NEURAL NETWORK AND FUZZY LOGIC

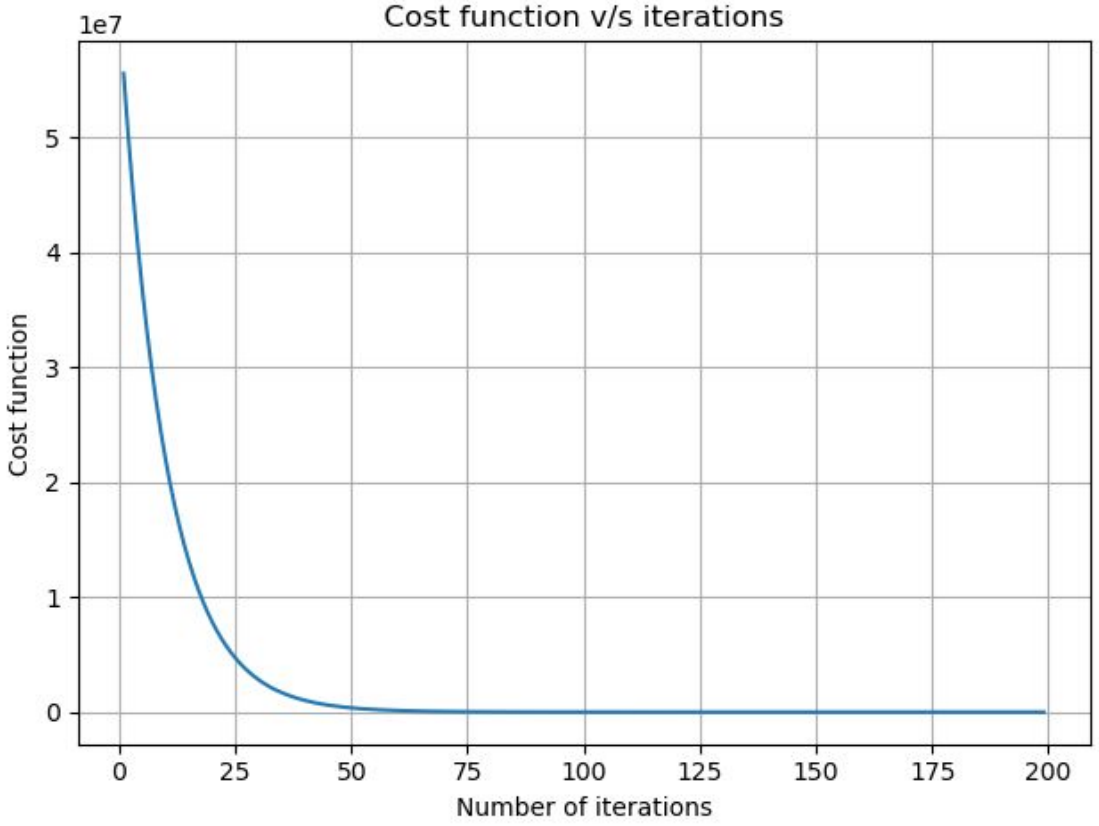
ASSIGNMENT-1

GRANDHI SNIGDHA

2015A8PS0495H

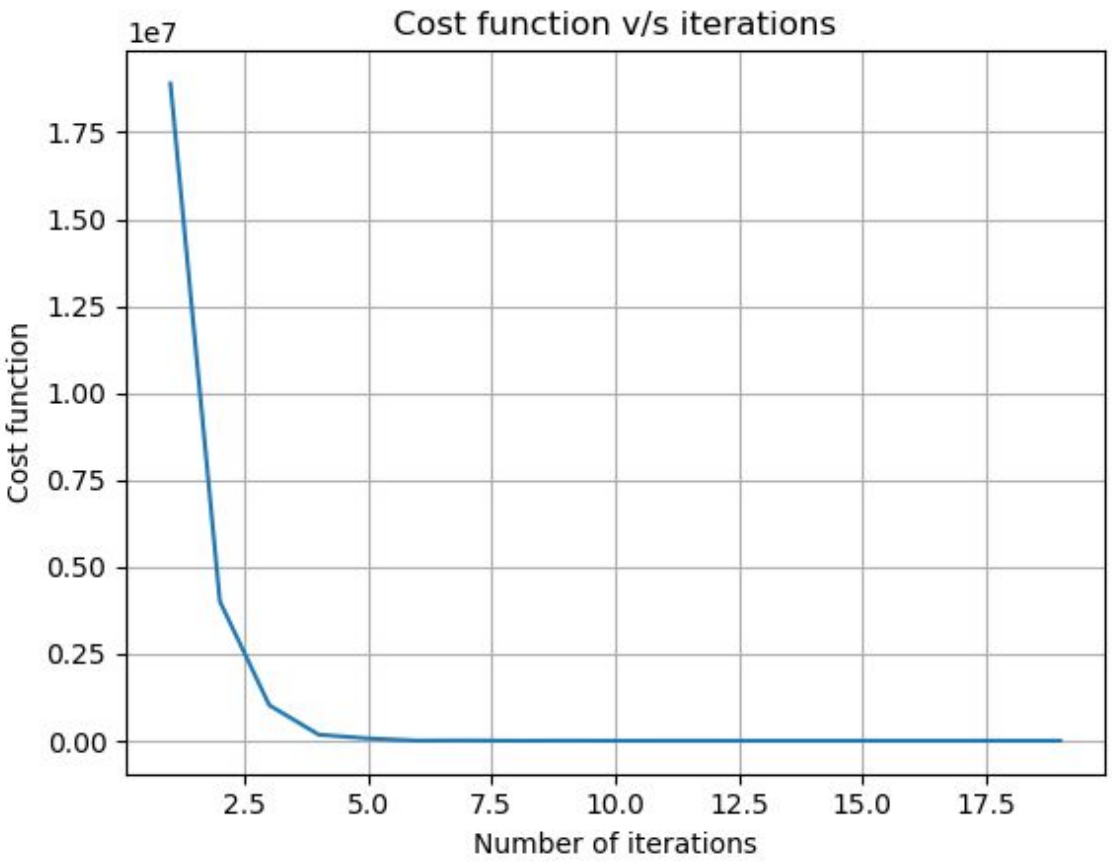
PROBLEM 1

```
# linear regression algorithm
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame

data = pd.read_excel('data.xlsx')
# print(data)
var1 = data['row1']
var2 = data['row2']
y = data['row3']
x = []
for i in data.index:
    temp = []
    temp.append(1);
    temp.append(var1[i])
    temp.append(var2[i])
    x.append(temp)
# print(x)
alpha=0.000000001; #learning rate
cost = []
w = [1,1,1]
w = np.array(w)
y = np.array(y)
for epoch in range(100):
    for j in range(len(x[0])):
        # print("me");
        sumx = 0
        tempcost = 0
        for i in range(len(x)):
            hx = w[0] + w[1]*x[i][1] + w[2]*x[i][2]
            hx = hx - y[i]
            tempcost += hx**2
            hx = hx*x[i][j]
            sumx += hx
            # print(hx)
        # print(hx)
        w[j] -= alpha*sumx
```


Cost function v/s iterations
1e7
5
4
3
2
1
0
Cost function
0
25
50
75
100
125
150
175
200
Number of iterations

