EE 482 Computational Neuroscience

Homework 3

Cemal Güven Adal

21703986

Name: Cemal Güven Adal
ID: 21703986

Question 1

In this question blood oxygen levels responses from a human virtual cortex are given. In question 1 the given data about these response are used.

a) In part a it has been asked to use ridge regression method in order to fit the model. Formula for ridge regression is given below:

$$(X^T X + \lambda I)\mathbf{w} = \mathbf{X^T y} \qquad (1)$$

$$w = (X^T X + \lambda I)^{-1} X^T y \qquad (2)$$

As it can be seen from equation one and two ridge parameter are used for better fit the model into linearization. With different ridge parameters tested, ordinary least squares result which is when parameter is 0 and optimal result which is when parameter minimize the error is found in the question. While doing the ridge regression we will try to minimize the loss function.

Firstly python libraries are implemented and data is taken from the hw3_data2.mat as it is asked in the homework. Code for it can be seen below:

```
#importing libraries
import matplotlib.pyplot as plt
import numpy as np
import scipy.io as sio
import h5py
from numpy import random
from scipy.stats import norm
#loading .mat file to python
data1 = h5py.File('hw3_data2.mat','r')
Y=np.array(data1['Yn']).T
X=np.array(data1['Xn']).T
np.size(Y)
```

Yn is taken to Y and Xn is taken to X matrix for later computations. Then to do ridge regression a function is defined and formula in the equation 2 is implemented. Code for the function can be seen below:

```
def Ridge(landa,x,y):
    xt=np.transpose(x)
    w=np.linalg.inv((np.dot(xt,x)+landa*np.identity(100)))@(xt)@(y)
    return w
```

To do cross validation with 10 folds as it is asked in the homework first 100 contiguous samples are taken as test data , next 100 is taken as a validation data and the next 800 are taken as the training data. This validation had been done 10 times so each with test training and validation samples are all taken differently with order. Code for the 10 fold cross validation can be seen below:

```
def foldvalidation(x,y):
    c=np.logspace(0,12,num=1000,base=10)
```

Name: Cemal Güven Adal
ID: 21703986

```
Rvalidarray=np.zeros(1000)
Rtestarray=np.zeros(1000)
rv=0
for i in range(0,1000):
    landa=c[i]
    Rvalidtoplam=0
    Rtesttoplam=0
    warray=np.zeros((100,1))


    for a in range(0,10):

        if(a<9):
            xtest=x[100*a:(a+1)*100,:]
            ytest=y[100*a:(a+1)*100,:]
            xvalid=x[100*(a+1):(a+2)*100,:]
            yvalid=y[100*(a+1):(a+2)*100,:]
            xtrain=np.concatenate((x[100*(a+2):,:],x[0:(a)*100,:]),axis=0)
            ytrain=np.concatenate((y[100*(a+2):,:],y[0:a*100,:]),axis=0)
            w=Ridge(landa,xtrain,ytrain)
            yp=xtest@w
            ypvalid=xvalid@w
            Rvalid=np.sum((yvalid-ypvalid)*(yvalid-ypvalid))/100
            Rtest=np.sum((ytest-yp)*(ytest-yp))/100
            R1valid=1-(Rvalid/np.var(ytrain))
            R1test=1-(Rtest/np.var(ytrain))
            Rvalidtoplam=Rvalidtoplam+R1valid
            Rtesttoplam=Rtesttoplam+R1test
        else:
            xtest=x[900:,:]
            ytest=y[900:,:]
            xvalid=x[0:100,:]
            yvalid=y[0:100,:]
            xtrain=x[100:900,:]
            ytrain=y[100:900,:]
            w=Ridge(landa,xtrain,ytrain)
            yp=xtest@w
            ypvalid=xvalid@w
            Rvalid=np.sum((yvalid-ypvalid)*(yvalid-ypvalid))/100
            Rtest=np.sum((ytest-yp)*(ytest-yp))/100
            R1valid=1-(Rvalid/np.var(ytrain))
            R1test=1-(Rtest/np.var(ytest))
            Rvalidtoplam=Rvalidtoplam+R1valid
            Rtesttoplam=Rtesttoplam+R1test

    if(rv<Rvalidtoplam):
        w1=w
        i0=i
        rv=Rvalidtoplam
```

```
        Rvalidarray[i]=Rvalidtoplam/10
        Rtestarray[i]=Rtesttoplam/10
    return Rtestarray,Rvalidarray,w1,i0
```

For each cross validation coefficient of determination is calculated and stored. Also for the highest value of R^2 the value of w is stored and also returned because in the later we need the optimal ridge coefficient(w). R^2 vs lambda graph is plotted for lambda=0 (ols) and lambda optimal. Code for it can be seen below and plot can be seen below:

```
c=np.logspace(0,12,num=1000,base=10)
plt.figure()
Rtest,Rvalid,w,i=foldvalidation(X,Y)
plt.plot(c,Rtest)
plt.plot(c,Rvalid)
plt.plot()
plt.title("R^2 vs λ")
plt.ylabel("R^2")
plt.xlabel("λ")
plt.xscale("log")
plt.grid()
plt.show()
display(w)
```

