4027845_ALISON_Etivity_2

August 3, 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]: #@title Current Date
     from datetime import date
     Today = date.today() #@param {type:"date"}
     print(Today)
```

```
2021-08-03
```

```
[2]: #@markdown ---
     #@markdown ### Enter your details here:
     Student_ID = "0427845" #@param {type:"string"}
     Student_full_name = "Alison O'Connor" #@param {type:"string"}
     #@markdown ---
```

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

1 Etivity-2

2 Introduction

The purpose of this notebook is to investigate the a dataset MNIST of handwritten numbers in the range of zero to nine using logistic regression. The dataset was provided as part of the class materials for this module but other forms of it are available to download online.

Both the training and test datasets used here contain 257 columns of data. The first column represents the known value of the handwritten number (i.e. the labelled class of the number), the remaining columns flattened arrays representing a 16×16 matrix of pixels. The training set contains examples from approximately 250 writers.

Logistic regression is a predictive analysis for binary dependent variables. The simplist form of logistic regression is known as binomial regression where the outcome for a dependent variable has only two possible types ('YES' or 'NO'). In our case given a dataset of handwritten values between 0-9 we look at each individual test set value and classify it as 'YES' or 'NO' in comparision to a specific target value. For example, if our test value is known to be a handwritten numeral '7' we test the variable against each possible target (i.e. 0, 1, 2.....9) and assign a probability that the test value is that target. The test class is assigned to the value with the greatest probability. Where the outcome has three or more possiblities multinominal logistic regression is applied. Multinominal logistic regression is almost identical to binary logistic regression excepting that there 'K' possible outcomes as opposed to two ('YES' or 'NO') so the example previously given is simplified as we no longer need to search every possible target for every test case as the model should be able to identify features that are associated with a specific categorical outcome.

2.1 Dataset

The MNIST dataset is imported from sklearn datasets.The dataset consists of 1797 samples with a flattened arrayof features representing an 8x8 image of a single handwritten digit.

The data is segregated into the following sections: 1. X represents the 8x8 features of the dataset 2. y represents the known classification of the value (i.e. the target)

Using sklearn the data are separated into a training set (comprised of 80% of the original dataset) and a test set (comprised of 20% of the original dataset). These arrays are named as follows: 1. Xtrain: 80% of original dataset features 2. ytrain: 80% of original known classification 3. Xtest: 20% of original dataset features 4. ytest: 20% of original known classification

[4]:
```
# import some data from sklearn
from sklearn import datasets

# load the MNIST (digits) dataset
mnist = datasets.load_digits()

##SPLIT INTO DATA AND TARGET
X=mnist.data
y=mnist.target

from sklearn.model_selection import train_test_split
##SPLIT DATASET INTO TRAINING AND TEST SET (80% training, 20% test)
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2)

print('The Xtrain dataset contains %s rows and %s columns of data representing␣
 ↪80%% of the original dataset'
      %(Xtrain.shape[0], Xtrain.shape[1]))
print('The Xtest dataset contains %s rows and %s columns of data 20%% of the␣
 ↪original dataset'
      %(Xtest.shape[0], Xtest.shape[1]))
```

```
The Xtrain dataset contains 1437 rows and 64 columns of data representing 80% of
the original dataset
```

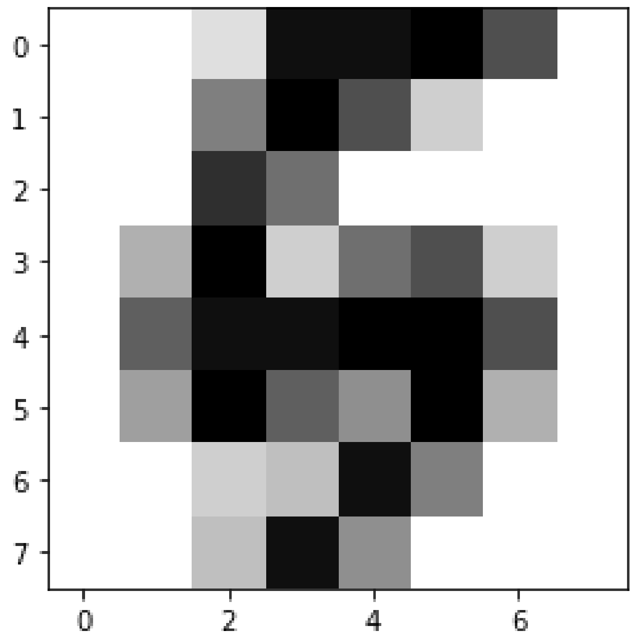
```
The Xtest dataset contains 360 rows and 64 columns of data 20% of the original
dataset
```

3 Plot a value from the training set

Here we show an image from the training set to show how these handwritten values appear to the machine. We also print the known value from the target list

[5]:
```
import matplotlib.pyplot as plt