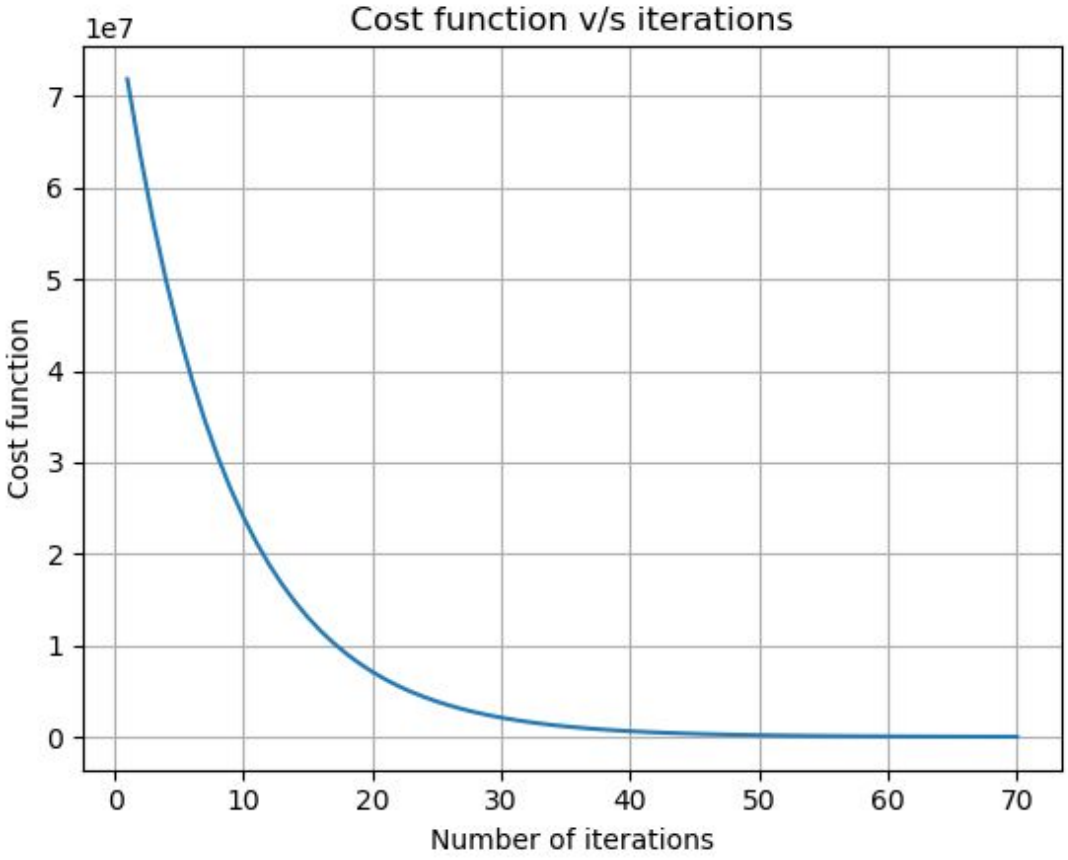
PROBLEM 2

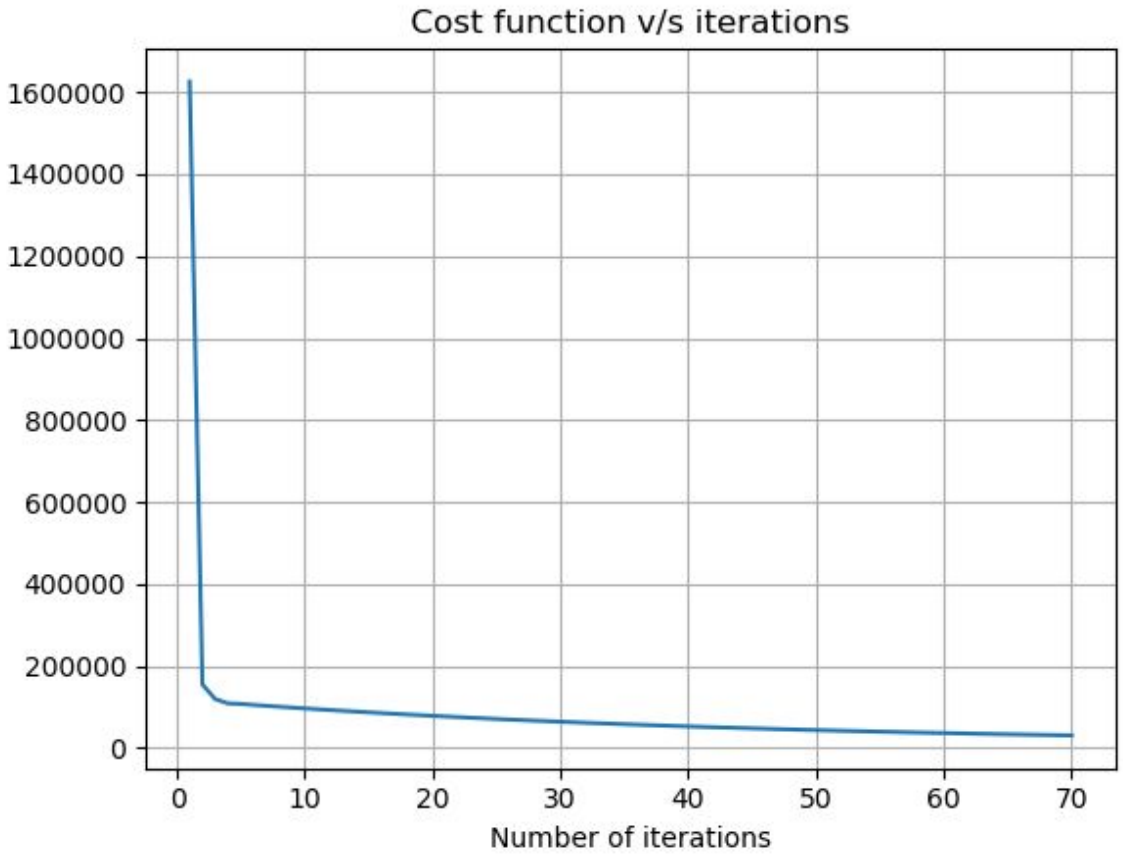
```python
# linear regression algorithm with stochastic gradient descent
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame

data = pd.read_excel('data.xlsx')
# print(data)
var1 = data['row1']
var2 = data['row2']
y = data['row3']
x = []
for i in data.index:
    temp = []
    temp.append(1);
    temp.append(var1[i])
    temp.append(var2[i])
    x.append(temp)
# print(x)
alpha=0.000000001; #learning rate
w = [1,1,1]
for epoch in range(100):
    for j in range(len(x[0])):
        # print("me");
        sumx = 0
        for i in range(len(x)):
            hx = w[0] + w[1]*x[i][1] + w[2]*x[i][2]
            hx = hx - y[i]
            hx = hx*x[i][j]
            sumx += hx
            # print(hx)
        # print(hx)
            w[j] -= alpha*sumx
    print(w)
    print("\n")
```


