NNFL (BITS F312) Assignment 1

2016B5A30572H
K Pranath Reddy

Question 1

1. Implement the linear regression algorithm to estimate the weight parameters for the feature matrix (X) and the class label vector (y). (a) Plot the cost function vs the number of iterations. (b) Plot the cost function (J) vs w1 and w2 in a contour or 3D surf graph (w= [w0 w1 w2]). Please use the dataset "data.xlsx". (Use for or while loop for the implementation)

Solution :

Code :

```
'''

***Multivariate Linear Regression***

With Batch Gradient Descent



Author :

Pranath Reddy

2016B5A30572H

'''

import pandas as pd

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D
```

```python
# A function to return the column at specified index

def getcol(data,c):

    col = []

    for i in range(len(data)):

        col.append(data[i][c])

    return col



# A function to return the updated values of m,c after one iteration of gradient
descent

# weight update rule

def wtupdate(m1,m2,c,x1,x2,y):

    sumvm1 = 0

    sumvm2 = 0

    sumvc = 0

    yp = [0 for i in range(len(x1))]

    cost = 0

    lrate = 0.001

    for i in range(len(x1)):

        yp[i] = (m1*x1[i]) + (m2*x2[i]) + c

        sumvm1 = sumvm1 - (y[i]-yp[i])*x1[i]

        sumvm2 = sumvm2 - (y[i]-yp[i])*x2[i]

        sumvc = sumvc - (y[i]-yp[i])

    m1 = m1 - lrate*sumvm1
```

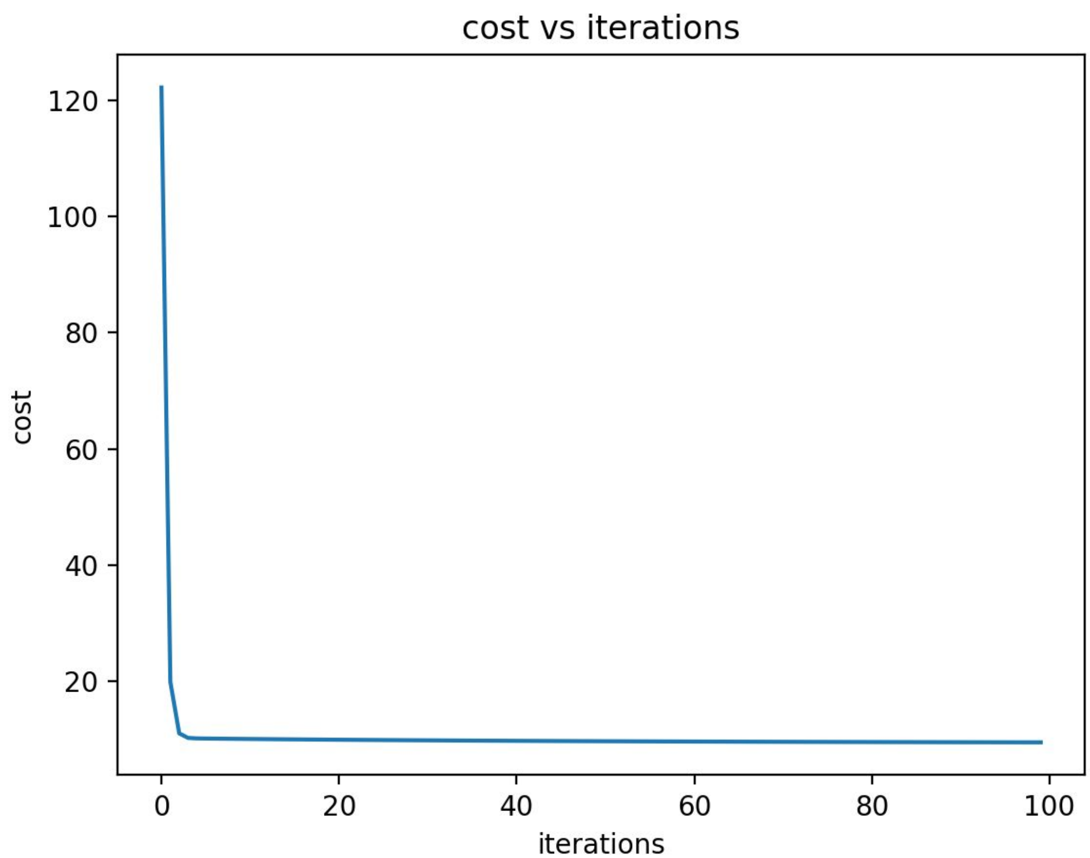
```
    m2 = m2 - lrate*sumvm2

    c = c - lrate*sumvc

    cost = costfn(yp,y)

    return m1,m2,c,cost


# A function for calculate the cost

def costfn(yp,y):

    j = 0

    scale = len(yp)

    for i in range(len(y)):

        j = j + float((yp[i]-y[i]))**2

    return j*0.5*(1/scale)


# A function to return the slope and intercept of y^

def linreg(x1,x2,y):

    m1 = 0

    m2 = 0

    c = 0

    iters = 100

    cost = [0 for i in range(iters)]

    m1list = [0 for i in range(iters)]

    m2list = [0 for i in range(iters)]
```

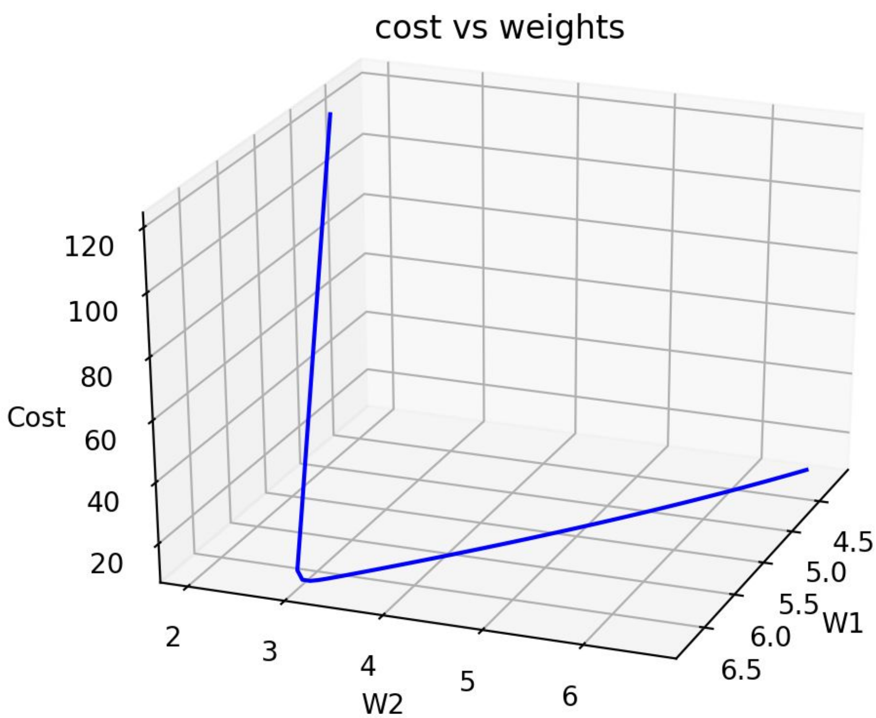
```
    i = 0

    while(i<iters):

        m1,m2,c,costval = wtupdate(m1,m2,c,x1,x2,y)

        cost[i] = costval

        m1list[i] = m1

        m2list[i] = m2

        i = i+1

    return m1,m2,c,cost,m1list,m2list



# A function to implement min-max normalization

def norm(data):

    ndata = data

    for i in range(2):

        maxval = max(getcol(data,i))

        minval = min(getcol(data,i))

        for j in range(len(data)):

            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)

    return ndata



# import the data

data = pd.read_excel('data.xlsx',header=None)

# normalize the data
```

```
data = np.asarray(data)

data = norm(data)



# split into dependent and independent variables

x1 = data[:,0]

x2 = data[:,1]

y = data[:,-1]



# run the linear regression

m1,m2,c,cost,m1list,m2list = linreg(x1,x2,y)



plt.plot(cost)

plt.title("cost vs iterations")

plt.xlabel("iterations")

plt.ylabel("cost")

plt.show()



fig = plt.figure()

ax = plt.axes(projection='3d')

ax.plot3D(m1list, m2list, cost, 'blue')

ax.set_xlabel('W1')

ax.set_ylabel('W2')
```

```
ax.set_zlabel('Cost')

ax.set_title('cost vs weights')

fig.show()

plt.show()
```

Results :





Question 2

2. Implement stochastic gradient descent for the linear regression problem in question number 1. (a) Plot the cost function vs the number of iterations. (b) Plot the cost function vs w1 and w2. (Please use the dataset “data.xlsx”). (Use for or while loop for the implementation)

Solution :

Code :

```
'''

***Multivariate Linear Regression***

With Stochastic Gradient Descent
```

```python
Author :
Pranath Reddy
2016B5A30572H
'''
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt
import random
from mpl_toolkits.mplot3d import Axes3D


# A function to return the column at specified index
def getcol(data,c):
    col = []
    for i in range(len(data)):
        col.append(data[i][c])
    return col


# A function to return the updated values of m,c after one iteration of gradient descent
# weight update rule
def wtupdate(m1,m2,c,x1,x2,y,i):
    sumvm1 = 0
```

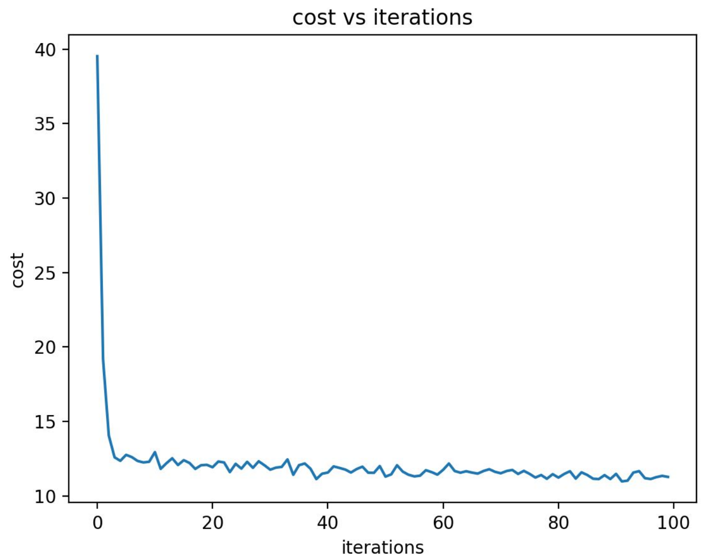
```python
    sumvm2 = 0

    sumvc = 0

    lrate = 0.001

    yp = [0 for i in range(len(x1))]

    yp[i] = (m1*x1[i]) + (m2*x2[i]) + c

    sumvm1 = sumvm1 - (y[i]-yp[i])*x1[i]

    sumvm2 = sumvm2 - (y[i]-yp[i])*x2[i]

    sumvc = sumvc - (y[i]-yp[i])

    m1 = m1 - lrate*sumvm1

    m2 = m2 - lrate*sumvm2

    c = c - lrate*sumvc

    return m1,m2,c


# A function for calculate the cost

def costfn(yp,y):

    j = 0

    scale = len(yp)

    for i in range(len(y)):

        j = j + float((yp[i]-y[i]))**2

    return j*0.5*(1/scale)


# A function to return the slope and intercept of y^
```

```python
def linreg(x1,x2,y):

    m1 = 0

    m2 = 0

    c = 0

    iters = 100

    cost = []

    m1list = []

    m2list = []

    j = 0

    yp = [0 for i in range(len(x1))]

    y_temp = y

    while(j<iters):

        for i in range(len(y)):

            random.shuffle(y_temp)

            m1,m2,c = wtupdate(m1,m2,c,x1,x2,y_temp,i)

            for i in range(len(x1)):

                yp[i] = (m1*x1[i]) + (m2*x2[i]) + c

        cost.append(costfn(yp,y))

        m1list.append(m1)

        m2list.append(m2)

        j = j+1

    return m1,m1list,m2,m2list,c,cost
```

