Satvinder Singh Panesar

UBITName: satvinde Person No: 50248888

Project 3

Implement and evaluate classification algorithms

1. **Extracting feature values and labels from the data:**

Corresponding Python Functions:
sklearn.datasets.fetch_mldata() : Fetch an mldata.org data set
input_data.read_data_sets() : download data to local training folder
Image.open(): open an image
Image.resize(): returns resized copy of image
Numpy.ravel(): return a flattened array

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
mnist.data[mnist.data>0]=1
```

Fig: Reading features and labels of MNIST Data (python code)

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Fig: Reading features and labels of MNIST Data using Tensorflow (python code)

```
def resize_usps_images():
  for i in range(0,10):
    for filename in g.glob(os.path.join("proj3_images/Numerals/"+str(i), '*.png')):
      img=Image.open(filename.replace("\\","/"))
      img=img.resize((28,28), Image.NEAREST)
      img.save(filename.replace("Numerals","Numerals_Resized”))
```

Fig: Reading, resizing and saving USPS images (python code)

```
def read_resized_usps_images(usps_test_data,usps_test_1d_label,usps_test_2d_label):
  for i in range(0,10):
    for filename in g.glob(os.path.join("proj3_images/Numerals_Resized/"+str(i), '*.png')):
      temp=[]
      img=misc.imread(filename.replace("\\","/"),flatten=1)
      img[img==0]=1
      img[img==255]=0
      img=np.ravel(img)
      usps_test_data.append(img)
      if i==0:
        usps_test_2d_label.append([1,0,0,0,0,0,0,0,0,0])
```

```
            usps_test_1d_label.append(0)
        elif i==1:
            usps_test_2d_label.append([0,1,0,0,0,0,0,0,0,0])
            usps_test_1d_label.append(1)
        elif i==2:
            usps_test_2d_label.append([0,0,1,0,0,0,0,0,0,0])
            usps_test_1d_label.append(2)
        elif i==3:
            usps_test_2d_label.append([0,0,0,1,0,0,0,0,0,0])
            usps_test_1d_label.append(3)
        elif i==4:
            usps_test_2d_label.append([0,0,0,0,1,0,0,0,0,0])
            usps_test_1d_label.append(4)
        elif i==5:
            usps_test_2d_label.append([0,0,0,0,0,1,0,0,0,0])
            usps_test_1d_label.append(5)
        elif i==6:
            usps_test_2d_label.append([0,0,0,0,0,0,1,0,0,0])
            usps_test_1d_label.append(6)
        elif i==7:
            usps_test_2d_label.append([0,0,0,0,0,0,0,1,0,0])
            usps_test_1d_label.append(7)
        elif i==8:
            usps_test_2d_label.append([0,0,0,0,0,0,0,0,1,0])
            usps_test_1d_label.append(8)
        elif i==9:
            usps_test_2d_label.append([0,0,0,0,0,0,0,0,0,1])
            usps_test_1d_label.append(9)
usps_test_data=np.array(usps_test_data)
usps_test_2d_label=np.array(usps_test_2d_label)
usps_test_1d_label=np.array(usps_test_1d_label)
```

Fig: Reading resized USPS images and generating numpy arrays (python code)

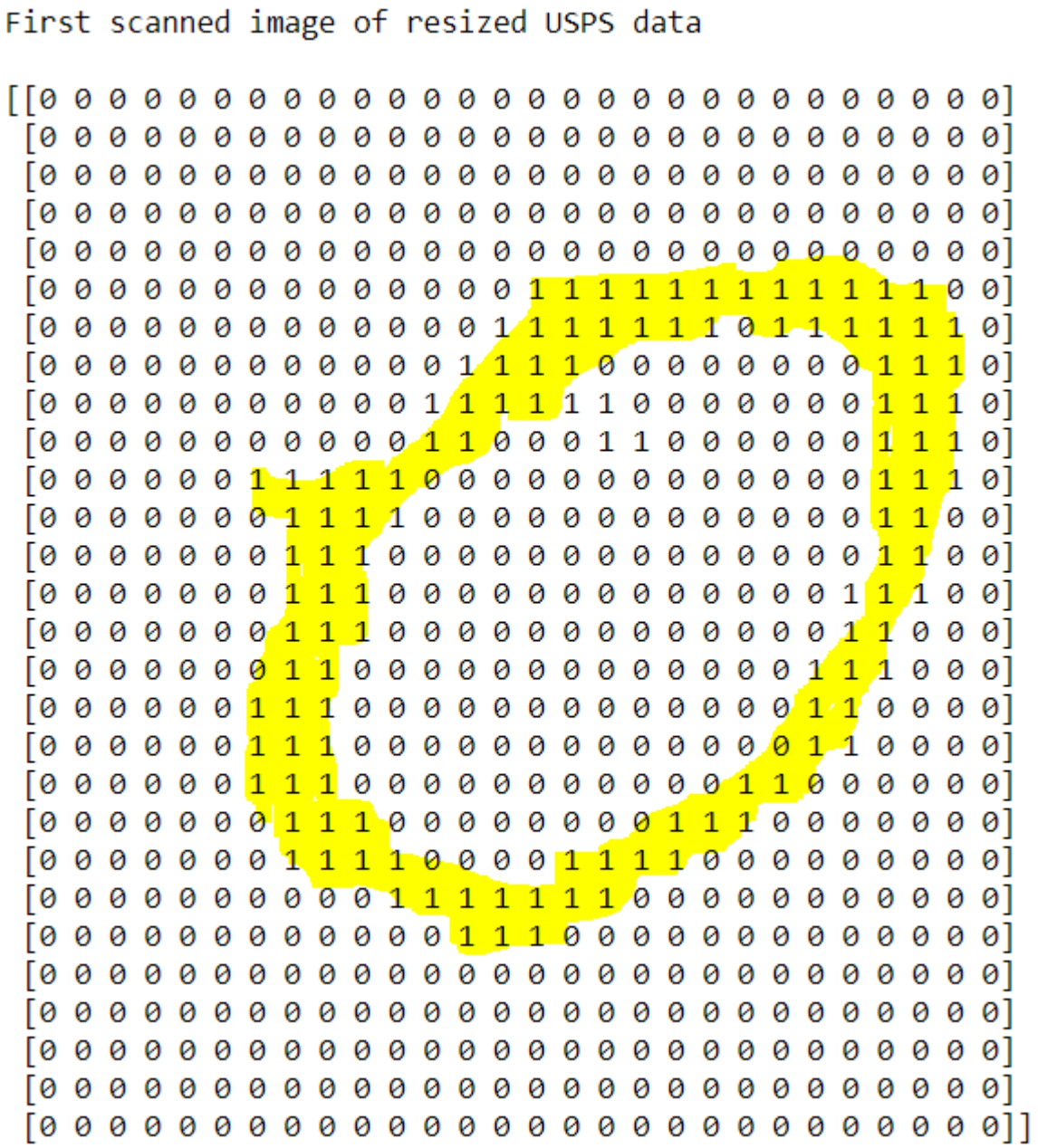

Fig: Matrix representation (28X28) of USPS number 0 as stored in numpy array

2. Data Partitioning into sets:

Corresponding Python Functions:
array_name[start_index:end_index] : return array partition specified by start and end index

```
mnist_train_data=mnist.data[0:int(len(mnist.data)*0.8)]
mnist_train_target=mnist.target[0:int(len(mnist.target)*0.8)]
mnist_test_data=mnist.data[int(len(mnist.data)*0.8):int(len(mnist.data))]
mnist_test_target=mnist.target[int(len(mnist.target)*0.8):int(len(mnist.target))]
```

Fig: Partitioning MNIST dataset (python code)

***Note: For Tensorflow, data is already partitioned**.*

3. Training model parameter:

3.1 Logistic regression

Corresponding Python Functions:
LogisticRegression. fit (): fit the model according to the given training data
lr.intercept_: get the bias added to decision function

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(fit_intercept=True)
lr.fit(mnist_train_data, mnist_train_target)
```

Fig: Fitting logistic regression model for MNIST dataset (python code)

3.2 Single Layer Neural Network

Corresponding Python Functions:
tf.placeholder(): inserts a placeholder for a tensor that will be always fed
tf.reduce_mean(): computes the mean of elements across dimensions of a tensor
tf.train(): support for training model
tf.InteractiveSession(): a TensorFlow Session for use in interactive contexts, such as a shell
sess.run(): execute session

```
y_ = tf.placeholder(tf.float32, [None, 10])
#Formula for cross_entropy
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(cross_entropy)
#Initialize session
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
#Updating weights batch wise
for _ in range(500):
  batch_xs, batch_ys = mnist.train.next_batch(50)
  sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
#Correct formula
correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y_,1))
#Accuracy formula
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
```

Fig: Single Layer Neural Network for MNIST dataset (python code)

3.3 Convolutional Neural Networks using Tensorflow

Corresponding Python Functions:
tf.truncated_normal(): outputs random values from a truncated normal distribution
tf.constant(): creates a constant tensor
tf.nn.conv2d(): computes a 2-D convolution given 4-D input and filter tensors
tf.nn.max_pool(): performs the max pooling on the input

tf.reshape(): reshapes a tensor
tf.nn.relu(): performs rectified linear
tf.nn.dropout(): computes dropout
tf.matmul(): performs matrix multiplication

```
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME')

W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1, 28, 28, 1])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Fig: Convolutional Neural Networks using Tensorflow for MNIST dataset (python code)

4. Tuning hyper parameters (explained in detail in following sections)

```
def get_bias_function(classifier):
   return np.around(classifier.intercept_,decimals=2)
```

Fig: Bias function for logistic regression (python code)

```
def get_accuracy(classifier,data,target,dataset):
   print("Accuracy with "+dataset+" Dataset: ",classifier.score(data, target))

#for logistic regression
get_accuracy(lr,mnist_test_data,mnist_test_target,"MNIST")

#for single layer convolution network
print("Accuracy with MNIST Dataset: ",sess.run(accuracy, feed_dict={x:mnist.test.images, y_: mnist.test.labels}))

#for multi-layer convolutional network
print('Accuracy with MNIST Dataset: %g' % accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels,
keep_prob: 1.0}))
```

Fig: Getting accuracy for each hyper parameter (python code)

5. Testing model using tuned parameters

Predicting classes for USPS data and evaluating performance using accuracy

```
#for logistic regression
get_accuracy(lr,usps_test_data,usps_test_1d_label,"USPS")

#for single layer convolution network
print("Accuracy with USPS Dataset: ",sess.run(accuracy, feed_dict={x:usps_test_data, y_: usps_test_2d_label}))

# for multi-layer convolutional network
print('Accuracy with USPS Dataset: %g' % accuracy.eval(feed_dict={x: usps_test_data, y_: usps_test_2d_label,
keep_prob: 1.0}))
```

Fig: Predicting values and accuracy with trained model (python code)

Logistic Regression

```
Training set size = (56000, 784)
Validation set size = (14000, 784)
Testing set size = (19999,784)
```

Training phase (bias added to decision function)

```
Bias added to decision function
 [ -7.1   -0.55  -4.7   -7.14  -3.98   0.39  -7.97  -1.32 -12.87  -9.19]
```

Validation phase (Hyper parameter tuning)

Observing impact of bias function on accuracy

```
Results with no bias:
-------------------------
Accuracy with MNIST Dataset:  0.863642857143 (Accuracy is lower without bias)

Results with bias:
----------------------
Accuracy with MNIST Dataset:  0.869142857143 (Accuracy is higher with bias)
```

Testing phase with tuned parameters

```
Results with tuned parameters:
-------------------------
Accuracy with USPS Dataset:  0.315165758288
```

Single Layer Neural Network

```
Training set size = (55000, 784)
Validation set size = (10000, 784)
Testing set size = (19999,784)
```

Training phase (learning rate=0.5, batch_size=100 and no_of_iterations=1000)

Validation phase (Hyper parameter tuning)

Tuning values of learning rate, batch_size and no_of_iterations

```
Results with learning rate=0.1, batch_size=50 and no_of_iterations=500
-------------------------------------------------------------------------
Accuracy with MNIST Dataset:  0.8994 (Accuracy is lower with smaller values)

Results with learning rate=0.5, batch_size=100 and no_of_iterations=1000
-------------------------------------------------------------------------
Accuracy with MNIST Dataset:  0.9216 (Accuracy is higher with higher values)
```

Testing phase using tuned parameters

```
Results with tuned parameters:
-------------------------
Accuracy with USPS Dataset:  0.363268 (Improvement over logistic regression)
```

Multilayer Convolutional Neural Network

```
Training set size = (55000, 784)
Validation set size = (10000, 784)
Testing set size = (19999,784)
```

Training phase (using Adam Optimizer(1e-4))

Validation phase (Hyper parameter tuning)

Tuning values of steps

```
Results with steps = 800:
-------------------------

step 0, training accuracy 0.08
step 100, training accuracy 0.78
step 200, training accuracy 0.92
step 300, training accuracy 0.94
step 400, training accuracy 0.86
step 500, training accuracy 0.98
step 600, training accuracy 0.96
step 700, training accuracy 0.88 (Accuracy lower with few steps size)

Results with steps > 2000:
-------------------------
step 0, training accuracy 0.08
.
.
step 1800, training accuracy 1
step 1900, training accuracy 0.96
step 2000, training accuracy 1
step 2100, training accuracy 0.96
step 2200, training accuracy 0.96
step 2300, training accuracy 0.96
step 2400, training accuracy 0.96
step 2500, training accuracy 1 (Accuracy higher with large steps size)

Accuracy with MNIST Dataset: 0.9898
```

Testing phase using tuned parameters

```
Results with tuned parameters(steps=10000):
-------------------------
Accuracy with USPS Dataset: 0.634132 (Huge Improvement over logistic regress
ion and Single Layer Neural Network)
```

Summary

	MNIST Dataset Accuracy	USPS Dataset Accuracy
Model		
Logistic Regression	0.8691	0. 3152
Single Layer Neural Network	0. 9216	0. 363268
Multi-Layer Convolutional Network	0. 9898	0. 634132

	Highest Accuracy
	Lowest Accuracy

No Free Lunch Theorem

It states that there is **no such thing as a perfect algorithm** that can solve all problems.

Whatever algorithm is designed efficiently can solve the problem at hand.

If algorithm performs very well at a specific problem, then it will perform badly at some other problem and thus the average performance of an algorithm on all possible problems is same as compared to other algorithms.

Findings of the project "support" No Free Lunch Theorem

Justification:

In our case if multi-layer convolutional network was poorly designed then it would perform poor as compared to single layer neural network.

Example:

If step count is reduced to 800 then single layer neural network performs better than multi-layer convolutional network.

	MNIST Dataset Accuracy
Model	
Single Layer Neural Network	0. 9216
Multi-Layer Convolutional Network	0.88

Fig: Number of steps reduced in multi-layer convolutional network

Above table justifies that findings support No Free Lunch Theorem.