Table of Contents

ECE 532 - HW6 - Fall 2017 1
(1) Data Fitting vs. Sparsity Tradeoff 1
(a) Implementation of the ISTA (iterative soft-thresholding via proximal descent) 1
(b) Trade-off: residual norm vs weight norm (training data) 1
(c) Trade-off: error-rate vs sparsity (training data) 2
(d) Trade-off (test data) 4
(e) Comparison of Lasso and Ridge Regression 6
APPENDIX 8

ECE 532 - HW6 - Fall 2017

By Roumen Guha

```
clear
close all
```

(1) Data Fitting vs. Sparsity Tradeoff

```
load hw6_532_fall17_BreastCancer.mat
```

(a) Implementation of the ISTA (iterative soft-thresholding via proximal descent)

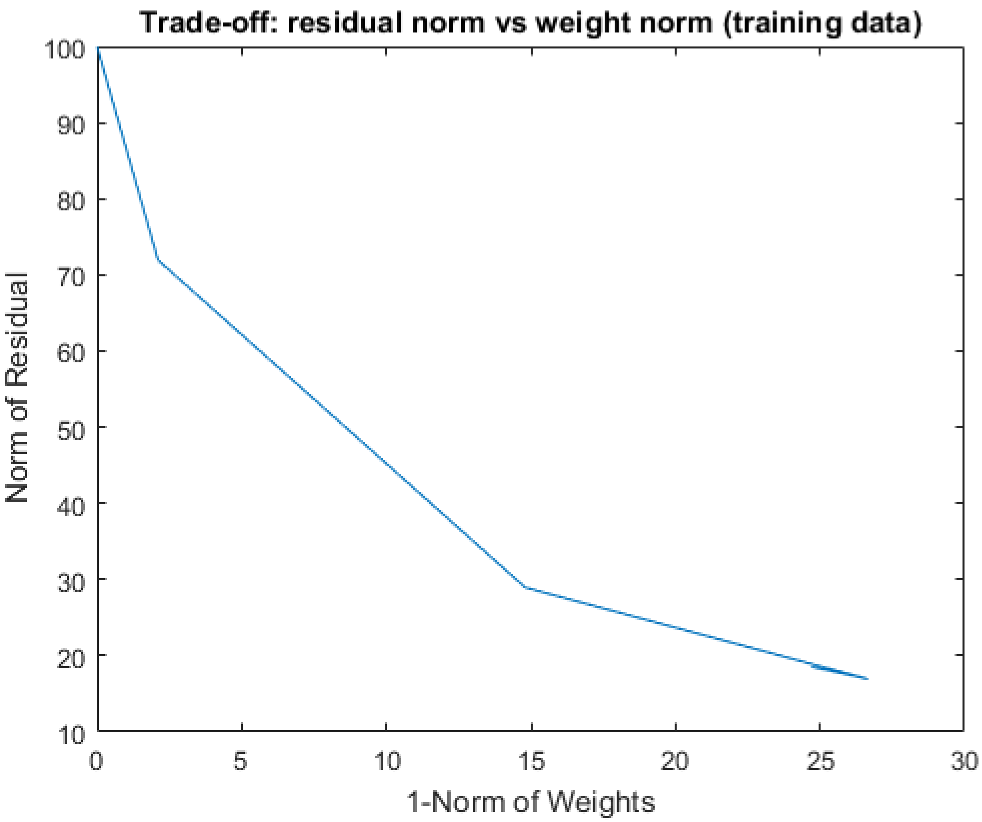
Can be found in the appendix (end)

(b) Trade-off: residual norm vs weight norm (training data)

We see that the norm of residuals falls with the increase of the 1-norm; it looks convex. In fact, looking at (c), it can be seen that as we increase lambda (i.e. penalize the 1-norm more), the sparser our solution.

```
% Specify some constants
lambda = [1e-3, 1e-2, 1e-1, 0, 1e0, 1e1, 1e2, 1e3];
tau = 1e-4;

% Divide up the dataset
num_train = 100;
num_test = length(y) - num_train;

X_train = X(1:num_train, :);
y_train = y(1:num_train);

X_test = X(num_train + 1:end, :);
y_test = y(num_train + 1:end);

% Preallocate some space
Beta = zeros(size(X, 2), length(lambda));
```

