THE UNIVERSITY OF TEXAS
AT AUSTIN

EE381V LARGE SCALE OPTIMIZATION

Problem Set 0

Edited by LaTeX

Department of Computer Science

STUDENT

Jimmy Lin

xl5224

COURSE COORDINATOR

Sujay Sanghavi

UNIQUE NUMBER

17350

RELEASE DATE

September 5, 2014

DUE DATE

September 11, 2014

TIME SPENT

10 hours

September 10, 2014

Table of Contents

Chapter 1

Matlab and Computational Assignment

1.1 Algorithm 1: Least Square

The command to invoke standarded least-squared regression:

```
>> algo1()
```

Note that *algo1.m* includes scripts for all three datasets.

1.1.1 Small-scale dataset: Succeed

The brief summary of applying standarded least-squared regression on small-scale dataset is as follows:

- Total CPU time (secs) = 0.18
- CPU time per iteration = 0.02
- Regression Error $||X\beta - y||$: 1.1698e-10
- Testing Error $||X_{test}\beta - y_{test}||$: 23.058394 (pretty large)

1.1.2 Medium-scale dataset: Succeed

The brief summary of applying standarded least-squared regression on medium-scale dataset is as follows:

- Total CPU time (secs) = 43.95
- CPU time per iteration = 5.49
- Regression Error $||X\beta - y||$: 3.2594e-09
- Testing Error $||X_{test}\beta - y_{test}||$: 19.862394 (pretty large)

1.1.3 Large-scale dataset: Failed

This standarded least-square regression task is too large-scaled to be computed.

1.2 Algorithm 2: optimization with LASSO

The command to invoke least-squared regression with LASSO:

```
>> algo2()
```

Note that *algo2.m* includes scripts for all three datasets.

1.2.1 Small-scale dataset: Succeed

The brief summary of applying least-squared regression with LASSO on small-scale dataset is as follows:

- Total CPU time (secs) = 0.38
- CPU time per iteration = 0.02
- Regression Error: 6.7886e-10
- Testing Error: 0.144338
- Supports (non-zeros entries of β): 43 (500 atoms in total)

1.2.2 Medium-scale dataset: Succeed

The brief summary of applying least-squared regression with LASSO on medium-scale dataset is as follows:

- Total CPU time (secs) = 126.66
- CPU time per iteration = 4.87
- Regression Error: 4.4292e-09
- Testing Error: 0.078289
- Supports (non-zeros entries of β): 342 (5000 atoms in total)

1.2.3 Large-scale dataset: Failed

This least-square regression with LASSO task is too large-scaled to be computed.

Remarks: Least-squared regression with LASSO does outperfrom standarded least-squared regression in its prediction accuracy. Besides, it has higher computational complexity since it requires more iterations for convergence and each iteration cost more time to complete.

1.3 Orthogonal Matching Pursuit

The command to invoke regression with OMP preprocessing:

```
>> regress_omp()
```

1.3.1 Small-scale Dataset: Succeed

The brief summary of applying regression with OMP feature selection on small-scale dataset is as follows:

- Indices of Features selected by OMP (with order): 402, 235, 86, 11, 108.
- Elapsed time is 0.198106 seconds.
- Regression Error $||X\beta - y||$: 5.3785e-02
- Testing Error $||X_{test}\beta - y_{test}||$: 4.4208e-02

1.3.2 Medium-scale Dataset: Succeed

The brief summary of applying regression with OMP feature selection on medium-scale dataset is as follows:

- Indices of Features selected by OMP (with order): 577, 2760, 561, 3614, 3958.
- Elapsed time is 0.209093 seconds.
- Regression Error $||X\beta - y||$: 2.1955e-01
- Testing Error $||X_{test}\beta - y_{test}||$: 1.8219e-02

1.3.3 Large-scale Dataset: Succeed

The brief summary of applying regression with OMP feature selection on large-scale dataset is as follows:

- Indices of Features selected by OMP (with order): 17099, 29426, 35373, 22452, 43354.
- Elapsed time is 2.994790 seconds.
- Regression Error $||X\beta - y||$: 6.9964e-01
- Testing Error $||X_{test}\beta - y_{test}||$: 6.4437e-03

Note that Elapsed time is defined as OMP preprocessing and regression for selected atoms on that dataset, but not included computation for regression error and testing error.

