CS 5783 - Machine Learning - Homework 3

Romit Maulik

September 2018

Question 1

The performance of the linear least-squares regression model based on polynomial basis functions is shown in Figure 1. A choice of an 18th order polynomial for prediction on the training data set led to the prediction observed in Figure 2.

Question 2

The performance of the linear least-squares regression model based on radial basis functions is shown in Figure 3. A choice of 10 basis centers for prediction on the training data set led to the prediction observed in Figure 4.

Question 3

The performance of the linear least-squares regression model based on 50 radial basis functions is shown in Figure 5. In this figure the x-axis corresponds to the choice of α in the prior. The optimal alpha was seen to be $\alpha = e^{-5}$ and its prediction is seen in Figure 6.

Question 4

The classification accuracy of the logitic regression was observed to be 98.33% using an $\alpha = e^{-5}$

Figure 1: Training and testing loss variation with differing complexity of measurement matrix (polynomial basis functions).

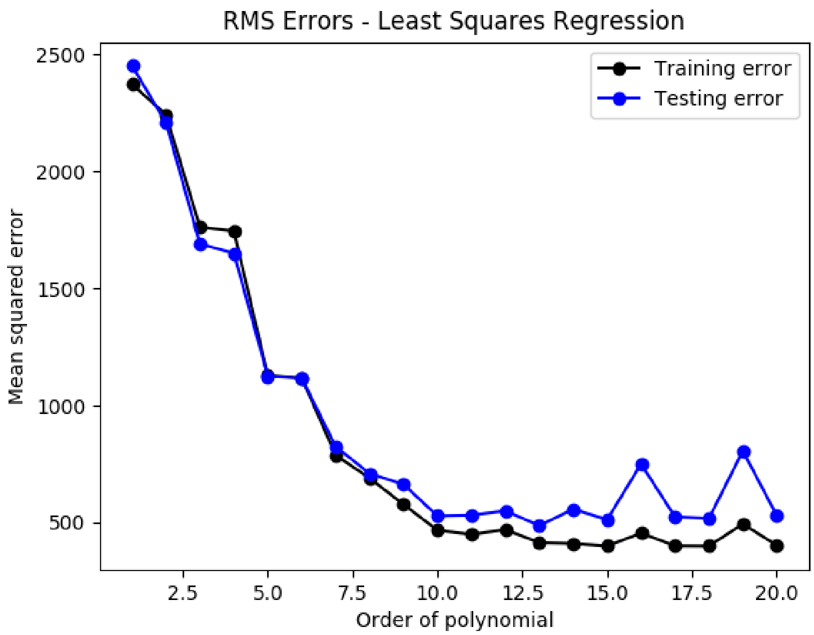


Figure 2: Behavior of 18th order polynomial prediction on the training data.

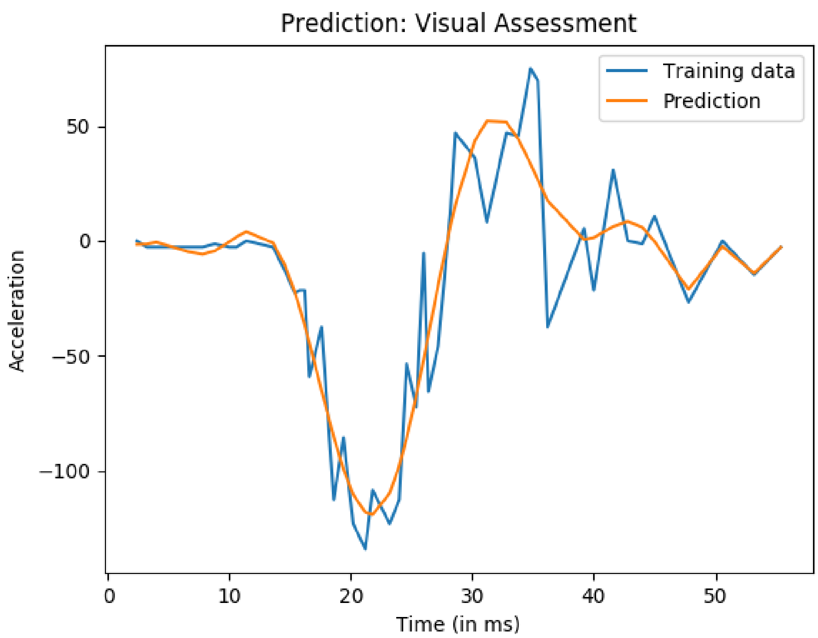


Figure 3: Training and testing loss variation with differing complexity of measurement matrix (radial basis functions).

