Final Exam, CPSC 8420, Fall 2023

Last Name, First Name

Due 12/16/2023, Saturday, 5:59PM EST

1 Problem 1 [15 pts]

Consider the following problem:

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda[\alpha\|\beta\|_2^2 + (1-\alpha)\|\beta\|_1]. \tag{1}$$

1. Show the objective can be reformulated into a lasso problem, with revised $\hat{\mathbf{X}}, \hat{\mathbf{y}}$.

   Assume we can find

$$\begin{aligned} \|\hat{\mathbf{y}} - \hat{\mathbf{X}}\beta\|^2 &= \|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda\alpha\|\beta\|_2^2 \\ \implies \|\hat{\mathbf{y}}\|^2 - 2\hat{\mathbf{y}}^T\hat{\mathbf{X}}\beta + \|\hat{\mathbf{X}}\beta\|^2 &= \|\mathbf{y}\|^2 - 2vy^T\mathbf{X}\beta \; + \underbrace{\|\mathbf{X}\beta\|^2 + \lambda\alpha\|\beta\|_2^2}_{\text{combine these 2 together as } \hat{\mathbf{X}}} \end{aligned}$$

$$\implies \hat{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \sqrt{\lambda\alpha}\mathbf{I} \end{bmatrix}$$

$$\hat{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix}$$

2. If $\alpha = 1/2, \lambda = 1$, please derive the closed-form solution by making use of alternating minimization that each time we fix the rest by optimizing one single element in β. You need randomly generate $\mathbf{X}, \mathbf{y}$ and initialize β_0, and show the objective decreases monotonically with updates.

(a)

$$\min_{\beta} \frac{1}{2}\|\hat{\mathbf{y}} - \hat{\mathbf{X}}\beta\|^2 + \lambda(1-\alpha)\|\beta\|_1$$

when we try to optimize β_i

$$\implies \min_{\beta_i} \frac{1}{2}\|\hat{\mathbf{y}} - \sum_{j\neq i} \hat{\mathbf{x}}_j\beta_j - \hat{\mathbf{x}}_i\beta_i\|^2 + \lambda(1-\alpha)|\beta_i|$$

$$\text{Set } \Delta_i = \hat{\mathbf{y}} - \sum_{j\neq i} \hat{\mathbf{x}}_j\beta_j \implies \min_{\beta_i} \frac{1}{2}\|\hat{\mathbf{x}}_i\beta_i - \Delta_i\|^2 + \lambda(1-\alpha)|\beta_i|$$

$$\implies \beta_i = \begin{cases} \dfrac{\langle \hat{\mathbf{x}}_i, \Delta_i\rangle - \lambda(1-\alpha)}{\|\hat{\mathbf{x}}_i\|^2}, \text{ if } \langle \hat{\mathbf{x}}_i, \Delta_i\rangle > \lambda(1-\alpha) \\ \dfrac{\langle \hat{\mathbf{x}}_i, \Delta_i\rangle + \lambda(1-\alpha)}{\|\hat{\mathbf{x}}_i\|^2}, \text{ if } \langle \hat{\mathbf{x}}_i, \Delta_i\rangle < -\lambda(1-\alpha) \\ 0, \text{ otherwise} \end{cases}$$

(b) See Figure 1

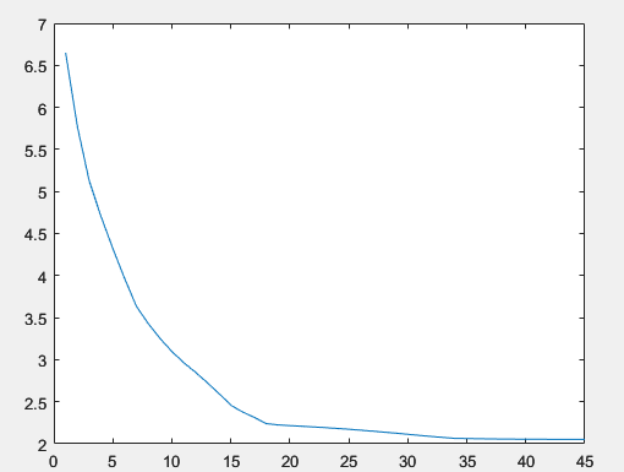

Figure 1: Q1.2

(c) Code 7.1

2 Problem 2 [10 pts]

1. For PCA, the loading vectors can be directly computed from the q columns of $\mathbf{U}$ where $[\mathbf{U}, \mathbf{S}, \mathbf{U}] = svd(\mathbf{X}^T\mathbf{X})$, please show that any $[\pm\mathbf{u}_1, \pm\mathbf{u}_2, \ldots, \pm\mathbf{u}_q]$ will be equivalent to $[\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_q]$ in terms of the same variance while satisfying the orthonormality constraint. This demonstrates that if the function is nonconvex, it may have various optimal solutions, which is different from (non-trivial) convex function.

   (a) For orthonormality

   $$\text{When } i \neq j, \langle \pm\mathbf{u}_i, \pm\mathbf{u}_j \rangle = \pm\langle \mathbf{u}_i, vu_j \rangle = 0$$

   (b) For variance

   $$\begin{aligned} \|\mathbf{X}\mathbf{u}_i\|^2 &= \mathbf{u}_i^T\mathbf{X}^T\mathbf{X}\mathbf{u}_i, \text{ s.t. } \|\mathbf{u}_i\|^2 = 1 \\ &= trace(\mathbf{u}_i^T\mathbf{X}^T\mathbf{X}\mathbf{u}_i) = trace((-\mathbf{u}_i)^T\mathbf{X}^T\mathbf{X}(-\mathbf{u}_i)) \\ &= \|\mathbf{X}(-\mathbf{u}_i)\|^2 \end{aligned}$$

2. Use the fact that $vec(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A})vec(\mathbf{X})$ to find the best solution to $\min_{\mathbf{X}} \|\mathbf{AXB} - \mathbf{Y}\|_F^2$, where $\mathbf{A} \in \mathbb{R}^{m\times p}, \mathbf{X} \in \mathbb{R}^{p\times q}, \mathbf{B} \in \mathbb{R}^{q\times n}, \mathbf{Y} \in \mathbb{R}^{m\times n}$.

