Lab Course - Distributed Data Analytics
Exercise 5

Function name : generate_dataset
Parameter : None

This function generates a random uniform input data of size 1000 with mean 0 and standard deviation 1. With this data, a toy data is generated as mentioned in the question. It also splits the data and target into test and train.

```
def generate_dataset():
    x = np.random.uniform(0,1,size=(1000))
    l = np.random.normal(0,50,1000)
    y = 0.5 * x + 2 + l
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.1)
    return x_train,x_test,y_train,y_test
```

Linear Regression – Univariate:

```
lr = 0.001
epochs = 100

cost_tensorflow = False

X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(tf.zeros([1]), name="weight")
b = tf.Variable(tf.zeros([1]), name="bias")

# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)

cost = tf.reduce_mean(tf.pow(pred-Y, 2))
optimizer = tf.train.GradientDescentOptimizer(lr).minimize(cost)
```

For the tensorflow implementation, we initialise the learning rate and the number of epochs to be performed. Also, the placeholders of input and target are defined. The scalar variable of Weight and bias are initialised to 0 and prediction is a linear regression. I have used the cost function as shown in the above snippet of code and the optimizer used is Gradient Descent.

```
init = tf.global_variables_initializer()
```

This command is used to initialise all the global variables and the optimizers. This is to be used only then all the placeholders, variables, cost functions and optimizers will be initialised before the starting of tensorflow execution.

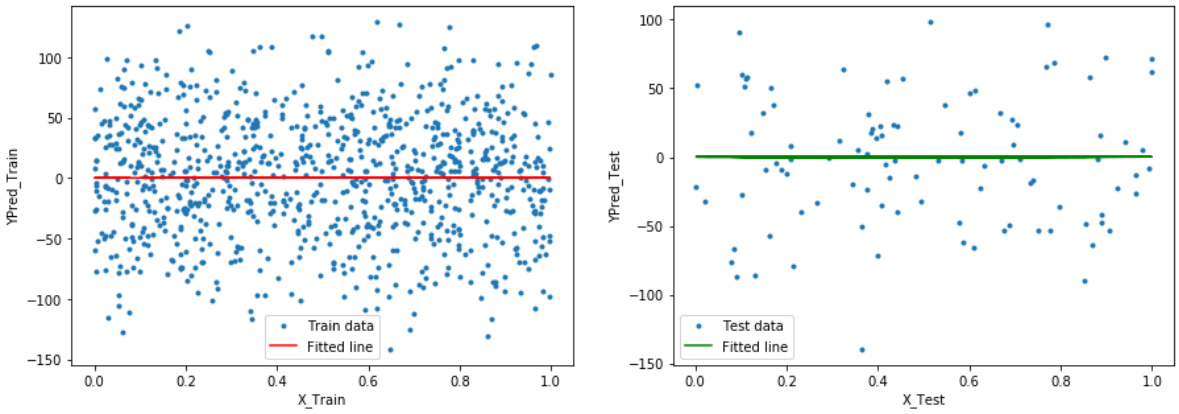
```
with tf.Session() as sess:
    print("Training...\n")
    sess.run(init)
    for epoch in range(epochs):
        sess.run(optimizer, feed_dict={X: x_train, Y: y_train})
        if (epoch+1) % 10 == 0:
            if cost_tensorflow == True:
                c = sess.run(cost, feed_dict={X: x_train, Y:y_train})
                print("Epoch:", (epoch+1), "cost=", round(c,4),
                "W=", round(sess.run(W)[0],4), "b=", round(sess.run(b)[0],4))
            else:
                ypred_train = sess.run(pred,feed_dict={X: x_train, Y:y_train})
                print("Epoch:", (epoch+1), "cost=",
                      round(np.mean((ypred_train - y_train)**2),4), \
                    "W=", round(sess.run(W)[0],4), "b=", round(sess.run(b)[0],4))
    print("\nTraining Completed!")
    if cost_tensorflow == True:
        training_cost = sess.run(cost, feed_dict={X: x_train, Y: y_train})
        print("Training cost=", np.round(training_cost,4),
              "W=", np.round(sess.run(W),4), "b=", np.round(sess.run(b),4), '\n')
        print("Error (RMSE)=", np.sqrt( training_cost))
    else:
        ypred_train = sess.run(pred, feed_dict={X: x_train, Y: y_train})
        print("\nTraining cost =", np.mean((y_train - ypred_train) ** 2))
        print("Error (RMSE)=", np.sqrt(np.mean((y_train - ypred_train) ** 2)))



    print("\nTesting...")
    if cost_tensorflow == True:
        testing_cost = sess.run(
            tf.reduce_mean(tf.pow(pred - Y, 2)), feed_dict={X: x_test, Y: y_test})
        print("Testing cost=", testing_cost)
        print("Error (RMSE)=", np.sqrt( testing_cost))

    else:
        ypred_test = sess.run(pred, feed_dict={X: x_test, Y: y_test})
        print("Testing cost=", np.mean((y_test - ypred_test) ** 2))
        print("Error (RMSE)=", np.sqrt(np.mean((y_test - ypred_test) ** 2)))
```

This part of the code creates a tensorflow session and based on the number of iterations, it updates the weights and bias using Gradient Descent optimizer.

