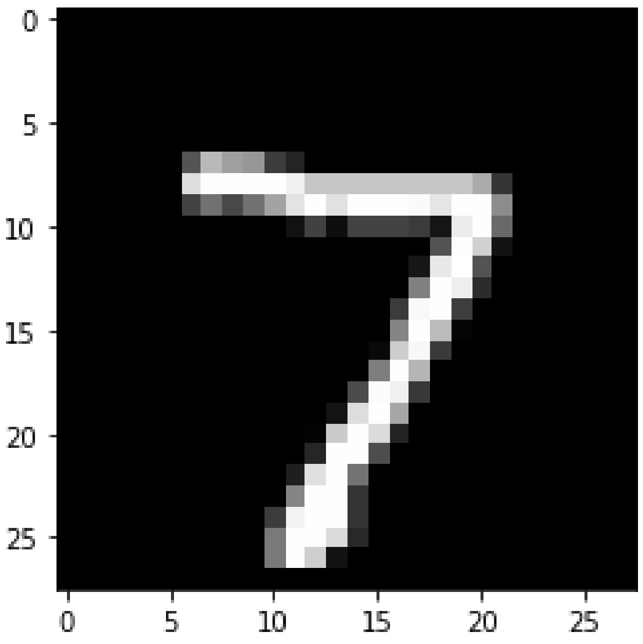
[8]:
```
# Displaying the image
import matplotlib.pyplot as plt
# Image number
im = 0

plt.imshow(X_train[im],cmap='gray')
print("y_train label: ", y_train[im])
```

```
y_train label:  5
```



[9]:
```
# Displaying the image
import matplotlib.pyplot as plt
# Image number
im = 0

plt.imshow(X_test[im],cmap='gray')
print("y_train label: ", y_test[im])
```

```
y_train label:  7
```



[10]:
```
# Reshaping the X_train and X_test set into row vectors
X_train_reshape = X_train.reshape(X_train.shape[0],784)
X_test_reshape = X_test.reshape(X_test.shape[0],784)
```

[11]:
```
# Scale images to 0,1
X_train_reshape = X_train_reshape/255
X_test_reshape = X_test_reshape/255
```

1.4 Train the Classifiers

1.5 Logistic Regression

[12]:
```
# Import NLPClassifier from sklearn
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(solver='saga',max_iter=500, tol=0.1)
LR.fit(X_train_reshape, y_train)
```

[12]:
```
LogisticRegression(max_iter=500, solver='saga', tol=0.1)
```

1.6 Multi-Layer Perceptron

[13]:
```
from sklearn.neural_network import MLPClassifier

MLP = MLPClassifier(hidden_layer_sizes=(50,),max_iter=500,
                    solver='sgd', alpha=1e-4, verbose=10,
                       random_state=1, learning_rate_init=0.1)
MLP.fit(X_train_reshape,y_train)
```

```
Iteration 1, loss = 0.32009978
Iteration 2, loss = 0.15347534
Iteration 3, loss = 0.11544755
Iteration 4, loss = 0.09279764
Iteration 5, loss = 0.07889367
Iteration 6, loss = 0.07170497
Iteration 7, loss = 0.06282111
Iteration 8, loss = 0.05530788
Iteration 9, loss = 0.04960484
Iteration 10, loss = 0.04645355
Iteration 11, loss = 0.04082169
Iteration 12, loss = 0.03828222
Iteration 13, loss = 0.03557957
Iteration 14, loss = 0.03054891
Iteration 15, loss = 0.02924761
Iteration 16, loss = 0.02610471
Iteration 17, loss = 0.02363894
Iteration 18, loss = 0.02208186
Iteration 19, loss = 0.01932900
Iteration 20, loss = 0.01830387
Iteration 21, loss = 0.01639227
Iteration 22, loss = 0.01392950
Iteration 23, loss = 0.01270193
Iteration 24, loss = 0.01234102
Iteration 25, loss = 0.01081313
Iteration 26, loss = 0.01028644
Iteration 27, loss = 0.00896707
Iteration 28, loss = 0.00744908
Iteration 29, loss = 0.00707946
Iteration 30, loss = 0.00573869
Iteration 31, loss = 0.00499554
Iteration 32, loss = 0.00477064
Iteration 33, loss = 0.00395458
Iteration 34, loss = 0.00355619
Iteration 35, loss = 0.00375497
Iteration 36, loss = 0.00304228
Iteration 37, loss = 0.00264245
Iteration 38, loss = 0.00241425
Iteration 39, loss = 0.00234957
Iteration 40, loss = 0.00233803
Iteration 41, loss = 0.00204653
Iteration 42, loss = 0.00199057
Iteration 43, loss = 0.00190567
Iteration 44, loss = 0.00180530
Iteration 45, loss = 0.00175054
Iteration 46, loss = 0.00168160
Iteration 47, loss = 0.00162517
Iteration 48, loss = 0.00159676
Iteration 49, loss = 0.00154993
Iteration 50, loss = 0.00152799
Iteration 51, loss = 0.00146697
Iteration 52, loss = 0.00145257
Iteration 53, loss = 0.00143422
Iteration 54, loss = 0.00135888
Iteration 55, loss = 0.00134281
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.
```