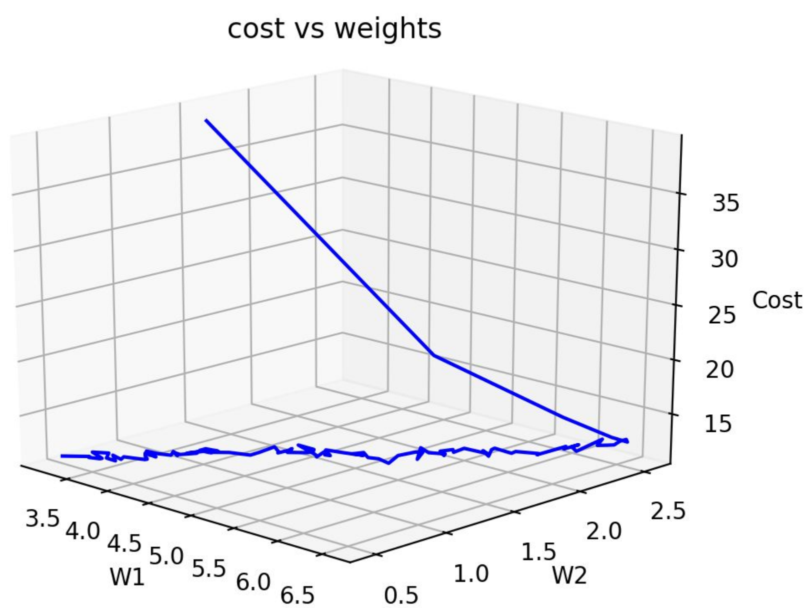
```python
# A function to implement min-max normalization

def norm(data):

    ndata = data

    for i in range(2):

        maxval = max(getcol(data,i))

        minval = min(getcol(data,i))

        for j in range(len(data)):

            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)

    return ndata



# import the data

data = pd.read_excel('data.xlsx',header=None)

# normalize the data

data = np.asarray(data)

data = norm(data)



# split into dependent and independent variables

x1 = data[:,0]

x2 = data[:,1]

y = data[:,-1]
```

```python
#run the linear regression

m1,m1list,m2,m2list,c,cost = linreg(x1,x2,y)



plt.plot(cost)

plt.title("cost vs iterations")

plt.xlabel("iterations")

plt.ylabel("cost")

plt.show()



fig = plt.figure()

ax = plt.axes(projection='3d')

ax.plot3D(m1list, m2list, cost, 'blue')

ax.set_xlabel('W1')

ax.set_ylabel('W2')

ax.set_zlabel('Cost')

ax.set_title('cost vs weights')

fig.show()

plt.show()
```

Results :



cost vs iterations
40
35
30
25
20
15
10
cost
0
20
40
60
80
100
iterations



cost vs weights
35
30
25
20
15
Cost
3.5
4.0
4.5
5.0
5.5
6.0
6.5
W1
0.5
1.0
1.5
2.0
2.5
W2

Question 3

3. Implement the ridge regression problem by considering both batch gradient descent and stochastic gradient descent. (a) Plot the cost function vs the number of iterations for both the cases. (b) Plot the cost function (J) vs w1 and w2 in a contour or 3D surf graph for both the cases. (Please use the dataset "data.xlsx"). (Use for or while loop for the implementation)

Solution :

Code :

```
'''

***Multivariate Linear Regression***

With Batch Gradient Descent and L2 norm regularization



Author :

Pranath Reddy

2016B5A30572H

'''



import math

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D



# A function to return the column at specified index

def getcol(data,c):
```

```python
    col = []
    for i in range(len(data)):
        col.append(data[i][c])
    return col


# A function to return the updated values of m,c after one iteration of gradient descent
# weight update rule
def wtupdate(m1,m2,c,x1,x2,y):
    sumvm1 = 0
    sumvm2 = 0
    sumvc = 0
    yp = [0 for i in range(len(x1))]
    cost = 0
    lrate = 0.001
    reg = 0.2
    for i in range(len(x1)):
        yp[i] = (m1*x1[i]) + (m2*x2[i]) + c
        sumvm1 = sumvm1 - (y[i]-yp[i])*x1[i]
        sumvm2 = sumvm2 - (y[i]-yp[i])*x2[i]
        sumvc = sumvc - (y[i]-yp[i])
    m1 = m1*(1-lrate*reg) - lrate*sumvm1
    m2 = m2*(1-lrate*reg) - lrate*sumvm2
```

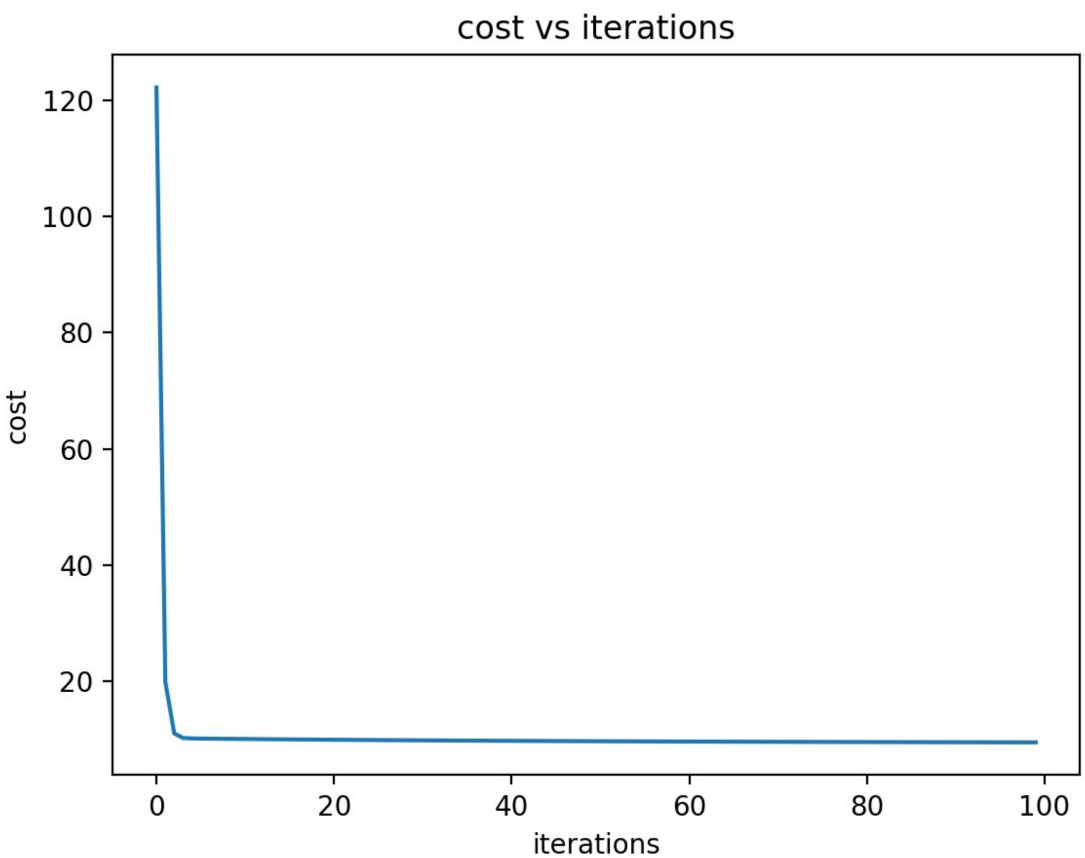
```python
    c = c*(1-lrate*reg) - lrate*sumvc

    cost = costfn(yp,y,m1,m2,c)

    return m1,m2,c,cost


# A function for calculate the cost

def costfn(yp,y,m1,m2,c):

    j = 0

    scale = len(yp)

    reg = 0.2

    for i in range(len(y)):

        j = j + float((yp[i]-y[i]))**2

    j = j + (reg/2)*(m1*m1+m2*m2+c*c)

    return j*0.5*(1/scale)


# A function to return the slope and intercept of y^

def linreg(x1,x2,y):

    m1 = 0

    m2 = 0

    c = 0

    iters = 100

    cost = [0 for i in range(iters)]

    m1list = [0 for i in range(iters)]
```

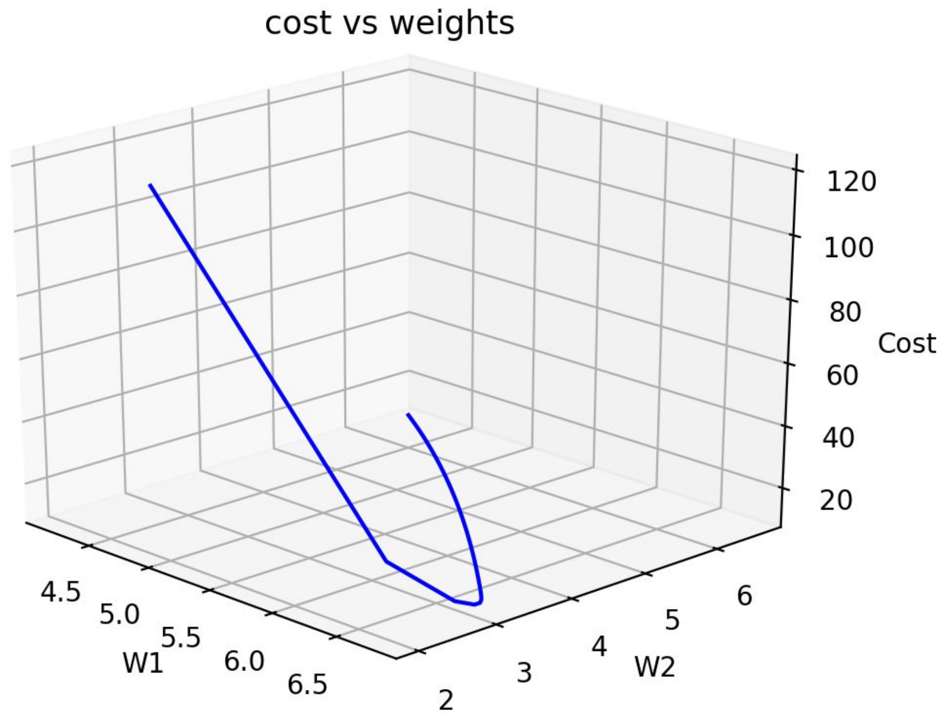
```python
    m2list = [0 for i in range(iters)]
    i = 0
    while(i<iters):
        m1,m2,c,costval = wtupdate(m1,m2,c,x1,x2,y)
        cost[i] = costval
        m1list[i] = m1
        m2list[i] = m2
        i = i+1
    return m1,m2,c,cost,m1list,m2list


# A function to implement min-max normalization
def norm(data):
    ndata = data
    for i in range(2):
        maxval = max(getcol(data,i))
        minval = min(getcol(data,i))
        for j in range(len(data)):
            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)
    return ndata


# import the data
data = pd.read_excel('data.xlsx',header=None)
```

```
# normalize the data

data = np.asarray(data)

data = norm(data)



# split into dependent and independent variables

x1 = data[:,0]

x2 = data[:,1]

y = data[:,-1]



# run the linear regression

m1,m2,c,cost,m1list,m2list = linreg(x1,x2,y)




plt.plot(cost)

plt.title("cost vs iterations")

plt.xlabel("iterations")

plt.ylabel("cost")

plt.show()




fig = plt.figure()

ax = plt.axes(projection='3d')

ax.plot3D(m1list, m2list, cost, 'blue')

ax.set_xlabel('W1')
```

```
ax.set_ylabel('W2')

ax.set_zlabel('Cost')

ax.set_title('cost vs weights')

fig.show()

plt.show()
```

