INTRODUCTION TO MACHINE LEARNING
PROJECT-3 REPORT

Prof. Dr. Srihari

GROUP MEMBERS

NITISH SHOKEEN (UB Person Number: 50247681)

MAHALAKSHMI PADMA SRI HARSHA MADDU (UB Person Number: 50246769)

CHARANYA SUDHARSANAN (UB person Number: 50245956)

Problem Statement: To implement and evaluate classification algorithms – Logistic Regression, Single hidden layer neural network , convolutional neural network . This is done on a task of recognizing a 28 X 28 grayscale handwritten digit image and identify it as a digit among 0 – 9.

Part1: Logistic Regression.

Introduction: We will use logistic regression on Tensorflow to classify the input, specifically if given an input image, we are going to classify it as one of the digits of 0–9.

Implementing logistic regression for MNIST dataset.

Our Input Dataset is MNIST Dataset and we are training our model with it. Then we test it on MNIST test dataset as well as the USPS dataset. For testing we have used the images in the folder Test.

Import the MNIST dataset using tensorflow. Every image in the input consists of 784 pixels. Initialize weights and biases with random values between 0 and 1. Find the Output of the model We use a cost function, Cross Entropy which involves three steps.

1. Convert actual image class vector into a one-hot vector, which is a probability distribution.
2. Convert prediction class vector into a probability distribution.
3. Use cross entropy function to calculate cost, which is the difference between 2 probability distribution function.

We train the model on MNIST training set and test the trained model to get an accuracy. Based on the accuracy we do the hyper parameter tuning to get the maximum accuracy. Finally, we test it on the USPS Dataset to get the accuracy on untrained data.

Test Accuracy on MNIST Test dataset and USPS Dataset before hyper-parameter tuning:

Screenshot:

```
In [1]: runfile('/Users/nitishshokeen/Desktop/Courses/CSE574/ML Proj 3/
Q1LogisticUSPSandMNIST.py', wdir='/Users/nitishshokeen/Desktop/Courses/CSE574/ML Proj 3')
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
MNIST Test Accuracy:
0.92
/Users/nitishshokeen/anaconda/lib/python3.5/site-packages/skimage/transform/_warps.py:84:
UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
USPS Test Accuracy:
0.0786143
```

We perform hyper-parameter tuning on the epoch and the batch size.

Hyper-parameter tuning:

Keeping Epoch size as constant (100):

BATCH SIZE	20	40	60	80	100
MNIST	0.8623	0.87	0.89	0.88	0.829
USPS	0.301	0.301	0.3071	0.302	0.3144

Keeping Batch Size as constant (100):

ECHO	200	400	600	800	1000
MNIST	0.9	0.911	0.908	0.91	0.917
USPS	0.31	0.33	0.3377	0.33977	0.3324

Keeping epoch size as 800 and Batch size as 100 which is optimal, we get the optimal test accuracy on USPS Test set and MNIST test set

Test Accuracy on MNIST Test dataset and USPS Dataset after hyper-parameter tuning:

```
In [11]: runfile('/Users/mahalakshmimaddu/untitled16.py', wdir='/Users/mahalakshmimaddu')
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
MNIST Test Accuracy:
0.9173
/Users/mahalakshmimaddu/anaconda/lib/python3.6/site-packages/skimage/transform/_warps.py:84: UserWarning: The default mode,
'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
USPS Test Accuracy:
0.339773
```

Accuracy: MNIST Test: 91.7%
USPS Test: 33.97%

Part2: Single Hidden Later Neural Network.

Introduction:

Single Hidden Layer Neural Network takes several input, here, from MNIST data set and processes it through multiple neurons of a single hidden layer and returns the result using an output layer. This estimation process is known as forward propagation. The resultant value is compared with the actual output. We have to make the output of the neural network as close as possible to the desired output. Since each of the neurons are contributing some error to the final output, we travel back to the neurons of the neural network and find where the error lies. This is called backward propagation. We use the gradient descent optimizer function to optimize the task and

reduce the number of iterations needed to minimize the error. This one round of forward propagation and back =ward propagation I called 'EPOCH'.

