CSCC11
ASSIGNMENT 2

Huan Ma
1000955120
mahuan2

CODE
knnClassify.m

```
function class = knnClassify(test, k, trainingInputs, trainingTargets)
%
% Inputs:
%   test: test input 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.

% make up a equal dimension test matrix to do matrix operation
m=size(trainingInputs, 1);
testMatrix = repmat(test, m, 1);

% find distance from test input
squaredDistance = sum((testMatrix-trainingInputs).^2, 2);

% sort distance and keep their corresponding label
combination = [squaredDistance trainingTargets];
[d1,d2] = sort(combination(:,1));
combination = combination(d2,:);

% take smallest k distances, and use the median class among
% those exemplars to label the test input.
median = k/2;
% extreme case for k=1
if k == 1
  class = combination(1, 2:3);
else
  class = combination(floor(median), 2:3);
end

end
```

knnStarter.m

```
function result = knnStarter(trainingInputs, trainingTargets, testInputs, testTargets)
%
% Inputs:
%   traingingInputs: array of training exemplars, one exemplar per row
%   traingingTargets: idenicator vector per row
%   testingInputs: array of testing exemplars, one exemplar per row
%   testingTargets: idenicator vector per row
%
% Outputs:
%   A 10*2 matrix where each row has the format:
%   (the value of k, the test error for this k)

% Use the function knnClassify to test performance on different datasets.
result = zeros(10, 2);
total = size(testTargets, 1);
count = 0;
% 0-1 loss function
for k=3:2:21
   error = 0;
   count = count + 1;
   for i = 1: total
      class = knnClassify(testInputs(i,:), k, trainingInputs, trainingTargets);
      if class ~= testTargets(i,:)
         error = error + 1;
      end
   end
   result(count, :) = [k error/total];
end
```

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 b
%   dll_db - gradient of negative log data likelihood wrt b


% YOUR CODE GOES HERE.

% posterior class probability
p1 = logistic(x1, w, b);
p2 = logistic(x2, w, b);

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

x = [x1 x2];
% label for x1 is 1, label for x2 is 0 in LR algorithm
y = [ones(1, n1) zeros(1, n2)];

ll = 0.5*alpha*transpose(w)*w-(sum(log(p1))+sum(log(1-p2)));

dll_dw = w/alpha - x*(y - [p1 p2])';
dll_db = -sum(y - [p1 p2]);
end
```

learnLogReg.m

```
function  [w,b] = learnLogReg(x1,x2,alpha)
%
% Inputs
%   x1  is an array of exemplars from class 1
%   x2  is an array of exemplars from class 2
%   alpha is an optional parameter for a weight decay regularizer
%
% Output
%   w  is a vector of estimated parameters for the logistic regressor
%   b  is the estimated bias term for the logistic regressor


%% 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);
%%%%%%%%%%%%%%% randomly choose the value?
```

```
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);

% Use finite differences to compare numerical and analytical gradients.
if checkGradient
   fd_dll_dw = zeros(size(w));
   h = .001;
   for i=1:length(w)
      w2 = w;
      w2(i)=w2(i)+h;
      fd_dll_dw(i) = (logisticNLP(x1,x2,w2,b,alpha) - ll)/h;
   end
   fd_dll_db = (logisticNLP(x1,x2,w,b+h,alpha) - ll)/h;

   disp('Analytic gradient:')
   disp(dll_dw)
   disp(dll_db)
   disp('Numerical gradient:')
   disp(fd_dll_dw)
   disp(fd_dll_db)
   disp('These should be nearly identical (i.e., within 1%)')
   pause
end

% Initial step-size
lambda = .001;

% Gradient descent inner-loop
l2 = [];
```

```
for t=1:1000
   l2 = [l2;ll];  %keep track of negative log-likelihoods in case you want to plot them
   lambda = lambda * 2;
   e = logisticNLP(x1,x2,w-lambda*dll_dw,b-lambda*dll_db,alpha);

   if sqrt(norm(dll_dw)^2 + norm(dll_db)^2) < eps
      warning('gradient is zero');
      break;
   end
   if isnan(e) || isinf(e)
      error('nan/inf error');
   end

   while (e >= ll) && (lambda > 0)
      lambda = lambda / 2;
      e = logisticNLP(x1,x2,w-lambda*dll_dw,b-lambda*dll_db,alpha);
   end
   if (lambda <= eps)
                  warning('Infinitesimal lambda')
      fprintf(2,'lambda = %f\n',lambda);
      break;
   end

   w = w - lambda*dll_dw;
   b = b - lambda*dll_db;
   [ll,dll_dw,dll_db] = logisticNLP(x1,x2,w,b,alpha);

   fprintf(2,'step = %d, lambda = %f, ll=%f\n\r',t,lambda,ll);
end
```

Logistic.m

```
function p = logistic(X, w, b)
  % Evaluate logistic function
  % X is a M x N matrix, columns are data vectors
  % w is a M x 1 vector.
  % b is a scalar.
  % p = exp(w'*X)./(1+ exp(w'*X)) a 1 x N vector.

  cnst = log(1e200);
  tmp = w(:)' * X;
  p = zeros(1, size(X,2));
  idx1 = (tmp>cnst);
```

```
  % sidx1=sum(idx1)
  if (any(idx1))
    p(idx1) = 1;
  end
  idx0 = (tmp < -cnst);
  % sidx1=sum(idx0)
  if (any(idx0))
    p(idx0) = 0;
  end
  idx = (~idx0 & ~idx1);
  if (any(idx))
    p(idx) = exp(tmp(:, idx) + b);
    p(idx) = p(idx) ./ (1 + p(idx));
  end

  return
```

logisticStarter.m

```
function result = logisticStarter(c1_train, c2_train, testInputs, testTargets)
% Use learnLogReg() to test performance on various datasets.

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

result = zeros(5, 2);
total = size(testTargets, 2);
count = 0;
% numel: number of elements
for i = 1:numel(alphas)
    error = 0;
    count = count + 1;
    % get estimated w and b
    [w, b]=learnLogReg(c1_train, c2_train, alphas(i));
    p = logistic(testInputs, w, b);
    for j= 1:total
        if round(p(j))~=testTargets(1, j)
            error = error +1;
        end
    end
    result(count, :) = [alphas(i) error/total];
end

end
```

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
%

% only for test
%x1 = rand(3);
%x2 = rand(5,3);

n1 = size(x1, 1);
n2 = size(x2, 1);
p1 = n1/(n1+n2);

m1 = sum(x1,1)'/n1;
m2 = sum(x2,1)'/n2;

% maximum likelihood estimator: divides sum by n instead of n-1
C1 = cov(x1, 1);
C2 = cov(x2, 1);
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.
C1_inverse = inv(C1);
C2_inverse = inv(C2);
aX=-0.5*transpose(x-m1)*C1_inverse*(x-m1)-0.5*log(det(C1))+0.5*transpose(x-m2)*C2_inverse
*(x-m2)+0.5*log(det(C2));

if aX>0
   class = [1 0];
else
   class = [0 1];
end

end
```

