Project 3 Report

Implement and Evaluate Classification Algorithms

Submitted by
Hima Sujani Adike, 50246828
Anjali Sujatha Nair ,50248735
Soumya Venkatesan, 50246599

Problem Statement

The purpose of the project was to implement and evaluate classification algorithms on MNIST dataset and USPS dataset.

Approaches

Three different approaches were attempted.

1. Simple logistic regression using tensorflow, model trained on MNIST , tested on USPS
2. MLP with single hidden layer, model trained on MNIST, tested on USPS
3. Convoluted neural network with back propagation, model trained on MNIST, tested on USPS
4. BackPropagation with a hidden layer

Approach 1 : Simple Logistic regression using tensorflow

Simple logistic regression was implemented using tensorflow. In built MNIST dataset from tensorflow was used to train the model. After training on MNIST, the given USPS dataset was used to test the accuracy of the model. Gradient descent was used for optimising. Achieved an accuracy of 91.26 MNIST and 28.16 on USPS datasets.

Approach 2 : MLP with single hidden layer

Perceptron with single hidden layer was used. MLP was implemented using tensorflow. Gradient descent was used to optimise. Achieved an accuracy of MNIST is 93.41 and for USPS is 29.28

Approach 3 : CNN

Convoluted neural network was used with back propagation. Gradient descent was used for optimising. Achieved an accuracy of MNIST is 99.18 for MNIST dataset and for USPS it is 34.32

Approach 4: BackPropagation for hidden layer

In this approach by feedforwarding the values using simple logistic regression updated the weights by backpropagating through the neural network and the accuracy obtained for MNIST is 92.79 and for USPS is 29.17

Implementation

The code involves the following files
Main.py- The main file to be executed. Loads both datasets mnist dataset and usps dataset. Calls the other files for executing the classification algorithms.
uspsImages.py-Used to preprocess the given usps dataset.
logisticRegression.py- Implements simple logistic regression and trains and tests the datasets.

hiddenLayerMLP.py-Implements the MLP with single hidden layer and trains and tests on the datasets.
convolutednn.py-Implements convoluted neural network with back propagation and trains and tests on the datasets and backpropagation.py

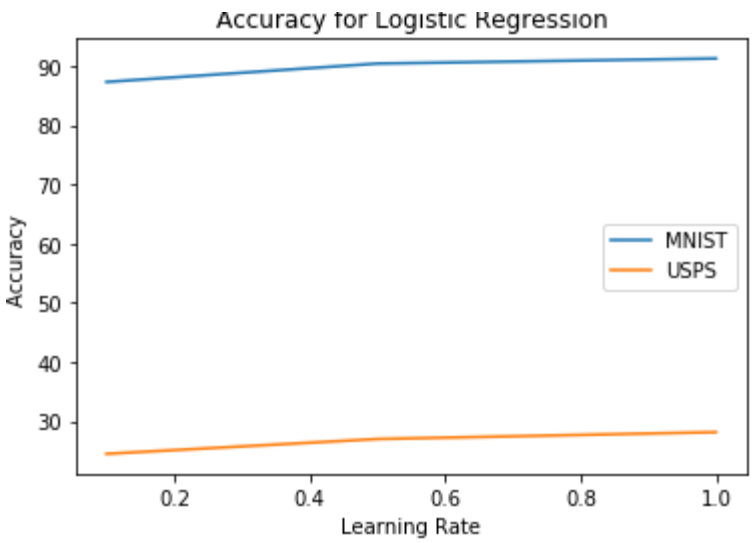
Main.py

Class to be called for the execution of different classification approaches listed above. Loads the MNIST data from tensorflow package. Loads the USPS data by calling preprocess method from uspsImages.py. After obtained the datasets, calls the three different files logisticRegression.py, hiddenLayerMLP.py and convolutednn.py to execute the algorithms. The files takes care of training and testing the model.

train_and_test_LR method from logisticregression.py is used to train and test the model on simple logistic regression. train_and_test_MLP method from hiddenLayerMLP.py is used to train and test the model on hidden layer perceptron. train_and_test_convNN method from convolutednn.py is used to train and test the model on convoluted neural network using back propagation.