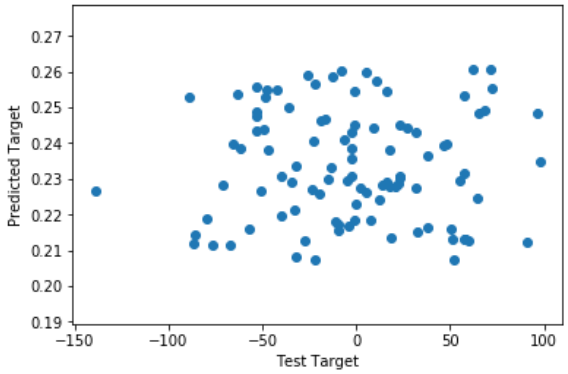
```
Training...

Epoch: 10 cost= 2459.7745 W= 0.0065 b= 0.0229
Epoch: 20 cost= 2459.7207 W= 0.0127 b= 0.0453
Epoch: 30 cost= 2459.6694 W= 0.0186 b= 0.0671
Epoch: 40 cost= 2459.6205 W= 0.0243 b= 0.0885
Epoch: 50 cost= 2459.5739 W= 0.0298 b= 0.1094
Epoch: 60 cost= 2459.5294 W= 0.035 b= 0.1299
Epoch: 70 cost= 2459.4871 W= 0.04 b= 0.1498
Epoch: 80 cost= 2459.4467 W= 0.0448 b= 0.1694
Epoch: 90 cost= 2459.4082 W= 0.0493 b= 0.1885
Epoch: 100 cost= 2459.3714 W= 0.0536 b= 0.2072

Training Completed!

Training cost = 2459.371422088731
Error (RMSE)= 49.59204998877069

Testing...
Testing cost= 2168.094947344863
Error (RMSE)= 46.56280648054693
```

Plots:

Training Data and Fitted line using GD optimizer



Scatter Plot of Ground Truth and Prediction



Linear Regression – Multivariate:

Similarly, for multivariate, the process remains the same except the fact that the weights and biases are now vector and they are initialised as follows and I've used different learning rates.

```
learningrate = [1e-7,1e-8,1e-9]
epochs = 100
RMSE_all,MSE_all,MAE_all = [],[],[]
for lr in learningrate:
    MAE,MSE,RMSE,ep = [],[],[],[]
    cost_tensorflow = False

    X = tf.placeholder("float")
    Y = tf.placeholder("float")

    # Set model weights
    W = tf.Variable(tf.zeros([1,x_train.shape[1]]), dtype= tf.float32, name = 'weight')
    b = tf.Variable(0, dtype = tf.float32, name="bias")
    # Construct a linear model
    pred = tf.tensordot(X, tf.transpose(W),1) + b

    MAerror = tf.reduce_mean(tf.abs(pred - Y))
    MSerror = tf.reduce_mean(tf.square(pred - Y))
    RMSerror = tf.sqrt(tf.reduce_mean(tf.square(pred - Y)))

    cost = tf.reduce_mean(tf.square(pred-Y))
    optimizer = tf.train.GradientDescentOptimizer(lr).minimize(cost)
    init = tf.global_variables_initializer()
```

```
with tf.Session() as sess:
    print("Training...\n")
    sess.run(init)
    for epoch in range(epochs):
        sess.run(optimizer, feed_dict={X: x_train, Y: y_train})
        if (epoch+1) % 10 == 0:
            ep.append(epoch+1)
            MAE.append(sess.run(MAerror,feed_dict={X: x_train, Y:y_train}))
            MSE.append(sess.run(MSerror,feed_dict={X: x_train, Y:y_train}))
            RMSE.append(sess.run(RMSerror,feed_dict={X: x_train, Y:y_train}))
            print("Epoch:", (epoch+1), "cost=", MSE[-1])
    print("\nTraining Completed!")
    ypred_test = sess.run(pred, feed_dict={X: x_test, Y: y_test})
    ypred_train = sess.run(pred, feed_dict={X: x_train, Y: y_train})
    print("\nTraining cost =", MSE[-1])
    print("Error (RMSE)=", RMSE[-1])
    print(sess.run(W))
    RMSE_all.append(RMSE)
    MSE_all.append(MSE)
    print(ypred_train[10])
    MAE_all.append(MAE)
```

Results:

LR: 1e-7

```
Training...

Epoch: 10 cost= 158.61772
Epoch: 20 cost= 108.39895
Epoch: 30 cost= 102.66211
Epoch: 40 cost= 101.72017
Epoch: 50 cost= 101.298416
Epoch: 60 cost= 100.937195
Epoch: 70 cost= 100.586716
Epoch: 80 cost= 100.24159
Epoch: 90 cost= 99.90111
Epoch: 100 cost= 99.56509

Training Completed!

Training cost = 99.56509
Error (RMSE)= 9.97823
[[ 1.8557441e-05 -4.6907715e-03 -1.0629369e-04  7.6756966e-03
   7.2447449e-04  2.9990440e-03  1.1699299e-04]]
[17.752922]
```

LR: 1e-8

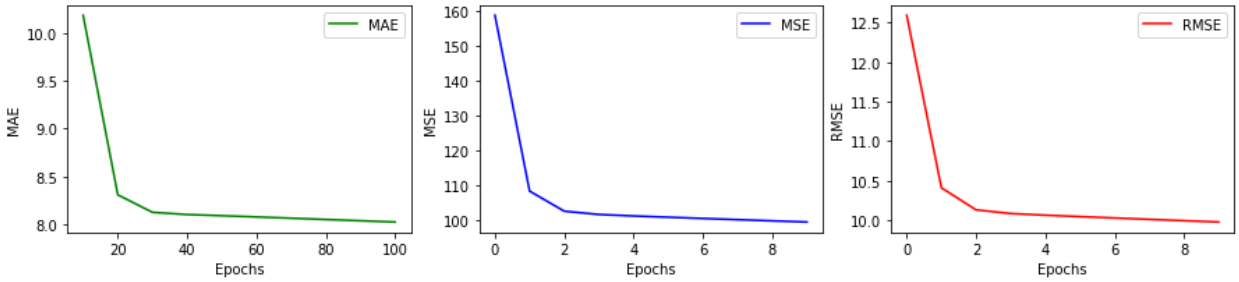
```
Training...

Epoch: 10 cost= 110.86871
Epoch: 20 cost= 103.17663
Epoch: 30 cost= 103.024155
Epoch: 40 cost= 102.98423
Epoch: 50 cost= 102.946
Epoch: 60 cost= 102.9082
Epoch: 70 cost= 102.87074
Epoch: 80 cost= 102.83272
Epoch: 90 cost= 102.794815
Epoch: 100 cost= 102.757324

Training Completed!

Training cost = 102.757324
Error (RMSE)= 10.136929
[[ 1.39316835e-05 -2.95405480e-05  2.23042240e-04  7.40664871e-03
   1.05682077e-04  4.63281380e-04  1.49781581e-05]]
[17.34223]
```

