Final Exam, CPSC 8420, Spring 2022

Huang, Gangtong

Due 05/06/2022, Friday, 11:59PM EST

Problem 1 [15 pts]

Consider the elastic-net optimization 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 a slightly different $\mathbf{X}, \mathbf{y}$.

   Define: $\mathbf{X}_1 = (1+\lambda\alpha)\begin{pmatrix}\mathbf{X}\\ \sqrt{\lambda\alpha}\mathbf{I}_p\end{pmatrix}$, $\mathbf{y}_1 = \begin{pmatrix}\mathbf{y}\\ 0\end{pmatrix}$, and $\beta_1 = \sqrt{1+\lambda\alpha}\beta$ then:

$$\begin{aligned}\mathbf{X}_1^T\mathbf{X}_1 &= (1+\lambda\alpha)(\mathbf{X}^T\mathbf{X} + \lambda\alpha\mathbf{I})\\ &= (1+\lambda\alpha)(\mathbf{X}^T\mathbf{X} + \lambda\alpha)\end{aligned} \tag{2}$$

   , and

$$\begin{aligned}\mathbf{y}_1^T\mathbf{y}_1 &= \mathbf{y}^T\mathbf{y} + 0\\ &= \mathbf{y}^T\mathbf{y}\end{aligned} \tag{3}$$

   Define $\mathbf{L}(\mathbf{y}_1, \mathbf{X}_1, \beta_1) = \|\mathbf{y}_1 - \mathbf{X}_1\beta_1\|^2 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1$. Formulata a Lasso problem using $\mathbf{X}_1$, $\mathbf{y}_1$, β_1:

$$\min_{\beta_1} \mathbf{L}(\mathbf{y}_1, \mathbf{X}_1, \beta_1) \tag{4}$$

   , where:

$$\begin{aligned}\mathbf{L}(\mathbf{y}_1, \mathbf{X}_1, \beta_1) &= \|\mathbf{y}_1 - \mathbf{X}_1\beta_1\|^2 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1\\ &= (\mathbf{y}_1 - \mathbf{X}_1\beta_1)^T(\mathbf{y}_1 - \mathbf{X}_1\beta_1) + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1\\ &= \mathbf{y}_1^T\mathbf{y}_1 - \beta_1^T\mathbf{X}^Ty_1 - y_1^T\mathbf{X}_1\beta + \beta_1^T\mathbf{X}_1^T\mathbf{X}_1\beta_1 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1\end{aligned} \tag{5}$$

   Plug eqns. (2), (3) into eqn. (4), we will arrive at:

$$\mathbf{L}(\mathbf{y}_1, \mathbf{X}_1, \beta_1) = \|\mathbf{y}_1 - \mathbf{X}_1\beta_1\|^2 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1 \tag{6}$$

, and the optimization problem in eqn. (4) becomes:

$$\min_{\beta_1} \|\mathbf{y}_1 - \mathbf{X}_1\beta_1\|^2 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1 \tag{7}$$

, where the only difference with the original optimization problem is the variable β_1. Since $\beta_1 = \sqrt{1+\lambda\alpha}\beta$, the optimization problem in eqn. (7) is equivalent to:

$$\min_{\beta} \|\mathbf{y}_1 - \mathbf{X}_1\beta_1\|^2 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1 \tag{8}$$

, which is the original objective function of the elastic net problem.

2. If we fix $\alpha = .5$, please derive the closed 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.

   For the Lasso problem $\min_{\beta_1} \|\mathbf{y}_1 - \mathbf{X}_1\beta_1\|^2 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1$, the closed solution of the i-th element of β_1 is:

   $$\beta_{1i} = signum(\beta_{1i}^{LS})(\|\beta_{i1}^{LS}\|_1 - \frac{\lambda(1-\alpha)}{1+\lambda\alpha})^+ \tag{9}$$

   , where $\beta_{i1}^{LS} = X_1^T y_1$ is the solution of the vanilla least square problem.
   Before the each time $beta_1^{i+1} - beta_1^i$ is evaluated against the tolerance, each component $beta_{1i}$ is optimized once. Fig. 1 shows the decrease of the objective function with the optimization of each single β_{1i}.