logisticRegression.py

Implements the simple logistic regression approach. Main methods are train_and_test_LR and logisticregression. Method train_and_test_LR method is invoked from main.py. Method logisticRegression is invoked from train_and_test_LR and it implements the logic from simple logistic regression. Weights and bias are defined as shown below. Returns the prediction, x,y values



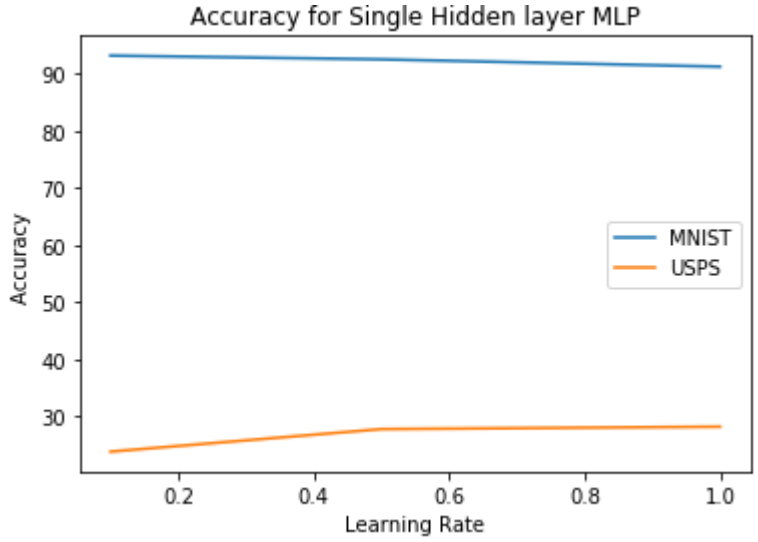

```
def logisticregression():
    n_hidden_1 = 256
    n_input = 784
    n_classes = 10
    weights = {
        'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])) // defining weights
    }
    biases = {
        'b1': tf.Variable(tf.random_normal([n_hidden_1])) //defining bias
    }
    x = tf.placeholder("float", [None, n_input])
    y = tf.placeholder("float", [None, n_classes])
    out_layer = tf.add(tf.matmul(x, weights['h1']), biases['b1']) //prediction
    return out_layer,x,y
```

Learning rate of 0.5, training epochs of 1800 and batch size of 750 was used in train_and_test_LR

```
def train_and_test_LR(mnist,uspsImages,uspsLabels):
    learning_rate = 0.5
    training_epochs = 3500
    batch_size = 950
    pred,x,y = logisticregression()
        cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y)) //cost is calculated
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) //using gradient descent
    init = tf.global_variables_initializer()
    with tf.Session() as sess:
        sess.run(init)
        for epoch in range(training_epochs):
            batch_x, batch_y = mnist.train.next_batch(batch_size) // trains the mnist in batches
            c = sess.run(optimizer, feed_dict={x: batch_x, y: batch_y})
        correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1)) //predicted is compared
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float")) // accuracy is calculated and reported
                print("Accuracy for MNIST test data with simple logistic regression:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})*100) // accuracy evaluated for mnist
        print("Accuracy for USPS test data with simple :", accuracy.eval({x: uspsImages, y: uspsLabels})*100) //accuracy evaluated for usps
```

hiddenLayerMLP.py

Main methods are create_multilayer_perceptron, train_and_test_MLP. create_multilayer_perceptron implements the logic for hidden layer perceptron with single hidden layer. In single hidden layer, using learning rate of 1 with training batch size of 950. In single hidden layer as the learning rate is increased accuracy for MNISt dataset decreased whereas accuracy for USPS dataset decreased.




```
def create_multilayer_perceptron():
    n_hidden_1 = 256
    n_input = 784
    n_classes = 10
    weights = {
```

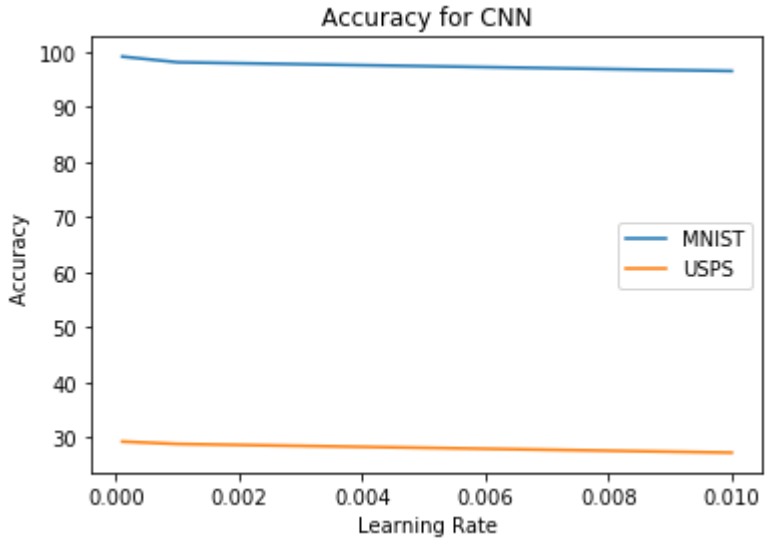
```
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'out': tf.Variable(tf.random_normal([n_hidden_1, n_classes])) // initialises weights
  }
  biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'out': tf.Variable(tf.random_normal([n_classes]))  //initialises bias
  }
  x = tf.placeholder("float", [None, n_input])
  y = tf.placeholder("float", [None, n_classes])
  layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
  layer_1 = tf.nn.relu(layer_1)  //creates the single hidden layer
  out_layer = tf.matmul(layer_1, weights['out']) + biases['out']
  return out_layer,x,y // returns the prediction
```

train_and_test_MLP methods trains and predicts the accuracy for mnist as well as usps.

```
def train_and_test_MLP(mnist,uspsImages,uspsLabels):
  learning_rate = 0.5
  training_epochs = 3500
  batch_size = 950
  pred,vg,y = create_multilayer_perceptron() // uses multilayer perceptron
  cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y)) // calculates cost
      optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)  //gradient descent
optimiser
  init = tf.global_variables_initializer()
  with tf.Session() as sess:
    sess.run(init)
    for epoch in range(training_epochs):
      batch_x, batch_y = mnist.train.next_batch(batch_size)
      c = sess.run(optimizer, feed_dict={x: batch_x, y: batch_y})
    correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1)) // evaluates the prediction
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
          print("Accuracy for MNIST test data with single hidden layer MLP:", accuracy.eval({x:
mnist.test.images, y: mnist.test.labels})*100) //accuracy calculated for mnist
      print("Accuracy for USPS test data with single hidden layer MLP:", accuracy.eval({x: uspsImages, y:
uspsLabels})*100) //accuracy calculated for usps images
```

convolutednn.py

Main methods are create_convoluted_multilayer_perceptron1 and train_and_test_convNN.
In this method we intialize the values with some biases and use a small convolution neural network which consists of convolution and pooling which will help to reduce the dimensionality of feature map and use a dropout to avoid overfitting the data and we update the weight vectors using adamoptimizer.




```
def create_convoluted_multilayer_perceptron1():
    n_input = 784
    n_classes = 10

    x = tf.placeholder("float", [None, n_input])
    y = tf.placeholder("float", [None, n_classes])
    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])
    out_layer = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

    return out_layer,x,y

def train_and_test_convNN(mnist,uspsImages,uspsLabels):
    learning_rate = 0.5
    training_epochs = 20000
```