Remarks: Least-squared regression on OMP feature selection performs much better than standarded least-squared regression and least-squared regression with LASSO. Besides, it has lower computational complexity since it allows the large-scale dataset (third dataset) to be regressed.

Chapter 2

Linear Algebra Review

2.1 More Range and Nullspace

2.1.1 Smallest and Largest rank of $C = AB$

Conditions: $A \in \mathbb{R}^{10\times 10}$ with $rank(A) = 5$ and $B \in \mathbb{R}^{10\times 10}$ with $rank(B) = 5$.
Sylvester's rank inequality: $\forall A \in R^{m\times k}, B \in \mathbb{R}^{k\times n}$

$$rank(A) + rank(B) - k \leq rank(AB)$$

Smallest rank of $C = AB$ is $rank(A) + rank(B) - k = 5 + 5 - 10 = 0$.
Largest rank of $C = AB$ is $min(rank(A), rank(B)) = min(5, 5) = 5$.

2.1.2 Largest rank of $C = AB$

Conditions: $A \in \mathbb{R}^{10\times 15}$ with $rank(A) = 7$ and $B \in \mathbb{R}^{15\times 11}$ with $rank(B) = 8$.
Largest rank of $C = AB$ is $min(rank(A), rank(B)) = min(7, 8) = 7$.

2.2 Riesz Representation Theorem

Linear map $f : \mathbb{R}^n \to \mathbb{R}$ has two critical properties due to its linearity:

$$\text{additivity: } f(x + y) = f(x) + f(y), \forall x, y \in dom(f) \tag{2.1}$$
$$\text{homogeneity: } f(\alpha x) = \alpha f(x), \forall \alpha \in \mathbb{R}, x \in dom(f) \tag{2.2}$$

Let arbitrary vector $\mathbf{w} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{R}^n$. Then we can denote $\mathbf{w}$ as linear combination of standard basis

$$\mathbf{w} = \alpha_1\mathbf{e}_1 + \alpha_2\mathbf{e}_2 + \cdots + \alpha_n\mathbf{e}_n \tag{2.3}$$

Now we start to show that $f(\mathbf{w})$ can be represented as inner product of $\mathbf{w}$ and another vector.

$$\begin{aligned}
f(\mathbf{w}) &= f(\alpha_1\mathbf{e}_1 + \alpha_2\mathbf{e}_2 + \cdots + \alpha_n\mathbf{e}_n) && \text{standard basis representation(2.3)} && (2.4)\\
&= f(\alpha_1\mathbf{e}_1) + f(\alpha_2\mathbf{e}_2) + \cdots + f(\alpha_n\mathbf{e}_n) && \text{additivity of linear map(2.1)} && (2.5)\\
&= \alpha_1 f(\mathbf{e}_1) + \alpha_2 f(\mathbf{e}_2) + \cdots + \alpha_n f(\mathbf{e}_n) && \text{additivity of linear map(2.2)} && (2.6)\\
&= \langle (f(\mathbf{e}_1), f(\mathbf{e}_2), \ldots, f(\mathbf{e}_n)), (\alpha_1, \alpha_2, \ldots, \alpha_n)\rangle && \text{definition of inner product} && (2.7)\\
&= \langle \mathbf{x}, \mathbf{w}\rangle && \mathbf{x} = (f(\mathbf{e}_1), f(\mathbf{e}_2), \ldots, f(\mathbf{e}_n)) && (2.8)
\end{aligned}$$

Hence, we have successfully proved that

$$\forall \text{ linear map } f : \mathbb{R}^n \to \mathbb{R}, \exists \mathbf{x} \in \mathbb{R}^n, f(\mathbf{w}) = \langle \mathbf{x}, \mathbf{w}\rangle \tag{2.9}$$

2.3 Polynomial Vector Spaces

2.3.1 $Tp = 2p(t) - tp'(t)$: False

T is represented by an diagonal matrix with all diagonal entries $b_i = 2 - i$. Obviously, for $i = 2, b_i = 0$. That is to say, the second row of T is zeros. Hence, T is not full-rank and then T is not surjective. There must exist a q that cannot be reached by Tp. For example: t^2.