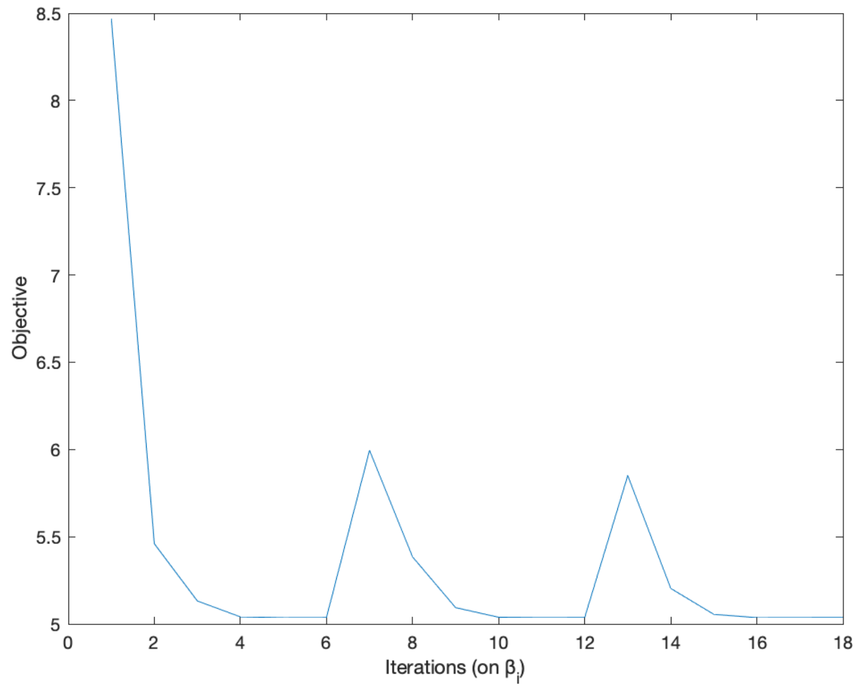


Figure 1: Decrease of $\mathbf{L}(\mathbf{y}_1, \mathbf{X}_1, \beta_1) = \|\mathbf{y}_1 - \mathbf{X}_1\beta_1\|^2 + \frac{\lambda(1-\alpha)}{1+\lambda\alpha}\|\beta_1\|_1$ with gradient descent w.r.t. each component of β_1.

Codes (MatLab):

```
n = 3;
m = 4;

lambda = 10;
alpha = 1/2;
gamma = lambda*(1-alpha)/(1+lambda*alpha)

X = rand(m,n);
y = 5 * rand(m,1);

Xcent = zscore(X);

X_1 = (1 + lambda*alpha) * [Xcent; eye(n)];
y_1 = [y; zeros(n,1)];

%% Output L in each step
mat_obj = lassoAlg_step(X_1, y_1, gamma)
l_obj = reshape(mat_obj,1,[])
```

```
plot(l_obj)
xlabel('Iterations (on beta_i)')
ylabel('Objective')

%%
lassoAlg(X_1, y_1, gamma);
%% Lasso Optimization Algorithm %%
% inputs: A (nxd matrix), y (nx1 vector), lam (scalar)
% return: xh (dx1 vector)

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

while norm(loss) > thresh
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
end
xh = xnew;
end

%% Lasso Optimization Algorithm %%
% inputs: A (nxd matrix), y (nx1 vector), lam (scalar)
% return: xh (dx1 vector)

function xh = lassoAlg_step(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
xh = [];
```

```
while norm(loss) > thresh
xold = xnew;    % need to store the previous iteration of xh
tmp = [];
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);
obj = norm(y - A * xnew, 2) + lam*(1-1/2)/(1+lam*1/2)*norm(xnew,1);
tmp = [tmp, obj];
end
loss = xnew - xold;     % update loss
%obj = norm(y - A * xnew, 2) + lam*(1-1/2)/(1+lam*1/2)*norm(xnew,1);
%xh = [xh, obj];
xh = [xh; tmp];
end
end
```

You may input your answers here. LaTeX version submission is encouraged.

Problem 2 [15 pts]

- 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.

  In PCA, the variance of the i-th principal component (PC) is:

$$\begin{aligned} Var(u_i) &= \mathbf{u}_i^T\mathbf{X}^T\mathbf{X}\mathbf{u}_i \\ &= \mathbf{u}_i^T s_i \mathbf{u}_i \\ &= s_i \mathbf{u}_i^T \mathbf{u}_i \\ &= s_i \end{aligned} \tag{10}$$

  , where s_i is the i-th singular value of X.
  Substituting u_i with $-u_i$, we have:

$$\begin{aligned} Var(-u_i) &= (-\mathbf{u}_i^T)\mathbf{X}^T\mathbf{X}(-\mathbf{u}_i) \\ &= (-\mathbf{u}_i^T)s_i(-\mathbf{u}_i) \\ &= s_i \mathbf{u}_i^T \mathbf{u}_i \\ &= s_i \end{aligned} \tag{11}$$

  , which is identical to the case of u_i. Also when $i \neq j$,

$$\begin{aligned} (-u_i)^T u_j &= -u_i^T u_j \\ &= 0 \end{aligned} \tag{12}$$

  still satisfies orthonormality.
  So $Var(u_i) = Var(-u_i)$, and any sign combination of $[\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_q]$ preserves the variance as well as orthonormality.

- We consider the case when original dimensionality of the data is much larger than the number of samples $d \gg m$ ($\mathbf{X} \in \mathbb{R}^{m\times d}$). What's the complexity of obtaining the optimal solution of PCA via Singular Value Decomposition? Please consider a more efficient solution by considering the relationships of eigenvalues/eigenvectors between $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$.

  Since $\mathbf{X} \in \mathbb{R}^{m\times d}$, the complexity of X^TX is: $O(d \times d \times m) = O(d^2m)$. Since $X^TX \in \mathbb{R}^{d\times d}$ is a square matrix, the complexity of $SVD(X^TX)$ is $O(d^3)$. Therefore, the complexity of PCA on X^TX is $O(d^2m + d^3)$.
  We know that the non-zero eigenvalues of X^TX and XX^T are the same, and hence the non-zero singular values. Similar to $PCA(X^TX)$, we know that the time complexity of the PCA of XX^T is $O(m^2d + m^3)$, which is smaller than $O(d^2m + d^3)$ when $d \gg m$. Therefore, to reduce computational cost, we can perform $PCA(XX^T)$ instead of $PCA(X^TX)$.

Problem 3 [10 pts]

Assume that in a community, there are 10% people suffer from COVID. Assume 80% of the patients come to breathing difficulty while 25% of those free from COVID also have symptoms of shortness of breath. Now please determine that if one has breathing difficulty, what's his/her probability to get COVID? (*hint*: you may consider Naive Bayes)

The conditional probability of getting shortness of breath B given getting Covid C is:

$$P(B|C) = 0.8 \tag{13}$$

The probability of shortness of breath without Covid (nC):

$$P(B|nC) = 0.25 \tag{14}$$

So the probability of getting shortness of breath is:

$$\begin{aligned} P(B) &= P(B|C)P(C) + P(B|nC)P(nC) \\ &= P(B|C)P(C) + P(B|nC)[1 - P(C)] \\ &= 0.8 \times 0.1 + 0.25 \times 0.9 \\ &= 0.305 \end{aligned} \tag{15}$$

The probability of getting Covid:

$$P(C) = 0.1 \tag{16}$$

From Naive Bayesian, the probability of Covid given shortness of breath is:

$$\begin{aligned} P(C|B) &= \frac{P(B|C)P(C)}{P(B)} \\ &= \frac{0.8 \times 0.1}{0.305} \\ &= 0.262 \end{aligned} \tag{17}$$

Problem 4 [20 pts]

Recall the objective for RatioCut: $RatioCut(A_1, A_2, ...A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i,\overline{A}_i)}{|A_i|}$. If we introduce indicator vector: $h_j \in \{h_1, h_2, ..h_k\}, j \in [1, k]$, for any vector $h_j \in R^n$, we define: $h_{ij} = \begin{cases} 0 & v_i \notin A_j \\ \frac{1}{\sqrt{|A_j|}} & v_i \in A_j \end{cases}$, we can prove: $h_i^T L h_i = \frac{cut(A_i,\overline{A}_i)}{|A_i|}$, and therefore:

$$RatioCut(A_1, A_2, ...A_k) = \sum_{i=1}^{k} h_i^T L h_i = \sum_{i=1}^{k} (H^T LH)_{ii} = tr(H^T LH), \tag{18}$$

thus we relax it as an optimization problem:

$$\underbrace{arg\ min}_{H}\ tr(H^T LH)\ \ s.t.\ H^T H = I. \tag{19}$$

Now let's explore Ncut, with objective: $NCut(A_1, A_2, ...A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i,\overline{A}_i)}{vol(A_i)}$, where $vol(A) := \sum_{i\in A} d_i, d_i := \sum_{j=1}^{n} w_{ij}$.Similar to Ratiocut, we define: $h_{ij} = \begin{cases} 0 & v_i \notin A_j \\ \frac{1}{\sqrt{vol(A_j)}} & v_i \in A_j \end{cases}$. Now

1. Please show that $h_i^T L h_i = \frac{cut(A_i,\overline{A}_i)}{vol(A_i)}$.

$$\begin{aligned}
h_i^T L h_i &= \frac{1}{2}\sum_{m=1}\sum_{n=1} w_{mn}(h_{mi} - h_{ni})^2 \\
&= \frac{1}{2}[\sum_{m\in A_i, n\notin A_i} (w_{mn}\frac{1}{\sqrt{vol(A_i)}} - 0)^2 + \sum_{m\notin A_i, n\in A_i} (0 - w_{mn}\frac{1}{\sqrt{vol(A_i)}})^2 + \\
&\sum_{m\in A_i, n\in A_i} (w_{mn}\frac{1}{\sqrt{vol(A_i)}} - w_{mn}\frac{1}{\sqrt{vol(A_i)}})^2 + \sum_{m\notin A_i, n\notin A_i} (0-0)] \\
&= \frac{1}{2}[\sum_{m\in A_i, n\notin A_i} \frac{w_{mn}}{vol(A_i)} + \sum_{m\notin A_i, n\in A_i} \frac{w_{mn}}{vol(A_i)}] \\
&= \frac{1}{2}[\frac{cut(A_i,\bar{A}_i)}{vol(A_i)} + \frac{cut(A_i,\bar{A}_i)}{vol(A_i)}] \\
&= \frac{cut(A_i,\bar{A}_i)}{vol(A_i)}
\end{aligned} \tag{20}$$