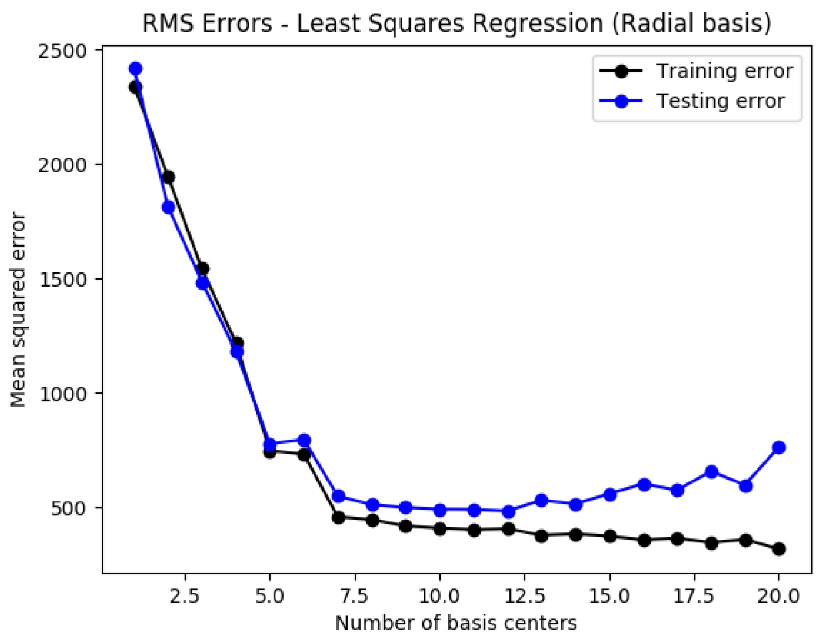


Figure 4: Behavior of prediction on the training data with 10 basis centers

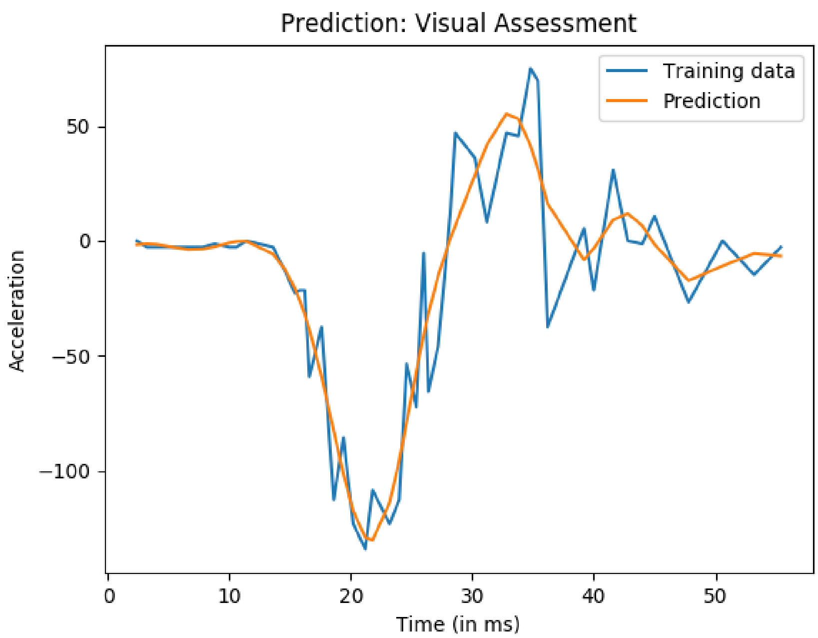


Figure 5: Training and testing loss variation with differing value of α in prior (50 radial basis functions).