[13]:

```
MLPClassifier(hidden_layer_sizes=(50,), learning_rate_init=0.1, max_iter=500,
              random_state=1, solver='sgd', verbose=10)
```

1.6.1 Find the accuarcy on the test sets

[14]:

```
#y_pred = LR.predict(X_test_reshape)
print("Logistic Regression Accuracy: ", LR.score(X_train_reshape,y_train))
```

```
Logistic Regression Accuracy:  0.9296
```

[15]:

```
print("MLP Accuracy: ", MLP.score(X_test_reshape,y_test))
```

```
MLP Accuracy:  0.9731
```

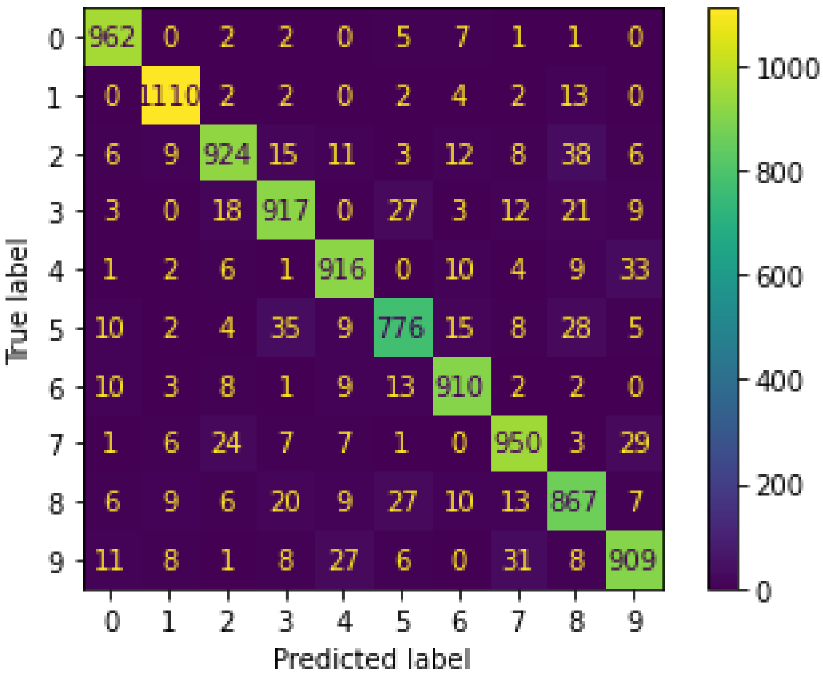
1.6.2 Plot the confusion matrices

[16]:

```
from sklearn.metrics import plot_confusion_matrix
```

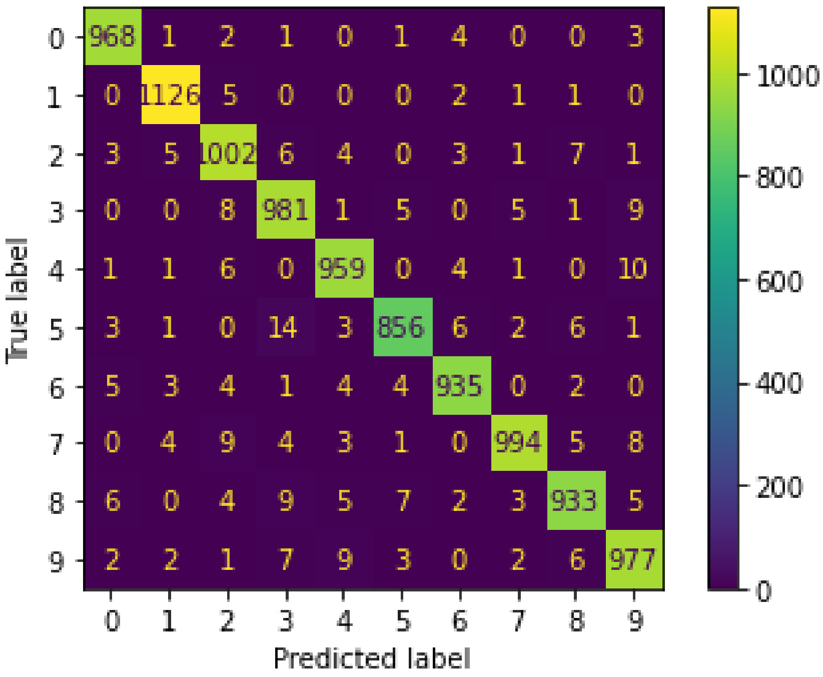
[17]:

```
## Classes for digits between 0 and 9 (Range of 10)
classes = [i for i in range(0,10)]

display = plot_confusion_matrix(LR, X_test_reshape, y_test,
                               display_labels=classes)
```



	0	1	2	3	4	5	6	7	8	9
0	962	0	2	2	0	5	7	1	1	0
1	0	1110	2	2	0	2	4	2	13	0
2	6	9	924	15	11	3	12	8	38	6