plt.imshow(Xtrain[0].reshape(8,8), cmap=plt.cm.gray_r, interpolation='nearest')

print('The known class of this image is %s'%(ytrain[0]))
```

```
The known class of this image is 5
```



4 Scaling the data

The imported dataset features represent the number of pixels (0-16) in a nonoverlapping 4x4 block. To prepare the dataset for logistic regression we scale the data to be in the range of 0-1. To apply this scaling we find the maximum value in feature list and then divide all features by this value.

[6]:
```
##FOR COMPARISION PURPOSES PRINT FIRST ROW OF Xtrain
print('Xtrain features prior to scaling')
print(Xtrain[0])
print('\n')
```

```
##Calculate the maximum feature value
my_max=max(Xtrain.max(), Xtest.max())


##APPLY SCALING TO THE X DATA
scaleXtrain=Xtrain/my_max
scaleXtest=Xtest/my_max

##CHECK THIS HAS BEEN APPLIED
print('Xtrain features post scaling')
print(scaleXtrain[0])
```

```
Xtrain features prior to scaling
[ 0.  0.  2. 15. 15. 16. 11.  0.  0.  0.  8. 16. 11.  3.  0.  0.  0.  0.
 13.  9.  0.  0.  0.  0.  0.  5. 16.  3.  9. 11.  3.  0.  0. 10. 15. 15.
 16. 16. 11.  0.  0.  6. 16. 10.  7. 16.  5.  0.  0.  0.  3.  4. 15.  8.
  0.  0.  0.  0.  4. 15.  7.  0.  0.  0.]


Xtrain features post scaling
[0.     0.     0.125  0.9375 0.9375 1.     0.6875 0.     0.     0.
 0.5    1.     0.6875 0.1875 0.     0.     0.     0.     0.8125 0.5625
 0.     0.     0.     0.     0.     0.3125 1.     0.1875 0.5625 0.6875
 0.1875 0.     0.     0.625  0.9375 0.9375 1.     1.     0.6875 0.
 0.     0.375  1.     0.625  0.4375 1.     0.3125 0.     0.     0.
 0.1875 0.25   0.9375 0.5    0.     0.     0.     0.     0.25   0.9375
 0.4375 0.     0.     0.    ]
```

4.1 Method

This model uses logistic regression from the sklearn package. Given the multi-class problem of the dataset (where the target has nine possible outcomes) a 'multinomial' parameter is selected with a 'lbfgs' solver applied. The fit intercept parameter is turned on to ensure a bias is added to the decision function.

The algorithm failed to converge to a solution when the maximum number of iterations was set to the default value of 100. This value was increased to 300 iterations. The increased number of maximum iterations allows the solver to converge the result but the question remains why did we have to increase the number of iterations in the first place?

The number of iterations required to converge a solution is closely linked to the normalising and scaling of feature data. Clearly scaling the data to a range of 0-1 allows data to be more quickly compared across arrays but it does not account for data spread (i.e. the standard deviation from mean) or shape. A good explanation of normalising and standardising datasets and why it is import to correctly assess this feature can be found here. For regression analyses it is often useful to scale features to have a mean of 0 as a mean of 0 is easier to interpret in terms of the bias.

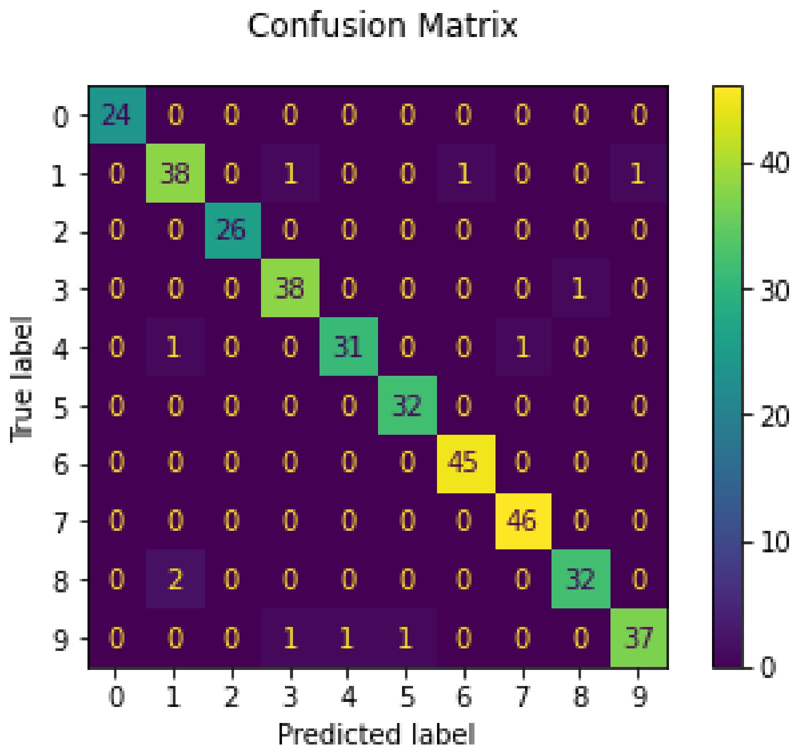
[7]:
```
##import logistic regression
from sklearn.linear_model import LogisticRegression
import numpy as np