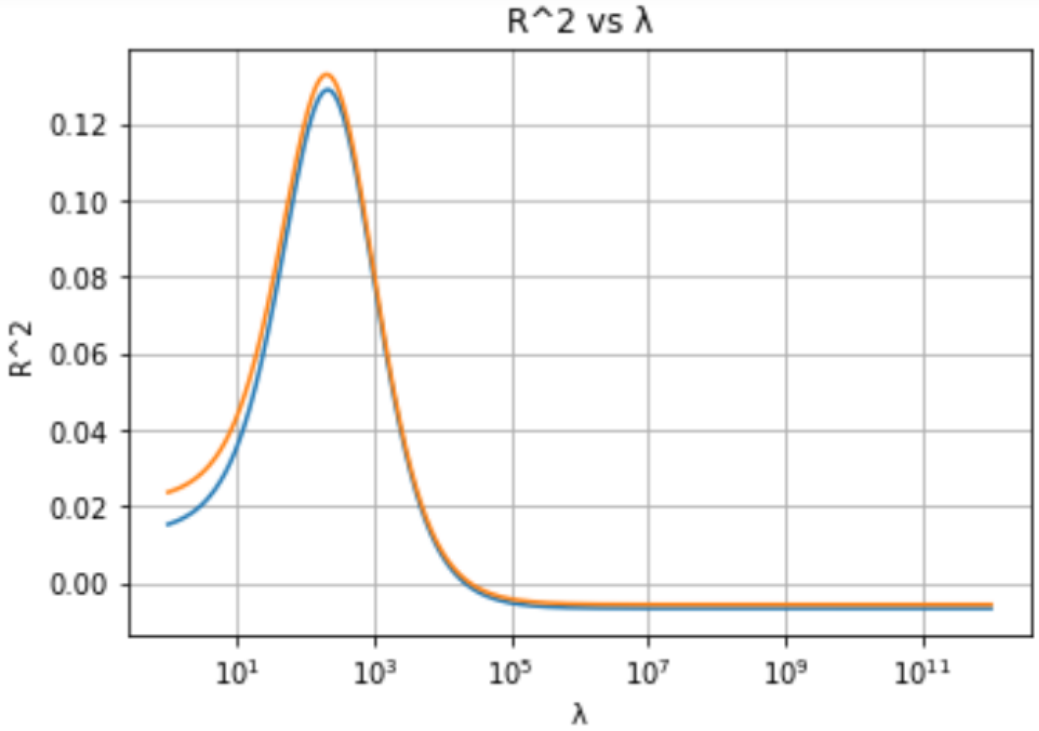

Figure 1:R^2 vs ols and optimum lambda(orange is optimum lambda and blue is for lambda=0)

Then from the array that holds the R^2 values of corresponding lambda is taken in the following code:

```
optlanda=np.argmax(Rtest)
print("lamda optimal =")
c[optlanda]
```

```
lamda optimal =

213.95888713434215
```

As it can seen from the result the optimal value for lambda is 213.95 .When lambda is too small model overfits and when its too large it cant be trained. In the value of 213 ridge regression works best.

B and C)

In the part b and C we are asked to generating 1000 samples from original data using bootstrap. For comparing and discussion purposes code , plot and resulting confidence indices are all discussed in the same part of the report. For this a bootstrap function has been written which randomly takes datas from original data set and creates a new dataset. Code for it can be seen below:

```
def bootstrap(x,y):
    bootx=np.zeros((1000,100))
    booty=np.zeros((1000,1))
    for j in range(1000):
        i = random.randint(1000)
        bootx[j,:]=x[i,:]
        booty[j,:]=y[i,:]
    return bootx,booty
```

500 bootstrap iterations has been done. Ridge regression for lambda=0 and optimal lambda with %95 confidence interval are plotted and for each indices are printed. In part b for lambda =0 is wanted and in part c for optimal lambda =213 is wanted. Code for it can be seen below:

```
#for ols
warrayols=np.zeros((500,100))
for i in range(500):
    bx,by=bootstrap(X,Y)
    w=Ridge(0,bx,by).reshape(1,100)
    warrayols[i,:]=w
meanols=np.mean(warrayols,axis=0,keepdims=True).reshape(100,1)
varols=np.var(warrayols,axis=0,keepdims=True).reshape(100,1)
boundols=(2.62*np.sqrt(varols)).reshape(100)
pols=meanols/((np.sqrt(varols)))
pols1=[]
for i in range(100):
    if (1.96<pols[i] or pols[i]<-1.96):
        pols1=np.append(pols1,i+1)
display("indexes of ols")
display(pols1)
#FOR OPTimum w
warrayopt=np.zeros((500,100))
for i in range(500):
    kx,ky=bootstrap(X,Y)
    w=Ridge(c[optlanda],kx,ky).reshape(1,100)
    warrayopt[i,:]=w
meanopt=np.mean(warrayopt,axis=0,keepdims=True).reshape(100,1)
```

```
varopt=np.var(warrayopt,axis=0,keepdims=True).reshape(100,1)
boundopt=(2.62*np.sqrt(varopt)).reshape(100)
x4=np.arange(1,101,1).reshape(100,1)

display(meanols)
display(meanols.shape)
plt.figure(figsize=(10,8))
plt.title("Ridge Regression with λ=0 ols and and %95 confidence interval")
plt.errorbar(x4,meanols,yerr=boundols,ecolor='r',elinewidth=1,fmt='k',capsize=2)
plt.ylabel("Weight values")
plt.xlabel("Weight indices")
plt.grid()
plt.show()
#part c
plt.figure(figsize=(10,8))
plt.title("Ridge Regression with λ=213 optimum and and %95 confidence interval")
plt.errorbar(x4,meanopt,yerr=boundopt,ecolor='r',elinewidth=1,fmt='k',capsize=2)
plt.ylabel("Weight values")
plt.xlabel("Weight indices")
plt.grid()
plt.show()
polopt=meanopt/((np.sqrt(varopt)))
pols2=[]
for i in range(100):
    if (1.96<polopt[i] or polopt[i]<-1.96):
        pols2=np.append(pols2,i+1)
display("indexes of opt")
display(pols2)
```

Resulting plots can be seen below:

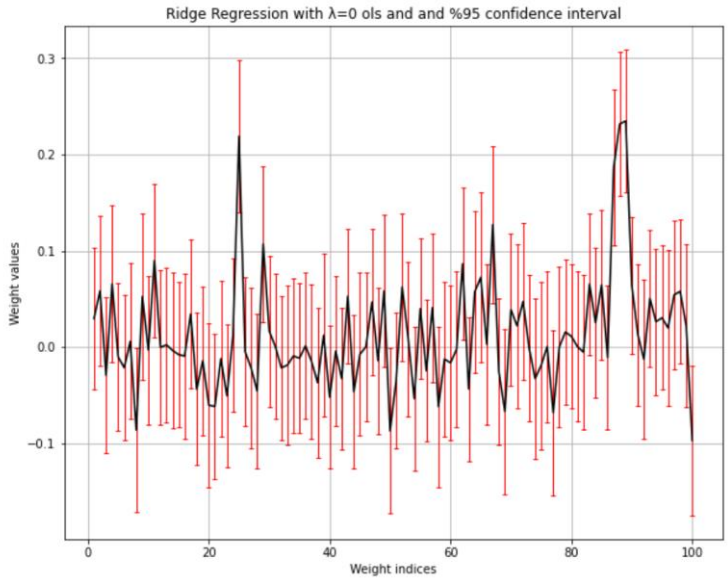

Figure 2: Weight values vs weight indices for lambda=0(for part b)

