Sanket Nitin Wagh – A20330391

Problem Statement:

This assignment is based on discriminative learning. This is going to be done with the use of Logistic as well as Neural Network algorithms. Experiments performed with different datasets and various parameters. From which we can decipher the correctness of the algorithms that will be implemented. Results are deduced and/or analyzed by looking at the confusion matrices and the impending accuracies.

Proposed Solution:

All the functions are user-defined. These mainly include defining the accuracy, fmeasure scores and confusion matrix. These functions are there along with the functions that define the core of the algorithms that are going to be implemented.

Implementation Details:

Logistic Regression:

1. Two class: The flow of this code is basically importing all the appropriate packages and loading the datasets to be used. Two classes are used in this case. Once thedatset is correctly loaded we calculate the Yhat value. This is done using the Ycap function. This calculates the sigmoid function.
   Using this we perform the decent. We set the learning rate and determine the theta values. Once the hypothesis is generated we use it to determine the new theta value. Technically we implement the two class logistic regression algorithm in this step.
   Once this is complete we go ahead and classify. Feeding the classification results to the confusion matrix function to determine the accuracy of the classification. Test-Split as well as KFold cross validation is performed.
2. Non-linear: The bulk of the problem is solved in the same way as (1), but we change the polynomial and see the difference in results.

3. K-Class: The logistic regression problem is solved in a slightly different way. We define a softmax function to be used in the gradient decent phase. Also, with this we have a IID function defined, which is set to return 1 only when y[i] = j. The softmax function is going to determine the theta values for us. There will be three theta values, one for each class.
4. Evaluating performance: This is done mainly by looking at the confusion matrix and determining accuracy. Precision, Recall and Fmeasure scores are also calculated in some cases as a secondary measurement.

Neural Networks:

We start of by adding a column filled with ones to the set of features. Followed by which we define functions to determine hidden as well as output layer weights. We find Z using the hidden layer weights. We determine yhat and use this in the decent. IID function will also be created as mentioned before and will be used in the decent. Once completed we classify and determine the confusion matrix for which we evaluate by getting the accuracy.

Results and Discussion:

Logistic Regression

1. Two class:

Folds	Iris	Banknote	MNIST
1	100	95.65	99.33
2	100	97.10	99.67
3	100	94.89	99.67
4	100	95.62	99.33
5	100	94.16	98.67
6	100	94.89	99.67
7	100	96.35	99.67
8	100	98.54	100
9	100	93.43	100
10	100	93.43	98.33

From this we can say that there is a 100% accuracy while using the iris dataset. There is a more realistic outcome using banknote dataset giving an accuracy of

95.406% and MNIST data produces an average accuracy of 99.434%. The classification results are successful and provided in the ipynb notebook given along with this report.

Here are the confusion matrices for 6th fold:

```
Iris: [[5, 0], [0, 5]]
Banknote: [[76, 0], [7, 54]]
MNIST: [[299, 1], [0, 0]] (Using 3000 records of the entire dataset)
```

2. To make the data non-linear we transform the data by defining a polynomial function of the power 2.

We predict that the accuracy must increase as we make it non-linear. Since iris was giving a 100% accuracy anyway I tried using Banknote dataset.

Confusion matrix and accuracy before applying non-linear combinations of Input:

```
[762, 0]
[61, 549]

Accuracy: 0.955539358601
```

Confusion matrix and accuracy after applying non-linear combinations of Input:

```
[241, 1]
[12, 199]

Accuracy: 0.971302428256
```

As we can see. The prediction is better and the classifier is working more efficiently giving a better result. This was expected and now proven.

3. KClass:

   Softmax function works effectively.

   The values returned by the softmax function while testing are:

```
s = softmax_fn(X[1],np.random.rand(3,4)/10000)
print s

Test 1 Outcome:
[0.33331804785885494, 0.33340181398684865, 0.33328013815429652]
Test 2 Outcome:
[0.33333707821384012, 0.33329809828368967, 0.33336482350247015]
```

We feed the theta value which are generated at random and are having a small value.