LR=LogisticRegression(multi_class='multinomial', solver='lbfgs',␣
 ↪fit_intercept=True, max_iter=300)
#FIT THE MODEL TO THE TRAINING SET
LR.fit(scaleXtrain, ytrain)
##GET A PREDICTION
y_pred=LR.predict(scaleXtest)

print('Training set score: %s'%(LR.score(scaleXtrain, ytrain)))

for i in np.arange(0, 10):
    print('The bias added to the decision function was %s for class %s'%(LR.
 ↪intercept_[i], i) )
```

```
Training set score: 0.9860821155184412
The bias added to the decision function was 0.7051154095193303 for class 0
The bias added to the decision function was -3.289767478001679 for class 1
The bias added to the decision function was -0.1590486774188588 for class 2
The bias added to the decision function was -0.2157964679250671 for class 3
The bias added to the decision function was 3.7610721807795833 for class 4
The bias added to the decision function was 0.021725538950562995 for class 5
The bias added to the decision function was -0.2821198248576183 for class 6
The bias added to the decision function was 1.8782162772957731 for class 7
The bias added to the decision function was -1.0620216716396575 for class 8
The bias added to the decision function was -1.3573752867024125 for class 9
```

[8]:
```
# classification_report builds a text report showing the main classification␣
 ↪metrics
from sklearn import metrics

print(f"Classification report for classifier {LR}:\n"
      f"{metrics.classification_report(ytest, y_pred)}\n")
```

```
Classification report for classifier LogisticRegression(max_iter=300,
multi_class='multinomial'):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        24
           1       0.93      0.93      0.93        41
           2       1.00      1.00      1.00        26
           3       0.95      0.97      0.96        39
           4       0.97      0.94      0.95        33
           5       0.97      1.00      0.98        32
           6       0.98      1.00      0.99        45
           7       0.98      1.00      0.99        46
           8       0.97      0.94      0.96        34
           9       0.97      0.93      0.95        40

    accuracy                           0.97       360
   macro avg       0.97      0.97      0.97       360
weighted avg       0.97      0.97      0.97       360
```

[9]:
```python
## Confusion matrix
# of the true digit values and the predicted digit values

disp = metrics.plot_confusion_matrix(LR, scaleXtest, ytest)
disp.figure_.suptitle("Confusion Matrix")

