Assignment 2

Manpreet Kaur

November 12, 2019

Class: ALY6020- **Predictive Analytics Professor:**Marco Montes de Oca

Introduction:

Although the term Logistic regression is used, it is a classification technique. It is geometrically a very elegant algorithm. Logistic regression can be derived from geometry, probability and loss function.

It can be used if the data is linearly separable (i.e. there are lines/planes separating the two classes). This data is separated by a plane represented as below:

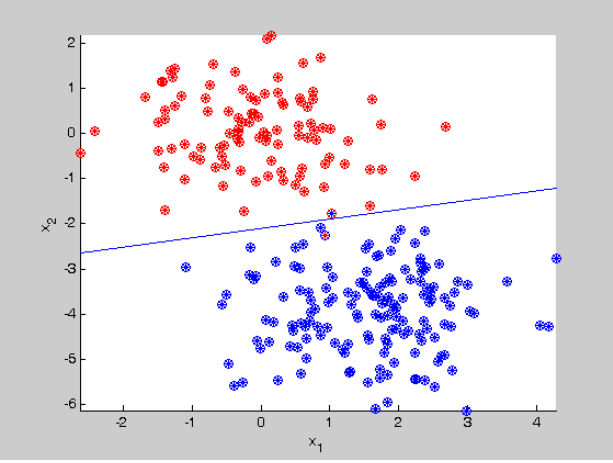
$$\prod = W^T X + b$$

Where W is the normal to the plane and b is the intercept. Both W and X are vectors while, b is a scalar.
If plane passes through the origin, then b= 0,

$$\prod = W^T X$$

Image representation of:

$\prod = W^T X + b$



Geometric derivation of the logistic regression model:

Given Data of Logistic Regression:
The dataset consists of two classes, red and blue in color. Let the red color class be the positive class and blue color class be the negative class

$D_n = \{+_{ve}, -_{ve}\}$

Assumption of Logistic Regression
Classes are almost/perfectly linearly separated.

Task of Logistic Regression
Find W and b which will be the used to find the best find plane - $\prod$ for the given classes. In other words, find the plane which best separated the two classes.

Geometric Derivation
For simplicity, I have assumed that the plane passes through the origin.

$$\prod = W^T X$$

If a query point belongs the positive class, it is denoted with +1 and if belongs the negative class, it is denoted by -1.

$$Y_i = \{+1, -1\}$$

The distance of the point 'i' to the plane is Di.

$$\mathrm{Di} = \frac{W^T Xi}{||W||}$$

Assuming that W is normal to the plane.

$$\mathrm{Di} = \frac{W^T Xi}{1}$$

When W and X_i are pointing to the same side of the plane,
$D_i = W^T X_i > 0$

When W and X_i are pointing to the opposite side of the plane,
$D_i = W^T X_i < 0$

When $W^TX_i > 0$ the point is classified as a positive point, $Y_i = +1$
When $W^TX_i < 0$ the point is classified as a negative point, $Y_i = -1$

Cases when point has been correctly classified and incorrectly classified-

Case 1:
Given $Y_i = +_{ve}$ and $W^TX_i > 0$, then the plane has **correctly** classified the point

$$\mathbf{Y_i(W^TX_i) > 0}$$

Case 2:
Given $Y_i = -_{ve}$ and $W^TX_i < 0$, then the plane has **correctly** classified the point

$$\mathbf{Y_i(W^TX_i) > 0}$$

Case 3:
Given $Y_i = +_{ve}$ and $W^TX_i < 0$, then the plane has **incorrectly** classified the point

$$\mathbf{Y_i(W^TX_i) < 0}$$

Case 4:
Given $Y_i = -_{ve}$ and $W^TX_i > 0$, then the plane has **correctly** classified the point

$$\mathbf{Y_i(W^TX_i) < 0}$$

We can maximize the number of points having points $Y_i(W^TX_i) > 0$
The variable 'W' decides the same.

Problem Statement

Character recognition with Logistic Regression

About the Dataset

There are 2 files with 28 *28 grayscale images. Each image is encoded as a row of 784 integer values between 0 and 255 indicating the brightness of each pixel. The label associated with each image is encoded as an integer value between 0 and 9. The dataset is an MNIST dataset.