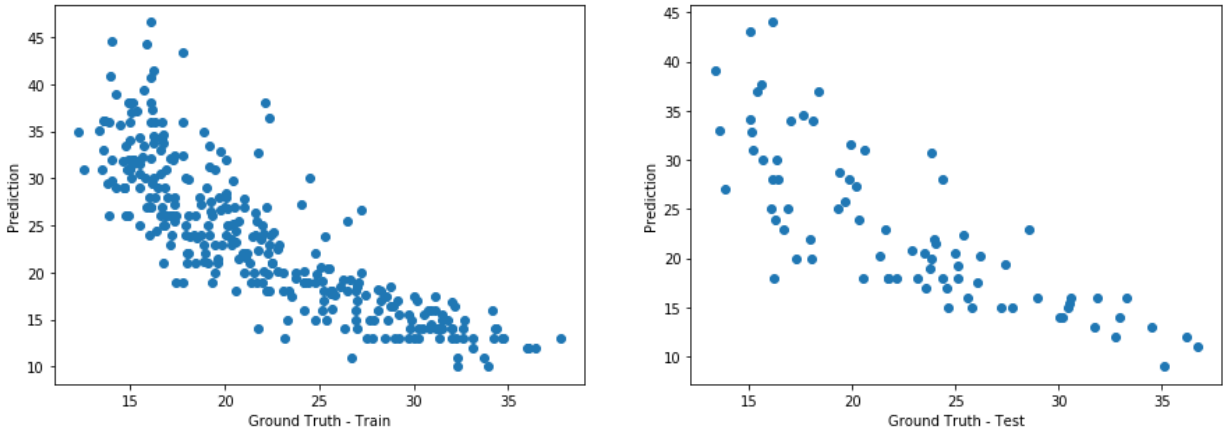
LR: 1e-9

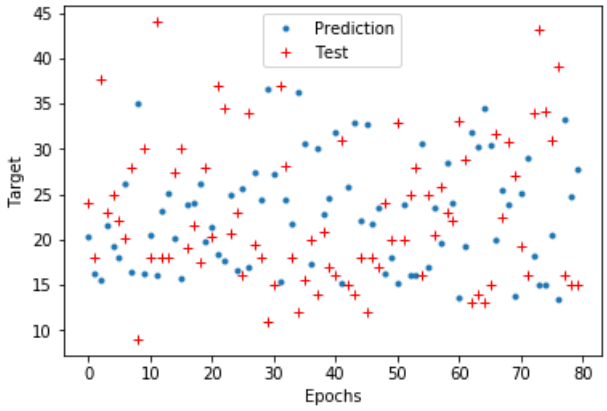
```
Training...

Epoch: 10 cost= 456.94565
Epoch: 20 cost= 344.4726
Epoch: 30 cost= 267.75076
Epoch: 40 cost= 215.4165
Epoch: 50 cost= 179.71559
Epoch: 60 cost= 155.36322
Epoch: 70 cost= 138.75023
Epoch: 80 cost= 127.41683
Epoch: 90 cost= 119.68504
Epoch: 100 cost= 114.40971

Training Completed!

Training cost = 114.40971
Error (RMSE)= 10.696248
[[1.1560858e-05 3.7856752e-04 2.1991332e-04 6.2894374e-03 3.7032412e-05
  1.7812499e-04 4.0325735e-06]]
[14.745646]
```