plt.show()
```

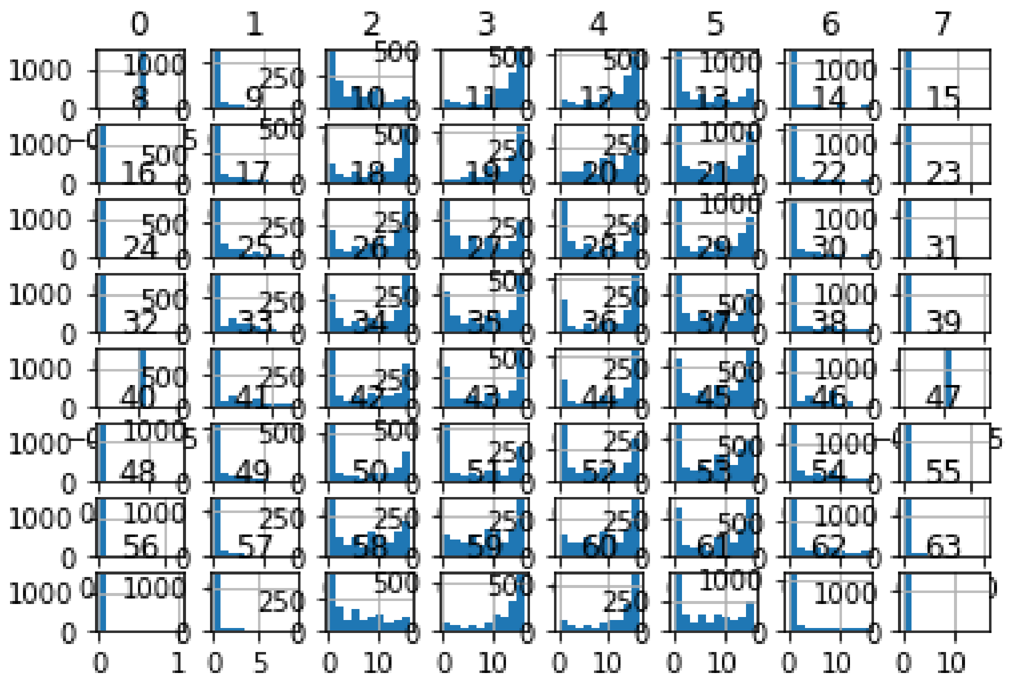


5 Scaling using multiple methods

Here we investigate the use of sklearn standardising and normalising features on the logistic regression method. We first plot a histogram of the original data (as-received). We then apply a standard scaler to look at how the x-axis data might be re-evaluated with regards to mean and standard deviation of the dataset. The resultant arrays are now outside the range 0-1 so we apply sklearn MinMax scaler to the standardised data to see what affect that has. The returned array is now identical to our original data indicating that standardisation doesn't affect our analysis.

```
[10]: ##FOR COMPARISION PURPOSES PRINT FIRST ROW OF Xtrain
print('Xtrain features prior to scaling')
print(Xtrain[0])
print('\n')

##IMPORT PANDAS
import pandas as pd

print('This histogram shows that the data is skewed significantly over the␣
 ↪array.')

##CONVERT ORIGINAL DATA TO DF AND PLOT HISTOGRAM
df=pd.DataFrame(Xtrain)
df.hist()
plt.show()
```

```
Xtrain features prior to scaling
[ 0.  0.  2. 15. 15. 16. 11.  0.  0.  0.  8. 16. 11.  3.  0.  0.  0.  0.
 13.  9.  0.  0.  0.  0.  0.  5. 16.  3.  9. 11.  3.  0.  0. 10. 15. 15.
 16. 16. 11.  0.  0.  6. 16. 10.  7. 16.  5.  0.  0.  0.  3.  4. 15.  8.
  0.  0.  0.  0.  4. 15.  7.  0.  0.  0.]


