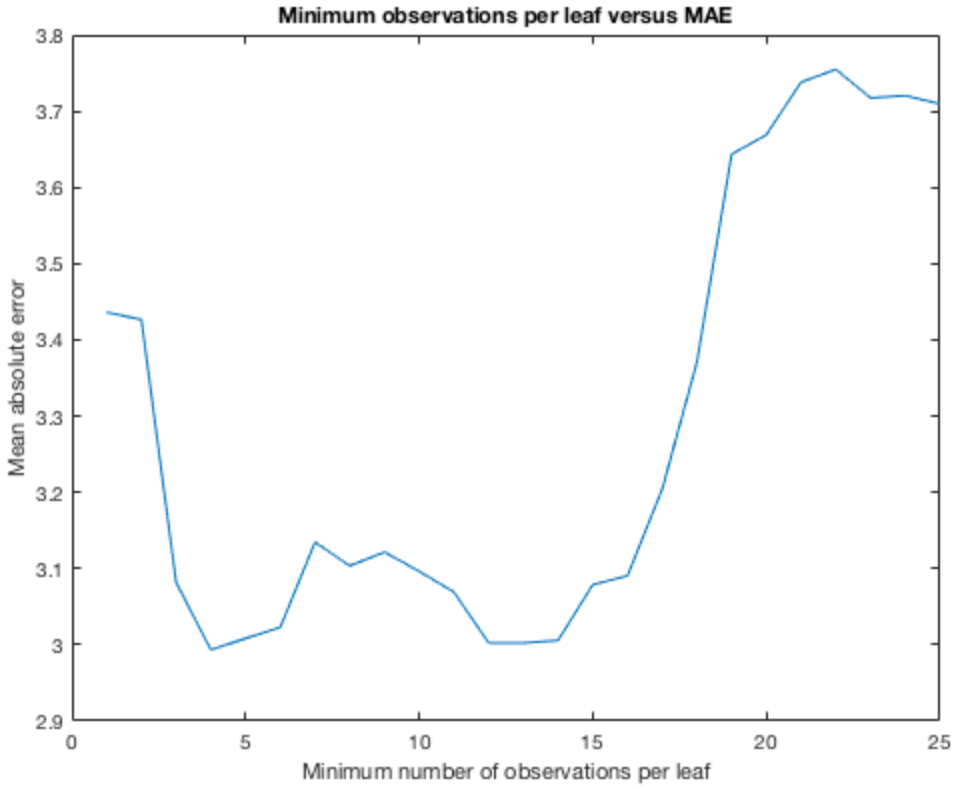
```
% Austin Welch
% EC503 HW7.1
% Exploring Boston Housing Data with Regression Trees
% https://archive.ics.uci.edu/ml/datasets/Housing

% (a) Use MATLAB's built-in functions to learn a regression tree using
 the
% training data provided. Visualize and submit an image of the
 regression
% tree that has a minimum of 20 observations per leaf.
% MATLAB functions: classregtree, fitrtree, eval, test, view.

% load data
clear; clc;
load('housing_data.mat');

% construct regression tree
tree = fitrtree(Xtrain, ytrain, 'PredictorNames', feature_names, ...
    'ResponseName',cell2mat(output_name),'MinLeafSize',20);

% display
view(tree,'Mode','graph')

% (b) For the trained regression tree from part (a) that has a minimum
 of
% 20 observations per leaf, what is the estimated MEDV value for the
% following test feature fector: CRIM = 5, ZN = 18, INDUS = 2.31, CHAS
 = 1,
% NOX = 0.5440, RM = 2, AGE = 64, DIS = 3.7, RAD = 1, TAX = 300,
 PTRATIO =
% 15, B = 390, LSTAT = 10?
testVec = [5 18 2.31 1 0.5440 2 64 3.7 1 300 15 390 10];
y_hat = predict(tree,testVec);
fprintf('Estimated MEDV value for the given feature vector is: %0.4f
\n',...
    y_hat);

% (c) Plot the mean absolute error (MAE) of the training and testing
 data
% as a function of the minimum observations per leaf ranging from 1 to
 25
% (one of the many ways to change the sensitivity of a tree). What
 trend do
% you notice?


% CHECK if I need train on train then test on test, or make plots for
 each
% train/train, train/test

% observations per leaf range
range = 25;
```

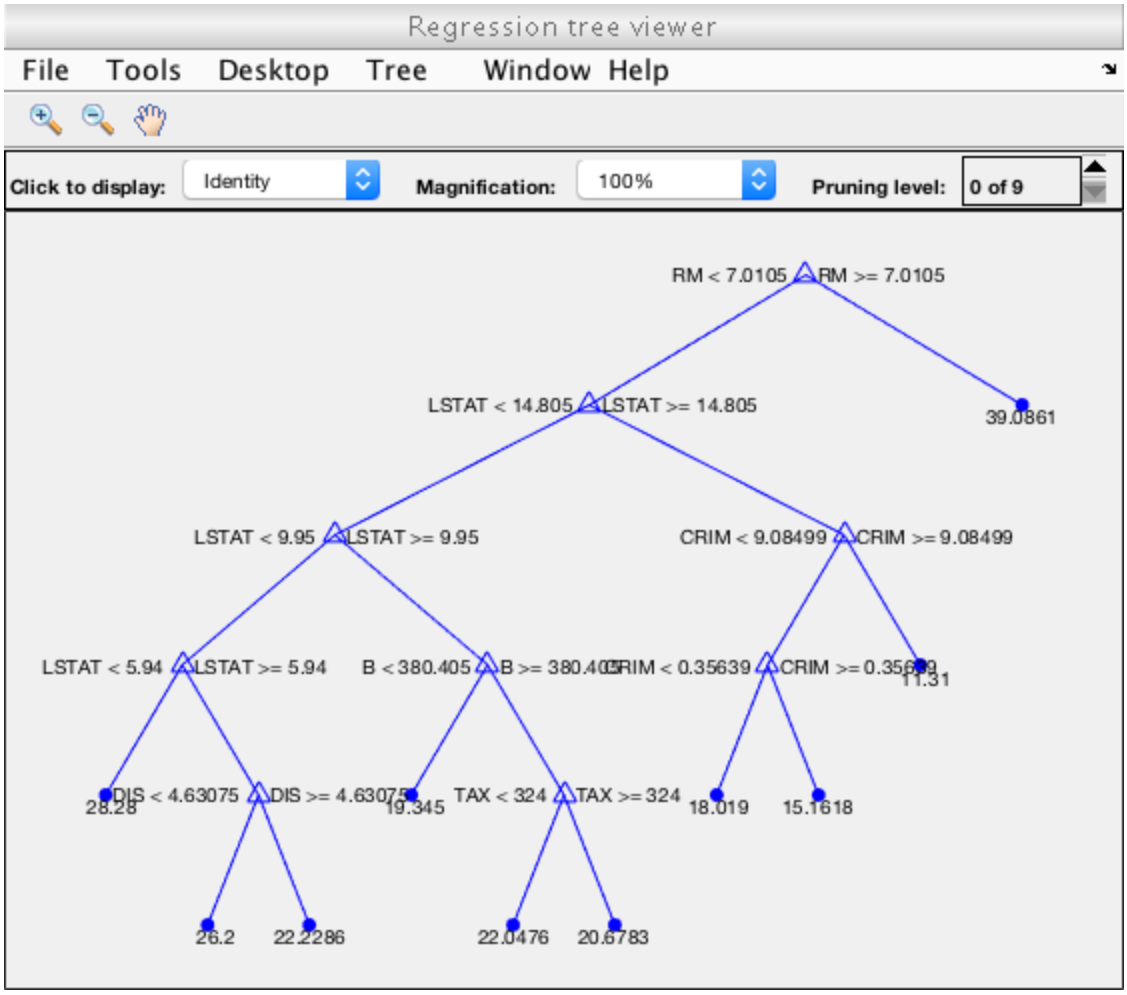
```
meanAbsErr = zeros(1,range);
for opl=1:range
    % train
    tree = fitrtree(Xtrain, ytrain, 'PredictorNames',
 feature_names, ...
    'ResponseName',cell2mat(output_name),'MinLeafSize',opl);
    % predict
    y_hat = predict(tree,Xtest);
    %error
    err = (ytest-y_hat);
    meanAbsErr(opl) = mae(err);
end

% plot
plot(1:range,meanAbsErr);
title('Minimum observations per leaf versus MAE');
xlabel('Minimum number of observations per leaf');
ylabel('Mean absolute error');

% observations
fprintf(['\nIt looks like the plot of # of observations per leaf
\n',...
    'versus MAE decreases as number of observations increases,\n',...
    'reaches a local minima, increases a bit before decreasing to
\n',...
    'another local minima, then continues to increase.\n\n']);

% best value for 'MinLeafSize'
[~,ind] = min(meanAbsErr);
fprintf('Best ''MinLeafSize'': %d\n\n', ind);
```

Estimated MEDV value for the given feature vector is: 22.0476

It looks like the plot of # of observations per leaf
versus MAE decreases as number of observations increases,
reaches a local minima, increases a bit before decreasing to
another local minima, then continues to increase.