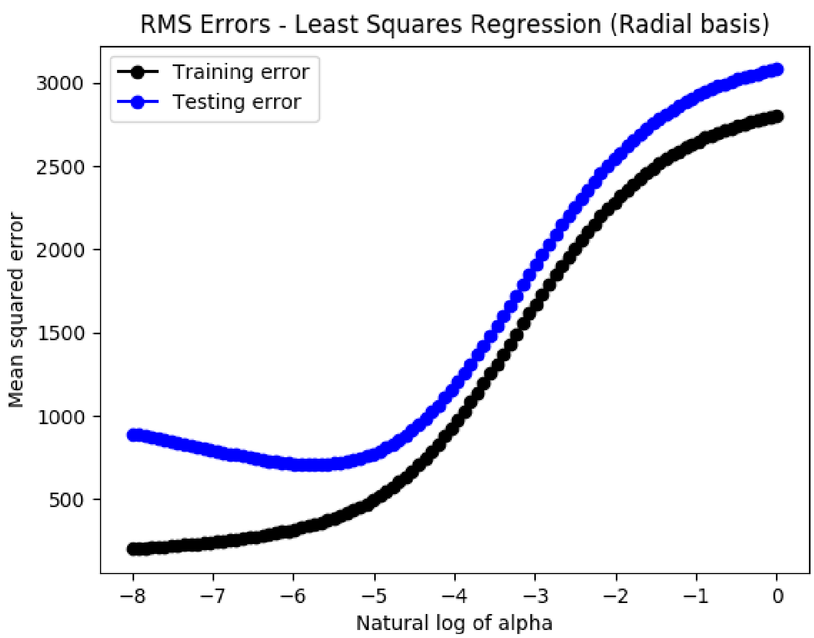
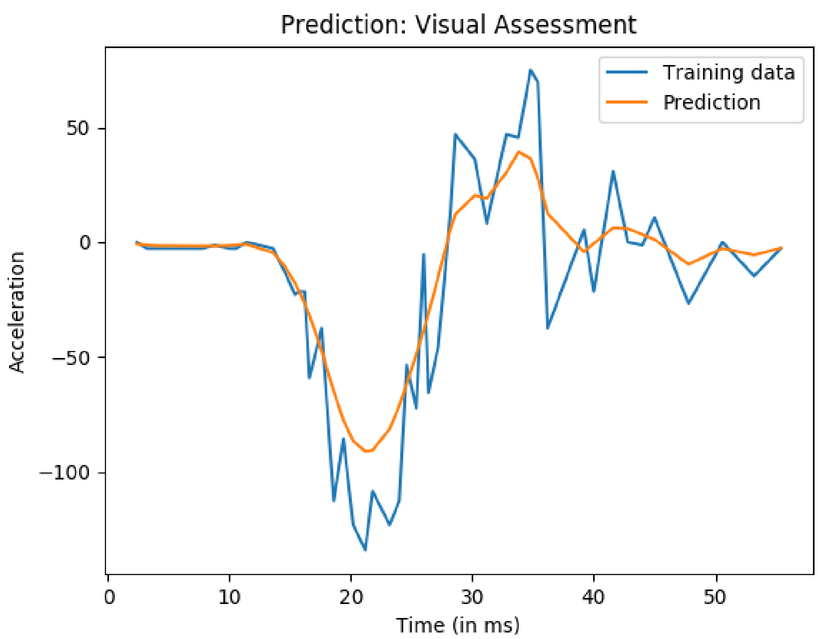


