Table of Contents

ECE 532: HW 4

by Roumen Guha

```
close all
clear all
```

1. Tikhonov regularization.

Sometimes we have competing objectives. For example, we want to find a w that minimizes `|y - Xw|_2` (least-squares), but we also want the weights w to be small. One way to achieve a compromise is to solve the following problem:

```
minimize (||y - Xw||_2)^2 + lambda (||w||_2)^2          (1)
```

where lambda > 0 is a paramater we choose that determines the relative weight we want to assign to each objective. This is called Tikhonov regularization (also known as L_2 regularization).

(a) Solve the optimization problem (1) by finding an expression for the minimizer wHat.

HINT: one approach is to reformulate (1) as a modified least-squares problem with different "X" and "y" matrices. Another approach is to use the vector derivative method we saw in class.

ECE 532

HW4

Rowmen Guha

1(a)

$$\|y - X\underline{w}\|_2^2 + \lambda \|\underline{w}\|_2^2$$

$$= \left\| \begin{bmatrix} y - X\underline{w} \\ \sqrt{\lambda}\, \underline{w} \end{bmatrix} \right\|^2$$

$$= \left\| \begin{bmatrix} y \\ 0 \end{bmatrix} - \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \right\|^2$$

$$\Rightarrow \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \hat{\underline{w}} = \begin{bmatrix} y \\ \underline{0} \end{bmatrix} \Rightarrow B\,\hat{\underline{w}} = \underline{z}$$

$$\begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}^T \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \hat{\underline{w}} = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}^T \begin{bmatrix} y \\ \underline{0} \end{bmatrix},$$

$$(X^T X + \lambda I)\,\hat{\underline{w}} = X^T y$$

(b) $X \in \mathbb{R}^{n \times p}$ (n rows, p columns)

$n < p \Rightarrow$ more unknowns than equations

"underdetermined"

There is always going to be a unique solution to this problem.

$\Rightarrow$

$A^T A \succeq 0$ (PSD)

and $\lambda I \succ 0$ (PD)

so $A^T A + \lambda I \succ 0$, hence it will always be invertible.

(b) Suppose that X is n x p, with n < p.

Is there a unique least-squares solution? Is there a unique solution to (1)? Explain your answers.

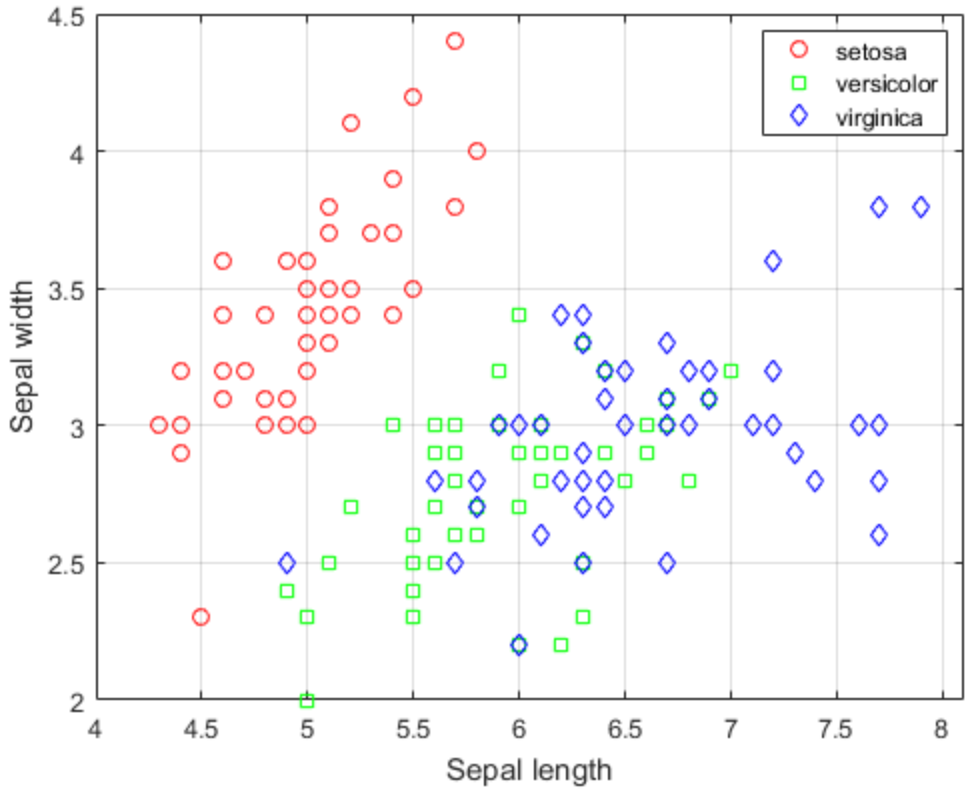
2.