Code Explanation:

Imported the necessary libraries for the project and also the train and test csv files from my local location

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
image_size = 28 # width and length
no_of_different_labels = 10 #  i.e. 0, 1, 2, 3, ..., 9
image_pixels = image_size * image_size

test_data = np.loadtxt("/Users/manu/Downloads/MNIST-data/mnist_test.csv", delimiter=",")
train_data = np.loadtxt("/Users/manu/Downloads/MNIST-data/mnist_train.csv", delimiter=",")
```

Before we begin the model formation, a look how each of the 784 pixels form the digits of the image

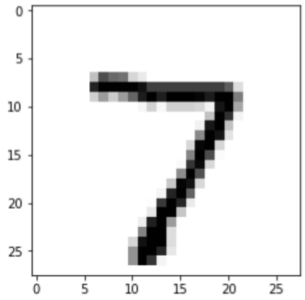
```
fac = 0.99 / 255
train_imgs = np.asfarray(train_data[:, 1:]) * fac + 0.01
test_imgs = np.asfarray(test_data[:, 1:]) * fac + 0.01
train_labels = np.asfarray(train_data[:, :1])
test_labels = np.asfarray(test_data[:, :1])
```

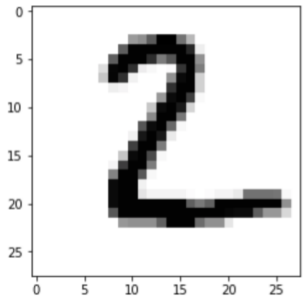
```
import numpy as np
lr = np.arange(10)
for label in range(10):
    one_hot = (lr==label).astype(np.int)
    print("label: ", label, " in one-hot representation: ", one_hot)
```

```
label:  0  in one-hot representation:  [1 0 0 0 0 0 0 0 0 0]
label:  1  in one-hot representation:  [0 1 0 0 0 0 0 0 0 0]
label:  2  in one-hot representation:  [0 0 1 0 0 0 0 0 0 0]
label:  3  in one-hot representation:  [0 0 0 1 0 0 0 0 0 0]
label:  4  in one-hot representation:  [0 0 0 0 1 0 0 0 0 0]
label:  5  in one-hot representation:  [0 0 0 0 0 1 0 0 0 0]
label:  6  in one-hot representation:  [0 0 0 0 0 0 1 0 0 0]
label:  7  in one-hot representation:  [0 0 0 0 0 0 0 1 0 0]
label:  8  in one-hot representation:  [0 0 0 0 0 0 0 0 1 0]
label:  9  in one-hot representation:  [0 0 0 0 0 0 0 0 0 1]
```

```
lr = np.arange(no_of_different_labels)
# transform labels into one hot representation
#train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)
# we don't want zeroes and ones in the labels neither:
#train_labels_one_hot[train_labels_one_hot==0] = 0.01
#train_labels_one_hot[train_labels_one_hot==1] = 0.99
test_labels_one_hot[test_labels_one_hot==0] = 0.01
test_labels_one_hot[test_labels_one_hot==1] = 0.99
```

```
for i in range(10):
    img = test_imgs[i].reshape((28,28))
    plt.imshow(img, cmap="Greys")
    plt.show()
```





The train dataset is of the type array

```
#Checking the output format of train dataset
#it is of type array
train_data
```

```
array([[5., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [4., 0., 0., ..., 0., 0., 0.],
       ...,
       [5., 0., 0., ..., 0., 0., 0.],
       [6., 0., 0., ..., 0., 0., 0.],
       [8., 0., 0., ..., 0., 0., 0.]])
```

It is simpler to work with data frames, hence converted the arrays to data frames

```
#Converting from array to data frame
df = pd.DataFrame.from_records(train_data)
df_test = pd.DataFrame.from_records(test_data)
```

Converting the target variable of test case to a list

```
# conerting target of test case values to list
Y_test = df_test[0].tolist()
print(Y_test)
```