Figure 6: Behavior of prediction on the training data with 50 radial bases and $\alpha = e^{-5}$



```python
import numpy as np
import matplotlib.pyplot as plt

#Load data
def load_data():

    data = np.loadtxt('crash.txt')

    #Segregate
    idx = np.arange(0,np.shape(data)[0],2)
    training_data = data[idx,:]

    idx = np.arange(1,np.shape(data)[0],2)
    test_data = data[idx,:]
    return training_data, test_data

def least_squares_regression(training_data,test_data,poly_order):


    #Generate transformed matrix
    phi = np.empty(shape=(np.shape(training_data)[0],poly_order+1))
    for col in range(poly_order+1):
        phi[:,col] = training_data[:,0]**col

    #Solve for optimal weights
    lhs = np.matmul(np.transpose(phi),phi)
    rhs = np.matmul(np.transpose(phi),training_data[:,1])

    w_opt = np.linalg.solve(lhs,rhs)

    #Find RMS error on training data
    pred = np.matmul(phi,w_opt)
    rms_train = np.sum((pred-training_data[:,1])**2,axis=0)
    rms_train = rms_train/np.shape(training_data)[0]

    #Find RMS error on test data
    phi = np.empty(shape=(np.shape(test_data)[0],poly_order+1))
    for col in range(poly_order+1):
        phi[:,col] = test_data[:,0]**col

    pred = np.matmul(phi,w_opt)
    rms_test = np.sum((pred-test_data[:,1])**2,axis=0)
    rms_test = rms_test / np.shape(test_data)[0]

    return rms_train, rms_test

def plot_rms():
    #Load data
    training_data, test_data = load_data()

    #Plot the errors
    fig, ax = plt.subplots(nrows=1,ncols=1)
    ax.set_title('RMS Errors - Least Squares Regression')
    ax.set_xlabel('Order of polynomial')
    ax.set_ylabel('Mean squared error')

    train_info = []
    test_info = []

    poly_order = 1
    while poly_order <= 20:
        rms_train, rms_test = least_squares_regression(training_data,test_data,poly_order)
        train_info.append([poly_order, rms_train])
```

```
        test_info.append([poly_order, rms_test])
        poly_order = poly_order + 1

    train_info = np.asarray(train_info)
    test_info = np.asarray(test_info)

    ax.plot(train_info[:,0],train_info[:,1], color='black',label='Training error',marker='o')
    ax.plot(test_info[:,0],test_info[:,1], color='blue',label='Testing error',marker='o')
    plt.legend()
    plt.show()

def plot_performance(poly_order):

    #Load training data
    training_data, _ = load_data()

    # Generate transformed matrix
    phi = np.empty(shape=(np.shape(training_data)[0], poly_order + 1))
    for col in range(poly_order + 1):
        phi[:, col] = training_data[:, 0] ** col

    # Solve for optimal weights
    lhs = np.matmul(np.transpose(phi), phi)
    rhs = np.matmul(np.transpose(phi), training_data[:, 1])

    w_opt = np.linalg.solve(lhs, rhs)

    # Find prediction on training data
    pred = np.matmul(phi, w_opt)

    fig,ax = plt.subplots(nrows=1,ncols=1)
    ax.set_title('Prediction: Visual Assessment')
    ax.set_xlabel('Time (in ms)')
    ax.set_ylabel('Acceleration')

    ax.plot(training_data[:,0],training_data[:,1],label='Training data')
    ax.plot(training_data[:, 0], pred[:], label='Prediction')

    plt.legend()
    plt.show()

plot_rms()
plot_performance(18)
```

```python
import numpy as np
import matplotlib.pyplot as plt


#Load data
def load_data():

    data = np.loadtxt('crash.txt')

    #Segregate
    idx = np.arange(0,np.shape(data)[0],2)
    training_data = data[idx,:]

    idx = np.arange(1,np.shape(data)[0],2)
    test_data = data[idx,:]
    return training_data, test_data


def least_squares_regression(training_data,test_data,poly_order):
    #Distribute basis centers
    basis_centers = np.arange(0.0,60.0,step=60.0/float(poly_order+1))
    sd = 60.0/float(poly_order+1)

    #Make transformed matrix
    phi = np.empty(shape=(np.shape(training_data)[0],poly_order+1))