Best 'MinLeafSize': 4


Minimum observations per leaf versus MAE
Mean absolute error
Minimum number of observations per leaf
3.8
3.7
3.6
3.5
3.4
3.3
3.2
3.1
3
2.9
0
5
10
15
20
25



Published with MATLAB® R2017a

Table of Contents

..... 1
load data 1
(a) 1
Normal equation: 1
visualize OLS regression line 2
(b) 2


```
% Austin Welch
% EC503 HW7.2
% Ordinary Least Squares (OLS) versus Robust Linear Regression
```

load data

```
clear; clc;
rng default; % For reproducibility
load('linear_data.mat');
```

(a)

Implement ordinary least squares (OLS) linear regression for input (xdata) and output (ydata) provided in the linear_data.mat file. Do not use MATLAB's robustfit or regress. (i) Will the input data matrix (xdata) yield a unique solution? Why or why not? (ii) If h_OLS(x) = x^T*W_OLS + b_OLS, report the values of w_OLS and b_OLS and the resulting mean-squared error (MSE) and mean absolute error (MAE)

```
fprintf('Part (a):\n\n');
```

```
Part (a):
```

Normal equation:

$\theta = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$ (n << 10000 so normal equations should outperform gradient descent)

```
xData = xData';
yData = yData';
xDataExt = [xData; ones(1,length(xData))];
theta_ols = inv(xDataExt*xDataExt')*xDataExt*yData';
w_ols = theta_ols(1);
b_ols = theta_ols(2);

% uniqueness of solution
fprintf(['There should be a unique solution to the input data matrix
\n',...
    'xdata because (X*X^T) is non-singular/invertible/full-rank.\n
\n']);
```

```
% parameters
fprintf('w_ols: %0.4f\n', w_ols); % slope
fprintf('b_ols: %0.4f\n\n', b_ols); % intercept

% decision rule
h_ols = xData*w_ols+b_ols;

% quantify error
MSE_ols = sum((yData-h_ols).^2)/length(yData);
MAE_ols = sum(abs(yData-h_ols))/length(yData);
fprintf('MSE_ols: %0.4f\n', MSE_ols);
fprintf('MAE_ols: %0.4f\n\n', MAE_ols);
```

```
There should be a unique solution to the input data matrix
xdata because (X*X^T) is non-singular/invertible/full-rank.

w_ols: 0.0234
b_ols: 2.9738

MSE_ols: 0.2588
MAE_ols: 0.3175
```

visualize OLS regression line

```
%{
figure(1);
scatter(xData,yData);
hold on;
plot(xData,h_ols)
%}
% to check: above line and 'lsline' function are collinear
%lsline
```

(b)

Use MATLAB's robustfit function to implement robust linear regression for the input and output in linear_data.mat. There are two outliers in the dataset which skew the OLS regression model because it is not robust to outliers.

```
fprintf('Part (b):\n\n');

% loss functions (const, coefficient) with default tuning constants
cauchy = robustfit(xData,yData,'cauchy'); % w = 1 ./ (1 + r.^2)
fair = robustfit(xData,yData,'fair'); % w = 1 ./ (1 + abs(r))
huber = robustfit(xData,yData,'huber'); % w = 1 ./ max(1, abs(r))
talwar = robustfit(xData,yData,'talwar'); % w = 1 * (abs(r)<1)
ols = robustfit(xData,yData,'ols'); % same as (a), repeated for
 consistency

% estimates
hCauchy = xData*cauchy(2) + cauchy(1);
hFair = xData*fair(2) + fair(1);
```

```
hHuber = xData*huber(2) + huber(1);
hTalwar = xData*talwar(2) + talwar(1);
hOls = xData*ols(2) + ols(1);

% MSEs
MSEcauchy = mse(yData,hCauchy);
MSEfair = mse(yData,hFair);
MSEhuber = mse(yData,hHuber);
MSEtalwar = mse(yData,hTalwar);
MSEols = mse(yData,hOls);

% MAEs
MAEcauchy = mae(yData,hCauchy);
MAEfair = mae(yData,hFair);
MAEhuber = mae(yData,hHuber);
MAEtalwar = mae(yData,hTalwar);
MAEols = mae(yData,hOls);

% Compare robust errors to OLS error
Cauchy = [MSEcauchy; MAEcauchy];
Fair = [MSEfair; MAEfair];
Huber = [MSEhuber; MAEhuber];
Talwar = [MSEtalwar; MAEtalwar];
OLS = [MSEols; MAEols];
T = table(Cauchy, Fair, Huber, Talwar, OLS, 'RowNames',
 {'MSE', 'MAE'});
disp(T);
fprintf('OLS has lowest MSE, but highest MAE\n\n');

% report values of wHuber and bHuber
fprintf('w_huber: %0.4f\n', hHuber(2));
fprintf('b_huber: %0.4f\n\n', hHuber(1));

% plot ols and robust methods to compare
figure(2);
scatter(xData,yData,'filled'); grid on; hold on
plot(xData, hOls, 'k', 'LineWidth', 1);
plot(xData, hCauchy, 'b', 'LineWidth', 1);
plot(xData, hFair, 'g', 'LineWidth', 1);
plot(xData, hHuber, 'r', 'LineWidth', 1);
plot(xData, hTalwar, 'y', 'LineWidth', 1);
legend({'Data','OLS', 'Cauchy', 'Fair', 'Huber', 'Talwar'},'Box','off')
set(legend,'position',[0.15  0.12  0.1286  0.3])
set(gca,'fontsize',7)
title('Comparison of OLS to robust linear regression loss
 functions', ...
    'fontsize',7);
xlabel('X'); ylabel('Y');

% observations
fprintf(['The Cauchy loss function has the lowest MAE. All of the
\n', ...
    'robust loss functions are much less sensitive to the outliers
\n',...
```

```
    'compared to ordinary least squares regression.\n\n']);
```

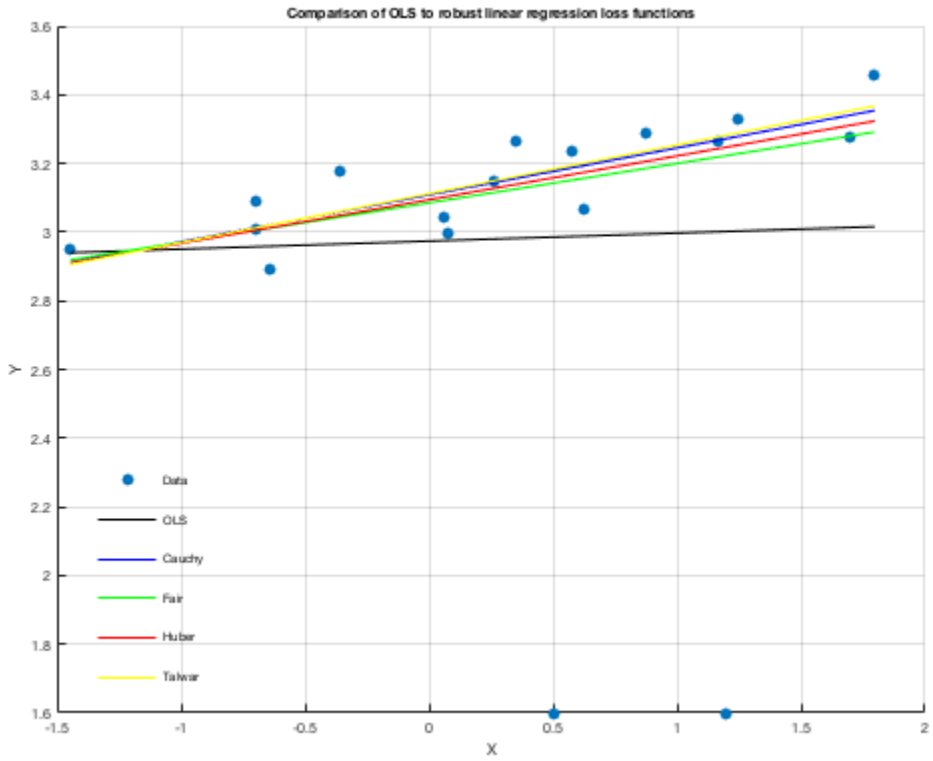
Part (b):

	Cauchy	Fair	Huber	Talwar	OLS
MSE	0.29955	0.28608	0.29218	0.30242	0.25884
MAE	0.24268	0.24683	0.24509	0.24349	0.31747

OLS has lowest MSE, but highest MAE

w_huber: 3.1743
b_huber: 3.2429

The Cauchy loss function has the lowest MAE. All of the robust loss functions are much less sensitive to the outliers compared to ordinary least squares regression.



