Problem Set 5

MAS 622J/1.126J: Pattern Recognition and Analysis

Due Monday, 8 November 2010. Resubmission due, 12 November 2010

Note: All instructions to plot data or write a program should be carried out using Matlab. In order to maintain a reasonable level of consistency and simplicity we ask that you do not use other software tools.

If you collaborated with other members of the class, please write their names at the end of the assignment. Moreover, you will need to write and sign the following statement: "In preparing my solutions, I did not look at any old homeworks, copy anybody's answers or let them copy mine."

Problem 1: Mixture of Gaussians [35 points]

Implement the EM algorithm for estimating the parameters of a mixture of Gaussians with isotropic covariances $\Sigma_j = \sigma_j I$. Note: Download the data file from the course website. There are two datasets each of which is two-dimensional.

To solve this problem, and just this problem, you can write your own code or use any MATLAB toolboxes available for the purpose. In particular, there is a MATLAB mixture of Gaussians algorithm available for download here that would be good to explore (Note: If you use this you will need to make some small adjustments to the code. Specifically to handle the requirement for isotropic covariances.):

`http://www.lx.it.pt/~mtf/mixturecode2.zip`

Also see the accompanying paper "Unsupervised Learning of Finite Mixture Models", M. Figueiredo and A.K. Jain.

a. Experiment with the number of mixtures and comment on the tradeoff between the number of mixtures and goodness of fit (i.e. loglikelihood) of the data. Suggestion: Plot the loglikelihood as a function of the number of components of a mixture of Gaussians to support your argument.

b. Find a fixed number of Gaussians that works well for each data set.

c. Plot the estimated Gaussians as one-sigma countours of each mixing component on top of the training data.

d. List mean, covariance and mixing weights of each mixture component.

e. Include your source code.

Solution:

To solve this we use the mixturecode downloaded from msu.edu. We have to modify the code slightly to have the option of isotropic covariances. In the code attached you'll notice there is an covoption '4' added, and in the M step there is an additional elseif that handles the variance estimate. Additionally, we added another output so the code returns a vector of the logliklihoods for each number of mixture components. Figure 1 shows these loglikelihood as a function of the number of components of a mixture of Gaussians for each data set. By the logliklihood graphs we see that Data1 needs a higher number of mixtures than Data2, the likelihood of Data2 levels off around k=10, whereas Data1 it is around k=14. Figures 2 and 3 show several plots for each dataset of the one-sigma contours of different numbers of mixing components.

Your means, covariances, and mixing weights will vary based on your estimation. The main issues is to make sure the you did isotropic covariance, $\Sigma_j = \sigma_j I$. You should report something like the following for each dataset, here well report just Data1:

Data1 best number of mixing components: 14

Means for these 14 mixing components:

$$\mu_{j=1:7} = \begin{bmatrix} -9.7238 & 1.4277 & 1.8600 & -4.3273 & -3.6817 & 9.8741 & -7.8339 \\ -1.4321 & -9.6724 & 9.7593 & -8.8903 & 9.1494 & -0.6459 & 6.0534 \end{bmatrix}$$

$$\mu_{j=7:14} = \begin{bmatrix} -7.6491 & 6.2195 & -10.0384 & -1.4586 & 8.0498 & 9.3336 & 4.6449 \\ -6.2729 & 8.0161 & 2.3012 & -10.0600 & -5.6900 & 3.9897 & -8.5214 \end{bmatrix}$$

Covarances for these 14 mixing components:

$$\Sigma_1 = \begin{bmatrix} 1.4499 & 0 \\ 0 & 1.4499 \end{bmatrix} \Sigma_2 = \begin{bmatrix} 0.8775 & 0 \\ 0 & 0.8775 \end{bmatrix} \Sigma_3 = \begin{bmatrix} 1.8164 & 0 \\ 0 & 1.8164 \end{bmatrix}$$