Results :






```
'''

***Multivariate Linear Regression***

With Stochastic Gradient Descent and L2 norm regularization



Author :

Pranath Reddy

2016B5A30572H

'''

import pandas as pd
```

```
import math

import numpy as np

import matplotlib.pyplot as plt

import random

from mpl_toolkits.mplot3d import Axes3D



# A function to return the column at specified index

def getcol(data,c):

    col = []

    for i in range(len(data)):

        col.append(data[i][c])

    return col



# A function to return the updated values of m,c after one iteration of gradient
descent

# weight update rule

def wtupdate(m1,m2,c,x1,x2,y,i):

    sumvm1 = 0

    sumvm2 = 0

    sumvc = 0

    lrate = 0.001

    reg = 0.2

    yp = [0 for i in range(len(x1))]
```

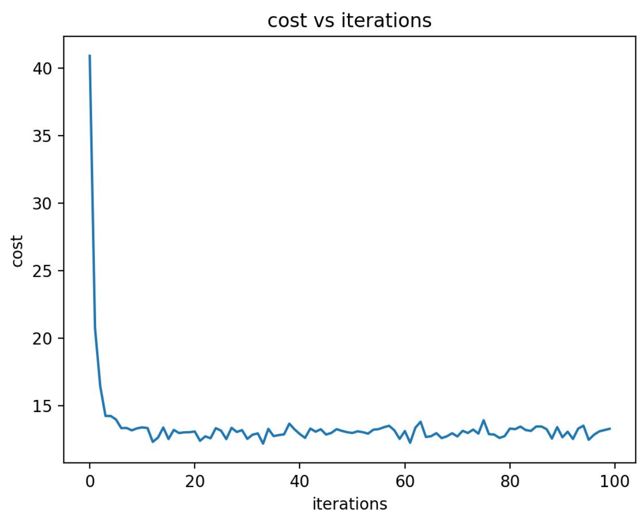
```
    yp[i] = (m1*x1[i]) + (m2*x2[i]) + c

    sumvm1 = sumvm1 - (y[i]-yp[i])*x1[i]

    sumvm2 = sumvm2 - (y[i]-yp[i])*x2[i]

    sumvc = sumvc - (y[i]-yp[i])

    m1 = m1*(1-lrate*reg) - lrate*sumvm1

    m2 = m2*(1-lrate*reg) - lrate*sumvm2

    c = c*(1-lrate*reg) - lrate*sumvc

    return m1,m2,c


# A function for calculate the cost

def costfn(yp,y,m1,m2,c):

    j = 0

    scale = len(yp)

    reg = 0.2

    for i in range(len(y)):

        j = j + float((yp[i]-y[i]))**2

    j = j + (reg/2)*(m1*m1+m2*m2+c*c)

    return j*0.5*(1/scale)


# A function to return the slope and intercept of y^

def linreg(x1,x2,y):

    m1 = 0
```

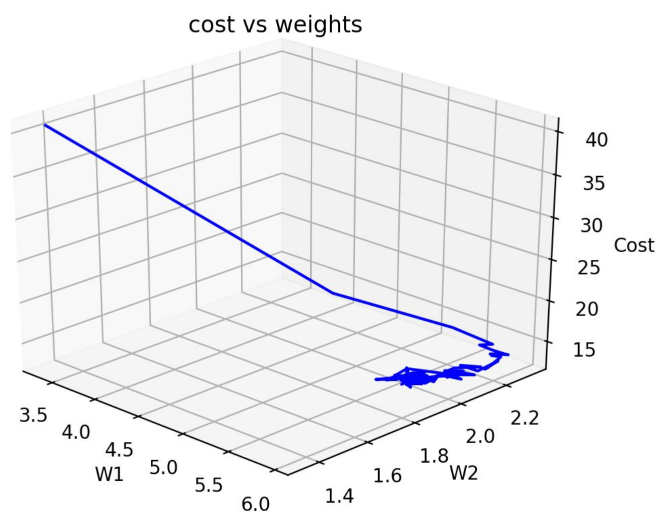
```
    m2 = 0

    c = 0

    iters = 100

    cost = []

    m1list = []

    m2list = []

    j = 0

    yp = [0 for i in range(len(x1))]

    y_temp = y

    while(j<iters):

        for i in range(len(y)):

            random.shuffle(y_temp)

            m1,m2,c = wtupdate(m1,m2,c,x1,x2,y_temp,i)

            for i in range(len(x1)):

                yp[i] = (m1*x1[i]) + (m2*x2[i]) + c

        cost.append(costfn(yp,y,m1,m2,c))

        m1list.append(m1)

        m2list.append(m2)

        j = j+1

    return m1,m1list,m2,m2list,c,cost



# A function to implement min-max normalization
```

```python
def norm(data):

    ndata = data

    for i in range(2):

        maxval = max(getcol(data,i))

        minval = min(getcol(data,i))

        for j in range(len(data)):

            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)

    return ndata



# import the data

data = pd.read_excel('data.xlsx',header=None)

# normalize the data

data = np.asarray(data)

data = norm(data)



# split into dependent and independent variables

x1 = data[:,0]

x2 = data[:,1]

y = data[:,-1]



#run the linear regression

m1,m1list,m2,m2list,c,cost = linreg(x1,x2,y)
```

```
plt.plot(cost)

plt.title("cost vs iterations")

plt.xlabel("iterations")

plt.ylabel("cost")

plt.show()



fig = plt.figure()

ax = plt.axes(projection='3d')

ax.plot3D(m1list, m2list, cost, 'blue')

ax.set_xlabel('W1')

ax.set_ylabel('W2')

ax.set_zlabel('Cost')

ax.set_title('cost vs weights')

fig.show()

plt.show()
```

Results :


cost vs iterations
cost
iterations
40
35
30
25
20
15
0
20
40
60
80
100


cost vs weights
W1
W2
Cost
3.5
4.0
4.5
5.0
5.5
6.0
1.4
1.6
1.8
2.0
2.2
40
35
30
25
20
15

Question 4

4. Implement the Vectorized linear regression problem to evaluate the weight parameters for question number 1. Compare the weight parameters with the weights obtained using both gradient descent and stochastic gradient descent based algorithms. (Please use the dataset “data.xlsx”).

Solution :

Code :

```
'''

***Multivariate Linear Regression***

Vector implementation



Author :

Pranath Reddy

2016B5A30572H

'''

import pandas as pd

import math

import numpy as np

from numpy.linalg import inv

import matplotlib.pyplot as plt



# A function to return the column at specified index

def getcol(data,c):

    col = []
```

```
    for i in range(len(data)):
        col.append(data[i][c])
    return col

# A function to implement min-max normalization
def norm(data):
    ndata = data
    for i in range(2):
        maxval = max(getcol(data,i))
        minval = min(getcol(data,i))
        for j in range(len(data)):
            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)
    return ndata

# import the data
data = pd.read_excel('data.xlsx',header=None)
# normalize the data
data = np.asarray(data)
data = norm(data)
# split into dependent and independent variables
x = data[:,:-1]
len = len(x)
```

```
x_temp = np.ones((len,1))

x = np.append(x_temp, x, axis=1)

y = data[:,-1]



w = 0

w = inv(np.dot(np.transpose(x),x))

w = np.dot(w,np.transpose(x))

w = np.dot(w,y)

print("The weight vector is : " + str(w))
```

Result :

The weight vector is : [10.19758757 1.80456645 8.22286455]

Question 5

5. Implement Least angle regression to estimate the weight parameters for the feature matrix (X) and the class label vector (y) by considering both gradient descent and stochastic gradient descent based algorithms. (Please use the dataset “data.xlsx”). (Use for or while loop for the implementation).

Solution :

Code :

```
'''

***Multivariate Linear Regression***

With Batch Gradient Descent and L1 norm regularization



Author :

Pranath Reddy
```

```
2016B5A30572H

'''

import pandas as pd

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D



# A function to return the column at specified index

def getcol(data,c):

    col = []

    for i in range(len(data)):

        col.append(data[i][c])

    return col



def sgn(x):

    if(x == 0):

        return 0

    else:

        sgn = float(abs(x)/x)

        return sgn
```

```python
# A function to return the updated values of m,c after one iteration of gradient descent

# weight update rule

def wtupdate(m1,m2,c,x1,x2,y):

    sumvm1 = 0

    sumvm2 = 0

    sumvc = 0

    yp = [0 for i in range(len(x1))]

    cost = 0

    lrate = 0.001

    reg = 0.2

    for i in range(len(x1)):

        yp[i] = (m1*x1[i]) + (m2*x2[i]) + c

        sumvm1 = sumvm1 - (y[i]-yp[i])*x1[i]

        sumvm2 = sumvm2 - (y[i]-yp[i])*x2[i]

        sumvc = sumvc - (y[i]-yp[i])

    m1 = m1 - lrate*(sumvm1 + reg*0.5*sgn(m1))

    m2 = m2 - lrate*(sumvm2 + reg*0.5*sgn(m2))

    c = c - lrate*(sumvc + reg*0.5*sgn(c))

    cost = costfn(yp,y,m1,m2,c)

    return m1,m2,c,cost



# A function for calculate the cost
```

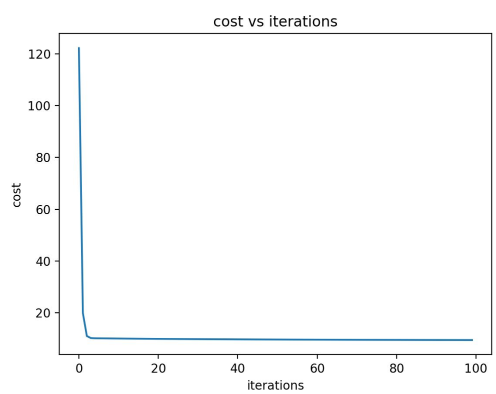
```python
def costfn(yp,y,m1,m2,c):

    j = 0

    scale = len(yp)

    reg = 0.2

    for i in range(len(y)):

        j = j + float((yp[i]-y[i]))**2

    j = j + (reg/2)*(abs(m1)+abs(m2)+abs(c))

    return j*0.5*(1/scale)



# A function to return the slope and intercept of y^

def linreg(x1,x2,y):

    m1 = 0

    m2 = 0

    c = 0

    iters = 100

    cost = [0 for i in range(iters)]

    m1list = [0 for i in range(iters)]

    m2list = [0 for i in range(iters)]

    i = 0

    while(i<iters):

        m1,m2,c,costval = wtupdate(m1,m2,c,x1,x2,y)

        cost[i] = costval
```