2. Show that $NCut(A_1, A_2, ...A_k) = tr(H^T LH)$.

   From the definition of Cut,

$$cut(A_1, A_2, ...A_k) = \frac{1}{2}\sum_{i=1}^{k} W(A_i, \bar{A}_i) \tag{21}$$

From the result of Problem 4.1: $h_i^T L h_i = \frac{cut(A_i, \overline{A}_i)}{vol(A_i)}$, we have:

$$\begin{aligned} h_i^T L h_i &= \frac{cut(A_i, \overline{A}_i)}{vol(A_i)} \\ &= \frac{1}{2} \frac{W(A_i, \bar{A}_i)}{vol(A_i)} \end{aligned} \tag{22}$$

So:

$$\begin{aligned} NCut(A_1, A_2, ...A_k) &= \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{vol(A_i)} \\ &= \sum_{i=1}^{k} \frac{cut(A_i, \overline{A}_i)}{vol(A_i)} \\ &= \sum_{i=1}^{k} h_i^T L h_i \\ &= \sum_{i=1}^{k} (H^T L H)_{ii} \\ &= tr(H^T L H) \end{aligned} \tag{23}$$

3. The constraint now is: $H^T D H = I$.

$$\begin{aligned} h_i^T D h_i &= \sum_j d_l h_{ji}^2 \\ &= \sum_{j \in A_i} d_l \frac{1}{vol(A_i)} \\ &= \frac{1}{vol(A_i)} \sum_{j \in A_i} \sum_{m=1}^{N} w_{jm} \\ &= 1 \end{aligned} \tag{24}$$

And since $(H^T D H)_{ii} = h_i^T D h_i$, all diagonal elements of $H^T D H$ are 1. Therefore, $H^T D H = \mathbf{I}$.

4. Find the solution to $\underbrace{arg\ min}_{H}\ tr(H^T L H)\ \ s.t.\ H^T D H = I$.

Define a new matrix, M:

$$M = D^{\frac{1}{2}} H \tag{25}$$

, where D is the degree matrix, and thus $H = D^{-\frac{1}{2}} M$. Then, the minimization problem is equivalent to:

$$\begin{aligned} \min_M Tr(M^T D^{-\frac{1}{2}} L D^{-\frac{1}{2}} M) \\ s.t. M^T M = I \end{aligned} \tag{26}$$

, where $(D^{-\frac{1}{2}})^T = D^{-\frac{1}{2}}$, because the degree matrix D is symmetrical. The problem can be further transformed into:

$$\begin{gathered}\min_{M} Tr(\frac{M^T D^{-\frac{1}{2}} L D^{-\frac{1}{2}} M}{M^T M}) \\ s.t. M^T M = I\end{gathered} \tag{27}$$

, in which form we can apply the Rayleigh-Ritz theorem, since all matrices in the problem are real matrices ($M^H = M^T$). We know that the minimun value of $\frac{M^T D^{-\frac{1}{2}} L D^{-\frac{1}{2}} M}{M^T M}$ is the smallest eigenvalue of $D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$. Therefore, the solution of M^T is a matrix that has the eigenvectors of L as columns. By substituting $H = D^{-\frac{1}{2}} M$, we can get the solution of H.

Problem 5 [10 pts]

We consider the following optimization problem ($\mathbf{Y}$ is given and generated randomly):

$$\min_{\mathbf{X}} \frac{1}{2}\|\mathbf{X}-\mathbf{Y}\|_F^2 + \|\mathbf{X}\|_* \tag{28}$$

where $\mathbf{Y}, \mathbf{X} \in \mathbb{R}^{100\times 100}$ and $\|\cdot\|_*$ denotes the nuclear norm (sum of singular values). Now please use gradient descent method to update $\mathbf{X}$. ($\frac{\partial\|\mathbf{X}\|_*}{\partial \mathbf{X}} = \mathbf{U}\mathbf{V}^T$, where $\mathbf{U}, \mathbf{V}$ is obtained from reduced SVD, namely $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = svd(\mathbf{X}, 0)$). Plot the objective changes with 1000 iteration.

The objective function is: $L(X,Y) = \frac{1}{2}\|X-Y\|_F^2 + \|X\|_*$, and its derivative w.r.t. X is:

$$\begin{aligned}\frac{\partial L}{\partial X} &= \frac{1}{2}(X-U) + UV^T \\ &= X - Y + UV^T\end{aligned} \tag{29}$$

, where $[U,S,V] = SVD_{red}(X)$ are derived from the reduced SVD of X.
The change of objective function L w.r.t. number of iterations in gradient descent method is shown in Fig. 1. Codes implemented with MatLab.

