The report of Project-1

张奕朗 16307130242

1 Linear Regression and Nonlinear Bases

1.1 Adding a Bias Variable

code files: leastSquaresBias.m and AddingABiasVariable.m

(1) Algorithm Skims

Least-squares linear regression minimizes the RSS, and leads to a closed-form expression for the estimated value w. The following, we consider adding a bias β to improve our model to $y_i = w^T x_i + \beta$. In one-dimension case, $\mathrm{RSS} = \sum_i (y_i - \hat{y_i})^2 = \sum_i (y_i - w x_i - \beta)^2$. To minimize RSS, let $\begin{cases} \frac{\partial RSS}{\partial w} = 0 \\ \frac{\partial RSS}{\partial \beta} = 0 \end{cases}$, we can get

$$\begin{cases} \hat{w} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = \frac{\sum_i x_i y_i - \frac{1}{n}(\sum_i x_i)(\sum_i y_i)}{\sum_i x_i{}^2 - \frac{1}{n}(\sum_i x_i)^2} \\ \hat{\beta} = \bar{y} - \hat{w} * \bar{x} \end{cases}.$$

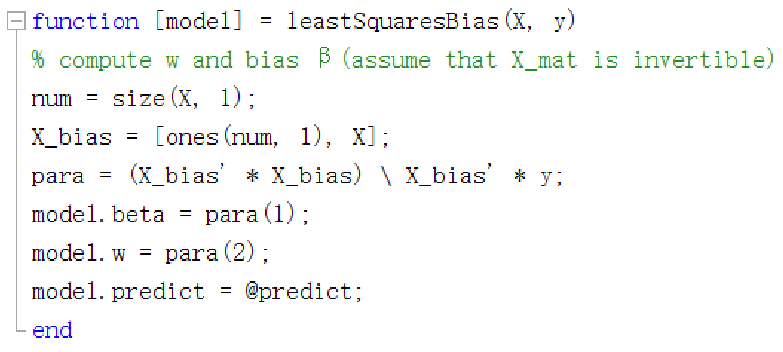
Transform them into matrix form to speed up computing :

$$\mathrm{X_{bias}} * \mathrm{w}_{bias} = y, \text{ where } \mathrm{X_{bias}} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ & \dots \\ 1 & x_n \end{bmatrix}, \ \mathrm{w_{bias}} = \begin{bmatrix} \beta \\ w \end{bmatrix}.$$

$$\mathrm{RSS} = \|y - \mathrm{X_{bias}} \mathrm{w}_{bias}\|_2^2$$

Let $\frac{\partial \mathrm{RSS}}{\partial \mathrm{w}_{bias}} = 2X_{bias}^T(\mathrm{X_{bias}} \mathrm{w}_{bias} - \mathrm{y}) = 0$, we get $\mathrm{w_{bias}} = (\mathrm{X}_{bias}^T \mathrm{X_{bias}})^{-1} \mathrm{X}_{bias}^T y$ as the least square solution.

