CSCC11 Fall 2015
Assignment 2

Student Name:	Maoting Zhang
Student Number:	999590633
UTORid(mathlab username):	zhangmao

Note: please see mathlab submission for a colored version of the report. Thanks!

- Matlab Scripts & functions
    - knnClassify.m

```
function class = knnClassify(test, k, trainingInputs, trainingTargets)
%
% Inputs:
%   test: test input vector as a row vector
%   k: number of nearest neighbours to use in classification.
%   traingingInputs: array of training exemplars, one exemplar per row
%   traingingTargets: idenicator vector per row
%
% Basic Algorithm of kNN Classification
% 1) find distance from test input to each training exemplar,
% 2) sort distances
% 3) take smallest k distances, and use the median class among
%    those exemplars to label the test input.
%


% YOUR CODE GOES HERE.
n = size(trainingInputs, 1);

% allocates matrix to store distances and their indexes
D = ones(n, 2);

for i  = 1:n
    distance = sum((trainingInputs(i, :) - test) .^ 2)^(1/2);
    D(i, :) = [distance i];
end

sorted = sortrows(D);

nearestMedian = sorted(floor(k/2), 2);

class = trainingTargets(nearestMedian, :);
```

- knnStarter.m

```
% Use the function knnClassify to test performance on different datasets.
figure(1);clf;


load('a2/dataSets/generic1');
n1t = size(c1_train, 2);
n2t = size(c2_train, 2);
trainingInputs = [c1_train, c2_train];
trainingTargets = [ones(1, n1t) zeros(1, n2t); zeros(1, n1t) ones(1, n2t)];
```

```
n1 = size(c1_test, 2);
n2 = size(c2_test, 2);
testInputs = [c1_test, c2_test];
testTargets = [ones(1, n1) zeros(1, n2); zeros(1, n1) ones(1, n2)];

E = [];
for k = 3:2:21

    num_error = 0;

    for i = 1:(n1+n2)
        class = knnClassify(testInputs(:, i)', k, trainingInputs', trainingTargets');
        if class ~= testTargets(:, i)'
            num_error = num_error + 1;
        end
    end
    E = [E; k num_error/(n1+n2)];
end

plot(E(:, 1), E(:, 2), '--ms');
hold on;
% ==============================================================

load('a2/dataSets/generic2');

n1t = size(c1_train, 2);
n2t = size(c2_train, 2);
trainingInputs = [c1_train, c2_train];
trainingTargets = [ones(1, n1t) zeros(1, n2t); zeros(1, n1t) ones(1, n2t)];

n1 = size(c1_test, 2);
n2 = size(c2_test, 2);
testInputs = [c1_test, c2_test];
testTargets = [ones(1, n1) zeros(1, n2); zeros(1, n1) ones(1, n2)];

E = [];
for k = 3:2:21

    num_error = 0;

    for i = 1:(n1+n2)
        class = knnClassify(testInputs(:, i)', k, trainingInputs', trainingTargets');
        if class ~= testTargets(:, i)'
            num_error = num_error + 1;
        end
    end
    E = [E; k num_error/(n1+n2)];
end
```

```matlab
plot(E(:, 1), E(:, 2), '--bs');
hold on;

% ==============================================================

load('a2/dataSets/fruit_train');
load('a2/dataSets/fruit_test');

E = [];
for k = 3:2:21

    num_error = 0;

    for i = 1:size(inputs_test, 2)
        class = knnClassify(inputs_test(:, i)', k, inputs_train', target_train');
        if class ~= target_test(:, i)'
            num_error = num_error + 1;
        end
    end
    E = [E; k num_error/size(inputs_test, 2)];
end

plot(E(:, 1), E(:, 2), '--go');
hold on;

% ==============================================================

load('a2/dataSets/mnist_train');
load('a2/dataSets/mnist_test');

E = [];
for k = 3:2:21

    num_error = 0;

    for i = 1:size(inputs_test, 2)
        class = knnClassify(inputs_test(:, i)', k, inputs_train', target_train');
        if class ~= target_test(:, i)'
            num_error = num_error + 1;
        end
    end
    E = [E; k num_error/size(inputs_test, 2)];
end

plot(E(:, 1), E(:, 2), '--yo');

legend('generic1', 'generic2', 'fruits', 'digits');
xlabel('k');
ylabel('test error');
title('Test Error of KNN as a function of k');
```