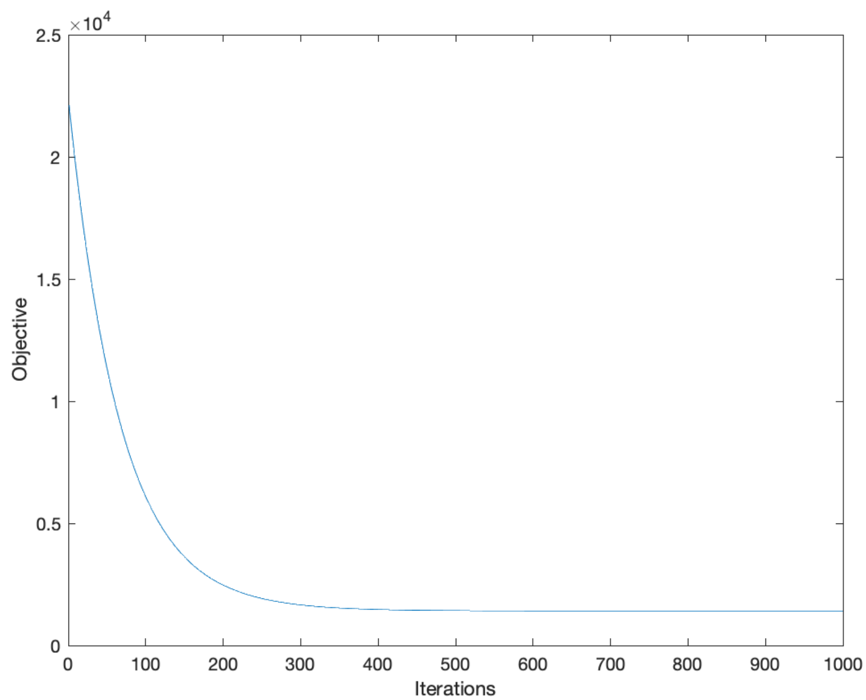


Figure 2: Decrease of $L(X,Y) = \frac{1}{2}\|X-Y\|_F^2 + \|X\|_*$ with gradient descent.

Codes (MatLab):

```
Y = 5*rand(100,100);
```

```
X = 5*rand(100,100);
eta = 0.007;
l_L = [];

for i = [1:1000]
if rem(i,100) == 0
i
L(X,Y)
end
l_L = [l_L, L(X,Y)];

X_1 = X - eta * dLdX(X,Y);
X = X_1;
i = i+1;
end

plot(l_L)
xlabel('Iterations')
ylabel('Objective')

%%
function drv = dLdX(X,Y)
[U,S,V] = svd(X,0);
drv = (X - Y) + U * V';
end

function obj = L(X,Y)
obj = 1/2 * norm(X-Y, 'fro')^2 + norm(svd(X,0),1);
end
```

Problem 6 [20 pts]

We turn to Logistic Regression:

$$\min_{\beta} \sum_{i=1}^{m} ln(1 + e^{\langle \beta, \hat{x}_i \rangle}) - y_i \langle \beta, \hat{x}_i \rangle, \tag{30}$$

where $\beta = (w; b), \hat{x} = (x; 1)$. Assume $m = 100, x \in \mathbb{R}^{99}$. Please randomly generate x, y and find the optimal β via 1) gradient descent; 2) Newton's method and 3) stochastic gradient descent (SGD) where the batch-size is 1. (need consider choosing appropriate step-size if necessary). Change $m = 1000, x \in \mathbb{R}^{999}$, observe which algorithm will decrease the objective faster in terms of iteration (X-axis denotes number of iteration) and CPU time. [You will receive another 5 bonus points if you implement backtracking line search]

The objective function is $L(\beta, x_i, y_i) = ln(1 + e^{\langle \beta, \hat{x}_i \rangle}) - y_i \langle \beta, \hat{x}_i \rangle$.
In gradient method, we have the gradient:

$$\frac{\partial L}{\partial \beta} = \frac{e^{<\beta, \hat{x}_i>} \hat{x}_i{}^T}{1 + e^{<\beta, \hat{x}_i>}} - y_i \hat{x}_i{}^T \tag{31}$$

and: $\beta^{i+1} = \beta^i - \lambda \frac{\partial L}{\partial \beta}$. The decrease of L with iterations is shown in the figure below.