```
[7.0, 2.0, 1.0, 0.0, 4.0, 1.0, 4.0, 9.0, 5.0, 9.0, 0.0, 6.0, 9.0, 0.0, 1.0, 5.0, 9.0, 7.0, 3.0, 4.0, 9.0, 6.0, 6.0, 5
.0, 4.0, 0.0, 7.0, 4.0, 0.0, 1.0, 3.0, 1.0, 3.0, 4.0, 7.0, 2.0, 7.0, 1.0, 2.0, 1.0, 1.0, 7.0, 4.0, 2.0, 3.0, 5.0, 1.0
, 2.0, 4.0, 4.0, 6.0, 3.0, 5.0, 5.0, 6.0, 0.0, 4.0, 1.0, 9.0, 5.0, 7.0, 8.0, 9.0, 3.0, 7.0, 4.0, 6.0, 4.0, 3.0, 0.0,
7.0, 0.0, 2.0, 9.0, 1.0, 7.0, 3.0, 2.0, 9.0, 7.0, 7.0, 6.0, 2.0, 7.0, 8.0, 4.0, 7.0, 3.0, 6.0, 1.0, 3.0, 6.0, 9.0, 3.
0, 1.0, 4.0, 1.0, 7.0, 6.0, 9.0, 6.0, 0.0, 5.0, 4.0, 9.0, 9.0, 2.0, 1.0, 9.0, 4.0, 8.0, 7.0, 3.0, 9.0, 7.0, 4.0, 4.0,
4.0, 9.0, 2.0, 5.0, 4.0, 7.0, 6.0, 7.0, 9.0, 0.0, 5.0, 8.0, 5.0, 6.0, 6.0, 5.0, 7.0, 8.0, 1.0, 0.0, 1.0, 6.0, 4.0, 6.
0, 7.0, 3.0, 1.0, 7.0, 1.0, 8.0, 2.0, 0.0, 2.0, 9.0, 9.0, 5.0, 5.0, 1.0, 5.0, 6.0, 0.0, 3.0, 4.0, 4.0, 6.0, 5.0, 4.0,
6.0, 5.0, 4.0, 5.0, 1.0, 4.0, 4.0, 7.0, 2.0, 3.0, 2.0, 7.0, 1.0, 8.0, 1.0, 8.0, 1.0, 8.0, 5.0, 0.0, 8.0, 9.0, 2.0, 5.
0, 0.0, 1.0, 1.0, 1.0, 0.0, 9.0, 0.0, 3.0, 1.0, 6.0, 4.0, 2.0, 3.0, 6.0, 1.0, 1.0, 1.0, 3.0, 9.0, 5.0, 2.0, 9.0, 4.0,
5.0, 9.0, 3.0, 9.0, 0.0, 3.0, 6.0, 5.0, 5.0, 7.0, 2.0, 2.0, 7.0, 1.0, 2.0, 8.0, 4.0, 1.0, 7.0, 3.0, 3.0, 8.0, 8.0, 7.
0, 9.0, 2.0, 2.0, 4.0, 1.0, 5.0, 9.0, 8.0, 7.0, 2.0, 3.0, 0.0, 4.0, 4.0, 2.0, 4.0, 1.0, 9.0, 5.0, 7.0, 7.0, 2.0, 8.0,
2.0, 6.0, 8.0, 5.0, 7.0, 7.0, 9.0, 1.0, 8.0, 1.0, 8.0, 0.0, 3.0, 0.0, 1.0, 9.0, 9.0, 4.0, 1.0, 8.0, 2.0, 1.0, 2.0, 9.
0, 7.0, 5.0, 9.0, 2.0, 6.0, 4.0, 1.0, 5.0, 8.0, 2.0, 9.0, 2.0, 0.0, 4.0, 0.0, 0.0, 2.0, 8.0, 4.0, 7.0, 1.0, 2.0, 4.0,
0.0, 2.0, 7.0, 4.0, 3.0, 3.0, 0.0, 0.0, 3.0, 1.0, 9.0, 6.0, 5.0, 2.0, 5.0, 9.0, 2.0, 9.0, 3.0, 0.0, 4.0, 2.0, 0.0, 7.
0, 1.0, 1.0, 2.0, 1.0, 5.0, 3.0, 3.0, 9.0, 7.0, 8.0, 6.0, 5.0, 6.0, 1.0, 3.0, 8.0, 1.0, 0.0, 5.0, 1.0, 3.0, 1.0, 5.0,
5.0, 6.0, 1.0, 8.0, 5.0, 1.0, 7.0, 9.0, 4.0, 6.0, 2.0, 2.0, 5.0, 0.0, 6.0, 5.0, 6.0, 3.0, 7.0, 2.0, 0.0, 8.0, 8.0, 5.