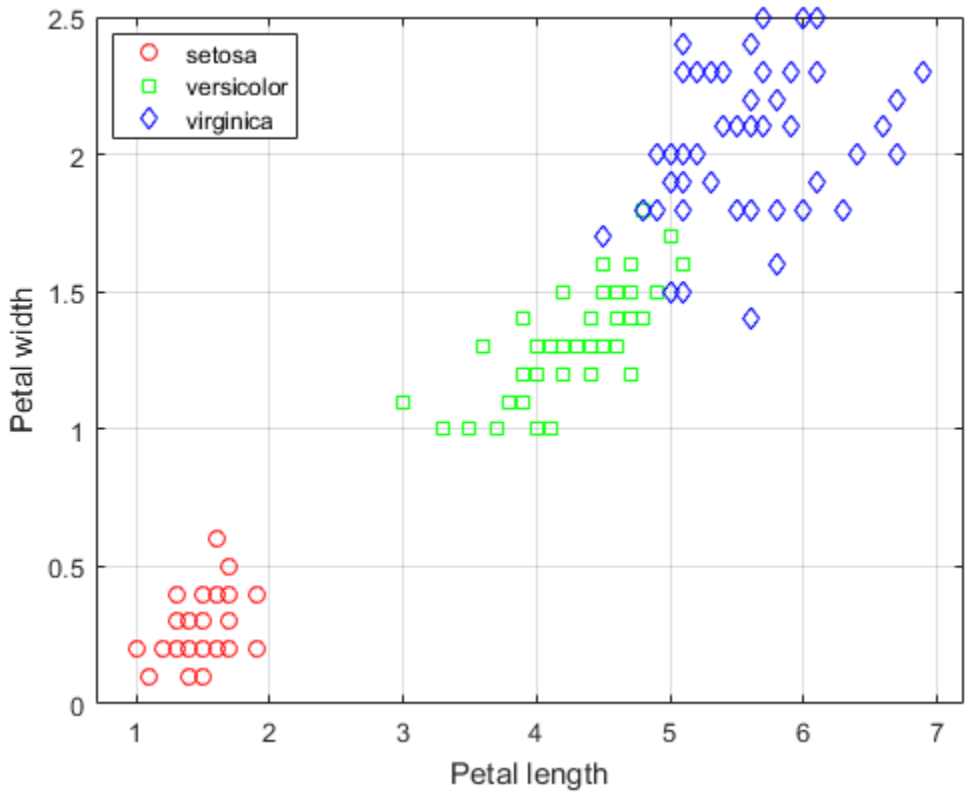
In 1936 Ronald Fisher published a famous paper on classification titled "The use of multiple measurements in taxonomic problems." In the paper, Fisher studied the problem of classifying iris flowers based on measurements of the sepal and petal widths and lengths.

The dataset consists of 50 examples of three types of iris flowers. The sepal and petal measurements can be used to classify the examples into the three types of flowers.

https://www.mathworks.com/help/stats/examples/classification.html

```
clear all
close all

load fisheriris

figure
gscatter(meas(:,1), meas(:,2), species,'rgb','osd');
xlabel('Sepal length');
ylabel('Sepal width');
grid;

figure
gscatter(meas(:,3), meas(:,4), species,'rgb','osd');
xlabel('Petal length');
ylabel('Petal width');
grid;
```


setosa
versicolor
virginica
Sepal width
Sepal length
4.5
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8


setosa
versicolor
virginica
Petal width
Petal length
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7

(a) Formulate the classification task as a least-squares problem.

Least-squares will produce real-valued predictions, not discrete labels or categories. What might you do to address this issue?

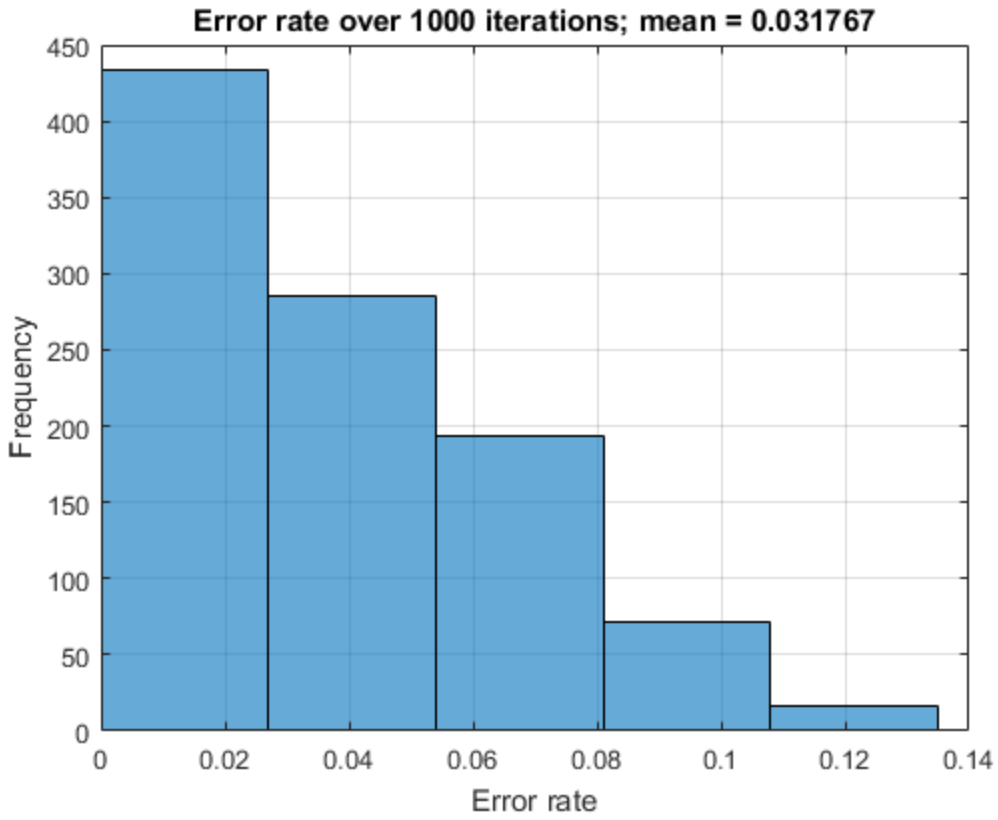
```
% The labels were generated by looking at 'meas' and noticing that
% flowers of identical type were grouped together.

% Looking at the species list, for the labels, we'll use setosa = -1,
% versicolor = 0, and virginica = 1.

y = [-1*ones(50,1);
    zeros(50,1);
    ones(50,1)];

% Solve the least-squares approximation of yHat.
yHat = meas*(meas\y);

% We can round the real-valued predictions to their nearest labels.
% Classify the real-valued yHat into a labelled yHat.
for k = 1:length(yHat)
    if  abs(yHat(k)) < 0.5
            yHat(k) = 0;
    else
            yHat(k) = sign(yHat(k));
    end
end
```