Cost function v/s iterations
1e7
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Cost function
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of iterations

PROBLEM 3

```python
# linear regression algorithm
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame

data = pd.read_excel('data.xlsx')
# print(data)
var1 = data['row1']
var2 = data['row2']
y = data['row3']
x = []
for i in data.index:
    temp = []
    temp.append(1);
    temp.append(var1[i])
    temp.append(var2[i])
    x.append(temp)
# print(x)
alpha=0.000000001; #learning rate
lmd = 1
w = [1,1,1]
for epoch in range(10):
    for j in range(len(x[0])):
        # print("me");
        sumx = 0
        for i in range(len(x)):
            hx = w[0] + w[1]*x[i][1] + w[2]*x[i][2]
            hx = hx - y[i]
            hx = hx*x[i][j]
            sumx += hx
            # print(hx)
        # print(hx)
        w[j] = (1-alpha*lmd)*w[j] - alpha*sumx
    print(w)
    print("\n")
```


Cost function v/s iterations
1e7
7
6
5
4
3
2
1
0
Cost function
0
10
20
30
40
50
60
70
Number of iterations


Cost function v/s iterations
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
0
10
20
30
40
50
60
70
Number of iterations

PROBLEM 4

```python
# vectorised linear regression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv

data = pd.read_excel('data.xlsx')
# print(data)
var1 = data['row1']
var2 = data['row2']
y = data['row3']
x = []
for i in data.index:
    temp = []
    temp.append(1);
    temp.append(var1[i])
    temp.append(var2[i])
    x.append(temp)
matx = np.array(x)
maty = np.array(y)
matxt = matx.transpose()
wght = np.matmul(matxt, matx)
wght = inv(wght)
wght = np.matmul(wght, matxt)
wght = np.matmul(wght, y)
print(wght)
```

PROBLEM 5

```python
# vectorisation based least angle regression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv

data = pd.read_excel('data.xlsx')
# print(data)
var1 = data['row1']
var2 = data['row2']
y = data['row3']
x = []
for i in data.index:
    temp = []
    temp.append(1);
    temp.append(var1[i])
    temp.append(var2[i])
    x.append(temp)

lamda = 2
matx = np.array(x)
maty = np.array(y)
matxt = matx.transpose()

wght = np.matmul(matxt, matx)
wght = inv(wght)
wght = np.matmul(wght, (np.matmul(matxt, maty) - (lamda/2)))

print(wght)
```