0, 4.0, 1.0, 1.0, 4.0, 0.0, 3.0, 3.0, 7.0, 6.0, 1.0, 6.0, 2.0, 1.0, 9.0, 2.0, 8.0, 6.0, 1.0, 9.0, 5.0, 2.0, 5.0, 4.0,
4.0, 2.0, 8.0, 3.0, 8.0, 2.0, 4.0, 5.0, 0.0, 3.0, 1.0, 7.0, 7.0, 5.0, 7.0, 9.0, 7.0, 1.0, 9.0, 2.0, 1.0, 4.0, 2.0, 9.
0, 2.0, 0.0, 4.0, 9.0, 1.0, 4.0, 8.0, 1.0, 8.0, 4.0, 5.0, 9.0, 8.0, 8.0, 3.0, 7.0, 6.0, 0.0, 0.0, 3.0, 0.0, 2.0, 6.0,
6.0, 4.0, 9.0, 3.0, 3.0, 3.0, 2.0, 3.0, 9.0, 1.0, 2.0, 6.0, 8.0, 0.0, 5.0, 6.0, 6.0, 6.0, 3.0, 8.0, 8.0, 2.0, 7.0, 5.
0, 8.0, 9.0, 6.0, 1.0, 8.0, 4.0, 1.0, 2.0, 5.0, 9.0, 1.0, 9.0, 7.0, 5.0, 4.0, 0.0, 8.0, 9.0, 9.0, 1.0, 0.0, 5.0, 2.0,
3.0, 7.0, 8.0, 9.0, 4.0, 0.0, 6.0, 3.0, 9.0, 5.0, 2.0, 1.0, 3.0, 1.0, 3.0, 6.0, 5.0, 7.0, 4.0, 2.0, 2.0, 6.0, 3.0, 2.
0, 6.0, 5.0, 4.0, 8.0, 9.0, 7.0, 1.0, 3.0, 0.0, 3.0, 8.0, 3.0, 1.0, 9.0, 3.0, 4.0, 4.0, 6.0, 4.0, 2.0, 1.0, 8.0, 2.0,
5.0, 4.0, 8.0, 8.0, 4.0, 0.0, 0.0, 2.0, 3.0, 2.0, 7.0, 7.0, 0.0, 8.0, 7.0, 4.0, 4.0, 7.0, 9.0, 6.0, 9.0, 0.0, 9.0, 8.
0, 0.0, 4.0, 6.0, 0.0, 6.0, 3.0, 5.0, 4.0, 8.0, 3.0, 3.0, 9.0, 3.0, 3.0, 3.0, 7.0, 8.0, 0.0, 8.0, 2.0, 1.0, 7.0, 0.0,
6.0, 5.0, 4.0, 3.0, 8.0, 0.0, 9.0, 6.0, 3.0, 8.0, 0.0, 9.0, 9.0, 6.0, 8.0, 6.0, 8.0, 5.0, 7.0, 8.0, 6.0, 0.0, 2.0, 4.
0, 0.0, 2.0, 2.0, 3.0, 1.0, 9.0, 7.0, 5.0, 1.0, 0.0, 8.0, 4.0, 6.0, 2.0, 6.0, 7.0, 9.0, 3.0, 2.0, 9.0, 8.0, 2.0, 2.0,
9.0, 2.0, 7.0, 3.0, 5.0, 9.0, 1.0, 8.0, 0.0, 2.0, 0.0, 5.0, 2.0, 1.0, 3.0, 7.0, 6.0, 7.0, 1.0, 2.0, 5.0, 8.0, 0.0, 3.
0, 7.0, 2.0, 4.0, 0.0, 9.0, 1.0, 8.0, 6.0, 7.0, 7.0, 4.0, 3.0, 4.0, 9.0, 1.0, 9.0, 5.0, 1.0, 7.0, 3.0, 9.0, 7.0, 6.0,
9.0, 1.0, 3.0, 7.0, 8.0, 3.0, 3.0, 6.0, 7.0, 2.0, 8.0, 5.0, 8.0, 5.0, 1.0, 1.0, 4.0, 4.0, 3.0, 1.0, 0.0, 7.0, 7.0, 0.
0, 7.0, 9.0, 4.0, 4.0, 8.0, 5.0, 5.0, 4.0, 0.0, 8.0, 2.0, 1.0, 0.0, 8.0, 4.0, 5.0, 0.0, 4.0, 0.0, 6.0, 1.0, 7.0, 3.0,
```

Forming 9 columns in the train datasets with values 0/1 depending on whether the values 0 -9 are present

