Homework Set 1, CPSC 8420

LastName, FirstName

Due 10/05/2023, 11:59PM EST

Ridge Regression

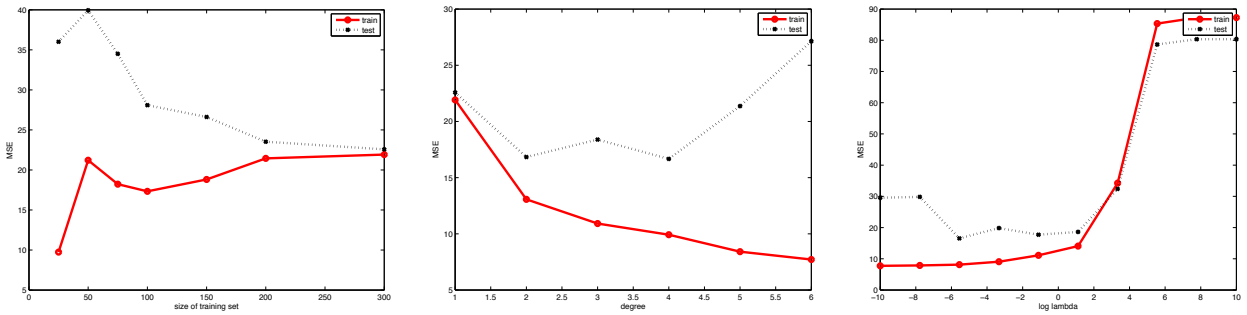
Please show that for arbitrary $\mathbf{A} \in \mathbb{R}^{n\times p}$, $(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}_p)^{-1}\mathbf{A}^T = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda\mathbf{I}_n)^{-1}$, where $\lambda > 0$. Now assume $n = 100$, please compare the time consumption when $p = [10, 100, 1000, 2000]$ and plot the results appropriately (*e.g.* in one figure where X-axis denotes p while Y-axis the time consumption).

Least Squares Extension

Assume $\mathbf{A}, \mathbf{X}, \mathbf{C}, \mathbf{Y}, \mathbf{U} \in \mathbb{R}^{n\times n}$. Given the fact that $vec(\mathbf{AUC}) = (\mathbf{C}^T \otimes \mathbf{A})vec(\mathbf{U})$, where $\otimes$ denotes *Kronecker product*, please use *least squares* we learned in class to solve:

$$\min_{\mathbf{X}} \|\mathbf{AX} + \mathbf{XC} - \mathbf{Y}\|_F^2. \tag{1}$$

To verify the correctness of your solution: 1) randomly generate $\mathbf{A}, \mathbf{X}^*, \mathbf{C}$; 2) set $\mathbf{Y} = \mathbf{AX}^* + \mathbf{X}^*\mathbf{C}$; 3) by making use of least squares, you can obtain the optimal $vec(\mathbf{X})$ given $\mathbf{A}, \mathbf{C}, \mathbf{Y}$ and 4) compare with $vec(\mathbf{X}^*)$.



(a) (b) (c)

Figure 1: MSE vs (a) training set size, (b) polynomial degree, (c) size of ridge penalty. Solid Red = training, dotted black = test.

	y
10	$(1,1)^T$
0	$(-1,-2)^T$
0	$(2,1)^T$
1	$(1,1)^T$
1	$(1,2)^T$
2	$(2,1)^T$

1. Compute $\hat{W}$ from the above data.

Shrinkage Methods

For vanilla linear regression model: $\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2$, we denote the solution as $\hat{\boldsymbol{\beta}}_{LS}$; for ridge regression model: $\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2$, we denote the solution as $\hat{\boldsymbol{\beta}}_{\lambda}^{Ridge}$; for Lasso model: $\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1$, we denote the solution as $\hat{\boldsymbol{\beta}}_{\lambda}^{Lasso}$; for Subset Selection model: $\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_0$, we denote the solution as $\hat{\boldsymbol{\beta}}_{\lambda}^{Subset}$, now please derive each $\hat{\boldsymbol{\beta}}$ given $\mathbf{y}, \mathbf{A} (s.t.\ \mathbf{A}^T\mathbf{A} = \mathbf{I})$, λ. Also, show the relationship of (each element in) $\hat{\boldsymbol{\beta}}_{\lambda}^{Ridge}, \hat{\boldsymbol{\beta}}_{\lambda}^{Lasso}, \hat{\boldsymbol{\beta}}_{\lambda}^{Subset}$ with (that in) $\hat{\boldsymbol{\beta}}_{LS}$ respectively. (you are encouraged to illustrate the relationship with figures appropriately.)

4 Linear, polynomial and ridge regression on Boston housing data (Matlab)

We will use linear regression to predict house prices, using the famous **Boston housing dataset**, described at `http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`. There are 506 records. We will use first 13 features as inputs, `x`, and the 14th feature, median house price, as the output y. All features are continuous, except feature 4, which is binary. However, we will treat this like any other continuous variable.

1. Load the `housing.data` file. We will use the first 300 cases for training and the remaining 206 cases for testing. However, the records seem to be sorted in some kind of order. To eliminate this, we will shuffle the data before splitting into a training/ test set. So we can all compare results, let use the following convention:

Listing 1: :

```
data = load('housing.data');
x = data(:, 1:13);
y = data(:, 14);
[n,d] = size(x);
% 14 features, median house price as the output
perm = randperm(n); % remove any possible ordering fx
x = x(perm,:); y = y(perm);
Ntrain = 300;
Xtrain = x(1:Ntrain,:); ytrain = y(1:Ntrain);
Xtest = x(Ntrain+1:end,:); ytest = y(Ntrain+1:end);
```

Linear Regression and its Extension

In the Boston housing dataset, there are 506 records. We will use first 13 features as inputs, x, and the 14th feature, median house prices, as the output y. All features are continuous, except feature 4, which is binary. However, we will treat this like any other continuous variable.

1. Load the housing data file. We will use the first 300 cases for training and the remaining 206 cases for testing. However, the records seem to be sorted in some kind of order. To eliminate this, we will shuffle the data before splitting into a training/test set. So we can all compare results, let use the following convention:

```
data = load('housing.data');
x = data(:, 1:13);
y = data(:, 14);
[n,d] = size(x);
```

2. Now extract the first N records of the training data, for $n \in \{25, 50, 75, 100, 150, 200, 300\}$. For each such training subset, standardize it, and fit a linear regression model using least squares. (Remember to include an offset term.) Then standardize the whole test set in the same way. Compute the mean squared error on the training subset and on the whole test set. Plot MSE versus training set size. You should get a plot like Figure 1(a). Turn in your plot and code. Explain why the test error decreases as n increases, and why the train error *increases* as n increases. Why do the curves eventually meet?

   As a debugging aid, here are the regression weights I get when I train on the first 25 cases (the first term is the offset, w_0):

```
seed = 2; rand('state',seed); randn('state', seed);
perm = randperm(n); % remove any possible ordering fx
x = x(perm,:); y = y(perm);
Ntrain = 300;
Xtrain = x(1:Ntrain,:); ytrain = y(1:Ntrain);
Xtest = x(Ntrain+1:end,:); ytest = y(Ntrain+1:end);
```

2. Now extract the first n records of the training data, for $n \in \{25, 50, 75, 100, 150, 200, 300\}$. For each such training subset, standardize it (you may use *zscore* function in Matlab), and fit a linear regression model using least squares. (Remember to include an offset term.) Then standardize the whole test set in the same way. Compute the mean squared error on the training subset and on the whole test set. Plot MSE versus training set size. You should get a plot like Figure 1(a). Turn in your plot and code. Explain why the test error decreases as n increases, and why the train error increases as n increases. Why do the curves eventually meet? As a debugging aid, here are the regression weights I get when I train on the first 25 cases (the first term is the offset, w0): $[26.11, -0.58, 3.02, \ldots, -0.21, -0.27, -1.16]$.

3. We will now replace the original features with an expanded set of features based on higher order terms. (We will ignore interaction terms.) For example, a quadratic expansion gives:

$$\begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{pmatrix} \rightarrow \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d} & x_{11}^2 & x_{12}^2 & \ldots & x_{1d}^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} & x_{n1}^2 & x_{n2}^2 & \ldots & x_{nd}^2 \end{pmatrix} \tag{2}$$

   The provided function degexpand(X,deg,addOnes) will replace each row of X with all powers up to degree deg. Use this function to train (by least squares) models with degrees 1 to 6. Use all the the training data. Plot the MSE on the training and test sets vs degree. You should get a plot like Figure 1(b). Turn in your plot and code. Explain why the test error decreases and then increases with degree, and why the train error decreases with degree.

4. Now we will use ridge regression to regularize the degree 6 polynomial. Fit models using ridge regression with the following values for λ:

$$lambdas = [0\ logspace(-10, 10, 10)]$$

   Use all the training data. Plot the MSE on the training and test sets vs $log_{10}(\lambda)$. You should get a plot like Figure 1(c). Turn in your plot and code. Explain why the test error goes down and then up with increasing λ, and why the train error goes up with increasing λ.

5. We turn to Lasso method with objective $\frac{1}{2}\|\mathbf{X}\beta - y\|^2 + \lambda\|\beta\|_1$ where λ varies in:

$$lambdas = [logspace(-10, 10, 10)]$$

   and we make use of all training samples with no feature expansion. Please plot the changes of β with λ changes.

Problem 1: Ridge Regression

To prove,

$$(A^TA + \lambda I_p)^{-1}A^T = A^T(AA^T + \lambda I_n)$$

Let,

$$A^TA + \lambda I_p = M$$

$$AA^T + \lambda I_n = N$$

$$X \cdot A^T = A^T \cdot A \cdot A^T + \lambda I_p A^T \quad \text{---------- (1)}$$

$$A^T Y = A^T A \cdot A^T + \lambda A^T \cdot I_n \quad \text{---------- (2)}$$

Subtracting (2) from (1)

$$X \cdot A^T - A^T \cdot Y = A^T \cdot A A^T + \lambda I_p A^T - A^T A \cdot A^T - \lambda A^T I_n$$

$$X \cdot A^T - A^T \cdot Y = \lambda I_p A^T - \lambda A^T I_n$$

$$X \cdot A^T - A^T \cdot Y = \lambda (I_p A^T - A^T I_n) \quad (\because \lambda \text{ is scalar})$$

Using Multiplicative Identity property of matrix

i.e $A \cdot I_n = I_m \cdot A$

we have, $X \cdot A^T - A^T \cdot Y = 0$

$$X \cdot A^T = A^T \cdot Y$$

$$A^T \cdot Y^{-1} = X^{-1} A^T$$

Substitute X & Y

$$[A^TA + \lambda I_p] A^T = A^T [AA^T + \lambda I_n]^{-1}$$

```
% Define the size and initialize variables
n = 100;
p_vals = [10, 100, 1000, 2000];
lambda = 0.1;
time_p = zeros(1, length(p_vals));
time_n = zeros(1, length(p_vals));

% Main loop
for idx = 1:numel(p_vals)
    p = p_vals(idx);
    Identity_n = eye(n);
    Identity_p = eye(p);
    RandomMatrix = randi([1, 100], n, p);

    % Calculate B
    tic;
    B = (RandomMatrix' * RandomMatrix + lambda * Identity_p) \ RandomMatrix';
    time_p(idx) = toc;

    % Calculate C
    tic;
    C = RandomMatrix' / (RandomMatrix * RandomMatrix' + lambda * Identity_n);
    time_n(idx) = toc;
end

% Convert to milliseconds
time_p = time_p * 1000;
time_n = time_n * 1000;

% Display results
fprintf("t_p: ");
```

```
t_p:
```

```
disp(time_p);
```

```
   29.7585    1.2663   48.6176  183.8293
```

```
fprintf("t_n: ");
```

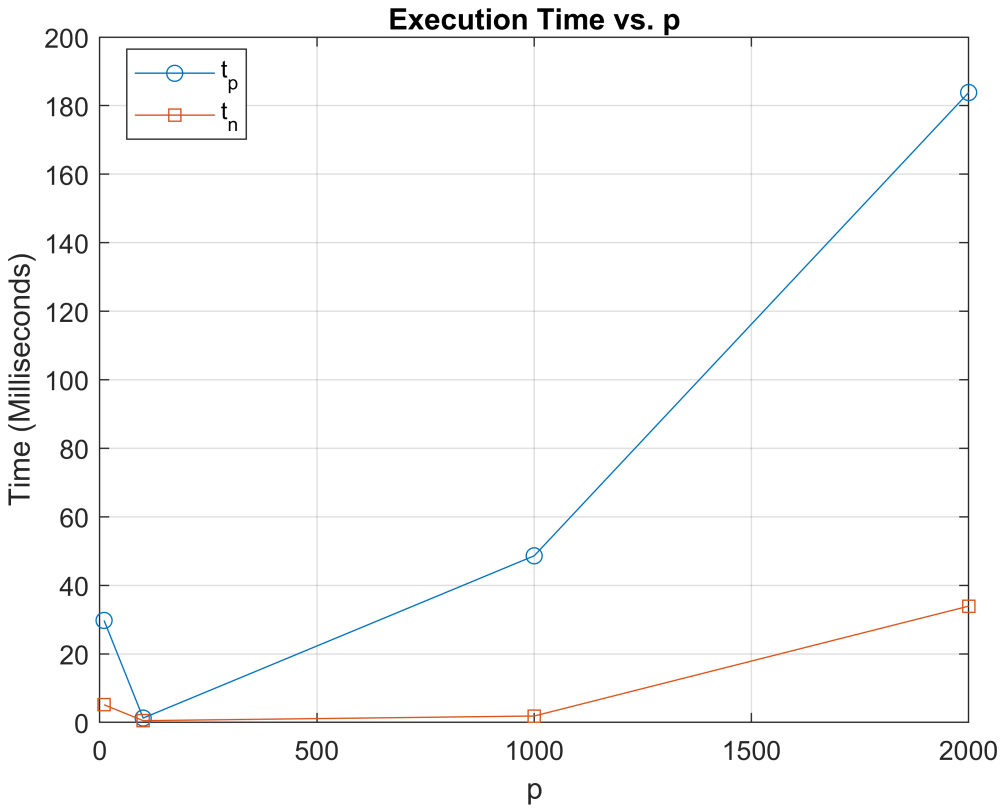
```
t_n:
```

```
disp(time_n);
```

```
    5.1527    0.5089    1.8675   33.9304
```

```

% Plotting
figure;
plot(p_vals, time_p, '-o', 'DisplayName', 't_p');
hold on;
plot(p_vals, time_n, '-s', 'DisplayName', 't_n');
```

```
xlabel('p');
ylabel('Time (Milliseconds)');
title('Execution Time vs. p');
legend('show', 'Location', 'Best');
grid on;
hold off;
```



Problem 2: Least Squares Extension

Ans 2. To solve,

$$\min_X \|AX + XC + Y\|_F^2 \quad \text{—— (1)}$$

given,

$$vec(AUC) = (C^T \otimes A)\, vec(U) \quad \text{— (2)}$$

Vectorizing the equation to solve (1),

$$\min_X \| vec(AX) + vec(XC) + vec(Y) \|_F^2$$

now, using the given property (2)

$$vec(AX) = vec(AXI) \quad \text{—— (3)}$$
$$= (I^T \otimes A)\, vec(X)$$

$$vec(XC) = vec(IXC) \quad \text{—— (4)}$$
$$= (C^T \otimes I)\, vec(X)$$

from (3), (4), we have

$$\min_X \| (I^T \otimes A)\, vec(X) + (C^T \otimes I)\, vec(X) - Y \|_F^2$$

$$\min_X \| [(I^T \otimes A) + (C^T \otimes I)]\, vec(X) - Y \|_F^2$$

putting $(I^T \otimes A) + (C^T \otimes I) = M$

we have

$$\min \| M \cdot vec(X) - Y \|_F^2$$

Finding solution using least squares approach,

$$\nabla f(\text{vec}(X)) = 2 M^T (M.\text{vec}(X) - Y)$$

Now equating $\nabla f(\text{vec}(X)) = 0$

$$2 M^T (M.\text{vec}(X) - Y) = 0$$

$$M^T.M.\text{vec}(X) = M^T.Y$$

$$\boxed{\text{vec}(X) = (M^T.M)^{-1} M^T.Y}$$

```
% Define the size of our example matrices
n = 2;

% Generate a random nxn matrix for A
A = randn(n, n);
disp('This is A: ')
```

```
This is A:
```

```
disp(A);
```

```
    0.6007   -0.6576
    1.6880    1.7340
```

```
% Generate a random nxn matrix for X_star (optimal X)
X_star = randn(n, n);
disp('This is X_Star: ')
```

```
This is X_Star:
```

```
disp(X_star);
```

```
    0.8961    0.4519
    0.4383   -1.1793
```

```
% Generate a random nxn matrix for C
C = randn(n, n);
disp('This is C: ')
```

```
This is C:
```

```
disp(C);
```

```
    0.5162   -0.7711
   -1.4565    0.5299
```

```
% Compute the matrix Y based on the relation: Y = AX* + X*C
Y = A * X_star + X_star * C;

% Create an identity matrix of size nxn
I = eye(n);

% Construct matrix M using the Kronecker product
% This is used for reshaping the matrix multiplication into vector form
M = kron(I, A) + kron(C.', I);

% Use matrix division (or the inverse of M) to compute vecX from Y
vecX = M \ Y(:);

% Reshape vecX back into matrix form to get X
X = reshape(vecX, [n, n]);
```

```
% Display the computed X and M matrices
disp('This is X: ')
```

```
This is X:
```

```
disp(X);
```

```
    0.8961    0.4519
    0.4383   -1.1793
```

```
disp('This is M: ')
```

```
This is M:
```

```
disp(M);
```

```
    1.1169   -0.6576   -1.4565         0
    1.6880    2.2501         0   -1.4565
   -0.7711         0    1.1307   -0.6576
         0   -0.7711    1.6880    2.2639
```

On comparing vec(X) with vec(X*), we find that they both are equivalent.

Problem 3: Shrinkage Methods

Vanilla linear Regression →

$$f(\beta) = \min_{\beta} \| y - A\beta \|_2^2$$

Using least squared approach to minimize $f(\beta)$

put $\nabla f(\beta) = 0$

$$\nabla f(\beta) = -2A^T.(y - A\hat{\beta}_{LS})$$

$$-2A^T.(y - A\hat{\beta}_{LS}) = 0$$

$$A^T.A\hat{\beta}_{LS} = A^T.y$$

$$\hat{\beta}_{LS} = (A^T.A)^{-1}A^T.y$$

Given $A^T.A = I$

Therefore $\boxed{\hat{\beta}_{LS} = I.A^T.y}$

Ridge Linear Regression →

$$f(\beta) = \min_{\beta} \| y - A\beta \|_2^2 + \lambda \|\beta\|_2^2$$

Using least squares approach to minimize $f(\beta)$

put $\nabla f(\beta) = 0$

$$\nabla f(\beta) = -2 \cdot A^T (y - A\beta) + 2\lambda\beta$$

$$A^T(A\beta - y) + \lambda\beta = 0$$

$$A^T \cdot A \cdot \beta - A^T y + \lambda\beta = 0$$

$$(A^T \cdot A + \lambda I)\beta = A^T \cdot y$$

$$\hat{\beta}_{Ridge} = (A^T \cdot A + \lambda I)^{-1} \cdot A^T \cdot y$$

Given, $A^T \cdot A = I$

$$\hat{\beta}_{ridge} = ((1+\lambda)I)^{-1} \cdot A^T \cdot y$$

$$\boxed{\hat{\beta}_{ridge} = \frac{1}{1+\lambda} \cdot I \cdot A^T \cdot y}$$

Lasso linear Regression →

$$f(\beta) = \min_{\beta} \|y - A\beta\|_2^2 + \lambda \|\beta\|_1$$

Given, $A^T \cdot A = I$

Therefore, A is an orthogonal matrix

For lasso problem, the solution $\hat{\beta}_{Lasso}$ satisfies the subgradient optimality condition,

$$\nabla f(\beta) + \lambda \, sgn(\hat{\beta}_{Lasso}) = 0 \quad \text{— (1)}$$

$$f(\beta) = f_L(\beta) = \min_{\beta} \frac{1}{2} \|A\beta - y\|_2^2$$

Using optimality condition for lasso problem, the gradient of loss function is,

$$\nabla f_L(\beta) = A'(A\beta - y)$$
$$= A' \cdot A \cdot \beta - A'y$$

given $A' \cdot A \to I$

$$\nabla f_L(\beta) = \beta - A'y$$

Getting back to equation (1)

$$\beta - A'y + \lambda \, sgn(\beta) = 0$$

$$\hat{\beta}_{Lasso} = A'y - \lambda \, sgn(\hat{\beta}_{Lasso})$$

Now, the soft-thresholding operator $S_\lambda(z)$ is defined as

$$S_\lambda(z) = sign(z) \max(|z| - \lambda, 0)$$

Given our optimality condition, each component $\hat{\beta}_{Lasso}(i)$ of vector $\hat{\beta}_{Lasso}$ can be found by applying the soft thresholding operation to the vector $A^T y$

$$\boxed{\hat{\beta}_{Lasso}(i) = S_\lambda((A^T y)_i)}$$

Subset selection Model →

$$f(\beta) = \min_{\beta} \frac{1}{2} \|y - A\beta\|_2^2 + \lambda \|\beta\|_0$$

$\|\beta\|_0$ norm counts the number of non-zero entries in β. It is non-convex and combinatorial which makes finding optimal solution β_λ^{subset} challenging.