- ## learnGCCmodel.m

```
function [p1, m1, m2, C1, C2] = learnGCCmodel(x1, x2)
%
% Inputs
%   x1 - training exemplars from class 1, one exemplar per row
%   x2 - training exemplars from class 2, one exemplar per row
%
% Outputs
%   p1 - prior probability for class 1
%   m1 - mean of Gaussian measurement likelihood for class 1
%   m2 - mean of Gaussian measurement likelihood for class 2
%   C1 - covariance of Gaussian measurement likelihood for class 1
%   C2 - covariance of Gaussian measurement likelihood for class 2
%

% convert examplars to one exemplar per column
x1 = x1';
x2 = x2';

% number of training examplars
n1 = size(x1, 2);
n2 = size(x2, 2);

% do necessary transposes in order to get mean
m1 = sum(x1')' / n1;
m2 = sum(x2')' / n2;

C1 = cov(x1', 1);
C2 = cov(x2', 1);

p1 = n1 / n2;

end
```

- ## gccClassify.m

```
function class = gccClassify(x, p1, m1, m2, C1, C2)
%
% Inputs
%   x - test examplar
%   p1 - prior probability for class 1
%   m1 - mean of Gaussian measurement likelihood for class 1
%   m2 - mean of Gaussian measurement likelihood for class 2
%   C1 - covariance of Gaussian measurement likelihood for class 1
%   C2 - covariance of Gaussian measurement likelihood for class 2
%
% Outputs
%   class - sgn(a(x)) (ie sign of decision function a(x))



% YOUR CODE GOES HERE.
a = - (1/2) * transpose(x - m1) * inv(C1) * (x - m1) - (1/2) * log(det(C1)) ...
    + (1/2) * transpose(x - m2) * inv(C2) * (x - m2) + (1/2) * log(det(C2));

if a > 0
    class = [1 0];
else
    class = [0 1];
end
end
```

- gccStarter.m

```
% Use the function learnGCCmodel to learn a Gaussian Class-Conditional
% model for classification.
% Use the function gccClassify for evaluating the decision function
% to perform classification.
% Then test the model under various conditions.
figure(1);clf;

load('a2/dataSets/generic1');


x1 = c1_train';
x2 = c2_train';

[p1, m1, m2, C1, C2] = learnGCCmodel(x1, x2);

for i = 1:size(c1_test, 2)
    class = gccClassify(c1_test(:, i), p1, m1, m2, C1, C2);
    plot(c1_test(1, i),c1_test(2, i), 'bx');hold on;
    if class ~= [1 0]
        plot(c1_test(1, i),c1_test(2, i), 'ro', 'MarkerSize', 5);
    end
end

for i = 1:size(c2_test, 2)
    class = gccClassify(c2_test(:, i), p1, m1, m2, C1, C2)
    plot(c2_test(1, i),c2_test(2, i), 'go');hold on;
    if class ~= [0 1]
        plot(c2_test(1, i),c2_test(2, i), 'rs', 'MarkerSize', 5);
    end
end

xlabel('x');
```

```
ylabel('y');
legend('C1','C2');
title('GCC fitting generic1');

==============================================================
figure(2);clf;
load('a2/dataSets/generic2');


x1 = c1_train';
x2 = c2_train';

[p1, m1, m2, C1, C2] = learnGCCmodel(x1, x2);

for i = 1:size(c1_test, 2)
    class = gccClassify(c1_test(:, i), p1, m1, m2, C1, C2);
    plot(c1_test(1, i),c1_test(2, i), 'bx');hold on;
    if class ~= [1 0]
        plot(c1_test(1, i),c1_test(2, i), 'ro', 'MarkerSize', 5);
    end
end

for i = 1:size(c2_test, 2)
    class = gccClassify(c2_test(:, i), p1, m1, m2, C1, C2)
    plot(c2_test(1, i),c2_test(2, i), 'go');hold on;
    if class ~= [0 1]
        plot(c2_test(1, i),c2_test(2, i), 'rs', 'MarkerSize', 5);
    end
end

xlabel('x');
ylabel('y');
legend('C1','C2');
title('GCC fitting generic2');

% ==============================================================

figure(3);clf;

load('a2/dataSets/fruit_train');
load('a2/dataSets/fruit_test');

c1 = find(target_train(1,:)==1);
c2 = find(target_train(2,:)==1);
x1 = inputs_train(:,c1);
x2 = inputs_train(:,c2);

[p1, m1, m2, C1, C2] = learnGCCmodel(x1', x2');

for i = 1:size(inputs_test, 2)
```

```
    class = gccClassify(inputs_test(:, i), p1, m1, m2, C1, C2)
    if class == [0 1]
        plot(inputs_test(1, i),inputs_test(2, i), 'bx');hold on;
    else
        plot(inputs_test(1, i),inputs_test(2, i), 'gs');hold on;
    end

    if class ~= target_test(:, i)'
        plot(inputs_test(1, i),inputs_test(2, i), 'ro', 'MarkerSize', 5);
    end
end

xlabel('x');
ylabel('y');
legend('C1','C2');
title('GCC fitting fruits');
```