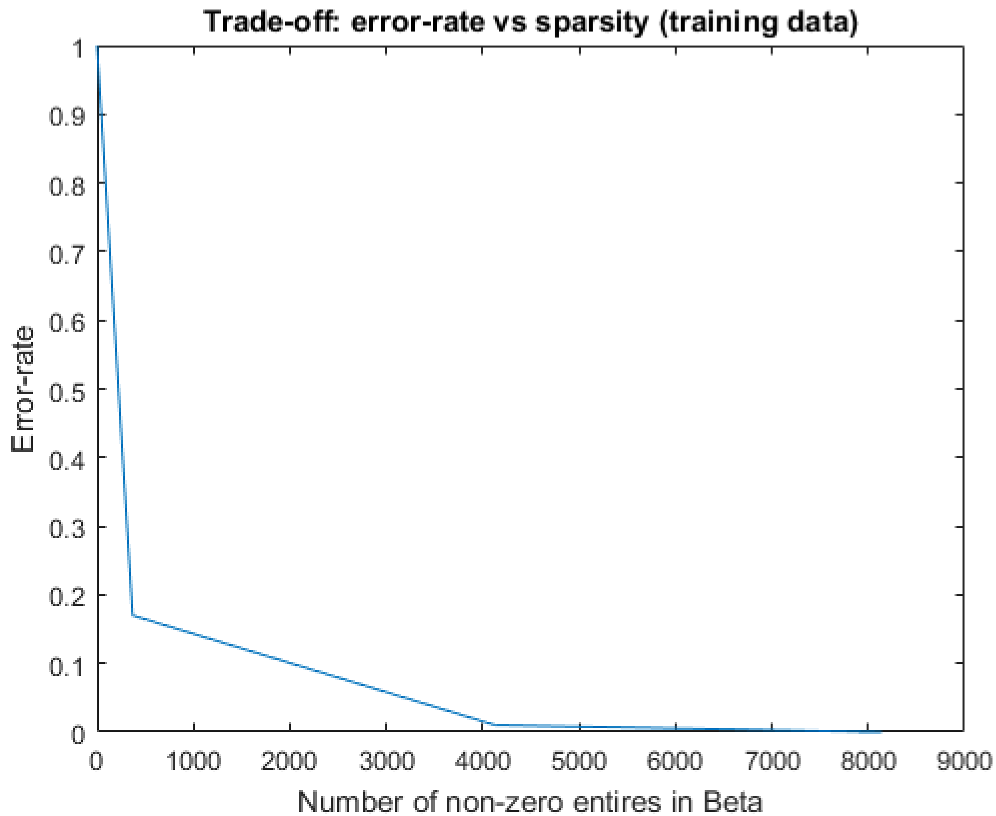
```
norms = zeros(length(lambda), 1);
residuals = zeros(length(lambda), 1);
y_hats = zeros(num_train, length(lambda));

% Do most of the work
for k = 1:length(lambda)
    Beta(:, k) = ISTA(X_train, y_train, lambda(k), tau);
    norms(k) = norm(Beta(:, k), 1);
    y_hats(:, k) = X_train * Beta(:, k);
    residuals(k) = norm(y_hats(:, k) - y_train, 1);
end

figure
grid
plot(norms, residuals)
title('Trade-off: residual norm vs weight norm (training data)')
xlabel('1-Norm of Weights')
ylabel('Norm of Residual')
```



(c) Trade-off: error-rate vs sparsity (training data)

It is obvious from the plots below that both the error-rate and the norm of the residuals increase as we increase lambda; lambda = 10 gives us both a sparse Beta and a low error-rate.