   $$\begin{aligned} \min_{\mathbf{X}} \|\mathbf{AXB} - \mathbf{Y}\|_F^2 &= \min_{\mathbf{X}} \|vec(\mathbf{AXB}) - vec(\mathbf{Y})\|_2^2 \\ &= \min_{\mathbf{X}} \|(\mathbf{B}^T \otimes \mathbf{A})vec(\mathbf{X}) - vec(\mathbf{Y})\|_2^2 \end{aligned}$$

   According to the regression model: $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2, \mathbf{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$

   $$\implies vec(\mathbf{X}^*) = ((\mathbf{B}^T \otimes \mathbf{A})^T(\mathbf{B}^T \otimes \mathbf{A}))^{-1}(\mathbf{B}^T \otimes \mathbf{A})^T vec(\mathbf{Y}),$$

3 Problem 3 [30 pts]

Please find *USArrests* dataset online and

1. Implement your own program to reproduce the image on page 16/26 of Dimensionality Reduction slides on Canvas.

   - Fig see figure 2

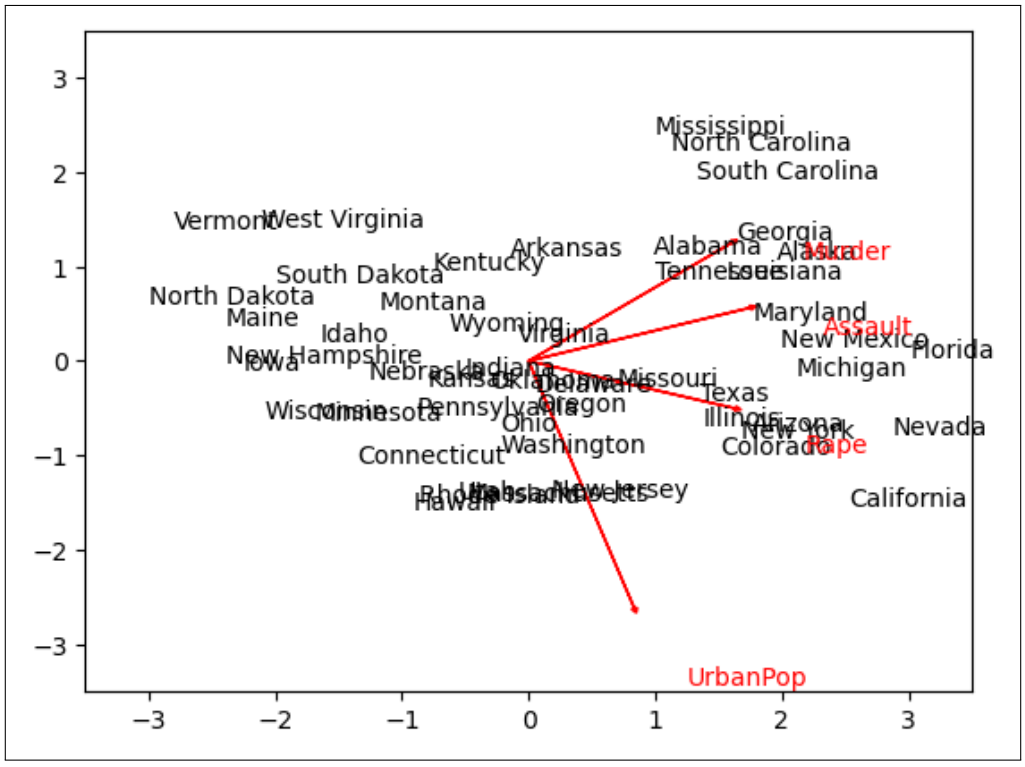

Figure 2: USArrtests

   - Code See Code 7.2

2. For each state, out of 4 features, please randomly mask one and assume it is missing (therefore you have your own Ω and X), please write a program following what we discussed in class (you may refer to ProximalGradientDescent.pdf on Canvas) to optimize

$$\min_{Z} \frac{1}{2}\|P_{\Omega}(X - Z)\|_F^2 + \|Z\|_* \tag{2}$$

4 Problem 4 [15 pts]

Please refer to here (for Python) or here (for Matlab) to create a *two (half) moon* dataset. Write your own *spectral clustering* codes to separate the data into two groups with different colors. You are not a allowed to call the built-in function for Python or Matlab.