The decent function also works effectively. Here is an excerpt from the code:

```
softie = decent(Y,X)

print softie

Test 1 outcome:

[[ 0.00327057  0.0194913  -0.04041838 -0.01878075]
 [-0.87316187 -0.44547662 -0.58852736 -0.19172862]
 [ 0.87001208  0.42615288  0.62908095  0.21068209]]

Test 2 Outcome:

[[  4.93205376e-04   7.20545554e-04  -5.88284026e-04  -3.37878296e-04]
 [  8.81776465e-01   4.92397238e-01   4.90950770e-01   1.45731607e-01]
 [ -8.82092558e-01  -4.92973936e-01  -4.90173270e-01  -1.45335675e-01]]
```

Thus confirming that the decent performs correctly producing a 3x4 matrix. This is for 3 classes and 4 features respectively.

Classifier, although is functioning correctly gives mediocre results. This result can be obtained only via trial and error method. By setting the learning rate and initial theta values effectively. The best accuracy that was obtained was 40% since two classes were successfully being classified. 1 feature of the second class was classified to be exact. The classification process using 10 fold cross validation is as follows (For Iris-Setosa vs 3 classes):

```
[[0, 15, 0], [0, 0, 0], [0, 0, 0]]
0.0
[[0, 15, 0], [0, 0, 0], [0, 0, 0]]
0.0
[[15, 0, 0], [0, 0, 0], [0, 0, 0]]
1.0
[[0, 5, 0], [0, 10, 0], [0, 0, 0]]
0.666666666667
[[0, 0, 0], [0, 0, 15], [0, 0, 0]]
0.0
[[0, 0, 0], [0, 0, 15], [0, 0, 0]]
0.0
[[0, 0, 0], [0, 10, 0], [0, 5, 0]]
```

```
0.666666666667
[[0, 0, 0], [0, 0, 0], [0, 0, 15]]
1.0
[[0, 0, 0], [0, 0, 0], [0, 15, 0]]
0.0
[[0, 0, 0], [0, 0, 0], [0, 0, 15]]
1.0
```

From this we can say that the average accuracy is: 43.334%
Test split gave 38% accuracy. To emphasize the correct classification can be obtained once there is an effective learning rate and theta value chosen. Hence although classifier functioned properly the accuracy obtained was bad.

Neural Networks:

Implementation:

	Class 0 (Setosa)	Class 1(Versicolor)	Class2(Virginica)
Class1(Versicolor)	19	15	16

As we can see similar problems persist as in Logistic K Class Regression. Hence a bad accuracy of 30% was obtained for 3 class classification. Implementation was successful.

2) a)

Given errorfunction: $E = \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$ — (1)

⇒ Here $y^{(i)} \to$ desired o/p of vector
$\hat{y}$ is the actual o/p

⇒ We must minimize the given Errorfunction.
This is done with respect to hidden layer (w) & output layer. (V)

~~for~~ minimizing (1) (similar to the way done in class)
⇒ we need to find the gradient for output layer parameters.

$$\Rightarrow \frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial V}$$

$$= \sum_i (\hat{y} - y) \cdot \hat{y} \cdot (1 - \hat{y}) \cdot z^{(i)}$$

$$\boxed{\Rightarrow V = V - \eta \sum_i (\hat{y} - y) \cdot \hat{y} \cdot (1 - \hat{y}) z^{(i)}}$$

⇒ we also need to find gradient of hidden layer parameters

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial g} \cdot \frac{\partial g}{\partial z} \cdot \frac{\partial z}{\partial w}$$

$$= \sum_i (\hat{y} - y) * \hat{y} * (1 - \hat{y}) \cdot V * z(1-z) * x^{(i)}$$

$$\boxed{\Rightarrow w = w - \eta \sum_i (\hat{y} - y) \cdot \hat{y} \cdot (1 - \hat{y}) V \cdot z(1-z) \cdot x^{(i)}}$$

(2) ⇒ The main difference between the outputs obtained here is that there is sigmoid activation on output in this case.
The algorithm in general remains the same & procedure stops when there is convergence.