3	3	0	18	917	0	27	3	12	21	9
4	1	2	6	1	916	0	10	4	9	33
5	10	2	4	35	9	776	15	8	28	5
6	10	3	8	1	9	13	910	2	2	0
7	1	6	24	7	7	1	0	950	3	29
8	6	9	6	20	9	27	10	13	867	7
9	11	8	1	8	27	6	0	31	8	909

[18]:
```python
## Classes for digits between 0 and 9 (Range of 10)
classes = [i for i in range(0,10)]

display = plot_confusion_matrix(MLP, X_test_reshape, y_test,
                                display_labels=classes)
```



	0	1	2	3	4	5	6	7	8	9
0	968	1	2	1	0	1	4	0	0	3
1	0	1126	5	0	0	0	2	1	1	0
2	3	5	1002	6	4	0	3	1	7	1
3	0	0	8	981	1	5	0	5	1	9
4	1	1	6	0	959	0	4	1	0	10
5	3	1	0	14	3	856	6	2	6	1
6	5	3	4	1	4	4	935	0	2	0
7	0	4	9	4	3	1	0	994	5	8
8	6	0	4	9	5	7	2	3	933	5
9	2	2	1	7	9	3	0	2	6	977

1.6.3 Note

It is interesting to note that both confusion matices are similar for both classifiers. Both are highly accurate for number '1' and least accurate for number '5'.

1 a simple figure and is often drawn as a straight line, but 5 has more lines and curves and seems to be misclassified most often as 3.