gccStarter.m

```
function [estimatedTargets, error] = gccStarter(c1_train, c2_train, testInputs, testTargets)
% 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.

[p1, m1, m2, C1, C2]=learnGCCmodel(c1_train', c2_train');
total = size(testTargets, 2);
estimatedTargets = zeros(total, 2);
count = 0;
error = 0;
figure();
for i=1:size(testInputs,2)
   count = count + 1;
   class = gccClassify(testInputs(:, i), p1, m1, m2, C1, C2);
   estimatedTargets(count, :) = class;
   if class ~= transpose(testTargets(:,i))
      error = error + 1;
      plot(testInputs(1,i), testInputs(2,i),'rs');hold on;
   else
      if class(1,1) == 1
         plot1=plot(testInputs(1,i), testInputs(2,i),'bx');hold on;
      elseif class(1,1) == 0
         plot2=plot(testInputs(1,i), testInputs(2,i),'go');hold on;
```

```
        end
    end
end

xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('GCC fitting model for generic data')
hold on;

error = error/total;
end
```

main.m

```
%% Generic1/2:
% This should be modified if the directory is different
load('generic1');

% c1_train and c2_train are one exp per column
c1_train_col = size(c1_train, 2);
c2_train_col = size(c2_train, 2);


trainingInputs = transpose([c1_train c2_train]);
trainingTargets = [ones(c1_train_col, 1) zeros(c1_train_col, 1);
    zeros(c2_train_col, 1) ones(c2_train_col, 1)];

% get size of data
c1_test_col = size(c1_test, 2);
c2_test_col = size(c2_test, 2);

testInputs = transpose([c1_test c2_test]);
testTargets = [ones(c1_test_col, 1) zeros(c1_test_col, 1);
    zeros(c2_test_col, 1) ones(c2_test_col, 1)];

% apply Knn algorithm
g1Knn = knnStarter(trainingInputs, trainingTargets, testInputs, testTargets);

% apply logistic Regression
g1Log = logisticStarter(c1_train, c2_train, testInputs', testTargets');

% apply GCC model
[g1EstimatedTargetGCC, g1ErrorGcc] = gccStarter(c1_train, c2_train, testInputs', testTargets');
```

```
%%
%
%% Plot logistic and KNN model fit for generic1/2
%% GCC
[trainTarget, trainError] = gccStarter(c1_train, c2_train, trainingInputs', trainingTargets');
%% Logistic
% since some alphas has the same test error value, choose the
% smallest alpha by occam's razor

% for testing data
alpha = 0.25;
total = size(testTargets, 1);
figure();
[w, b]=learnLogReg(c1_train, c2_train, alpha);
p = logistic(testInputs', w, b);
for j= 1:total
   if round(p(j))~=testTargets(j, 1)
      plot(testInputs(j, 1), testInputs(j, 2),'rs');hold on;
   else
      if round(p(j)) == 1
         plot1 = plot(testInputs(j, 1), testInputs(j, 2),'bx');hold on;
      elseif  round(p(j)) == 0
         plot2 = plot(testInputs(j, 1), testInputs(j, 2),'go');hold on;
      end
   end
end
xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('Logistic fitting model for generic1 test data')
hold on;

%%
% for training data
alpha = 0.25;
total = size(trainingTargets, 1);
figure();
[w, b]=learnLogReg(c1_train, c2_train, alpha);
p = logistic(trainingInputs', w, b);
for j= 1:total
   if round(p(j))~=trainingTargets(j, 1)
      plot(trainingInputs(j, 1), trainingInputs(j, 2),'rs');hold on;
   else
```

```
        if round(p(j)) == 1
            plot1 = plot(trainingInputs(j, 1), trainingInputs(j, 2),'bx');hold on;
        elseif  round(p(j)) == 0
            plot2 = plot(trainingInputs(j, 1), trainingInputs(j, 2),'go');hold on;
        end
    end
end
xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('Logistic fitting model for generic1 train data')
hold on;
%%
%KNN
% since some ks has the same test error value, choose the
% smallest k by occam's razor

% for test data

k=3;
total = size(testTargets, 1);
figure();
 for i = 1: total
     class = knnClassify(testInputs(i,:), k, trainingInputs, trainingTargets);
     if class ~= testTargets(i,:)
        plot(testInputs(i, 1), testInputs(i, 2),'rs');hold on;
     else
        if class(1,1) == 1
            plot1 = plot(testInputs(i, 1), testInputs(i, 2),'bx');hold on;
        elseif class(1,1) == 0
            plot2 = plot(testInputs(i, 1), testInputs(i, 2),'go');hold on;
        end
    end
 end

xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('KNN fitting model for generic1 test data')
hold on;

%% for training
k=3;
```

```
total =  size(trainingTargets, 1);
figure();
 for i = 1: total
     class = knnClassify(trainingInputs(i,:), k, trainingInputs, trainingTargets);
     if class ~= testTargets(i,:)
        plot(trainingInputs(i, 1), trainingInputs(i, 2),'rs');hold on;
     else
        if class(1,1) == 1
            plot1 = plot(trainingInputs(i, 1), trainingInputs(i, 2),'bx');hold on;
        elseif class(1,1) == 0
            plot2 = plot(trainingInputs(i, 1), trainingInputs(i, 2),'go');hold on;
        end
    end
 end

xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('KNN fitting model for generic1 train data')
hold on;
%%
%
%% Basicly Generic2 is the same as generic1 but only different data value
load('generic2');

% c1_train and c2_train are one exp per column
c1_train_col = size(c1_train, 2);
c2_train_col = size(c2_train, 2);

trainingInputs = transpose([c1_train c2_train]);
trainingTargets = [ones(c1_train_col, 1) zeros(c1_train_col, 1);
    zeros(c2_train_col, 1) ones(c2_train_col, 1)];

c1_test_col = size(c1_test, 2);
c2_test_col = size(c2_test, 2);

testInputs = transpose([c1_test c2_test]);
testTargets = [ones(c1_test_col, 1) zeros(c1_test_col, 1);
    zeros(c2_test_col, 1) ones(c2_test_col, 1)];

% apply Knn algorithm to generic2
g2Knn = knnStarter(trainingInputs, trainingTargets, testInputs, testTargets);
```

```
% apply logistic Regression to generic2
g2Log = logisticStarter(c1_train, c2_train, testInputs', testTargets');

% apply GCC model
[g2EstimatedTargetGcc, g2ErrorGcc] = gccStarter(c1_train, c2_train, testInputs', testTargets');
%%
%
%% Plot logistic, GCC and KNN model fit for generic1/2
%% GCC
[trainTarget, trainError] = gccStarter(c1_train, c2_train, trainingInputs', trainingTargets');

%%
% Logistic
alpha = 0.25;
total = size(testTargets, 1);
error = 0;
figure();
[w, b]=learnLogReg(c1_train, c2_train, alpha);
p = logistic(testInputs', w, b);
for j= 1:total
   if round(p(j))~=testTargets(j, 1)
      error = error + 1;
      plot(testInputs(j, 1), testInputs(j, 2),'rs');hold on;
   else
      if round(p(j)) == 1
         plot1 = plot(testInputs(j, 1), testInputs(j, 2),'bx');hold on;
      elseif round(p(j)) == 0
         plot2 = plot(testInputs(j, 1), testInputs(j, 2),'go');hold on;
      end
   end
end
xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('Logistic fitting model for generic2 test data')
hold on;
disp(error/total);
%%
% for training data
alpha = 0.25;
total = size(trainingTargets, 1);
figure();
[w, b]=learnLogReg(c1_train, c2_train, alpha);
```

```
p = logistic(trainingInputs', w, b);
for j= 1:total
   if round(p(j))~=trainingTargets(j, 1)
      plot(trainingInputs(j, 1), trainingInputs(j, 2),'rs');hold on;
   else
      if round(p(j)) == 1
         plot1 = plot(trainingInputs(j, 1), trainingInputs(j, 2),'bx');hold on;
      elseif  round(p(j)) == 0
         plot2 = plot(trainingInputs(j, 1), trainingInputs(j, 2),'go');hold on;
      end
   end
end
xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('Logistic fitting model for generic2 train data')
hold on;

%%
% KNN
% since some ks has the same test error value, choose the
% smallest k by occam's razor

k=3;
total = size(testTargets, 1);
figure();

 for i = 1: total
    class = knnClassify(testInputs(i,:), k, trainingInputs, trainingTargets);
    disp(class);
    if class ~= testTargets(i,:)
      plot(testInputs(1,i), testInputs(2,i),'rs');hold on;
    else
      if class(1,1) == 1
         plot1=plot(testInputs(i, 1), testInputs(i, 2),'bx');hold on;
      elseif class(1,1) == 0
         plot2=plot(testInputs(i, 1), testInputs(i, 2),'go');hold on;
      end
   end
 end

xlabel('x')
ylabel('y')
```