1. Result
   See Figure 3

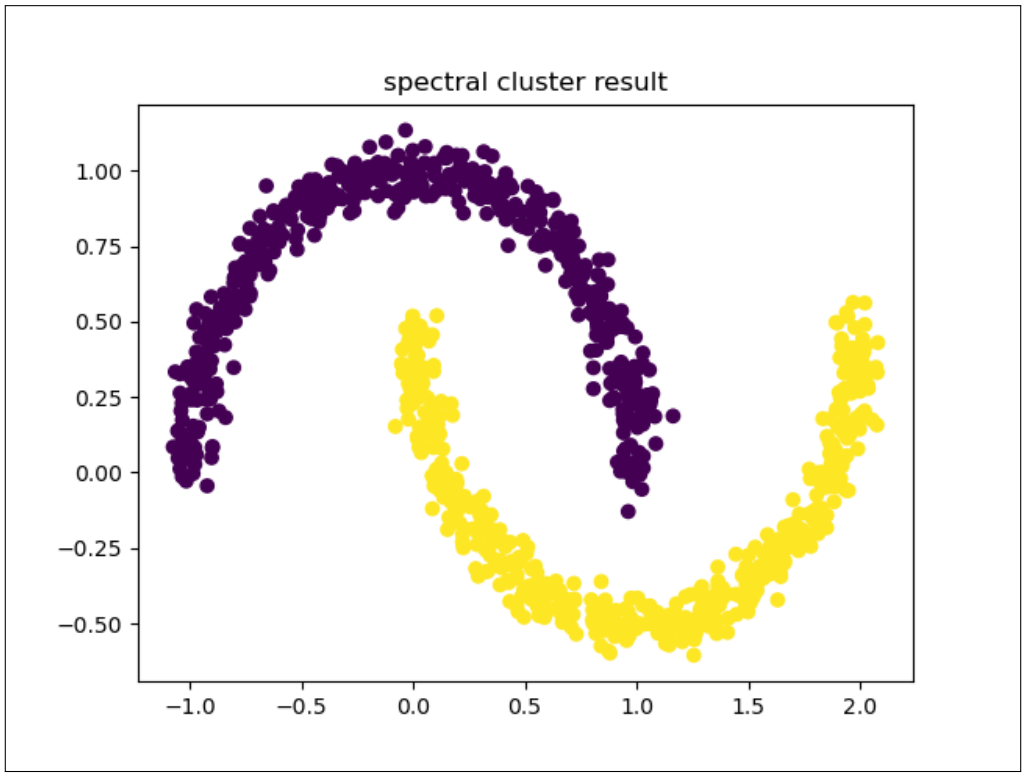

Figure 3: Q4 Sepctral Clustering

2. Code
   See Code 7.3

5 Problem 5 [35 pts]

For Logistic Regression, if the label is ± 1, the objective is:

$$\min_{\mathbf{w}} \sum_{i=1}^{m} log(1 + exp(-y_i \mathbf{w}^T \mathbf{x}_i)) \tag{3}$$

while if the label is $\{1, 0\}$ the objective is:

$$\min_{\mathbf{w}} \sum_{i=1}^{m} log(1 + exp(\mathbf{w}^T \mathbf{x}_i)) - y_i \mathbf{w}^T \mathbf{x}_i \tag{4}$$

1. Write a program to show that the optimal solutions to the two cases are the same by making use of gradient descent method where $m = 100$ (please carefully choose the stepsize as we discussed in class). You can generate two class samples, one class's label is 1 and the other is -1 or 0 corresponding to the two formulations respectively. You can initialize $\mathbf{w}$ as $\mathbf{0}$.

   (a) Convert to Matrix form
   Seperatly convert objective equations 4 and 3 to Matrix Format
   For Lable $\{1, 0\}$

   $$\mathbf{L}_{\{1,0\}} = -(\mathbf{y}^T \mathbf{X}\mathbf{w} - \mathbf{I}^T log(\mathbf{I} + exp(\mathbf{X}\mathbf{w}))) \tag{5}$$

   For Lable $\{1, -1\}$

   $$\mathbf{L}_{\{1,-1\}} = \mathbf{I}^T log(\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w})) \tag{6}$$

   where, $\mathbf{I} \in \mathbb{R}^{m \times 1}$ and all elements are 1s and

   $$\mathbf{Y} = \underbrace{[\mathbf{y}, \cdots, \mathbf{y}]}_{\#features+1}, \ \mathbf{X} = \left[\begin{bmatrix} 1 \\ \mathbf{x}_1 \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{x}_2 \end{bmatrix} \cdots \begin{bmatrix} 1 \\ \mathbf{x}_m \end{bmatrix} \right]^T$$

   (b) Gradient
   Firstly, we express the sigmoid function as

   $$h_{\mathbf{w}}(\mathbf{Z}) = \frac{1}{1 + exp(-\mathbf{Z}\mathbf{w})} \tag{7}$$

   For Lable $\{1, 0\}$, the derivitive can be easily get from slides

   $$\frac{\partial \mathbf{L}_{\{1,0\}}}{\partial \mathbf{w}} = \mathbf{X}^T (h_{\mathbf{w}}(\mathbf{X}) - \mathbf{Y}) \tag{8}$$

For Lable $\{1, -1\}$

$$\begin{aligned}
\mathrm{d}\mathbf{L}(\mathbf{w}) &= \mathrm{d}\mathbf{I}^T log(\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w})) + \mathbf{I}^T \, \mathrm{d}log(\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w})) \\
&= \mathbf{I}^T \, \mathrm{d}log(\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w})) \\
&= \mathbf{I}^T \left(\frac{1}{\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w})} \odot \mathrm{d}\left(\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w})\right) \right) \\
&= \left(\mathbf{I} \odot \frac{1}{\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})vw)} \right)^T \mathrm{d}(\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})vw)) \\
&= \left(\mathbf{I} \odot \frac{1}{\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})vw)} \right)^T (exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w}) \odot \mathrm{d}(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w})) \\
&= \left(\mathbf{I} \odot \frac{1}{\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})vw)} \odot exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w}) \right)^T \mathrm{d}(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w}) \\
&= -\left(\mathbf{I} \odot \frac{1}{\mathbf{I} + exp(-(\mathbf{Y} \odot \mathbf{X})vw)} \odot exp(-(\mathbf{Y} \odot \mathbf{X})\mathbf{w}) \right)^T (\mathbf{Y} \odot \mathbf{X}) \, \mathrm{d}\mathbf{w} \\
&= -\left(\mathbf{I} - h_{\mathbf{w}} \left(\mathbf{Y} \odot \mathbf{X}\right)\right)^T (\mathbf{Y} \odot \mathbf{X}) \, \mathrm{d}\mathbf{w}
\end{aligned}$$

from above we get, for Lable $\{1, -1\}$

$$\frac{\partial \mathbf{L}_{\{1,-1\}}}{\partial \mathbf{w}} = -(\mathbf{Y} \odot \mathbf{X})^T \left(\mathbf{I} - h_{\mathbf{w}} \left(\mathbf{Y} \odot \mathbf{X}\right)\right) \tag{9}$$

(c) StepSize
The Hessian matrix will change with each iteration, unlike Least Squares which remains same. for convenience, I picked a fix value $\alpha = 0.0001$ as the stepsize

(d) Figure See Figue(4)
from Figue(4), We can tell that values of 2 objective functions are the same with each iteration, which mean the $\mathbf{w}$s are also same with each other.
After $50k$ iterations, $\mathbf{w}$ values are

$$\mathbf{w}_{\{0,1\}} = \mathbf{w}_{\{-1,1\}} = \begin{bmatrix} 0.19903285 \\ 0.38847427 \\ -0.34194269 \\ -0.72259845 \end{bmatrix}$$

(e) Code
See Code(7.4)

2. Consider the case where class label is $\{1, 0\}$ and $P(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1+exp(-\mathbf{w}^T\mathbf{x})}$, the maximum likelihood function is $p^y(1-p)^{1-y}$, which is equivalent to $\min -ylog(p) - (1-y)log(1-p)$, exactly the binary cross entropy. Please find optimal p.

$$\begin{aligned}
\frac{\partial \left(-ylog(p) - (1-y)log(1-p)\right)}{\partial p} &= 0 \\
\implies -\frac{y}{p} + \frac{1-y}{1-p} &= 0 \\
\implies p &= y
\end{aligned}$$

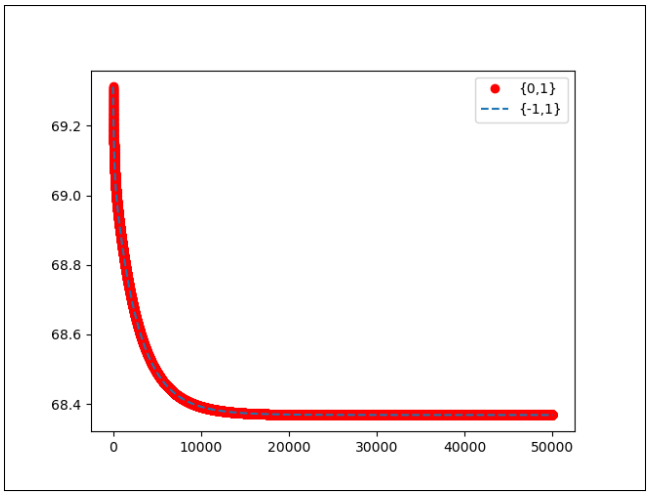

Figure 4: Q5-1 {0,1} vs {-1,1}

3. If we use Mean Square Error instead of cross entropy: min $(y-p)^2$, and assume $y=1$ and our initial weight $\mathbf{w}$ result in p very close to 0, if we optimize $\mathbf{w}$ by making use of gradient descent method, what will happen? Convince yourself that it will stuck at initial point and explain briefly why.

   Since

$$p = \frac{1}{1+exp(-\mathbf{w}^T\mathbf{x})}$$

   If we use MSE, then the Gradient will be

$$\begin{aligned}
\frac{\partial \mathbf{L}}{\mathbf{w}} &= -2\,(y-p)\,\frac{\partial p}{\partial exp(-\mathbf{w}^T\mathbf{x})}\frac{\partial exp(-\mathbf{w}^T\mathbf{x})}{\partial \mathbf{w}} \\
&= 2\,(y-p)\,\frac{1}{(1+exp(-\mathbf{w}^T\mathbf{x}))^2}\frac{\partial exp(-\mathbf{w}^T\mathbf{x})}{\partial \mathbf{w}} \\
&= 2\,(y-p)\,\frac{exp(-\mathbf{w}^T\mathbf{x})}{(1+exp(-\mathbf{w}^T\mathbf{x}))^2}\frac{-\mathbf{w}^T\mathbf{x}}{\partial \mathbf{w}} \\
&= 2\,(y-p)\times p\times(1-p)\times\frac{\partial -\mathbf{w}^T\mathbf{x}}{\partial \mathbf{w}}
\end{aligned}$$

   which tells us either $p\to 0$ or $p\to 1$, the gradient $\frac{\partial \mathbf{L}}{\mathbf{w}}$ will become 0, in our case, initial $\mathbf{w}$ makes $p\to 0$ then the gradient for the first iteration is 0m it will be stuck at the initial point

4. For the second objective where the label is $\{1,0\}$, implement Newton method (with back-tracking line search if necessary) where $m=100$. Compare with gradient descent method and plot objective versus time consumption in one figure to observe which is faster.

$$\mathbf{H} = \mathbf{X}^T\mathbf{W}\mathbf{X} \tag{10}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha\mathbf{H}^{-1}\frac{\partial \mathbf{L}_{\{1,0\}}}{\partial \mathbf{w}} \tag{11}$$

where α is the stepsize, and

$$\mathbf{W} = \begin{bmatrix} h_{\mathbf{w}}(\mathbf{x}_1)(1-h_{\mathbf{w}}(\mathbf{x}_1)) & & \\ & \ddots & \\ & & h_{\mathbf{w}}(\mathbf{x}_m)(1-h_{\mathbf{w}}(\mathbf{x}_m)) \end{bmatrix} = Diag\left(h_{\mathbf{w}}(\mathbf{X}) \odot (\mathbf{I} - h_{\mathbf{w}}(\mathbf{X}))\right)$$

(a) Figure See Figure (5)

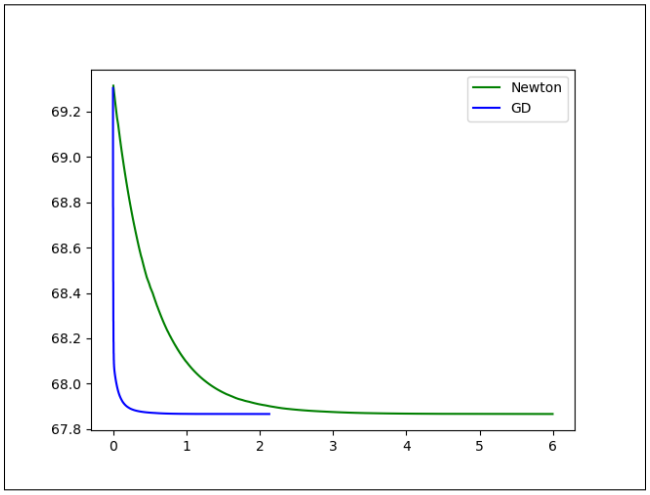

Figure 5: Q5-4 Newton vs GD

From Figure (5) , we can tell GD is faster

(b) Code See Code(**??**)

5. From now on, let's focus on the first objective where the label is ± 1. Please write a program to find the optimal $\mathbf{w}$ by using gradient descent method where $m = 10K$, the stepsize in this case we set it as $\frac{1}{\|\mathbf{X}\|_F^2}$ where each column of $\mathbf{X}$ is $\mathbf{x}_i$.