2.3.2 $Tp = 2p(t) - 3tp'(t)$: True

T is represented by an diagonal matrix with all diagonal entries $b_i = 2 - 3 * i$. Obviously, for $\nexists i \in \mathbb{Z}, b_i = 0$. That is to say, the matrix T is full rank and then T is surjective: for every polynomial $q \in V$, there exists a polynomial $p \in V$, with $Tp = q$.

2.3.3 Characterization of Surjectivity: $a_0 \neq 0$

To make sure that the corresponding mapping T to be surjective: for every polynomial(vector) $q \in V$, there does exist a polynomial(vector) $p \in V$ such that $Tp = q$, we need to gurantee the T corresponds to full-rank matrix. In general, matrix T is still a diagonal matrix with its diagonal entries $b_i (i \in [0, d])$ to be

$$
\begin{aligned}
b_i &= a_0 + a_1 \cdot i + a_2 \cdot i(i-1) + \cdots + a_d \cdot i(i-1)...(i-d) && (2.10)\\
&= \sum_{j=0}^{d} \mathbb{I}(i \geq j) \cdot a_j \frac{i!}{(i-j)!} && (2.11)
\end{aligned}
$$

Since T is diagonal matrix, we need to make sure every diagonal entries is non-zero, that is

$$b_i \neq 0 \tag{2.12}$$

In conclusion, we have

$$\forall i, \sum_{j=0}^{d} \mathbb{I}(i \geq j) \cdot a_j \frac{i!}{(i-j)!} \neq 0 \implies \forall q \in V, \exists p \in V, s.t.\ Tp = q \tag{2.13}$$

2.4 Rank

2.4.1 Show that $rank(A) \leq min\{m, n\}$

Proof. rank(A) is defined as the number of columns that are linearly independent. Then, we have

$$rank(A) \leq \text{number of linearly independent columns} \leq n \tag{2.14}$$

Since the number of linearly independent columns equals to the number of linearly independent rows,

$$rank(A) \leq \text{number of linearly independent rows} \leq m \tag{2.15}$$

From (2.14) and (2.15), it is easy to derive the desired result:

$$rank(A) \leq min\{m, n\} \tag{2.16}$$

□

2.4.2 Sylvester's rank inequality

Proof of $rank(AB) \leq min\{rank(A), rank(B)\}$. Let $AB\mathbf{x} \in Col(AB), \mathbf{x} \in \mathbb{R}^k$, then $AB\mathbf{x} = A(B\mathbf{x}) \in Col(A)$. Since $B\mathbf{x}$ may not fill up the whole $Col(A)$, we have

$$Col(AB) \subset Col(A) \tag{2.17}$$

Since *rank* is defined to be the dimensionality of column space, we proved

$$rank(AB) \leq rank(A) \tag{2.18}$$

By rank-nullity theorem:

$$rank(B) = n - nullity(B) \tag{2.19}$$

Similarly, we have

$$rank(AB) = n - nullity(AB) \tag{2.20}$$

Since $B\mathbf{x} \Rightarrow AB\mathbf{x}$ and $AB\mathbf{x} \not\Rightarrow B\mathbf{x}$, we have

$$nullity(B) \leq nullity(AB) \tag{2.21}$$

Based on (2.19) and (2.20), we have

$$rank(AB) \leq rank(B) \tag{2.22}$$

In terms of (2.18) and (2.22), we have

$$rank(AB) \leq min\{rank(A), rank(B)\} \tag{2.23}$$

□

Proof of $rank(A) + rank(B) - k \leq rank(AB)$. We can make use of Frobenius rank inequality by instantiating B as I, C as B.