Implementation:

- Imported MNIST data set using tensor flow.
- Initialize the output classes as 10 as we have 0to 9 (10) digits and create a hidden layer with a constant number of neurons, here we take 128 neurons after parameter tuning. Every image in the input consists of 784 pixels. Using these variables, we create a perceptron.
- Initialize weights and biases with random values between 0 and 1. This is followed by calculating the hidden layer input. This involves the matrix dot product of input and weight and adding the product to the random bias.
- Performed non-linear transformation on hidden layer input using an activation function – sigmoid from tensor flow. This output is passed to the softmax regression function using tensorflow.
- We find the error by using cross entropy formula and minimize it using a mean function. We finally minimize the error using gradient descent optimizer.

Results:

Test Accuracy on MNIST Test dataset and USPS Test and Numeral Folder after Hyper-parameter Tuning:

After hyper-parameter tuning, we take the optimum values of batch size and EPOCH as 20 and 15 respectively.

```
Users/nitishshokeen/Desktop/Courses/CSE574/ML Proj 3')
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
0.9782
/Users/nitishshokeen/anaconda/lib/python3.5/site-packages/skimage/transform/_warps.py:84:
UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
oneHotEncoding
0.0819454
oneHotEncoding
0.48425
```

Test Accuracy: MNIST Test: 97.82 %
USPS Test: 8.19 %
USPS Numeral: 48.425 %

Part-3: Convolutional Neural Network Classification.

Convolutional Neural Networks (CNN) are biologically-inspired variants of MLPs. CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. In other words, the inputs of hidden units in layer **m** are from a subset of units in layer **m-1**, units that have spatially contiguous receptive fields.

In addition, in CNNs, each filter h_i is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a *feature map*.

Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt. The constraints on the model enable CNNs to achieve better generalization on vision problems.

A feature map is obtained by repeated application of a function across sub-regions of the entire image, in other words, by *convolution* of the input image with a linear filter, adding a bias term and then applying a non-linear function.

A CNN is composed of a stack of convolutional modules that perform feature extraction. Each module consists of a convolutional layer followed by a pooling layer. The last convolutional module is followed by one or more dense layers that perform classification.

Implementation:

- Imported MNIST data set using tensor flow.
- Initialize weights and biases with random zero values.
- We create a CNN Architecture with three layers: Convolutional, Pooling and Dense .
- Here we pass the data through two layers of convolutional and pooling and then through two layers of Dense.
- Convolution Layer applied a specified number of filters to the image. We apply an activation function called ReLu to the output to introduce the nonlinearities into the model.
- Pooling layer down sample the image data extracted by convolutional layer to reduce the dimensionality to give a better processing time.
- Each module consists of a convolutional and pooling layer. After two modules we pass the output to the dense layer 1.
- Dense layer performs the classification. We use one dense layer with 1024 neurons and a final dense layer with 10 neurons as the number of target values is 10 for us.
- We calculate the loss and optimize the loss using gradient descent optimizer and finally predict the output.

Result Screenshot:

Test Accuracy on MNIST Test dataset and USPS Test and Numeral Folder after Hyper-parameter Tuning:

```
wdir='/Users/nitishshokeen/Desktop/Courses/CSE574/ML Proj 3')
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
MNIST Test Accuracy:
0.9919
/Users/nitishshokeen/anaconda/lib/python3.5/site-packages/skimage/transform/_warps.py:84:
UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
oneHotEncoding
USPS Test Accuracy:
0.08994
```

Test Accuracy: MNIST Test: 99.19%
USPS Test: 8.994 %

Conclusion:

No free lunch theorem states that even after the observation of the frequent or constant conjunction of objects, we have no reason to draw any inference concerning any object beyond those of which we have had experience

No Free Lunch theorem is supported here as we can see that no algorithm is universally good.
there is no one single model that works best for every problem.
In supervised learning algorithms, each of the algorithms need to be evaluated for trade-offs between speed, accuracy, complexity, efficiency and find one which works best for a particular problem. We can see how different algorithms perform on the MNIST and USPS dataset and conclude the same.

Plots for Logistic Regression: