Shirui Ye
CS542
PS2
Cover

I tried to use Latex and work on Overleaf. However, I found out it took me 1 hr and 15min just for first question, since It took a lot of time to type in the math symbols and doing so really interrupted my logic. In this case, to give my best effort, I decide to hand write this assignment. Thanks for the understanding.

Shirui Ye
CS542
PS2

3.3 (i) data dependent noise variance

we have $E_D(w)=\frac{1}{2}\sum_{n=1}^{N} r_n\{t_n-w^T\phi(x_n)\}^2$.

we take the $E_D(w)$' (derivative) then set it to 0, finally, we solve for w which is the answer. The process is below:

$$\frac{d\,E_D(w)}{dw}=-\sum_{n=1}^{N} r_n\{t_n-W^T\phi(X_n)\}\phi(X_n)=0$$

$$W=\left(\sum_{n=1}^{N} r_n t_n\phi(X_n)\right)\left(\sum_{n=1}^{N} r_n\phi(X_n)\phi(X_n)^T\right)^{-1}$$

(ii) replicated data points

we define $R=\mathrm{diagonal}(r_1,r_2,\ldots,r_n)$

Then we use matrix products to write the error function

$$E_D(w)=R(\phi w-t)\frac{1}{2}(\phi w-t)^T$$

$$=\frac{1}{2}(W^T\phi^T R\phi W-2t^T R\phi W+t^T R t) \quad \cdots\cdots R=\mathrm{diagonal}(r_1,r_2,\ldots,r_n)$$

Finally, get gradient of the error function

$$\nabla E_D(w)=\phi^T R\phi w-t^T R\phi$$

$$w=Rt(\phi^T R\phi)^{-1}\phi^T$$

Shirui Ye
CS542
PS2

3.11

According to the linear regression function given by 3.59, we have

$$\sigma^2_{N+1}(x) = \phi(x)^T S_{N+1} \phi(x) + \frac{1}{B}$$

Then, based on $\ln p(t|w,\beta) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(w)$

we have $S_{N+1} = (S_N^{-1} + B\phi_{N+1} + \phi^T_{N+1})^{-1} = S_N - \frac{\beta S_N \phi_{N+1} + \phi^T_{N+1} S_N}{\beta \phi^T_{N+1} S_N \phi_{N+1} + 1}$

combine with previous formula, we get

$$\sigma^2_{N+1}(x) = \phi(x)^T\phi(x)\left(S_N - \frac{\beta S_N \phi_{N+1}\phi^T_{N+1} S_N}{1+\beta\phi^T_{N+1} S_N \phi_{N+1}}\right) + \frac{1}{B}$$

$$= \sigma^2_N(x) - \frac{\beta\phi(x)^T S_N \phi_{N+1}\phi^T_{N+1} S_N \phi(x)}{\beta\phi^T_{N+1} S_N \phi_{N+1} + 1}$$

we know $S_N \geqslant 0$, so top: $\beta\phi(x)^T S_N \phi_{N+1}\phi^T_{N+1} S_N \phi(x) \geqslant 0$

bottom: $\beta\phi^T_{N+1} S_N \phi_{N+1} + 1 \geqslant 0$

Thus, we successfully show $\sigma^2_N(x)$ satisfies $\sigma^2_{N+1}(x) \leq \sigma^2_N(x)$

Shirui Ye
CS542
PS2

3.14

we know we suppose $\phi_j(x)$ are linearly independent and $\phi_0(x)=1$. we have 3.115 and need to prove $\sum_{n=1}^{N} k(x,x_n)=1$

According to the question, we have $\alpha=0$, we know $S_N=(\beta\phi^T\Phi)^{-1}$

Then, $k(x,x')=\psi(x)^T\psi(x')$

we know $\psi(x)=V\phi(x)$ $V=M\times M$ $\phi(x)=V^{-1}\phi(n)$

According to $\psi(x)=V\phi(x)$ and $\Phi=\begin{pmatrix}\phi_0(x_1) & \cdots & \phi_{M-1}(x_1) \\ \vdots & & \vdots \\ \phi(x_N) & \cdots & \phi_{M-1}(x_N)\end{pmatrix}$

we have $\Psi=\Phi V^T$, then $\Phi=\Psi V^{-T}$ $[V^{-T}=(V^{-1})^T \Rightarrow \Psi^{-T}\Psi=I)$

Now we can have $S_N=\beta^{-1}(\Phi^T\Phi)^{-1}=\beta^{-1}VV^T$

$\psi(x)=1$ $\sum_{n=1}^{N}\psi_i(x_n)\psi_i(x_n)=\sum_{n=1}^{N}\psi_i(x_n)=S_{iI}$ $\nearrow$ we also know $\begin{bmatrix}\psi_1(x_1) \\ \vdots \\ \psi_n(x)\end{bmatrix}[\psi_1(x_1)\cdots\psi_n(x_n)]=I$

Finally, $\sum_{n=1}^{N}k(x,x_n)=\sum_{n=1}^{N}\psi(x)^T\psi(x_n)=\sum\psi_i(x)S_{iI}=\psi_I(x)=1$