Learning Rate: 1e-7

MSE, MAE, RMSE, scatter plot and test with prediction plots

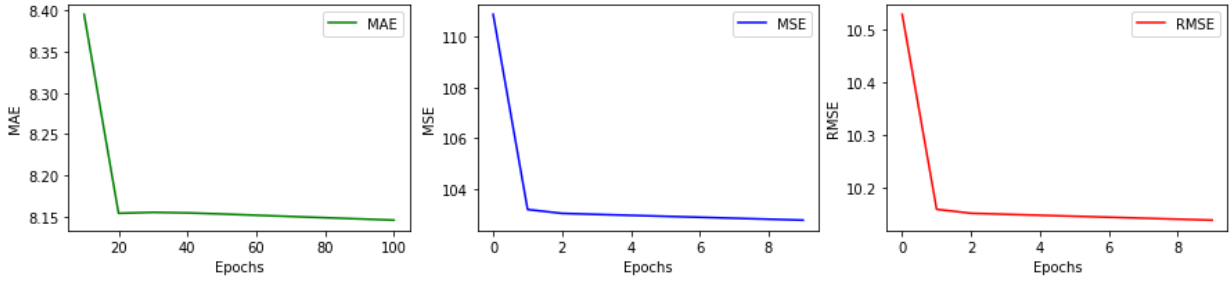


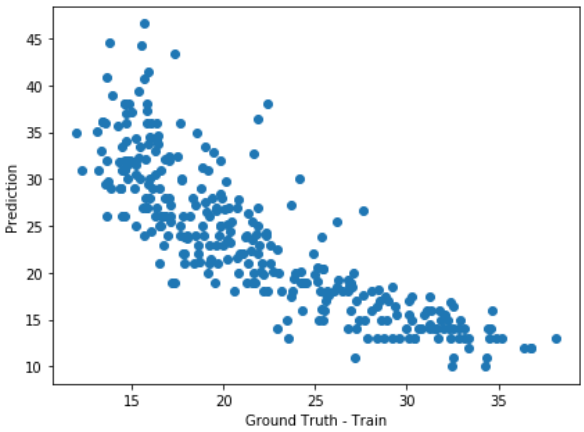


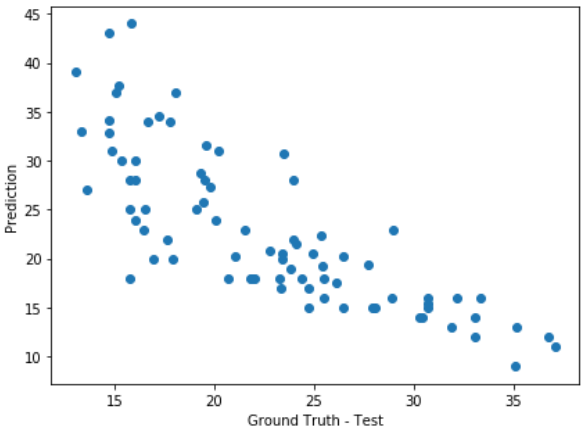


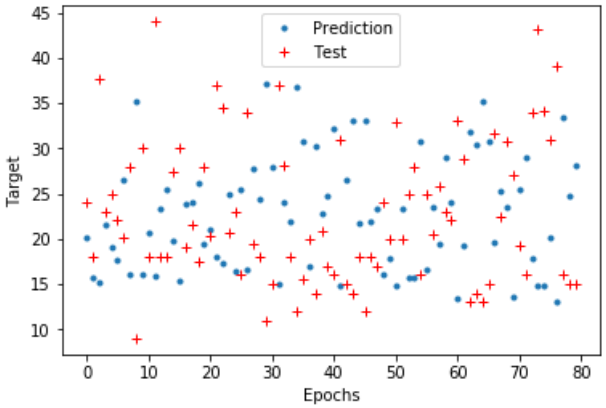
Learning Rate: 1e-8

MSE, MAE, RMSE, scatter plot and test with prediction plots



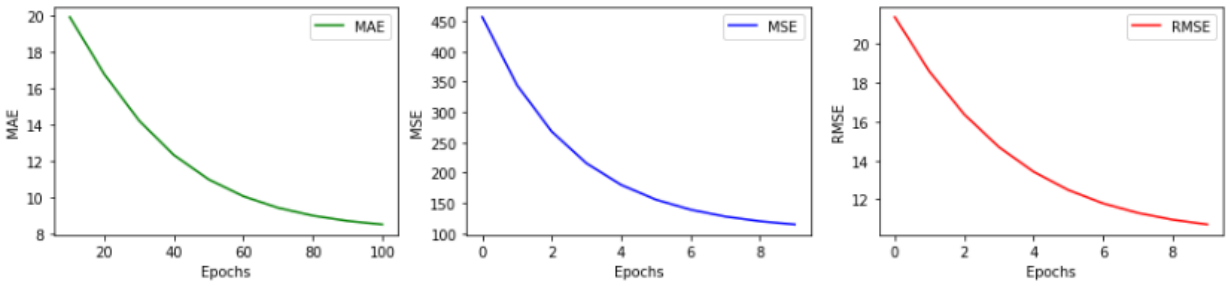


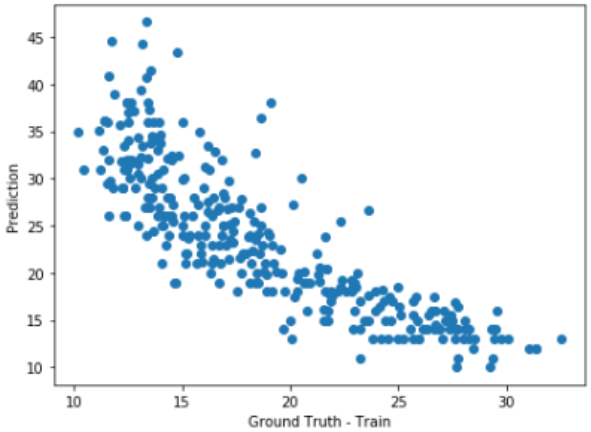


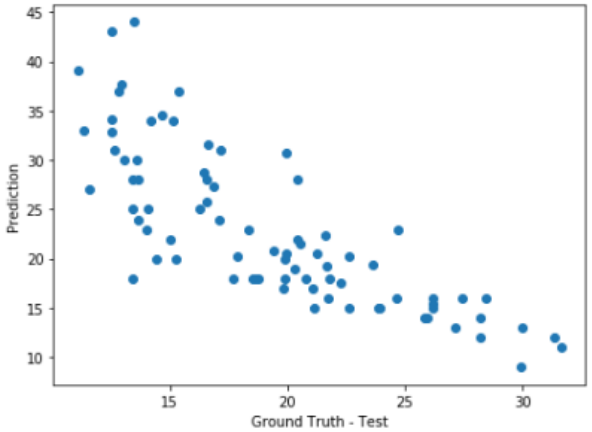


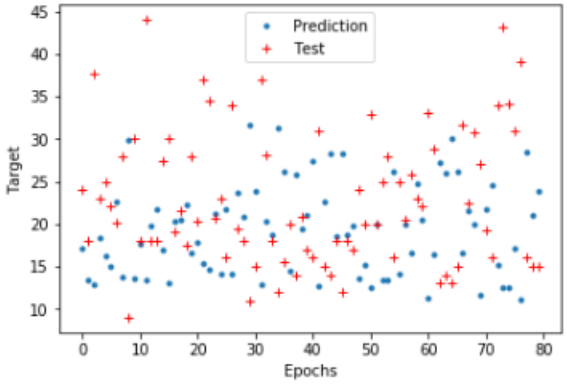
Learning Rate: 1e-9

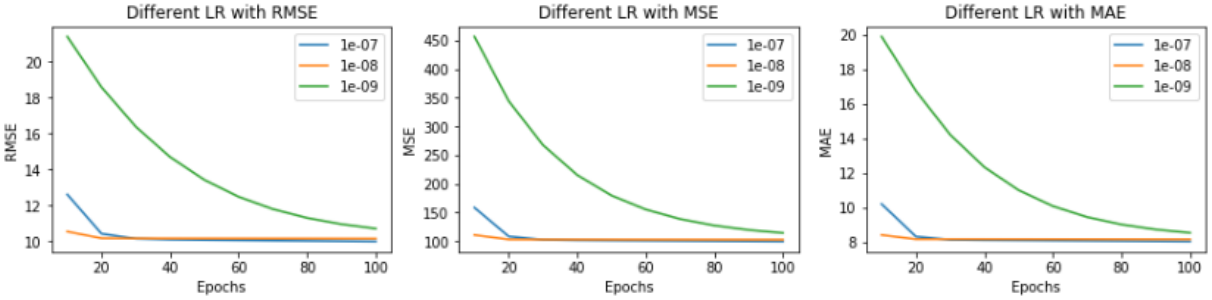
MSE, MAE, RMSE, scatter plot and test with prediction plots