Published with MATLAB® R2017a

Table of Contents

..... 1
load data 1
(a) 1
Normal equation: 1
visualize OLS regression line 2
(b) 2


```
% Austin Welch
% EC503 HW7.2
% Ordinary Least Squares (OLS) versus Robust Linear Regression
```

load data

```
clear; clc;
rng default; % For reproducibility
load('linear_data.mat');
```

(a)

Implement ordinary least squares (OLS) linear regression for input (xdata) and output (ydata) provided in the linear_data.mat file. Do not use MATLAB's robustfit or regress. (i) Will the input data matrix (xdata) yield a unique solution? Why or why not? (ii) If h_OLS(x) = x^T*W_OLS + b_OLS, report the values of w_OLS and b_OLS and the resulting mean-squared error (MSE) and mean absolute error (MAE)

```
fprintf('Part (a):\n\n');
```

```
Part (a):
```

Normal equation:

$\theta = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$ (n << 10000 so normal equations should outperform gradient descent)

```
xData = xData';
yData = yData';
xDataExt = [xData; ones(1,length(xData))];
theta_ols = inv(xDataExt*xDataExt')*xDataExt*yData';
w_ols = theta_ols(1);
b_ols = theta_ols(2);

% uniqueness of solution
fprintf(['There should be a unique solution to the input data matrix
\n',...
    'xdata because (X*X^T) is non-singular/invertible/full-rank.\n
\n']);
```

```
% parameters
fprintf('w_ols: %0.4f\n', w_ols); % slope
fprintf('b_ols: %0.4f\n\n', b_ols); % intercept

% decision rule
h_ols = xData*w_ols+b_ols;

% quantify error
MSE_ols = sum((yData-h_ols).^2)/length(yData);
MAE_ols = sum(abs(yData-h_ols))/length(yData);
fprintf('MSE_ols: %0.4f\n', MSE_ols);
fprintf('MAE_ols: %0.4f\n\n', MAE_ols);
```

```
There should be a unique solution to the input data matrix
xdata because (X*X^T) is non-singular/invertible/full-rank.

w_ols: 0.0234
b_ols: 2.9738

MSE_ols: 0.2588
MAE_ols: 0.3175
```

visualize OLS regression line

```
%{
figure(1);
scatter(xData,yData);
hold on;
plot(xData,h_ols)
%}
% to check: above line and 'lsline' function are collinear
%lsline
```

(b)

Use MATLAB's robustfit function to implement robust linear regression for the input and output in linear_data.mat. There are two outliers in the dataset which skew the OLS regression model because it is not robust to outliers.

```
fprintf('Part (b):\n\n');

% loss functions (const, coefficient) with default tuning constants
cauchy = robustfit(xData,yData,'cauchy'); % w = 1 ./ (1 + r.^2)
fair = robustfit(xData,yData,'fair'); % w = 1 ./ (1 + abs(r))
huber = robustfit(xData,yData,'huber'); % w = 1 ./ max(1, abs(r))
talwar = robustfit(xData,yData,'talwar'); % w = 1 * (abs(r)<1)
ols = robustfit(xData,yData,'ols'); % same as (a), repeated for
 consistency

% estimates
hCauchy = xData*cauchy(2) + cauchy(1);
hFair = xData*fair(2) + fair(1);
```

```
hHuber = xData*huber(2) + huber(1);
hTalwar = xData*talwar(2) + talwar(1);
hOls = xData*ols(2) + ols(1);

% MSEs
MSEcauchy = mse(yData,hCauchy);
MSEfair = mse(yData,hFair);
MSEhuber = mse(yData,hHuber);
MSEtalwar = mse(yData,hTalwar);
MSEols = mse(yData,hOls);

% MAEs
MAEcauchy = mae(yData,hCauchy);
MAEfair = mae(yData,hFair);
MAEhuber = mae(yData,hHuber);
MAEtalwar = mae(yData,hTalwar);
MAEols = mae(yData,hOls);

% Compare robust errors to OLS error
Cauchy = [MSEcauchy; MAEcauchy];
Fair = [MSEfair; MAEfair];
Huber = [MSEhuber; MAEhuber];
Talwar = [MSEtalwar; MAEtalwar];
OLS = [MSEols; MAEols];
T = table(Cauchy, Fair, Huber, Talwar, OLS, 'RowNames',
 {'MSE', 'MAE'});
disp(T);
fprintf('OLS has lowest MSE, but highest MAE\n\n');

% report values of wHuber and bHuber
fprintf('w_huber: %0.4f\n', hHuber(2));
fprintf('b_huber: %0.4f\n\n', hHuber(1));

% plot ols and robust methods to compare
figure(2);
scatter(xData,yData,'filled'); grid on; hold on
plot(xData, hOls, 'k', 'LineWidth', 1);
plot(xData, hCauchy, 'b', 'LineWidth', 1);
plot(xData, hFair, 'g', 'LineWidth', 1);
plot(xData, hHuber, 'r', 'LineWidth', 1);
plot(xData, hTalwar, 'y', 'LineWidth', 1);
legend({'Data','OLS', 'Cauchy', 'Fair', 'Huber', 'Talwar'},'Box','off')
set(legend,'position',[0.15  0.12  0.1286  0.3])
set(gca,'fontsize',7)
title('Comparison of OLS to robust linear regression loss
 functions', ...
    'fontsize',7);
xlabel('X'); ylabel('Y');

% observations
fprintf(['The Cauchy loss function has the lowest MAE. All of the
\n', ...
    'robust loss functions are much less sensitive to the outliers
\n',...
```

```
    'compared to ordinary least squares regression.\n\n']);
```

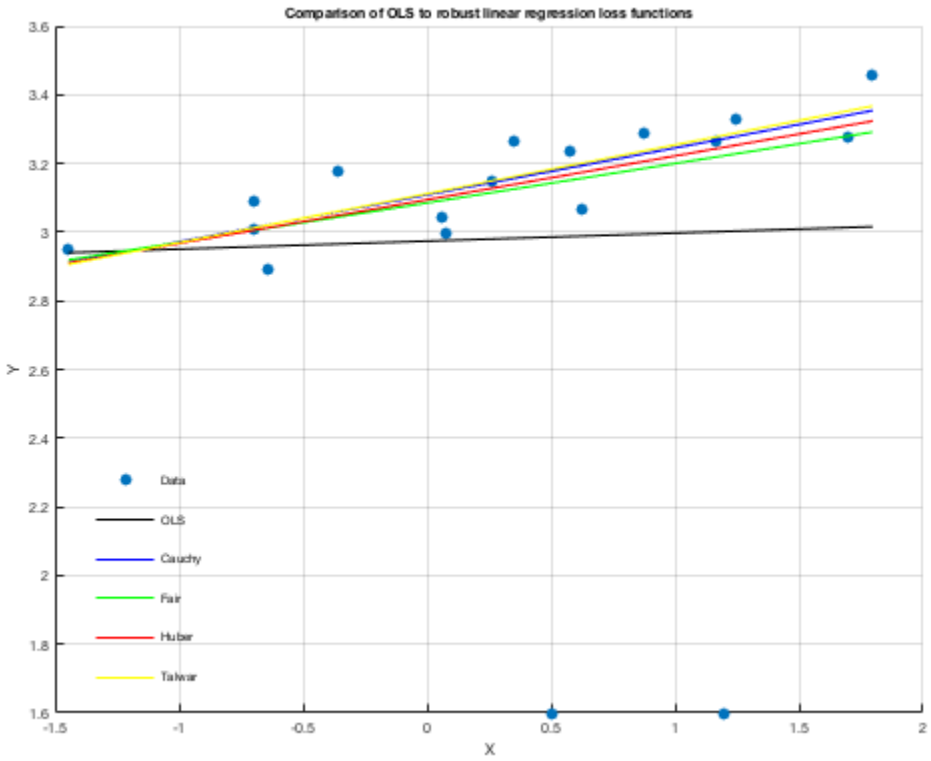
Part (b):

	Cauchy	Fair	Huber	Talwar	OLS
MSE	0.29955	0.28608	0.29218	0.30242	0.25884
MAE	0.24268	0.24683	0.24509	0.24349	0.31747

OLS has lowest MSE, but highest MAE

w_huber: 3.1743
b_huber: 3.2429

The Cauchy loss function has the lowest MAE. All of the robust loss functions are much less sensitive to the outliers compared to ordinary least squares regression.