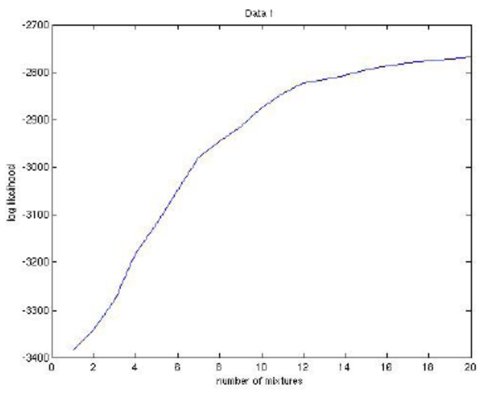
$$\Sigma_4 = \begin{bmatrix} 1.0715 & 0 \\ 0 & 1.0715 \end{bmatrix} \Sigma_5 = \begin{bmatrix} 1.9517 & 0 \\ 0 & 1.9517 \end{bmatrix} \Sigma_6 = \begin{bmatrix} 1.9618 & 0 \\ 0 & 1.9618 \end{bmatrix}$$

$$\Sigma_7 = \begin{bmatrix} 1.1109 & 0 \\ 0 & 1.1109 \end{bmatrix} \Sigma_8 = \begin{bmatrix} 1.3383 & 0 \\ 0 & 1.3383 \end{bmatrix} \Sigma_9 = \begin{bmatrix} 1.2574 & 0 \\ 0 & 1.2574 \end{bmatrix}$$

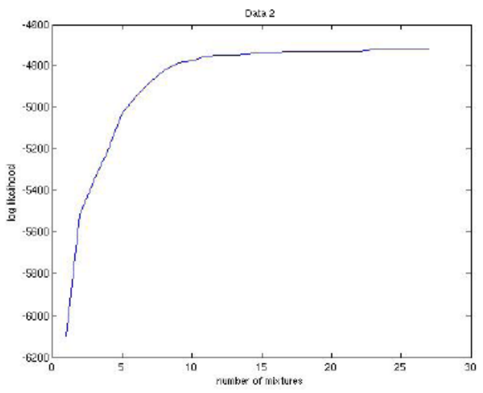
$$\Sigma_{10} = \begin{bmatrix} 1.6882 & 0 \\ 0 & 1.6882 \end{bmatrix} \Sigma_{11} = \begin{bmatrix} 0.8612 & 0 \\ 0 & 0.8612 \end{bmatrix} \Sigma_{12} = \begin{bmatrix} 1.1077 & 0 \\ 0 & 1.1077 \end{bmatrix}$$

$$\Sigma_{13} = \begin{bmatrix} 1.8865 & 0 \\ 0 & 1.8865 \end{bmatrix} \Sigma_{14} = \begin{bmatrix} 1.3117 & 0 \\ 0 & 1.3117 \end{bmatrix}$$

mixweights $(1:14) = 0.0669, 0.0645, 0.0843, 0.0472, 0.0898, 0.1201, 0.0647, 0.0588, 0.0625, 0.0784, 0.0502, 0.0740, 0.0716, 0.0670$

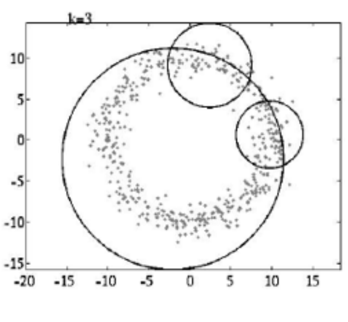


(a) Data1

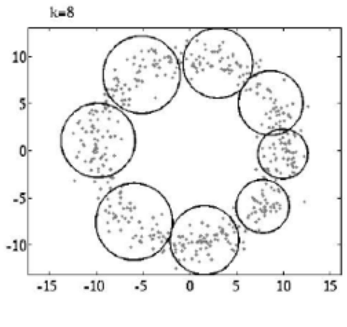


(b) Data2

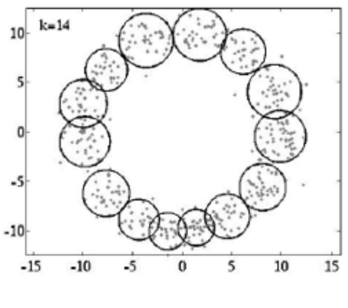
Figure 1: Loglikelihood as a function of the number of components of a mixture of Gaussians.