    for col in range(poly_order+1):
        phi[:, col] = np.exp(-(training_data[:, 0] - basis_centers[col])**2/(2.0*sd**2))

    #Solve for optimal weights
    lhs = np.matmul(np.transpose(phi),phi)
    rhs = np.matmul(np.transpose(phi),training_data[:,1])

    w_opt = np.linalg.solve(lhs,rhs)

    #Find RMS error on training data
    pred = np.matmul(phi,w_opt)
    rms_train = np.sum((pred-training_data[:,1])**2,axis=0)
    rms_train = rms_train/np.shape(training_data)[0]

    #Find RMS error on test data
    phi = np.empty(shape=(np.shape(test_data)[0],poly_order+1))
    for col in range(poly_order+1):
        phi[:, col] = np.exp(-(test_data[:, 0] - basis_centers[col])**2/(2.0*sd**2))

    pred = np.matmul(phi,w_opt)
    rms_test = np.sum((pred-test_data[:,1])**2,axis=0)
    rms_test = rms_test / np.shape(test_data)[0]

    return rms_train, rms_test


def plot_rms():
    #Load data
    training_data, test_data = load_data()

    #Plot the errors
    fig, ax = plt.subplots(nrows=1,ncols=1)
    ax.set_title('RMS Errors - Least Squares Regression (Radial basis)')
    ax.set_xlabel('Number of basis centers')
    ax.set_ylabel('Mean squared error')

    train_info = []
    test_info = []
```

```
    poly_order = 1
    while poly_order <= 20:
        rms_train, rms_test = least_squares_regression(training_data,test_data,poly_order)
        train_info.append([poly_order, rms_train])
        test_info.append([poly_order, rms_test])
        poly_order = poly_order + 1

    train_info = np.asarray(train_info)
    test_info = np.asarray(test_info)

    ax.plot(train_info[:,0],train_info[:,1], color='black',label='Training error',marker='o')
    ax.plot(test_info[:,0],test_info[:,1], color='blue',label='Testing error',marker='o')
    plt.legend()
    plt.show()


def plot_performance(poly_order):

    #Load training data
    training_data, _ = load_data()

    # Distribute basis centers
    basis_centers = np.arange(0.0, 60.0, step=60.0 / float(poly_order + 1))
    sd = 60.0 / float(poly_order + 1)

    # Make transformed matrix
    phi = np.empty(shape=(np.shape(training_data)[0], poly_order + 1))
    for col in range(poly_order + 1):
        phi[:, col] = np.exp(-(training_data[:, 0] - basis_centers[col]) ** 2 / (2.0 * sd ** 2
))

    # Solve for optimal weights
    lhs = np.matmul(np.transpose(phi), phi)
    rhs = np.matmul(np.transpose(phi), training_data[:, 1])

    w_opt = np.linalg.solve(lhs, rhs)

    # Find prediction on training data
    pred = np.matmul(phi, w_opt)

    fig,ax = plt.subplots(nrows=1,ncols=1)
    ax.set_title('Prediction: Visual Assessment')
    ax.set_xlabel('Time (in ms)')
    ax.set_ylabel('Acceleration')

    ax.plot(training_data[:,0],training_data[:,1],label='Training data')
    ax.plot(training_data[:, 0], pred[:], label='Prediction')

    plt.legend()
    plt.show()

plot_rms()
plot_performance(10)
```

```
import numpy as np
import matplotlib.pyplot as plt

#Load data
def load_data():

    data = np.loadtxt('crash.txt')

    #Segregate
    idx = np.arange(0,np.shape(data)[0],2)
    training_data = data[idx,:]

    idx = np.arange(1,np.shape(data)[0],2)
    test_data = data[idx,:]
    return training_data, test_data

def least_squares_regression(training_data,test_data,alpha):
    poly_order = 50

    #Distribute basis centers
    basis_centers = np.arange(0.0,60.0,step=60.0/float(poly_order+1))
    sd = 60.0/float(poly_order+1)

    #Add prior information
    beta = 0.0025

    #Make transformed matrix
    phi = np.empty(shape=(np.shape(training_data)[0],poly_order+1))
    for col in range(poly_order+1):
        phi[:, col] = np.exp(-(training_data[:, 0] - basis_centers[col])**2/(2.0*sd**2))

    #Prior matrix
    prior_mat = alpha/beta*np.identity(np.shape(phi)[1])