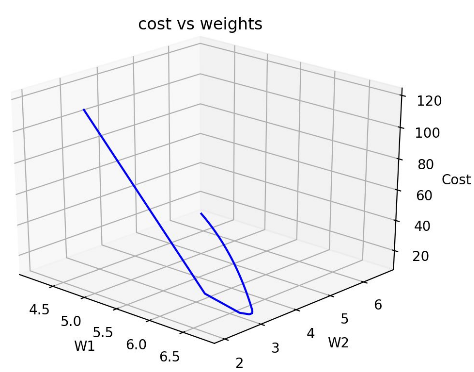
```
        m1list[i] = m1

        m2list[i] = m2

        i = i+1

    return m1,m2,c,cost,m1list,m2list



# A function to implement min-max normalization

def norm(data):

    ndata = data

    for i in range(2):

        maxval = max(getcol(data,i))

        minval = min(getcol(data,i))

        for j in range(len(data)):

            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)

    return ndata



# import the data

data = pd.read_excel('data.xlsx',header=None)

# normalize the data

data = np.asarray(data)

data = norm(data)



# split into dependent and independent variables
```

```
x1 = data[:,0]

x2 = data[:,1]

y = data[:,-1]


# run the linear regression

m1,m2,c,cost,m1list,m2list = linreg(x1,x2,y)


'''

plt.plot(cost)

plt.title("cost vs iterations")

plt.xlabel("iterations")

plt.ylabel("cost")

plt.show()


fig = plt.figure()

ax = plt.axes(projection='3d')

ax.plot3D(m1list, m2list, cost, 'blue')

ax.set_xlabel('W1')

ax.set_ylabel('W2')

ax.set_zlabel('Cost')

ax.set_title('cost vs weights')

fig.show()
```

```
plt.show()

'''

print('The weights are :')

print('W1 : ' + str(m1))

print('W2 : ' + str(m2))

print('W0 : ' + str(c))
```

Results :





The weights are :

W1 : 4.151745301352734

W2 : 6.5884568782801844

W0 : 8.59223419079168

```
'''
***Multivariate Linear Regression***
With Stochastic Gradient Descent and L1 norm regularization

Author :
Pranath Reddy
2016B5A30572H
'''
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt
import random
from mpl_toolkits.mplot3d import Axes3D

# A function to return the column at specified index
```

```python
def getcol(data,c):

    col = []

    for i in range(len(data)):

        col.append(data[i][c])

    return col



def sgn(x):

    if(x == 0):

        return 0

    else:

        sgn = float(abs(x)/x)

        return sgn



# A function to return the updated values of m,c after one iteration of gradient
descent

# weight update rule

def wtupdate(m1,m2,c,x1,x2,y,i):

    sumvm1 = 0

    sumvm2 = 0

    sumvc = 0

    lrate = 0.001

    reg = 0.2

    yp = [0 for i in range(len(x1))]
```

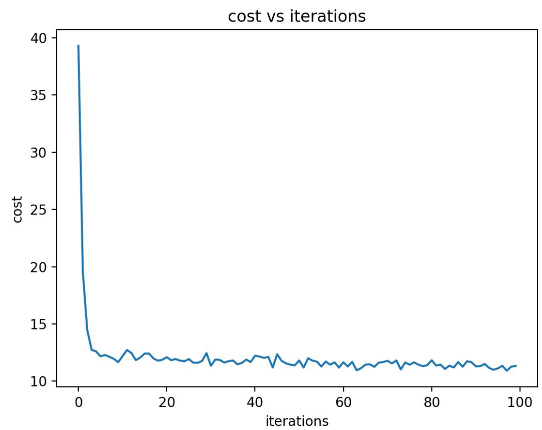
```
        yp[i] = (m1*x1[i]) + (m2*x2[i]) + c
        sumvm1 = sumvm1 - (y[i]-yp[i])*x1[i]
        sumvm2 = sumvm2 - (y[i]-yp[i])*x2[i]
        sumvc = sumvc - (y[i]-yp[i])
    m1 = m1 - lrate*(sumvm1 + reg*0.5*sgn(m1))
    m2 = m2 - lrate*(sumvm2 + reg*0.5*sgn(m2))
    c = c - lrate*(sumvc + reg*0.5*sgn(c))
    return m1,m2,c


# A function for calculate the cost
def costfn(yp,y,m1,m2,c):
    j = 0
    scale = len(yp)
    reg = 0.2
    for i in range(len(y)):
        j = j + float((yp[i]-y[i]))**2
    j = j + (reg/2)*(abs(m1)+abs(m2)+abs(c))
    return j*0.5*(1/scale)


# A function to return the slope and intercept of y^
def linreg(x1,x2,y):
    m1 = 0
```

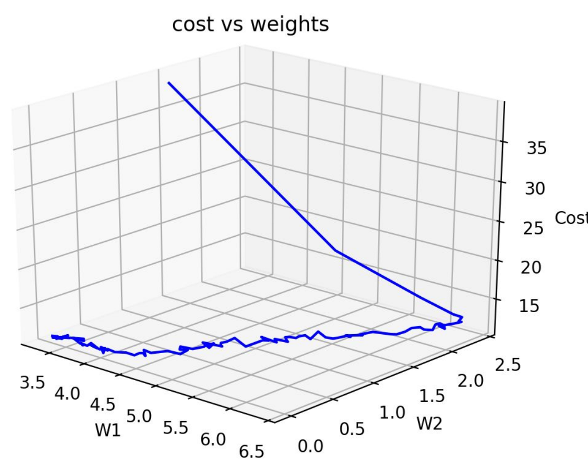
```python
    m2 = 0

    c = 0

    iters = 100

    cost = []

    m1list = []

    m2list = []

    j = 0

    yp = [0 for i in range(len(x1))]

    y_temp = y

    while(j<iters):

        for i in range(len(y)):

            random.shuffle(y_temp)

            m1,m2,c = wtupdate(m1,m2,c,x1,x2,y_temp,i)

            for i in range(len(x1)):

                yp[i] = (m1*x1[i]) + (m2*x2[i]) + c

        cost.append(costfn(yp,y,m1,m2,c))

        m1list.append(m1)

        m2list.append(m2)

        j = j+1

    return m1,m1list,m2,m2list,c,cost



# A function to implement min-max normalization
```

```python
def norm(data):

    ndata = data

    for i in range(2):

        maxval = max(getcol(data,i))

        minval = min(getcol(data,i))

        for j in range(len(data)):

            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)

    return ndata



# import the data

data = pd.read_excel('data.xlsx',header=None)

# normalize the data

data = np.asarray(data)

data = norm(data)



# split into dependent and independent variables

x1 = data[:,0]

x2 = data[:,1]

y = data[:,-1]



#run the linear regression

m1,m1list,m2,m2list,c,cost = linreg(x1,x2,y)
```

```
'''

plt.plot(cost)

plt.title("cost vs iterations")

plt.xlabel("iterations")

plt.ylabel("cost")

plt.show()



fig = plt.figure()

ax = plt.axes(projection='3d')

ax.plot3D(m1list, m2list, cost, 'blue')

ax.set_xlabel('W1')

ax.set_ylabel('W2')

ax.set_zlabel('Cost')

ax.set_title('cost vs weights')

fig.show()

plt.show()

'''

print('The weights are :')

print('W1 : ' + str(m1))

print('W2 : ' + str(m2))

print('W0 : ' + str(c))
```

Results :





The weights are :

W1 : 3.2229229441881926

W2 : 0.016064966007475895

W0 : 11.69356036170692

Question 6

6. Implement K-means clustering based unsupervised learning algorithm for the dataset (“data2.xlsx”). Plot the estimated class labels vs features. Use the number of clusters as K=2.

Solution :

Code :

```
'''

*** K-means clustering ***

two clusters



Author :

Pranath Reddy

2016B5A30572H

'''

import pandas as pd

import cmath as math

import numpy as np

import random

from random import randint

import matplotlib.pyplot as plt
```

```python
# A function to calculate the mean of an array

def mean(val):

    if len(val) == 0:

        return 0

    else:

        return sum(val) / len(val)



# A function to return a column of the data at the specified index

def col(array, i):

    return [row[i] for row in array]



# A function to return the max of two values

def higher(x,y):

    if x>y:

        return x

    else:

        return y



# A function to calculate the distance between two points

def dist(x,y):

    sum = 0

    for a in range(4):
```

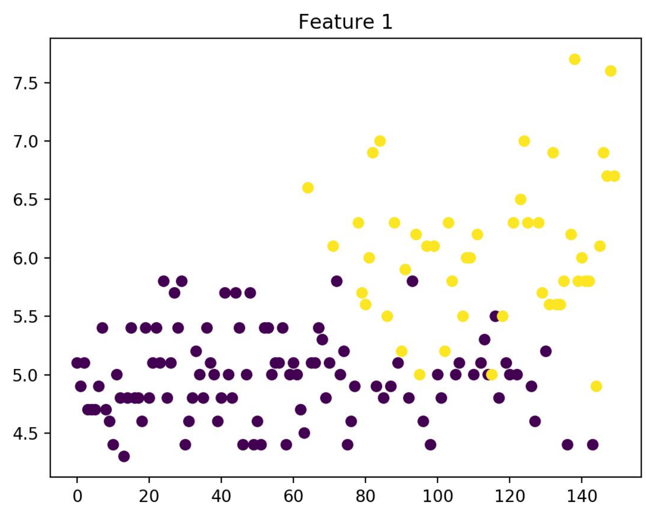
```
        sum = sum + (x[a]-y[a])**2

    return (math.sqrt(sum)).real



# A function to calculate the distances from initialized centroids

def distcen_init(data,cen):

    distc1 = [0 for x in range(len(data))]

    distc2 = [0 for x in range(len(data))]

    for k in range(len(data)):

        distc1[k] = (dist(data[k],data[cen[0]]))

        distc2[k] = (dist(data[k],data[cen[1]]))

    return distc1, distc2



# A function to calculate the distances from centroids

def distcen(data,cen):

    distc1 = [0 for x in range(len(data))]

    distc2 = [0 for x in range(len(data))]

    for k in range(len(data)):

        distc1[k] = (dist(data[k],cen[0]))

        distc2[k] = (dist(data[k],cen[1]))

    return distc1, distc2



def getcol(data,c):
```