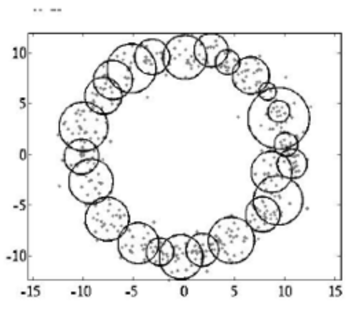
(a) k=3



(b) k=8

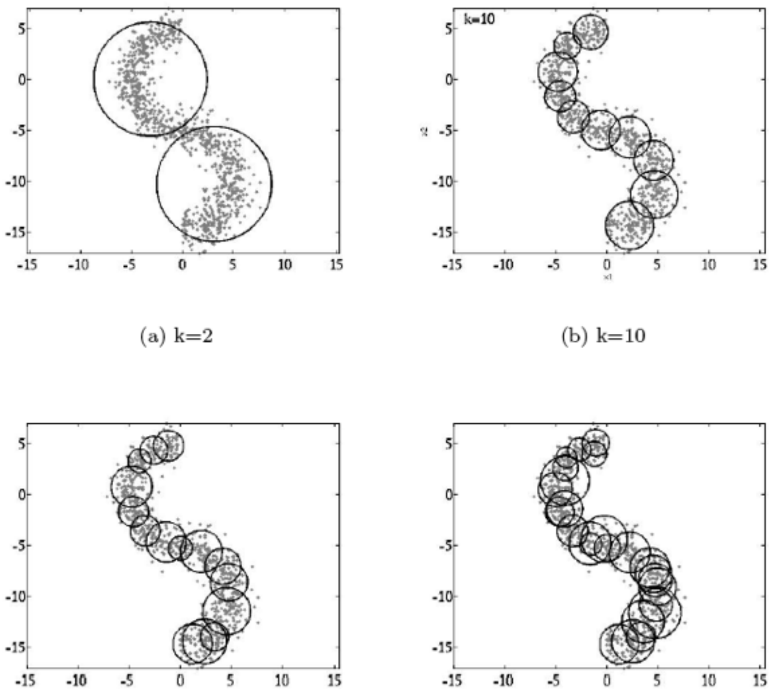


(c) k=14



(d) k=25

Figure 2: One-sigma contours of various numbers of mixing components for Data1, note Fig. 2(c) shows the number 14, which was estimated as optimal for this set.