PROBLEM 6

```python
# clustering with k=2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv
def dist(a, b):
    sm=0
    for i in range(len(a)):
        sm += (a[i]-b[i])**2
    return sm

data = pd.read_excel('data2.xlsx')
# print(data)
var1 = data['row1']
var2 = data['row2']
var3 = data['row3']
var4 = data['row4']
v1 = [1, 1, 1, 1]
v2 = [2, 10, 5, 2]
x = []
for i in data.index:
    temp = []
    temp.append(var1[i])
    temp.append(var2[i])
    temp.append(var3[i])
    temp.append(var4[i])
    x.append(temp)
for epoch in range(10):
    cls1 = []
    cls2 = []
    for i in range(len(x)):
        if dist(x[i], v1)<dist(x[i], v2):
            cls1.append(x[i])
        else:
            cls2.append(x[i])
    # find new centre
    v1 = [0, 0, 0, 0]
    for i in range(len(cls1)):
        for j in range(len(cls1[0])):
            v1[j] += cls1[i][j]
    v1 = np.array(v1)
    if len(cls1)!= 0:
        v1 = v1/len(cls1)

    v2 = [0, 0, 0, 0]
    for i in range(len(cls2)):
        for j in range(len(cls2[0])):
            v2[j] += cls2[i][j]
    v2 = np.array(v2)
    if len(cls2)!= 0:
        v2 = v2/len(cls2)
```

PROBLEM 7

```python
# logistic regression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv
import math
from sklearn.cross_validation import train_test_split

class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=10, fit_intercept=True, verbose=False):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) #/ y.size
            self.theta -= self.lr * gradient

            # if(i % 10000 == 0):
            #     z = np.dot(X, self.theta)
            #     h = self.__sigmoid(z)
            #     print('loss: {self.__loss(h, y)} \t')
```

```
            #     h = self.__sigmoid(z)
            #     print('loss: {self.__loss(h, y)} \t')
            print(self.theta)

    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X, threshold):
        return self.predict_prob(X) >= threshold

def main():
    data = pd.read_excel('data3.xlsx')
    var1 = data['row1']
    var2 = data['row2']
    var3 = data['row3']
    var4 = data['row4']
    var5 = data['row5']
    x = []
    y = var5
    for i in data.index:
        temp = []
        # temp.append(1)
        temp.append(var1[i])
        temp.append(var2[i])
        temp.append(var3[i])
        temp.append(var4[i])
        x.append(temp)
    x = np.array(x)
    y = np.array(y)
    xtrain, xtest, ytrain, ytest = train_test_split(x, y, train_size=0.6)
    model = LogisticRegression()
    model.fit(xtrain, ytrain)
    preds = model.predict(xtest, 0.5)
    # print(preds)
    acc = (preds == ytest).mean()
    print(acc)
    # alpha=0.001; #learning rate
    # w = [0, 0, 0, 0, 0]
    # xtrain = np.array(xtrain)
    # ytrain = np.array(ytrain)
```