(b) Write a program to "train" a classifier.

Use LS based on 40 labelled examples of each of the three flower types, then test the performance of your classifier using the remaining 10 examples from each type. Repeat this with many different randomly chosen subsets of training and test. What is the average test error (number of mistakes divided by 30)?

```
numTrials = 1000;
errors = zeros(numTrials, 1);
trainingSetSize = 40;
totalDataSetSize = 50;

for k = 1:numTrials

    % For each flower type
    randomIndices = randperm(totalDataSetSize);
    trainingIndices = randomIndices(1:trainingSetSize);
    holdoutIndices = setdiff(randomIndices, trainingIndices);

    % Corresponding indices for all flower types
    trainingData = [trainingIndices, trainingIndices +
 totalDataSetSize, trainingIndices + 2*totalDataSetSize];
    holdoutData = [holdoutIndices, holdoutIndices + totalDataSetSize,
 holdoutIndices + 2*totalDataSetSize];
```

```
    % Grab data for training the classifier
    X_classifier = meas(trainingData, :);
    y_classifier = y(trainingData);

    % Grab holdout data for testing
    X_test = meas(holdoutData, :);
    y_test = y(holdoutData);

    % Use least-squares to find optimal weights
    wHat = X_classifier\y_classifier;

    % Use optimal weights to label holdout data
    yHat = X_test*wHat;

    % Classify the output
    for l = 1:length(yHat)
        if  abs(yHat(l)) < 0.5
            yHat(l) = 0;
        else
            yHat(l) = sign(yHat(l));
        end
    end

    % Compute error
    errors(k) = mean(y_test ~= yHat);
end

% Compute average error
averageTestError = mean(errors);

% Plot frequency of error rates
figure
histogram(errors, 5)
xlabel('Error rate')
ylabel('Frequency')
title(['Error rate over ', num2str(numTrials),' iterations; mean = ',
 num2str(averageTestError)])
grid;
```

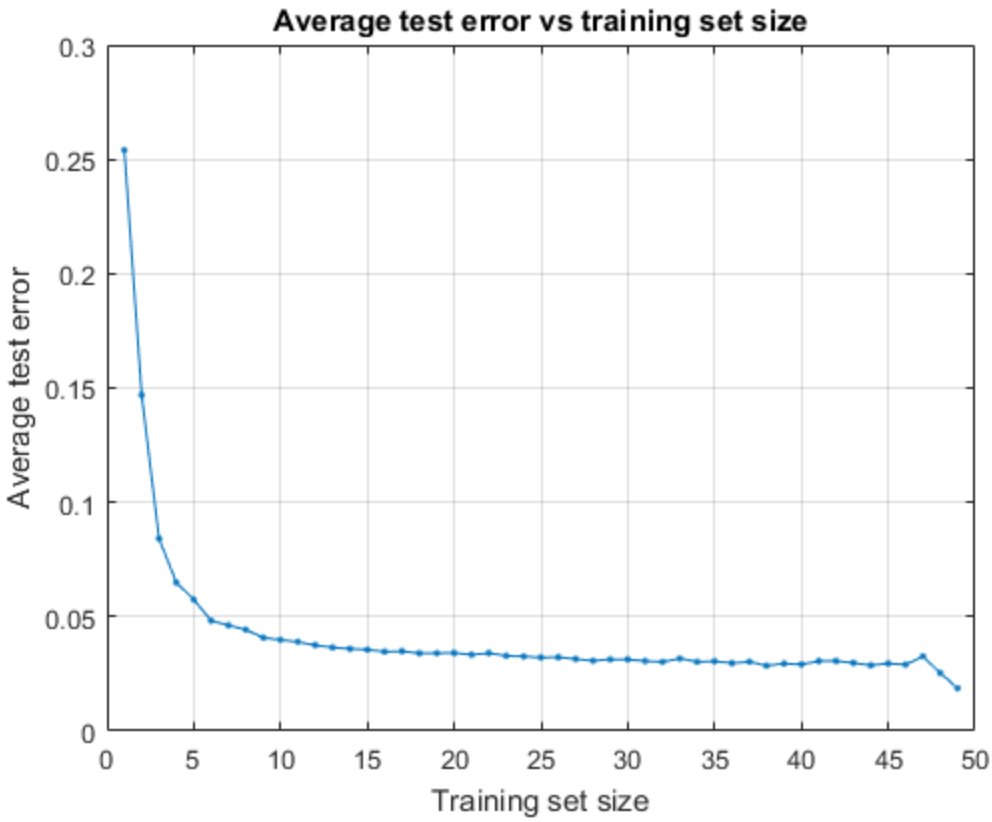


(c) Experiment with even smaller sized training sets.