$\therefore$ Thus, we successfully show that the kernel satisfies the summation constraint $\sum_{n=1}^{N}k(x,x_n)=1$

Shirui Ye
CS542
PS2

3.21

We have $\frac{d}{d\alpha}\ln|A| = Tr(A^{-1}\frac{d}{d\alpha}A)$

Now we follow the question, consider the eigenvalue expansion of a real symmetric matrix A, we write A in $Au_i = u_i u_i$

Then $\ln|A| = \ln \prod_{i=1}^{m} u_i = \sum_{i=1}^{m} \ln u_i$ and $\frac{d}{d\alpha}\ln|A| = \sum_{i=1}^{m}\frac{1}{u_i}\frac{d}{d\alpha}u_i$

We now expand $Au_i = \lambda u_i$ by an expansion of its eigenvalue, we have $A = \sum_{i=1}^{M}\lambda_i u_i u_i^T$, Then A^{-1} can be expressed $A^{-1} = \sum_{i=1}^{M}\frac{1}{\lambda_i}u_i u_i^T$

Now, let's work on $Tr(A^{-1}\frac{d}{d\alpha}A) = Tr(\sum_{i=1}^{M}\frac{1}{\lambda_i}u_i u_i^T \frac{d}{d\alpha}\sum_{k=1}^{M}\lambda_k u_k u_k^T)$

$$= Tr(\sum_{i=1}^{M}\frac{1}{\lambda_i}u_i u_i^T [\sum_{k=1}^{M}\frac{du_k}{d\alpha}u_k u_k^T + \lambda_k(\frac{du_k}{d\alpha}u_k^T + u_k\frac{du_k^T}{d\alpha})])$$

$$= Tr(\sum_{i=1}^{M}\frac{1}{\lambda_i}u_i u_i^T \sum_{k=1}^{M}\lambda_k(\frac{du_k}{d\alpha}u_k^T + u_k\frac{du_k}{d\alpha}^T))$$

$$= Tr(\sum_{i=1}^{M}\frac{du_k}{d\alpha}u_k^T + u_k\frac{du_k}{d\alpha}^T) = Tr(\frac{d}{d\alpha}\sum_{i}^{M}u_i u_i^T)$$

Since we know $\sum_{i}^{M}u_i u_i^T = I$

Then $Tr(A^{-1}\frac{d}{d\alpha}A) = \sum_{i=1}^{M}\frac{1}{\lambda_i}\frac{d\lambda_i}{d\alpha}$

Finally $\frac{d}{d\alpha}\ln p(t|\alpha B) = \frac{M}{2}\frac{1}{\alpha} - \frac{1}{2}Tr(A^{-1}\frac{d}{d\alpha}A) - \frac{1}{2}m_N^T m_N$ (factor out $\frac{1}{2}$)

$$= \frac{1}{2}(\frac{M}{2} - Tr(A^{-1}\frac{d}{d\alpha}A) - M_N^T M_N)$$

$$= \frac{1}{2}(\frac{M}{2} - Tr(A^{-1}) - M_N^T M_N)$$

Shirui Ye

CS542

PS2

2. programming (Report & calculation)

(a) Linear Regression

According to the question, FTP and WE are two key variables, we use the linear function to calculate the third variable.

Thus, we use $Y = B_0 + B_1 X_1 + \cdots B_k X_k + \varepsilon$ which is $y = BX + \varepsilon$

Then we write out the matrix format:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad B = \begin{bmatrix} B_0 \\ B_1 \\ \vdots \\ B_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1n} \\ 1 & \vdots & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \cdots & x_{nn} \end{bmatrix} \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

And $\hat{y} = X\hat{B}$ where $\hat{B} = X'y(X'X^{-1})$

We know the cost function is $\frac{1}{2m}\sum_{i=1}^{m}(y - y_i)^2$, now we need find all case for all possibles.

After the procedures, we can determine LIC is the third variable.

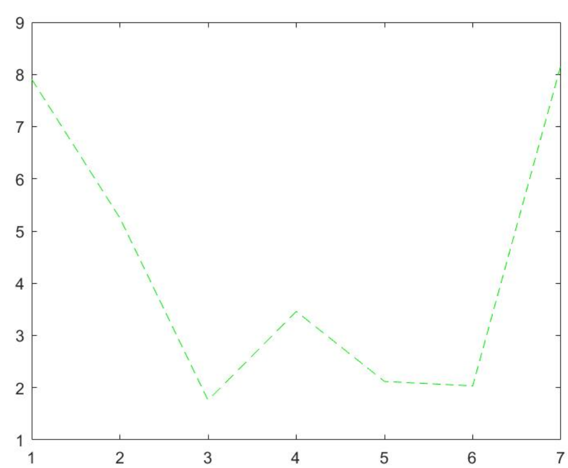
The final formula we will use is below:

$$y = 0.017\,LIC + 0.185\,FTP + 0.107\,WE - 58.124$$

As for the code, please see the next pages.