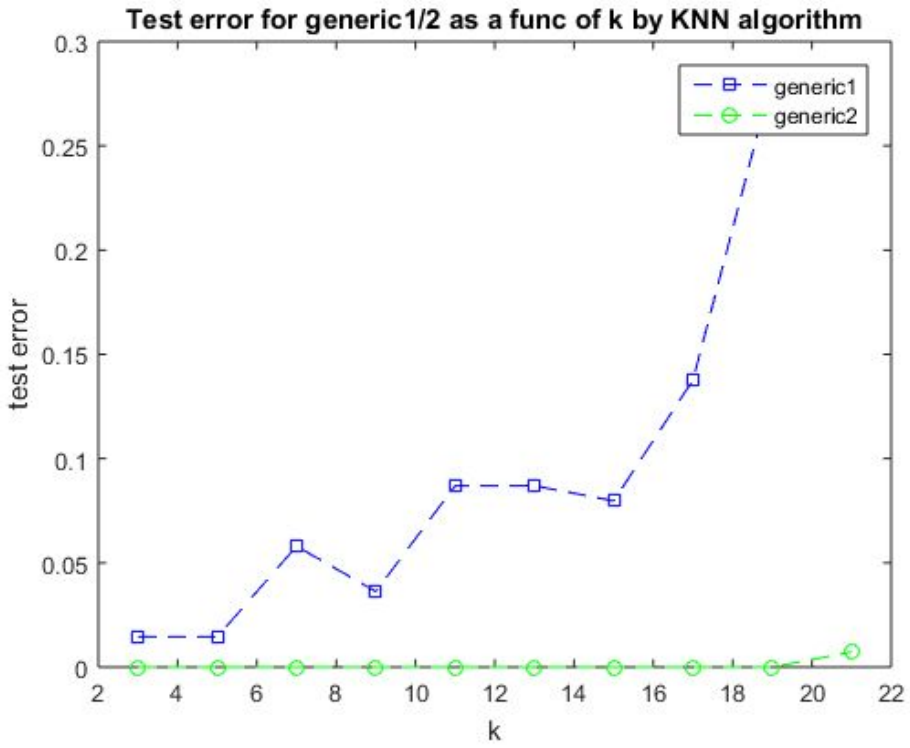
```
legend([plot1,plot2],'C1','C2');
title('KNN fitting model for generic2 test data')
hold on;

%%
k=3;
total = size(trainingTargets, 1);
figure();
 for i = 1: total
     class = knnClassify(trainingInputs(i,:), k, trainingInputs, trainingTargets);
     if class ~= testTargets(i,:)
       plot(trainingInputs(i, 1), trainingInputs(i, 2),'rs');hold on;
     else
       if class(1,1) == 1
          plot1 = plot(trainingInputs(i, 1), trainingInputs(i, 2),'bx');hold on;
       elseif class(1,1) == 0
          plot2 = plot(trainingInputs(i, 1), trainingInputs(i, 2),'go');hold on;
       end
    end
 end

xlabel('x')
ylabel('y')
legend([plot1,plot2],'C1','C2');
title('KNN fitting model for generic2 train data')
hold on;
%% Apply 3 Algorithms to Fruit
load('fruit_train');
load('fruit_test');

c1Index = find(target_train(1,:)==1);
c2Index = find(target_train(2,:)==1);

%seperate training data by their class 1 or 2
c1_train = inputs_train(:, c1Index);
c2_train = inputs_train(:, c2Index);

fruKnn = knnStarter(inputs_train', target_train', inputs_test', target_test');
fruLog = logisticStarter(c1_train, c2_train, inputs_test, target_test);
[fruEstimatedTargetGcc, fruErrorGcc] = gccStarter(c1_train, c2_train, inputs_test, target_test);
%% Apply 2 Algorithms to Digits
load('mnist_train');
load('mnist_test');
```

```
c1Index = find(target_train(1,:)==1);
c2Index = find(target_train(2,:)==1);

c1_train = inputs_train(:, c1Index);
c2_train = inputs_train(:, c2Index);

mniKnn = knnStarter(inputs_train', target_train', inputs_test', target_test');

mniLog = logisticStarter(c1_train, c2_train, inputs_test, target_test);
%% Plots

% For KNN, plot the test error as a function of k. Use k = 3, 5, 7, ..., 21
figure();
plot(g1Knn(:,1), g1Knn(:,2), '--bs');
hold on;
plot(g2Knn(:,1), g2Knn(:,2), '--go');
legend('generic1', 'generic2');
xlabel('k');
ylabel('test error')
title('Test error for generic1/2 as a func of k by KNN algorithm')
hold on;

% For LR, plot of the test error as a function of the regularization weight.
% Use alpha = 0.25, 0.5, 1, 2, and 5
figure();
plot(g1Log(:,1), g1Log(:,2), '--bs');
hold on;
plot(g2Log(:,1), g2Log(:,2), '--go');
legend('generic1', 'generic2');
xlabel('k');
ylabel('test error')
title('Test error for generic1/2 as a func of the regularization weight by LR algorithm')
hold on;

% For one instance of KNN, one instance of LR, and the GCC model,
% plot the model fit to the data.

%%%%%%%%%%%%%%%%%%%%%% I choose k=3 for both generic1 and generic2
because
%%%%%%%%%%%%%%%%%%%%%% the minimum test error value occurs at k=3
choose
%%%%%%%%%%%%%%%%%%%%%% alpha=0.25 since the plot for different alpha is a
```