Name: Cemal Güven Adal
ID: 21703986

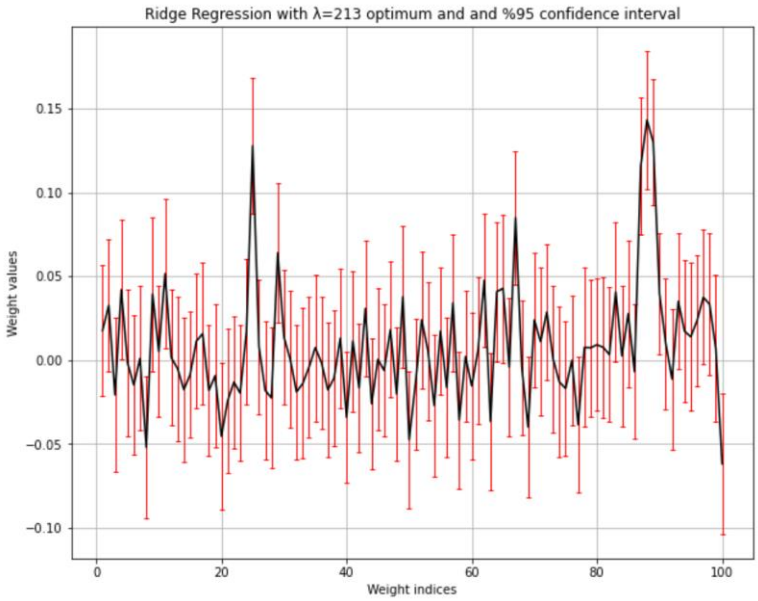

Figure 3: Weight values vs weight indices for optimal lambda lambda=213(for part c)

Indices for confidence interval for lambda=0 for part can be seen below:

```
array([  2.,   4.,   8.,  11.,  21.,  25.,  29.,  43.,  50.,  52.,  62.,
        65.,  67.,  69.,  77.,  83.,  85.,  87.,  88.,  89.,  90.,  98.,
       100.])
```

Indices for confidence interval for optimal lambda=213 for part c can be seen below:

```
array([  2.,   4.,   8.,   9.,  11.,  20.,  25.,  29.,  40.,  43.,  49.,
        50.,  57.,  58.,  62.,  63.,  64.,  65.,  67.,  69.,  77.,  83.,
        87.,  88.,  89.,  90.,  93.,  97.,  98., 100.])
```

Confidence interval is smaller when lambda is optimal because optimal lambda decreases variance thus confidence interval is smaller. When lambda is 0 confidence interval is larger because variance is larger.

Question 2

In this question we are given neural response measurements in the file hw3_data3.mat. This datasets are used for question 2 parts.

a) In the part a we are asked to calculate two sided p value from the 1000 samples we are created using bootstrap from pop 1 and pop 2. For this we will need to use null hypothesis test:

$$H_0 = \mu_1 - \mu_2 = 0 \tag{3}$$

$$H_A = \mu_1 - \mu_2 \neq 0 \tag{4}$$

Another bootstrap function is defined for part a in order to fill the arrays which has 7 and 5 which are different size randomly. Code for this can be seen below:

Name: Cemal Güven Adal
ID: 21703986

```
def bootstrap2(x,y):
    bootmerged=np.concatenate((x,y),axis=0)
    bootrandom=np.zeros((12,1))
    for j in range(12):
        i = random.randint(12)
        bootrandom[j,:]=bootmerged[i,:]
    bootx=bootrandom[0:7,:]
    booty=bootrandom[7:12,:]
    return bootx,booty
```

For hypothesis we are checking if two datasets are following the same distribution. I have checked the difference of means and plotted in a histogram. Also p and the z value are calculated. Code for it and the histogram can be seen below:

```
differencemean=[]
for j in range(10000):
    popboot1,popboot2=bootstrap2(pop1,pop2)
    meanpop1=np.mean(popboot1)
    meanpop2=np.mean(popboot2)

    differencemean=np.append(differencemean,meanpop1-meanpop2)
plt.figure()
plt.title("Population of the Difference Mean")
plt.ylabel("p(x)")
plt.xlabel("Difference of Means")
plt.hist(differencemean,bins=100,edgecolor='black')
plt.show()

vardifmean=np.std(differencemean)
meandifmean=np.mean(differencemean)
zvalue=abs((meandifmean/vardifmean))
print("z value")
print(zvalue)
p=2*(1-norm.cdf(zvalue))
print("two sided p value")
print(p)
```

Name: Cemal Güven Adal
ID: 21703986

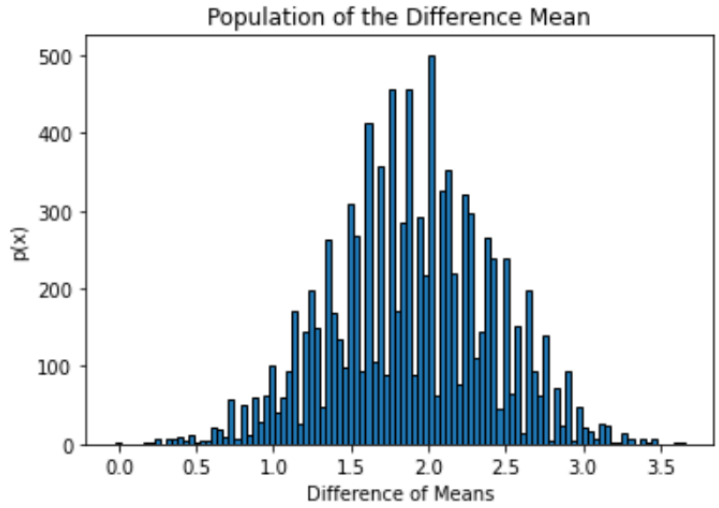

Figure 4: Population of the Difference of Means

```
z value
3.626967060731832
two sided p value
0.0002867698602062596
```