    #Solve for optimal weights
    lhs = np.matmul(np.transpose(phi),phi) + prior_mat
    rhs = np.matmul(np.transpose(phi),training_data[:,1])

    w_opt = np.linalg.solve(lhs,rhs)

    #Find RMS error on training data
    pred = np.matmul(phi,w_opt)
    rms_train = np.sum((pred-training_data[:,1])**2,axis=0)
    rms_train = rms_train/np.shape(training_data)[0]

    #Find RMS error on test data
    phi = np.empty(shape=(np.shape(test_data)[0],poly_order+1))
    for col in range(poly_order+1):
        phi[:, col] = np.exp(-(test_data[:, 0] - basis_centers[col])**2/(2.0*sd**2))

    pred = np.matmul(phi,w_opt)
    rms_test = np.sum((pred-test_data[:,1])**2,axis=0)
    rms_test = rms_test / np.shape(test_data)[0]

    return rms_train, rms_test

def plot_rms():
    # Load data
    training_data, test_data = load_data()

    # Plot the errors
    fig, ax = plt.subplots(nrows=1, ncols=1)
    ax.set_title('RMS Errors - Least Squares Regression (Radial basis)')
```

```
    ax.set_xlabel('Natural log of alpha')
    ax.set_ylabel('Mean squared error')

    alpha = np.logspace(-8, 0, 100,base=np.e)

    train_info = []
    test_info = []

    iter = 0
    while iter < np.shape(alpha)[0]:
        alpha_val = alpha[iter]
        rms_train, rms_test = least_squares_regression(training_data, test_data, alpha_val)
        train_info.append([np.log(alpha_val), rms_train])
        test_info.append([np.log(alpha_val), rms_test])
        iter = iter + 1

    train_info = np.asarray(train_info)
    test_info = np.asarray(test_info)

    ax.plot(train_info[:, 0], train_info[:, 1], color='black', label='Training error', marker=
'o')
    ax.plot(test_info[:, 0], test_info[:, 1], color='blue', label='Testing error', marker='o')
    plt.legend()
    plt.show()

def plot_performance(alpha):
    # Load data
    training_data, _ = load_data()

    poly_order = 50

    # Distribute basis centers
    basis_centers = np.arange(0.0, 60.0, step=60.0 / float(poly_order + 1))
    sd = 60.0 / float(poly_order + 1)

    # Add prior information
    beta = 0.0025

    # Make transformed matrix
    phi = np.empty(shape=(np.shape(training_data)[0], poly_order + 1))
    for col in range(poly_order + 1):
        phi[:, col] = np.exp(-(training_data[:, 0] - basis_centers[col]) ** 2 / (2.0 * sd ** 2
))

    # Prior matrix
    prior_mat = alpha / beta * np.identity(np.shape(phi)[1])

    # Solve for optimal weights
    lhs = np.matmul(np.transpose(phi), phi) + prior_mat
    rhs = np.matmul(np.transpose(phi), training_data[:, 1])

    w_opt = np.linalg.solve(lhs, rhs)

    # Find prediction on training data
    pred = np.matmul(phi, w_opt)

    fig,ax = plt.subplots(nrows=1,ncols=1)
    ax.set_title('Prediction: Visual Assessment')
    ax.set_xlabel('Time (in ms)')
    ax.set_ylabel('Acceleration')

    ax.plot(training_data[:,0],training_data[:,1],label='Training data')
    ax.plot(training_data[:, 0], pred[:], label='Prediction')
```

```
    plt.legend()
    plt.show()

plot_rms()
plot_performance(np.exp(-5))
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize


def load_data():

    np.random.seed(10)

    #Load data from text file
    irises = np.loadtxt('iris.data',delimiter=',',dtype='str')

    irises_inputs = np.array(irises[:,0:4])
    irises_inputs = irises_inputs.astype(np.float)

    #Code string to float
    irises_outputs = irises[:,4]
    irises_outputs[np.where(irises_outputs[:] == r'Iris-setosa')] = str(0)
    irises_outputs[np.where(irises_outputs[:] == r'Iris-versicolor')] = str(1)
    irises_outputs[np.where(irises_outputs[:] == r'Iris-virginica')] = str(2)
    irises_outputs = np.reshape(irises_outputs.astype(np.int32),newshape=(np.shape(irises_outp
uts)[0],1))

    #Add column of ones to the input data