Published with MATLAB® R2017a

Table of Contents

..... 1
load data 1
(a) 1
Observations 4
(b) 5
(c) 8


```
% Austin Welch
% EC503 HW7.3
% Overfitting and Ridge Regression
```

load data

```
clear; clc;
rng default; % For reproducibility
load('quad_data.mat');
```

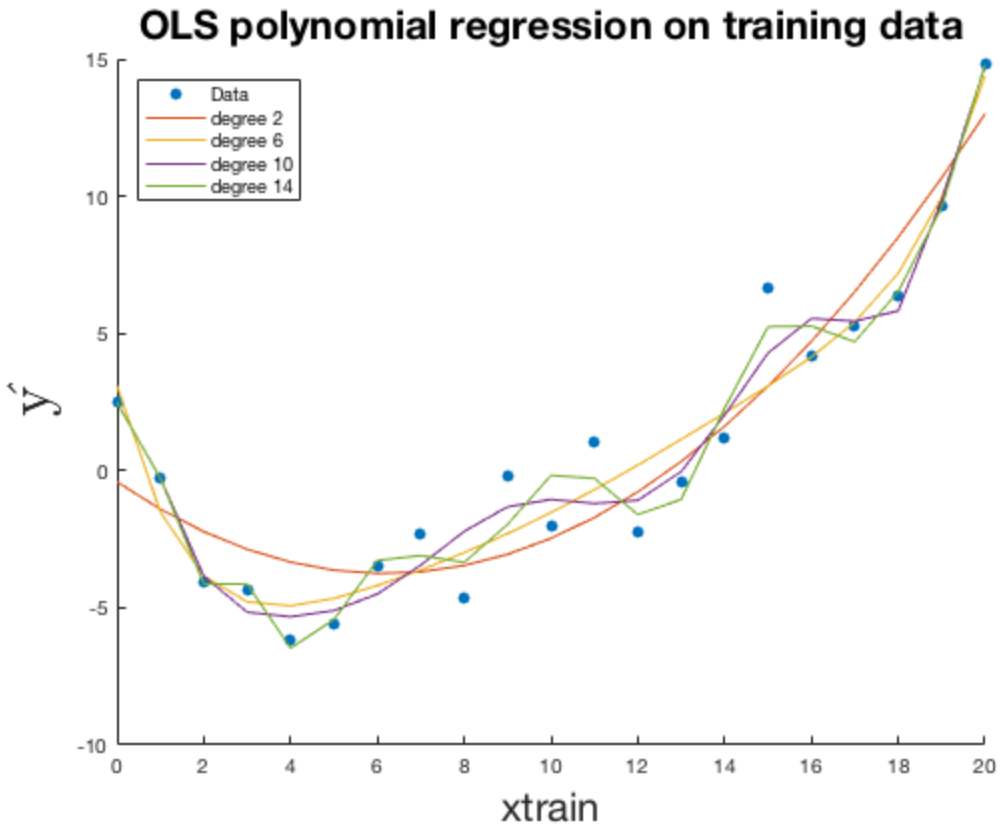
(a)

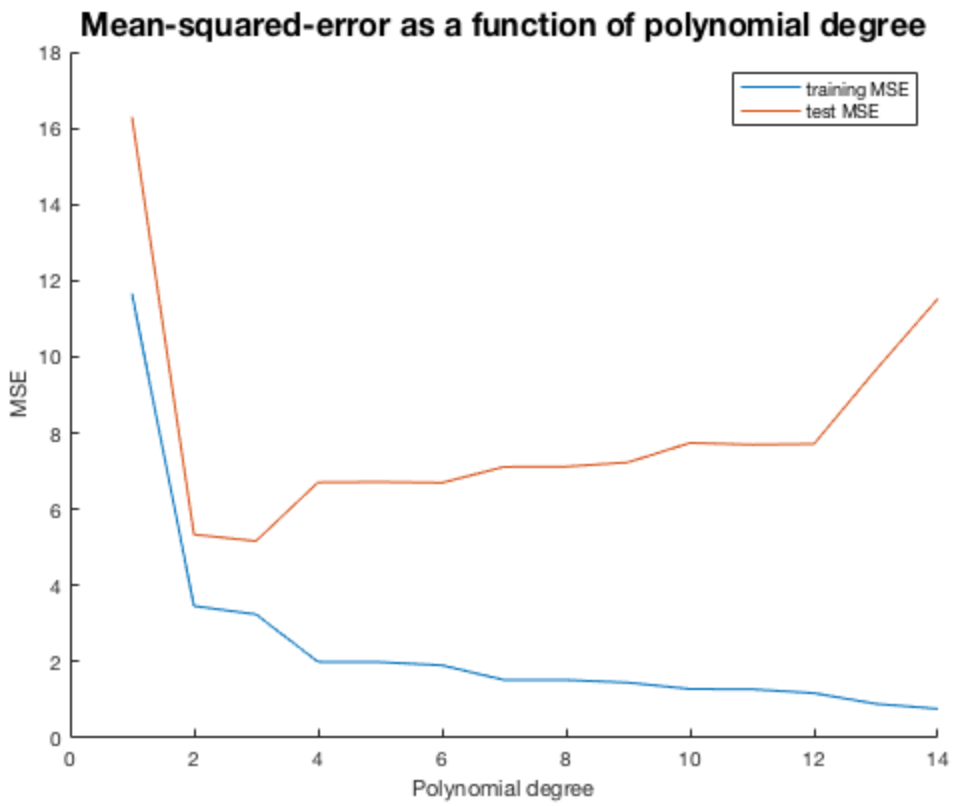
Use OLS to fit polynomials of degree d=1,...,14 to the training data. For this, use MATLAB's ridge function with the vector of ridge parameters (regularization parameters) k set to zero in order to reduce ridge regression to OLS. Also set the parameter 'scaled' to zero in order to ensure that the coefficients estimated are on the same scale as the original data. (i) On one figure, plot the training points, with polynomial curves of degree 2, 6, 10 and 14 overlaid on top. Label accordingly. (ii) On another figure, plot the MSE of the training and testing data as a function of polynomial degree (1 to 14). What do you observe? Note: Polynomials of degree d in the variable x_j are of the form: w_0 + sum_{i=1}^{d} w_i(x_j)^{i}, where x_j is the given sample

```
% OLS to fit polynomials to degree d=1,...,14
% reduce ridge to OLS by setting reg params and scale to zero
d = 14; % polynomial degree range
D = xtrain; % design matrix
W = cell(1,d); % polynomial regression coefficients
H = cell(1,d); % model/hypothesis (y_hat)
for i=1:d
    if i~=1
        D = [D D(:,1).^i];    %#ok<AGROW>
    end
    W{i} = ridge(ytrain,D,0,0);
    H{i} = D*W{i}(2:end,:)+W{i}(1,:);
end


% plot polynomial curves of degree 2,6,10,14 overlaid
figure(1);
hold on;
```

```
scatter(xtrain,ytrain,'filled');
for i = [2 6 10 14]
    plot(xtrain,H{i});
end
legend('Data','degree 2','degree 6','degree 10','degree 14', ...
    'Location','northwest');
title('OLS polynomial regression on training data','FontSize',20);
xlabel('xtrain','FontSize',20);
ylabel('y$\hat{}$','Interpreter','latex','FontSize',30);

% use weights from xtrain for xtest
D2 = xtest;
H2 = cell(1,d);
for i=1:d
    if i~=1
        D2 = [D2 D2(:,1).^i];    %#ok<AGROW>
    end
    H2{i} = D2*W{i}(2:end,:)+W{i}(1,:);
end


%plotting curves for xtest
%{
figure(2)
hold on;
scatter(xtest,ytest);
for i = [2 6 10 14]
    plot(xtest,H2{i});
end
legend('Data','degree 2','degree 6','degree 10','degree 14', ...
    'Location','northwest');
title('OLS polynomial regression on test data (with train
 weights)',...
    'FontSize',14);
xlabel('xtest','FontSize',20);
ylabel('y$\hat{}$','Interpreter','latex','FontSize',30);
%}


% Plot MSE of training and testing data
trainMSE=zeros(1,d);
testMSE=zeros(1,d);
for i=1:d
    trainMSE(i) = mse(ytrain,H{i});
    testMSE(i) = mse(ytest,H2{i});
end

figure(3);
hold on;
plot(trainMSE);
plot(testMSE);
title('Mean-squared-error as a function of polynomial degree', ...
    'FontSize',16);
xlabel('Polynomial degree');
```

```
ylabel('MSE');
legend('training MSE','test MSE');
```