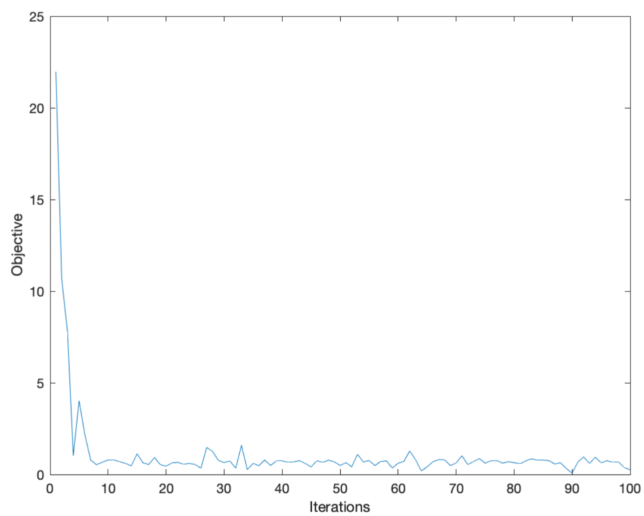


Figure 3: Gradient descent (m=100, n=99).

In Newton's method, we have:

$$\begin{aligned} \nabla &= \sum_{i=1}^{m} [p(x_i) - y_i] = X(p - y) \\ H &= XWX^T \end{aligned} \tag{32}$$

, where W is a diagonal matrix with the i-th diagonal element as $p(x_i)[1-p(x_i)]$. In each update, $w^{i+1} = w^i - H_i^{-1} \cdot \nabla$. The decrease of L with iterations is shown in the figure below.

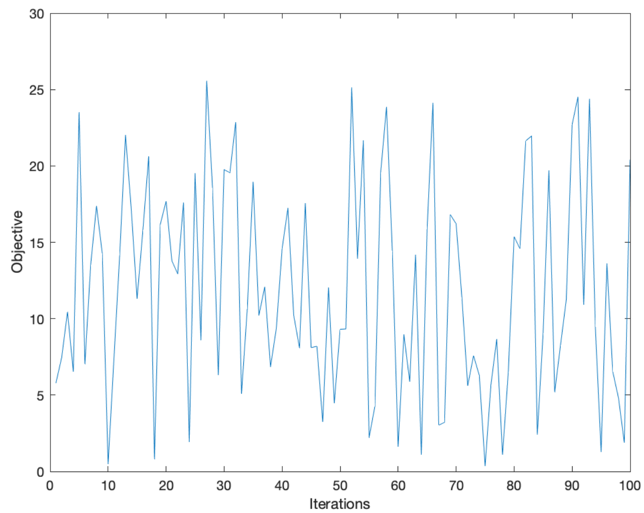


Figure 4: Newton's method (m=100, n=99).

Note: In Newton's method, the objective function can reach very low values, but will then increase. The performance of Newton's method is not ideal in this case.
In SGD, we have:

$$g = \frac{\partial L}{\partial w} = (p(x_i) - y_i)x_i \tag{33}$$

and for each update $w^{i+1} = w^i - \alpha g$. The decrease of L with iterations is shown in the figure below.

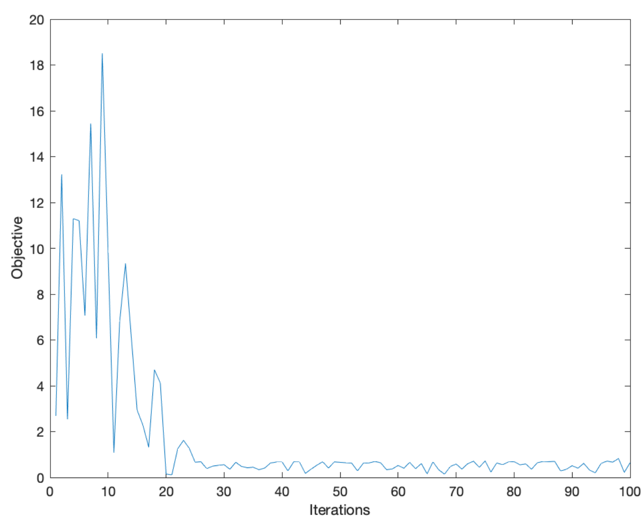


Figure 5: SGD method (m=100, n=99, $\alpha = 0.1$).

From the above results, we can see that the three methods in the order of fastest to slowest convergence is: gradient descent, SGD, Newton's method. The time consumption of the three methods are: gradient descent - 0.01s, Newton's method - 0.29s, SGD - 0.01s.
In the case of $m = 1000, n = 999$, the decrease of L with iterations is shown in the figures below.

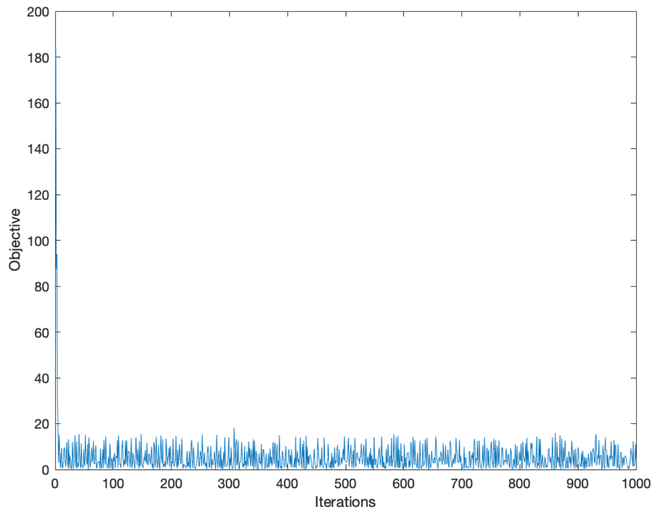


Figure 6: Gradient descent (m=1000, n=999).