○ logisticNLP.m

```
function [ll, dll_dw, dll_db] = logisticNLP(x1, x2, w, b, alpha)
% [ll, dll_dw, dll_db] = logisticNLP(x1, x2, w, b, alpha)
%
% Inputs:
%   x1 - array of exemplar measurement vectors for class 1.
%   x2 - array of exemplar measurement vectors for class 2.
%   w - an array of weights for the logistic regression model.
%   b - the bias parameter for the logistic regression model.
%   alpha - weight decay parameter
% Outputs:
%   ll - negative log probability (likelihood) for the data
%        conditioned on the model (ie w).
%   dll_dw - gradient of negative log data likelihood wrt w
%   dll_db - gradient of negative log data likelihood wrt b


% YOUR CODE GOES HERE.

    sigmoid1 = logistic(x1, w, b);
    sigmoid2 = logistic(x2, w, b);

    n1 = size(x1, 2);
    n2 = size(x2, 2);

    x = [x1 x2];
    y = [ones(1, n1) zeros(1, n2)];


    % since yi is either 0 or 1
    ll=(1/2*alpha) * (w)' * w - (sum(log(sigmoid1)) + sum(log(1 - sigmoid2)));
    % find w
    dll_dw = w/alpha - x * (y - [sigmoid1 sigmoid2])';

    % find b
    dll_db = - sum(y - [sigmoid1 sigmoid2]);

end
```

○ learLogReg.m (only changed parts)

```
%% TODO 1:
%  Initialize the weights and bias.  The weights w are a vector whose dimension
%  will depend on the dimension of each training exempla and the bias b is a scalar

w = zeros(size(x1, 1), 1);
b = 10;

% Set this to one in order to (roughly) check whether your gradient is correct.
% Set to zero if you're confident that they're correct and you want
% to skip the check
checkGradient = 0;

%%%% TODO 2: you need to implement the function, logisticNLP, that takes
%%%% data x1 and x2 from classes 1 and 2, and the weight vector w,
%%%% and returns the negative log likelihood and the gradient of the
%%%% negative log likelihood.
%%%% Most of the code below can remain unchanged, up to the end of the inner
%%%% loop, but you should look at the code to make sure you understand it.

% Initial negative log-likelihood and gradient
[ll,dll_dw,dll_db] = logisticNLP(x1,x2,w,b,alpha);
```