Observations

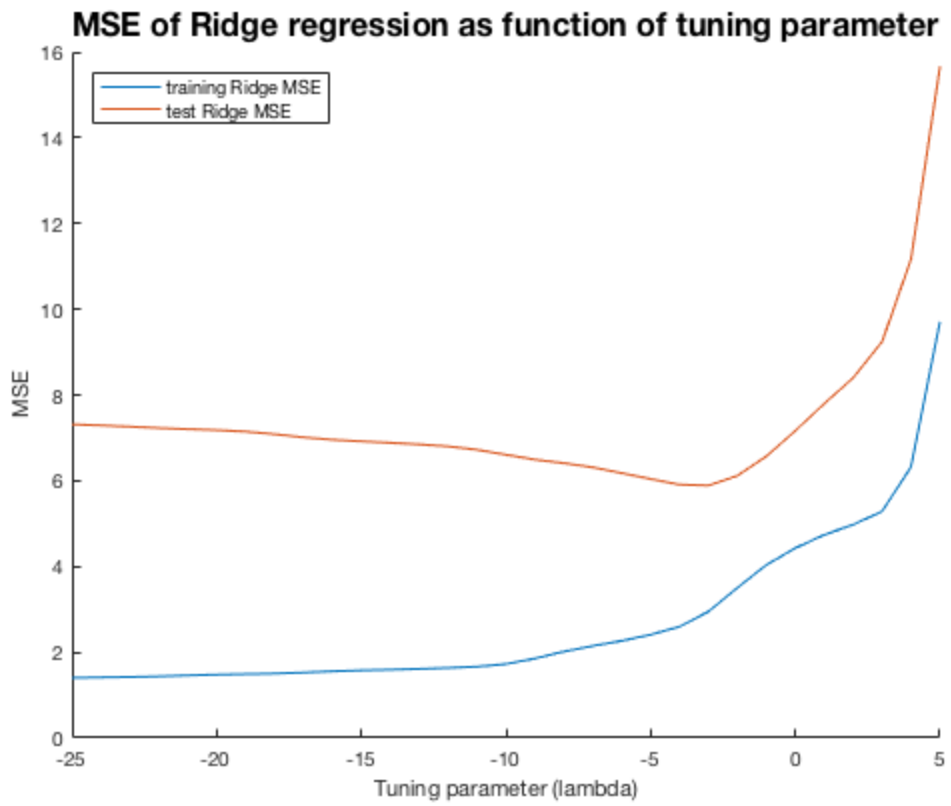
```
% Because the weights in OLS polynomial regression were chosen in
 order
% to minimize the training cost function, the MSE for the training
 data
% will be monotonically decreasing as flexibility increases and allows
 for
% a closer fit.

% On the other hand, fitting the training data more closely will cause
% overfitting on the test data at some threshold. This is obvious from
% the U-shaped graph of the test MSE, where the minima is the best fit
% polynomial degree based on the weights obtained from training, and
 to
% the right of this minima the MSE increases due to overfitting. This
 curve
% represents the bias-variance trade-off of the model.

% The best fit polynomial for the testing data with these weights is
 of
% degree 3.
```

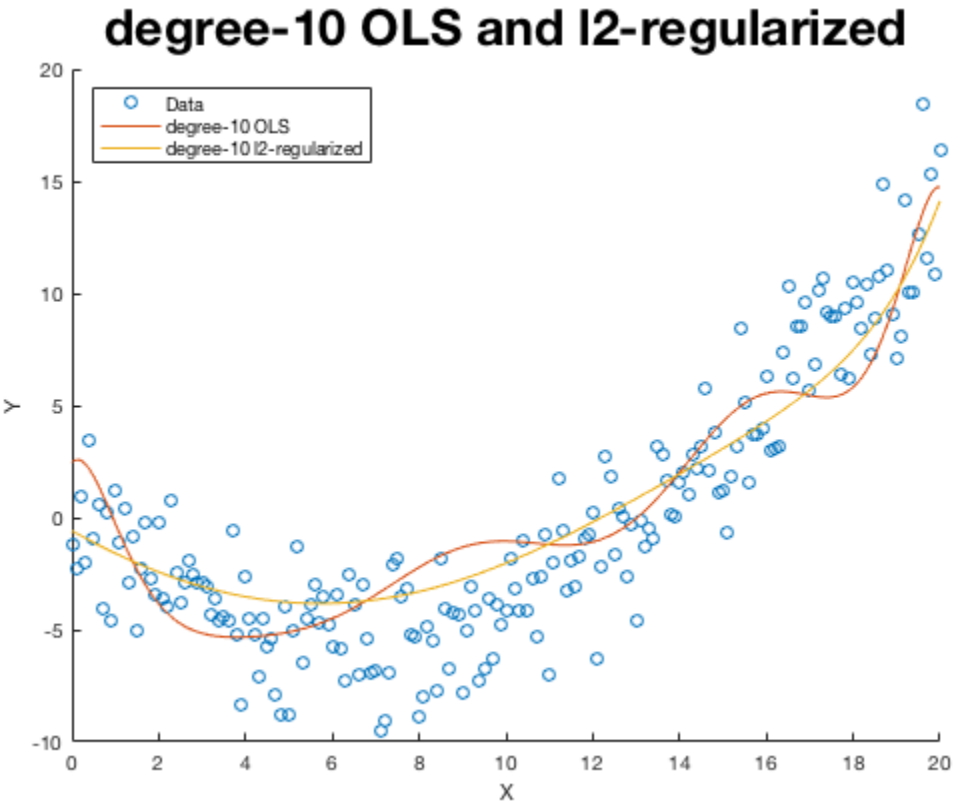
(b)