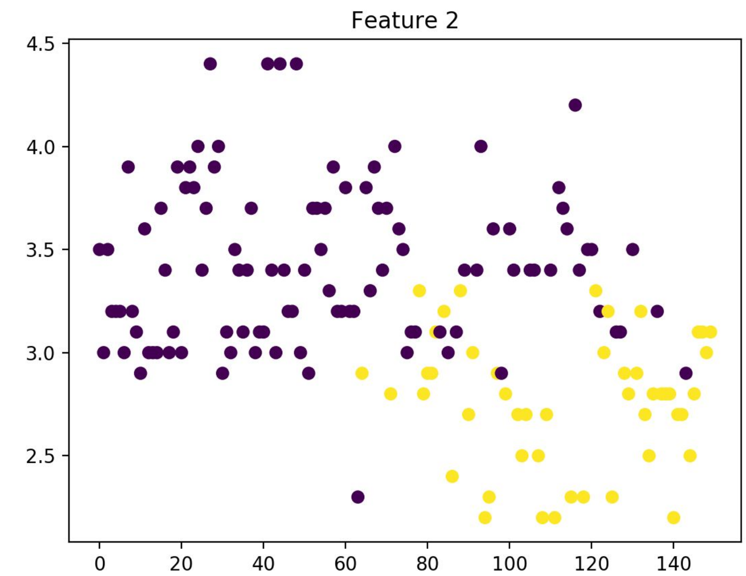
```
    col = []
    for i in range(len(data)):
        col.append(data[i][c])
    return col

# A function to implement min-max normalization
def norm(data):
    ndata = data
    for i in range(4):
        maxval = max(getcol(data,i))
        minval = min(getcol(data,i))
        for j in range(len(data)):
            ndata[j][i] = (data[j][i]-minval)/(maxval-minval)
    return ndata

# import the data
data = pd.read_excel('data2.xlsx',header=None)
# normalize the data
data = np.asarray(data)
data = norm(data)

# initiate the centroids
```

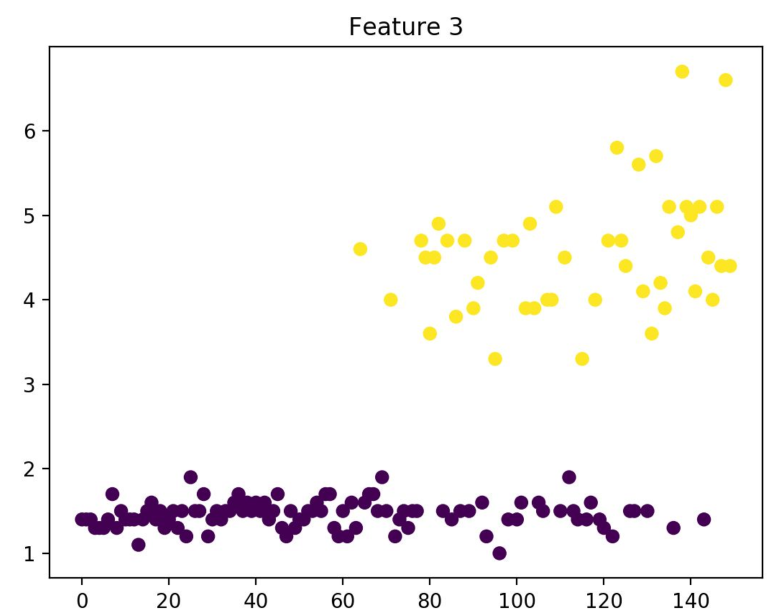
```
randindex = [randint(0, len(data)) for b in range(2)]

dsvc1, dsvc2 = distcen_init(data,randindex)

iters = 50


for j in range(iters):

    # assign cluster indexes

    cval = [1 for x in range(len(data))]

    for l in range(len(data)):

        if dsvc2[l]<dsvc1[l]:

            cval[l] = 2

    # divide into clusters using cluster indexes found above

    clist1 = []

    clist2 = []

    for m in range(len(data)):

        if cval[m] == 1:

            clist1.append(data[m])

        else:

            clist2.append(data[m])

    # update the centroids

    c1 = []

    c2 = []

    for n in range(4):
```

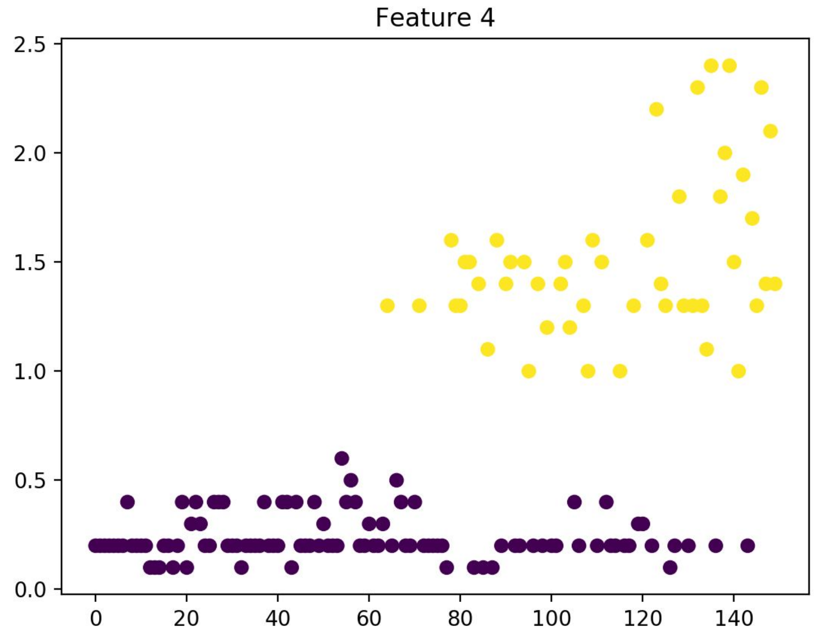
```
        c1.append(mean(col(clist1,n)))

        c2.append(mean(col(clist2,n)))

    cen = [c1,c2]

    # update the distances from centroids

    dsvc1, dsvc2 = distcen(data,cen)



index = [0 for x in range(len(data))]

for i in range(len(data)):

    index[i] = i+1



plt.scatter(np.arange(len(data[:,0])),data[:,0],c=cval)

plt.title('Feature 1')

plt.show()

plt.scatter(np.arange(len(data[:,1])),data[:,1],c=cval)

plt.title('Feature 2')

plt.show()

plt.scatter(np.arange(len(data[:,2])),data[:,2],c=cval)

plt.title('Feature 3')

plt.show()

plt.scatter(np.arange(len(data[:,3])),data[:,3],c=cval)

plt.title('Feature 4')

plt.show()
```

Results :




Feature 2
4.5
4.0
3.5
3.0
2.5
0
20
40
60
80
100
120
140


Feature 3
6
5
4
3
2
1
0
20
40
60
80
100
120
140



Question 7

7. Implement the logistic regression algorithm for the binary classification using the dataset ("data3.xlsx"). Divide the dataset into training and testing using hold-out cross- validation technique with 60 % of instances as training and the remaining 40% as testing. Evaluate the accuracy, sensitivity and specificity values for the binary classifier.

Solution :

Code :

```
'''

***Logistic Regression***

With Gradient Descent



Author :

Pranath Reddy

2016B5A30572H
```

```
'''


import math

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split


# A function to return the column at specified index

def getcol(data,c):

    col = []

    for i in range(len(data)):

        col.append(data[i][c])

    return col


def set(y):

    for i in range(len(y)):

        if(y[i]>0.5):

            y[i] = 1

        if(y[i]<0.5):

            y[i] = 0

    return y
```

```
def sigmoid(x):

    return 1 / (1 + math.exp(-x))



# A function to return the updated values of m,c after one iteration of gradient
descent

def wtupdate(m1,m2,m3,m4,c,x,y):

    sumvm1 = 0

    sumvm2 = 0

    sumvm3 = 0

    sumvm4 = 0

    sumvc = 0

    yp = [0 for i in range(len(x))]

    for i in range(len(x)):

        yp[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + c

        yp[i] = sigmoid(yp[i])

        sumvm1 = sumvm1 - (y[i]-yp[i])*x[i,0]

        sumvm2 = sumvm2 - (y[i]-yp[i])*x[i,1]

        sumvm3 = sumvm3 - (y[i]-yp[i])*x[i,2]

        sumvm4 = sumvm4 - (y[i]-yp[i])*x[i,3]

        sumvc = sumvc - (y[i]-yp[i])

    m1 = m1 - 0.05*sumvm1

    m2 = m2 - 0.05*sumvm2

    m3 = m3 - 0.05*sumvm3
```

```
    m4 = m4 - 0.05*sumvm4

    c = c - 0.05*sumvc

    return m1,m2,m3,m4,c


# A function to return the slope and intercept of y^

def linreg(x,y):

    m1 = 0

    m2 = 0

    m3 = 0

    m4 = 0

    c = 0

    iters = 1000

    i = 0

    while(i<iters):

        m1,m2,m3,m4,c = wtupdate(m1,m2,m3,m4,c,x,y)

        i = i+1

    return m1,m2,m3,m4,c


# A function to implement min-max normalization

def norm(data):

    ndata = data

    for i in range(5):
```

```
        maxval = max(getcol(data,i))

        minval = min(getcol(data,i))

        for j in range(len(data)):

            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)

    return ndata



# import the data

data = pd.read_excel('data3.xlsx',header=None)

# normalize the data

data = np.asarray(data)

data = norm(data)



# split into dependent and independent variables

x = data[:,:-1]

y = data[:,-1]



# split into testing and training sets

x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.4)

m1,m2,m3,m4, c = linreg(x_tr,y_tr)

x = x_ts

yp = [0 for i in range(len(x))]

for i in range(len(x)):
```

```
    yp[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + c

    yp[i] = sigmoid(yp[i])

y_ts = set(y_ts)

yp = set(yp)

y_actual = pd.Series(y_ts, name='Actual')

y_pred = pd.Series(yp, name='Predicted')

confmat = pd.crosstab(y_actual, y_pred)

print(confmat)

confmat = np.asarray(confmat)

tp = confmat[1][1]

tn = confmat[0][0]

fp = confmat[0][1]

fn = confmat[1][0]



Acc = (tp+tn)/(tp+tn+fp+fn)

SE = tp/(tp+fn)

SP = tn/(tn+fp)



print('Accuracy : ' + str(Acc))

print('sensitivity : ' + str(SE))

print('specificity : ' + str(SP))
```

Results :

```
Predicted    0   1
Actual
0.0         21   0
1.0          0  19
Accuracy : 1.0
sensitivity : 1.0
specificity : 1.0
```

Question 8

8. Implement the multiclass logistic regression algorithm using both “One VS All” and “One VS One” multiclass coding techniques. Evaluate the performance of the multiclass classifier using individual class accuracy and overall accuracy measures. Use the hold-out cross-validation approach (60% training and 40% testing) for the selection of training and test instances. (Please use the dataset “data4.xlsx”)

Solution :