$$rank(AI) + rank(BI) \leq rank(I) + rank(AIB) \tag{2.24}$$

Since $rank(I_{p \times k}) \leq min\{p, k\} \leq k$, then

$$rank(A) + rank(B) \leq k + rank(AB) \tag{2.25}$$

$$rank(A) + rank(B) - k \leq rank(AB) \tag{2.26}$$

Hence, we leave the essential part of proof to frobenius rank inequality. □

2.4.3 Subadditivity: $rank(A+B) \leq rank(A) + rank(B)$

Proof. Since both A and B are $m \times n$ matrix, we can denote them as

$$A = \begin{pmatrix} | & | & | & | & | & | \\ a_0 & a_1 & \dots & a_j & \dots & a_n \\ | & | & | & | & | & | \end{pmatrix},\ B = \begin{pmatrix} | & | & | & | & | & | \\ b_0 & b_1 & \dots & b_j & \dots & b_n \\ | & | & | & | & | & | \end{pmatrix} \tag{2.27}$$

Obviously a_j, b_j are column vector of A and B respectively. Then we can represent $(A+B)\mathbf{x}$ as

$$(A+B)\mathbf{x} = A\mathbf{x} + B\mathbf{x} = \sum_j^n a_j\mathbf{x}_j + \sum_j^n b_j\mathbf{x}_j \tag{2.28}$$

It is easy to see that

$$Col(A+B) = Col(A) + Col(B) - Col(A) \cap Col(B) \tag{2.29}$$

Note that we remove the space overlapped by $Col(A)$ and $Col(B)$ since this space should not be counted twice for $Col(A+B)$.

In general, some a_j or b_j are coupled (mutally dependent), in which cases, $\emptyset \subset Col(A) \cap Col(B)$. In summary, we have

$$rank(A+B) \leq rank(A) + rank(B) \tag{2.30}$$

At the best case, all a_j and b_j are linearly independent and then $Col(A) \cap Col(B) = \emptyset$. Hence, in this case, $rank(A+B) = rank(A) + rank(B)$. □

2.4.4 Frobenius Rank Inequality

Proof. If $U \subset V$ and $X : U \to W$, then

$$dim\ kerX|_U \leq dim\ kerX \tag{2.31}$$

By Rank-Nullity Theorem, we have

$$dim\ kerX = dimV - dim\ RanX \tag{2.32}$$

Hence, in general we have

$$dim\ kerX|_U \leq dimV - dim\ RanX \tag{2.33}$$

Let $U = RanBC$ and $V = RanB$ and $X = A$, we have

$$dim\ kerA|_{RanBC} \leq dim\ RanB - dim\ RanA \tag{2.34}$$

$$\leq dim\ RanB - dim\ RanAB \tag{2.35}$$

Since $dim\ kerA|_{RanBC} = dim\ RanB - dim\ RanABC$, we have

$$dim\ RanBC - dim\ RanABC \leq dim\ RanB - dim\ RanAB \tag{2.36}$$

That is

$$dim\ RanAB + dim\ RanBC \leq dim\ RanB + dim\ RanABC \tag{2.37}$$

□

Appendix A

Codes Printout

A.1 Sparse Recovery

A.1.1 Algorithm 1: Least Square

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Scripts invoking cvx least-square routines to
%%% solve problems using our three datasets.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% standard least-square for Small-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cvx_begin
    variable b1(size(X1,2))
    minimize( norm( X1*b1-y1 ) )
cvx_end

RegressionError1 = norm( X1*b1-y1 )
TestingError1 = norm( X1test*b1 - y1test )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% standard least-square for Medium-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cvx_begin
    variable b2(size(X2,2))
    minimize( norm( X2*b2 - y2 ) )
cvx_end