PROBLEM 8

```python
# logistic regression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv
import math
from sklearn.cross_validation import train_test_split

class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=1000, fit_intercept=False, verbose=False):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) #/ y.size
            self.theta -= self.lr * gradient

            # if(i % 10000 == 0):
            #     z = np.dot(X, self.theta)
            #     h = self.__sigmoid(z)
            #     print('loss: {self.__loss(h, y)} \t')
```

```
            #     print('loss: {self.__loss(h, y)} \t')
        # print(self.theta)
        return self.theta

    def prob(self, X, w):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, w))

def main():
    data = pd.read_excel('data4.xlsx')
    var1 = data['row1']
    var2 = data['row2']
    var3 = data['row3']
    var4 = data['row4']
    var5 = data['row5']
    x1 = []
    x2 = []
    x3 = []
    xmain = []
    for i in data.index:
        temp = []
        temp.append(var1[i])
        temp.append(var2[i])
        temp.append(var3[i])
        temp.append(var4[i])
        temp.append(var5[i])
        xmain.append(temp)
    xmain = np.array(xmain)
    xtra, xte, ytr, yte = train_test_split(xmain[:,:4], xmain[:, 4], train_size=0.6)
    # print(xtr)
    # print(ytr.T)
    xtr = []
    for i in range(len(ytr)):
        # temp = []
        # temp.append(xtra[i].tolist())
        temp = xtra[i].tolist()
        temp = np.append(temp, ytr[i].tolist()).tolist()
        xtr.append(temp)
    # print(xtr)
    for xx in xtr:
        if xx[4]==1 :
```

```
    x3t=x3
    x = np.concatenate((x1, x2), axis=0)
    x = [np.append(xx.tolist(),0) for xx in x]
    x3t = [np.append(xx.tolist(),1) for xx in x3t]
    x = np.concatenate((x, x3t), axis=0)
    x = np.array(x)
    # training
    model = LogisticRegression()
    wa = model.fit(x[:, :4], x[:, 4])
    # x.empty()

    # 1 and 3 vs 2
    x2t = x2
    xb = np.concatenate((x1, x3), axis=0)
    xb = [np.append(xx.tolist(),0) for xx in xb]
    x2t = [np.append(xx.tolist(),1) for xx in x2t]
    xb = np.concatenate((xb, x2t), axis=0)
    xb = np.array(xb)
    # training
    wb = model.fit(xb[:, :4], xb[:, 4])

    # 3 and 2 vs 1
    xc = np.concatenate((x3, x2), axis=0)
    xc = [np.append(xx.tolist(),0) for xx in xc]
    x1 = [np.append(xx.tolist(),1) for xx in x1]
    xc = np.concatenate((xc, x1), axis=0)
    xc = np.array(xc)
    # training
    wc = model.fit(xc[:, :4], xc[:, 4])
    # computing the accuracy
    y = []
    for xx in xte:
        a = model.prob(xx, wa)
        b = model.prob(xx, wb)
        c = model.prob(xx, wc)
        if a>b and a>c:
            y.append(3)
        elif b>a and b>c:
            y.append(2)
        else:
            y.append(1)

    y = np.array(y)
```

PROBLEM 9

```
        return self.__sigmoid(np.dot(X, w))

def main():
    data = pd.read_excel('data4.xlsx')
    var1 = data['row1']
    var2 = data['row2']
    var3 = data['row3']
    var4 = data['row4']
    var5 = data['row5']
    xmain = []
    for i in data.index:
        temp = []
        temp.append(var1[i])
        temp.append(var2[i])
        temp.append(var3[i])
        temp.append(var4[i])
        temp.append(var5[i])
        xmain.append(temp)
    xmain = np.array(xmain)
    # print(xmain)
    kfold = KFold(5, True, 1)
    for train, test in kfold.split(xmain):
        # xtra, xte, ytr, yte = train_test_split(xmain[:,:4], xmain[:, 4], train_size=0.6)
        # print(xmain[test])
        # print(ytr.T)
        xtra = xmain[train][:, :4]
        ytr = xmain[train][:, 4]
        xte = xmain[test][:, :4]
        yte = xmain[test][:, 4]
        xtr = []
        for i in range(len(ytr)):
            # temp = []
            # temp.append(xtra[i].tolist())
            temp = xtra[i].tolist()
            temp = np.append(temp, ytr[i].tolist()).tolist()
            xtr.append(temp)
        # print(xtr)
        x1 = []
        x2 = []
        x3 = []
        for xx in xtr:
            if xx[4]==1 :
```

```
        x1 = np.array(x1)
        x2 = np.array(x2)
        x3 = np.array(x3)

        # 1 and 2 vs 3
        x3t=x3
        x = np.concatenate((x1, x2), axis=0)
        x = [np.append(xx.tolist(),0) for xx in x]
        x3t = [np.append(xx.tolist(),1) for xx in x3t]
        x = np.concatenate((x, x3t), axis=0)
        x = np.array(x)
        # training
        model = LogisticRegression()
        wa = model.fit(x[:, :4], x[:, 4])
        # x.empty()

        # 1 and 3 vs 2
        x2t = x2
        xb = np.concatenate((x1, x3), axis=0)
        xb = [np.append(xx.tolist(),0) for xx in xb]
        x2t = [np.append(xx.tolist(),1) for xx in x2t]
        xb = np.concatenate((xb, x2t), axis=0)
        xb = np.array(xb)
        # training
        wb = model.fit(xb[:, :4], xb[:, 4])

        # 3 and 2 vs 1
        xc = np.concatenate((x3, x2), axis=0)
        xc = [np.append(xx.tolist(),0) for xx in xc]
        x1 = [np.append(xx.tolist(),1) for xx in x1]
        xc = np.concatenate((xc, x1), axis=0)
        xc = np.array(xc)
        # training
        wc = model.fit(xc[:, :4], xc[:, 4])
        # computing the accuracy
        y = []
        for xx in xte:
            a = model.prob(xx, wa)
            b = model.prob(xx, wb)
            c = model.prob(xx, wc)
            if a>b and a>c:
                y.append(3)
```