By adding l2-regularization to linear regression (making it ridge regression) the effects of over-fitting can be alleviated. Fix the polynomial degree to 10. (i) Again, on one figure, plot the mean-squared- error (MSE) of the training and testing data as a function of ln(lambda) for ln(lambda) ranging from -25 to 5 in increments of 1. (ii) On another figure, plot the testing points along with the original (no-regularization) OLS degree 10 polynomial fit and the l2-regularized degree 10 fit which has the smallest testing MSE. What do you observe?

```
% tuning parameter, coefficient of the shrinkage penalty
lambda = exp(-25:5);
degree = 10;

% training data design matrix for 10-degree polynomial
xtr = xtrain;
D3 = [xtr xtr.^2 xtr.^3 xtr.^4 xtr.^5 xtr.^6 xtr.^7 xtr.^8 xtr.^9
 xtr.^10];
W3 = cell(1,length(lambda));
H3 = cell(1,length(lambda));
trainRidgeMSE = zeros(1,length(lambda));

% test data design matrix
xte = xtest;
D4 = [xte xte.^2 xte.^3 xte.^4 xte.^5 xte.^6 xte.^7 xte.^8 xte.^9
 xte.^10];
H4 = cell(1,length(lambda));
testRidgeMSE = zeros(1,length(lambda));

% repeat over all values of lambda
for i=1:length(lambda)
    W3{i} = ridge(ytrain,D3,lambda(i),0); % polynomial coefficients
    H3{i} = D3*W3{i}(2:end,:)+W3{i}(1,:); % train estimate
    trainRidgeMSE(i) = mse(ytrain,H3{i}); % train MSE
    H4{i} = D4*W3{i}(2:end,:)+W3{i}(1,:); % test estimate
    testRidgeMSE(i) = mse(ytest,H4{i}); % test MSE
end

% Plot
figure(4);
hold on;
plot(-25:5,trainRidgeMSE);
plot(-25:5,testRidgeMSE);
title('MSE of Ridge regression as function of tuning parameter', ...
    'FontSize',16);
xlabel('Tuning parameter (lambda)');
ylabel('MSE');
legend('training Ridge MSE','test Ridge MSE', 'Location','northwest');

[~,m] = min(testRidgeMSE); % m = 23
bestLambda = lambda(m); % = e^-3

% The value of the tuning parameter that minimizes the MSE turns out
 to be
```

```
% e^-3. Penalizing complexity with l2-regularization has an obvious
 effect on
% both the training and test MSEs. The shrinkage term has the affect
 of a
% trade-off being made between the normal residual error term and this
% shrinkage term, where as the residual error term gets smaller with
% increased flexibility, the shrinkage term gets larger.


% Plot degree-10 OLS vs l2-regularized degree-10 fit with smallest MSE
figure(5);
hold on;
scatter(xtest,ytest);
plot(xtest,H2{10});
plot(xtest,H4{23});
title('degree-10 OLS and l2-regularized','FontSize',25);
xlabel('X');
ylabel('Y');
legend('Data','degree-10 OLS','degree-10 l2-regularized', ...
    'Location','northwest');

% Observation
% Comparing the degree-10 OLS model versus the degree-10 l2-
regularized
% model, the regularized version has lower MSE. It also visually looks
 like
% it fits the data better, as the non-regularized version looks like
 it was
% overfit to the training daa.
```

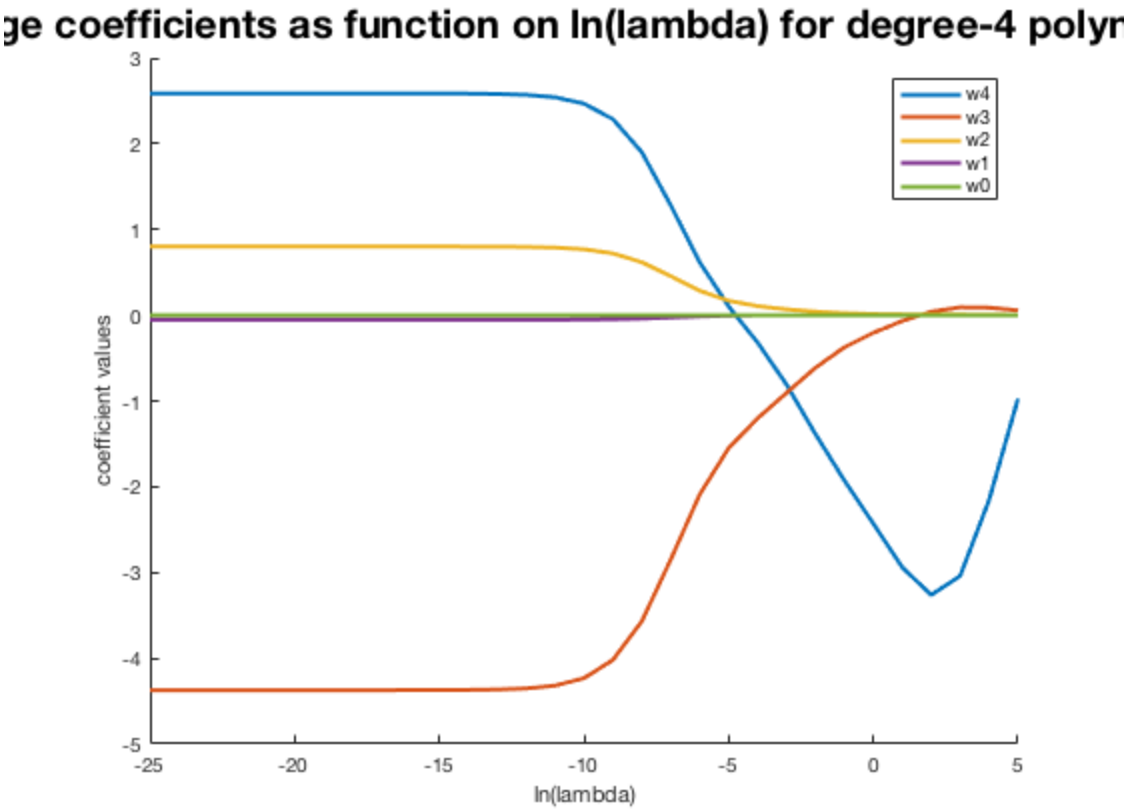

MSE of Ridge regression as function of tuning parameter
training Ridge MSE
test Ridge MSE
MSE
Tuning parameter (lambda)
16
14
12
10
8
6
4
2
0
-25
-20
-15
-10
-5
0
5


degree-10 OLS and l2-regularized
Data
degree-10 OLS
degree-10 l2-regularized
Y
X
20
15
10
5
0
-5
-10
0
2
4
6
8
10
12
14
16
18
20

(c)

On another figure, plot the ridge coefficients as a function of ln(lambda) for polynomial degree equal to 4. Plot the coefficient values as a function of ln(lambda) with ln(lambda) ranging from -25 to 5. There will be 5 curves w_0,...,w_4 (4+1, including the bias term). What do you observe?

```
D4 = [xtr xtr.^2 xtr.^3 xtr.^4];
W4 = cell(1,length(lambda));
vecs = zeros(length(lambda),5);

% repeat over all values of lambda
for i=1:length(lambda)
    W4{i} = ridge(ytrain,D4,lambda(i),0); % polynomial coefficients
    vecs(i,:) = W4{i};
end

% Plots
figure(6);
hold on;
for i=1:5
    plot(-25:5,vecs(:,i),'LineWidth',2);
end
title(['Ridge coefficients as function on ln(lambda) for
 degree-4', ...
    ' polynomial'],'FontSize',20);
legend('w4','w3','w2','w1','w0');
xlabel('ln(lambda)');
ylabel('coefficient values');

% Observations
% This plot shows the complexity penalization as a function of the
 ridge
% tuning parameter.
```



Published with MATLAB® R2017a

Table of Contents

..... 1
(a) 1
Cylic coordinate descent 2
(b) 5
(c) 5
(d) 7


```
% Austin Welch
% EC503 HW7.4
% LASSO vs. Ridge

% In this problem, we explore the trade-offs between l1-regularization
% (lasso) and l2-regularization (ridge) in a multi-dimensional
 prostate
% cancer dataset prostateStnd.mat. In this dataset, 8 features: lcavol
 -
% log(cancer volume), lweight - log(prostate weight), age, lbph -
% log(benign prostate hyperplasia amount), svi (seminal vesicle
 invasion),
% lcp - log(capsular penetration), gleason (Gleason score), and pgg45
% (percentage Gleason scores 4 or 5) are used to estimate lpsa (log
% (prostate specific antigen)). In the same fashion as the last
 problem,
% training and test data re provided: (xtrain, ytrain), (xtest,
 ytest). The
% first 8 features correspond to the first 8 entries of names. The
 ninth
% entry of names (the last one) is the output in (ytest, ytrain).
```

(a)

Implement cyclic coordinate descent for lasso (shooting algorithm). Use centered data. Update co-efficients at most 100 times or until coefficients have converged (no change). Do not use MATLAB's lasso. (i) Plot the lasso coefficient of each feature (8 total - yielding 8 curves) as a function of ln(lambda) for ln(lambda) ranging from -5 to 10 in steps of 1. Label the plot accordingly. (ii) In another figure, plot the mean-squared-error (MSE) of both the training and the test data as a function of ln(lambda).