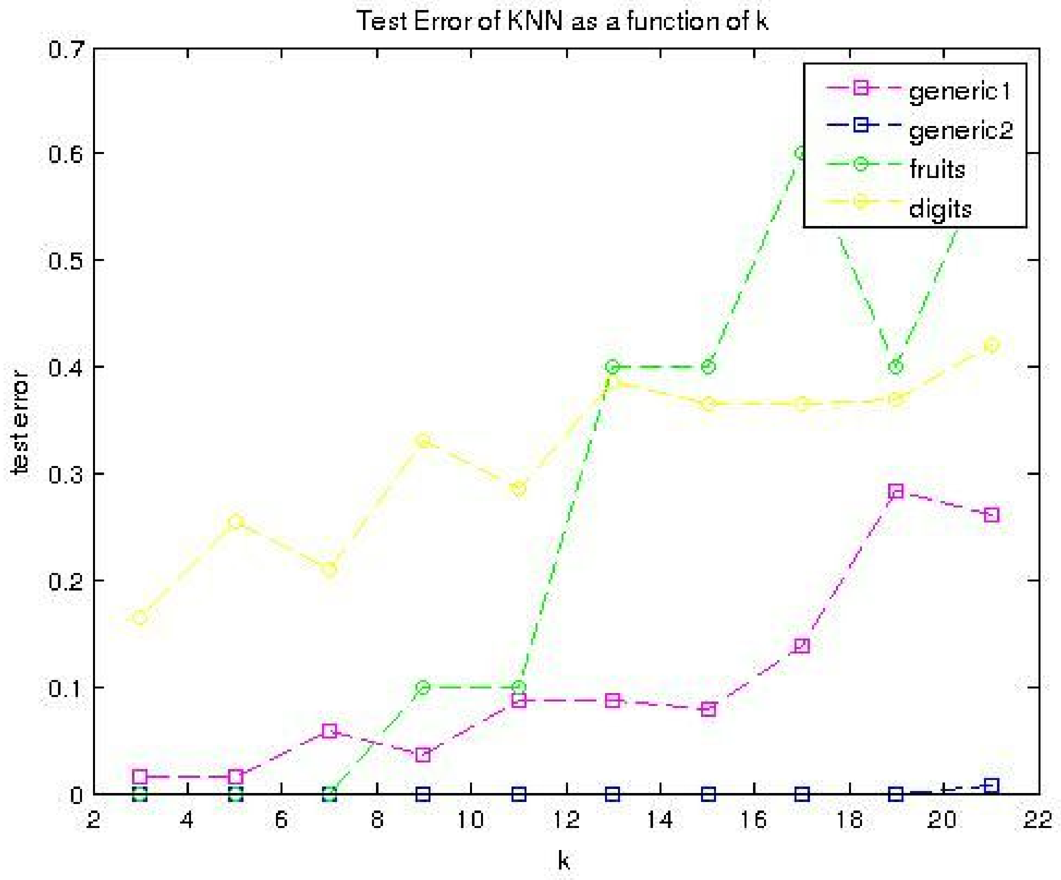
○ logisticStarter.m

```
% Use learnLogReg() to test performance on various datasets.
figure(1);clf;

load('a2/dataSets/generic1');

x1 = c1_train;
x2 = c2_train;

alphas = [0.25, 0.5, 1, 2, 5];

g1E = [];
```