Code :

```
'''

***Logistic Regression***

one vs all multiclass



Author :

Pranath Reddy

2016B5A30572H

'''
```

```python
import pandas as pd

import math

import numpy as np

from sklearn.model_selection import train_test_split



# A function to return the column at specified index

def getcol(data,c):

    col = []

    for i in range(len(data)):

        col.append(data[i][c])

    return col



def set(y):

    for i in range(len(y)):

        if(y[i]>=0.5):

            y[i] = 1

        if(y[i]<0.5):

            y[i] = 0

    return y



def sigmoid(x):

    return 1.0 / (1.0 + np.exp(-x))
```

```python
# A function to return the updated values of m,c after one iteration of gradient descent

def wtupdate(m1,m2,m3,m4,m5,m6,m7,c,x,y):

    sumvm1 = 0

    sumvm2 = 0

    sumvm3 = 0

    sumvm4 = 0

    sumvm5 = 0

    sumvm6 = 0

    sumvm7 = 0

    sumvc = 0

    yp = [0 for i in range(len(x))]

    for i in range(len(x)):

        yp[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) + (m6*x[i,5]) + (m7*x[i,6]) + c

        yp[i] = sigmoid(yp[i])

        sumvm1 = sumvm1 - (y[i]-yp[i])*x[i,0]

        sumvm2 = sumvm2 - (y[i]-yp[i])*x[i,1]

        sumvm3 = sumvm3 - (y[i]-yp[i])*x[i,2]

        sumvm4 = sumvm4 - (y[i]-yp[i])*x[i,3]

        sumvm5 = sumvm5 - (y[i]-yp[i])*x[i,4]

        sumvm6 = sumvm6 - (y[i]-yp[i])*x[i,5]
```

```
        sumvm7 = sumvm7 - (y[i]-yp[i])*x[i,6]

        sumvc = sumvc - (y[i]-yp[i])

    m1 = m1 - 0.1*sumvm1

    m2 = m2 - 0.1*sumvm2

    m3 = m3 - 0.1*sumvm3

    m4 = m4 - 0.1*sumvm4

    m5 = m5 - 0.1*sumvm5

    m6 = m6 - 0.1*sumvm6

    m7 = m7 - 0.1*sumvm7

    c = c - 0.1*sumvc

    return m1,m2,m3,m4,m5,m6,m7,c


# A function to return the slope and intercept of y^

def linreg(x,y):

    m1 = 0

    m2 = 0

    m3 = 0

    m4 = 0

    m5 = 0

    m6 = 0

    m7 = 0

    c = 0
```

```
    iters = 2000

    i = 0

    while(i<iters):

        m1,m2,m3,m4,m5,m6,m7,c = wtupdate(m1,m2,m3,m4,m5,m6,m7,c,x,y)

        i = i+1

    return m1,m2,m3,m4,m5,m6,m7,c



# A function to implement min-max normalization

def norm(data):

    ndata = data

    for i in range(7):

        maxval = max(getcol(data,i))

        minval = min(getcol(data,i))

        for j in range(len(data)):

            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)

    return ndata




# import the data

data = pd.read_excel('data4.xlsx',header=None)

data = np.asarray(data)

y = data[:,-1]
```

```python
data = norm(data)

x = data[:,:-1]



# split into testing and training sets

x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.4)


y1_tr = [1 for i in range(len(y_tr))]

y2_tr = [1 for i in range(len(y_tr))]

y3_tr = [1 for i in range(len(y_tr))]

for i in range(len(y_tr)):

    if(y_tr[i] != 1):

        y1_tr[i] = 0

    if(y_tr[i] != 2):

        y2_tr[i] = 0

    if(y_tr[i] != 3):

        y3_tr[i] = 0


x = x_ts


m1,m2,m3,m4,m5,m6,m7,c = linreg(x_tr,y1_tr)

yp1 = [0 for i in range(len(x))]
```

```
for i in range(len(x)):
    yp1[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) +
(m6*x[i,5]) + (m7*x[i,6]) + c
    yp1[i] = sigmoid(yp1[i])
yp1 = set(yp1)


m1,m2,m3,m4,m5,m6,m7,c = linreg(x_tr,y2_tr)
yp2 = [0 for i in range(len(x))]
for i in range(len(x)):
    yp2[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) +
(m6*x[i,5]) + (m7*x[i,6]) + c
    yp2[i] = sigmoid(yp2[i])
yp2 = set(yp2)


m1,m2,m3,m4,m5,m6,m7,c = linreg(x_tr,y3_tr)
yp3 = [0 for i in range(len(x))]
for i in range(len(x)):
    yp3[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) +
(m6*x[i,5]) + (m7*x[i,6]) + c
    yp3[i] = sigmoid(yp3[i])
yp3 = set(yp3)


cval = [0 for i in range(len(y_ts))]
for i in range(len(y_ts)):
```

```
    if (yp1[i] == 1):

        cval[i] = 1.0

    if (yp2[i] == 1):

        cval[i] = 2.0

    if (yp3[i] == 1):

        cval[i] = 3.0




for i in range(len(cval)):

    if (cval[i] == 0):

        cval[i] = 'None'

y_actual = pd.Series(y_ts, name='Actual')

y_pred = pd.Series(cval, name='Predicted')

confmat = pd.crosstab(y_actual, y_pred)

print(confmat)



confmat = np.asarray(confmat)

Acc = (confmat[0][0] + confmat[1][1] + confmat[2][2])/sum(sum(confmat))

Acc1 = confmat[0][0]/sum(confmat[0])

Acc2 = confmat[1][1]/sum(confmat[1])

Acc3 = confmat[2][2]/sum(confmat[2])

print('Overall Accuracy : ' + str(Acc))
```

```
print('Accuracy of class 1 : ' + str(Acc1))

print('Accuracy of class 2 : ' + str(Acc2))

print('Accuracy of class 3 : ' + str(Acc3))
```

Results :

```
Predicted  1.0  2.0  3.0
Actual
1.0         20    0    0
2.0          0   21    1
3.0          0    0   18
Overall Accuracy : 0.9833333333333333
Accuracy of class 1 : 1.0
Accuracy of class 2 : 0.9545454545454546
Accuracy of class 3 : 1.0
```

```
'''

***Logistic Regression***

one vs one multiclass



Author :

Pranath Reddy

2016B5A30572H

'''
```

```python
import pandas as pd

import math

import numpy as np

from sklearn.model_selection import train_test_split



# A function to return the column at specified index

def getcol(data,c):

    col = []

    for i in range(len(data)):

        col.append(data[i][c])

    return col



def set(y):

    for i in range(len(y)):

        if(y[i]>=0.5):

            y[i] = 1

        if(y[i]<0.5):

            y[i] = 0

    return y



def sigmoid(x):

    return 1.0 / (1.0 + np.exp(-x))
```

```python
# A function to return the updated values of m,c after one iteration of gradient descent

def wtupdate(m1,m2,m3,m4,m5,m6,m7,c,x,y):

    sumvm1 = 0

    sumvm2 = 0

    sumvm3 = 0

    sumvm4 = 0

    sumvm5 = 0

    sumvm6 = 0

    sumvm7 = 0

    sumvc = 0

    yp = [0 for i in range(len(x))]

    for i in range(len(x)):

        yp[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) + (m6*x[i,5]) + (m7*x[i,6]) + c

        yp[i] = sigmoid(yp[i])

        sumvm1 = sumvm1 - (y[i]-yp[i])*x[i,0]

        sumvm2 = sumvm2 - (y[i]-yp[i])*x[i,1]

        sumvm3 = sumvm3 - (y[i]-yp[i])*x[i,2]

        sumvm4 = sumvm4 - (y[i]-yp[i])*x[i,3]

        sumvm5 = sumvm5 - (y[i]-yp[i])*x[i,4]

        sumvm6 = sumvm6 - (y[i]-yp[i])*x[i,5]
```

```
        sumvm7 = sumvm7 - (y[i]-yp[i])*x[i,6]

        sumvc = sumvc - (y[i]-yp[i])

    m1 = m1 - 0.1*sumvm1

    m2 = m2 - 0.1*sumvm2

    m3 = m3 - 0.1*sumvm3

    m4 = m4 - 0.1*sumvm4

    m5 = m5 - 0.1*sumvm5

    m6 = m6 - 0.1*sumvm6

    m7 = m7 - 0.1*sumvm7

    c = c - 0.1*sumvc

    return m1,m2,m3,m4,m5,m6,m7,c



# A function to return the slope and intercept of y^

def linreg(x,y):

    m1 = 0

    m2 = 0

    m3 = 0

    m4 = 0

    m5 = 0

    m6 = 0

    m7 = 0

    c = 0
```

```
    iters = 2000
    i = 0
    while(i<iters):
        m1,m2,m3,m4,m5,m6,m7,c = wtupdate(m1,m2,m3,m4,m5,m6,m7,c,x,y)
        i = i+1
    return m1,m2,m3,m4,m5,m6,m7,c


# A function to implement min-max normalization
def norm(data):
    ndata = data
    for i in range(7):
        maxval = max(getcol(data,i))
        minval = min(getcol(data,i))
        for j in range(len(data)):
            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)
    return ndata



# import the data
data = pd.read_excel('data4.xlsx',header=None)
data = np.asarray(data)
y = data[:,-1]
```

```
data = norm(data)

x = data[:,:-1]



# split into testing and training sets

x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.4)


y1_tr = [] # class 1 vs class 2

y2_tr = [] # class 1 vs class 3

y3_tr = [] # class 2 vs class 3

x1_tr = []

x2_tr = []

x3_tr = []

for i in range(len(y_tr)):

    if(y_tr[i] != 3):

        y1_tr.append(y_tr[i])

        x1_tr.append(x_tr[i])

    if(y_tr[i] != 2):

        y2_tr.append(y_tr[i])

        x2_tr.append(x_tr[i])

    if(y_tr[i] != 1):

        y3_tr.append(y_tr[i])
```

```
        x3_tr.append(x_tr[i])


x1_tr = np.asarray(x1_tr)

x2_tr = np.asarray(x2_tr)

x3_tr = np.asarray(x3_tr)


# for 1 vs 2, we consider 1 as positive class

for i in range(len(y1_tr)):

    if y1_tr[i] == 1:

        y1_tr[i] = 1

    else:

        y1_tr[i] = 0


# for 1 vs 3, we consider 3 as positive class

for i in range(len(y2_tr)):

    if y2_tr[i] == 3:

        y2_tr[i] = 1

    else:

        y2_tr[i] = 0


# for 2 vs 3, we consider 2 as positive class

for i in range(len(y3_tr)):
```

```
    if y3_tr[i] == 2:

        y3_tr[i] = 1

    else:

        y3_tr[i] = 0



x = x_ts



m1,m2,m3,m4,m5,m6,m7,c = linreg(x1_tr,y1_tr)

yp1 = [0 for i in range(len(x))]

for i in range(len(x)):

    yp1[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) +
(m6*x[i,5]) + (m7*x[i,6]) + c

    yp1[i] = sigmoid(yp1[i])

yp1 = set(yp1)



m1,m2,m3,m4,m5,m6,m7,c = linreg(x2_tr,y2_tr)

yp2 = [0 for i in range(len(x))]

for i in range(len(x)):

    yp2[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) +
(m6*x[i,5]) + (m7*x[i,6]) + c

    yp2[i] = sigmoid(yp2[i])

yp2 = set(yp2)
```

```
m1,m2,m3,m4,m5,m6,m7,c = linreg(x3_tr,y3_tr)

yp3 = [0 for i in range(len(x))]

for i in range(len(x)):

    yp3[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) +
(m6*x[i,5]) + (m7*x[i,6]) + c

    yp3[i] = sigmoid(yp3[i])

yp3 = set(yp3)




cval = [0 for i in range(len(y_ts))]

for i in range(len(y_ts)):

    if (yp1[i] == 1 and yp2[i] == 0):

        cval[i] = 1.0

    if (yp1[i] == 0 and yp3[i] == 1):

        cval[i] = 2.0

    if (yp2[i] == 1 and yp3[i] == 0):

        cval[i] = 3.0




for i in range(len(cval)):

    if (cval[i] == 0):

        cval[i] = 'None'

y_actual = pd.Series(y_ts, name='Actual')
```

```python
y_pred = pd.Series(cval, name='Predicted')

confmat = pd.crosstab(y_actual, y_pred)

print(confmat)



confmat = np.asarray(confmat)

Acc = (confmat[0][0] + confmat[1][1] + confmat[2][2])/sum(sum(confmat))

Acc1 = confmat[0][0]/sum(confmat[0])

Acc2 = confmat[1][1]/sum(confmat[1])

Acc3 = confmat[2][2]/sum(confmat[2])

print('Overall Accuracy : ' + str(Acc))

print('Accuracy of class 1 : ' + str(Acc1))

print('Accuracy of class 2 : ' + str(Acc2))

print('Accuracy of class 3 : ' + str(Acc3))
```