(a) k=2 (b) k=10

(c) k=15 (d) k=25

Figure 3: One-sigma contours of various numbers of mixing components for Data2, note Fig. 3(b) shows the number 10, which was estimated as optimal for this set.

```
% This code uses the data clustering by mixtures of gaussians code
% written by Mario Figueiredo
% Available for download at: dataclustering.cse.msu.edu
% Described in the paper:
% M. Figueiredo and A.K.Jain, "Unsupervised learning of finite mixture models",
% IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 24, no. 3,
% pp. 381-396, March 2002.
%
% The mixtures4 function was modified slightly to have an option of
% isotropic covariances and to return a vector of the loglikelihoods of each k value,
% Excerpts from this modified version of mixtures4 are included below.
%
% There are two datasets we want to model. For each we will use the
% mixtures4 function to run through a range of possible numbers of
% mixtures. This function expects 6 inputs
% 1) data: for n observations in d dimensions, data must have
% d lines and n columns.
% 2) min number of clusters to try; well use 1
% 3) max number of clusters to try; well use 25
% 4) regularizing factor; well use 0
% 5) stopping threshold; well use 1e-4
% 6) covariance option; well use isotropic, 4
% load the data and transpose it so its in the expected form
y = load('problem2/data1train.dat');

y = transpose(y);

[bestk,bestpp,bestmu,bestcov,dl,countf, loglikes] = mixtures4(y,1,25,0,1e-4,4)

% mixtures4 plots the 1-sigma circles on the data for each k value, now we
% plot the logliklihood of each k value
figure;
plot(loglikes);

%list the mean, covariance and mixing weights of each mixture component,
%for the bestk
fprintf(1,'Data1 best number of mixing components:%d',bestk);

fprintf(1,'Means for these %d mixing components:',bestk);
bestmu

fprintf(1,'Covarances for these %d mixing components:',bestk);
bestcov

fprintf(1,'Mixing weights for these %d mixing components:',bestk);
bestpp

% same for data2
y = load('problem2/data2train.dat');
y = transpose(y);
[bestk,bestpp,bestmu,bestcov,dl,countf, loglikes] = mixtures4(y,1,30,0,1e-4,4)
figure;
plot(loglikes);
fprintf(1,'Data2 best number of mixing components:%d',bestk);
fprintf(1,'Means for these %d mixing components:',bestk);
bestmu
fprintf(1,'Covarances for these %d mixing components:',bestk);
bestcov

fprintf(1,'Mixing weights for these %d mixing components:',bestk);
bestpp
```

```
% Now well include the parts of mixtures4 that needed to change to handle isotropic
% covariance, and returning the loglikelihoods
% Add an output to the function call...

function [bestk,bestpp,bestmu,bestcov,dl,countf,loglikes] = ...
    mixtures4(y,kmin,kmax,regularize,th,covoption)
% initialize a variable to keep track of the best logliklihood of each
% mixing component tried
...
verb=1; % verbose mode; change to zero for silent mode
bins = 40; % number of bins for the univariate data histograms for visualization
dl = []; % vector to store the consecutive values of the cost function
[dimens,npoints] = size(y);
loglikes = []; %keep track of the log likelihood of each value of k
...
%% the M step is the only place we need to alter the code to handle our
%% covoption=4
...
% now we perform the standard M-step for mean and covariance
normalize = 1/sum(normindic(comp,:));
aux = kron(normindic(comp,:),ones(dimens,1)).*y;
estmu(:,comp)= normalize*sum(aux,2);

if (covoption == 0)|(covoption == 2)
    estcov(:,:,comp) = normalize*(aux*y') - estmu(:,comp)*estmu(:,comp)' ...
    + regularize*eye(dimens);
%%%%%%%%%%%%%%
%% This elseif stmt is added to handle a fourth type of
%% convariance, isotropic. i.e., the covariance matrix of each
%% of the j mixture components is diagonal and equal to sigma_j*I
%%%%%%%%%%%%%%
elseif(covoption == 4)
    normcalc = [];
    for kk=1:size(y,2)
    normcalc(kk) = (norm(y(:,kk) - estmu(:,comp))^2);
    end
    aux4 = normindic(comp,:).*normcalc;
    estcov(:,:,comp) = eye(dimens)*(normalize*(sum(aux4,2)))/dimens;
else
    estcov(:,:,comp) = normalize*diag(sum(aux.*y,2)) - diag(estmu(:,comp).^2) ;
end

if covoption == 2
```

```
    comcov = zeros(dimens,dimens);
    for comp2 = 1:k
    comcov = comcov + estpp(comp2)*estcov(:,:,comp2);
    end
    for comp2 = 1:k
    estcov(:,:,comp2) = comcov;
    end
end

if covoption == 3
    comcov = zeros(dimens,dimens);
    for comp2 = 1:k
    comcov = comcov + estpp(comp2)*diag(diag(estcov(:,:,comp2)));
    end
    for comp2 = 1:k
    estcov(:,:,comp2) = comcov;
    end
end
...
%When the threshold is met to move on to a new number of mixing components,
% save the best likelihood for this number of components in our loglikes var
...
% compute the change in loglikelihood to check if we should stop
deltlike = loglike(countf) - loglike(countf-1);
if (verb~=0)
    disp(sprintf('deltaloglike/th = %0.7g', abs(deltlike/loglike(countf-1))/th));
end

if (abs(deltlike/loglike(countf-1)) < th)
% if the relative change in loglikelihood is below the threshold, we stop CEM2
    cont=0;
    loglikes(k) = loglike(countf);
end
...
```

Problem 2: k-Means [30 points]

a. Prove that k-Means algorithm is guaranteed to converge to a solution. Use $C(i)$ as the cluster associated to the data point x_i, where $i = 1 \dots n$.