1.6.4 Examples of bad predictions

[19]:
```
import numpy as np
```

Using Enriques plot_images function to plot all set of bad predictions

[20]:
```
# Function to plot an arrange of images
def plot_images(instances, images_per_row=5, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image,  cmap='gray', **options)
    plt.axis("off")
```

[21]:
```
pred_LR = LR.predict(X_test_reshape)
```

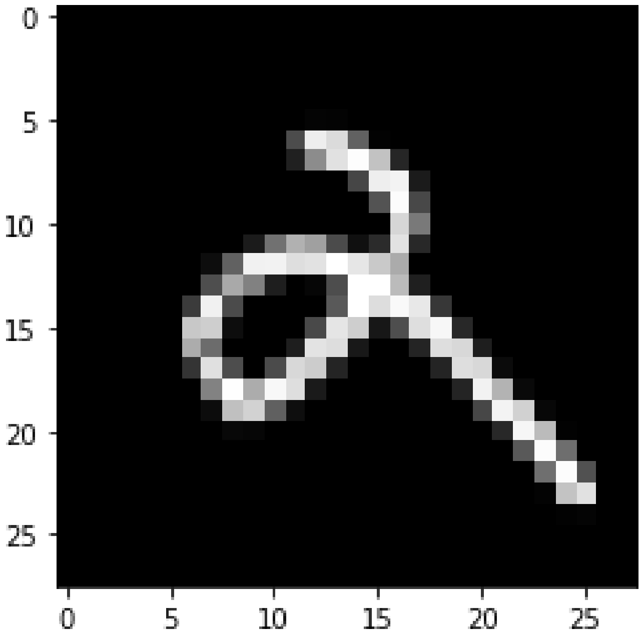
[22]:
```
pred_MLP = MLP.predict(X_test_reshape)
```

[23]:
```
# Lets find some examples that were not classified very well
bad_LR = []
for i in range(len(y_test)):
    if y_test[i] != pred_LR[i]:
        bad_LR.append(i)
```

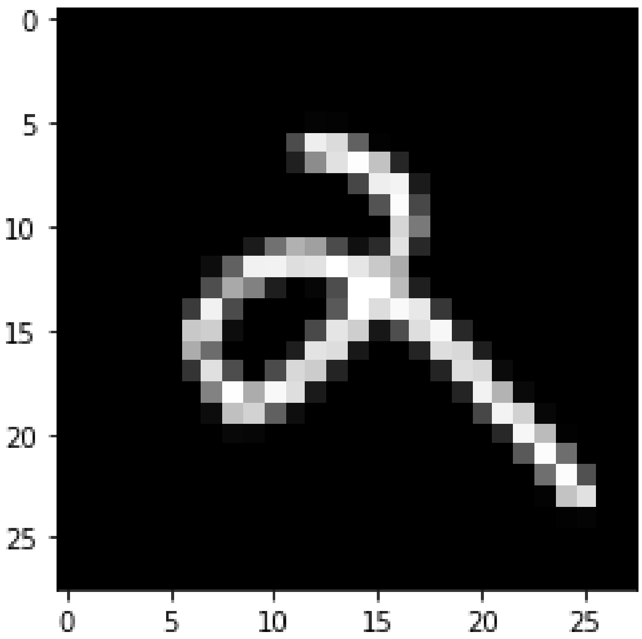
[24]:
```
# Lets find some examples that were not classified very well
bad_MLP = []
for i in range(len(y_test)):
    if y_test[i] != pred_MLP[i]:
        bad_MLP.append(i)
```

```
[25]: # Find some bad predictions common to both sets
      bad_all = np.intersect1d(bad_LR,bad_MLP)
```

```
[26]: example_num = bad_all[0]
      print("Prediction:", LR.predict([X_test_reshape[example_num]]))
      print("Actual:", y_test[example_num])
      plt.imshow(X_test[example_num],cmap='gray')

      Prediction: [9]
      Actual: 2

[26]: <matplotlib.image.AxesImage at 0x21270c19100>
```