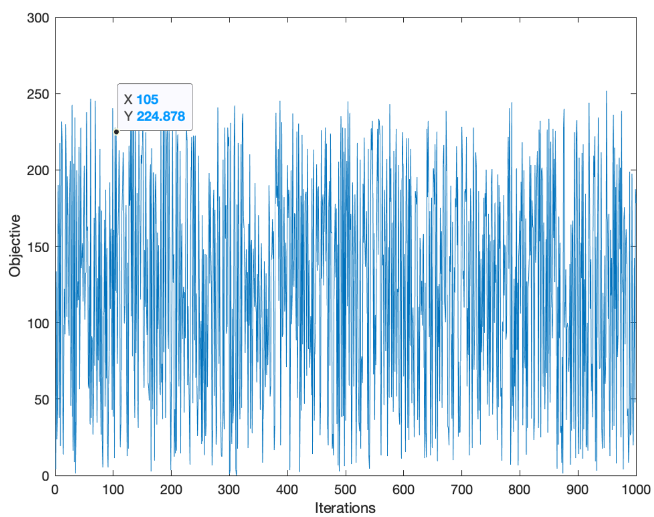


Figure 7: Newton's method (m=1000, n=999).

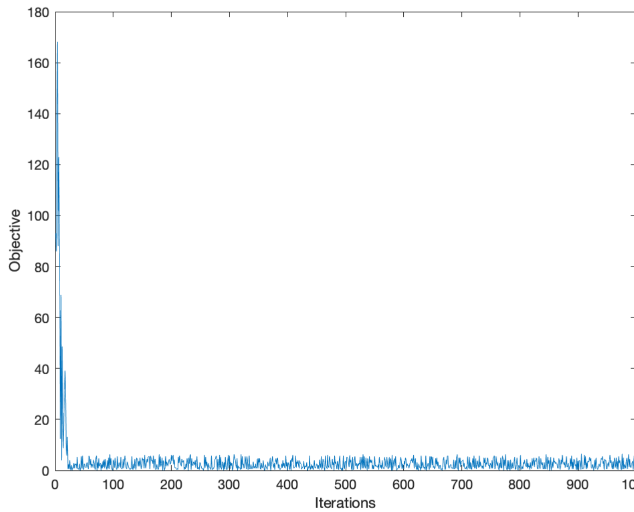


Figure 8: SGD method (m=1000, n=999, $\alpha = 0.1$).

From the above results, we can see that when m, n are increased, the three methods in the order of fastest to slowest convergence is: gradient descent, SGD, Newton's method. The time consumption of the three methods with increased m, n are: gradient descent - 0.04s, Newton's method - 187.81s, SGD - 0.02s.

Codes (MatLab):

```
clear all;

m = 1000;
n = 999;

y = rand(1,m);
w = rand(n,1);
x = [w; rand]
beta = [w; rand];
lambda = 0.1;
l = [];

%%Grd descent
% t = cputime;
% for i = [1:m]
%     x_i = [rand(n,1); 1];
%     y_i = y(i);
%     beta = beta - dLdb(beta,x_i,y_i) * lambda;
%     obj = L(beta,x_i,y_i);
%     l = [l, obj];
% end
```

```
% e = cputime - t;
% disp('GD')
% disp(n)
% disp(e)


%Newtons method

% X = [rand(m,n), ones(m,1)];
% p_vec = zeros(1,m);
%
% t = cputime;
% for i = [1:m]
%     x_i = X(i,:);
%     y_i = y(i);
%
%     px_i = pxi(beta, x_i);
%     obj = L(beta,x_i,y_i);
%     grd = x_i * (px_i - y_i);
%
%     W = zeros(m,m);
%     for i = [1:m]
%        W(i,i)=pxi(beta, X(i,:));
%     end
%     hes = X' * W * X;
%     beta = beta - pinv(hes) * grd';
%
% %     for i = [1:m]
% %         p_vec(i) = pxi(beta, x_i);
% %     end
% %     grd = X' * (p_vec - y)';
% %
% %     W = zeros(m,m);
% %     for i = [1:m]
% %        W(i,i)=pxi(beta, X(i,:));
% %     end
% %     hes = X' * W * X;
% %     beta = beta - inv(hes) * grd;
%
%     l = [l, obj];
% end
% e = cputime - t;
% disp('Newton')
% disp(n)
% disp(e)
```

```
%%SGD

t = cputime;
alpha = 0.1;
for i = [1:m]
y_i = y(i);
x_i = [rand(n,1); 1];
px_i = pxi(beta, x_i);
g = (px_i - y_i) * x_i;
beta = beta - alpha * g;
obj = L(beta,x_i,y_i);
l = [l, obj];
end
e = cputime - t;
disp('SGD')
disp(n)
disp(e)

%% Test

%%
plot(l)
xlabel('Iterations')
ylabel('Objective')
%%
function obj = L(beta,x_i,y_i)
obj = log(1 + exp(dot(beta' , x_i))) - y_i * dot(beta' , x_i);
end

%%
function grd = dLdb(beta,x_i,y_i)
grd = exp(dot(beta' , x_i))/(1 + exp(dot(beta' , x_i))) - y_i * x_i;
end

%%
function likelihood = pxi(beta, x_i)
likelihood = 1/(1+exp(-dot(beta', x_i)));
end
```