   Solution:

   The goal of K-means is to the select the μ and C(i) that minimize the within-cluster sum of squares:

$$J(C,\mu) = \sum_{i}^{n} (\mathbf{x}_i - \mu_{C(i)})^2$$

Given some initial parameters (i.e. C^0 and μ^0), the algorithm iteratively repeats the following two steps:

(a) Decide class memberships. The goal is to determine what cluster center is closest to each data point. Assuming a particular value of μ, we can minimize J by computing:

$$C(i)^{(t+1)} = \arg\min_{j} (\mathbf{x}_i - \mu_j^t)^2 \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, k$$

(b) Re-estimate cluster centers. The goal is to find μ that minimizes J. Assuming a particular value of C, this can be done by performing Maximum Likelihood Estimation. The result is:

$$\mu_j^{(t+1)} = \frac{1}{|C_j^{(t+1)}|} \sum_{x_i \in C_j^{(t+1)}} x_i$$

Given that both steps minimize the expression, we can see that $J(C^0,\mu^0) \leq J(C^1,\mu^0) \leq J(C^1,\mu^1) \leq J(C^2,\mu^1) \leq J(C^2,\mu^2) \ldots$. Therefore, K-means is converging and, eventually, will reach a local optimum. These two steps can been seen as the Expectation Maximization algorithm where the first step corresponds to the E-step and the second step corresponds to the M-step.

b. Does the algorithm always converge to the same solution? Prove this or provide a counter example.

Solution:

For different initialization of the clusters centers (a.k.a. centoids) we obtain different clusterings. Figure 1 shows different possibilities for k = 2.

Problem 3: k-Nearest-Neighbor vs Generalized Linear Discriminant [35 points]

Compare the performance of Generalized Linear Discriminant (GLD) and kNN to detect cases of breast cancer [1]. For each of the methods you will need to find their optimal parameter. That is, k for kNN and the polynomial degree p for GLD (i.e. $y^T = [1\ x\ x^2\ \ldots x^p]$). For the latter case, let the margin vector be $\mathbf{b} = \mathbf{1}$.

These undetermined parameters will be estimated using leave-one-out validation as follows:

[1] This is an altered version of the Breast Cancer Wisconsin Data Set located at the UCI Machine Learning repository (http://archive.ics.uci.edu/ml/datasets.html)