Given, $A^T \cdot A = I$

Therefore, A is an orthogonal matrix and the correlation between any 2 columns of A is zero.

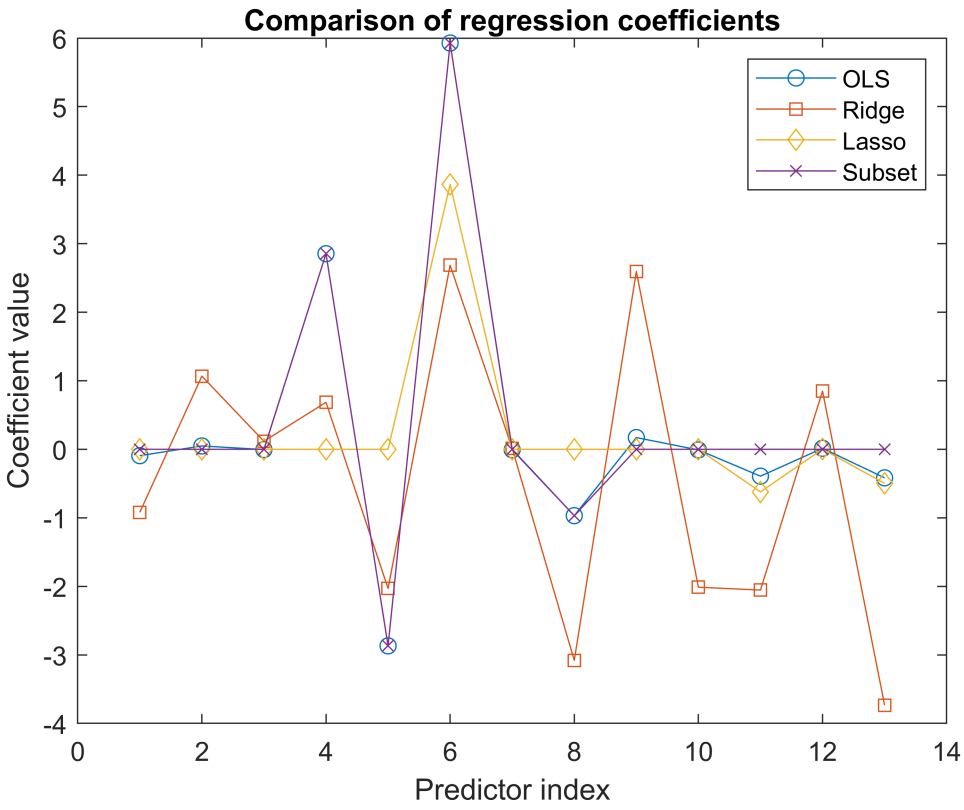
Therefore, the correlation between y and the j-th column of A is just, $A_j^T \cdot y$

So, for each predictor j:

compute $c_j = A_j^T \cdot y$

$$\hat{\beta}_\lambda^{subset} = \begin{cases} \beta_j = 0 & \text{if } |c_j| < \lambda \\ \beta_j = c_j & \text{if otherwise} \end{cases}$$

```
% Load the Boston housing dataset
data = load(['C:\Users\param\OneDrive - Clemson University' ...
    '\Desktop\CU\Fall23\Advance Machine Learning\HW1/housing.data']);

% Extract features and target variable
x = data(:, 1:13);
y = data(:,14);

% Vanilla OLS
x_ols = x\y; % Using the backslash operator for least squares

% Ridge regression
lambda_ridge = 1; % Example regularization parameter for Ridge
x_ridge = ridge(y, x, lambda_ridge);

% Lasso
lambda_lasso = 1; % Example regularization parameter for Lasso
[x_lasso, fitinfo] = lasso(x, y, 'Lambda', lambda_lasso);

% Subset (Simple implementation using a thresholding on OLS coefficients)
threshold = 0.5; % Example threshold
x_subset = x_ols;
x_subset(abs(x_ols) < threshold) = 0;

% Plot
figure;
plot(1:13, x_ols, 'o-', 'DisplayName', 'OLS');
hold on;
plot(1:13, x_ridge, 's-', 'DisplayName', 'Ridge');
plot(1:13, x_lasso, 'd-', 'DisplayName', 'Lasso');
plot(1:13, x_subset, 'x-', 'DisplayName', 'Subset');
xlabel('Predictor index');
ylabel('Coefficient value');
legend();
title('Comparison of regression coefficients');
hold off;
```


Comparison of regression coefficients
OLS
Ridge
Lasso
Subset
Coefficient value
Predictor index
6
5
4
3
2
1
0
-1
-2
-3
-4
0
2
4
6
8
10
12
14