This histogram shows that the data is skewed significantly over the array.
```

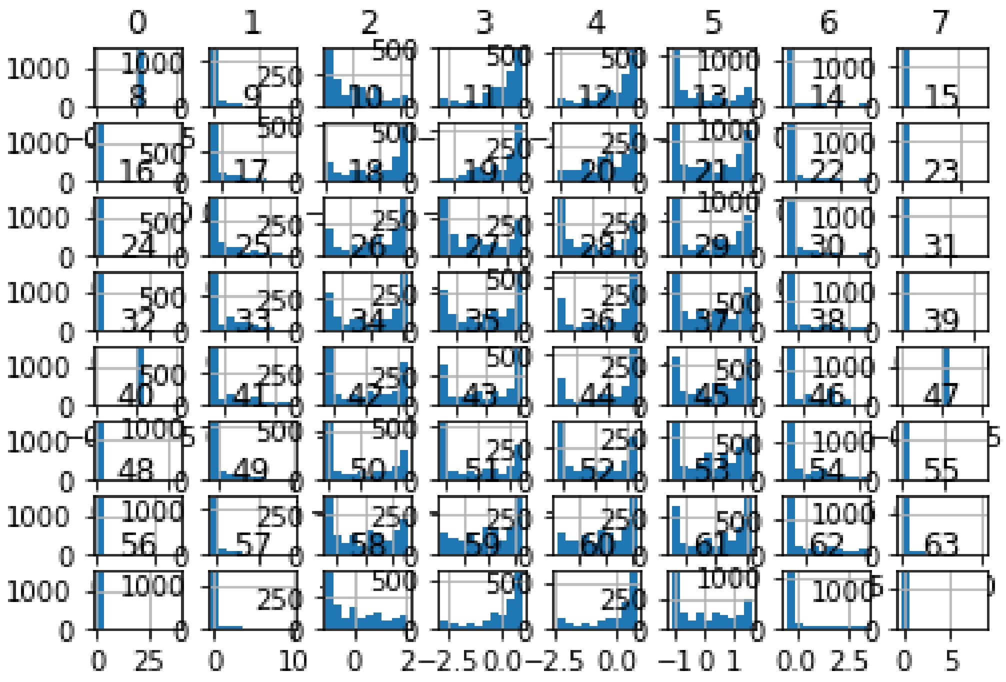


```
[11]: print('Here we apply the standard scaler to the data to see how this affects␣
       ↪data')
      ##IMPORT STANDARDSCALER AND MAXMINSCALER
      from sklearn.preprocessing import StandardScaler

      scaler=StandardScaler()
      ss_Xtrain=scaler.fit_transform(Xtrain)
      ss_Xtest=scaler.fit_transform(Xtest)


      ##CHECK THIS HAS BEEN APPLIED
      print('Xtrain features post scaling')
      print(ss_Xtrain[0])

      print('\n')
      print('The application of the standard scaler has not affected the shape of the␣
       ↪data but does affect the x-axis range considerably')

      ##convert to df and plot histogram
      df=pd.DataFrame(ss_Xtrain)
      df.hist()
      plt.show()
```

```
Here we apply the standard scaler to the data to see how this affects data
Xtrain features post scaling
```

```
[ 0.         -0.33795605 -0.69080606  0.7445758   0.74140249  1.81808399
  2.96593461 -0.12262293 -0.06141664 -0.63125904 -0.45942525  1.0004059
  0.14047017 -0.85304072 -0.51049523 -0.13276033 -0.04991522 -0.73969771
  0.53702245  0.33311852 -1.15152555 -1.26700467 -0.54650062 -0.11790505
 -0.02638899  0.79517825  1.12896612 -0.97432338 -0.13238266  0.58689344
  0.17891409 -0.04573894  0.          2.16430265  1.15884589  0.95598405
  0.96854015  1.24280712  2.28422701  0.         -0.05396308  1.5101777
  1.39651736  0.44236214 -0.10663146  1.37616702  0.36141118 -0.08905824
 -0.03980054 -0.40831918 -0.79848991 -1.05700458  1.02880706 -0.13332629
 -0.75315088 -0.21703826 -0.02638899 -0.29755744 -0.31643257  0.66001752
 -1.00358738 -1.13912012 -0.50183892 -0.2034412 ]
```

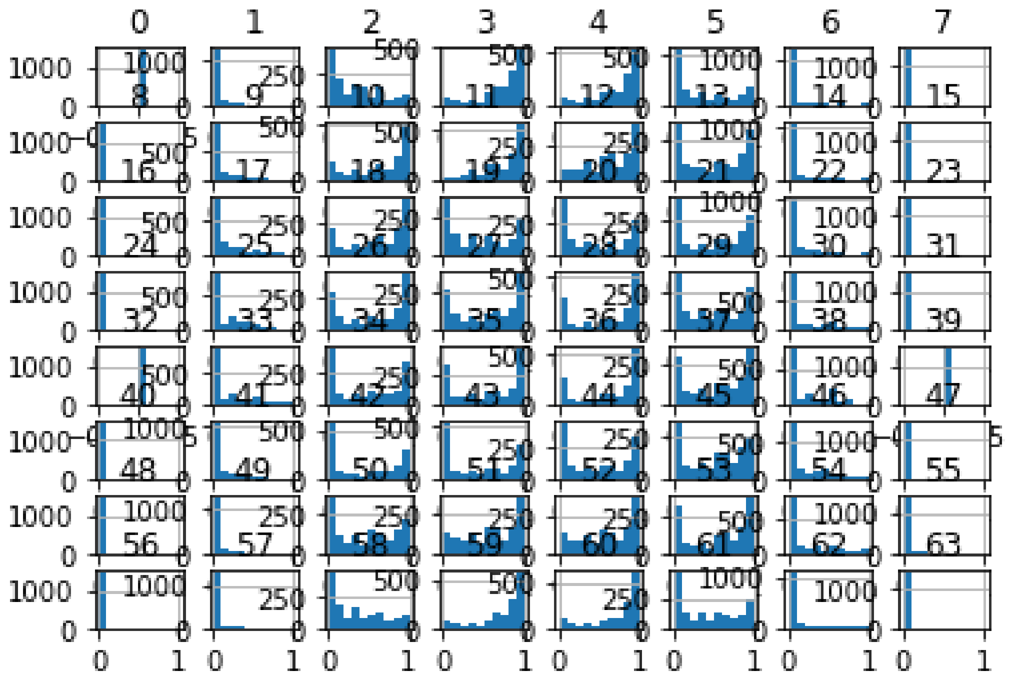
The application of the standard scaler has not affected the shape of the data but does affect the x-axis range considerably