```
df['Target_0'] = np.where(df[0]==0, 1, 0)
df['Target_1'] = np.where(df[0]==1, 1, 0)
df['Target_2'] = np.where(df[0]==2, 1, 0)
df['Target_3'] = np.where(df[0]==3, 1, 0)
df['Target_4'] = np.where(df[0]==4, 1, 0)
df['Target_5'] = np.where(df[0]==5, 1, 0)
df['Target_6'] = np.where(df[0]==6, 1, 0)
df['Target_7'] = np.where(df[0]==7, 1, 0)
df['Target_8'] = np.where(df[0]==8, 1, 0)
df['Target_9'] = np.where(df[0]==9, 1, 0)
```

Importing the libraries to carry out logistic regression.

```
from sklearn.datasets import make_classification
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
```

There will be 10 models, each predicting separately whether the target digit is 0-9. To carry out this there are 10 training sets suitably formed.

```
x_train_0= df.drop([0,'Target_0','Target_1', 'Target_2', 'Target_3', 'Target_4', 'Target_5', 'Target_6', 'Target_7', '
Target_8', 'Target_9'], axis=1)
x_train_1= df.drop([0,'Target_0','Target_1', 'Target_2', 'Target_3', 'Target_4', 'Target_5', 'Target_6', 'Target_7', '
Target_8', 'Target_9'], axis=1)
x_train_2= df.drop([0,'Target_1', 'Target_2','Target_0', 'Target_3', 'Target_4', 'Target_5', 'Target_6', 'Target_7', '
Target_8', 'Target_9'], axis=1)
x_train_3= df.drop([0,'Target_1', 'Target_3','Target_2', 'Target_0', 'Target_4', 'Target_5', 'Target_6', 'Target_7', '
Target_8', 'Target_9'], axis=1)

x_train_4= df.drop([0,'Target_1', 'Target_4','Target_2', 'Target_3', 'Target_0', 'Target_5', 'Target_6', 'Target_7', '
Target_8', 'Target_9'], axis=1)
x_train_5= df.drop([0,'Target_1', 'Target_5','Target_2', 'Target_3', 'Target_4', 'Target_0', 'Target_6', 'Target_7', '
Target_8', 'Target_9'], axis=1)
x_train_6= df.drop([0,'Target_1', 'Target_6','Target_2', 'Target_3', 'Target_4', 'Target_5', 'Target_0', 'Target_7', '
Target_8', 'Target_9'], axis=1)
x_train_7= df.drop([0,'Target_1', 'Target_7','Target_2', 'Target_3', 'Target_4', 'Target_5', 'Target_6', 'Target_0', '
Target_8', 'Target_9'], axis=1)
x_train_8= df.drop([0,'Target_1', 'Target_8','Target_2', 'Target_3', 'Target_4', 'Target_5', 'Target_6', 'Target_7', '
Target_0', 'Target_9'], axis=1)
x_train_9= df.drop([0,'Target_1', 'Target_9','Target_2', 'Target_3', 'Target_4', 'Target_5', 'Target_6', 'Target_7', '
Target_8', 'Target_0'], axis=1)
```

The target value of each train dataset is from Target_0 – Target_9 values