Problem 4: Linear Regression and its Extension

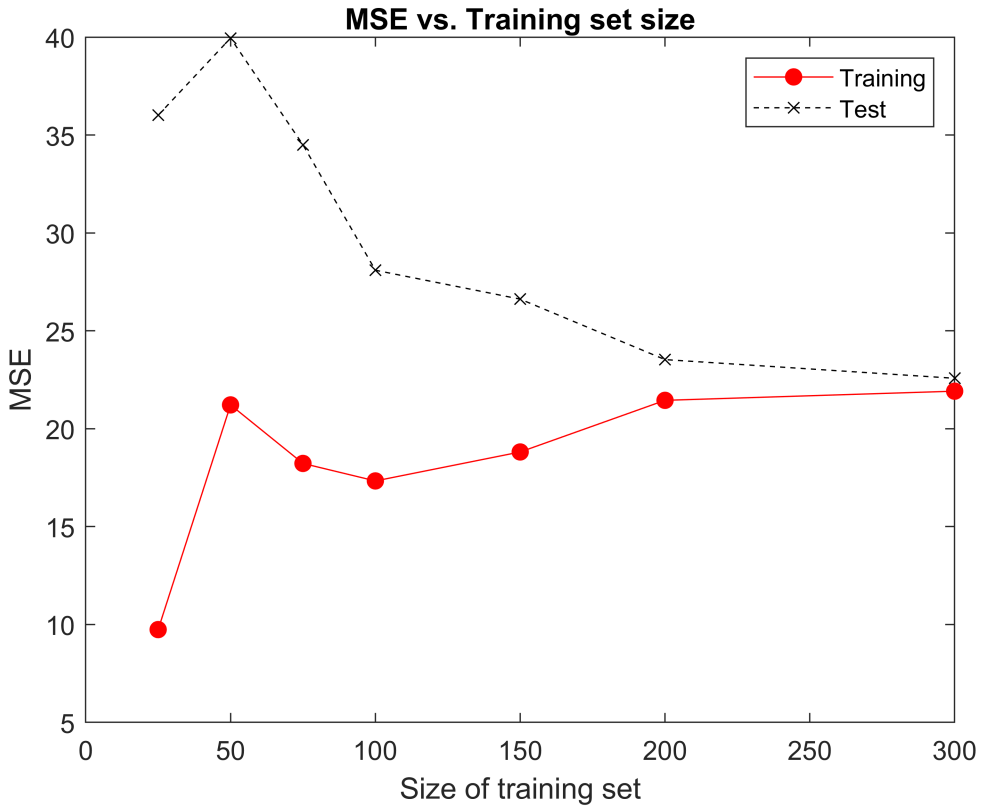
Ques 4 Part 1

```
% Load the Boston housing dataset
data = load(['C:\Users\param\OneDrive - Clemson University' ...
    '\Desktop\CU\Fall23\Advance Machine Learning\HW1/housing.data']);

% Extract features and target variable
x = data(:, 1:13);
y = data(:,14);

% Get the number of records (n) and features (d)
[n, d] = size(x);

% Set seed for reproducibility
seed = 2;
rand('state', seed);
randn('state', seed);

% Shuffle the dataset
perm = randperm(n);
x = x(perm, :);
y = y(perm);

% Split into training and test sets
Ntrain = 300;
Xtrain = x(1:Ntrain, :);
ytrain = y(1:Ntrain);
Xtest = x(Ntrain+1:end, :);
ytest = y(Ntrain+1:end);
```

Ques4 Part 2

```
subset_sizes = [25, 50, 75, 100, 150, 200, 300];
num_sizes = numel(subset_sizes);
train_errors = zeros(num_sizes, 1);
test_errors = zeros(num_sizes, 1);

for i=1:num_sizes
    n = subset_sizes(i);

    % Extract first n records from training data
    X_sub = Xtrain(1:n, :);
    y_sub = ytrain(1:n);

    % Standardize the data
    [X_sub_standardized, mu, sigma] = zscore(X_sub);
    Xtest_standardized = (Xtest - mu) ./ sigma;

    % Add offset term (bias/intercept)
    X_sub_standardized = [ones(n, 1), X_sub_standardized];
```

```
    Xtest_standardized = [ones(size(Xtest,1), 1), Xtest_standardized];

    % Fit linear regression model using least squares
    w = X_sub_standardized \ y_sub;

    % Compute MSE for training subset and test set
    train_errors(i) = mean((X_sub_standardized * w - y_sub).^2);
    test_errors(i) = mean((Xtest_standardized * w - ytest).^2);
end

% Plot the results
figure;
plot(subset_sizes, train_errors, '-ro', 'MarkerFaceColor', 'r');
hold on;
plot(subset_sizes, test_errors, '--xk'); % Black line with crosses for test error
xlabel('Size of training set');
ylabel('MSE');
legend('Training', 'Test');
title('MSE vs. Training set size');
```



Why does the test error decrease as n increases?

As the training set size increases, the model gets more data to learn from, which generally results in better generalization on unseen data, thus decreasing the test error.

Why does the train error increase as n increases?

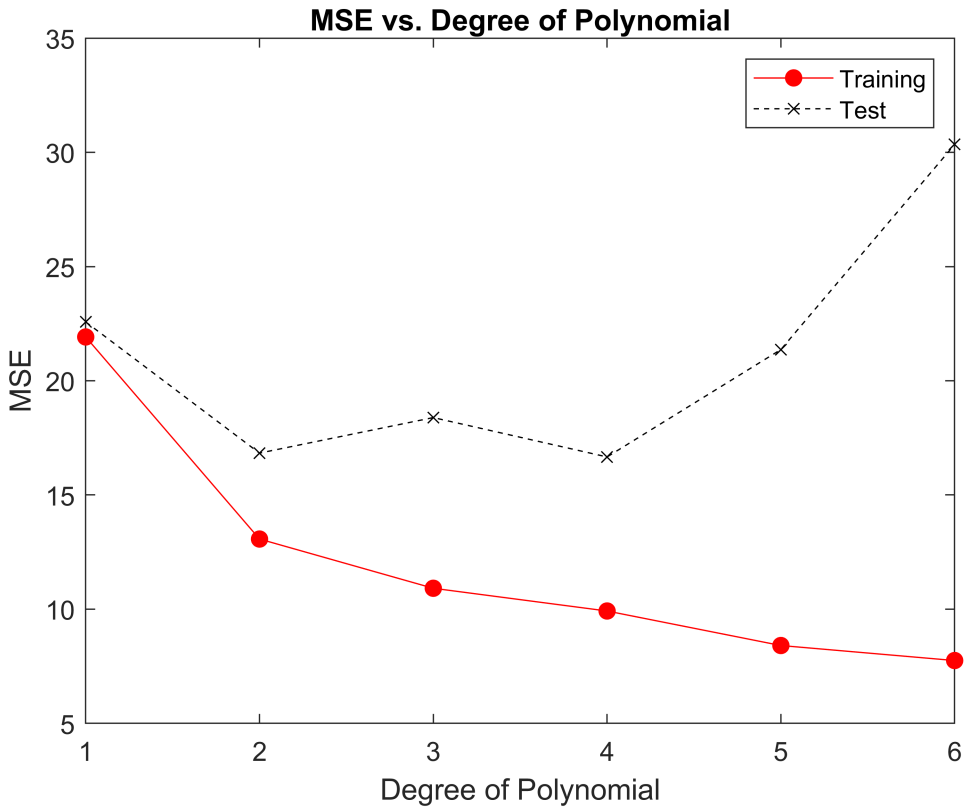
When the training set is small, the model can fit the training data more closely, sometimes even overfitting it. As the training size increases, the data may become more diverse and harder to fit perfectly, leading to an increase in training error.