PROBLEM 10

```python
# likelihood ratio test (LRT)
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv
import math
from sklearn.cross_validation import train_test_split
from numpy.linalg import inv
from numpy import cov

def main():
    data = pd.read_excel('data3.xlsx')
    var1 = data['row1']
    var2 = data['row2']
    var3 = data['row3']
    var4 = data['row4']
    var5 = data['row5']
    x = []
    for i in data.index:
        temp = []
        # temp.append(1)
        temp.append(var1[i])
        temp.append(var2[i])
        temp.append(var3[i])
        temp.append(var4[i])
        temp.append(var5[i])
        x.append(temp)
    x = np.array(x)
    x1 = []
    x2 = []
    # print(x)
    # print('printed x\n')
    xtrain, xtest, ytrain, ytest = train_test_split(x[:,:4], x[:,4], train_size=0.6)
    for i in range(len(ytrain)):
        if ytrain[i] == 0:
            temp = xtrain[i].tolist()
            # temp.append(1)
            x1.append(temp)
        else:
            temp = xtrain[i].tolist()
            x2.append(temp)
```

```
    u1 = [0,0,0,0]
    u1 = np.array(u1)
    x1 = np.array(x1)
    for xx in x1:
        u1 = np.add(u1, xx)
    u1 = u1/len(x1)

    u2 = [0,0,0,0]
    u2 = np.array(u2)
    x2 = np.array(x2)
    for xx in x2:
        u2 = np.add(u2, xx)
    u2 = u2/len(x2)
    cov1 = cov(x[:,:4].T)
    cov1 = inv(cov1)
    print(cov1)
    # [[ 11.74272179  -7.24443147  -7.60918297   5.13458222]
    #  [ -7.24443147  11.4468994    7.29671775  -6.39827837]
    #  [ -7.60918297   7.29671775  17.29107951 -33.05113455]
    #  [  5.13458222  -6.39827837 -33.05113455  78.26853974]]

    w = np.dot(cov1, (u2 - u1))
    print(w)
    # [-0.09231329  0.91722705 -1.33392136  0.56030023]

    b = (1/2)*(np.dot(u2.T, np.dot(cov1, u2)) - np.dot(u1.T, np.dot(cov1, u1)))
    print(b)
    # -1.06923186932

    # computing the accuracy
    y = []
    for xx in xtest:
        temp = np.dot(w.T, xx) + b
        if temp<0:
            y.append(0)
        else:
            y.append(1)
    y = np.array(y)
    acc = (y == ytest).mean()
    print(acc)
    # 0.65

if __name__ == "__main__":
```