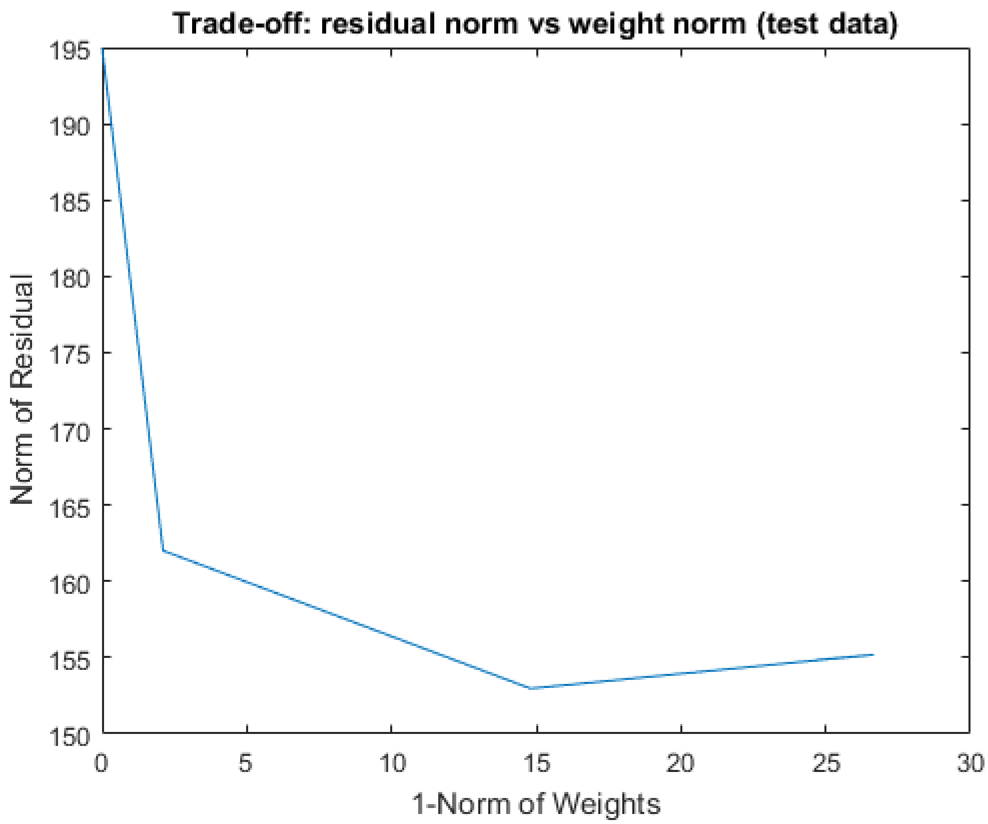
```
% Note: The error-rate is the number of incorrect predictions divided
 by
%       the total number of predictions.
%       The sparsity is the number of nonzero entries in Beta. For
 this
%       purpose, we'll say an entry Beta_i is nonzero if |Beta_i| >
 1e-6.

% Preallocate some space
error_rates = zeros(length(lambda), 1);

% Figure out the number of nonzero entries in Beta
sparsities = sum(abs(Beta) > 1e-6);

% Calculate error-rates
for k = 1:length(lambda)
    error_rates(k) = sum(sign(y_hats(:, k)) ~= y_train) / num_train;
end

figure
grid
plot(sparsities, error_rates)
title('Trade-off: error-rate vs sparsity (training data)')
xlabel('Number of non-zero entires in Beta')
ylabel('Error-rate')
```

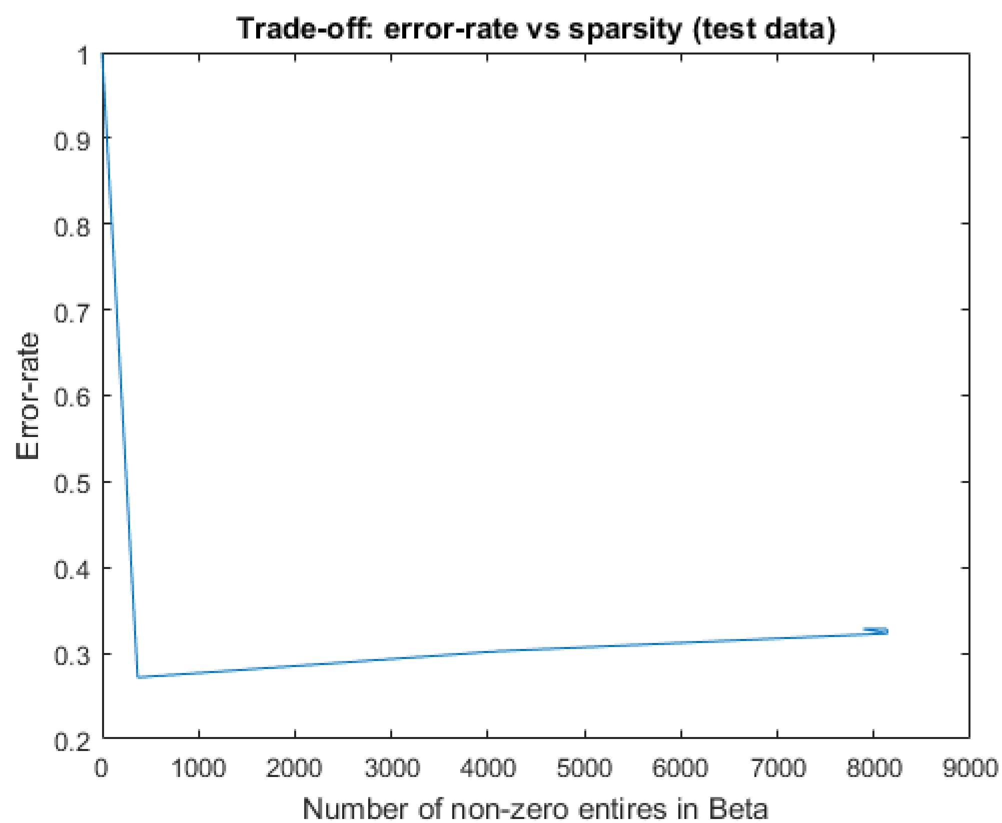


(d) Trade-off (test data)

Re-preallocate space

```
y_hats = zeros(num_test, length(lambda));

% Dome some of the work
for k = 1:length(lambda)
    y_hats(:, k) = X_test * Beta(:, k);
    residuals(k) = norm(y_hats(:, k) - y_test, 1);
end

figure
grid
plot(norms, residuals)
title('Trade-off: residual norm vs weight norm (test data)')
xlabel('1-Norm of Weights')
ylabel('Norm of Residual')

% Do rest of the work
for k = 1:length(lambda)
    error_rates(k) = sum(sign(y_hats(:, k)) ~= y_test) / num_test;
end

figure
grid
plot(sparsities, error_rates)
title('Trade-off: error-rate vs sparsity (test data)')
xlabel('Number of non-zero entires in Beta')
ylabel('Error-rate')
```


Trade-off: residual norm vs weight norm (test data)
Norm of Residual
1-Norm of Weights
195
190
185
180
175
170
165
160
155
150
0
5
10
15
20
25
30


Trade-off: error-rate vs sparsity (test data)
Error-rate
Number of non-zero entires in Beta
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