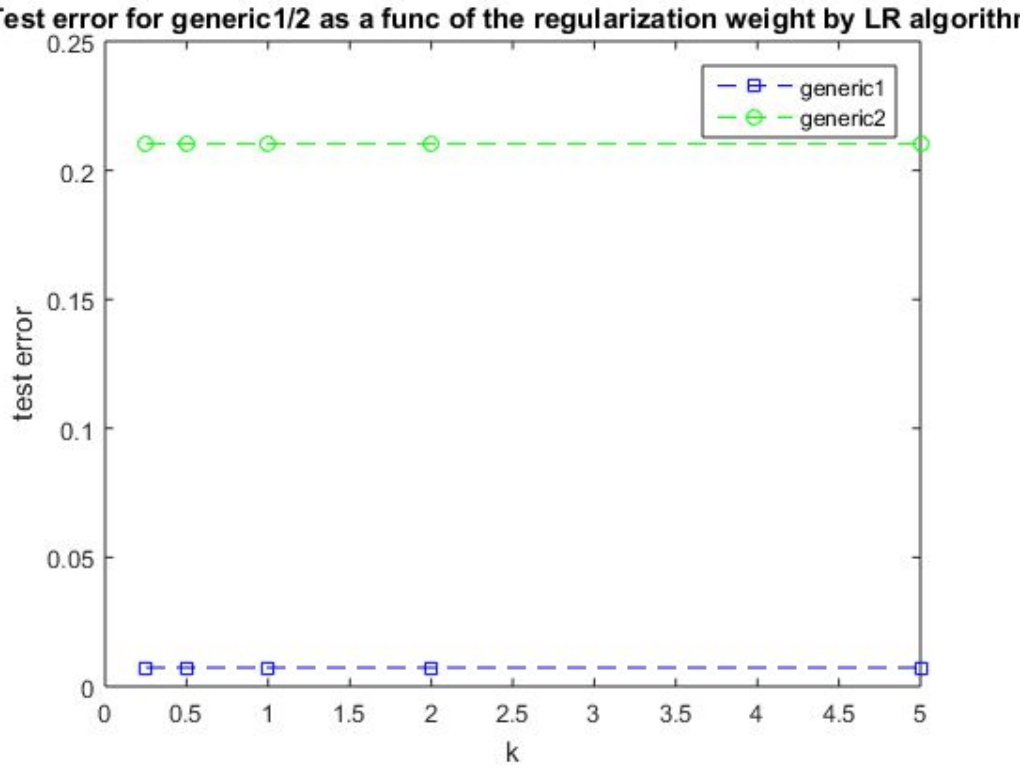
```
%%%%%%%%%%%%%%%%%%%%%% horizontal line which means the test error won't
%%%%%%%%%%%%%%%%%%%%%% change if the value of alpha change


%% Plot test error rate for 3 algorithms for fruit
figure();
plot(fruKnn(:,1), fruKnn(:,2), '--bs');
hold on;
plot(fruLog(:,1), fruLog(:,2), '--go');
hold on;

xAxis = [1; 5; 10; 15; 20];
yAxis = [fruErrorGcc; fruErrorGcc; fruErrorGcc; fruErrorGcc; fruErrorGcc];
plot(xAxis, yAxis, '--ro');
legend('Knn', 'Logistic', 'GCC');
xlabel('parameter');
ylabel('test error')
title('Test error for fruit data by 3 algorithms')
hold on;
%% Plot test error rate for 2 algorithms for mnist
figure();
plot(mniKnn(:,1), mniKnn(:,2), '--bs');
hold on;
plot(mniLog(:,1), mniLog(:,2), '--go');
hold on;

legend('Knn', 'Logistic');
xlabel('parameter');
ylabel('test error')
title('Test error for mnist data by 2 algorithms')
hold on;

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