Why do the curves eventually meet?

As the training set size increases and approaches the full dataset size, the model sees almost all of the available data during training. Consequently, its performance on the training set and the test set becomes more similar, leading to convergence of the two error curves.

Ques 4 Part 3

```
max_degree = 6;
train_errors = zeros(max_degree, 1);
test_errors = zeros(max_degree, 1);

for deg = 1:max_degree
    % Expand the feature set
    Xtrain_exp = degexpand(Xtrain, deg, 1);
    Xtest_exp = degexpand(Xtest, deg, 1);

    % Standardize the data
    [Xtrain_exp(:, 2:end), mu2, sigma2] = zscore(Xtrain_exp(:, 2:end));
    Xtest_exp(:, 2:end) = (Xtest_exp(:, 2:end) - mu2) ./ sigma2;

    % Train a linear regression model using least squares
    w = pinv(Xtrain_exp' * Xtrain_exp) * Xtrain_exp' * ytrain;

    % Compute the training error
    train_pred = Xtrain_exp * w;
    train_errors(deg) = mean((ytrain - train_pred).^2);

    % Compute the test error
    test_pred = Xtest_exp * w;
    test_errors(deg) = mean((ytest - test_pred).^2);
end

% Plot the errors
figure;
plot(1:max_degree, train_errors, '-ro', 'MarkerFaceColor', 'r');
hold on;
plot(1:max_degree, test_errors, '--xk');
xlabel('Degree of Polynomial');
ylabel('MSE');
legend('Training', 'Test');
title('MSE vs. Degree of Polynomial');
```



Training Error Decreasing:

As you increase the polynomial degree, the model has more flexibility and can fit the training data more closely, which reduces the training error.

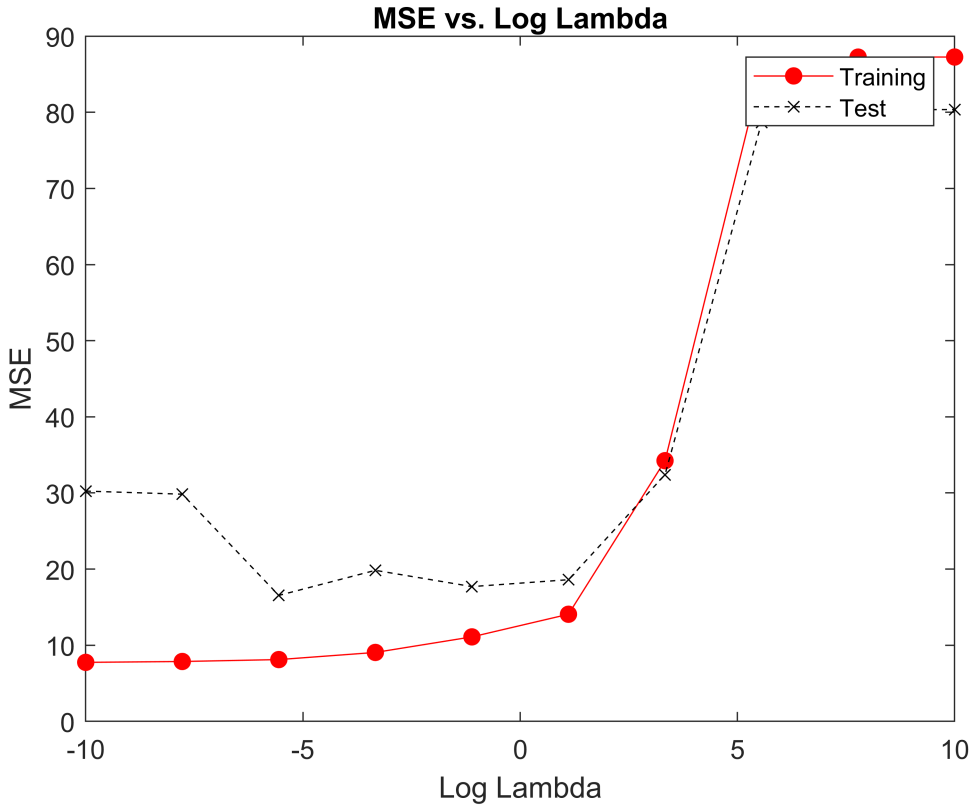
Test Error Behavior:

Initially, the increased flexibility of the model helps to capture the underlying pattern in the data, reducing the test error. However, after a certain point, the model starts to overfit. This means it's fitting the training data too closely, including its noise and outliers. This reduces its ability to generalize to new data, leading to an increase in test error.