Results :

```
Predicted  1.0  2.0  3.0
Actual
1.0         14    0    0
2.0          0   20    2
3.0          0    0   24
Overall Accuracy : 0.9666666666666667
Accuracy of class 1 : 1.0
Accuracy of class 2 : 0.9090909090909091
Accuracy of class 3 : 1.0
```

Question 9

9. Evaluate the performance of multiclass logistic regression classifier using 5-fold cross- validation approach. Evaluate the individual class accuracy and overall accuracy measures for the multiclass classifier along each fold. (Please use the dataset “data4.xlsx”)

Solution :

Code :

```
'''

***Logistic Regression***

one vs all multiclass and 5-fold cross validation



Author :

Pranath Reddy

2016B5A30572H

'''

import pandas as pd

import math

import numpy as np



# A function to return the column at specified index

def getcol(data,c):

    col = []
```

```
    for i in range(len(data)):

        col.append(data[i][c])

    return col



def set(y):

    for i in range(len(y)):

        if(y[i]>=0.5):

            y[i] = 1

        if(y[i]<0.5):

            y[i] = 0

    return y



def sigmoid(x):

    return 1.0 / (1.0 + np.exp(-x))



# A function to return the updated values of m,c after one iteration of gradient
descent

def wtupdate(m1,m2,m3,m4,m5,m6,m7,c,x,y):

    sumvm1 = 0

    sumvm2 = 0

    sumvm3 = 0

    sumvm4 = 0

    sumvm5 = 0
```

```
    sumvm6 = 0

    sumvm7 = 0

    sumvc = 0

    yp = [0 for i in range(len(x))]

    for i in range(len(x)):

        yp[i] = (m1*x[i,0]) + (m2*x[i,1]) + (m3*x[i,2]) + (m4*x[i,3]) + (m5*x[i,4]) + (m6*x[i,5]) + (m7*x[i,6]) + c

        yp[i] = sigmoid(yp[i])

        sumvm1 = sumvm1 - (y[i]-yp[i])*x[i,0]

        sumvm2 = sumvm2 - (y[i]-yp[i])*x[i,1]

        sumvm3 = sumvm3 - (y[i]-yp[i])*x[i,2]

        sumvm4 = sumvm4 - (y[i]-yp[i])*x[i,3]

        sumvm5 = sumvm5 - (y[i]-yp[i])*x[i,4]

        sumvm6 = sumvm6 - (y[i]-yp[i])*x[i,5]

        sumvm7 = sumvm7 - (y[i]-yp[i])*x[i,6]

        sumvc = sumvc - (y[i]-yp[i])

    m1 = m1 - 0.1*sumvm1

    m2 = m2 - 0.1*sumvm2

    m3 = m3 - 0.1*sumvm3

    m4 = m4 - 0.1*sumvm4

    m5 = m5 - 0.1*sumvm5

    m6 = m6 - 0.1*sumvm6

    m7 = m7 - 0.1*sumvm7
```

```
    c = c - 0.1*sumvc

    return m1,m2,m3,m4,m5,m6,m7,c



# A function to return the slope and intercept of y^

def linreg(x,y):

    m1 = 0

    m2 = 0

    m3 = 0

    m4 = 0

    m5 = 0

    m6 = 0

    m7 = 0

    c = 0

    iters = 2000

    i = 0

    while(i<iters):

        m1,m2,m3,m4,m5,m6,m7,c = wtupdate(m1,m2,m3,m4,m5,m6,m7,c,x,y)

        i = i+1

    return m1,m2,m3,m4,m5,m6,m7,c



# A function to implement min-max normalization

def norm(data):
```

```
    ndata = data
    for i in range(7):
        maxval = max(getcol(data,i))
        minval = min(getcol(data,i))
        for j in range(len(data)):
            ndata[j][i] = (data[j][i]-minval)/((maxval-minval)+0.05)
    return ndata



# import the data
data = pd.read_excel('data4.xlsx',header=None)
data = np.asarray(data)

y = data[:,-1]
data = norm(data)
x = data[:,:-1]

rand_index = np.arange(len(x))
np.random.shuffle(rand_index)
x = x[rand_index]
y = y[rand_index]
```

```
Acc_list = []

Acc1_list = []

Acc2_list = []

Acc3_list = []


p = 0

q = 30

for folds in range(5):


    # split into testing and training sets

    x_ts = x[p:q,:]

    y_ts = y[p:q]

    x_tr = np.concatenate((x[:p,:],x[q:,:]),axis=0)

    y_tr = np.concatenate((y[:p],y[q:]),axis=0)


    y1_tr = [1 for i in range(len(y_tr))]

    y2_tr = [1 for i in range(len(y_tr))]

    y3_tr = [1 for i in range(len(y_tr))]

    for i in range(len(y_tr)):

        if(y_tr[i] != 1):

            y1_tr[i] = 0

        if(y_tr[i] != 2):
```

```
            y2_tr[i] = 0

        if(y_tr[i] != 3):

            y3_tr[i] = 0



    m1,m2,m3,m4,m5,m6,m7,c = linreg(x_tr,y1_tr)

    yp1 = [0 for i in range(len(x_ts))]



    for i in range(len(x_ts)):

        yp1[i] = (m1*x_ts[i,0]) + (m2*x_ts[i,1]) + (m3*x_ts[i,2]) + (m4*x_ts[i,3]) +
(m5*x_ts[i,4]) + (m6*x_ts[i,5]) + (m7*x_ts[i,6]) + c

        yp1[i] = sigmoid(yp1[i])

    yp1 = set(yp1)




    m1,m2,m3,m4,m5,m6,m7,c = linreg(x_tr,y2_tr)

    yp2 = [0 for i in range(len(x_ts))]



    for i in range(len(x_ts)):

        yp2[i] = (m1*x_ts[i,0]) + (m2*x_ts[i,1]) + (m3*x_ts[i,2]) + (m4*x_ts[i,3]) +
(m5*x_ts[i,4]) + (m6*x_ts[i,5]) + (m7*x_ts[i,6]) + c

        yp2[i] = sigmoid(yp2[i])

    yp2 = set(yp2)
```

```
    m1,m2,m3,m4,m5,m6,m7,c = linreg(x_tr,y3_tr)

    yp3 = [0 for i in range(len(x_ts))]



    for i in range(len(x_ts)):

        yp3[i] = (m1*x_ts[i,0]) + (m2*x_ts[i,1]) + (m3*x_ts[i,2]) + (m4*x_ts[i,3]) +
(m5*x_ts[i,4]) + (m6*x_ts[i,5]) + (m7*x_ts[i,6]) + c

        yp3[i] = sigmoid(yp3[i])

    yp3 = set(yp3)



    cval = [0 for i in range(len(y_ts))]

    for i in range(len(y_ts)):

        if (yp1[i] == 1):

            cval[i] = 1.0

        if (yp2[i] == 1):

            cval[i] = 2.0

        if (yp3[i] == 1):

            cval[i] = 3.0



    for i in range(len(cval)):

        if (cval[i] == 0):

            cval[i] = 'None'

    y_actual = pd.Series(y_ts, name='Actual')
```

```
    y_pred = pd.Series(cval, name='Predicted')

    confmat = pd.crosstab(y_actual, y_pred)

    print('Fold ' + str(folds+1) + ' Results : ')

    print(confmat)



    confmat = np.asarray(confmat)

    Acc = (confmat[0][0] + confmat[1][1] + confmat[2][2])/sum(sum(confmat))

    Acc1 = confmat[0][0]/sum(confmat[0])

    Acc2 = confmat[1][1]/sum(confmat[1])

    Acc3 = confmat[2][2]/sum(confmat[2])

    Acc_list.append(Acc)

    Acc1_list.append(Acc1)

    Acc2_list.append(Acc2)

    Acc3_list.append(Acc3)



    p = int(p + int(len(y)/5))

    q = int(q + int(len(y)/5))



    print('***************')




print('Final Results :')
```

```python
print('Overall Accuracy : ' + str(Acc_list))

print('Accuracy of class 1 : ' + str(Acc1_list))

print('Accuracy of class 2 : ' + str(Acc2_list))

print('Accuracy of class 3 : ' + str(Acc3_list))
```

Results :

// vectors of length 5 with each value corresponding to each fold

```
Final Results :
Overall Accuracy : [0.8666666666666667, 1.0, 0.9666666666666667, 0.9666666666666667, 1.0]
Accuracy of class 1 : [1.0, 1.0, 1.0, 1.0, 1.0]
Accuracy of class 2 : [0.9090909090909091, 1.0, 0.9, 0.9, 1.0]
Accuracy of class 3 : [0.75, 1.0, 1.0, 1.0, 1.0]
```

Question 10

10. Use the likelihood ratio test (LRT) for the binary classification using the dataset ("data3.xlsx"). Divide the dataset into training and testing using hold-out cross-validation technique with 60 % of instances as training and the remaining 40% as testing. Evaluate the accuracy, sensitivity and specificity values for the binary classifier.