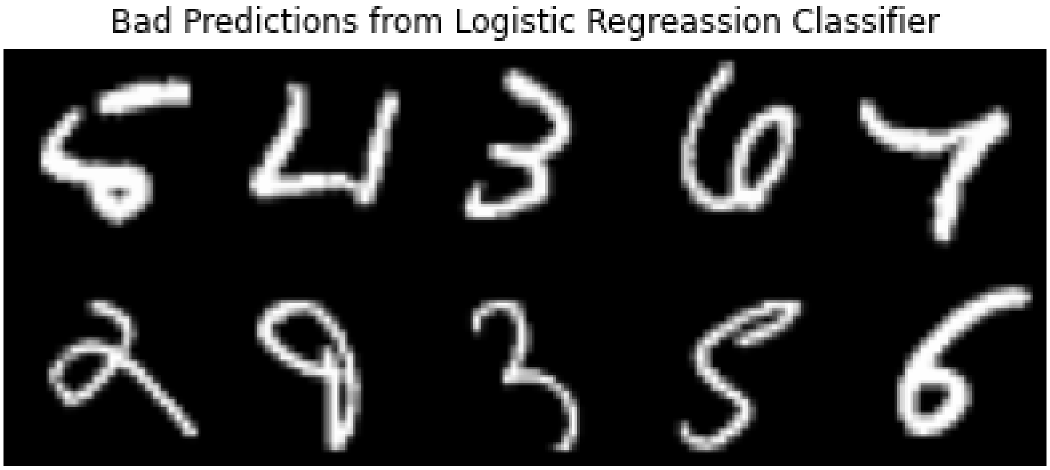
```
[27]: example_num = bad_all[0]
      print("Prediction:", MLP.predict([X_test_reshape[example_num]]))
      print("Actual:", y_test[example_num])
      plt.imshow(X_test[example_num],cmap='gray')

      Prediction: [4]
      Actual: 2

[27]: <matplotlib.image.AxesImage at 0x21270c61910>
```



[28]:

```
plt.figure(figsize=(7,7))
plot_images(X_test[bad_LR[:10]])
plt.title("Bad Predictions from Logistic Regreassion Classifier")
print("Labels: ", y_test[bad_LR[:10]])
print("Predictions: ", pred_LR[bad_LR[:10]])
```

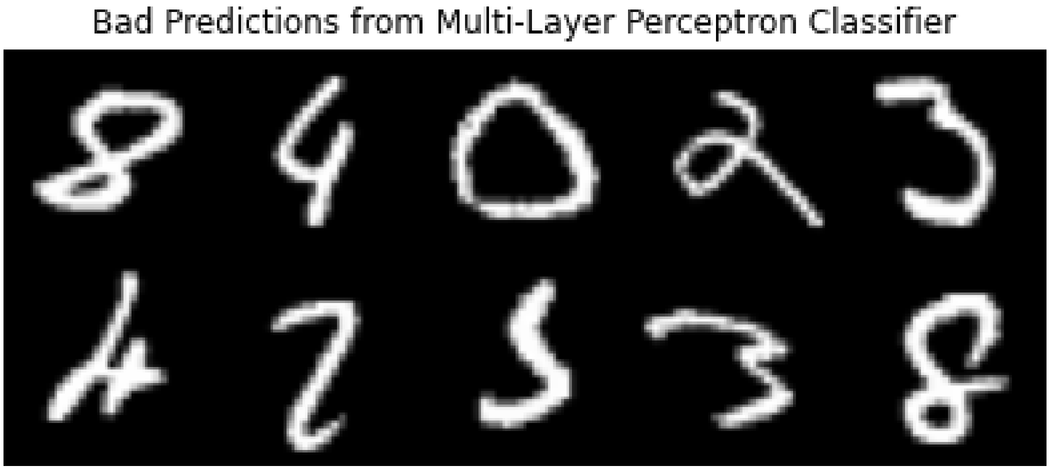
```
Labels:  [5 4 3 6 7 2 9 3 5 6]
Predictions:  [6 6 2 2 4 9 3 5 7 5]
```