RegressError2 = norm( X2*b2 - y2 )
TestError2 = norm( X2test*b2 - y2test )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% standard least-square for Large-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cvx_begin
    variable b3(size(X3,2))
    minimize( norm( X3*b3-y3 ) )
cvx_end

RegressionError3 = norm( X3*b3 - y3 )
TestingError3 = norm( X3test*b3 - y3test )
```

A.1.2 Algorithm 2: Optimization with LASSO

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Scripts invoking cvx least-square routines to
%%% solve LASSO problems using our three datasets.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format short e
EPSILON = 10e-5;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% LASSO least-square for Small-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cvx_begin
    variable b1(size(X1,2))
    minimize( norm( X1*b1-y1 ) + norm(b1,1) )
cvx_end

RegressionError1 = norm( X1*b1-y1 )
TestingError1 = norm( X1test * b1 - y1test )
Support1 = sum(((b1 < EPSILON) + (b1 > -EPSILON)) < 2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% LASSO least-square for Medium-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cvx_begin
    variable b2(size(X2,2))
    minimize( norm( X2*b2-y2 ) + norm(b2, 1))
cvx_end

RegressionError2 = norm( X2*b2-y2 )
TestingError2 = norm( X2test * b2 - y2test )
Support2 = sum(((b2 < EPSILON) + (b2 > -EPSILON)) < 2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% LASSO least-square for Large-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cvx_begin
    variable b3(size(X3,2))
    minimize( norm( X3*b3-y3 ) + norm(b3, 1) )
cvx_end

RegressionError3 = norm( X3*b3-y3 )
TestingError3 = norm( X3test * b3 - y3test )
Support3 = sum(((b3 < EPSILON) + (b3 > -EPSILON)) < 2)
```

A.2 Orthogonal Matching Pursuit

A.2.1 OMP Routine

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Orthogonal matching Pursuit
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Iset = omp (X, y, SPARSITY)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% INITIALIZATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[target_feat_dot_prod, target_feat_idx] = max(X' * y);
Iset = [target_feat_idx];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% AUGMENTATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
residual = y;
for iter = 1:(SPARSITY-1),
    % perpendicular complement of y to X_i
    phi = X(:, Iset);
    P = phi * inv(phi'*phi) * phi';
    I = eye(size(P));
    residual = (I - P) * residual;
    % elect new atom and add to selected atom set
    [target_feat_dot_prod, target_feat_idx] = max(X' * residual);
    % NOTE that new feature(atom) will not pre-exist in Iset
    % This is theoreotically guaranteed by orthogonal projection
    Iset = [Iset, target_feat_idx];
end
end
```

A.2.2 Regression Scripts

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Invoke CVX least square regression after OMP
%%% feature selection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SPARSITY = 5; % SPARSITY parameter for OMP

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Small-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
Iset1 = omp(X1, y1, SPARSITY);
subX1 = X1(:, Iset1);
cvx_begin
    variable sub_b1(SPARSITY);
    minimize( norm(subX1 * sub_b1 - y1) )
cvx_end
toc

Iset1
RegressionError1 = norm(subX1*sub_b1 - y1)
TestingError1 = norm(X1test(:,Iset1)*sub_b1 - y1test)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Medium-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
Iset2 = omp(X2, y2, SPARSITY);
subX2 = X2(:, Iset2);
cvx_begin
    variable sub_b2(SPARSITY);
    minimize( norm(subX2 * sub_b2 - y2) )
cvx_end
toc

Iset2
RegressionError2 = norm(subX2*sub_b2 - y2)
TestingError2 = norm(X2test(:,Iset2)*sub_b2 - y2test)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Large-scale dataset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
Iset3 = omp(X3, y3, SPARSITY);
subX3 = X3(:, Iset3);
cvx_begin
    variable sub_b3(SPARSITY);
    minimize( norm(subX3 * sub_b3 - y3) )
cvx_end
toc

Iset3
RegressionError3 = norm(subX3*sub_b3 - y3)
TestingError3 = norm(X3test(:,Iset3)*sub_b3 - y3test)
```