Clearly we need at least one training example from each type of flower. Make a plot of average test error as a function of training set size.

```
averageTestErrors = zeros(totalDataSetSize, 1);

for trainingSetSize = 1:totalDataSetSize

    for k = 1:numTrials

        % For each flower type
        randomIndices = randperm(totalDataSetSize);
        trainingIndices = randomIndices(1:trainingSetSize);
        holdoutIndices = setdiff(randomIndices, trainingIndices);

        % Corresponding indices for all flower types
        trainingData = [trainingIndices, trainingIndices +
totalDataSetSize, trainingIndices + 2*totalDataSetSize];
        holdoutData = [holdoutIndices, holdoutIndices +
totalDataSetSize, holdoutIndices + 2*totalDataSetSize];

        % Grab data for training the classifier
        X_classifier = meas(trainingData, :);
```

```
            y_classifier = y(trainingData);

            % Grab holdout data for testing
            X_test = meas(holdoutData, :);
            y_test = y(holdoutData);

            % Use least-squares to find optimal weights
            wHat = X_classifier\y_classifier;

            % Use optimal weights to label holdout data
            yHat = X_test*wHat;

            % Classify the output
            for l = 1:length(yHat)
                if  abs(yHat(l)) < 0.5
                    yHat(l) = 0;
                else
                    yHat(l) = sign(yHat(l));
                end
            end

            % Compute error
            errors(k) = mean(y_test ~= yHat);
        end

        % Compute average error
        averageTestErrors(trainingSetSize) = mean(errors);
    end

    % Plot average errors
    figure
    plot(averageTestErrors, '.-')
    xlabel('Training set size')
    ylabel('Average test error')
    title('Average test error vs training set size')
    grid;
```

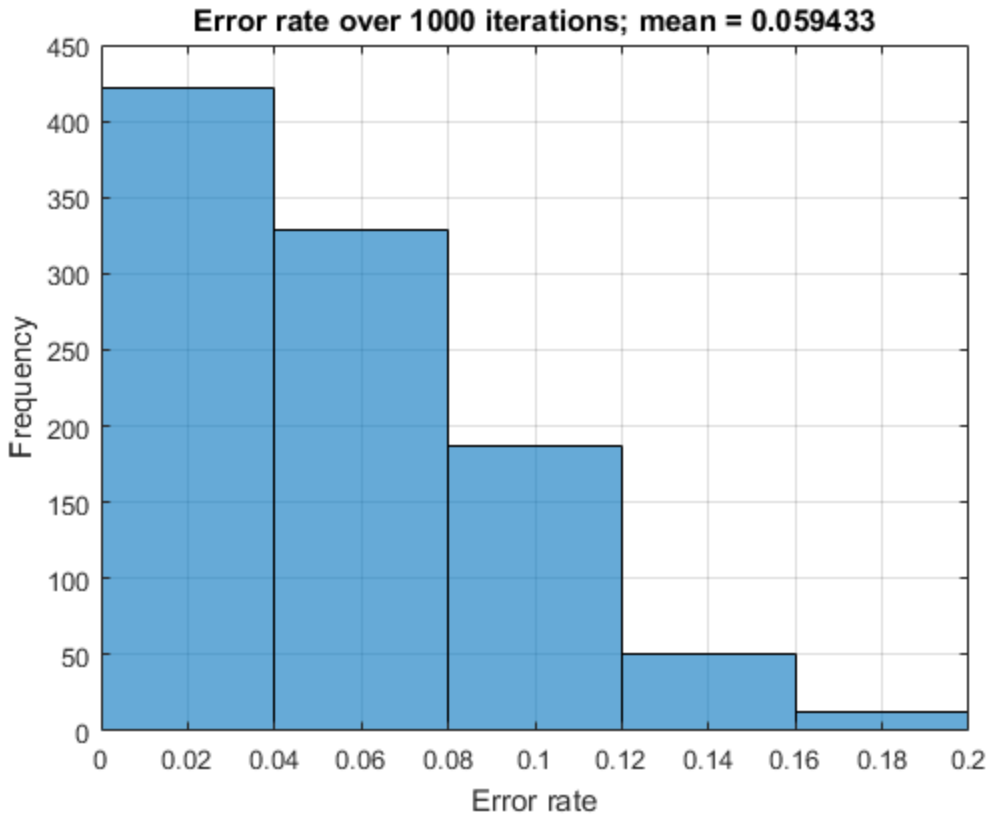


(d) Now design a classifier using only the first three measurements

(sepal length, sepal width, and petal length). What is the average test error in this case?

```
trainingSetSize = 40;

for k = 1:numTrials

    % For each flower type
    randomIndices = randperm(totalDataSetSize);
    trainingIndices = randomIndices(1:trainingSetSize);
    holdoutIndices = setdiff(randomIndices, trainingIndices);

    % Corresponding indices for all flower types
    trainingData = [trainingIndices, trainingIndices +
 totalDataSetSize, trainingIndices + 2*totalDataSetSize];
    holdoutData = [holdoutIndices, holdoutIndices + totalDataSetSize,
 holdoutIndices + 2*totalDataSetSize];

    % Grab data for training the classifier
    X_classifier = meas(trainingData, 1:3);
    y_classifier = y(trainingData);

    % Grab holdout data for testing
```