[29]:

```python
plt.figure(figsize=(7,7))
plot_images(X_test[bad_MLP[:10]])
plt.title("Bad Predictions from Multi-Layer Perceptron Classifier")
print("Labels: ", y_test[bad_MLP[:10]])
print("Predictions: ", pred_MLP[bad_MLP[:10]])
```

```
Labels:  [8 4 0 2 3 4 2 5 3 8]
Predictions:  [2 9 9 4 2 2 7 3 7 2]
```

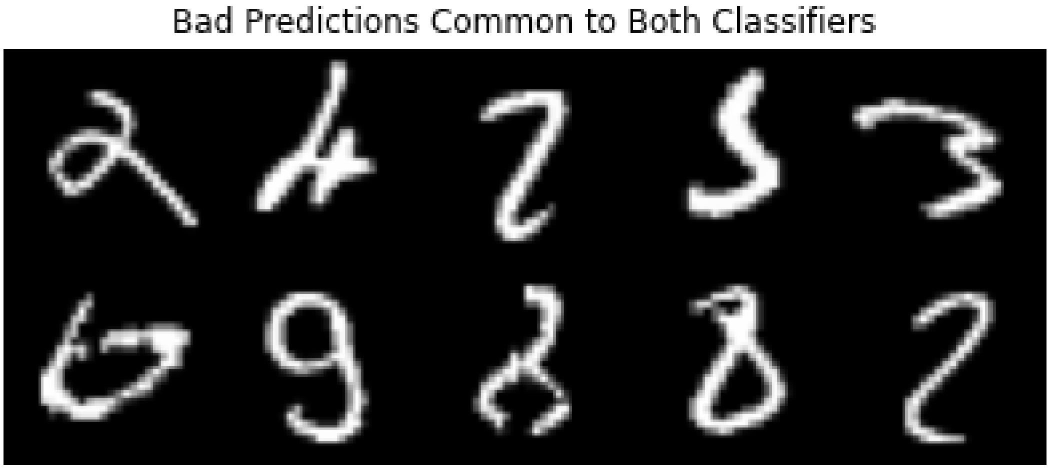


Bad Predictions from Multi-Layer Perceptron Classifier

[30]:

```python
plt.figure(figsize=(7,7))
plot_images(X_test[bad_all[:10]])
plt.title("Bad Predictions Common to Both Classifiers")
print("Labels: ", y_test[bad_all[:10]])
```

```
Labels:  [2 4 2 5 3 6 9 8 8 2]
```



Bad Predictions Common to Both Classifiers

1.7 Summary

- In this E-tivity I trained a Logestic Regression and a Multi-Layer Perceptron algorithims to predict what number is written in a set of hand written numbers.
- The MNIST set was split into training and test sets and labels.
- The training data was reshaped into row vectors.
- The data set is large but contains some poorly drawn numbers, as shown above.
- The perceptron appears to be more accurate in predicting the numbers.
- The confusion matrix gives some insight into how difficult some numbers are to classify.
- 1 appears to be the easiest for both classifiers and 5 appears to be the most difficult.
- From looking at the bad predictions that are common between both classifiers there are some that are not human readable, so some of these bad predictions could be down to bad data.

1.8 References

Logistic Regression Explained. [— Logistic Regression explained… | by z_ai | Towards Data Science. (n.d.). Retrieved August 1, 2021, from https://towardsdatascience.com/logistic-regression-explained-9ee73cede081

MNIST classification using multinomial logistic + L1 — scikit-learn 0.24.2 documentation. (n.d.). Retrieved August 6, 2021, from https://scikit-learn.org/stable/auto_examples/linear_model/plot_sparse_logistic_regression_mnist.html

Crash Course On Multi-Layer Perceptron Neural Networks. (n.d.). Retrieved August 1, 2021, from https://machinelearningmastery.com/neural-networks-crash-course/

Visualization of MLP weights on MNIST — scikit-learn 0.24.2 documentation. (n.d.). Retrieved August 6, 2021, from https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html

Confusion matrix — scikit-learn 0.24.2 documentation. (n.d.). Retrieved August 6, 2021, from https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html