(e) Comparison of Lasso and Ridge Regression

In general, it seems as though LASSO gives a slightly lower (better) squared error than ridge regression. However, it also seems as though ridge regression gives a lower misclassification error than LASSO, though LASSO has the benefit of producing sparse solutions, which ridge regression does not.

```
% We loop over this 90 times, because there are 10C1 ways to choose 1
 set from a
% 10 choices, and then 9C1 ways to choose 1 set from the remaining 9
 choices of
% sets (note that this has the effect of choosing our 8 sets as the
 remaining
% set).

% How many iterations to average error over
num_iter = 90;

% Specify some constants
% We've learned that lambda is best around 10, so we'll respecify our
 values
lambda = [1e-2, 1e-1, 0, 1e0, 3, 5, 7, 1e1, 13, 16, 20, 25, 50, 75,
 1e2];
tau = 10e-5;

% Preallocate some space
squaredErrors_RR = zeros(1, num_iter);
squaredErrors_L = zeros(1, num_iter);

classificationErrors_RR = zeros(1, num_iter);
classificationErrors_L = zeros(1, num_iter);

for j = 1:num_iter
    % Divide up the dataset into training, tuning and testing.
    [trainInd, tuneInd, testInd] = dividerand(length(y), 0.8, 0.1,
 0.1);

    X_train = X(trainInd, :);
    y_train = y(trainInd, :);

    X_tune = X(tuneInd, :);
    y_tune = y(tuneInd);

    X_test = X(testInd, :);
    y_test = y(testInd);

    % Preallocate some space
    Beta_RR_hat = zeros(size(X, 2), length(lambda));
    Beta_L_hat = zeros(size(X, 2), length(lambda));

    y_RR_hat = zeros(length(tuneInd), length(lambda));
    y_L_hat = zeros(length(tuneInd), length(lambda));

    error_rates_RR = zeros(length(lambda), 1);
```

```
    error_rates_L = zeros(length(lambda), 1);

    % Compute the SVD of X_train
    [U, S, V] = svd(X_train, 'econ');

    for k = 1:length(lambda)
        % Compute Beta_hat for both Lasso and Ridge Regression
        %Beta_RR_hat(:, k) = (V / (S' * S + lambda(k) * eye(size(S,
 1))) * S') * U' * y_train;
        Beta_RR_hat(:, k) = V * diag(diag(S) ./ (diag(S).^2 +
 lambda(k))) * U' * y_train;
        Beta_L_hat(:, k) = ISTA(X_train, y_train, lambda(k), tau);

        % Evaluate both classifiers on tuning data
        y_RR_hat(:, k) = X_tune * Beta_RR_hat(:, k);
        y_L_hat(:, k) = X_tune * Beta_L_hat(:, k);

        % Measure squared error rates for both classifiers
 (validation)
        error_rates_RR(k) = sum((y_RR_hat(:, k) - y_tune).^2);
        error_rates_L(k) = sum((y_L_hat(:, k) - y_tune).^2);
    end

    % Find the minimum error rate (corresponds to the best lambda and
 best classifier of each type)
    [min_error_RR, best_lambda_RR] = min(error_rates_RR);
    [min_error_L, best_lambda_L] = min(error_rates_L);

    % Pick the best classifiers each
    min_error_Beta_RR_hat = Beta_RR_hat(:, best_lambda_RR);
    min_error_Beta_L_hat = Beta_L_hat(:, best_lambda_L);

    % Compute the final prediction on the test set
    final_y_RR_hat = X_test * min_error_Beta_RR_hat;
    final_y_L_hat = X_test * min_error_Beta_L_hat;

    % Compute the final squared-error in both classifiers
    squaredErrors_RR(j) = sum((y_test - final_y_RR_hat).^2);
    squaredErrors_L(j) = sum((y_test - final_y_L_hat).^2);

    % Compute the final classification error in both classifiers
    classificationErrors_RR(j) = sum(y_test ~= sign(final_y_RR_hat)) /
 length(y_test);
    classificationErrors_L(j) = sum(y_test ~= sign(final_y_L_hat)) /
 length(y_test);
end

snapnow
AverageLassoSquaredError = mean(squaredErrors_L)
snapnow
AverageRidgeSquaredError = mean(squaredErrors_RR)
snapnow

AverageLassoClassificationError = mean(classificationErrors_L)
```