(2) Critical Codes



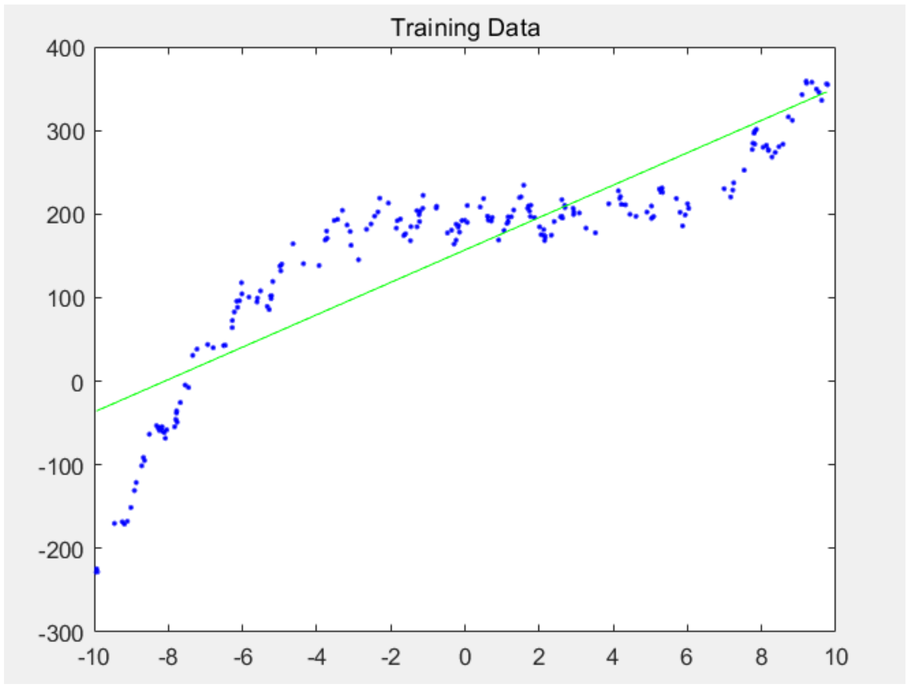
```
function [model] = leastSquaresBias(X, y)
% compute w and bias β (assume that X_mat is invertible)
num = size(X, 1);
X_bias = [ones(num, 1), X];
para = (X_bias' * X_bias) \ X_bias' * y;
model.beta = para(1);
model.w = para(2);
model.predict = @predict;
end
```

In the codes above, X_bias and para respectively represent for $\mathrm{X_{bias}}$ and w_{bias}.

(3) Experimental Analysis

In the improved model which has a bias term, we get updated training error = 3551.35 and test error = 3393.87. Compared to training error = 28122.82 and test error = 28298.97 in ‘example_basis’, it’s a huge improvement.

Updated plot: The green line is the predicted y for $\mathrm{x} \in [-10,10]$, and the blue points are real data in training set. We can see from the plot that when $\mathrm{x} = 0$, $\mathrm{y} \approx 160$, a huge deviation to 0, which manifest that adding a bias is reasonable to some degree.



(4) Discussion of Proposed Method

We can see from the plot that x and y are not strictly linear, but more likely a polynomial function.

1.2 Polynomial Basis

code files: leastSquaresBasis.m and PolynomialBasis.m

(1) Algorithm Skims

In the circumstance of polynomial $y = w_0 + w_1x + w_2x^2 + \cdots + w_{deg}x^{deg}$.Similarly to the bias case, we have

$$\mathrm{X_{ploy}}t = y, \text{ where } \mathrm{X_{poly}} = \begin{bmatrix} 1 & x_1 & {x_1}^2 & \dots & {x_1}^{deg} \\ 1 & x_2 & {x_2}^2 & \dots & {x_2}^{deg} \\ & & \dots & & \\ 1 & x_n & {x_n}^2 & \dots & {x_n}^{deg} \end{bmatrix}, \ \mathrm{t} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_{deg} \end{bmatrix}.$$

In order to minimize RSS (i.e. minimize the distance between y and $\mathrm{X_{ploy}}t$ in higher dimensional space), $\mathrm{t} = (\mathrm{X}_{poly}^T\mathrm{X_{poly}})^{-1}\mathrm{X}_{poly}^T y$.

(2) Critical Codes

```
function [model] = leastSquaresBasis(x, y, deg)
% use least squares to compute the polynomial coefficient t
%(assume that X_poly'*X_poly is invertible)
x_poly = zeros(size(x, 1), deg+1);
for i = 0:deg
    x_poly(:, i+1) = x.^i;
end
t = (x_poly' * x_poly) \ x_poly' * y;
model.t = t;
model.deg = deg;
model.predict = @predict;
end
```

In the codes above, I use a for-loop to generate the matrix $\mathrm{X}_{\mathrm{ploy}}$, then compute the coefficient vector $\mathbf{t}$.

(3) Experimental Analysis

The results are as follows:

for deg = 0:

Training error = 15480.52, test error = 14390.76

for deg = 1:

Training error = 3551.35, test error = 3393.87

for deg = 2:

Training error = 2167.99, test error = 2480.73

for deg = 3:

Training error = 252.05, test error = 242.80

for deg = 4:

Training error = 251.46, test error = 242.13

for deg = 5:

Training error = 251.14, test error = 239.54

for deg = 6:

Training error = 248.58, test error = 246.01

for deg = 7:

Training error = 247.01, test error = 242.89

for deg = 8:

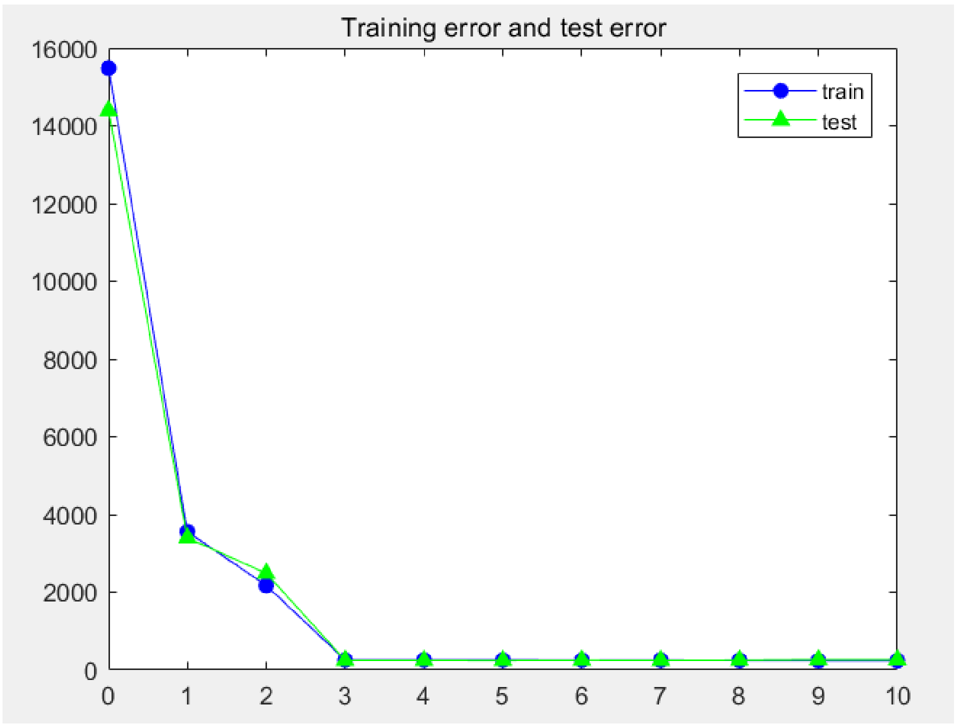
Training error = 241.31, test error = 245.97

for deg = 9:

Training error = 235.76, test error = 259.30

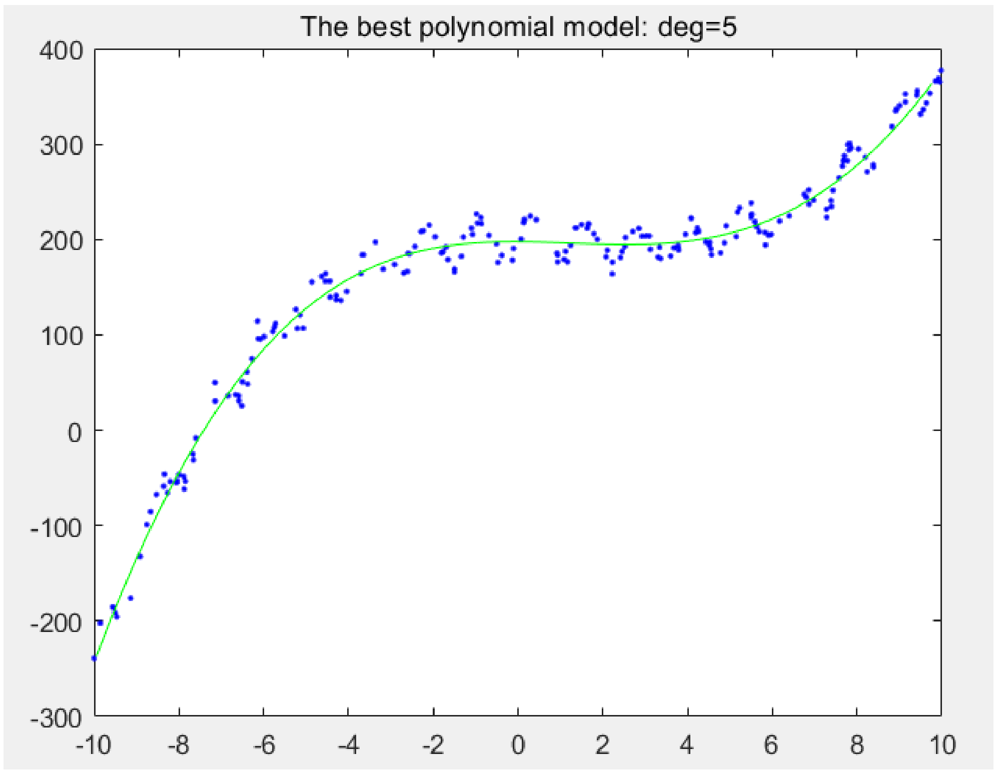
for deg = 10:

Training error = 235.07, test error = 256.30



With the increasing of degree from 0 to 10, training error gets lower and lower, but test error gets lower until deg=5, then goes higher slightly, which manifests that overfitting occurred due to the increasing of complexity.

Choose the model with the lowest test error as the best model and plot it: The green line is the predicted y for $x \in [-10,10]$, and the blue points are real data in test set.



(4) Discussion of Proposed Method

Although the fitting is so good, there is still something need to notice: when the degree goes to 9 and 10, there is a warning by Matlab, 'the matrix approaches singular values or scaling errors. The result may not be accurate. RCOND = 3.582888e-18.' The reason for it is overflowing when the degree of x gets too high, then underflowing when computing the pseudo-inverse of matrix X_{ploy}.

A practical solution to this problem is standardization before computing, which makes $x' \sim N(0,1)$.

2 Regularization

code files: RidgeRegression.m, CrossValidation.m and ridge.m

2.2 Load and standardize the data

Here are some constants in the code below, num_vars, num_patients and num_train are respectively the number of variables (9), patients (97) and training data (50).

Use 'textscan' to load data from 'prostate.data.txt', then randomly shuffle the order and split them into training data (50 patients) and test data (47 patients).

Next, do standardization by the formula: $x_{ij} = \frac{x_{ij}-\overline{x_j}}{\sigma_j}$, where $\overline{x_j} = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$ is the empirical mean and $\sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x_j})^2}$ is the empirical standard variance. And $y_i =$

$y_i - \overline{y_i}$, where $\overline{y_i} = \frac{1}{n}\sum_{i=1}^{n} y_i$. Now, $x_j(j = 1,2,\dots,8)$ obeys the standard normal distribution and y have zero mean.

```
% Load data
filename = '../data/prostate.data.txt';
fileID = fopen(filename);
names = textscan(fileID, '%s', num_vars);
data = textscan(fileID, '%f');
data = reshape(data{1}, num_vars, num_patients)';
X = data(1:num_patients, 1:num_vars-1);
y = data(1:num_patients, num_vars);

% Use mask index to shuffle the orders of patients
mask = randperm(num_patients)';
X = X(mask, :);
y = y(mask, :);

% Split it into training data and test data
X_train = X(1:num_train, :);
y_train = y(1:num_train, :);
X_test = X(num_train+1:num_patients, :);
y_test = y(num_train+1:num_patients, :);

% standardization
X_mean = mean(X_train);
y_mean = mean(y_train);
X_std = sqrt(mean((X_train - X_mean).^2));
X_train = (X_train - X_mean) ./ X_std;
y_train = y_train - y_mean;
X_test = (X_test - X_mean) ./ X_std;
y_test = y_test - y_mean;
```