Ques 4 Part 4

```
% Expand the feature set
Xtrain_exp = degexpand(Xtrain, 6, 1);
Xtest_exp = degexpand(Xtest, 6, 1);

% Standardize the data
[Xtrain_exp(:, 2:end), mu2, sigma2] = zscore(Xtrain_exp(:, 2:end));
Xtest_exp(:, 2:end) = (Xtest_exp(:, 2:end) - mu2) ./ sigma2;

lambdas = [0 logspace(-10, 10, 10)];

%I = eye(size(Xtrain_exp(:, 2:end), 2));
```

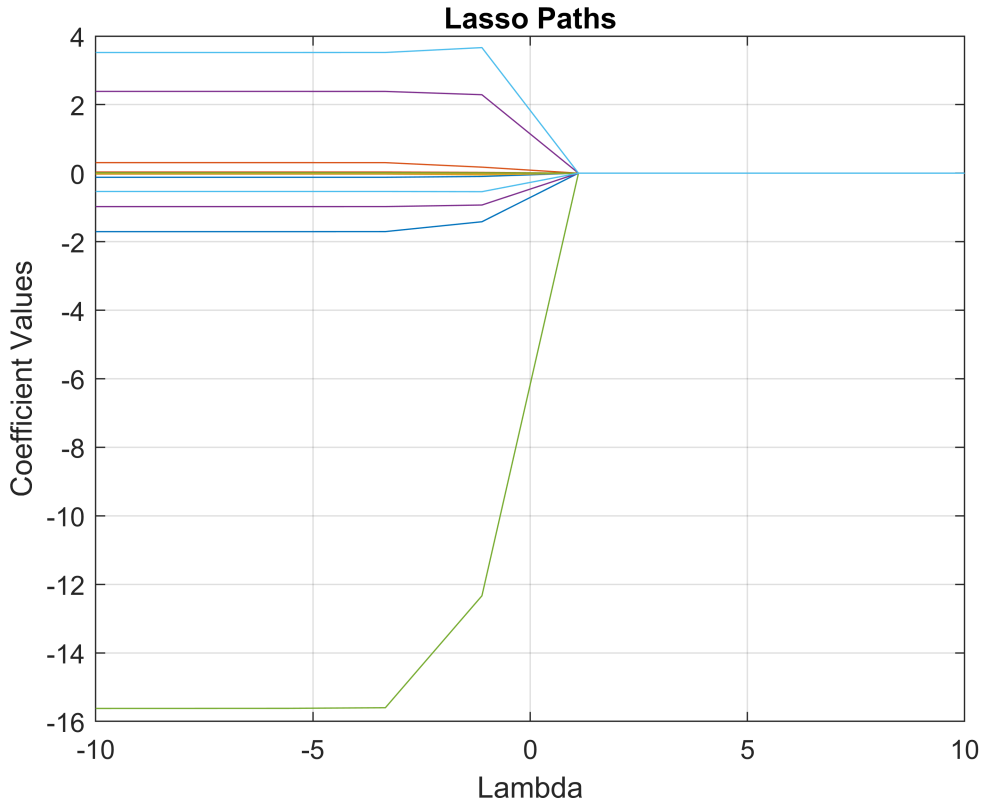
```
train_errors = zeros(size(lambdas,1), 1);
test_errors = zeros(size(lambdas,1), 1);
i = 1;

for k = lambdas
    % Compute the X'X term
    XtX = Xtrain_exp' * Xtrain_exp;

    % Construct the regularization term.
    I = eye(size(XtX, 1));
    I = [zeros(size(XtX, 1), 1), I(:, 2:end)];
    regularization = k * I;

    % Add the regularization to XtX
    ridge_matrix = XtX + regularization;

    % Compute the weight vector w
    w = pinv(ridge_matrix) * Xtrain_exp' * ytrain;

    % Compute the training error
    train_pred = Xtrain_exp * w;
    train_errors(i) = mean((ytrain - train_pred).^2);

    % Compute the test error
    test_pred = Xtest_exp * w;
    test_errors(i) = mean((ytest - test_pred).^2);

    i = i + 1;
end

% Plot the errors
figure;

plot(log10(lambdas), train_errors, '-ro', 'MarkerFaceColor', 'r');
hold on;
plot(log10(lambdas), test_errors, '--xk');
%axis([log10(lambdas(2)), log10(max(lambdas)), 0, 90]);
xlabel('Log Lambda');
ylabel('MSE');
legend('Training', 'Test');
title('MSE vs. Log Lambda');
```



Initially, as λ increases from a small value, the added regularization helps combat overfitting, so test error goes down. However, after a certain point, increasing λ results in underfitting, causing both training and test error to rise.

Ques 4 Part 5

```
% Define the range of lambda values
lambdas = logspace(-10, 10, 10);

% Use MATLAB's built-in lasso function
[B, FitInfo] = lasso(Xtrain, ytrain, 'Lambda', lambdas);

% Plot
figure;
plot(log10(FitInfo.Lambda), B);
xlabel('Lambda');
ylabel('Coefficient Values');
title('Lasso Paths');
grid on;
```


Lasso Paths
Coefficient Values
Lambda
4
2
0
-2
-4
-6
-8
-10
-12
-14
-16
-10
-5
0
5
10