As it can be seen from the result the z value is 3.6269 and the two sided p value is really close to 0. This shows that hypothesis is true for small amount of samples. Thus this shows that they are not coming from same distribution thus from different populations. As the answer of the question in hint the result shows that we shouldn't have separated because they are from different populations.

b) In this part it two voxels are given in the human brain are stored in variables vox1 and vox2. I have used these datasets in this part. In this part we should take identical samples. Again we will generate more samples with bootstrap. Then these samples' correlations will be computed. 1.96 value is used when calculating the confidence interval because the %95 percent is given. In the following code corresponding bootstrap and correlation calculations are given:

```
#Part b
def bootstrap4(x,y):
    bootx=np.zeros((50,1))
    booty=np.zeros((50,1))
    for j in range(50):
        i = random.randint(50)
        bootx[j,:]=x[i,:]
        booty[j,:]=y[i,:]
    return bootx,booty
data3 = h5py.File('hw3_data3.mat','r')
vox1=np.array(data3['vox1'])
vox2=np.array(data3['vox2'])
np.size(vox2)
```

Name: Cemal Güven Adal
ID: 21703986

```
corr=[]
c1=np.zeros(10000)
for j in range(10000):
    voxboot1,voxboot2=bootstrap4(vox1,vox2)
    meanvox1=np.mean(voxboot1)
    meanvox2=np.mean(voxboot2)



    toplamvox=np.sum(voxboot1*voxboot2)
    c1[j]=((1/50)*toplamvox-meanvox1*meanvox2)/(np.std(voxboot1)*np.std(voxboot2))


plt.figure()
plt.title("Population of the Difference Correlation")
plt.ylabel("p(x)")
plt.xlabel("Correlation")
plt.hist(c1,bins=100,edgecolor='black')
plt.show()
meancor=np.mean(c1)
varcor=np.var(c1)
print("mean of correlation")
print(meancor)
print("std of correlation")
print(np.sqrt(varcor))
upconfidence=meancor+1.96*np.sqrt(varcor)
lowconfidence=meancor-1.96*np.sqrt(varcor)
print("upper interval")
print(upconfidence)
print("lower interval")
print(lowconfidence)
#Percentile of bootstrap
#Percentile is 0 because there is no pdf at 0 as it can be seen in histogram
```

Name: Cemal Güven Adal
ID: 21703986

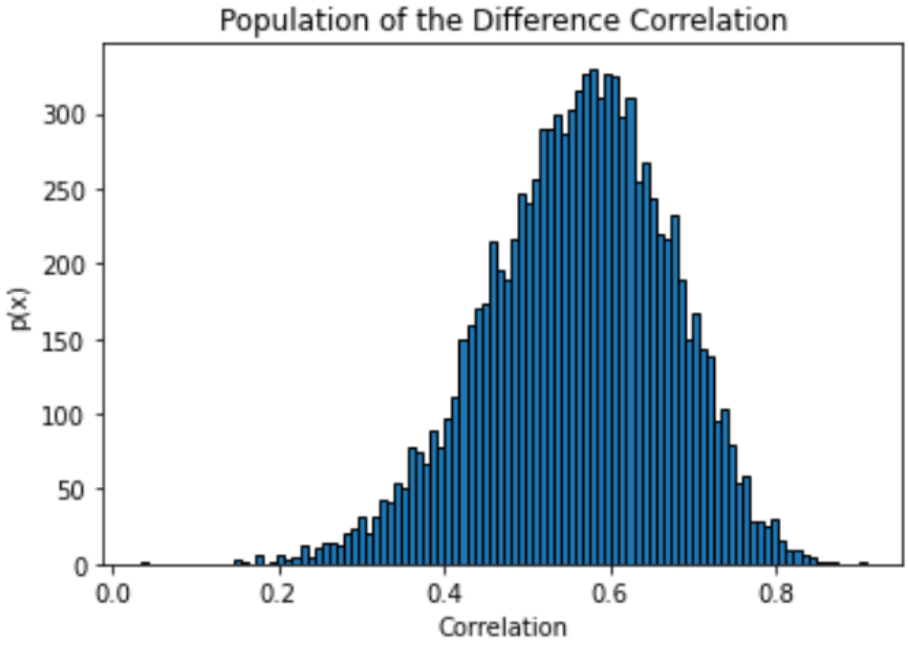

Figure 5: Population of the Difference Correlation

```
mean of correlation
0.5598539533782979
std of correlation
0.10956345214981228
upper interval
0.7745983195919299
lower interval
0.34510958716466583
```

As it can be seen from the result mean of correlation is 0.559, standard deviation of correlation is 0.109, upper confidence interval is 0.774 and lower confidence interval is 0.345. In the question it is also asked that percentile of bootstrap distribution to a correlation value of 0. As it can be seen from the histogram in figure 5 there is no value at 0 so the percentile is %0.

c) Ba In this part we are asked to do another hypothesis testing. From equations 3 and 4 it is taken as:

H_0 is voxel response have 0 correlation, H_A is voxel response have positive correlation. In this part bootstrap sampling is done independently so different bootstrap function is written for picking different values from vox1 and vox2. Code for this bootstrap method and for calculating correlation vox1 and vox2 which is done similarly to part b is given below:

```
#Part c
def bootstrap3(x,y):
    bootx=np.zeros((50,1))
    booty=np.zeros((50,1))
    for j in range(50):
        i = random.randint(50)
        bootx[j,:]=x[i,:]
```

```
        booty[j,:]=y[i,:]
    for j in range(50):
        i = random.randint(50)
        booty[j,:]=y[i,:]
    return bootx,booty
data3 = h5py.File('hw3_data3.mat','r')
vox1=np.array(data3['vox1'])
vox2=np.array(data3['vox2'])
np.size(vox2)
corr=[]
c1=np.zeros(10000)
for j in range(10000):
    voxboot1,voxboot2=bootstrap3(vox1,vox2)
    meanvox1=np.mean(voxboot1)
    meanvox2=np.mean(voxboot2)



    toplamvox=np.sum(voxboot1*voxboot2)
    c1[j]=((1/50)*toplamvox-meanvox1*meanvox2)/(np.std(voxboot1)*np.std(voxboot2))


plt.figure()
plt.title("Population of the Difference Correlation")
plt.ylabel("p(x)")
plt.xlabel("Correlation")
plt.hist(c1,bins=100,edgecolor='black')
plt.show()
meancor=np.mean(c1)
varcor=np.var(c1)
print("mean of correlation")
print(meancor)
print("std of correlation")
print(np.sqrt(varcor))
upconfidence=meancor+1.96*np.sqrt(varcor)
lowconfidence=meancor-1.96*np.sqrt(varcor)
print("upper interval")
print(upconfidence)
print("lower interval")
print(lowconfidence)
zc=np.abs(meancor/varcor)
print("z value for c")
print(zc)

pvaluec=(norm.cdf(zc))
print("one sided p value for c")
print(pvaluec)
```

Name: Cemal Güven Adal
ID: 21703986

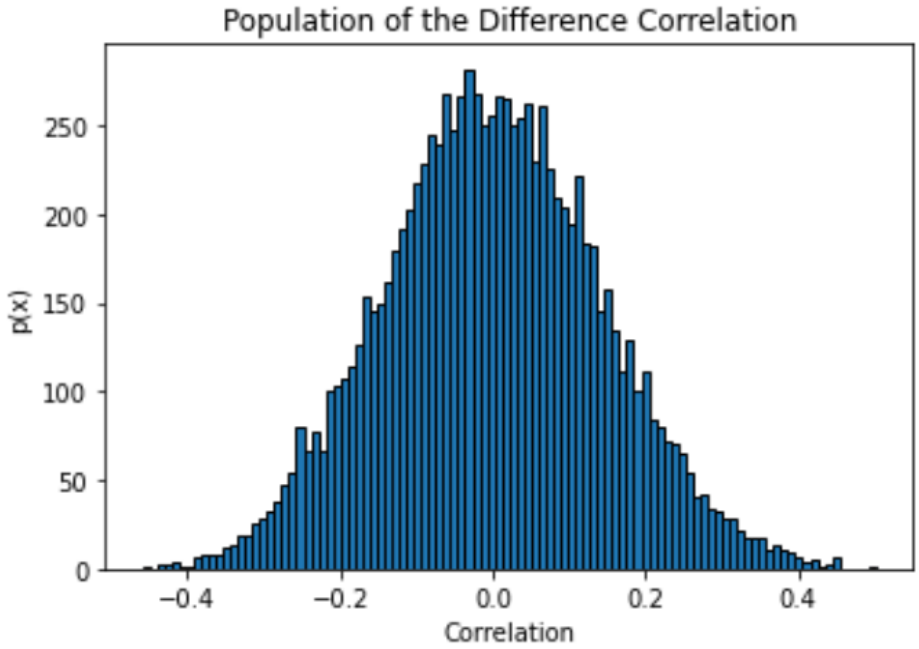

Figure 6: Population of the Difference Correlation

```
mean of correlation
-0.0007618759406318814
std of correlation
0.1422945170096907
upper interval
0.2781353773983619
lower interval
-0.27965912927962566
z value for c
0.037627722059380385
one sided p value for c
0.515007747719212
```

As it can be seen from the result mean of correlation is close to 0, standard deviation is 0.142, upper confidence interval is 0.278 , the lower confidence interval is -0.279, z value is 0.037 and one sided p value is 0.515.

d) In this question we are given BOLD responses in a face selective region. We are given datasets from two experiments face and building. These datasets are used in this question. Again null hypothesis testing will be made similar to previous questions with different hypothesis. From equations 3 and 4 hypothesis is:

H_0 is difference between building and face responses, H_A is no difference between building and face responses. Again we can generate 10000 samples using bootstrap method from same indexes of face and building datasets. Code for this and results can be seen below:

Name: Cemal Güven Adal
ID: 21703986

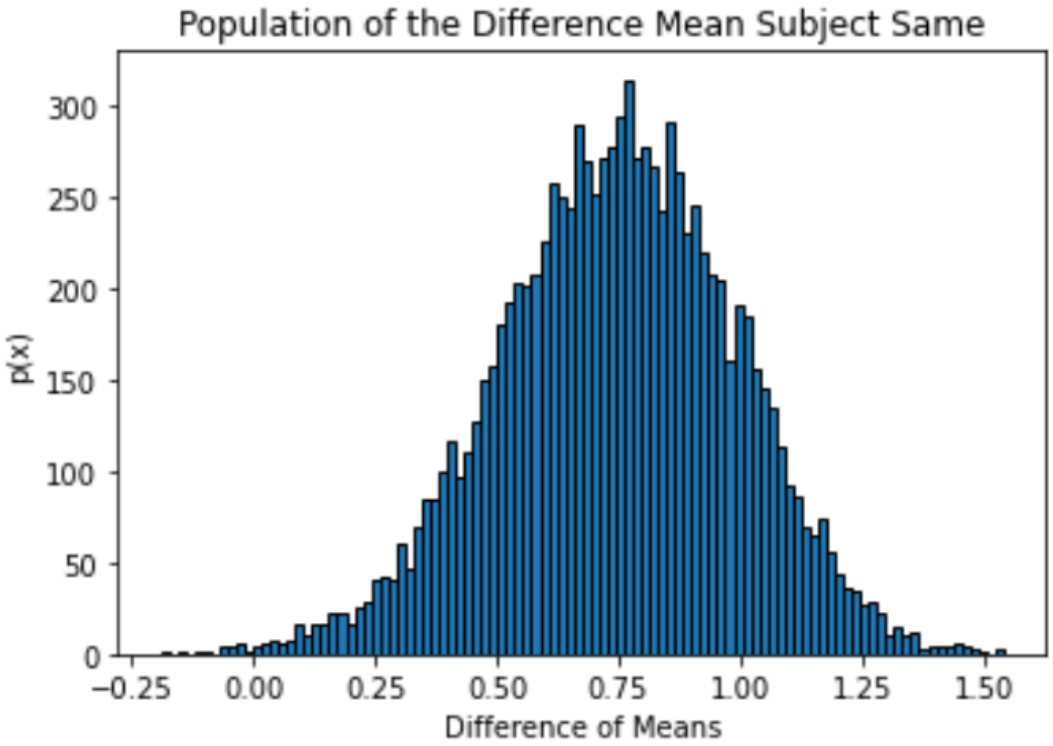

Figure 7: Population of the Mean Difference for Subject Same

```
z value
3.0690771900274165
two sided p value
0.002147211042149877
```

As it can be seen from the result z value is 3.06 and two sided p value 0.0021 which is very small. Because it is very small it can be concluded that building and face images are not creating the same response for the same population.

e) In this question we will conduct the same experiment in part d but in part e we will assume that groups are different. For this purpose different bootstrap method will be used for creating 10000 samples. Mean difference method will be used as in the previous part. Then two sided p value and z value will be computed. Code and result for it can be seen below:

```
#part e
def bootstrap6(x,y):
    bootx=np.zeros((20,1))
    booty=np.zeros((20,1))
    for j in range(20):
        i = random.randint(20)
        bootx[j,:]=x[i,:]
        booty[j,:]=y[i,:]
    for j in range(20):
        i = random.randint(20)
        booty[j,:]=y[i,:]
```

```
    return bootx,booty
differencemean=[]
for j in range(10000):
    popboot1,popboot2=bootstrap6(face,build)
    meanpop1=np.mean(popboot1)
    meanpop2=np.mean(popboot2)

    differencemean=np.append(differencemean,meanpop1-meanpop2)
plt.figure()
plt.title("Population of the Difference Mean Subject Different")
plt.ylabel("p(x)")
plt.xlabel("Difference of Means")
plt.hist(differencemean,bins=100,edgecolor='black')
plt.show()

vardifmean=np.std(differencemean)
meandifmean=np.mean(differencemean)
zvalue=abs((meandifmean/vardifmean))
print("z value")
print(zvalue)
p=2*(1-norm.cdf(zvalue))
print("two sided p value")
print(p)
```

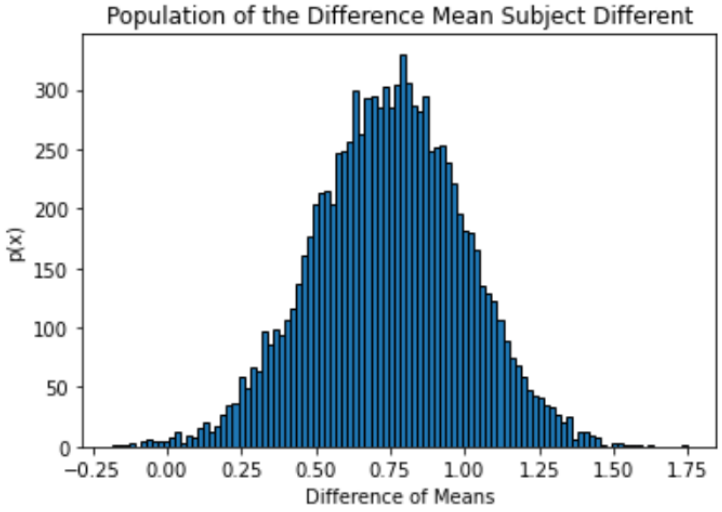

Figure 8: Population of the Mean Difference for Subject Different

```
z value
2.9229666328028805
two sided p value
0.003467136691003736
```

As it can be seen from the result z value is 2.922 and p value is 0.003. Again we have really small p value which means that null hypothesis is wrong. This means that there is a difference between face and building images responses.

Name: Cemal Güven Adal
ID: 21703986

References

Greenberg, M. D. (1988). *Advanced engineering mathematics*. Englewood Cliffs, NJ: Prentice-Hall Internat.

Bertsekas, D. P., & Tsitsiklis, J. N. (2008). *Introduction to probability: Dimitri P. Bertsekas and John N. Tsitsikis*. Belmont, Massachussets: Athena Scientific.

Appendices

Code

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:




#importing libraries
import matplotlib.pyplot as plt
import numpy as np
import scipy.io as sio
import h5py
from numpy import random
from scipy.stats import norm
def CemalGuven_Adal_21703986_hw3(question):
    if question == '1' :
        #------------------------------------------------------------------------Question1----------
----------------------------------------------
        #loading .mat file to python
        data1 = h5py.File('hw3_data2.mat','r')
        Y=np.array(data1['Yn']).T
        X=np.array(data1['Xn']).T
        np.size(Y)




        def Ridge(landa,x,y):
            xt=np.transpose(x)
            w=np.linalg.inv((np.dot(xt,x)+landa*np.identity(100)))@(xt)@(y)
            return w
```

Name: Cemal Güven Adal
ID: 21703986