```
Y_0 = df["Target_0"].tolist()
Y_1 = df["Target_1"].tolist()
Y_2 = df["Target_2"].tolist()
Y_3 = df["Target_3"].tolist()
Y_4 = df["Target_4"].tolist()
Y_5 = df["Target_5"].tolist()
Y_6 = df["Target_6"].tolist()
Y_7 = df["Target_7"].tolist()
Y_8 = df["Target_8"].tolist()
Y_9 = df["Target_9"].tolist()
```

Converting the train datasets to list

```
X_0 = x_train_0.values.tolist()
X_1 = x_train_1.values.tolist()
X_2 = x_train_2.values.tolist()
X_3 = x_train_3.values.tolist()
X_4 = x_train_4.values.tolist()
X_5 = x_train_5.values.tolist()
X_6 = x_train_6.values.tolist()
X_7 = x_train_7.values.tolist()
X_8 = x_train_8.values.tolist()
X_9 = x_train_9.values.tolist()
```

Forming the logistic models and fitting it to the respective 10 train datasets. This took approximated 30 mins to run successfully.

```
lr_0 = LogisticRegression()
lr_1 = LogisticRegression()
lr_2 = LogisticRegression()
lr_3 = LogisticRegression()
lr_4 = LogisticRegression()
lr_5 = LogisticRegression()
lr_6 = LogisticRegression()
lr_7 = LogisticRegression()
lr_8 = LogisticRegression()
lr_9 = LogisticRegression()

lr_0.fit(X_0, Y_0)
lr_1.fit(X_1, Y_1)
lr_2.fit(X_2, Y_2)
lr_3.fit(X_3, Y_3)
lr_4.fit(X_4, Y_4)
lr_5.fit(X_5, Y_5)
lr_6.fit(X_6, Y_6)
lr_7.fit(X_7, Y_7)
lr_8.fit(X_8, Y_8)
lr_9.fit(X_9, Y_9)
```

To test the train model, the output and type of the test data was checked as below

```
df_test_0
```

	1	2	3	4	5	6	7	8	9	10	...	775	776	777	778	779	780	781	782	783	784
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10 rows × 784 columns

The final target variables of the test data was saved as a list in the variable ‘df_test_final’

```
df_test_final= df_test.drop([0], axis=1)
#df_test_final
```

The trained datasets were predicted against each of the models. The output is a list of lists indicating the probability that the value will be wither 0 or 1. The first value is the prediction for 0, while the other value is the prediction for 1.

```
y_pred_0 = lr_0.predict_proba(df_test_final)
y_pred_1 = lr_1.predict_proba(df_test_final)
y_pred_2 = lr_2.predict_proba(df_test_final)
y_pred_3 = lr_3.predict_proba(df_test_final)

y_pred_4 = lr_4.predict_proba(df_test_final)
y_pred_5 = lr_5.predict_proba(df_test_final)
y_pred_6 = lr_6.predict_proba(df_test_final)
y_pred_7 = lr_7.predict_proba(df_test_final)

y_pred_8 = lr_8.predict_proba(df_test_final)
y_pred_9 = lr_9.predict_proba(df_test_final)
```

```
print(y_pred_0)
```

```
[[1.00000000e+00 8.24615623e-19]
 [9.99999991e-01 8.56466181e-09]
 [9.99999686e-01 3.13692034e-07]
 ...
 [9.99999999e-01 1.24319875e-09]
 [1.00000000e+00 1.94722107e-18]
 [1.00000000e+00 1.32017097e-15]]
```

Independently checked for the probability of the input to be either of the 10 digits is true

```
lst0 = [x[1] for x in y_pred_0]
lst1 = [x[1] for x in y_pred_1]
lst2 = [x[1] for x in y_pred_2]
lst3 = [x[1] for x in y_pred_3]
lst4 = [x[1] for x in y_pred_4]

lst5 = [x[1] for x in y_pred_5]
lst6 = [x[1] for x in y_pred_6]
lst7 = [x[1] for x in y_pred_7]
lst8 = [x[1] for x in y_pred_8]
lst9 = [x[1] for x in y_pred_9]
```