Generic1/2

1. For KNN, plot the test error as a function of k. Use k = 3, 5, 7, ..., 21.



2. For LR, plot of the test error as a function of the regularization weight. Use α = 0.25, 0.5, 1, 2, and 5.

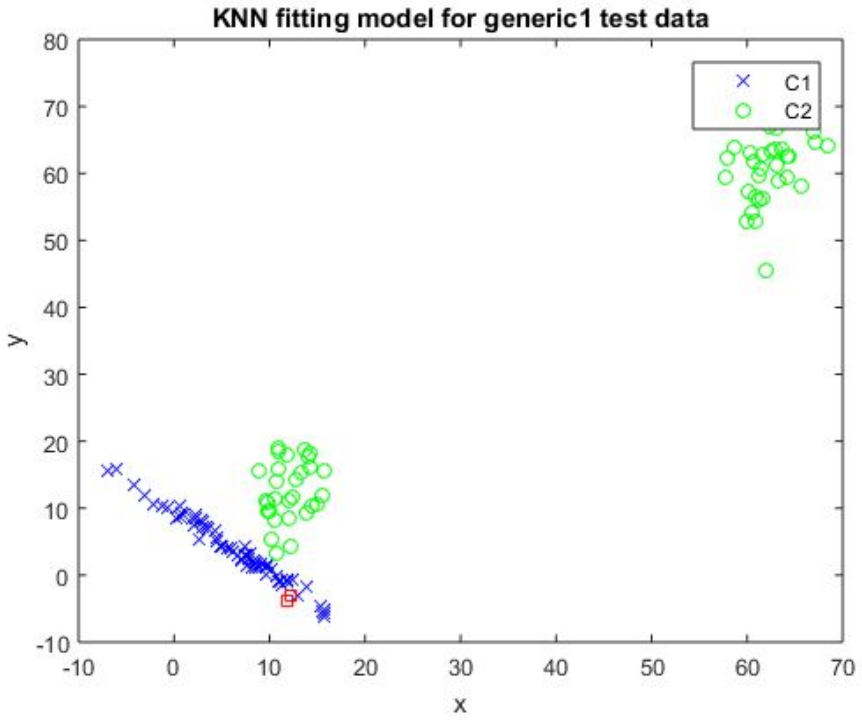


3. For one instance of KNN (your choice of k), one instance of LR (your choice of α), and the GCC model, plot the model fit to the data. To this end you could plot the decision boundaries, and/or you could plot which training and test points are correctly or incorrectly labelled.

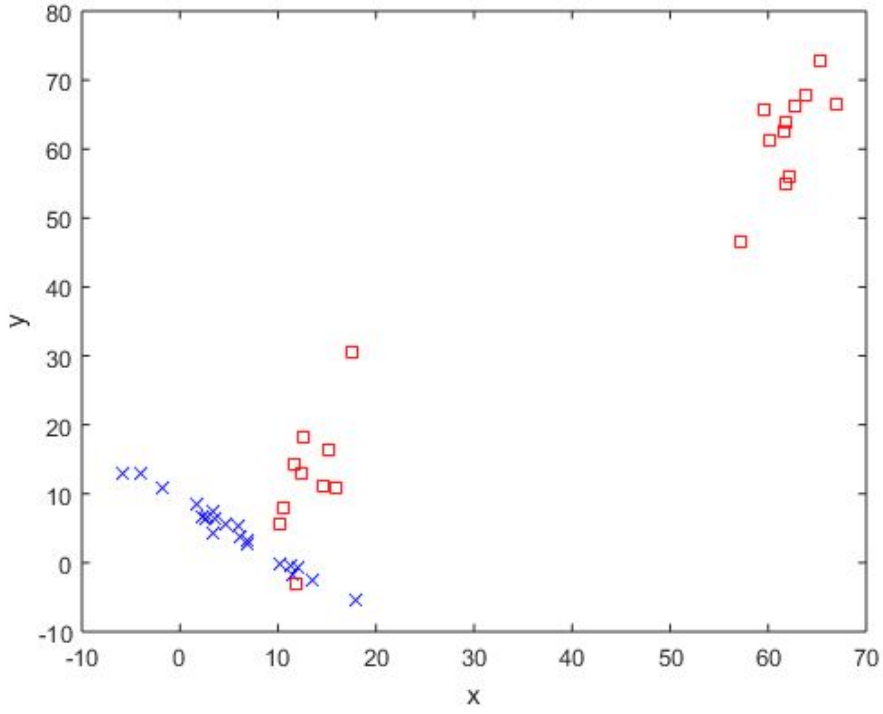
Answer:
Plot the training and test points are correctly or incorrectly labelled. Blue for class 1, Green for class 2, red for mislabeled points.

1)KNN: choose **k=3** for both generic1 and generic 2, since the minimum test error occurs at k=3 and k=5 for both generic1&2, choose the smaller value by occam's razor.

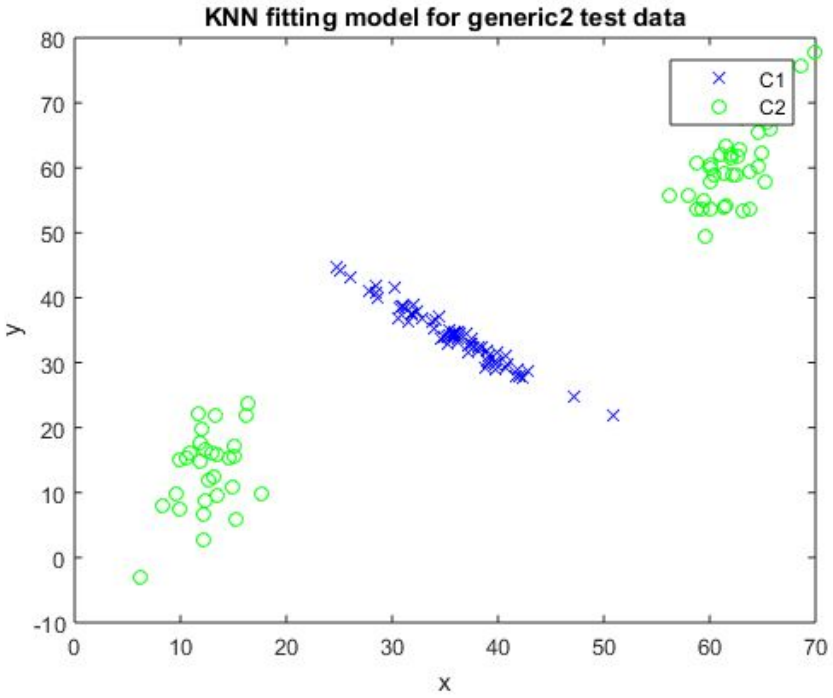
Generic1:
Testing data:



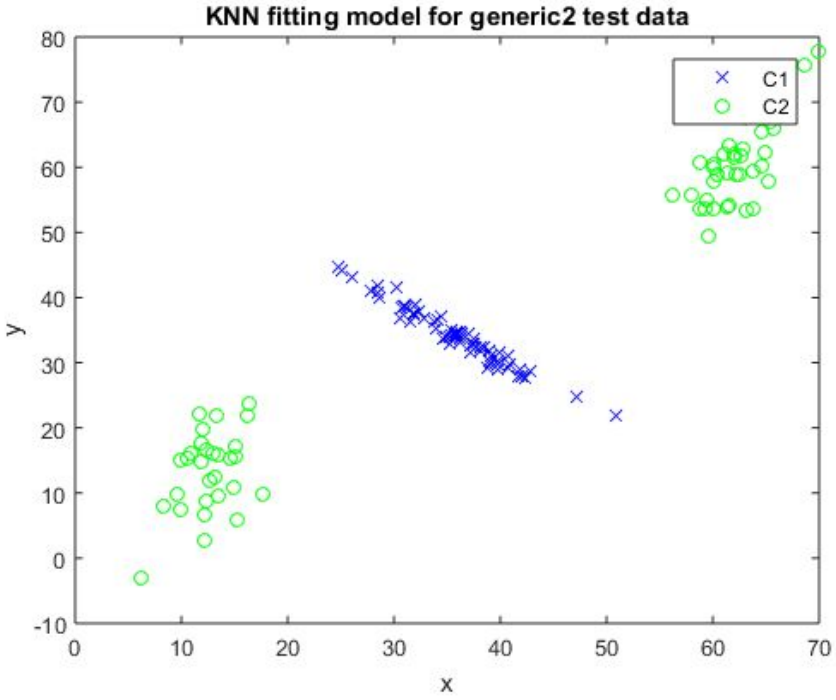
Training Data:



Generic2:
Testing data:

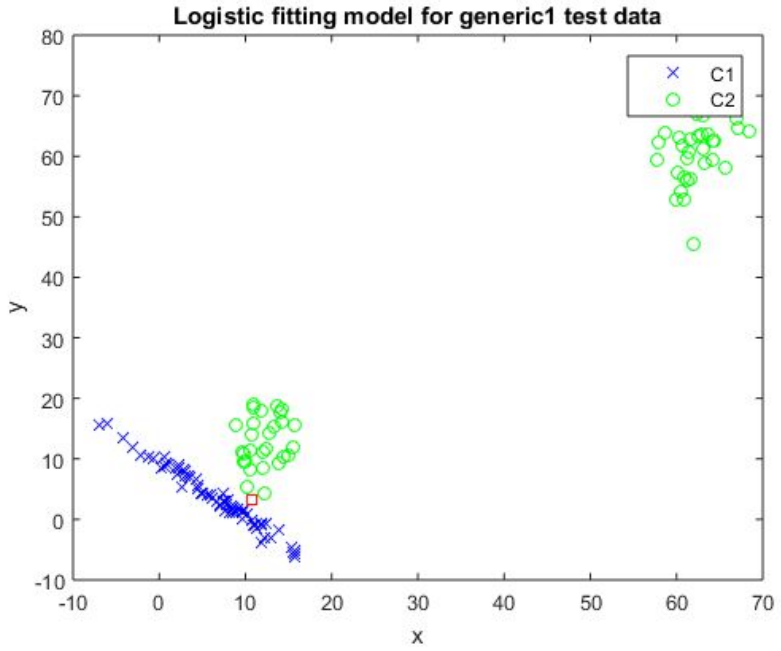


Training data:

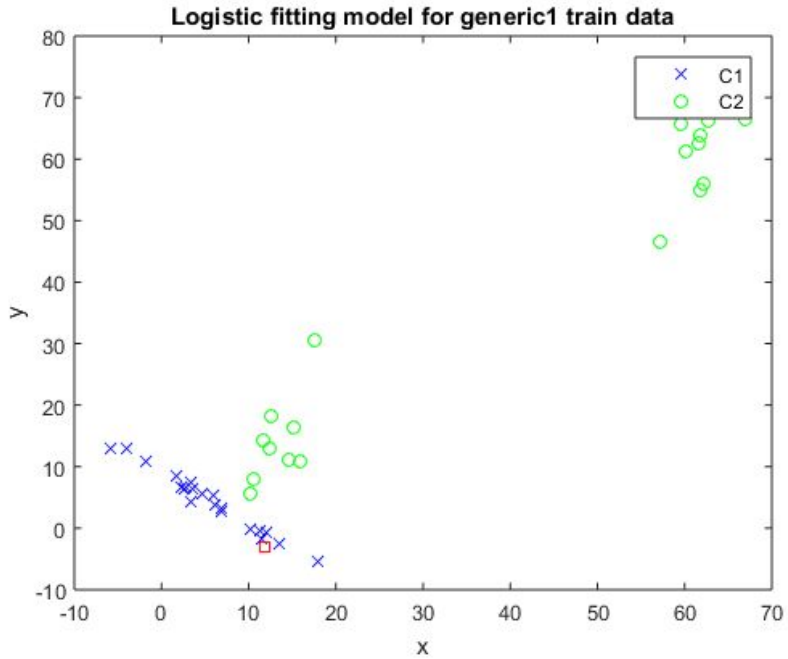


2)Logistic: choose **alpha=0.25** for both generic1 and generic 2, since the test error is stable for all the alpha values on generic1&2, choose the smallest value by occam's razor.

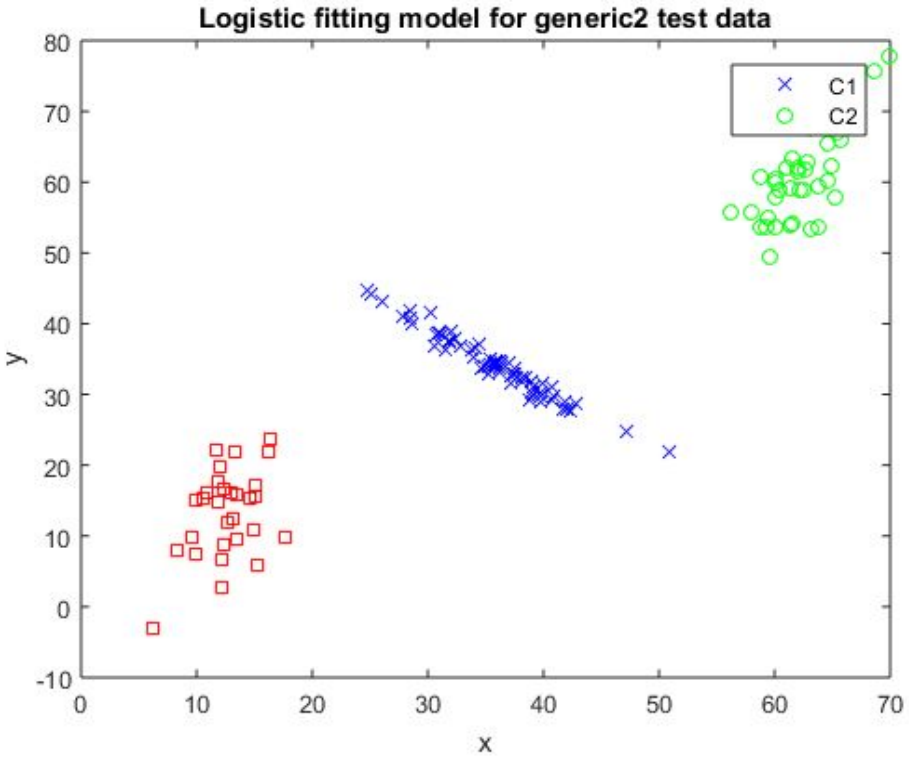
Generic1:
Testing data:



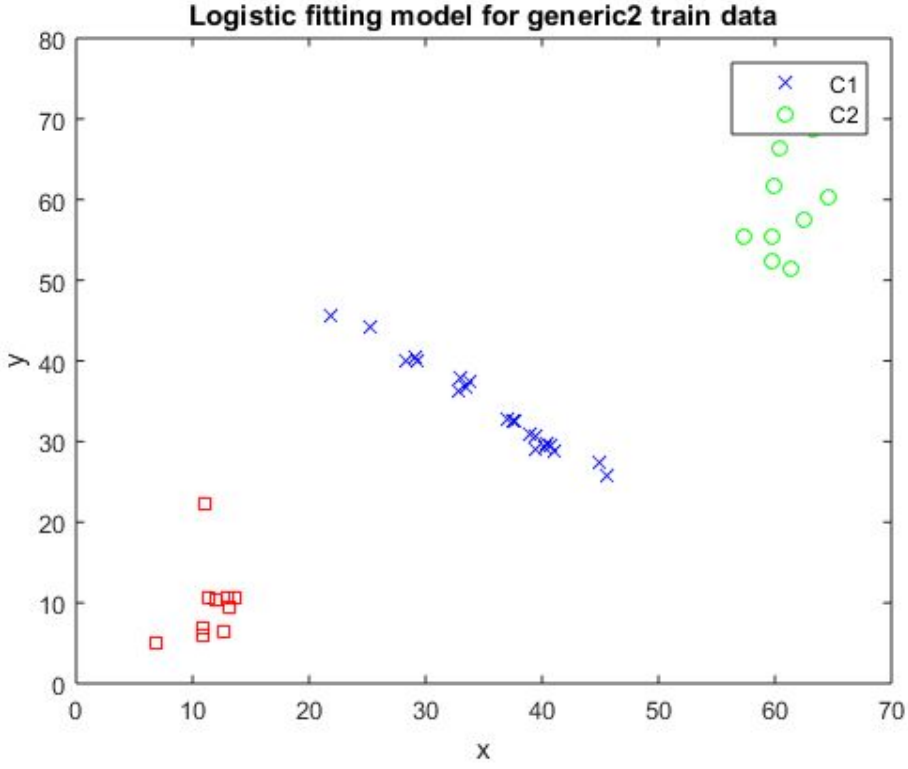
Training data:



Generic2:
Testing data:



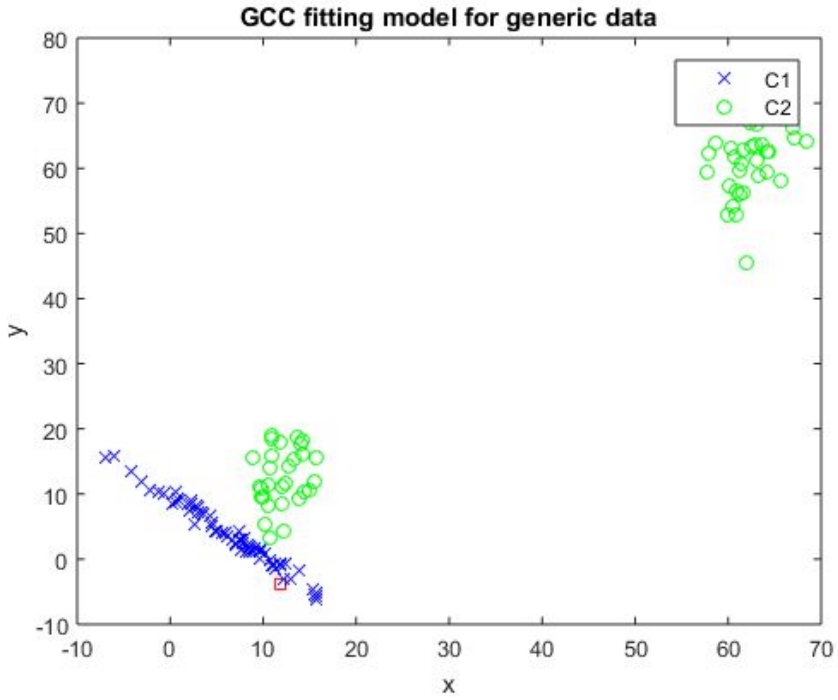
Training data:



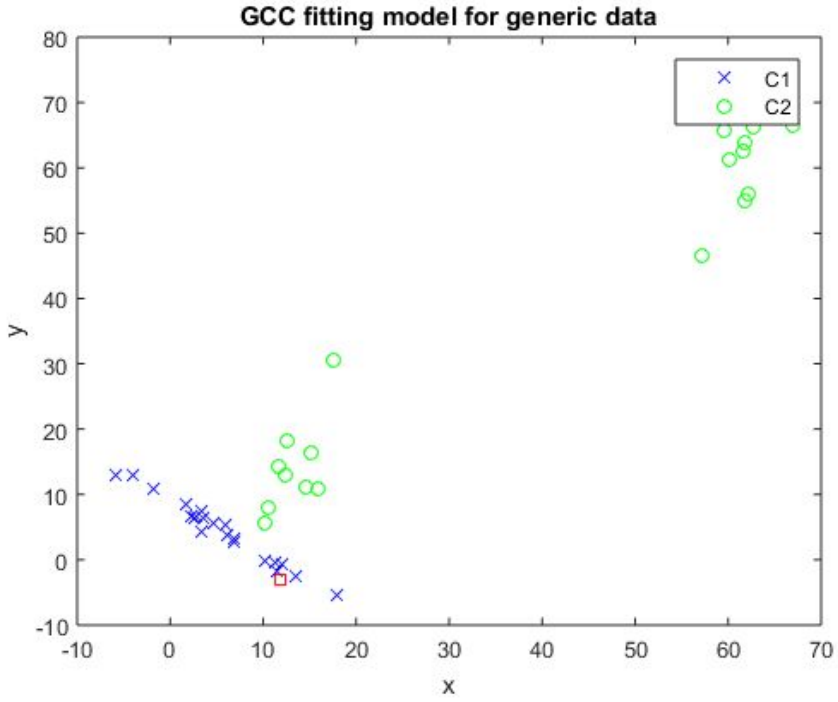
3)GCC model

Generic1:

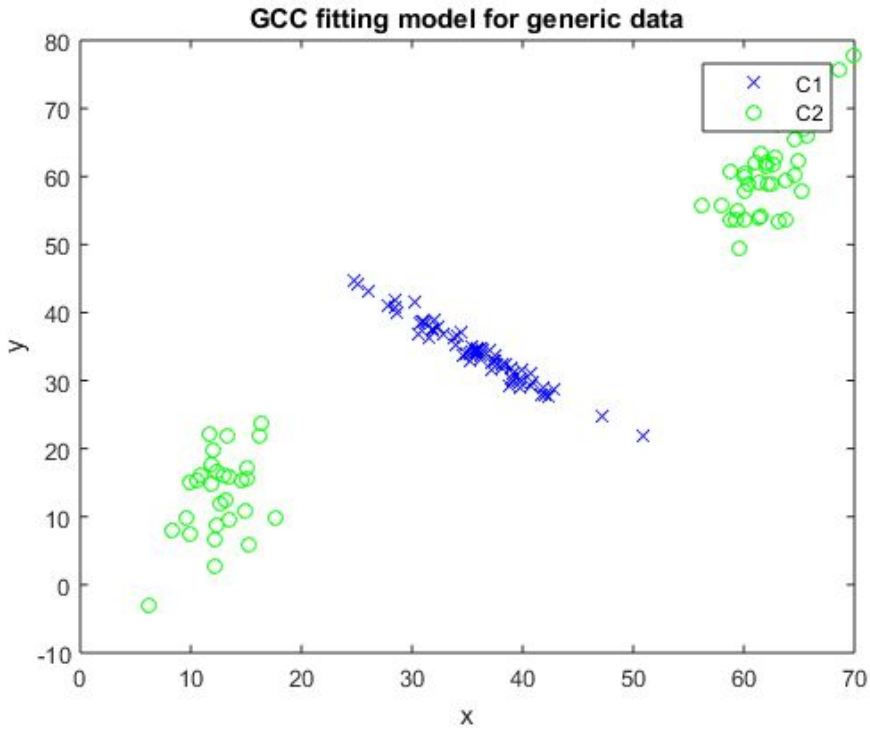
Testing data:



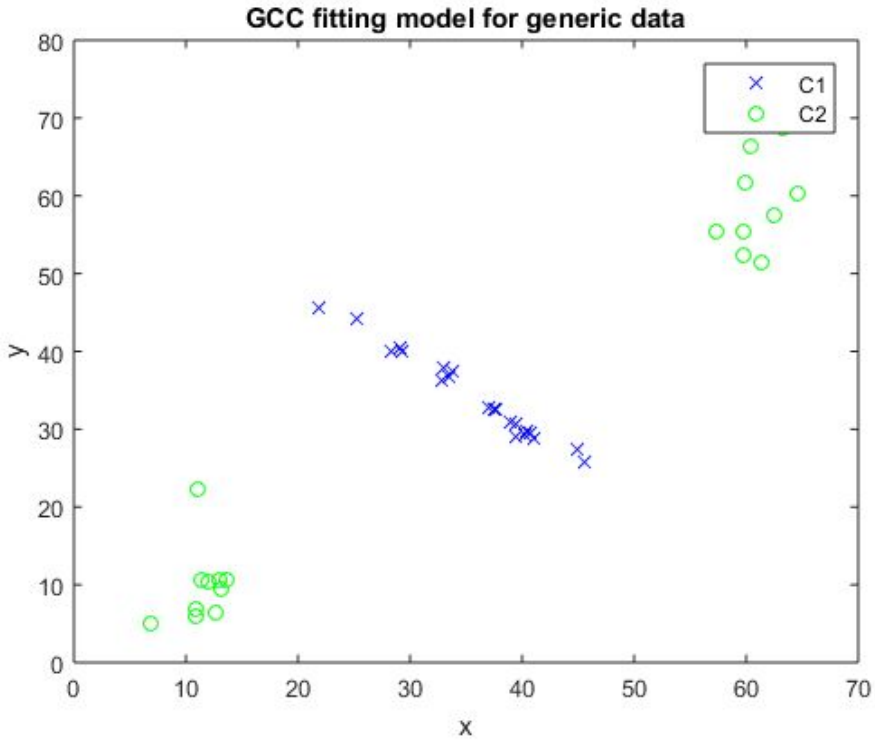
Training data:



Generic2:
Testing data:



Training data:



4. Provide answers, with explanations, for the following questions: Which of these methods worked best for this data set? Which seemed easiest to use? Which were easiest to understand?