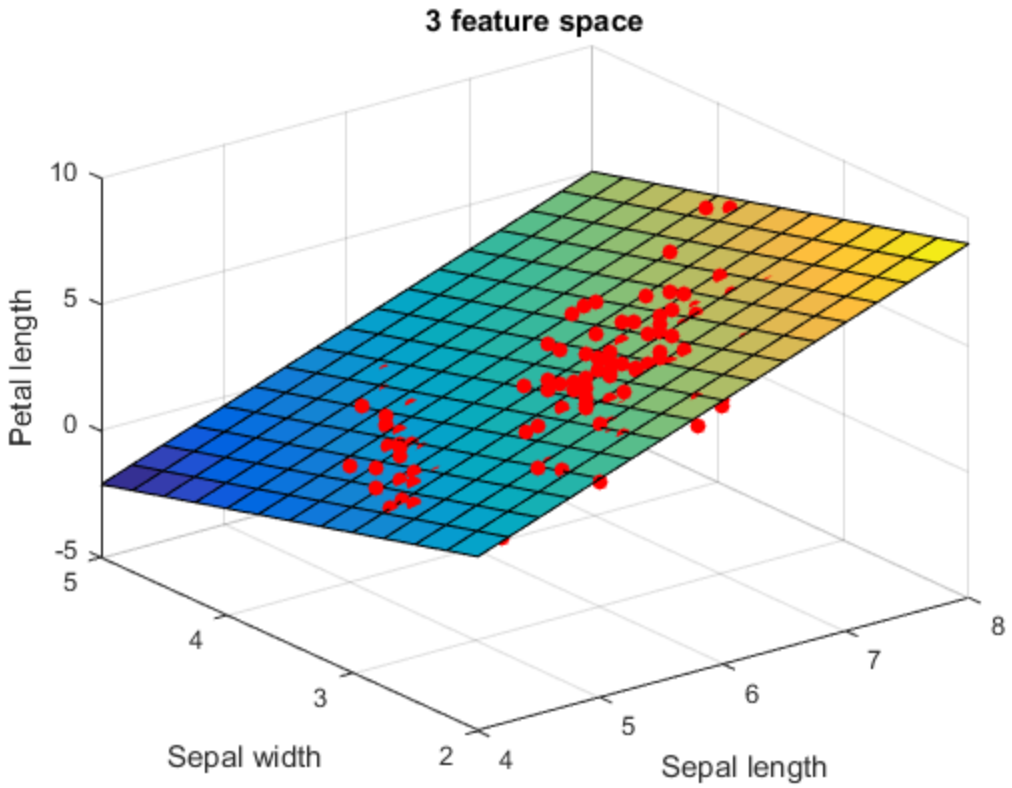
```
    X_test = meas(holdoutData, 1:3);
    y_test = y(holdoutData);

    % Use least-squares to find optimal weights
    wHat = X_classifier\y_classifier;

    % Use optimal weights to label holdout data
    yHat = X_test*wHat;

    % Classify the output
    for l = 1:length(yHat)
        if  abs(yHat(l)) < 0.5
            yHat(l) = 0;
        else
            yHat(l) = sign(yHat(l));
        end
    end

    % Compute error
    errors(k) = mean(y_test ~= yHat);
end

% Compute average error
averageTestError = mean(errors);

% Plot frequency of error rates
figure
histogram(errors, 5)
xlabel('Error rate')
ylabel('Frequency')
title(['Error rate over ', num2str(numTrials),' iterations; mean = ',
 num2str(averageTestError)])
grid;
```



(e) Use a 3D scatter plot to visualize the measurements in (d).

Can you find a 2D subspace that the data approximately lie in? You can do this by rotating the plot and looking for a plane that approximately contains the data points.

```
[xx,yy] = meshgrid(4:.25:8, 2:.25:5);

C = x2fx(meas(:,1:2), 'linear') \ meas(:,3);
zz = x2fx([xx(:) yy(:)], 'linear') * C;
zz = reshape(zz, size(xx));

figure
scatter3(meas(:,1), meas(:,2), meas(:,3), 'red', 'filled')
surface(xx,yy,zz)
xlabel('Sepal length')
ylabel('Sepal width')
zlabel('Petal length')
title('3 feature space')
```