```
def foldvalidation(x,y):
    c=np.logspace(0,12,num=1000,base=10)
    Rvalidarray=np.zeros(1000)
    Rtestarray=np.zeros(1000)
    rv=0
    for i in range(0,1000):
        landa=c[i]
        Rvalidtoplam=0
        Rtesttoplam=0
        warray=np.zeros((100,1))


        for a in range(0,10):

            if(a<9):
                xtest=x[100*a:(a+1)*100,:]
                ytest=y[100*a:(a+1)*100,:]
                xvalid=x[100*(a+1):(a+2)*100,:]
                yvalid=y[100*(a+1):(a+2)*100,:]
                xtrain=np.concatenate((x[100*(a+2):,:],x[0:(a)*100,:]),axis=0)
                ytrain=np.concatenate((y[100*(a+2):,:],y[0:a*100,:]),axis=0)
                w=Ridge(landa,xtrain,ytrain)
                yp=xtest@w
                ypvalid=xvalid@w
                Rvalid=np.sum((yvalid-ypvalid)*(yvalid-ypvalid))/100
                Rtest=np.sum((ytest-yp)*(ytest-yp))/100
                R1valid=1-(Rvalid/np.var(ytrain))
                R1test=1-(Rtest/np.var(ytrain))
                Rvalidtoplam=Rvalidtoplam+R1valid
                Rtesttoplam=Rtesttoplam+R1test
            else:
                xtest=x[900:,:]
                ytest=y[900:,:]
                xvalid=x[0:100,:]
                yvalid=y[0:100,:]
                xtrain=x[100:900,:]
                ytrain=y[100:900,:]
                w=Ridge(landa,xtrain,ytrain)
                yp=xtest@w
                ypvalid=xvalid@w
                Rvalid=np.sum((yvalid-ypvalid)*(yvalid-ypvalid))/100
                Rtest=np.sum((ytest-yp)*(ytest-yp))/100
                R1valid=1-(Rvalid/np.var(ytrain))
                R1test=1-(Rtest/np.var(ytest))
                Rvalidtoplam=Rvalidtoplam+R1valid
                Rtesttoplam=Rtesttoplam+R1test
```

```
        if(rv<Rvalidtoplam):
            w1=w
            i0=i
            rv=Rvalidtoplam

        Rvalidarray[i]=Rvalidtoplam/10
        Rtestarray[i]=Rtesttoplam/10
    return Rtestarray,Rvalidarray,w1,i0
c=np.logspace(0,12,num=1000,base=10)
plt.figure()
Rtest,Rvalid,w,i=foldvalidation(X,Y)
plt.plot(c,Rtest)
plt.plot(c,Rvalid)
plt.plot()
plt.title("R^2 vs λ")
plt.ylabel("R^2")
plt.xlabel("λ")
plt.xscale("log")
plt.grid()
plt.show()
display(w)


optlanda=np.argmax(Rtest)
print("lamda optimal =")
c[optlanda]


def bootstrap(x,y):
    bootx=np.zeros((1000,100))
    booty=np.zeros((1000,1))
    for j in range(1000):
        i = random.randint(1000)
        bootx[j,:]=x[i,:]
        booty[j,:]=y[i,:]
    return bootx,booty
#for ols
warrayols=np.zeros((500,100))
for i in range(500):
    bx,by=bootstrap(X,Y)
    w=Ridge(0,bx,by).reshape(1,100)
    warrayols[i,:]=w
meanols=np.mean(warrayols,axis=0,keepdims=True).reshape(100,1)
varols=np.var(warrayols,axis=0,keepdims=True).reshape(100,1)
boundols=(2.62*np.sqrt(varols)).reshape(100)
pols=meanols/((np.sqrt(varols)))
pols1=[]
for i in range(100):
    if (1.96<pols[i] or pols[i]<-1.96):
        pols1=np.append(pols1,i+1)
```

```
    display("indexes of ols")
    display(pols1)
    #FOR OPTimum w
    warrayopt=np.zeros((500,100))
    for i in range(500):
       kx,ky=bootstrap(X,Y)
       w=Ridge(c[optlanda],kx,ky).reshape(1,100)
       warrayopt[i,:]=w
    meanopt=np.mean(warrayopt,axis=0,keepdims=True).reshape(100,1)
    varopt=np.var(warrayopt,axis=0,keepdims=True).reshape(100,1)
    boundopt=(2.62*np.sqrt(varopt)).reshape(100)
    x4=np.arange(1,101,1).reshape(100,1)

    display(meanols)
    display(meanols.shape)
    plt.figure(figsize=(10,8))
    plt.title("Ridge Regression with λ=0 ols and and %95 confidence interval")
    plt.errorbar(x4,meanols,yerr=boundols,ecolor='r',elinewidth=1,fmt='k',capsize=2)
    plt.ylabel("Weight values")
    plt.xlabel("Weight indices")
    plt.grid()
    plt.show()
    #part c
    plt.figure(figsize=(10,8))
    plt.title("Ridge Regression with λ=213 optimum and and %95 confidence interval")
    plt.errorbar(x4,meanopt,yerr=boundopt,ecolor='r',elinewidth=1,fmt='k',capsize=2)
    plt.ylabel("Weight values")
    plt.xlabel("Weight indices")
    plt.grid()
    plt.show()
    polopt=meanopt/((np.sqrt(varopt)))
    pols2=[]
    for i in range(100):
       if (1.96<polopt[i] or polopt[i]<-1.96):
          pols2=np.append(pols2,i+1)
    display("indexes of opt")
    display(pols2)
elif question == '2' :

    #-------------------QUESION2------------


    #Question 2
    #part a
    #loading .mat file to python
    data2 = h5py.File('hw3_data3.mat','r')
    pop1=np.array(data2['pop1'])
    pop2=np.array(data2['pop2'])
    np.size(pop1)
```

Name: Cemal Güven Adal
ID: 21703986

```
def bootstrap2(x,y):
    bootx=np.zeros(7)
    booty=np.zeros(5)
    for j in range(7):
        i = random.randint(7)
        bootx[j]=x[i,:]
    for j in range(5):
        i = random.randint(5)
        booty[j]=y[i,:]
    return bootx,booty
differencemean=[]
for j in range(10000):
    popboot1,popboot2=bootstrap2(pop1,pop2)
    meanpop1=np.mean(popboot1)
    meanpop2=np.mean(popboot2)

    differencemean=np.append(differencemean,meanpop1-meanpop2)
plt.figure()
plt.title("Population of the Difference Mean")
plt.ylabel("p(x)")
plt.xlabel("Difference of Means")
plt.hist(differencemean,bins=100,edgecolor='black')
plt.show()

vardifmean=np.std(differencemean)
meandifmean=np.mean(differencemean)
zvalue=abs((meandifmean/vardifmean))
print("z value")
print(zvalue)
p=2*(1-norm.cdf(zvalue))
print("two sided p value")
print(p)