There is a noticeable thing after doing standardization. x_j and y now both have zero mean, so $\mathrm{E}(x_j) = \mathrm{E}(y) = 0, \mathrm{E}(\beta_0) = \mathrm{E}(y) - \sum_{j=1}^{8}\theta_j x_j = 0$, where β_0 is the bias term which is expected to be 0. (I'll show the value of β_0 in the next section)

2.3 Ridge regression

(1) Algorithm Skims

Ridge regression add a L-2 regularization term to RSS as the updated loss function $\mathrm{L} = \|y - X\theta\|_2^2 + \delta^2\|\theta\|_2^2$, where δ^2 is the regularization parameter, controlling the strength of L-2 regularization. Our aim is to minimize the loss function, so let $\frac{\partial \mathrm{L}}{\partial \theta} =$

$2\mathrm{X}^T(\mathrm{X}\theta - \mathrm{y}) + 2\delta^2\theta = 0$, we get $\theta = (\mathrm{X}^T\mathrm{X} + \delta^2\mathrm{I})^{-1}\mathrm{X}^T y$, where I is an 8-by-8 unit matrix.

Notice that $\mathrm{v}^T\mathrm{X}^T\mathrm{X}\mathrm{v} = \|\mathrm{X}\mathrm{v}\|_2^2 \geq 0,$ so $\mathrm{X}^T\mathrm{X}$ is positive semidefinite. And $\delta^2 > 0,$ we can infer that $\mathrm{X}^T\mathrm{X} + \delta^2\mathrm{I}$ is a positive definite matrix which is invertible.

(2) Critical Codes

1. Function ridge:

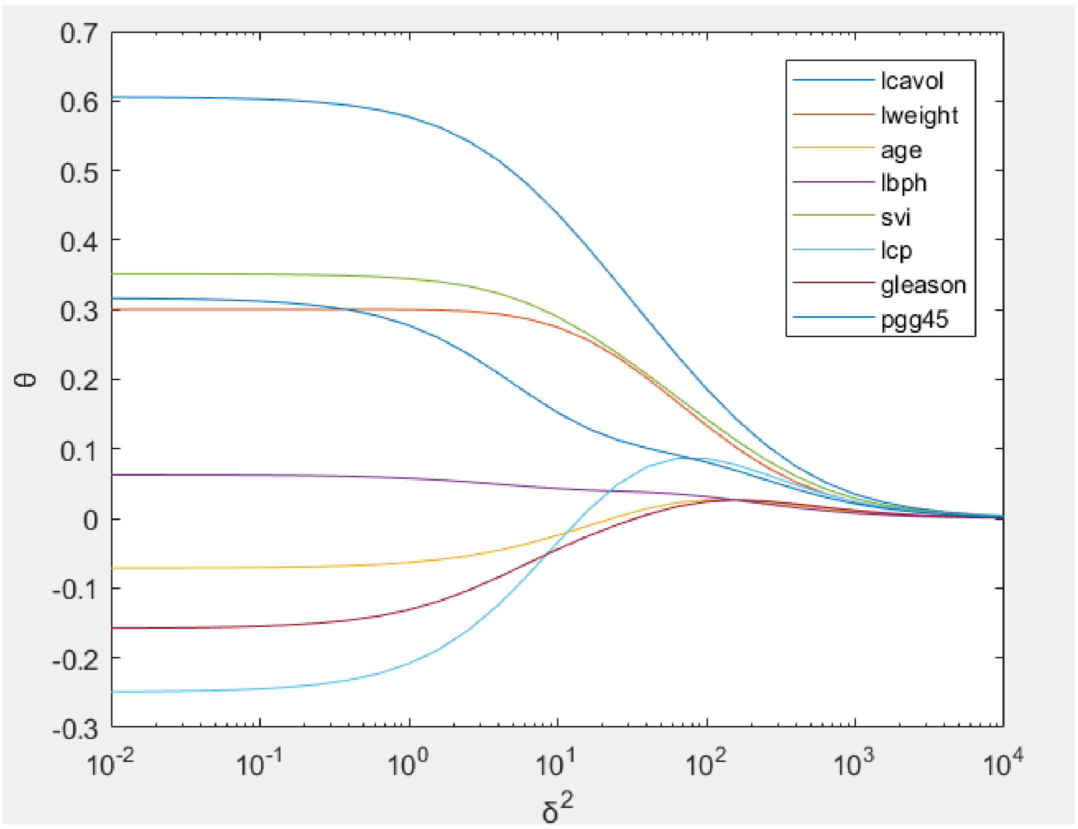
```
function [theta] = ridge(X, y, d2)
% compute theta (A' * A + d2 * I must be a invertible matrix,
% details in the report)
num_vars = size(X, 2);
theta = (X' * X + d2 * eye(num_vars)) \ X' * y;
end
```