Solution :

Code :

```python
'''

*** LRT Binary Classification ***



Author :

Pranath Reddy

2016B5A30572H

'''
```

```python
import pandas as pd

import math

import numpy as np

from sklearn.model_selection import train_test_split



# A function to return a column of the data at the specified index

def col(array, i):

    return [row[i] for row in array]



# A function to calculate the mean of an array

def mean(array):

    m = []

    for i in range(4):

        m.append(sum(col(array,i))/len(col(array,i)))

    return m



# a function to implement LRT

def rule(x_ts,x,y):

    p1 = len([i for (i, val) in enumerate(y) if val == 1])

    p2 = len([i for (i, val) in enumerate(y) if val == 2])

    p1, p2 = p1/(len(y)), p2/(len(y))

    x1 = np.array([x[i] for (i, val) in enumerate(y) if val == 1])
```

```
    x2 = np.array([x[i] for (i, val) in enumerate(y) if val == 2])

    m1 = mean(x1)

    m2 = mean(x2)

    cov1 = np.cov(x1.T)

    cov2 = np.cov(x2.T)

    coeff1 = 1/(((2*3.14)**2)*np.linalg.det(cov1)**0.5)

    coeff2 = 1/(((2*3.14)**2)*np.linalg.det(cov2)**0.5)

    l1 = coeff1*np.exp(-0.5*np.dot(np.dot((x_ts - m1),np.linalg.inv(cov1)),(x_ts -
m1).T))

    l2 = coeff2*np.exp(-0.5*np.dot(np.dot((x_ts - m2),np.linalg.inv(cov2)),(x_ts -
m2).T))

    if (l1/p2) > (l2/p1):

        return 1

    else:

        return 2



def confmat(y_pred,y_ts):

    a, b, c, d = 0, 0, 0, 0

    for i in range(len(y_ts)):

        if y_ts[i] == 1:

            if y_pred[i] == 1:

                a = a + 1

            if y_pred[i] == 2:
```

```
                b = b + 1
        if y_ts[i] == 2:
            if y_pred[i] == 1:
                c = c + 1
            if y_pred[i] == 2:
                d = d + 1
    return a, b, c, d


# input the data csv
data = pd.read_excel('data3.xlsx',header=None)
data = np.asarray(data)


x = data[:,:-1]
y = data[:,-1]
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.4)


y_pred = []
for i in range(len(x_ts)):
    y_pred.append(rule(x_ts[i],x_tr,y_tr))


a, b, c, d = confmat(y_pred,y_ts)
acc = (a+d)/(a+b+c+d)
```

```
sens = (a)/(a+b)

spec = (d)/(d+c)


print('we are assuming class 1 to be positive and class2 to be negative')

print('tp: ',a,'fp: ',c,'tn: ',d,'fn: ',b)

print('accuracy: ',acc,'sensitivity: ',sens,'specificity: ',spec)
```

Results :

```
we are assuming class 1 to be positive and class2 to be negative
tp:  18 fp:  0 tn:  22 fn:  0
accuracy:  1.0 sensitivity:  1.0 specificity:  1.0
```

Question 11

11. Implement the Maximum a posteriori (MAP) decision rule for multiclass classification task. Use the hold-out cross-validation approach (70% training and 30% testing) for the selection of training and test instances. (Please use the dataset “data4.xlsx”).

Solution :

Code :

```
'''

*** MAP multiclass Classification ***


Author :

Pranath Reddy

2016B5A30572H

'''
```

```python
import pandas as pd

import math

import numpy as np

from sklearn.model_selection import train_test_split



# A function to return a column of the data at the specified index

def col(array, i):

    return [row[i] for row in array]



# A function to calculate the mean of an array

def mean(array):

    m = []

    for i in range(7):

        m.append(sum(col(array,i))/len(col(array,i)))

    return m



# a function to implement LRT

def rule(x_ts,x,y):

    p1 = len([i for (i, val) in enumerate(y) if val == 1])

    p2 = len([i for (i, val) in enumerate(y) if val == 2])

    p3 = len([i for (i, val) in enumerate(y) if val == 3])

    # priors P(y)
```

```
    p1, p2, p3 = p1/(len(y)), p2/(len(y)), p3/(len(y))

    x1 = np.array([x[i] for (i, val) in enumerate(y) if val == 1])

    x2 = np.array([x[i] for (i, val) in enumerate(y) if val == 2])

    x3 = np.array([x[i] for (i, val) in enumerate(y) if val == 3])

    # evidence P(x)

    e1, e2, e3 = len(x1)/(len(x)), len(x2)/(len(x)), len(x3)/(len(x))

    m1 = mean(x1)

    m2 = mean(x2)

    m3 = mean(x3)

    cov1 = np.cov(x1.T)

    cov2 = np.cov(x2.T)

    cov3 = np.cov(x3.T)

    coeff1 = 1/(((2*3.14)**2)*np.linalg.det(cov1)**0.5)

    coeff2 = 1/(((2*3.14)**2)*np.linalg.det(cov2)**0.5)

    coeff3 = 1/(((2*3.14)**2)*np.linalg.det(cov3)**0.5)

    # likelihoods P(x|y)

    l1 = coeff1*np.exp(-0.5*np.dot(np.dot((x_ts - m1),np.linalg.inv(cov1)),(x_ts -
m1).T))

    l2 = coeff2*np.exp(-0.5*np.dot(np.dot((x_ts - m2),np.linalg.inv(cov2)),(x_ts -
m2).T))

    l3 = coeff3*np.exp(-0.5*np.dot(np.dot((x_ts - m3),np.linalg.inv(cov3)),(x_ts -
m3).T))

    # Posteriors P(y|x)

    prob1, prob2, prob3 = (l1*p1)/e1, (l2*p2)/e2, (l3*p3)/e3
```

```
    if max(prob1,prob2,prob3) == prob1:

        return 1

    elif max(prob1,prob2,prob3) == prob2:

        return 2

    else:

        return 3



# input the data csv

data = pd.read_excel('data4.xlsx',header=None)

data = np.asarray(data)



x = data[:,:-1]

y = data[:,-1]

x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)



y_pred = []

for i in range(len(x_ts)):

    y_pred.append(rule(x_ts[i],x_tr,y_tr))



y_actual = pd.Series(y_ts, name='Actual')

y_pred = pd.Series(y_pred, name='Predicted')

confmat = pd.crosstab(y_actual, y_pred)
```

```
print(confmat)

confmat = np.asarray(confmat)

Acc = (confmat[0][0] + confmat[1][1] + confmat[2][2])/sum(sum(confmat))

Acc1 = confmat[0][0]/sum(confmat[0])

Acc2 = confmat[1][1]/sum(confmat[1])

Acc3 = confmat[2][2]/sum(confmat[2])

print('Overall Accuracy : ' + str(Acc))

print('Accuracy of class 1 : ' + str(Acc1))

print('Accuracy of class 2 : ' + str(Acc2))

print('Accuracy of class 3 : ' + str(Acc3))
```

Results :

```
Predicted    1   2   3
Actual
1.0         13   0   0
2.0          0  18   1
3.0          0   0  13
Overall Accuracy : 0.9777777777777777
Accuracy of class 1 : 1.0
Accuracy of class 2 : 0.9473684210526315
Accuracy of class 3 : 1.0
```

Question 12

12. Implement the Maximum likelihood (ML) decision rule for multiclass classification task. Use the hold-out cross-validation approach (70% training and 30% testing) for the selection of training and test instances. (Please use the dataset “data4.xlsx”).

Solution :

Code :

```
'''

*** Max likelihood multiclass Classification ***



Author :

Pranath Reddy

2016B5A30572H

'''

import pandas as pd

import math

import numpy as np

from sklearn.model_selection import train_test_split



# A function to return a column of the data at the specified index

def col(array, i):

    return [row[i] for row in array]



# A function to calculate the mean of an array
```

```
def mean(array):
    m = []
    for i in range(7):
        m.append(sum(col(array,i))/len(col(array,i)))
    return m


# a function to implement LRT
def rule(x_ts,x,y):
    x1 = np.array([x[i] for (i, val) in enumerate(y) if val == 1])
    x2 = np.array([x[i] for (i, val) in enumerate(y) if val == 2])
    x3 = np.array([x[i] for (i, val) in enumerate(y) if val == 3])
    m1 = mean(x1)
    m2 = mean(x2)
    m3 = mean(x3)
    cov1 = np.cov(x1.T)
    cov2 = np.cov(x2.T)
    cov3 = np.cov(x3.T)
    coeff1 = 1/(((2*3.14)**2)*np.linalg.det(cov1)**0.5)
    coeff2 = 1/(((2*3.14)**2)*np.linalg.det(cov2)**0.5)
    coeff3 = 1/(((2*3.14)**2)*np.linalg.det(cov3)**0.5)
    # likelihoods P(x|y)
    l1 = coeff1*np.exp(-0.5*np.dot(np.dot((x_ts - m1),np.linalg.inv(cov1)),(x_ts -
m1).T))
```

```
    l2 = coeff2*np.exp(-0.5*np.dot(np.dot((x_ts - m2),np.linalg.inv(cov2)),(x_ts -
m2).T))

    l3 = coeff3*np.exp(-0.5*np.dot(np.dot((x_ts - m3),np.linalg.inv(cov3)),(x_ts -
m3).T))

    if max(l1,l2,l3) == l1:

        return 1

    elif max(l1,l2,l3) == l2:

        return 2

    else:

        return 3




def confmat(y_pred,y_ts):

    a, b, c, d = 0, 0, 0, 0

    for i in range(len(y_ts)):

        if y_ts[i] == 1:

            if y_pred[i] == 1:

                a = a + 1

            if y_pred[i] == 2:

                b = b + 1

        if y_ts[i] == 2:

            if y_pred[i] == 1:

                c = c + 1
```

```
            if y_pred[i] == 2:
                d = d + 1
    return a, b, c, d


# input the data csv
data = pd.read_excel('data4.xlsx',header=None)
data = np.asarray(data)


x = data[:,:-1]
y = data[:,-1]
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)


y_pred = []
for i in range(len(x_ts)):
    y_pred.append(rule(x_ts[i],x_tr,y_tr))


y_actual = pd.Series(y_ts, name='Actual')
y_pred = pd.Series(y_pred, name='Predicted')
confmat = pd.crosstab(y_actual, y_pred)
print(confmat)
confmat = np.asarray(confmat)
Acc = (confmat[0][0] + confmat[1][1] + confmat[2][2])/sum(sum(confmat))
```

```
Acc1 = confmat[0][0]/sum(confmat[0])

Acc2 = confmat[1][1]/sum(confmat[1])

Acc3 = confmat[2][2]/sum(confmat[2])

print('Overall Accuracy : ' + str(Acc))

print('Accuracy of class 1 : ' + str(Acc1))

print('Accuracy of class 2 : ' + str(Acc2))

print('Accuracy of class 3 : ' + str(Acc3))
```

Results :

```
Predicted   1   2   3
Actual
1.0        10   0   0
2.0         0  18   1
3.0         0   1  15
Overall Accuracy : 0.9555555555555556
Accuracy of class 1 : 1.0
Accuracy of class 2 : 0.9473684210526315
Accuracy of class 3 : 0.9375
```

Question 13

13. Please write in your own words that what you have learned by solving the Assignment 1..

Solution :

Topics Learned/Observations :

- Linear regression algorithm, Analysis of the cost function and its relation with the weight parameters
- Linear regression using stochastic and batch gradient descent methods
- Vectorized implementation of linear regression and Linear regression with regularization using Ridge regression and least angle regression methods
- Implementation of the unsupervised algorithm K-means Clustering
- Logistic regression of classification problems
- ''One vs all'' and ''one vs one'' algorithms for multiclass classification
- Hold out cross validation and K-fold cross validation
- Confusion matrix, accuracy, sensitivity and specificity for measuring the performance of a classification algorithm
- Probabilistic classifiers (LRT, MAP and ML)
- Working with Pandas library for importing data, numpy library for math and array operations, and matplotlib for plotting
- We have seen that regularization methods were able to produce a lower value of cost function for a given number of iterations
- Stochastic gradient descent method produced a noisy cost plot when compared to batch gradient descent method
- Probabilistic classifiers are able to produce comparable results to logistic regression based methods in a lower amount of processing time