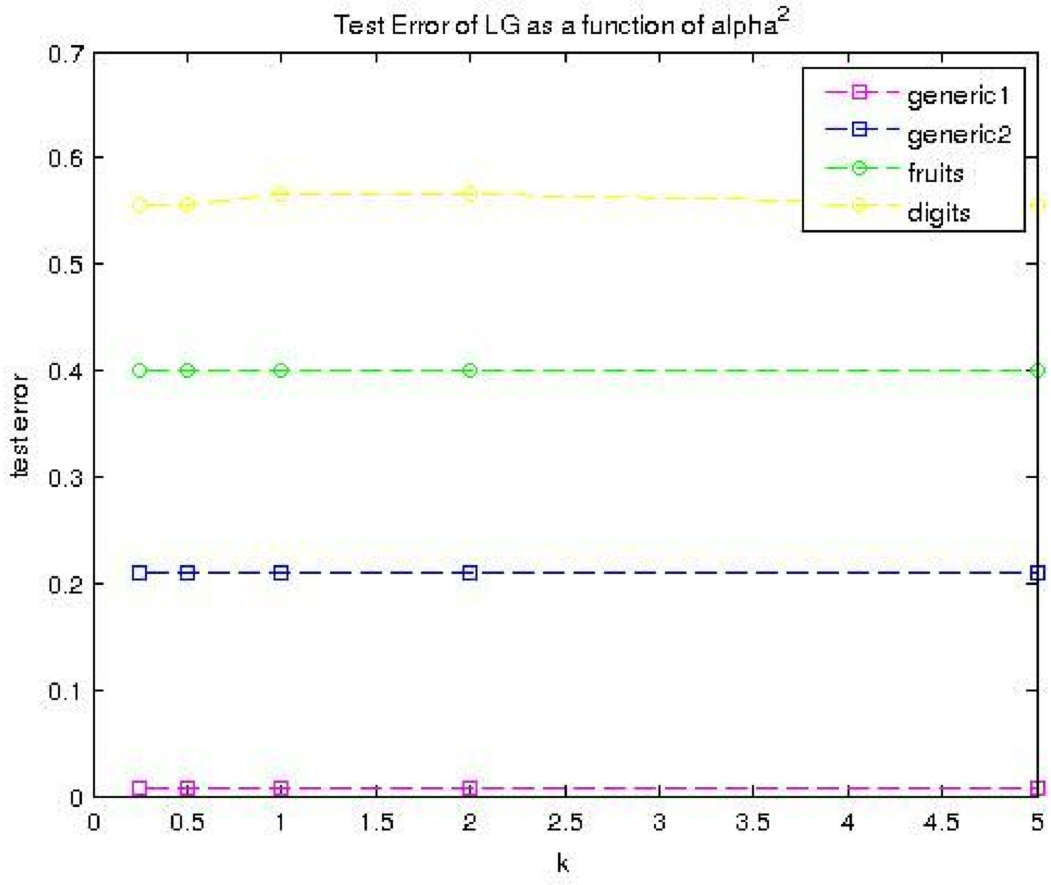
```
n1 = size(c1_test, 2);
n2 = size(c2_test, 2);

for i = 1:numel(alphas)
    num_error = 0;

    testTargets = [ones(1, n1) zeros(1, n2)];

    for j = 1:(n1+n2)
        [w,b] = learnLogReg(x1,x2,alphas(i));
        p = logistic([c1_test c2_test], w, b)

        if round(p(j)) ~= testTargets(j)
            num_error = num_error + 1;
        end
    end
    g1E = [g1E; alphas(i) num_error/(n1+n2)];
end

plot(g1E(:, 1), g1E(:, 2), '--ms');
hold on;
% =======================================================

load('a2/dataSets/generic2');

x1 = c1_train;
x2 = c2_train;

g2E = [];

n1 = size(c1_test, 2);
n2 = size(c2_test, 2);

for i = 1:numel(alphas)
    num_error = 0;

    testTargets = [ones(1, n1) zeros(1, n2)];

    for j = 1:(n1+n2)
        [w,b] = learnLogReg(x1,x2,alphas(i));
        p = logistic([c1_test c2_test], w, b)

        if round(p(j)) ~= testTargets(j)
            num_error = num_error + 1;
        end
    end
    g2E = [g2E; alphas(i) num_error/(n1+n2)];
end

plot(g2E(:, 1), g2E(:, 2), '--bs');
```

```
hold on;

% ======================================================
load('a2/dataSets/fruit_train');
load('a2/dataSets/fruit_test');

E = [];

c1 = find(target_train(1,:)==1);
c2 = find(target_train(2,:)==1);
x1 = inputs_train(:,c1);
x2 = inputs_train(:,c2);

for i = 1:numel(alphas)
    num_error = 0;

    testTargets = [ones(1, n1) zeros(1, n2)];

    for j = 1:size(inputs_test, 2)
        [w,b] = learnLogReg(x1,x2,alphas(i));
        p = logistic(inputs_test, w, b)

        if round(p(j)) ~= target_test(j)
            num_error = num_error + 1;
        end
    end
    E = [E; alphas(i) num_error/size(inputs_test, 2)];
end

plot(E(:, 1), E(:, 2), '--go');
hold on;
% ======================================================
load('a2/dataSets/mnist_train');
load('a2/dataSets/mnist_test');

E = [];

c1 = find(target_train(1,:)==1);
c2 = find(target_train(2,:)==1);
x1 = inputs_train(:,c1);
x2 = inputs_train(:,c2);

for i = 1:numel(alphas)
    num_error = 0;

    testTargets = [ones(1, n1) zeros(1, n2)];

    for j = 1:size(inputs_test, 2)
        [w,b] = learnLogReg(x1,x2,alphas(i));
        p = logistic(inputs_test, w, b)
        if round(p(j)) ~= target_test(j)
            num_error = num_error + 1;
        end
    end
    E = [E; alphas(i) num_error/size(inputs_test, 2)];
end

plot(E(:, 1), E(:, 2), '--yo');

legend('generic1', 'generic2', 'fruits', 'digits');
xlabel('k');
ylabel('test error');
title('Test Error of LG as a function of alpha^2');;
```