Then generate δ^2 range from 10^{-2} to 10^4, step by $10^{0.2}$, and use the function above to compute corresponding parameter θ.

```
% Compute the ridge regression solutions for a range of regularizers (δ^2)
delta2 = zeros(num_delta, 1);
theta = zeros(num_delta, num_vars - 1);
for i = 1:num_delta
    ldelta2 = -2 + 0.2 * (i - 1);  % lg(δ^2) range from -2 to 4
    delta2(i) = 10 ^ (ldelta2);  % δ^2 range from 10^(-2) to 10^(4)
    theta(i,:) = ridge(X_train, y_train, delta2(i));
end
```

Plot δ^2 and corresponding θ:

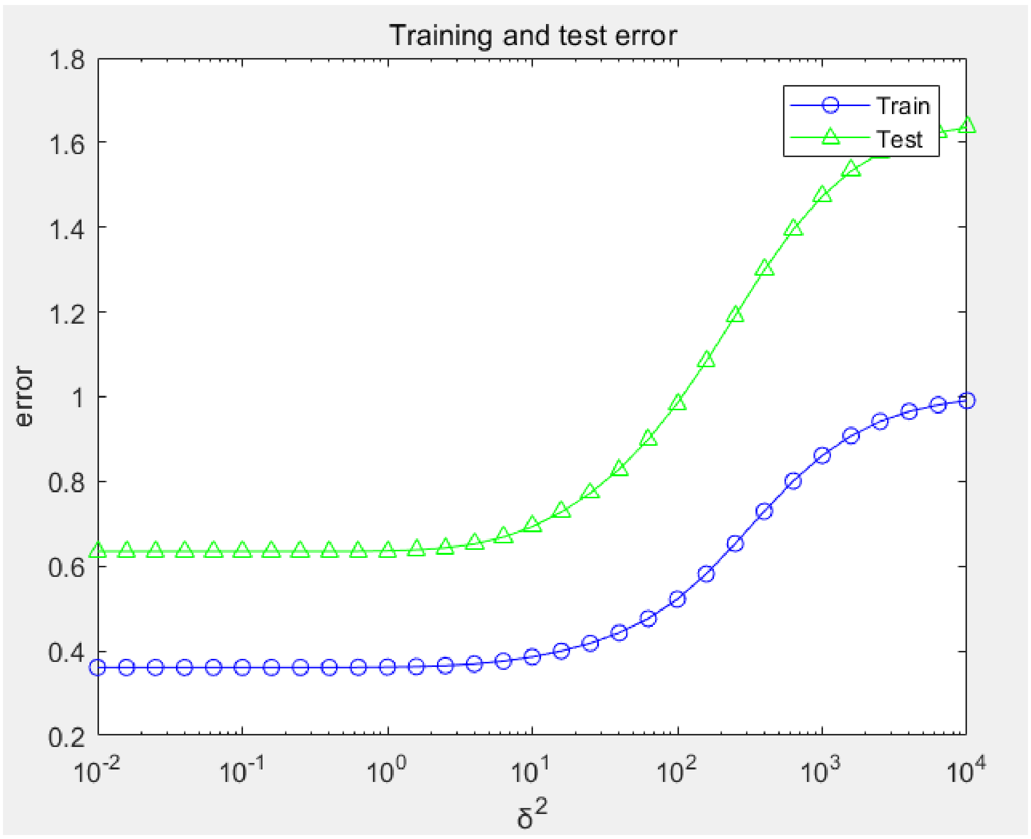
```
% Plot theta and log10(delta2)
semilogx(delta2, theta);
legend(names{1}{1:num_vars - 1});
xlabel('δ^2');
ylabel('θ');
```