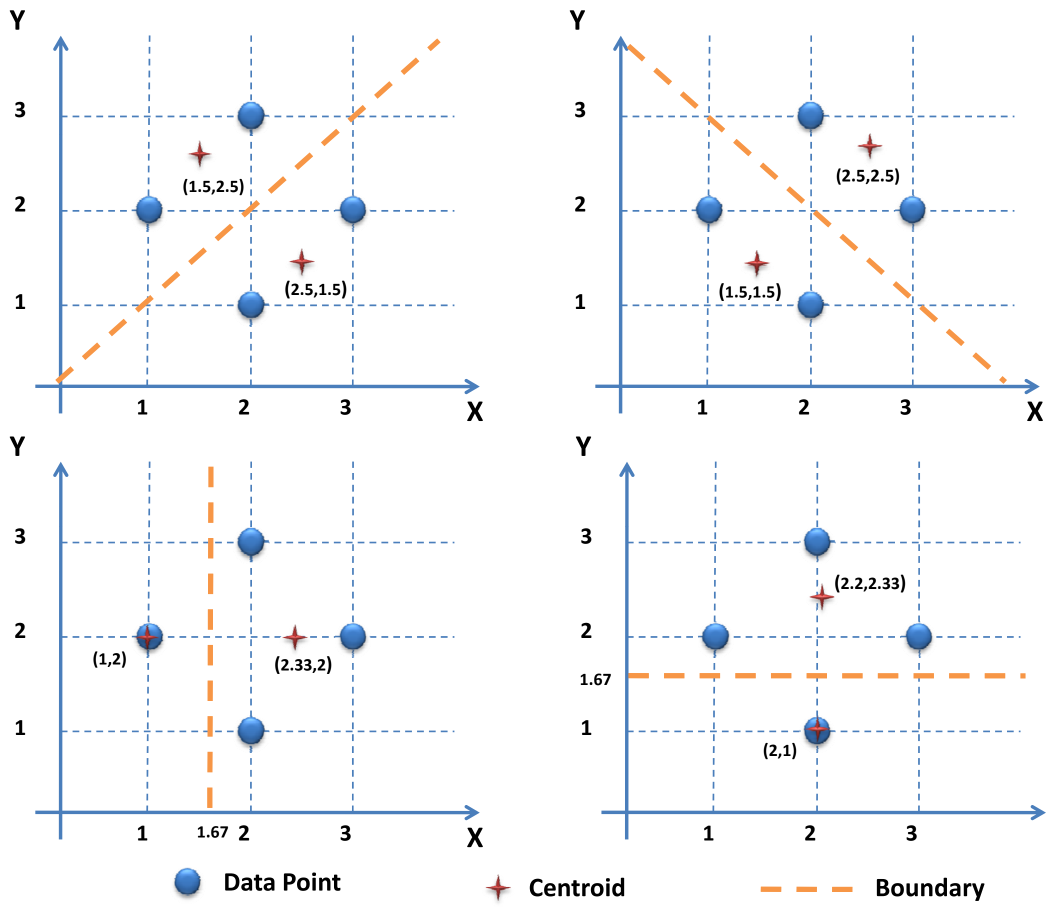


Figure 1: K-means clustering (k = 2) with four 2D points

- Break the training data set into two parts, A and B, where B contains only a single sample.
- Train your algorithms using A only.
- Determine if B (the sample left out) is classified correctly.
- Repeat this process for every possible choice of B. Define $cv(h)$ as the leave-one-out accuracy of one algorithm for the parameter h.

a. Use the training data to generate $cv(i)$ for both algorithms. You should constrain the parameter search to $i = [1 : 2 : 20]$ for kNN and $i = [1 : 7]$ for GLD. Plot your cv's and indicate the best parameters. [You can use the helper function "*[Xtrain Xtest Ytrain Ytest] = loadData;*"]

Solution: Figure 2 shows cv's for both algorithms. The best parameters are $k = 5$ (96.6% accuracy) for kNN, and $p = 2$ (96% accuracy) for GLD.

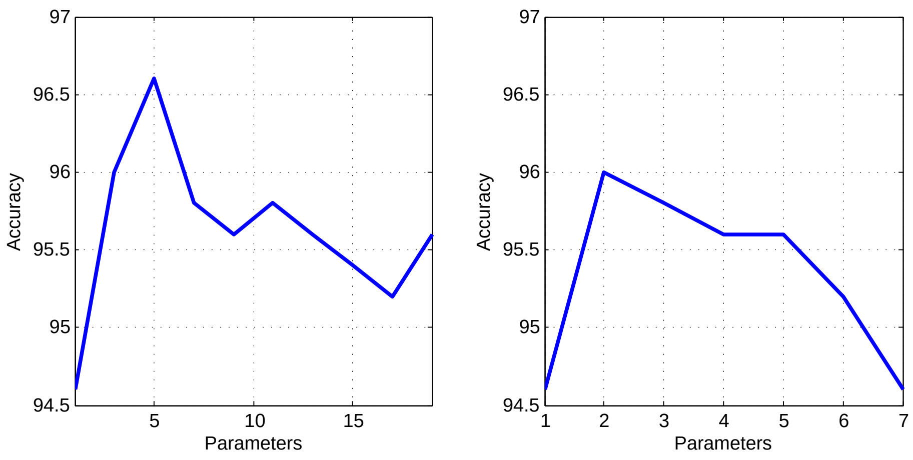


Figure 2: (Left) cv of kNN, (right) cv of GLD

b. Use the previous parameters and all your training data to re-train your algorithms. What are the accuracies for the testing data set?