   (a) Figure
   See Fig(6)

   (b) Code
   See Code7.5

6. Please write a stochastic gradient descent version for $m = 10K$ (you may set the stepsize as $2/(t+1)$ where $t = 1, \ldots, T$ and $T = 100K$) with the final output being $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \frac{2t}{T+1}\mathbf{w}_t$.

   (a) To make it timing friendly, I set the stepsize= 0.01 and the number of iteration is 50

   (b) Code See Code 7.6

   (c) Result

$$\mathbf{w} = \begin{bmatrix} -1.7870602 \\ 0.52960937 \\ 0.59429952 \\ -0.21795669 \end{bmatrix}$$

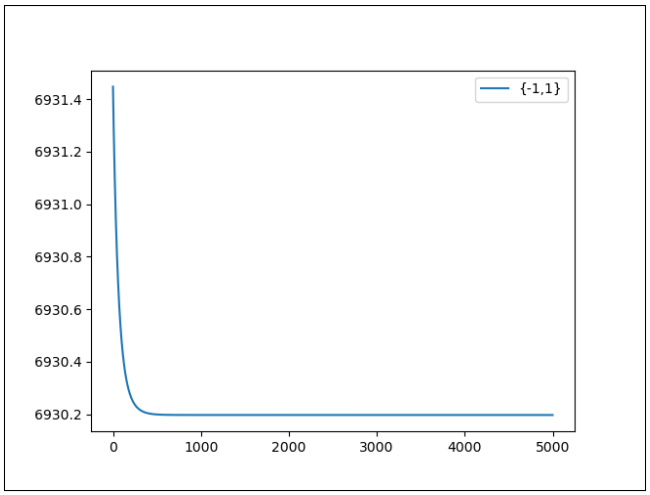

Figure 6: Q5-5 stepsize is frobenius norm

7. Please compare those two methods (gradient descent vs. stochastic gradient descent) for $m = 10K$ and $m = 100$ by plotting objective changes versus time consumption respectively.

   To make it converge faster, instead of using only one sample to update $\mathbf{w}$, I randomly choose *batchsize* samples to update $\mathbf{w}$.

   (a) $m = 100$
   See Figure(7)

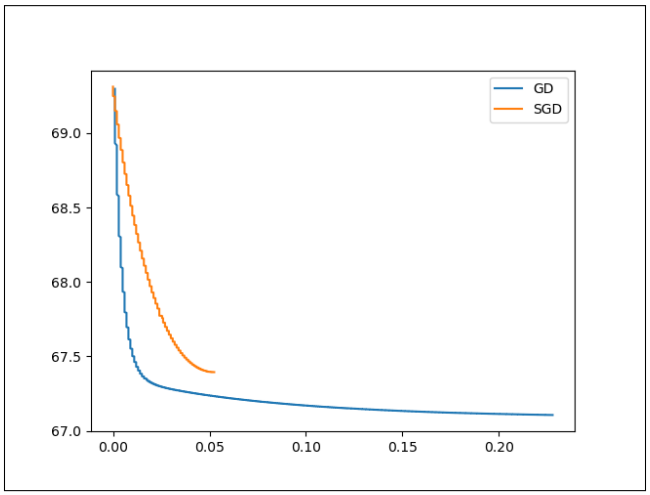

Figure 7: Q5-7 , m=100

   (b) $m = 10K$
   See Figure(8)

   (c) Code
   See Code7.7

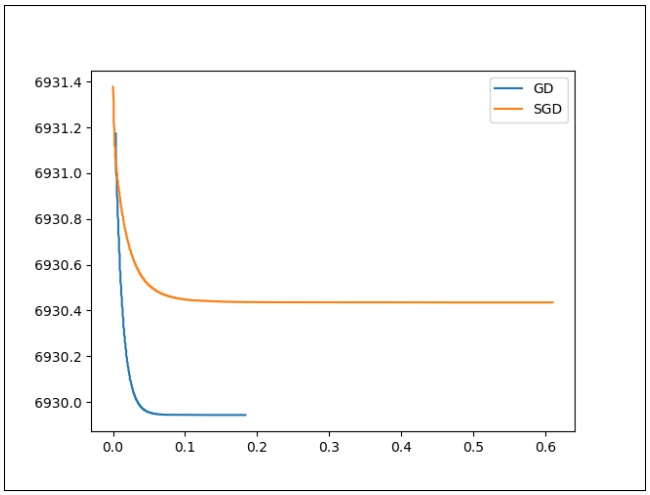

Figure 8: Q5-7, m=10k

6 Problem 6 [15 pts]

We consider multiclass SVM based on binary SVM. There are two options we can consider: one versus one and one versus all. Assume we have 4 classes data where each class has 2 samples: class 1 $\{\{1,0\},\{2,0\}\}$, class 2 $\{\{0,-1\},\{0,-2\}\}$, class 3 $\{\{-1,0\},\{-2,0\}\}$ and class 4 $\{\{0,1\},\{0,2\}\}$. Now use the two options (one versus one and one versus all) respectively to determine the predicted class of new data $\{0.25,1.5\}$. You should explicitly find and write each hyperplane to get full credits.