```
print(lst0)
```

```
[8.2461562349225195e-19, 8.564661813351862e-09, 3.136920343941586e-07, 0.999582698813646, 0.0011238351289612222, 1.54
650664964961309 65e-07, 4.116105463160546e-06, 1.0654777868870154e-14, 0.0005786219878589414, 1.5198301016005015e-16, 0.9
917310176203888, 0.00017988175443033012, 5.069652819871545e-14, 0.9994468953144159, 5.34617244802962e-10, 0.000155547
40145354258, 4.8820773682721655e-06, 4.81272093383758e-15, 8.6083441895432 87e-07, 0.00019201743183998538, 2.269801164
5820567e-06, 1.2213953358369974e-05, 3.5756064028884047e-10, 9.02006551200632e-05, 0.0015004861750545866, 0.999989512
6770548, 2.1045882440003278e-30, 5.412741187485536e-05, 0.9999862403679618, 2.6384335240689026e-09, 5.715083358495957
e-09, 6.743204313489364e-08, 2.5101516369908304e-07, 0.17412084691209162, 8.84644368198457e-07, 0.0001496268428690303
8, 2.4736016265044707e-13, 9.618283689589506e-10, 0.0052223575137874375, 3.935884028672456e-08, 1.1020686243687866e-0
6, 2.288320920647428e-10, 1.1606646980615745e-09, 0.00024156526021655076, 2.975409822401017e-07, 0.000374144938928741
75, 1.0182370440733717e-11, 7.795955061251259e-21, 1.2348785146287174e-09, 3.348952922654613e-05, 0.01276370193372732
3, 4.8512400296900566e-05, 0.00018052575411527298, 0.0026013301993428137, 4.1357684290409126e-07, 0.9981899562747025,
3.8215371529788955e-05, 2.4156848692309075e-08, 1.4576158221923864e-13, 0.01278502027505293, 3.6058636819793376e-38,
5.0420700603690014e-05, 5.648300572591114e-07, 2.086224216250154e-05, 5.608330552715699e-16, 2.373377082008532e-05, 3
.985233887134516e-08, 0.003010372163320679, 1.1863513885474305e-07, 0.997689378431164, 6.161136377312005e-08, 0.99997
89722168804, 0.0017821181052675849, 7.324897371625239e-08, 4.099755310391715e-09, 2.0053651029258867e-16, 5.288124579
464603e-05, 0.00011224587168809534, 6.2105873344937 98e-09, 1.3048789448713995e-18, 1.1409865241426873e-19, 0.00166211
70180532276, 0.0001425088251053813, 5.1443716848589385e-27, 2.8531985085120 5e-05, 2.4132959825913356e-06, 2.793576288
```

Implemented the SoftMax function to normalize the probability distribution

```
df_final_1['sum'] = df_final_1.sum(axis=1)
softmax = df_final_1.loc[:,'lst0':'lst9'].div(df_final_1["sum"], axis=0)
#softmax
#Dataframe with maximum values
#df_final_3 = df_final_1.idxmax(axis=1)
```

The values represent the chances of the input being the particular digit. The larger the value, higher is the probability of it being the digit

```
print(softmax)
```

```
           lst0          lst1          lst2          lst3          lst4  \
0  9.785937e-07  1.212587e-28  4.370989e-08  4.772061e-09  4.668999e-14
1  8.739429e-07  1.080124e-22  4.693976e-01  3.911194e-06  1.641823e-28
2  4.983605e-12  9.999886e-01  6.999690e-07  6.458759e-06  3.134099e-08
3  9.999822e-01  2.691772e-31  4.305052e-16  3.423074e-31  3.268781e-24
4  3.815462e-12  2.690244e-16  3.430649e-08  1.384743e-17  9.633661e-01
5  1.087604e-14  9.999423e-01  1.260796e-07  2.974237e-09  6.451926e-11
6  9.328616e-14  1.347362e-14  1.441833e-25  3.969322e-10  9.999573e-01
7  2.354201e-19  3.583002e-02  5.062560e-17  2.194537e-04  6.219448e-09
8  1.432200e-06  1.352592e-01  8.573988e-01  1.140367e-30  7.339950e-03
9  1.675448e-10  7.914207e-24  7.727860e-24  1.307885e-23  1.737007e-07