Solution: The final testing accuracy is 98.99% for kNN and 99.49% for GLD. The MATLAB code is as follows:

```
function main()
%Code written by Javier Hernandez Rivera (javierhr@mit.edu)

%Load the data
load('data');

%Choose algorithm
algorithm = 'KNN';
algorithm = 'Discriminant';

%Learn optimal parameters
[best_param] = learnParam(Xtrain,Ytrain,algorithm);

%Train final model
finalModel = trainModel(Xtrain, Ytrain,best_param,algorithm);
%Get predictions
testPredictions = testModel(Xtest,algorithm,finalModel);
finalScore =  computeScore(testPredictions, Ytest)

end


function [best_param] = learnParam(Xtrain,Ytrain,algorithm)
%Learn optimal parameters
n = size(Xtrain,1);
```

```
%Define search space for parameters
switch algorithm
    case 'KNN'
        params = [1:2:20];
    case 'Discriminant'
        params = [1:7];
end

%Start leave one out cross validation
cv_score = zeros(1,length(params));
indices = 1:n;
for i = 1:length(params)%for all parameters
    curr_param = params(i);
    fprintf('%d/%d - Parameter %d\n',i,length(params),curr_param);

    prediction = zeros(n,1);
    for j = 1:n%leave one out cross validation
        %Select sub-training and sub-testing indices
        train = [indices ~= j];
        test = [indices == j];

        model = trainModel(Xtrain(train,:), Ytrain(train),curr_param,algorithm);
        prediction(j) = testModel(Xtrain(test,:),algorithm,model);
    end
    %compute accuracy for one parameter
    cv_score(i) = computeScore(prediction,Ytrain);
end


%Find best parameter
[v p] = max(cv_score);
best_param = params(p);
fprintf('Best parameter [%d] with leave one out accuracy of [%.2f%%]\n',best_param,cv_sc

%Show CVs
figure;
plot(params,cv_score,'LineWidth',2);
grid on; xlim([1 params(end)]);
xlabel('Parameters'); ylabel('Accuracy');
title(algorithm);
end


function score = computeScore(prediction,Y)
```

```
%compute accuracy
score = (sum(prediction == Y)/length(prediction))*100;
end



function model = trainModel(X, Y,curr_param,algorithm)
%Learn the parameters for each algorithm
switch algorithm
    case 'KNN'%There is no training for KNN
        model.X = X;
        model.Y = Y;
        model.K = curr_param;
    case 'Discriminant'
        %Add intercept and expand exponential terms for X
        newX = expandX(X,curr_param);
        idx = find(Y == -1);
        newX(idx,:) = newX(idx,:).*-1;
        model.B = 1;
        B = ones(size(X,1),1)*model.B;
        %Compute pseudoinverse
        model.W = newX\B;

        model.K = curr_param;
end
end


function pred = testModel(X,algorithm,model);
switch algorithm
    case 'KNN'
        for i = 1:size(X,1)%for each testing sample
            %Compute Euclidean distance
            [vals ord] = sort(sqrt(sum((model.X - repmat(X(i,:),size(model.X,1),1)).^2,2
            %Find the most common class label in K
            pred(i) = mode(model.Y(ord(1:model.K)));
        end

    case 'Discriminant'
        for i = 1:size(X,1)
            %Add intercept and expand exponential terms for X
            newX = expandX(X(i,:),model.K);
            pred(i) = double((model.W' * newX') > 0);
            pred(i) = pred(i)*2 - 1;%Transform class label to 1 and -1
        end
```

```
end
pred = pred';
end


function newX = expandX(X,curr_param)
%Add intercept and expand exponential terms for X
n = size(X,1);
d = size(X,2);
newX = zeros(n,d*curr_param);%memory allocation
newX(:,1:d) = X;
for i = 2:curr_param
    start = d*(i-1)+1;%select indices
    newX(:,start:start+d-1) = X.^i;
end
newX = [ones(size(X,1),1) newX];
end
```