Programming Spam classification using logistic regression Consider the email spam data set discussed on p300 of (Hastie et al. 2009). This consists of 4601 email messages, from which 57 features have been extracted. The feature descriptions are shown in Fig. 1. Please do the following tasks:

- Load the data (spamData.mat for Matlab, spam.data for Python), which contains a training set (of size 3065) and a test set (of size 1536).
- Feature normalization: standardize each columns so they all have mean 0 and unit variance.
- Fit a standard logistic regression model (without regularization).
- Fit a logistic regression model with ℓ_2 regularization, and use cross validation to choose the tradeoff parameter λ of the ℓ_2 regularizer.
- Fit a logistic regression model with ℓ_1 regularization, and use cross validation to choose the tradeoff parameter λ of the ℓ_1 regularizer. (Optional)

Q7

1-2 Load the data and choose training set, Nominalization

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

all_data = pd.read_csv("spam.data", header=None, sep=" ")
sample_training=all_data.sample(n=3056)
x_training = sample_training.iloc[:, 0:57].to_numpy()
y_training = sample_training.iloc[:, 57].to_numpy()
sample_testing=all_data.drop(sample_training.index)

x_testing = sample_testing.iloc[:, 0:57].to_numpy()
y_testing = sample_testing.iloc[:, 57].to_numpy()

# regulization
X_tr = (x_training-x_training.mean())/x_training.std()
X_ts = (x_testing-x_testing.mean())/x_testing.std()
X_tr = X_tr.to_numpy();
X_ts = X_ts.to_numpy()
```

```
normalized_X = traindatafe      #normalized data
normalized_X2 = train_rep2_fe
Y = np.maximum(train_labels, 0)  #change labels to 1:0 config

costgraph=[]

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def hypothesis(w, X):
    z = np.array(w[0] + w[1]*np.array(X[:,0]) + w[2]*np.array(X[:,1])) #linear
model dot product
    return sigmoid(z)

def cost(w, X, Y):
    y_pred = hypothesis(w, X)
    observations = len(Y)
    return  sum(-Y*np.log(y_pred) - (1-Y)*np.log(1-y_pred)) / observations

def partial_derivative(w, X, Y):
    y_pred = hypothesis(w, X)                         #1 - Get Predictions
    d = [0, 0, 0]
    d[0] = -1 * sum(Y*(1-y_pred) - (1-Y)*y_pred)
    d[1] = -1 * sum(Y*(1-y_pred)*X[:,0] - (1-Y)*y_pred*X[:,0])
    d[2] = -1 * sum(Y*(1-y_pred)*X[:,1] - (1-Y)*y_pred*X[:,1])
    return d

def train_model(w, lr, X, Y):
    costp=cost(w, X, Y)
    i=0
    wnew=[0, 0, 0]
    while True:
        costp=cost(wnew, X, Y)  # updating the value
        costgraph.append(costp) #adding cost for graph
        w = wnew
        w0 = w[0] - lr*partial_derivative(w, X, Y)[0]
        w1 = w[1] - lr*partial_derivative(w, X, Y)[1]
        w2 = w[2] - lr*partial_derivative(w, X, Y)[2]
        wnew = [w0, w1, w2]
        #print(wnew)
        #print("cost",cost(wnew, X, Y))
        costn=cost(wnew, X, Y)
        #print("abs",abs(costn-costp))                        # you may check
convergence steps
         if abs(costn-costp)<pow(10,-4):
            print("Completed at",i,"steps")
            return wnew ,costn
        if i>2000:                                            # if there is no
converge iteration num.
            print(i,"failure")
            return wnew ,costn
        i+=1


costgraph=[]
w = [0.3, 0.4, 0.5]
w1 = train_model(w, 0.005, x_training, y_training)[0]
costgraph1=costgraph
costgraph=[]
w2 = train_model()[0]
costgraph2=costgraph
costgraph=[]
w3 = train_model(w, 0.0005, x_training, y_training)[0]
costgraph3=costgraph
costgraph=[]
w4 = train_model(w, 0.0002, x_training, y_training)[0]
costgraph4=costgraph
costgraph=[]
w5 = train_model(w, 0.0001, x_training, y_training)[0]
costgraph5=costgraph