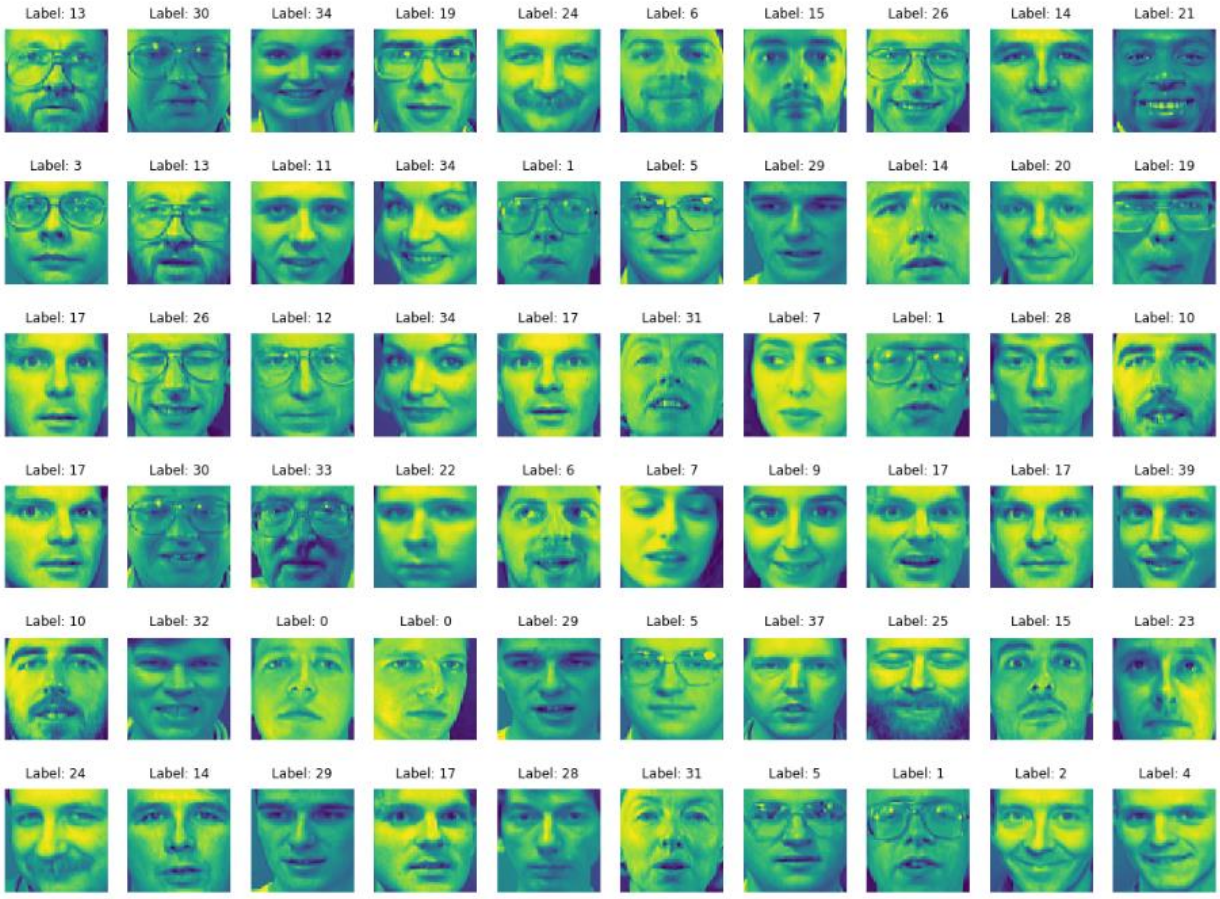




Logistic Regression:

Loading Olivetti Dataset:

The dataset is loaded from sklearn and a sample of images with labels is shown below.



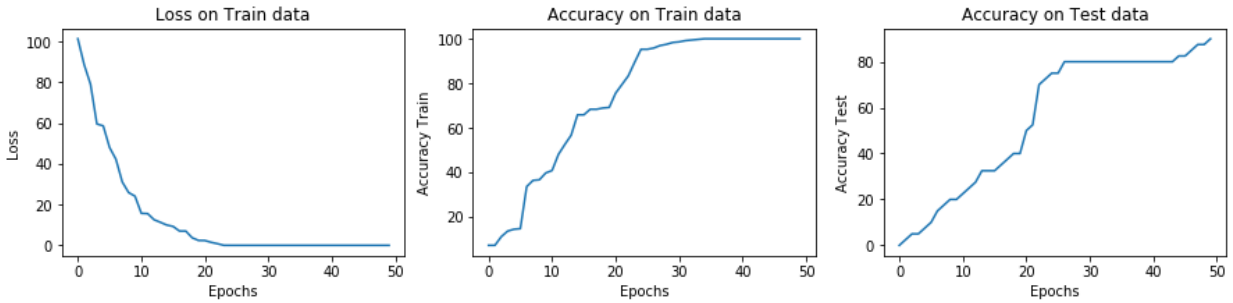
Logistic Regression with Gradient Descent Optimizer:

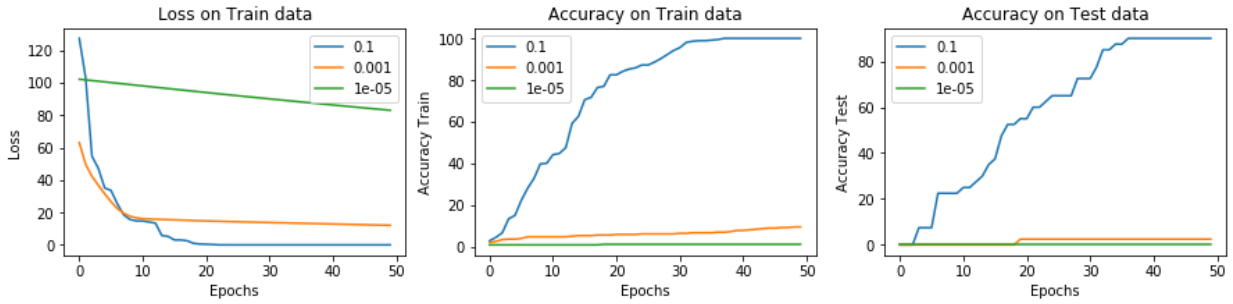
```
learningrate = [0.1,1e-3,1e-5]
lp,at,ate = [],[],[]
for lr in learningrate:
    W = tf.Variable(tf.random_normal(shape=[4096, 40]),name = "W")
    b = tf.Variable(tf.zeros([40]),name = "b")

    init = tf.global_variables_initializer()
    sess = tf.Session()
    sess.run(init)

    # Define placeholders
    data = tf.placeholder("float", name = "data")
    target = tf.placeholder("float",name = "target")

    # Declare the model you need to learn
    logits = tf.matmul(data, W) + b

    # cross-entropy loss function
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=target, logits=logits))

    # Define the optimizer
    optimizer = tf.train.GradientDescentOptimizer(lr).minimize(loss)

    # Predictions
    prediction = tf.nn.softmax(logits)

    # In[3]:

    init = tf.global_variables_initializer()
    iter_num = 50
```

As discussed in the linear regression, here in logistic regression, I've chosen 3 different learning rates and Gradient descent optimizer. The loss function chosen is softmax cross entropy loss because of the different number of classes to be classified in the example. As we know, there are 40 different faces which have to be classified.

```
with tf.Session() as sess:
    # Run the initializer
    sess.run(init)
    # Training cycle
    for epoch in range(iter_num):
        total_batch = 100
        batch_x_train = np.array_split(train_X,total_batch)
        batch_y_train = np.array_split(train_y,total_batch)
        # Loop over all batches
        for x_batch,y_batch in zip(batch_x_train,batch_y_train):
            # Prepare the feed dict
            feed_dict = {data : x_batch,target : y_batch}
    # run one step of computation
            opt, lo, predictions = sess.run([optimizer, loss, prediction],feed_dict=feed_dict)
        loss_plot.append(lo)
        feed_dict = {data : train_X,target : train_y}
        predictions = sess.run(prediction,feed_dict=feed_dict)
        accuracy_train.append(accuracy(predictions, train_y))
        feed_dict = {data : test_X,target : test_y}
        predictions = sess.run(prediction,feed_dict=feed_dict)
        accuracy_test.append(accuracy(predictions, test_y))
lp.append(loss_plot)
at.append(accuracy_train)
ate.append(accuracy_test)
```

Results on Test Data



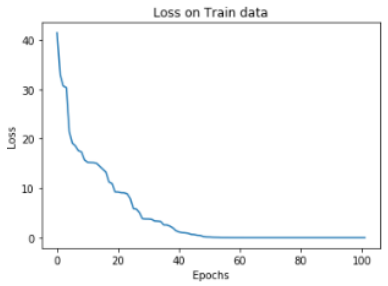


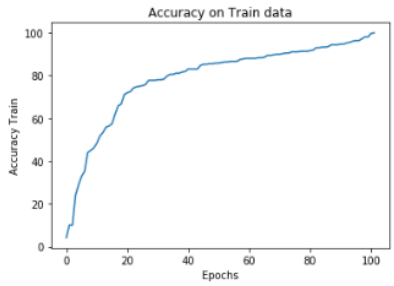
Logistic Regression with RMSProp Optimizer:

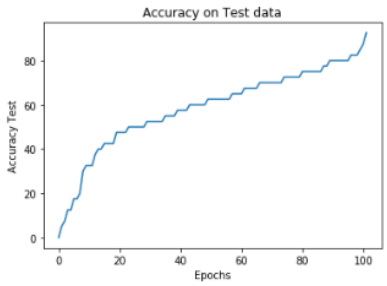
```
# Define the optimizer
optimizer = tf.train.RMSPropOptimizer(lr,momentum=0.9).minimize(loss)
```

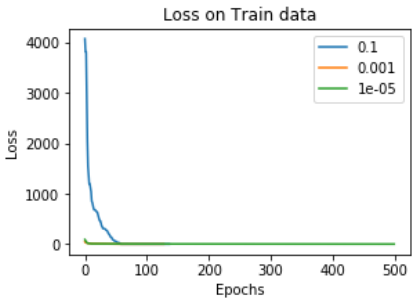
For different learning rates, the logistic regression is done with RMSProp Optimizer for which I've given a momentum of 0.9 as a parameter.

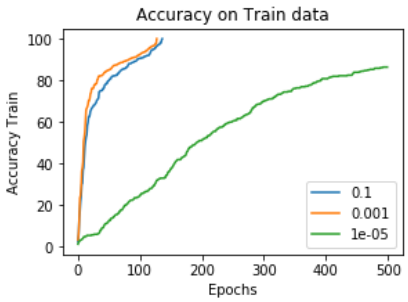
Results:

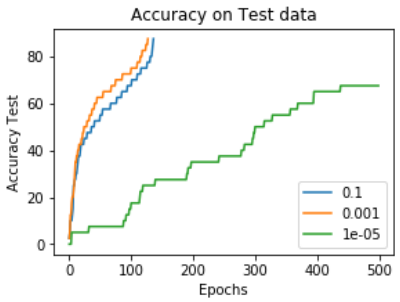








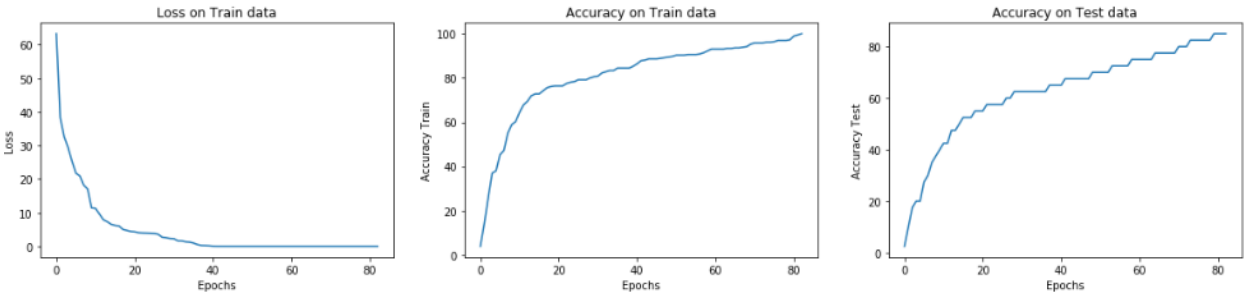


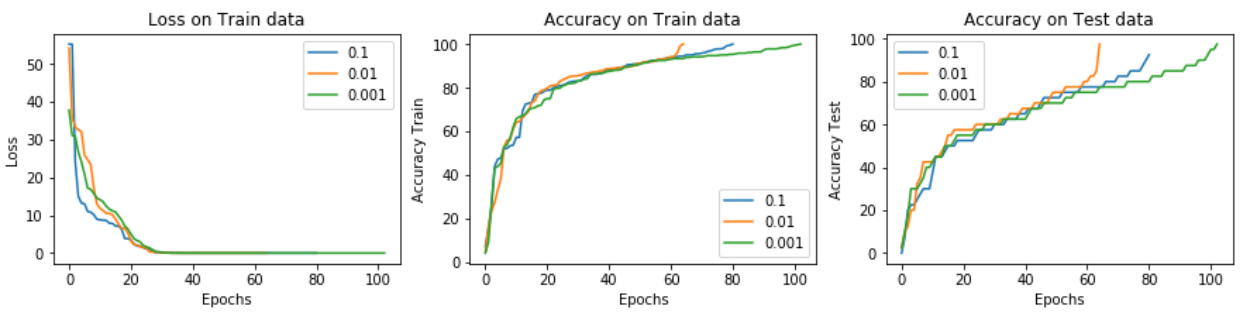


Logistic Regression with Adam Optimizer:

```
# Define the optimizer
optimizer = tf.train.AdamOptimizer(lr).minimize(loss)
```

For different learning rates, the logistic regression is done with Adam Optimizer.





Varying different learning rate on 3 different optimizers, it can be seen that Adam Optimizer with learning rate 0.01 performs well on the test data. Also, the loss for train data with all the train data converges to 0 and the accuracy on train and test data is really good with all the optimizers for certain learning rates. RMSProp optimizer performs well with a learning rate of 0.001and momentum 0.9 and Gradient Descent optimizer performs well with a learning rate of 0.1