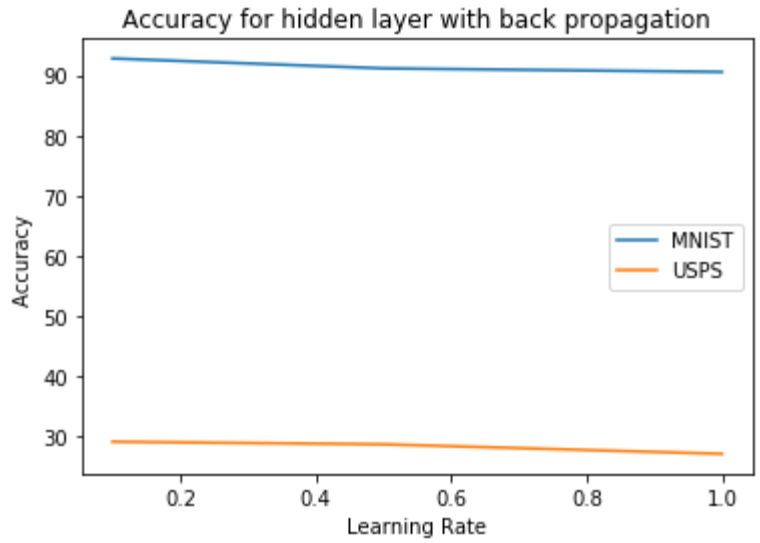
```
batch_size = 50
pred,vg,y = create_convoluted_multilayer_perceptron1() // prediction
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) // gradient descet optimizer
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(training_epochs):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        c = optimizer.run(feed_dict={vg: batch_x, y: batch_y, keep_prob: 0.5})
    correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
      print("Accuracy for MNIST test data for convoluted Neural network trained data:", accuracy.eval({x:
mnist.test.images, y: mnist.test.labels, keep_prob: 1.0})*100) //accuracy for mnist
       print("Accuracy for USPS test data for convoluted Neural network trained data:", accuracy.eval({x:
uspsImages, y: uspsLabels, keep_prob: 1.0})*100)  //accuracy for usps
```

backpropagation.py

In backpropagation, we updated the weight vectors w1,w2 using gradient descent method by calculating error gradient descent at both the layers using

w1=w1-errorgradient1($\partial E/\partial w(1) = \delta j * xi$) and w2=w2-errorgradient2($\partial E/\partial w(2) = \delta k * zj$),

and training the model as a whole dataset for 2000 times




```
def testandtrainMLP(mnist,uspsImages,uspsLabels):
    M=256
    w1=np.random.rand(784,M)
    w2=np.random.rand(10,M)
    a_j=np.matmul(mnist.train.images,w1)
    z_j=np.tanh(a_j)
    a_k=np.matmul(z_j,w2.transpose())
    y_output=[]
    for i in range(0,len(a_k)):
        temp1=[math.exp(j)/sum(a_k[i]) for j in a_k[i]]
        y_output.append(temp1)
    del_k=y_output-mnist.train.labels
    del_j=np.multiply((1-np.power(z_j,2)),np.matmul(del_k,w2))
```

```
errorGrad1=np.matmul(mnist.train.images.transpose(),del_j)
errorGrad2=np.matmul(del_k.transpose(),z_j)
learningRate=0.5
for i in range(0,1):
    w1old=w1;
    w1=w1old-learningRate*errorGrad1;
    w2old=w2;
    w2=w2old-learningRate*errorGrad2;
    print(mnist.train.images[0])
    a_j=np.matmul(mnist.train.images,w1)
    print('gt')
    print(a_j)
    z_j=np.tanh(a_j)
    a_k=np.matmul(z_j,w2.transpose())
    y_output=[]
    for i in range(0,len(mnist.train.labels)):
        temp1=[math.exp(j)/sum(a_k[i]) for j in a_k[i]]
        y_output.append(temp1)
    del_k=y_output-mnist.train.labels
    del_j=np.multiply((1-np.power(z_j,2)),np.matmul(del_k,w2))
    errorGrad1=np.matmul(mnist.train.images.transpose(),del_j)
    errorGrad2=np.matmul(del_k.transpose(),z_j)

    error_value=0;
    for n in range(1,length(mnist.train.images)):
        for k in range(1,10):
            error_value=error_value+np.mamtmul(t[n],log(y_output[n]))
    error_value=-1*error_value
```

Observations

<u>No free lunch theorem</u>

It is supported for all the approaches attempted in this project. No free lunch theorema states that a learning algorithm cannot perform universally well for all datasets. In all the approaches here the accuracy obtained for mnist was significantly higher than the accuracy obtained for usps dataset. Hence No free lunch holds in all the three approaches.s