Problem 7 [10 pts]

Please design an (either toy or real-world) experiment to demonstrate that PCA can be helpful for denoising.

Take a 480*360 picture of a snowy owl as an example.

Figure 9: Original picture.

Add Gaussian noise with $\mu = 0, \sigma = 0.1$ to all three RGB channels, we have a noisey image.

Figure 10: Noisey picture.

Performing PCA on the noisey picture with the first 100 PCs, we have a reconstructed picture.

Figure 11: Reconstructed picture.

We can see that the noises in the reconstructed picture are smoothed out.

Codes (Mathematica):

```
cwd = NotebookDirectory[];
{m, n} = Import[cwd <> "snowy_owl.jpg", "ImageSize"];
original = Import[cwd <> "snowy_owl.jpg"];
noisey = ImageEffect[original, {"GaussianNoise", 0.1}]
pxl = N[Flatten[ImageData[noisey], 1]];
mean = Mean[pxl];
pxlCtr = # - mean & /@ pxl;
(*RGB channels*)
R = ArrayReshape[pxlCtr[[All, 1]], {m, n}]; G =
ArrayReshape[pxlCtr[[All, 2]], {m, n}]; B =
ArrayReshape[pxlCtr[[All, 3]], {m, n}];

(*Reconstruction single channel*)

recon[channel_, nPC_] := Module[{u, s, v, picrecon},
(*SVD*){u, s, v} = SingularValueDecomposition[channel];
(*Reconstruction using nPC columns*)
picrecon =
u[[All, ;; nPC]].s[[;; nPC, ;; nPC]].Transpose[v[[All, ;; nPC]]];
picrecon]
(*Reconstruct 3 channels*)
```

```
reconRGB[nPC_] :=
ArrayReshape[
Transpose[
Flatten /@ {recon[R, nPC], recon[G, nPC], recon[B, nPC]}], {n, m,
3}]
rgbPlot[plot_] :=
ArrayPlot[ArrayReshape[plot, {n, m, 3}],
ColorFunction -> Function[p, RGBColor[p[[1]], p[[2]], p[[3]]]],
ColorFunctionScaling -> True]

rgbPlot[reconRGB[#]] & /@ {100}
```

Bonus Problem 8 [10 pts]

$$\textbf{Solve:} \quad \min \|\mathbf{x}\|_0 \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} = \mathbf{y}. \tag{34}$$

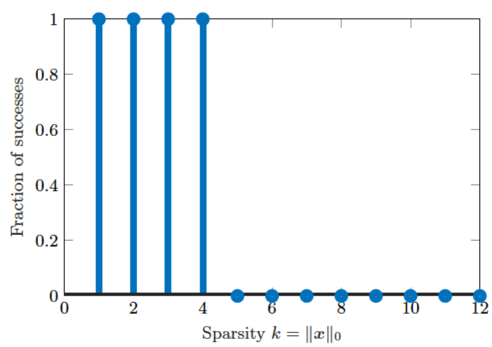
We have proved that if $\mathbf{y} = \mathbf{A}\mathbf{x}_o$ with

$$\|\mathbf{x}_o\|_0 \ \leq \ \frac{1}{2}\,\text{krank}(\mathbf{A}). \tag{35}$$

Then $\mathbf{x}_o$ is the unique optimal solution to the ℓ^0 minimization problem

$$\min \|\mathbf{x}\|_0 \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} = \mathbf{y}. \tag{36}$$

However, when $\mathbf{A}$ is of size 5×12, the following figure illustrates the fraction of success across 100 trials. Apparently $krank(\mathbf{A}) \leq rank(\mathbf{A}) \leq 5$, therefore, when sparsity $k = 1, 2$ satisfying Eq. (35)



it has 100% recovery success rate is not surprising. However, the above experiment also shows even $k = 3, 4$ which violates Eq. (35), still it can be recovered at 100%. Please explain this phenomenon.