6.1 One vs. One

1. 1 vs 2
$$y = -x$$
2. 1 vs 3
$$y = 0$$
3. 1 vs 4
$$y = x$$
4. 2 vs 3
$$y = x$$
5. 2 vs 4
$$x = 0$$
6. 3 vs 4
$$y = -x$$

6.2 one vs. all

assume hyperplane is $w_0 x + w_1 y + b = 0$

1. 1 vs $\hat{1}$

$$\begin{cases} \min \frac{1}{2}\|\mathbf{w}\|^2 \\ s.t. \\ w_0 + b \geq 1 \\ 2w_0 + b \geq 1 \\ w_0 - b \geq 1 \\ 2w_0 - b \geq 1 \\ w_1 - b \geq 1 \\ 2w_1 - b \geq 1 \\ -w_1 - b \geq 1 \\ -2w_1 - b \geq 1 \end{cases} \tag{12}$$

$$\begin{aligned} \mathbf{L} = \frac{1}{2}(w_0^2 + w_1^2) + \lambda_1(1 - w_0 - b) + \lambda_2(1 - 2w_0 - b) + \\ \lambda_3(1 - w_0 + b) + \lambda_4(1 - 2w_0 + b) + \\ \lambda_5(1 - w_1 + b) + \lambda_6(1 - 2w_1 + b) + \\ \lambda_7(1 + w_1 + b) + \lambda_8(1 + 2w_1 + b) \end{aligned}$$

using KKT we get

$$\{asdasd \tag{13}$$

2.

7 Codes

7.1 Code for section 1.2

```
n = 100;
m = 10;

alpha = 0.5;
lambda = 1;

X = rand(n,m);
y = rand(n,1);


theta = ones(1,m)* sqrt(alpha*lambda);
X_hat = [X;theta];
y_hat = [y;0];
```

```
wh = lassoAlg(X_hat, y_hat, alpha*lambda);


function xh = lassoAlg(A,y,lam)
    xnew = rand(size(A,2),1);   % "initial guess"
    xold = xnew+ones(size(xnew)); % used zeros so the while loop initiates
    loss = xnew - xold;
    thresh = 10e-3;     % threshold value for optimization

    iter = 0;
    objs = [];
    iters=[];
    while norm(loss) > thresh
        iter = iter+1;

        xold = xnew;    % need to store the previous iteration of xh
        for i = 1:length(xnew)
            a = A(:,i);     % get column of A
            p = (norm(a,2))^2;
            % from notes: -t = sum(aj*xj) - y for all j != i
            % i.e., sum(aj*xj) - ai*xi - y (my interpretation)
            % hence t = (above) * -1
            % want to be sure this the correct definition of t?
            t =  a*xnew(i) + y - A*xnew;
            q = a'*t;
            % update xi
            xnew(i) = (1/p) * sign(q) * max(abs(q)-lam, 0);
        end
        loss = xnew - xold;     % update loss

        obj = 0.5*norm(y-A*xnew,2)+lam*norm(xnew,1);
        objs=[objs,obj];
        iters = [iters,iter];
    end
    xh = xnew;
    plot(iters,objs,'DisplayName','Test')

end
```

7.2 Code for section 3.1

```
import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler
from numpy import linalg as LA
import matplotlib.pyplot as plt

dataset = pd.read_csv("USArrests.csv")
dataset.head(5)
```

```
states = dataset. iloc[:,0]

scaler = StandardScaler()
data = dataset[['Murder', "Assault", "UrbanPop","Rape"]]

scaled_data = scaler.fit_transform(data)

center = np.mean(scaled_data,axis=0)

n_samples = scaled_data.shape[0]
scaled_data = scaled_data - center
Vari = np.dot(scaled_data.T,scaled_data)/n_samples

eigenvalues, eigenvectors = LA.eig(Vari)

PC1 = eigenvectors[:,0]
PC2 = eigenvectors[:,1]

x_list = np.dot(scaled_data,PC1.T)
y_list = np.dot(scaled_data,PC2.T)

murder = [PC1[0],PC2[0]]
assault = np.array(PC1[1],PC2[1])
urbanpop = np.array(PC1[2],PC2[2])
rape = np.array(PC1[3],PC2[3])

features = ["Murder","Assault", "UrbanPop","Rape"]

plt.xlim(-3.5,3.5)
plt.ylim(-3.5,3.5)

for s in range(50):
    plt.text(x_list[s],y_list[s], states[s])

for i in range(4):
# (starting_x, starting_y, dx, dy, ...)
    plt.arrow(0,0, PC1[i]*3,PC2[i]*3, head_width=0.05, head_length=0.05, color='red')
    plt.annotate(features[i],
             xy=(PC1[i]*3.5, PC2[i]*3.5),
             xytext=(20, -20),
             textcoords='offset pixels', color='red')


plt.show()
```