2(a) source code and graph

```
%load data according to the question
 d = load( 'detroit.mat' );
 %process data
 s = d.data(:, 9:10);
 %definition
 v = [1;1;1;1;1;1;1;1;1;1;1;1;1];
 e = [] ;
 %variables and factors
 HOM = d.data(:,10);
 LIC = d.data(:,4);
 FTP = d.data(:,1);
 WE = d.data(:,9);
 matrix = [v, FTP, WE];
 %procedures
 i = 2
 while(i < 9)
     store = d.data(:,i);
     new = [matrix, store];
     %formula
     b = (((new')*new)^(-1))*(new')*HOM;
     y = new * b ;
     sub = y - HOM;
     sub2 = sub.^2;
     e1 = sum(sub2);
     l_e = e1/(2*13);
     e = [e; l_e];
     i = i + 1 ;
 end
result = e
plot(result,'--','color',[0 0.9 0]);
```



2(b) Since I can’t figure out how to use Matlab to process lenses and CA data, I use Python to work on this question. To process the data, I use Panda Package.

i. I replaced all unknow first features. Then I calculate median of possible values of the missing features that are not numbers. I also replace the attribute by the mode of all attributes. Finally, I use label conditioned mean for real-valued with plus label. The formula is

$$\frac{sum\ of\ plus\ label\ data\ set's\ all\ feature}{no.\ of\ plus\ label\ data\ set}$$

As for minus data set, the formula is $\frac{sum\ of\ minus\ label\ data\ set's\ all\ feature}{no.\ of\ minus\ label\ data\ set}$

The detailed procedures are below and in process.py.

```
#Use Panda to process data
from sys import argv
import pandas as pd
import numpy as np
#command run
script, a, b = argv
#function below
def process(data):
    data = data.replace('?', np.NaN)
    r = [0,3,4,5,6]
  #missing features
    column = [1,2,7,10,13,14]
    for i in r:
    #replace by mode
            data[i] = data[i].fillna(data[i].mode()[0])
    r = [1,13]
  #plus, minus label
    lab = ['+', '-']
    for m in r:
            data[m] = data[m].apply(float)
            for n in lab:
      #get real-value missing ones
                    data.loc[ (data[m].isnull()) & ( data[15]==n ), m ] =
data[m][data[15] == n].mean()
    for c in column:
            data[c] = (data[c] - data[c].mean())/data[c].std()
    return data

#Use panda to process TrD which stands for training Data and TeD which
stands for testing data
TrD = pd.read_csv(a, header=None)
#process the data
TrD = process(TrD)
TrD.to_csv('crx.training.processed', header=False, index=False)
```

```
#Same Usage as above
TeD= pd.read_csv(b, header=None)
TeD = process(TeD)
TeD.to_csv('crx.testing.processed', header=False, index=False)
```

ii. KNN theory and its knowledge are cited from
*https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
*https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/
In this question, I am required to write a k-NN algorithm with L2 distance which is $D_{L2}(a,b)=\sqrt{\sum_i (a_i - b_i)^2}$
To run the program under command lines, I still use sys import argv in Python. The command line in this case should have 3 parameters. There are 2 scripts needed to run with the Python to make the command work. The code is below:

```
import math
from sys import argv
import pandas as pd
#command run, parameters
script, KNN, Tr, Te = argv
#Use panda to read
TrD = pd.read_csv(Tr, header=None)
TeD = pd.read_csv(Te, header=None)
LAB = []
R1, C1 = TrD.shape
R2, C2 = TeD.shape
nei = []

#find real label to its testing data
def label(list1, TrD):
  for i in range(len(list1)):
    label = TrD.iloc[list1[i]][ C1-1 ]
    LAB.append(label)
    #need the label with most times
  return max(set(LAB), key=LAB.count)

#distance is calculated below
def dist(m,n):
  res = 0
```

```
    list = []
    for i in range(R1):
        for j in range(C1 - 1 ):
            if(isinstance(m.iloc[i][j], str ) == True):
                if(m.iloc[i][j] != n[j]):
                    res = res + 1
                else:
                    res = res
            else:
                #DL2(m,n) = sqrt(res(m , n)^2))
                diff = math.pow((m.iloc[i][j] - n[j]), 2)
                res = res + diff
        list.append(math.sqrt(res))
        res = 0
    return list

TeD[C2] = TeD[C2-1]
a = 0
#run through the data
for index,row in TeD.iterrows():
    dis = dist(TrD,row)
    ordered = sorted(range(len(dis)), key=lambda k: dis[k])
    for i in range(int(KNN)):
        nei.append(ordered[i])
    res = label(nei, TrD)
    TeD.loc[a, C2] = res
    a = a + 1

#use panda to output
TeD.to_csv(Te, header=False, index=False)
```

iii. The accuracy result is below:

k	Lenses Acc	Crx Acc
1	7/7	131/132
3	7/7	131/132

iv. Citation:

Theory and knowledge of KNN are referenced from
*https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
*https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/