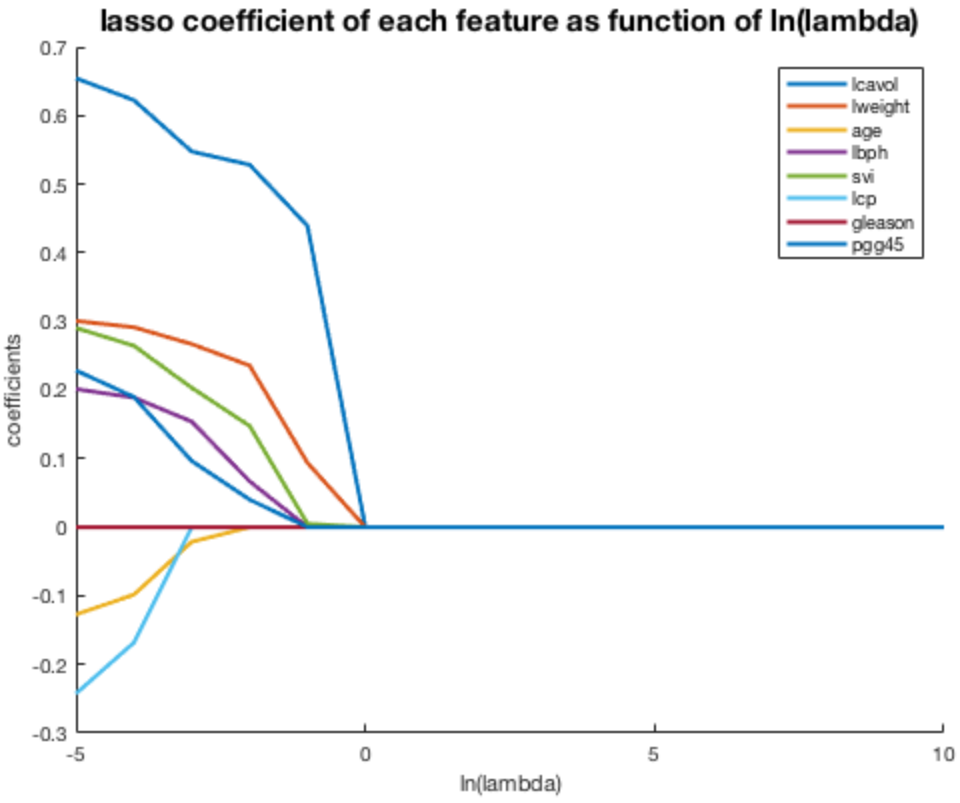
```
% clear and load data
clear; clc;
load('prostateStnd.mat');

% center training data
muXtrain = mean(Xtrain);
xTrainTilda = Xtrain - muXtrain;
% center test data
muXtest = mean(Xtest);
xTestTilda = Xtest - muXtest;

% ridge weights
```

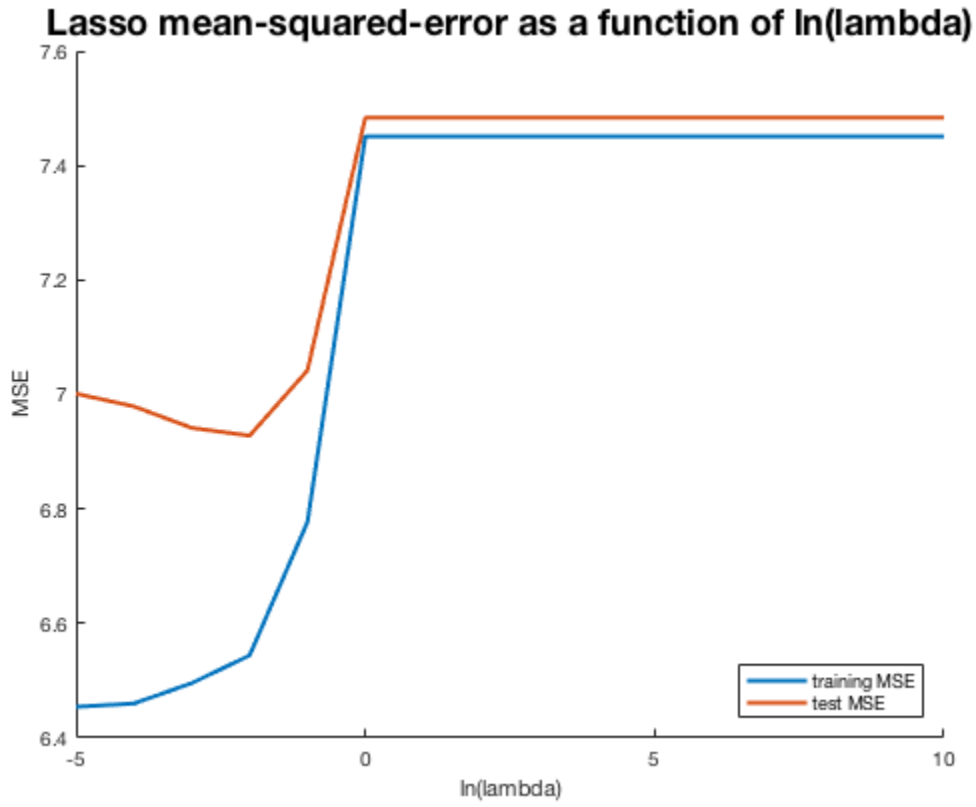
```
lambda = exp(-5:10);
W_ridge = cell(1,length(lambda));
coeff_ridge = zeros(length(W_ridge),size(xTrainTilda,2));
HtrainRidge = cell(1,length(lambda));
HtestRidge = cell(1,length(lambda));
ridgeTrainMSE = zeros(1,length(lambda));
ridgeTestMSE = zeros(1,length(lambda));

% ridge estimates and MSEs to compare against lasso
for i=1:length(lambda)
    W_ridge{i} = ridge(ytrain,xTrainTilda,lambda(i));
    coeff_ridge(i,:) = W_ridge{i};
    HtrainRidge{i} = xTrainTilda*W_ridge{i};
    HtestRidge{i} = xTestTilda*W_ridge{i};
    ridgeTrainMSE(i) = mse(ytrain,HtrainRidge{i});
    ridgeTestMSE(i) = mse(ytest,HtestRidge{i});
end
```

Cylic coordinate descent

```
% initialize lasso weights to W_ridge
W_lasso = W_ridge;
% max iterations
tmax = 100;
% keep track of previous value for convergence
W_k_past = W_lasso;
numIterations = zeros(1,length(lambda));
coeffs = zeros(16,8); % same as W_lasso, rearranged for convenience
 later
% estimates
H_train = cell(1,length(lambda));
H_test = cell(1,length(lambda));
% MSEs
trainMSE = zeros(1,length(lambda));
testMSE = zeros(1,length(lambda));
% tuning parameter
for i=1:length(lambda)
    d = length(W_lasso{1});
    for t=1:tmax % interations
        for k=1:d % cycle through each coordinate
            % argmin(z) of cost()
            a_k = 2*sum(xTrainTilda(:,k).^2);
            c_k = 0;
            n = length(xTrainTilda);
            for j=1:n
                c_k = c_k + (2*xTrainTilda(j,k)*(ytrain(j)- ...
                    W_lasso{i}'*xTrainTilda(j,:)'+ ...
                    W_lasso{i}(k)*xTrainTilda(j,k)));
            end
            W_lasso{i}(k) = wthresh(c_k/a_k,'s',lambda(i));
        end
        % check for convergence
        if W_lasso{i} == W_k_past{i}
```

```
            break;
        end
        W_k_past{i} = W_lasso{i};
    end
    numIterations(i) = t;
    coeffs(i,:) = W_lasso{i};
    H_train{i} = xTrainTilda*W_lasso{i};
    H_test{i} = xTestTilda*W_lasso{i};
    trainMSE(i) = mse(ytrain,H_train{i});
    testMSE(i) = mse(ytest,H_test{i});
end
%disp(numIterations);

% (i) plot lasso coeff. of each feat. as fn. of ln(lambda)
figure(1);
hold on;
for m=1:size(coeffs,2)
plot(-5:10, coeffs(:,m), 'LineWidth',2);
end
xlabel('ln(lambda)');
ylabel('coefficients');
title('lasso coefficient of each feature as function of
 ln(lambda)', ...
    'FontSize', 15);
legend(names(1:end-1));

% (ii) plot MSE for training and test as function on ln(lambda)
figure(2);
hold on;
plot(-5:10,trainMSE, 'LineWidth',2);
plot(-5:10,testMSE, 'LineWidth',2);
title('Lasso mean-squared-error as a function of ln(lambda)', ...
    'FontSize', 18);
xlabel('ln(lambda)');
ylabel('MSE');
legend('training MSE', 'test MSE', 'Location','southeast');
```


lasso coefficient of each feature as function of ln(lambda)
lcavol
lweight
age
lbph
svi
lcp
gleason
pgg45
coefficients
ln(lambda)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-5
0
5
10