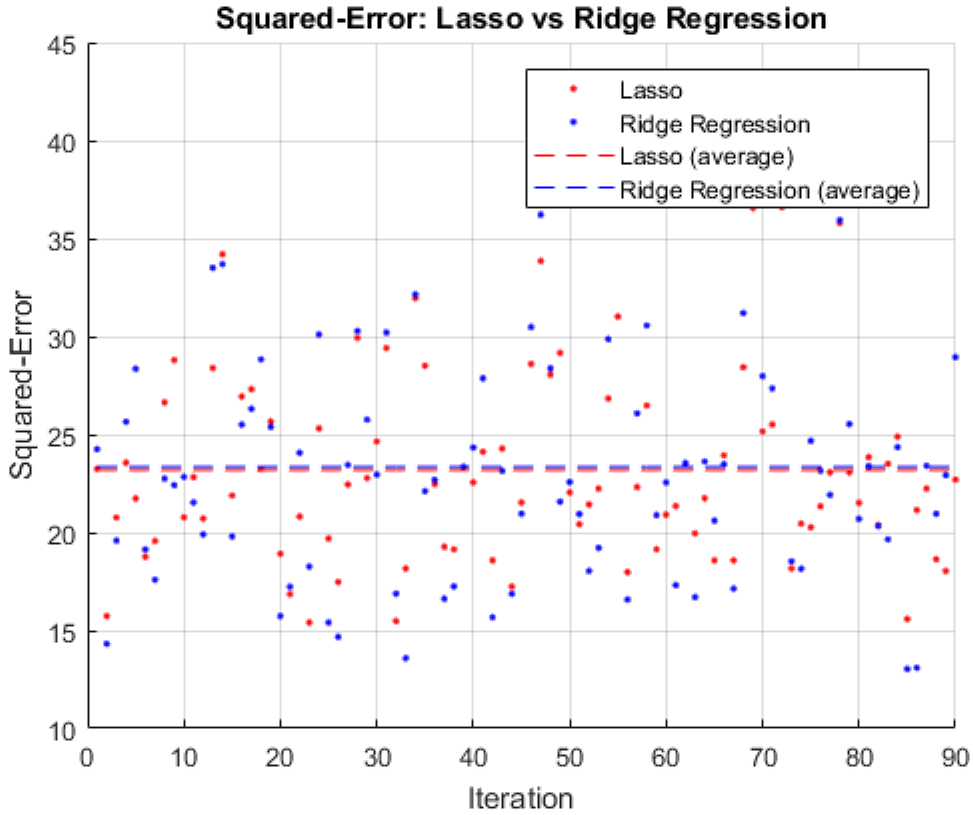
```
snapnow
AverageRidgeClassificationError = mean(classificationErrors_RR)

figure
grid
hold on
plot(1:num_iter, squaredErrors_L, 'r.', 1:num_iter,
 squaredErrors_RR, 'b.')
plot(1:num_iter, ones(1, num_iter)*AverageLassoSquaredError, 'r--',
 1:num_iter, ones(1, num_iter)*AverageRidgeSquaredError, 'b--')
hold off
title('Squared-Error: Lasso vs Ridge Regression')
legend('Lasso', 'Ridge Regression', 'Lasso (average)', 'Ridge
 Regression (average)')
ylabel('Squared-Error')
xlabel('Iteration')
snapnow

figure
grid
hold on
plot(1:num_iter, classificationErrors_L, 'r.', 1:num_iter,
 classificationErrors_RR, 'b.')
plot(1:num_iter, ones(1,
 num_iter)*AverageLassoClassificationError, 'r--', 1:num_iter, ones(1,
 num_iter)*AverageRidgeClassificationError, 'b--')
hold off
title('Classification-Error: Lasso vs Ridge Regression')
legend('Lasso', 'Ridge Regression', 'Lasso (average)', 'Ridge
 Regression (average)')
ylabel('Misclassification Rate')
xlabel('Iteration')
snapnow

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
```