2. For each computed value of θ, compute MSE on training and test set. (See detailed values in variable error_train and error_test in RidgeRegression.m.)

```
% Compute MSE on training and test set
error_train = mean((yhat_train - y_train).^2);
error_test = mean((yhat_test - y_test).^2);
```

Plot them as follows:



3. Use 5-folds cross validation to choose the value of δ^2 with the lowest test error.

There are totally 97 patients' data, in order to split them into 5 folds evenly, I abandoned 1 patients' data randomly .

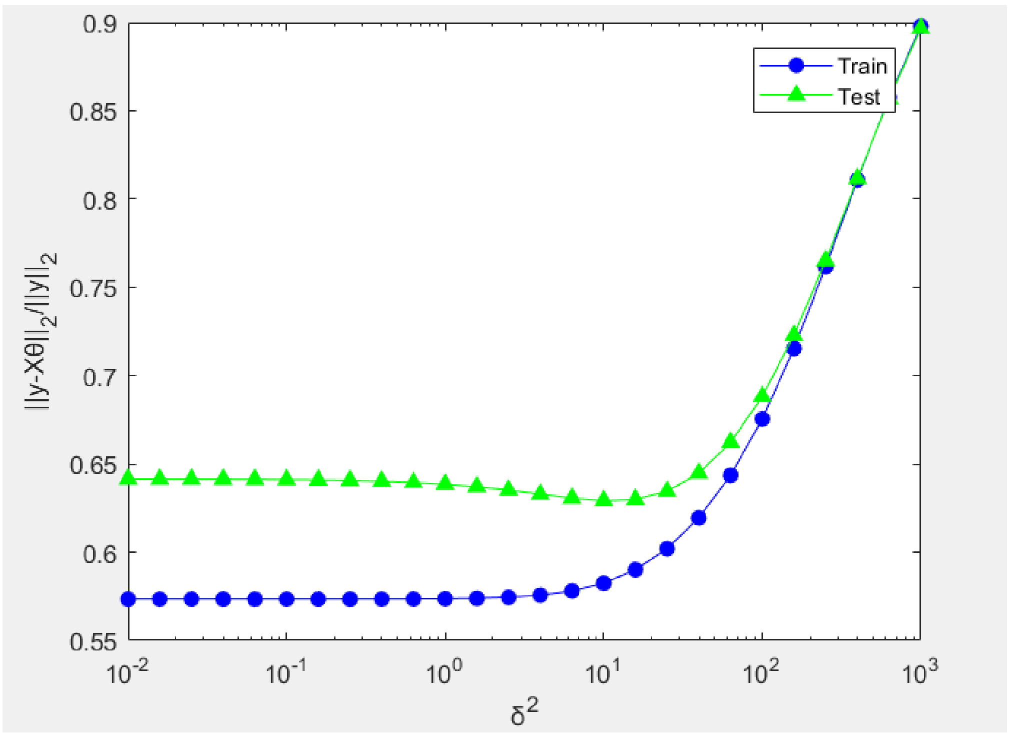
```
% Use mask index to shuffle the orders of patients and discard 2 lines of
% data randomly that we can divide the data into 5 folds evenly
mask = randperm(num_patients, num_patients - 2)';
X = X(mask, :);
y = y(mask, :);
```