PROBLEM 11

```python
# logistic regression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv
import math
from sklearn.cross_validation import train_test_split

class MaximumLikelihood:
    def __init__(self, lr=0.01, num_iter=1000, fit_intercept=False, verbose=False):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) #/ y.size
            self.theta -= self.lr * gradient

            # if(i % 10000 == 0):
            #     z = np.dot(X, self.theta)
            #     h = self.__sigmoid(z)
            #     print('loss: {self.__loss(h, y)} \t')
        # print(self.theta)
        return self.theta
```

```
def Main():
    data = pd.read_excel('data4.xlsx')
    var1 = data['row1']
    var2 = data['row2']
    var3 = data['row3']
    var4 = data['row4']
    var5 = data['row5']
    x1 = []
    x2 = []
    x3 = []
    xmain = []
    for i in data.index:
        temp = []
        temp.append(var1[i])
        temp.append(var2[i])
        temp.append(var3[i])
        temp.append(var4[i])
        temp.append(var5[i])
        xmain.append(temp)
    xmain = np.array(xmain)
    xtra, xte, ytra, yte = train_test_split(xmain[:,:4], xmain[:, 4], train_size=0.6)
    # print(xtr)
    # print(ytr.T)
    xtr = []
    for i in range(len(ytr)):
        temp = xtra[i].tolist()
        temp = np.append(temp, ytr[i].tolist()).tolist()
        xtr.append(temp)
    # print(xtr)
    for xx in xtr:
        if xx[4]==1 :
            x1.append(xx[:4])
        elif xx[4]==2:
            x2.append(xx[:4])
        else :
            x3.append(xx[:4])

    x1 = np.array(x1)
    x2 = np.array(x2)
    x3 = np.array(x3)
    model = MaximumLikelihood()
    # training
    w = model.fit(xc[:, :4], xc[:, 4])
    # computing the accuracy
    y = []
```

PROBLEM 12

```python
# logistic regression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv
import math
from sklearn.cross_validation import train_test_split

class MaximumLikelihood:
    def __init__(self, lr=0.01, num_iter=1000, fit_intercept=False, verbose=False):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) #/ y.size
            self.theta -= self.lr * gradient

            # if(i % 10000 == 0):
            #     z = np.dot(X, self.theta)
            #     h = self.__sigmoid(z)
            #     print('loss: {self.__loss(h, y)} \t')
```

```
        #     print('loss: {self.__loss(h, y)} \t')
        # print(self.theta)
        return self.theta

    def prob(self, X, w):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, w))

def main():
    data = pd.read_excel('data4.xlsx')
    var1 = data['row1']
    var2 = data['row2']
    var3 = data['row3']
    var4 = data['row4']
    var5 = data['row5']
    x1 = []
    x2 = []
    x3 = []
    xmain = []
    for i in data.index:
        temp = []
        temp.append(var1[i])
        temp.append(var2[i])
        temp.append(var3[i])
        temp.append(var4[i])
        temp.append(var5[i])
        xmain.append(temp)
    xmain = np.array(xmain)
    xtra, xte, ytra, yte = train_test_split(xmain[:,:4], xmain[:, 4], train_size=0.6)
    # print(xtr)
    # print(ytr.T)
    xtr = []
    for i in range(len(ytr)):
        temp = xtra[i].tolist()
        temp = np.append(temp, ytr[i].tolist()).tolist()
        xtr.append(temp)
    # print(xtr)
    for xx in xtr:
        if xx[4]==1 :
            x1.append(xx[:4])
        elif xx[4]==2:
```

```
    xtra, xte, ytra, yte = train_test_split(xmain[:,:4], xmain[:, 4], train_size=0.6)
    # print(xtr)
    # print(ytr.T)
    xtr = []
    for i in range(len(ytr)):
        temp = xtra[i].tolist()
        temp = np.append(temp, ytr[i].tolist()).tolist()
        xtr.append(temp)
    # print(xtr)
    for xx in xtr:
        if xx[4]==1 :
            x1.append(xx[:4])
        elif xx[4]==2:
            x2.append(xx[:4])
        else :
            x3.append(xx[:4])

    x1 = np.array(x1)
    x2 = np.array(x2)
    x3 = np.array(x3)
    model = MaximumLikelihood()
    # training
    w = model.fit(xc[:, :4], xc[:, 4])
    # computing the accuracy
    y = []
    for xx in xte:
        a = model.prob(xx, wa)
        b = model.prob(xx, wb)
        c = model.prob(xx, wc)
        if a>b and a>c:
            y.append(3)
        elif b>a and b>c:
            y.append(2)
        else:
            y.append(1)

    y = np.array(y)

    acc = (yte == y).mean()
    print(acc)

if __name__ == "__main__":
    main()
```