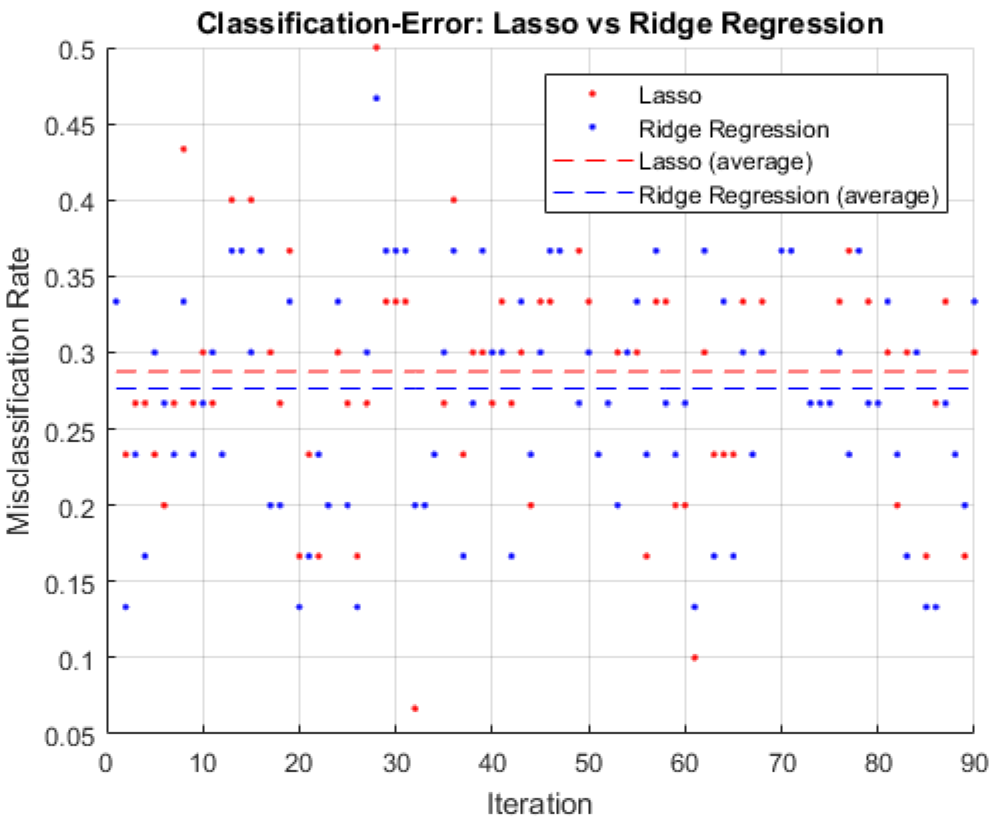
APPENDIX

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
function w = ISTA(X, y, lambda, tau)
% ISTA Lasso: Iterative Soft-Thresholding via Proximal Descent
 Algorithm
% The objective function: f(x) = 1/2 ||y - X*w||^2 + lambda*|x|
%
% Takes:
%       X: covariate matrix,
%       y: response vector,
%       lambda: penalty parameter, lambda > 0,
%       tau: training rate
% Returns:
%       w: a (p x 1) weight vector
```

```
MAX_ITER = 3000;
ABS_TOL = 25e-4;

% Preallocate some space
w = zeros(size(X,2), 1);

for k = 1:MAX_ITER
    % Store the previous w
    w_prev = w;

    % Step in the Negative Gradient Direction
    z = w - tau * X' * (X * w - y);

    % Apply soft-threshold
    w = wthresh(z, 's', lambda * tau / 2);

    % Check for convergence
    if norm(w - w_prev) < ABS_TOL
        break;
    end
end
end
```

```
AverageLassoSquaredError =

   23.1767



AverageRidgeSquaredError =

   23.3546



AverageLassoClassificationError =

    0.2874



AverageRidgeClassificationError =

    0.2763
```


Squared-Error: Lasso vs Ridge Regression
Squared-Error
Iteration
Lasso
Ridge Regression
Lasso (average)
Ridge Regression (average)


Classification-Error: Lasso vs Ridge Regression
Misclassification Rate
Iteration
Lasso
Ridge Regression
Lasso (average)
Ridge Regression (average)

Published with MATLAB® R2017a