Then use a for-loop to do 5-folds cross validation, where δ^2 ranges from 10^{-2} to 10^3 (from above we can know that $\delta^2 = 10^4$ is too high, so we shrink the range). Each time choose 1 fold as test set in proper order, the rest as training set.

```
for n = 1:num_folds

    % Split it into training data and test data
    X_test = X(data_per_fold * (n - 1) + 1:data_per_fold * n, :);
    y_test = y(data_per_fold * (n - 1) + 1:data_per_fold * n, :);
    X_train = X([1:data_per_fold * (n - 1), data_per_fold * n + 1:num_patients - 2], :);
    y_train = y([1:data_per_fold * (n - 1), data_per_fold * n + 1:num_patients - 2], :);

    % Nomalization
    X_mean = mean(X_train);
    y_mean = mean(y_train);
    X_std = sqrt(mean((X_train - X_mean).^2));
    X_train = (X_train - X_mean) ./ X_std;
    y_train = y_train - y_mean;
    X_test = (X_test - X_mean) ./ X_std;
    y_test = y_test - y_mean;

    % Compute the ridge regression solutions for a range of regularizers (δ^2)
    delta2 = zeros(num_delta, 1);
    theta = zeros(num_delta, num_vars - 1);
    for i = 1:num_delta
        ldelta2 = -2 + 0.2 * (i - 1);  % lg(δ^2) range from -2 to 3
        delta2(i) = 10 ^ (ldelta2);  % δ^2 range from 10^(-2) to 10^(3)
        theta(i,:) = ridge(X_train, y_train, delta2(i));
    end

    % Compute yhat on training and test set
    yhat_train = X_train * theta';
    yhat_test = X_test * theta';

    % Compute relative error on training and test set
    relative_error_train(n, :) = sqrt(sum((yhat_train - y_train).^2) / sum(y_train.^2));
    relative_error_test(n, :) = sqrt(sum((yhat_test - y_test).^2) / sum(y_test.^2));

end
```

Finally, compute and plot the average relative errors in 5 folds on training and test data, choosing the one with lowest test error:

```
% Compute average relative errors in 5-fold cross validation
mean_error_train =  mean(relative_error_train);
mean_error_test =  mean(relative_error_test);

% Plot relative training and test error against δ^2
semilogx(delta2, mean_error_train, 'b-o', 'markerfacecolor', 'b');
hold on;
semilogx(delta2, mean_error_test, 'g-^', 'markerfacecolor', 'g');
legend('Train','Test');
xlabel('δ^2');
ylabel('||y-Xθ||_2/||y||_2');

% Choose the value of δ^2 with the lowest test error
[minerror, loc] = min(mean_error_test);
fprintf('δ^2 = %.2f has the lowest error (%.2f) on test set\n', delta2(loc), minerror);
```



And we get $\delta^2 = 10.00$ (in this experiment) which has the lowest relative error (0.63) on test set.

(3) Experimental Analysis

① As I analyzed in last section, the expectation of the bias term β_0 is 0. I computed the values of β_0 for different values of δ^2 in the file 'RidgeRegression.m' and store them in the variable 'beta0'.

```
>> max(beta0)

ans =

  1.3456e-15
```

We can see that the maximum of β_0 is so small compared to the value of y that is generated by random error and can be omitted. My analysis is confirmed by this phenomenon, so I omitted the bias term β_0 in my model.

② With the exponential increasing of δ^2, the relative error gets lower gradually, then goes higher quickly. It implies that proper regularization improved the model and excessive regularization forced the parameters become so small that didn't fit the data.

③ It seems that the relative error is a little bit high, and the lowest one is about 0.63. This is because we use the square root of relative MSE as the error. Actually, the lowest relative MSE is $0.63^2 = 0.3969$, an acceptable error.

(4) Discussion of Proposed Method

The value of δ^2 is a problem because the δ^2 with the smallest test error varies from time to time, and it approximately located in $[2.51,15.85]$ (for 5-folds cross validation) . I also tried 3-folds and 10-folds cross validation, the situation still exists. So, when we decide the value of δ^2, we may randomly choose a value from the range, or the magnitude of 10 may be a good choice.