```python
[12]: print('Here we normalise the scaled data using sklearns MinMaxScaler function')

from sklearn.preprocessing import MinMaxScaler

minScaler=MinMaxScaler()

sn_Xtrain=minScaler.fit_transform(ss_Xtrain)
sn_Xtest=minScaler.fit_transform(ss_Xtest)
```

```python
print('\n')

##CHECK THIS HAS BEEN APPLIED
print('Xtrain features post scaling and normalising')
print(sn_Xtrain[0])
print('\n')

print('The application of the min max scaler has now normalised the x-axis to␣
↪range of 0-1')
print('The returned value is now identical to our original analysis.')
print('This indicates that standardisation doesn''t impact the analysis')

##convert to df and plot histogram
df=pd.DataFrame(sn_Xtrain)
df.hist()
plt.show()
```

```
Here we normalise the scaled data using sklearns MinMaxScaler function


Xtrain features post scaling and normalising
[0.         0.         0.125      0.9375     0.9375     1.
 0.6875     0.         0.         0.         0.5        1.
 0.6875     0.1875     0.         0.         0.         0.
 0.8125     0.5625     0.         0.         0.         0.
 0.         0.33333333 1.         0.1875     0.5625     0.6875
 0.2        0.         0.         0.71428571 0.9375     0.9375
 1.         1.         0.78571429 0.         0.         0.375
 1.         0.625      0.4375     1.         0.3125     0.
 0.         0.         0.1875     0.25       0.9375     0.5
 0.         0.         0.         0.         0.25       0.9375
 0.4375     0.         0.         0.        ]


The application of the min max scaler has now normalised the x-axis to range of
0-1
The returned value is now identical to our original analysis.
This indicates that standardisation doesnt impact the analysis
```



[ ]:

5.1 Summary

5.1.1 Guidelines

Use this cell as a guideline and remove it when submitting the Etivity

In this notebook I have imported the MNIST dataset and analysed, using logistic regression, the classification of handwritten numbers. The data was preprocessed to normalise arrays in the range 0-1.

The logistic regression method shows a high precision and recall indicating the model was able to correctly classify digits approximately 97% of the time. The confusion matrix shows the number '4' was misclassified on three occasions as a '1' and '8' and a '9'.

I investigated the use of standardisation and normalisation on the dataset and found that standardising the data did not influence the result on logistic regression as post-standardising the data to a normalised range produced arrays that exactly match what can be achieved by normalisation alone. Hence re-running the logistic regression method on the standardised and normalised dataset would return the same result achieved previously.

This dataset is a reduced dataset compared to the one previously used. Here the arrays are mapped to an 8x8 matrix which returns a reduced quality image. This may account for why the misclassification for the number '4' occurred so regularly. The blurry nature of the image makes it more difficult to assess. On the other hand, the reduced quality resulted in significantly improved performance in terms of computing time.