plt.plot(costgraph1,label='lr=0.005')
plt.plot(costgraph2,label='lr=0.001')
plt.plot(costgraph3,label='lr=0.0005')
plt.plot(costgraph4,label='lr=0.0002')
plt.plot(costgraph5,label='lr=0.0001')
plt.title(" Representation 1 Training")
plt.ylabel("cost function in log scale")
plt.yscale('log')
plt.xlabel("number of steps")
plt.legend()
plt.show()
w
```

Q4: L2

```
costgraph=[]

def costReg(w, X, Y,lambd):
    y_pred = hypothesis(w,X)
    obsv=len(Y)
    return -1 * sum(Y*np.log(y_pred) + (1-Y)*np.log(1-y_pred)) /obsv  -
(lambd)*np.linalg.norm(w,ord=2) /obsv

def partial_derivative2(w, X, Y,lmb):
    y_pred = hypothesis(w,X)                       #1 - Get Predictions
    g = [0]*3
    g[0] = -1 * sum(Y*(1-y_pred) - (1-Y)*y_pred) - 2* w[0] * lmb
    g[1] = -1 * sum(Y*(1-y_pred)*X[:,0] - (1-Y)*y_pred*X[:,0]) - 2* w[1] * lmb
    g[2] = -1 * sum(Y*(1-y_pred)*X[:,1] - (1-Y)*y_pred*X[:,1]) - 2* w[2] * lmb
    return g

def train_model2(w,lr,X,Y,lmb):
    costp=costReg(w, X,Y, lmb)
    i=0
    wnew=[0,0,0]
    while True:
        costp=costReg(wnew, X, Y,lmb) # updating the value
        costgraph.append(costp) #adding cost for graph
        w = wnew
        w0 = w[0] - lr*partial_derivative2(w, X, Y,lmb)[0]
        w1 = w[1] - lr*partial_derivative2(w, X, Y,lmb)[1]
        w2 = w[2] - lr*partial_derivative2(w, X, Y,lmb)[2]
        wnew = [w0, w1, w2]
        #print(wnew)
        #print("cost",cost(wnew, X, Y))
        costn=costReg(wnew, X, Y,lmb)
        #print("abs",abs(costn-costp))                        # you may check
convergence steps
        if abs(costn-costp)<pow(10,-4):
            print("Completed at",i,"steps")
            return wnew, costn
        if i>2000:                                            # if there is no
converge iteration num.
            print(i,"failure")
            return wnew , costn
        i+=1
```

```
costgraph=[]
w = [0.3, 0.4, 0.5]
w1r = train_model2(w, 0.005, X_tr, X_ts,0.01)[0]
costgraph11=costgraph

plt.plot(costgraph11,label='lr=0.0005,lambda= 0.01')
plt.title(" Representation 1 Training with Regularization")
plt.ylabel("cost function in log scale")
plt.yscale('log')
plt.xlabel("number of steps")
plt.legend()
plt.show()
```

```
#Cross Validation
def cv(X,Y,lr,lamb):
    Z=np.column_stack([X,Y])
    np.random.shuffle(Z)
    totaldata=np.split(Z[0:3056],5)
    cvaccarray=[]
    for i in range(5):
        cvtestdata=totaldata[i]
        cvtraindata= np.concatenate(np.delete(totaldata,i,0),axis=0)
        cvweight,cvcost = train_model2(w,lr,cvtraindata[:,0:57],
cvtraindata[:,57],lamb)
        cvaccarray.append(test_cv(cvweight,cvtestdata[:,0:57],cvtestdata[:,57]))
    return np.mean(cvaccarray),np.std(cvaccarray)
```