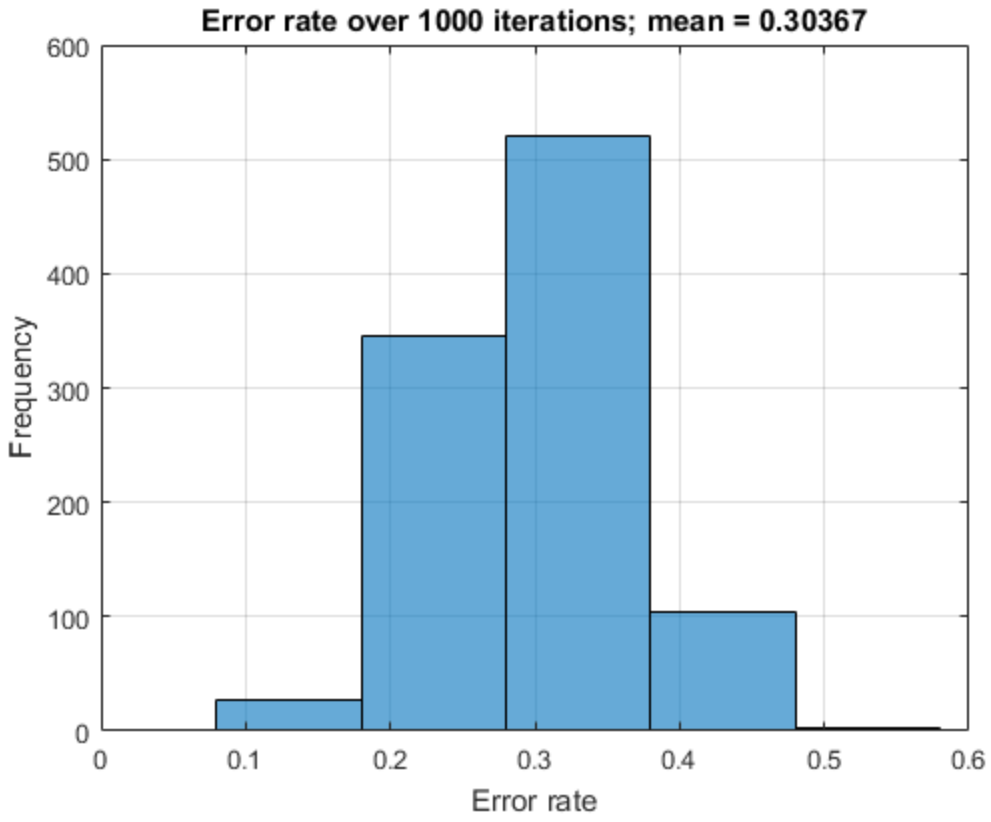


(f) Use this subspace to find a 2D classification rule.

What is the average test error in this case?

```
[coeff,score,roots] = pca(meas(:, 1:3));
basis = coeff(:,1:2);
normal = coeff(:,3);

trainingSetSize = 40;

for k = 1:numTrials

    % For each flower type
    randomIndices = randperm(totalDataSetSize);
    trainingIndices = randomIndices(1:trainingSetSize);
    holdoutIndices = setdiff(randomIndices, trainingIndices);

    % Corresponding indices for all flower types
    trainingData = [trainingIndices, trainingIndices +
 totalDataSetSize, trainingIndices + 2*totalDataSetSize];
    holdoutData = [holdoutIndices, holdoutIndices + totalDataSetSize,
 holdoutIndices + 2*totalDataSetSize];

    % Grab data for training the classifier
```

```
    X_classifier = meas(trainingData, 1:2);
    y_classifier = y(trainingData);

    % Grab holdout data for testing
    X_test = meas(holdoutData, 1:2);
    y_test = y(holdoutData);

    % Use least-squares to find optimal weights
    wHat = X_classifier\y_classifier;

    % Use optimal weights to label holdout data
    yHat = X_test*wHat;

    % Classify the output
    for l = 1:length(yHat)
        if  abs(yHat(l)) < 0.5
            yHat(l) = 0;
        else
            yHat(l) = sign(yHat(l));
        end
    end

    % Compute error
    errors(k) = mean(y_test ~= yHat);
end

% Compute average error
averageTestError = mean(errors);

% Plot frequency of error rates
figure
histogram(errors, 5)
xlabel('Error rate')
ylabel('Frequency')
title(['Error rate over ', num2str(numTrials),' iterations; mean = ',
 num2str(averageTestError)])
grid;
```



3.

In this problem you will work to analyze the dataset hw4_532_jesterdata.mat. The dataset contains an m = 100 by n = 7200 dimensional matrix X. Each row of X corresponds to a joke, and each column corresponds to a user. Each of the users rated the quality of each joke on a scale of [-10,10].

```
clear all
close all

load('hw4_532_jesterdata.mat')
```

(a) Suppose that you work for a company that makes joke recommendations to customers.

You are given a large dataset X of jokes and ratings. It contains n reviews for each of m jokes. The reviews were generated by n users who represent a diverse set of tastes. Each reviewer rated every movie on a scale of [-10,10]. A new customer has rated k = 25 of the jokes, and the goal is to predict another joke that the customer will like based on her k ratings. Use the first n = 20 columns of X for this prediction problem (so that the problem is overdetermined). Her ratings are contained in the file 'hw4_532_newuser.mat', in a vector b. The jokes she didn't rate are indicated by a (false) score of -99. Compare your predictions to her complete set of ratings, contained in the vector trueb. Her actual favorite joke was number 29. Does it seem like your predictor is working well?

```
load('hw4_532_newuser.mat')
```

```
X_first20 = X(:, 1:20);

X_cleaned20 = X_first20(b ~= -99, :);

bHat = X_first20 * inv(X_cleaned20' * X_cleaned20) * X_cleaned20' *
 b(b ~= -99);

b(29)
trueb(29)
bHat(29)

errorRate = sum((bHat - trueb).^2)
```

```
ans =

   -99


ans =

    6.2600


ans =

    3.5155


errorRate =

   2.2301e+03
```

(b) Repeat the prediction problem above, but this time use the entire X matrix.