#Part b
def bootstrap4(x,y):
    bootx=np.zeros((50,1))
    booty=np.zeros((50,1))
    for j in range(50):
        i = random.randint(50)
        bootx[j,:]=x[i,:]
        booty[j,:]=y[i,:]
    return bootx,booty
data3 = h5py.File('hw3_data3.mat','r')
vox1=np.array(data3['vox1'])
vox2=np.array(data3['vox2'])
np.size(vox2)
corr=[]
```

```
    c1=np.zeros(10000)
    for j in range(10000):
        voxboot1,voxboot2=bootstrap4(vox1,vox2)
        meanvox1=np.mean(voxboot1)
        meanvox2=np.mean(voxboot2)



        toplamvox=np.sum(voxboot1*voxboot2)
        c1[j]=((1/50)*toplamvox-
meanvox1*meanvox2)/(np.std(voxboot1)*np.std(voxboot2))



    plt.figure()
    plt.title("Population of the Difference Correlation")
    plt.ylabel("p(x)")
    plt.xlabel("Correlation")
    plt.hist(c1,bins=100,edgecolor='black')
    plt.show()
    meancor=np.mean(c1)
    varcor=np.var(c1)
    print("mean of correlation")
    print(meancor)
    print("std of correlation")
    print(np.sqrt(varcor))
    upconfidence=meancor+1.96*np.sqrt(varcor)
    lowconfidence=meancor-1.96*np.sqrt(varcor)
    print("upper interval")
    print(upconfidence)
    print("lower interval")
    print(lowconfidence)
    #Percentile of bootstrap
    #Percentile is 0 because there is no pdf at 0 as it can be seen in histogram



    #Part c
    def bootstrap3(x,y):
        bootx=np.zeros((50,1))
        booty=np.zeros((50,1))
        for j in range(50):
            i = random.randint(50)
            bootx[j,:]=x[i,:]
            booty[j,:]=y[i,:]
        for j in range(50):
            i = random.randint(50)
            booty[j,:]=y[i,:]
        return bootx,booty
    data3 = h5py.File('hw3_data3.mat','r')
    vox1=np.array(data3['vox1'])
    vox2=np.array(data3['vox2'])
    np.size(vox2)
```

```
corr=[]
c1=np.zeros(10000)
for j in range(10000):
   voxboot1,voxboot2=bootstrap3(vox1,vox2)
   meanvox1=np.mean(voxboot1)
   meanvox2=np.mean(voxboot2)


   toplamvox=np.sum(voxboot1*voxboot2)
   c1[j]=((1/50)*toplamvox-
meanvox1*meanvox2)/(np.std(voxboot1)*np.std(voxboot2))


plt.figure()
plt.title("Population of the Difference Correlation")
plt.ylabel("p(x)")
plt.xlabel("Correlation")
plt.hist(c1,bins=100,edgecolor='black')
plt.show()
meancor=np.mean(c1)
varcor=np.var(c1)
print("mean of correlation")
print(meancor)
print("std of correlation")
print(np.sqrt(varcor))
upconfidence=meancor+1.96*np.sqrt(varcor)
lowconfidence=meancor-1.96*np.sqrt(varcor)
print("upper interval")
print(upconfidence)
print("lower interval")
print(lowconfidence)
zc=np.abs(meancor/varcor)
print("z value for c")
print(zc)

pvaluec=(norm.cdf(zc))
print("one sided p value for c")
print(pvaluec)



#part d
#taking data
data4 = h5py.File('hw3_data3.mat','r')
face=np.array(data4['face'])
build=np.array(data4['building'])
np.size(face)

def bootstrap5(x,y):
   bootx=np.zeros((20,1))
```

```
    booty=np.zeros((20,1))
    for j in range(20):
        i = random.randint(20)
        bootx[j,:]=x[i,:]
        booty[j,:]=y[i,:]
    return bootx,booty
differencemean=[]
for j in range(10000):
    popboot1,popboot2=bootstrap5(face,build)
    meanpop1=np.mean(popboot1)
    meanpop2=np.mean(popboot2)

    differencemean=np.append(differencemean,meanpop1-meanpop2)
plt.figure()
plt.title("Population of the Difference Mean Subject Same")
plt.ylabel("p(x)")
plt.xlabel("Difference of Means")
plt.hist(differencemean,bins=100,edgecolor='black')
plt.show()

vardifmean=np.std(differencemean)
meandifmean=np.mean(differencemean)
zvalue=abs((meandifmean/vardifmean))
print("z value")
print(zvalue)
p=2*(1-norm.cdf(zvalue))
print("two sided p value")
print(p)


#part e
def bootstrap6(x,y):
    bootx=np.zeros((20,1))
    booty=np.zeros((20,1))
    for j in range(20):
        i = random.randint(20)
        bootx[j,:]=x[i,:]
        booty[j,:]=y[i,:]
    for j in range(20):
        i = random.randint(20)
        booty[j,:]=y[i,:]
    return bootx,booty
differencemean=[]
for j in range(10000):
    popboot1,popboot2=bootstrap6(face,build)
    meanpop1=np.mean(popboot1)
    meanpop2=np.mean(popboot2)

    differencemean=np.append(differencemean,meanpop1-meanpop2)
plt.figure()
```

```
        plt.title("Population of the Difference Mean Subject Different")
        plt.ylabel("p(x)")
        plt.xlabel("Difference of Means")
        plt.hist(differencemean,bins=100,edgecolor='black')
        plt.show()

        vardifmean=np.std(differencemean)
        meandifmean=np.mean(differencemean)
        zvalue=abs((meandifmean/vardifmean))
        print("z value")
        print(zvalue)
        p=2*(1-norm.cdf(zvalue))
        print("two sided p value")
        print(p)

question=input("enter question number")
CemalGuven_Adal_21703986_hw3(question)


# In[ ]:
```