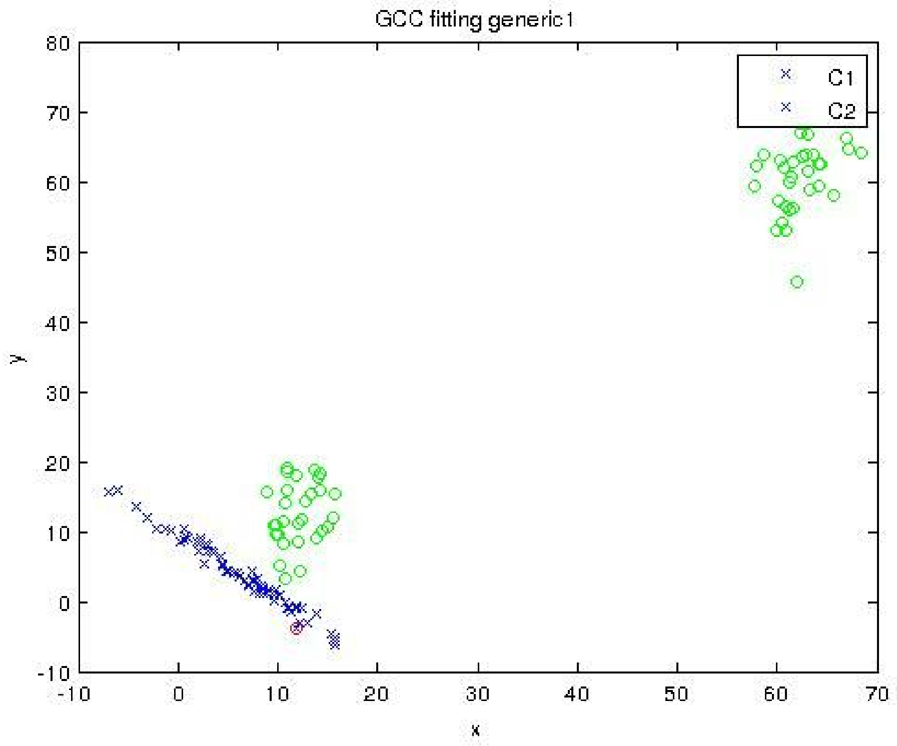
- Error Plots

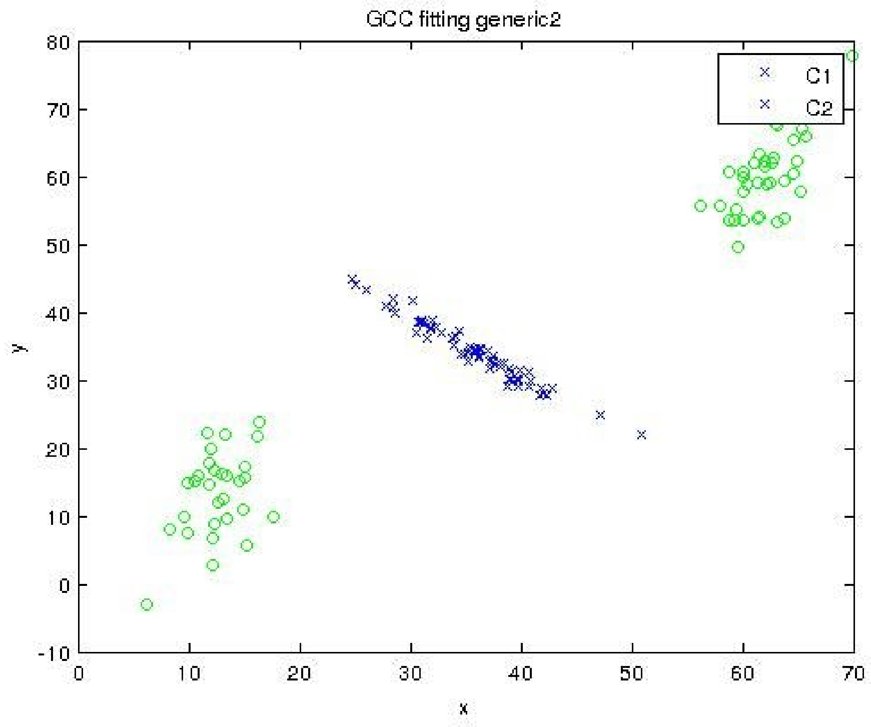


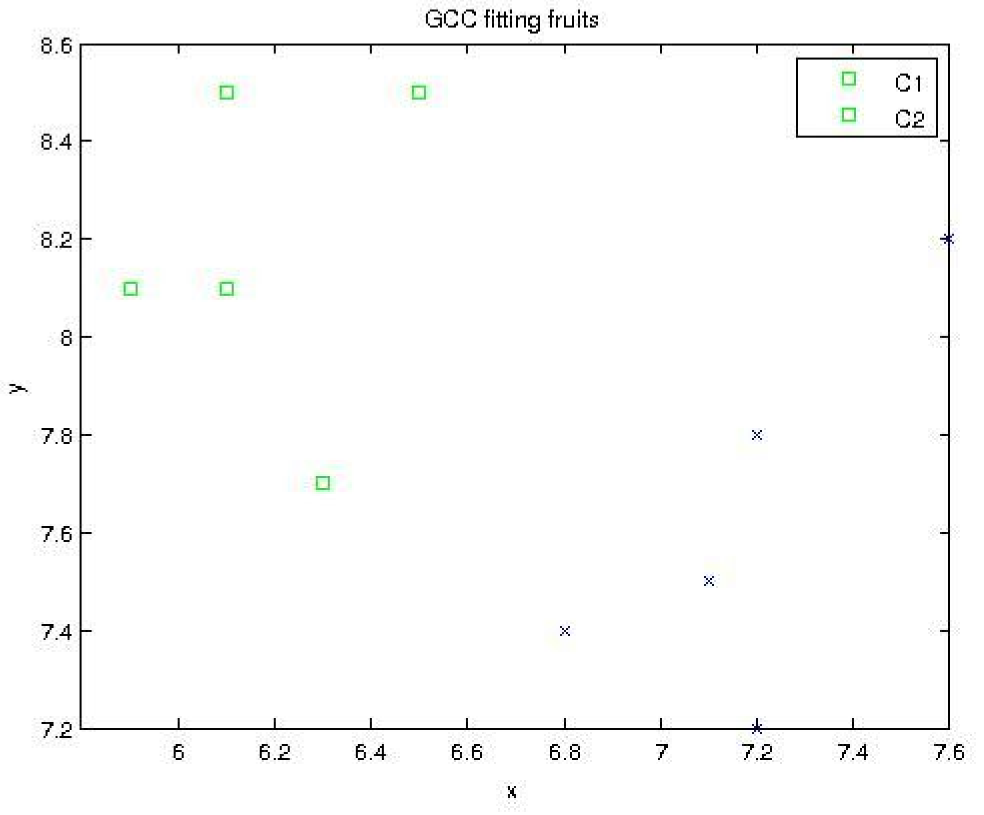
Total Error of **KNN** on **test data** as a function of K



Total Error of **LG** on **test data** as a function of alpha^2


GCC fitting generic1
C1
C2
x
y


GCC fitting generic2
C1
C2
x
y



Sorry, I didn't have enough time to plot the data fit for KNN and LG.
It could basically be done in a same manner as the data plotting implemented in gccStarter.m (and given starter code plotDatasets.m), where we plot the data for both classes separately, do classification on each data point, and highlight those with a different output class as expected (the class defined in target_test).

- Analysis on DataSets
  - **Generic1**

    The best fit would be <u>LogisticRegression</u> as we can see from the error plot it minimizes the possible errors.
    The easiest fit to use would be LG, as KNN and GCC does not fit the data.
    The easiest fit to understand would be LG as well, as KNN and GCC does not fit the data(same reason above).

  - **Generic2**

    The best fit would be <u>KNN</u> as we can see from the error plot it minimizes the possible errors.
    The easiest fit to use would be GCC, as it plots the test data with no error.
    The easiest fit to understand would be GCC, as we can see from the data plot (GCC fitting generic2) that the decision boundary tends to conic.

  - **Fruits**

    The best fit would be <u>GCC</u> as we can see from the data plot (GCC fitting fruits) that the decision boundary tends to conic, and no error was found on test data.
    The easiest fit to use would be LG, as in the test data plotting two classes can be separated by a linear decision boundary.
    The easiest fit to understand would be GCC, as lemon and oranges' height and width seem to be of different but conic relationship.

  - **Digits**

    The best fit would be <u>LogisticRegression</u> as we can see from the error plot it minimizes the possible errors.
    The easiest fit to use would be KNN, as it requires less work compared to LG.
    The easiest fit to understand would be KNN as well, since it makes sense for each input data points to be separated (as they represent a number each).

    Applying GCC to this dataset we got "Matrix is singular to working precision" error in matlab, which is because the covariance matrix is not invertible (needed when we compute the decision boundary a(x)) due to the high dimension of the input array(number of unknowns in the covariance matrix grows quadratically with d).