    irises_inputs = np.concatenate((np.ones(shape=(np.shape(irises_inputs)[0],1)),irises_input
s),axis=1)

    #One hot encoding
    irises_labels = np.zeros(shape=(np.shape(irises_outputs)[0],3),dtype='double')
    mask = irises_outputs[:,0]

    for i in range(np.shape(irises_labels)[0]):
        irises_labels[i,mask[i]] = 1.0

    #Segregate into training and test
    idx = np.arange(0, np.shape(irises_inputs)[0], 2)
    training_inputs = irises_inputs[idx, :]
    training_labels = irises_labels[idx, :]

    idx = np.arange(1, np.shape(irises_inputs)[0], 2)
    test_inputs = irises_inputs[idx, :]
    test_labels = irises_labels[idx, :]

    return training_inputs, training_labels, test_inputs, test_labels


def multiclass_logistic_regression():

    global training_labels, training_inputs
    global test_labels, test_inputs

    num_classes = np.shape(training_labels)[1]
    num_features = np.shape(training_inputs)[1]

    weights = np.ones(shape=(num_features*num_classes),dtype='double')

    def softmax_error(weights):
        global training_labels, training_inputs

        # Finding softmax transformation
        a1 = np.reshape(np.sum(weights[0:5] * training_inputs[:, :], axis=1),
                        newshape=(np.shape(training_inputs)[0], 1))
        a2 = np.reshape(np.sum(weights[5:10] * training_inputs[:, :], axis=1),
```

```
                        newshape=(np.shape(training_inputs)[0], 1))
        a3 = np.reshape(np.sum(weights[10:15] * training_inputs[:, :], axis=1),
                        newshape=(np.shape(training_inputs)[0], 1))

        amat = np.concatenate((a1, a2, a3), axis=1)
        ymat = np.copy(amat)

        ymat[:, 0] = np.exp(amat[:, 0]) / (np.exp(amat[:, 0]) + np.exp(amat[:, 1]) + np.exp(am
at[:, 2]))
        ymat[:, 1] = np.exp(amat[:, 1]) / (np.exp(amat[:, 0]) + np.exp(amat[:, 1]) + np.exp(am
at[:, 2]))
        ymat[:, 2] = np.exp(amat[:, 2]) / (np.exp(amat[:, 0]) + np.exp(amat[:, 1]) + np.exp(am
at[:, 2]))

        #Prior for stabilization
        alpha = np.exp(-5)
        prior_val = alpha*np.sum(weights**2,axis=0)

        #Finding error function - Equation 4.108 - Bishop
        softmax_error_val = prior_val-np.sum(training_labels[:, 0] * np.log(ymat[:, 0]) + trai
ning_labels[:, 1] * np.log(
            ymat[:, 1]) + training_labels[:, 2] * np.log(ymat[:, 2]),axis=0)
        return softmax_error_val

    w_hat = minimize(softmax_error,weights,options={'disp':True}).x


    #Prediction on testing data
    # Finding softmax transformation
    z1 = np.reshape(np.sum(w_hat[0:5] * training_inputs[:, :], axis=1),
                    newshape=(np.shape(training_inputs)[0], 1))
    z2 = np.reshape(np.sum(w_hat[5:10] * training_inputs[:, :], axis=1),
                    newshape=(np.shape(training_inputs)[0], 1))
    z3 = np.reshape(np.sum(w_hat[10:15] * training_inputs[:, :], axis=1),
                    newshape=(np.shape(training_inputs)[0], 1))

    zmat = np.concatenate((z1, z2, z3), axis=1)
    smat = np.copy(zmat)

    smat[:, 0] = np.exp(zmat[:, 0]) / (np.exp(zmat[:, 0]) + np.exp(zmat[:, 1]) + np.exp(zmat[:
, 2]))
    smat[:, 1] = np.exp(zmat[:, 1]) / (np.exp(zmat[:, 0]) + np.exp(zmat[:, 1]) + np.exp(zmat[:
, 2]))
    smat[:, 2] = np.exp(zmat[:, 2]) / (np.exp(zmat[:, 0]) + np.exp(zmat[:, 1]) + np.exp(zmat[:
, 2]))

    classification_pred = np.argmax(smat,axis=1)
    classification_true = np.argmax(test_labels, axis=1)

    correct = 0
    for i in range(np.shape(classification_true)[0]):
        if classification_true[i] == classification_pred[i]:
            correct = correct + 1

    print('Accuracy of logistic regression:',100.0*correct/np.shape(classification_pred)[0],'%
')



training_inputs, training_labels, test_inputs, test_labels = load_data()
multiclass_logistic_regression()
```