Note that now the problem is underdetermined. Explain how you will solve this prediction problem and apply it to the data. Does it seem like your predictor is working? How does it compare to the first method based on only 20 users?

```
% It seems like this is an ideal use case for the Tikhonov
 regularization.
% We will use it in an attempt to minimize (||w||_2)^2 while also
 trying to
% minimize (||y - Xw||_2)^2.
%
%      wHat = argmin(w)     (||y - Xw||_2)^2 + lambda*(||w||_2)^2

X_cleaned = X(b ~= -99, :);
b_cleaned = b(b ~= -99);
```

```
lambda = 50;

bTilde = [b_cleaned;
          zeros(7200,1)];
XTilde = [X_cleaned;
           lambda*eye(7200)];

wHat = inv(X_cleaned' * X_cleaned + lambda * eye(size(X_cleaned, 2)))
 * X_cleaned' * b_cleaned;

bHat = X * wHat;

errorRate = sum((bHat - trueb).^2)
```

```
errorRate =

  915.7955
```

(c) Propose a method for finding one other user that seems to give the best predictions for the new user.

How well does this approach perform? Now try to find the best two users to predict the new user.

```
% It would be reasonable to think that we can use a user who's given
% similar ratings to predict ratings for our new user.

totalSum = sum((X_cleaned - b_cleaned * ones(1, size(X_cleaned,
 2))).^2);

[v1, in1] = min(totalSum);
[v2, in2] = min([totalSum(1:in1 - 1), totalSum(in1 + 1:end)]);

if in2 > in1
    in2 = in2 + 1;
end

closestUsers = [X(:, in1), X(:, in2)];
closestUsers_cleaned = closestUsers(b ~= -99, :);

bHat = closestUsers * inv(closestUsers_cleaned' *
 closestUsers_cleaned) * closestUsers_cleaned' * b_cleaned;

errorRate = sum((bHat - trueb).^2)
```

```
errorRate =
```

```
1.5231e+03
```

(d) Use the MATLAB function svd with the 'economy size' option to compute the SVD of X = UEV'.

Plot the spectrum of X. What is the rank of X? How many dimensions seem important? What does this tell us about the jokes and the users?

```
[U, S, V] = svd(X, 0);
```

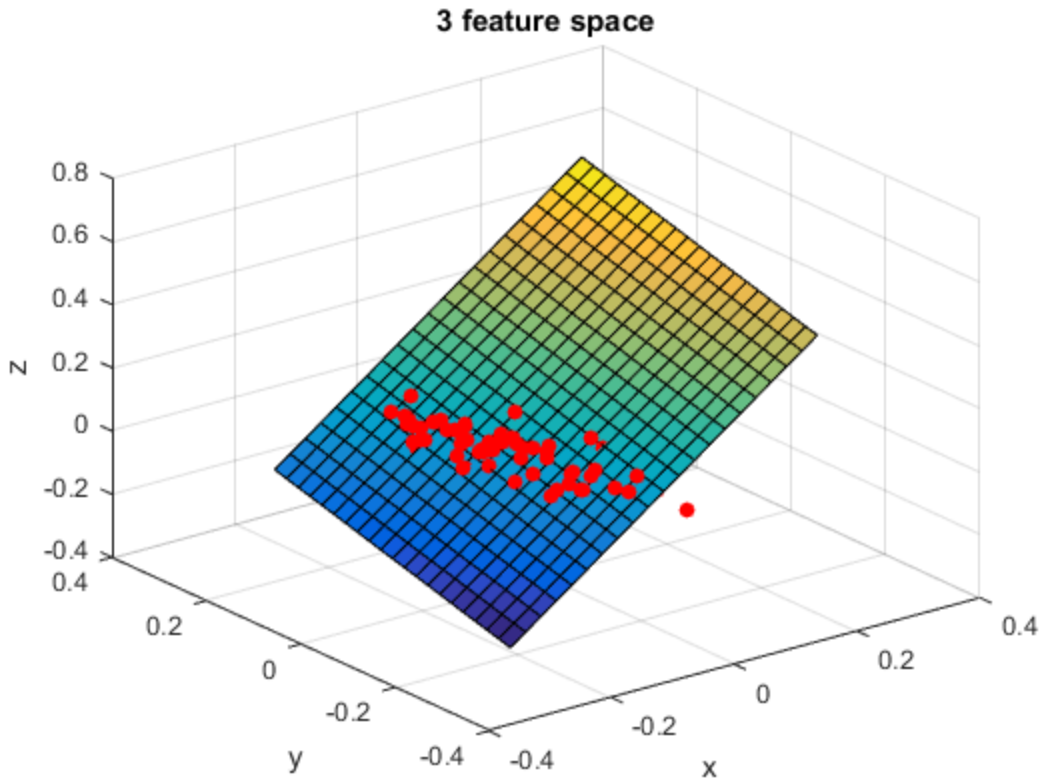
(e) Visualize the dataset by projecting the columns and rows on to the first three principle component directions.