           lst5          lst6          lst7          lst8          lst9
0  7.275548e-17  1.457133e-23  9.999990e-01  8.416781e-11  2.134391e-13
1  7.123208e-09  5.305976e-01  1.609018e-35  3.563909e-18  1.173936e-12
2  7.519593e-12  2.054505e-06  1.909710e-06  7.761346e-08  1.318132e-07
3  1.516496e-10  5.209207e-11  1.783050e-05  5.732940e-22  1.930725e-24
4  1.940200e-26  1.257995e-06  6.634947e-10  3.064076e-04  3.632617e-02
5  1.509865e-14  1.063983e-11  5.760209e-05  3.781541e-09  1.358081e-08
6  4.079876e-05  8.240710e-14  7.223972e-19  5.303258e-08  1.882297e-06
7  2.360177e-08  1.707913e-07  1.479123e-08  4.468690e-13  9.639503e-01
8  1.233466e-10  6.228275e-07  1.422910e-29  6.136357e-18  2.999790e-17
9  2.555539e-17  2.861477e-11  6.196717e-02  3.753786e-10  9.380327e-01
```

```
df_final_3 = softmax.idxmax(axis=1)
```

```
type(df_final_3)
```

```
pandas.core.series.Series
```

Replaced the column names to their corresponding digits in numerical format

```
df_final_3 = df_final_3.replace(to_replace = ['lst0', 'lst1','lst2', 'lst3','lst4', 'lst5','lst6', 'lst7','lst8', 'lst9',], value = [0,1,2,3,4,5,6,7,8,9])
```

```
df_final_3
```

```
0     7
1     2
2     1
3     0
4     4
5     1
6     4
7     9
8     6
9     9
10    0
11    6
12    9
13    0
14    1
15    5
16    9
17    7
18    3
19    4
20    9
21    6
```

Imported the libraries to find the confusion matrix, accuracy and classification report

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
results = confusion_matrix(df_test[0], df_final_3)

print ('Confusion Matrix :')
print(results)

print ('Accuracy Score :',accuracy_score(df_test[0], df_final_3) )
print ('Report : ')
print (classification_report(df_test[0], df_final_3))
```

The model has an accuracy of 0.9175. The maximum number of misclassifications is for the digit 5. While the minimum number of misclassifications is for the digit 0.

```
Confusion Matrix :
[[ 960    0    2    1    0    3    7    3    3    1]
 [   0 1108    4    3    0    1    5    1   13    0]
 [  10   10  904   20    7    3   12   15   48    3]
 [   4    0   17  918    2   23    5   10   23    8]
 [   1    2    5    5  911    0    9    3   10   36]
 [   9    1    0   39   10  767   16    7   35    8]
 [   8    4    8    0    5   20  907    0    6    0]
 [   4    8   25    5    7    2    1  941    4   31]
 [  14   13    6   20   12   20   10   11  858   10]
 [   9    7    2   14   29    8    0   27   12  901]]
Accuracy Score : 0.9175
Report :
              precision    recall  f1-score   support

         0.0       0.94      0.98      0.96       980
         1.0       0.96      0.98      0.97      1135
         2.0       0.93      0.88      0.90      1032
         3.0       0.90      0.91      0.90      1010
         4.0       0.93      0.93      0.93       982
         5.0       0.91      0.86      0.88       892
         6.0       0.93      0.95      0.94       958
         7.0       0.92      0.92      0.92      1028
         8.0       0.85      0.88      0.86       974
         9.0       0.90      0.89      0.90      1009

   micro avg       0.92      0.92      0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

References:

*Berd Klein (2011 - 2019) , Retrieved from Python machine learning Tutorial: https://www.python-course.eu/neural_network_mnist.php

*2019, Retrieved from Applied AI Course: https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/3011/geometric-intuition-of-logistic-regression/3/module-3-foundations-of-natural-language-processing-and-machine-learning

*Vibhu Singh (Jan 24, 2018) , Retrieved from Introduction to K-NN: https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwj7yfST0OXlAhVvhOAKHaNbDI0QMwhGKAIwAg&url=http%3A%2F%2Fstrijov.com%2Fsources%2FdemoDataGen.php&psig=AOvVaw2y8-sTxud9a5-4GVf9kwHT&ust=1573680855153767&ictx=3&uact=3