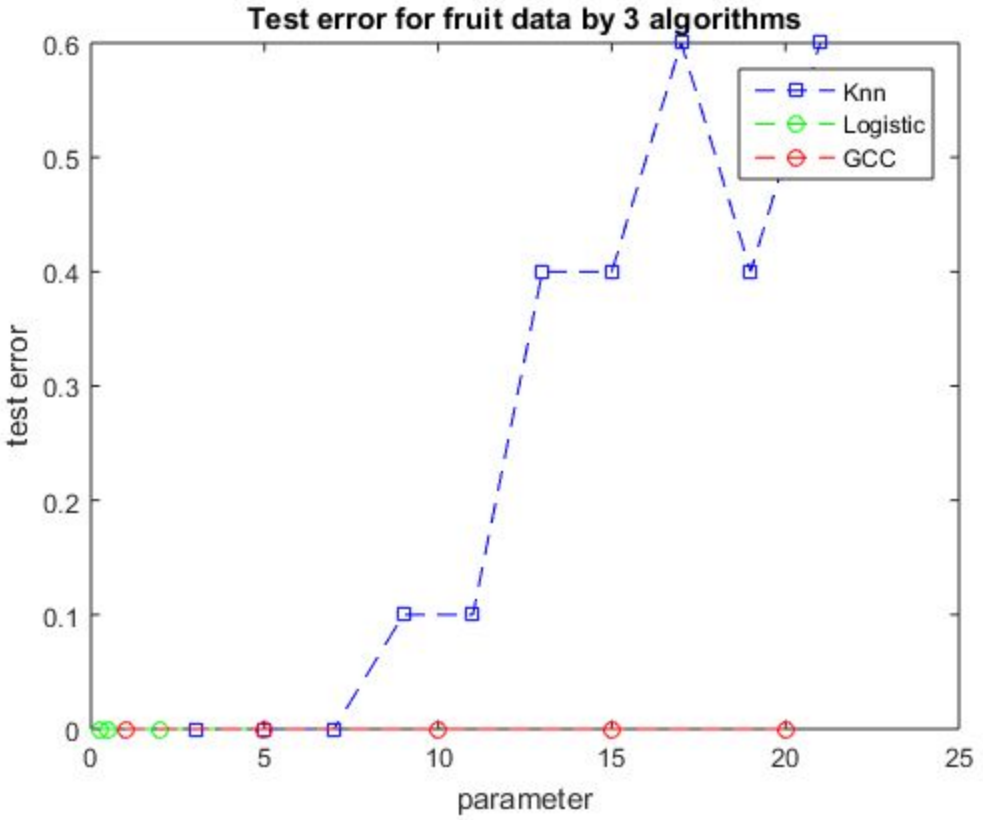
Answer:
I. For Generic1, test error rate for GCC = 0.0072, the smallest test error rate for KNN = 0.0144, for logistic= 0.0072. For Generic2, test error rate for GCC = 0, the smallest test error rate for KNN = 0, for logistic = 0.2101. Thus, the best method is GCC because it has the smallest test error among 3 models on Generic1/2.

II. GCC is the easiest to use, it has no parameters compared to KNN has parameter k and Logistic Regression has parameter alpha. For the algorithms that have parameters, we need to find the best value for their parameters first, but for GCC, we can train the classifier directly.

III. I personally think KNN is the easiest to understand. The idea is intuitive that the label of a test point should be relevant to the labels of its nearest several points.

Fruit

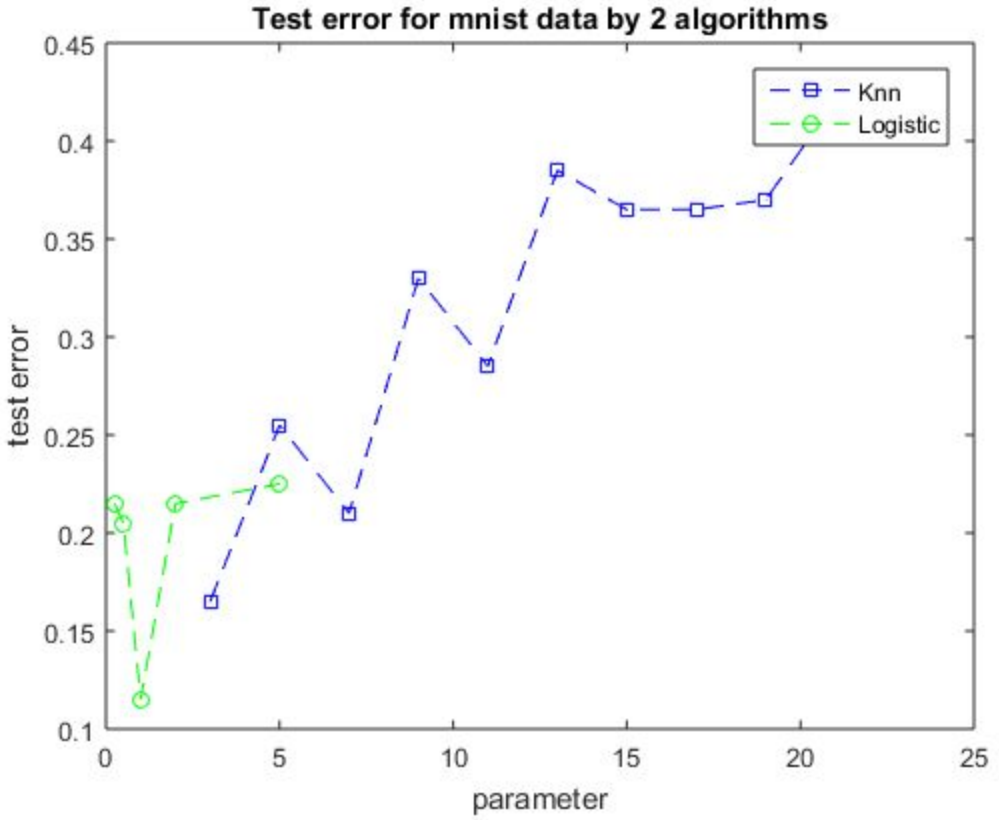
Apply all three algorithms to the fruit data. Which of these methods appeared to work best for this data set? Why?



As we can see from the plot, the test error rate = 0 on Fruit on GCC, Logistic Regression for all alpha values and KNN for k=3, 5, 7. By Occam's razor, we choose the easiest and simplest algorithm: GCC.

Digits

Apply kNN and LR to this dataset. Which method appears to work best for this data set? Why? What problems can you foresee if you were to try to apply GCC to this data?



Logistic Regression at alpha =1 seems to work best for the data set. Because from the plot we can see, the smallest test error rate occurs when using Logistic Regression at alpha = 1.

GCC won't be a good choice for this data set because each column in this data set represents a digit number on a square plot with black background, thus elements in the columns is not i.i.d, it could never be normally distributed, so we can't use Gaussian class conditionals.