7.3 Code for section 4

```
import shadow.utils
shadow.utils.set_seed(0, cudnn_deterministic=True)  # set seeds for reproducibility
#%matplotlib inline
import matplotlib.pyplot as plt
```

```
from sklearn import datasets
import numpy as np
import random
import math as m

n_samples = 1000  # number of samples to generate
noise = 0.05  # noise to add to sample locations
X, y = datasets.make_moons(n_samples=n_samples, noise=noise)


class my_kmeans:
    def __init__(self, clusers=2):
        self.k = clusers

    def cal_dis(self, data, centeroids):
        dis = []
        for i in range(len(data)):
            dis.append([])
            for j in range(self.k):
                dis[i].append(m.sqrt((data[i, 0] - centeroids[j, 0])**2 + (data[i, 1]-centeroids[j, 1])
        return np.asarray(dis)

    def divide(self, data, dis):
        clusterRes = [0] * len(data)
        for i in range(len(data)):
            seq = np.argsort(dis[i])
            clusterRes[i] = seq[0]

        return np.asarray(clusterRes)

    def centeroids(self, data, clusterRes):
        centeroids_new = []
        for i in range(self.k):
            idx = np.where(clusterRes == i)
            sum = data[idx].sum(axis=0)
            avg_sum = sum/len(data[idx])
            centeroids_new.append(avg_sum)
        centeroids_new = np.asarray(centeroids_new)
        return centeroids_new[:, 0: 2]

    def cluster(self, data, centeroids):
        clulist = self.cal_dis(data, centeroids)
        clusterRes = self.divide(data, clulist)
        centeroids_new = self.centeroids(data, clusterRes)
        err = centeroids_new - centeroids
        return err, centeroids_new, clusterRes

    def fit(self,data):
        clu = random.sample(data[:, 0:2].tolist(), 2)
        clu = np.asarray(clu)
        err, clunew,  clusterRes = self.cluster(data, clu)
```

```
        while np.any(abs(err) > 0):
            #print(clunew)
            err, clunew,  clusterRes = self.cluster(data, clunew)

        clulist = self.cal_dis(data, clunew)
        clusterResult = self.divide(data, clulist)

        return clusterResult


def myKNN(S, k, sigma=2.0):
    N = len(S)
    A = np.zeros((N,N))

    for i in range(N):
        dist_with_index = zip(S[i], range(N))
        dist_with_index = sorted(dist_with_index, key=lambda x:x[0])
        neighbours_id = [dist_with_index[m][1] for m in range(k+1)] # xi's k nearest neighbours

        for j in neighbours_id: # xj is xi's neighbour
            A[i][j] = np.exp(-S[i][j]/2/sigma/sigma)
            A[j][i] = A[i][j] # mutually

    return A

def calLaplacianMatrix(adjacentMatrix):

    # compute the Degree Matrix: D=sum(A)
    degreeMatrix = np.sum(adjacentMatrix, axis=1)

    # compute the Laplacian Matrix: L=D-A
    laplacianMatrix = np.diag(degreeMatrix) - adjacentMatrix

    # normailze
    # D^(-1/2) L D^(-1/2)
    sqrtDegreeMatrix = np.diag(1.0 / (degreeMatrix ** (0.5)))
    return np.dot(np.dot(sqrtDegreeMatrix, laplacianMatrix), sqrtDegreeMatrix)

def euclidDistance(x1, x2, sqrt_flag=False):
    res = np.sum((x1-x2)**2)
    if sqrt_flag:
        res = np.sqrt(res)
    return res

def Distance(X):
    X = np.array(X)
    S = np.zeros((len(X), len(X)))
    for i in range(len(X)):
        for j in range(i+1, len(X)):
            S[i][j] = 1.0 * euclidDistance(X[i], X[j])
            S[j][i] = S[i][j]
```

```
    return S

clusters = 2

Similarity = Distance(X)
Adjacent = myKNN(Similarity, k=5)
Laplacian = calLaplacianMatrix(Adjacent)
x, V = np.linalg.eig(Laplacian)
x = zip(x, range(len(x)))
x = sorted(x, key=lambda x:x[0])
H = np.vstack([V[:,i] for (v, i) in x[:clusters]]).T
result = my_kmeans(2).fit(H)
plt.title('spectral cluster result')
plt.scatter(X[:,0], X[:,1],marker='o',c=result)
plt.show()
```

7.4 Code for section 5.1

```
import numpy as np
import matplotlib.pyplot as plt

m = 100
n = 3

y_0 = np.random.choice([0, 1], size=(m,1) , p=[.5, .5])
y_1 = np.array([y*2-1 for y in y_0])

X = np.random.rand(m,n)
x0 = np.ones((m,1))
X = np.hstack((x0,X))

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

def L(w, X, y):
    I = np.ones((X.shape[0],1))
    Xw =np.dot(X,w)
    YtXw = np.dot(np.transpose(y),Xw)
    Delta = np.log(I+np.exp(Xw))
    return np.sum(-YtXw+np.dot(np.transpose(I),Delta),axis=0)


def dL(w, X, y):
    Xw=np.dot(X,w)
    distance = sigmoid(Xw)-y
    return np.dot(np.transpose(X),distance)



def GD(w, X, y, epoch, lr):
    l_list = []
```

```
    for i in range(epoch):
        dw= dL(w,X, y)
        w -= lr * dw
        l =  L(w, X, y)
        l_list = np.append(l_list,l)
    return w, l_list

def L1(w, X, y):
    I = np.ones((X.shape[0],1))
    n = X.shape[1]
    Y = np.tile(y,(1,n))
    YXW = np.dot(Y*X,w)
    Delta = np.log(I + np.exp(-YXW))
    return np.sum(np.dot(np.transpose(I),Delta),axis=0)

def dL1(w, X, y):
    I = np.ones((X.shape[0],1))
    n = X.shape[1]
    Y = np.tile(y,(1,n))
    YXW = np.dot(Y*X,w)
    Delta = I - sigmoid(YXW)
    return -np.dot(np.transpose(Y*X), Delta)


def GD1(w, X, y, epoch, lr):
    l_list = []
    for i in range(epoch):
        dw= dL1(w,X, y)
        w -= lr * dw
        l =  L1(w, X, y)
        l_list = np.append(l_list,l)
    return w, l_list

w0 = np.zeros([n+1,1])
w1 = np.zeros([n+1,1])

ITER = 50000
w0,L0_LIST = GD(w0,X,y_0,ITER,0.0001)
w1,L1_LIST= GD1(w1,X,y_1,ITER,0.0001)
iters = list(range(0,ITER))

print(w0)
print(w1)

plt.plot(iters,L0_LIST, 'or' ,label = "{0,1}")
plt.plot(iters,L1_LIST,'--'  ,label = "{-1,1}")
plt.legend()
plt.show()
```