Lasso mean-squared-error as a function of ln(lambda)
training MSE
test MSE
MSE
ln(lambda)
7.6
7.4
7.2
7
6.8
6.6
6.4
-5
0
5
10

(b)

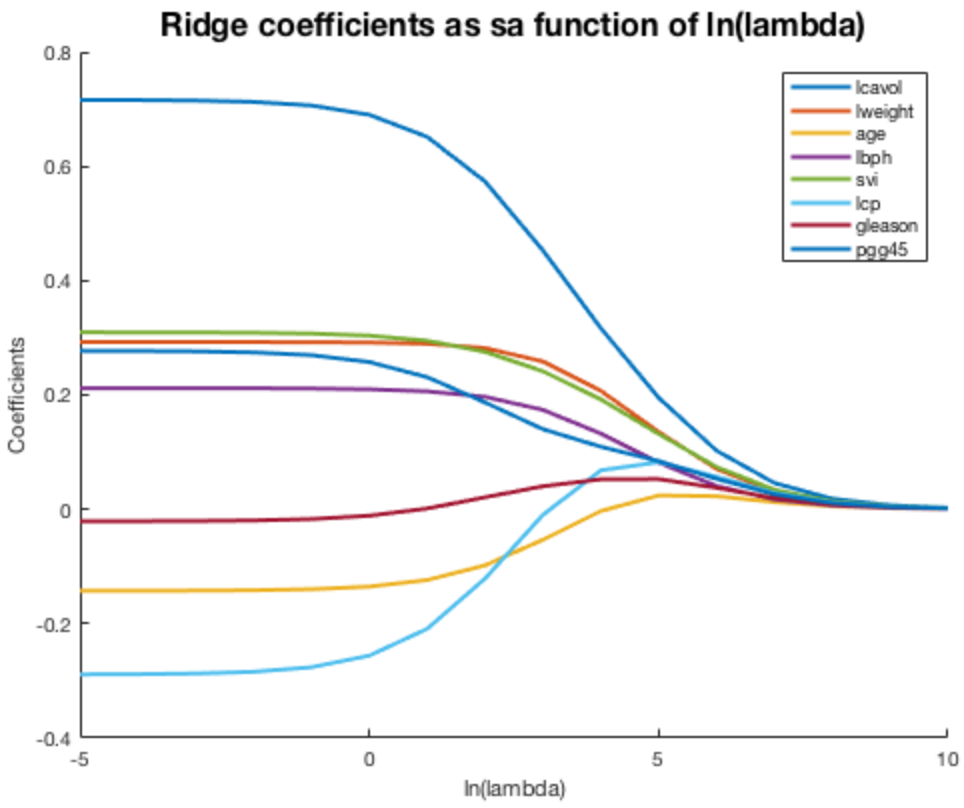
What happens to the lasso coefficients as lambda gets larger? What are the 2 most meaningful f eatures with lasso (last 2 features to converge)? How can this information be used?

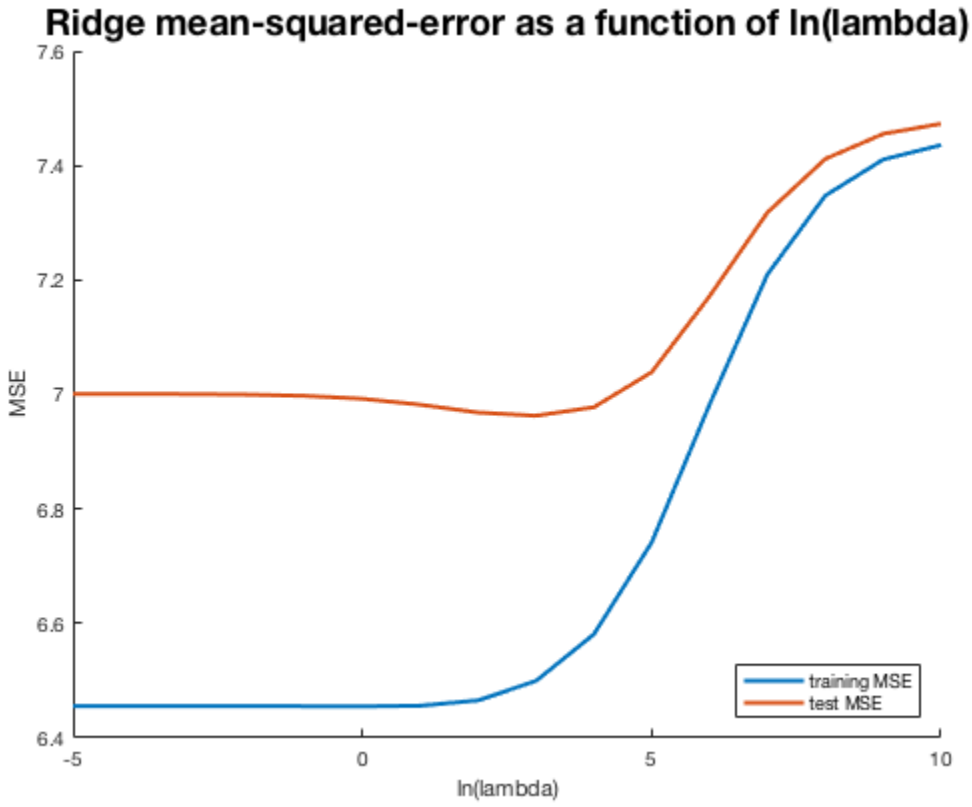
```
% As the tuning parameter increases the cost function this term begins
 to
% outway the first term which decreases due to increased flexibility.
 This
% causes the coefficients to move toward zero.

% From the graph, it looks like 'lcavol' and 'lweight' are the last
% features to converge. You can use subset selection to model on the
 most
% important features that were found. This reduces the dimensionality
 of
% the data which will alleviate exponential growth as a function of
% dimensions for many models such as K-nearest-neighbors.
```

(c)

Use MATLAB's ridge regression on this dataset. (i) Again, plot the ridge coefficients of each feature (8 total - yielding 8 curves) against ln(lambda) for values of ln(lambda) ranging from -5 to 10 in steps of 1. Label the plot accordingly. (ii) in another figure, plot the mean-squared error (MSE) of both the training and testing data.

```
% (i) plot ridge coefficients against ln(lambda)
figure(3);
hold on;
for k=1:size(coeff_ridge,2)
    plot(-5:10,coeff_ridge(:,k), 'LineWidth',2);
end
title('Ridge coefficients as sa function of ln(lambda)', ...
    'FontSize', 16);
xlabel('ln(lambda)');
ylabel('Coefficients');
legend(names(1:end-1));

% (i) plot ridge mean-squared-error for train/test

figure(4);
hold on;
plot(-5:10,ridgeTrainMSE, 'LineWidth',2);
plot(-5:10,ridgeTestMSE, 'LineWidth',2);
title('Ridge mean-squared-error as a function of ln(lambda)', ...
    'FontSize', 18);
xlabel('ln(lambda)');
ylabel('MSE');
legend('training MSE', 'test MSE', 'Location','southeast');
```


Ridge coefficients as sa function of ln(lambda)
Coefficients
ln(lambda)
lcavol
lweight
age
lbph
svi
lcp
gleason
pgg45
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-5
0
5
10


Ridge mean-squared-error as a function of ln(lambda)
MSE
ln(lambda)
training MSE
test MSE
7.6
7.4
7.2
7
6.8
6.6
6.4
-5
0
5
10

(d)

What happens to the ridge coefficients as lambda becomes larger? How is this different from lasso?

```
% The ridge coefficients also shrink as a function of lambda. The
% difference compared to lasso though, is that the ridge coefficients
 seem
% to approach zero, but not quite get there. I believe this means that
 it
% does not create sparsity so it cannot be used for feature selection
 like
% in the case of lasso.
```

Published with MATLAB® R2017a