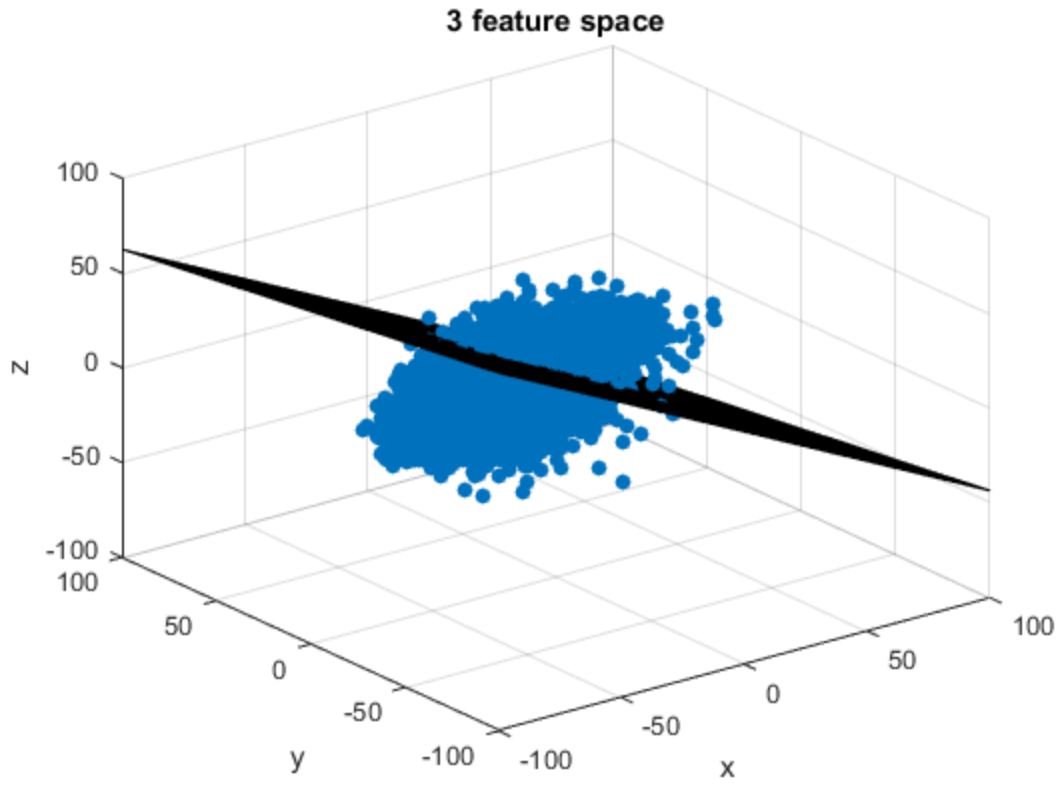
Use the rotate tool in the MATLAB plot to get different views of the 3D projections. Discuss the structure of the projections and what it might tell us about the jokes and the users.

```
[xx,yy] = meshgrid(-0.25:.025:0.25, -0.25:.025:0.25);

C = x2fx(U(:,1:2), 'linear') \ U(:,3);
zz = x2fx([xx(:) yy(:)], 'linear') * C;
zz = reshape(zz, size(xx));

figure
scatter3(U(:,1), U(:,2), U(:,3), 'red', 'filled')
surface(xx,yy,zz)
xlabel('x')
ylabel('y')
zlabel('z')
title('3 feature space')

S(1:3, 1:3)

projectedX = U(:, 1:3)' * X;
[xx,yy] = meshgrid(-100:10:100, -100:10:100);

C = x2fx(projectedX(:,1:2), 'linear') \ projectedX(:,3);
zz = x2fx([xx(:) yy(:)], 'linear') * C;
zz = reshape(zz, size(xx));

figure
scatter3(projectedX(1,:), projectedX(2,:), projectedX(3,:), 'filled')
surface(xx,yy,zz)
xlabel('x')
ylabel('y')
zlabel('z')
```

```
title('3 feature space')
```

```
ans =

   1.0e+03 *

    2.4201         0         0
         0    1.3632         0
         0         0    0.8860
```





(f) One easy way to compute the first principle component for large datasets like this is the so-called power method.

Explain the power method and why it works. Write your own code to implement the power method in MATLAB and use it to compute the first column of U and V in the SVD of X. Does it produce the same result as MATLAB's built-in svd function?

```
% The power method is based on the idea that any random vector is
 going to
% have some component of it that can be projected upon some other
 given basis.
% So, by computing the projection of the vector onto the vector we'd
 like,
% and normalizing it, and then repeating this process over and over,
 that
% we will eventually converge upon the goal vector.

% For the first right singular vector
A = X' * X;
firstRightSingularVector = power_iteration(A, 20);

% Note that it seems that firstRightSingularVector and V(:,1) point in
 opposite
```

```
% directions; one is equal to the other scaled by -1.
% It's very impressive that it converges so quickly.

errorRate = sum(V(:,1) + firstRightSingularVector)
```

```
errorRate =

   7.6968e-09
```

(g) The power method is based on an initial starting vector.

Give one example of a starting vector for which the power method will fail to find the first left and right singular vectors in this problem.

```
% If the given starting vector was completely orthogonal to the basis
 we
% are trying to project it onto, this will fail because there is no
% component of that vector that would lie within our basis.

% (Also, in the trivial case that our starting vector was the zero
 vector, this
% method would obviously fail)
```

Published with MATLAB® R2016b

```
function [vector, value] = power_iteration(A, num_iterations, start)
% Power method for computing eigenvalues

if nargin < 3
    start = 0;
end

if nargin < 2
    num_iterations = 1000000000000000;
end

if nargin < 1
    error('Matrix A is a required input!')
    return;
elseif start == 0
    [n,m] = size(A);
    if n ~= m
        disp('Matrix A is not square!');
        return;
    end

    start = rand(n,1);
end

x = start;

while num_iterations > 0
    x = A*x;
    x = x/norm(x);
    num_iterations = num_iterations - 1;
end
vector = x;
value = norm(x);

end

```