7.5 Code for section 5.5

```
import numpy as np
import matplotlib.pyplot as plt

m = 10000
n = 3
ITER = 5000

y_0 = np.random.choice([0, 1], size=(m,1) , p=[.5, .5])
y_1 = np.array([y*2-1 for y in y_0])

X = np.random.rand(m,n)
x0 = np.ones((m,1))
X = np.hstack((x0,X))

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

def L1(w, X, y):
    I = np.ones((X.shape[0],1))
    n = X.shape[1]
    Y = np.tile(y,(1,n))
    YXW = np.dot(Y*X,w)
    Delta = np.log(I + np.exp(-YXW))
    return np.sum(np.dot(np.transpose(I),Delta),axis=0)

def dL1(w, X, y):
    I = np.ones((X.shape[0],1))
    n = X.shape[1]
    Y = np.tile(y,(1,n))
    YXW = np.dot(Y*X,w)
    Delta = I - sigmoid(YXW)
    return -np.dot(np.transpose(Y*X), Delta)


def GD1(w, X, y, epoch, lr):
    l_list = []
    for i in range(epoch):
        dw= dL1(w,X, y)
        w -= lr * dw
        l =  L1(w, X, y)
        l_list = np.append(l_list,l)
    return w, l_list

w1 = np.zeros([n+1,1])



lr =1/ np.linalg.norm(X,"fro")
lr = lr*lr
```

```
w1,L1_LIST= GD1(w1,X,y_1,ITER,lr)
iters = list(range(0,ITER))

print(w1)
plt.plot(iters,L1_LIST,label = "{-1,1}")
plt.legend()
plt.show()
```

7.6 Code for section 5.6

```
import numpy as np
import matplotlib.pyplot as plt

m = 10000
n = 3
ITER = 50
lr = 0.01

y_0 = np.random.choice([0, 1], size=(m,1) , p=[.5, .5])
y_1 = np.array([y*2-1 for y in y_0])
I = np.ones((m,1))
X = np.random.rand(m,n)
x0 = np.ones((m,1))
X = np.hstack((x0,X))

batchsize = 4000
T = 100000
t_list = range(T)

from time import time
import datetime
def sigmoid(x):
    return 1. / (1. + np.exp(-x))

def L(w, X, y):
    Xw =np.dot(X,w)
    YtXw = np.dot(np.transpose(y),Xw)
    Delta = np.log(I+np.exp(Xw))
    return np.sum(-YtXw+np.dot(np.transpose(I),Delta),axis=0)


def dL(w, X, y):
    Xw=np.dot(X,w)
    distance = sigmoid(Xw)-y
    return np.dot(np.transpose(X),distance)


def dL_SGD(w, X, y,index_list):
    X_sub = X[index_list,:]
    y_sub = y[index_list,:]
    return dL(w, X_sub, y_sub)
```

```
def SGD(w, X, y, epoch, lr, batchsize):
    l_list = []
    idx_list = np.random.choice(range(X.shape[0]), batchsize,replace = False)
    for i in range(epoch):
        dw= dL_SGD(w,X, y,idx_list)
        w -= lr * dw
        l =  L(w, X, y)
        l_list = np.append(l_list,l)
    return w, l_list

w_hat = np.zeros([n+1,1])

for t in t_list:
    time0 = time()
    w0 = np.zeros([n+1,1])
    weight = (1/T)*(t+1)/(T+1)
    w0,L0_LIST = SGD(w0,X,y_0,ITER,lr,batchsize)
    w_hat +=  w0 *weight
    print(t ,time()-time0)
print(w_hat)
```

7.7 Code for section 5.7

```
import numpy as np
import matplotlib.pyplot as plt


m_list= [100,10000]
n=3

ITER = 50000
lr = 0.0001



from time import time
import datetime
def sigmoid(x):
    return 1. / (1. + np.exp(-x))

def L(w, X, y):
    I = np.ones((X.shape[0],1))
    Xw =np.dot(X,w)
    YtXw = np.dot(np.transpose(y),Xw)
    Delta = np.log(I+np.exp(Xw))
    return np.sum(-YtXw+np.dot(np.transpose(I),Delta),axis=0)
```

```
def dL(w, X, y):
    Xw=np.dot(X,w)
    distance = sigmoid(Xw)-y
    return np.dot(np.transpose(X),distance)

def GD(w, X, y, thresdhold, lr):
    l_list = []
    times = []
    time0 = time()

    l = 0
    err = np.inf
    while err > thresdhold:
    #for i in range(epoch):
        dw= dL(w,X, y)
        w -= lr * dw
        l_new =  L(w, X, y)
        err = np.linalg.norm(l_new-l,1)/X.shape[0]
        times = np.append(times,time()-time0)
        l_list = np.append(l_list,l_new)
        l = l_new
    return w, l_list, times

def dL_SGD(w, X, y,index_list):
    X_sub = X[index_list,:]
    y_sub = y[index_list,:]

    return dL(w, X_sub, y_sub)


def SGD(w, X, y, thresdhold, lr, batchsize):
    w= w.copy()
    l_list = []
    idx_list = np.random.choice(range(X.shape[0]), batchsize,replace = False)
    times = []
    time0 = time()

    l = 0
    err = np.inf

    while err > thresdhold:
    #for i in range(epoch):
        dw= dL_SGD(w,X, y,idx_list)
        w -= lr * dw
        l_new =  L(w, X, y)
        err = np.linalg.norm(l_new-l,1)/X.shape[0]
        l_list = np.append(l_list,l_new)
        l = l_new
        times = np.append(times,time()-time0)
    return w, l_list, times
```

```
for m in m_list:
    thresdhold = 0.0001/(m**2)
    y_0 = np.random.choice([0, 1], size=(m,1) , p=[.5, .5])

    I = np.ones((m,1))
    X = np.random.rand(m,n)
    x0 = np.ones((m,1))
    X = np.hstack((x0,X))

    w_gd = np.zeros([n+1,1])
    w_sgd = np.zeros([n+1,1])
    w_gd,L_GD,time_gd = GD(w_gd,X,y_0,thresdhold,lr)

    w_sgd,L_SGD,time_sgd= SGD(w_sgd,X,y_0,thresdhold,lr,m//2)

    iters = range(ITER)
        # plt.plot(iters,L_GD, 'or' ,label = "GD")
        # plt.plot(iters,L_SGD,'--'  ,label = "SGD")
    plt.plot(time_gd,L_GD ,label = "GD")
    plt.plot(time_sgd,L_SGD ,label = "SGD")
    plt.legend()

    plt.show()

print(L_GD[-1])
```