Q1

Part 1

Optimal Theoretical Classifier

The theoretically optimal classifier can be expressed as the minimum expected risk classification rule with 0-1 loss.

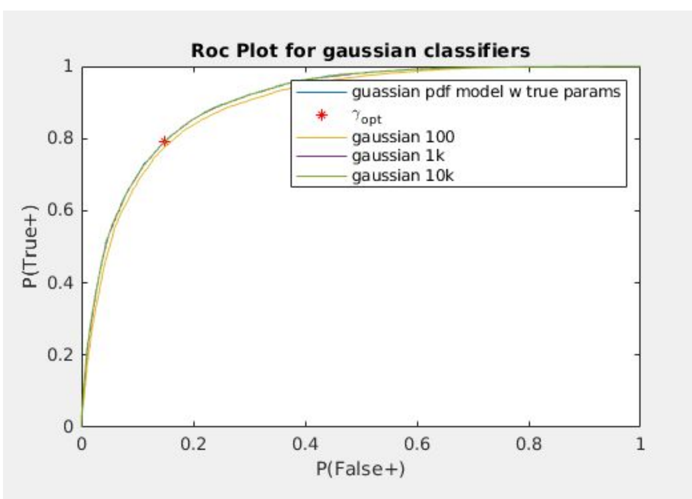
$$\frac{p(x|L=1)}{p(x|L=0)} \underset{D2}{\overset{D1}{\gtrless}} \frac{P(L=0)(\lambda_{10}-\lambda_{00})}{P(L=1)(\lambda_{01}-\lambda_{11})}$$

Substituting for 0-1 loss and problem priors

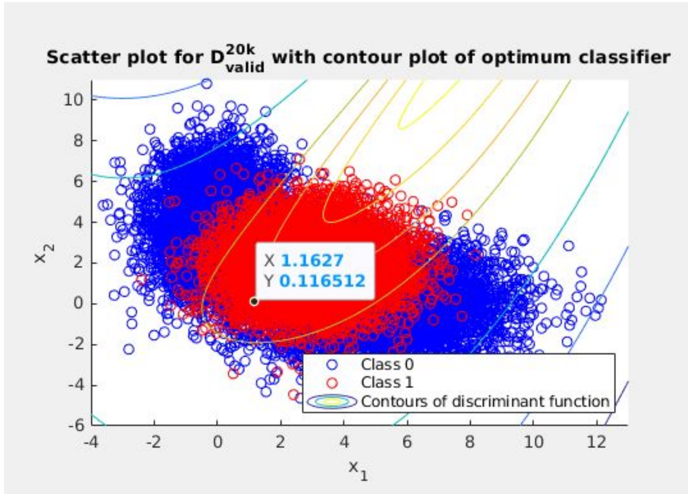
$$\frac{p(x|L=1)}{p(x|L=0)} \underset{D2}{\overset{D1}{\gtrless}} 1.3953$$

The above classifier when applied to the 20k validation set results in a perror = 0.1727

Below is the ROC plot for the discriminant scores evaluated using true knowledge of the data pdf (blue line).. Note highlighted operating point at threshold = 0.4189 and min perror = 0.1724. This score slightly outperformed the theoretically optimal classifier by a negligible perror.



Optimal decision boundary contour plot



Part 2

ML for parameter estimation

For both class posteriors the ML parameter estimation technique can be expressed as

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}} \ \mathcal{L}(\Theta|\mathcal{X}).$$

For both class posteriors Θ^* are the parameters that maximise the likelihood of the data. For L=0, Θ represents the mean, covariance, and component proportion matrices for the gaussian mixture. For L=1, Θ represents the mean and covariance of the multivariate gaussian. For the multivariate gaussian, the mean and covariance derived from MLE are

$$\widehat{\boldsymbol{\mu}}_{ML} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i.$$

$$\widehat{\boldsymbol{\Sigma}}_{ML} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T.$$

Where $\mu = \widehat{\mu}_{ML}$

Below is the iterative algorithm for finding the optimal parameters using the expectation maximization algorithm for the L=0 class posteriors.

$$p(y_i|x_i, \Theta^g) = \frac{\alpha_{y_i}^g p_{y_i}(x_i|\theta_{y_i}^g)}{p(x_i|\Theta^g)} = \frac{\alpha_{y_i}^g p_{y_i}(x_i|\theta_{y_i}^g)}{\sum_{k=1}^M \alpha_k^g p_k(x_i|\theta_k^g)}$$

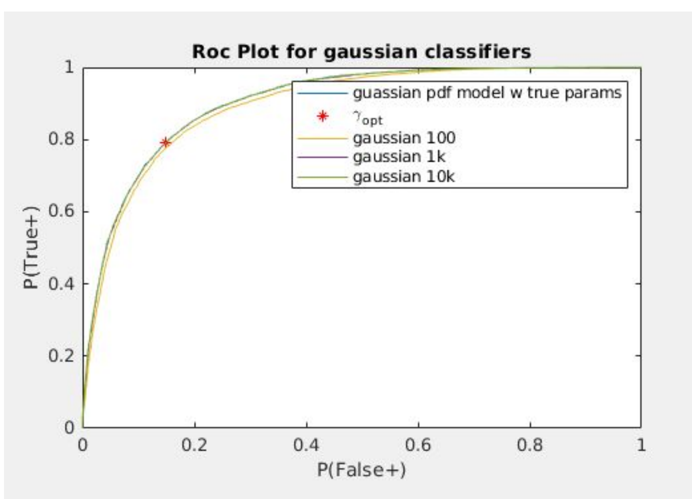
$$\alpha_\ell^{new} = \frac{1}{N}\sum_{i=1}^N p(\ell|x_i, \Theta^g)$$

$$\mu_\ell^{new} = \frac{\sum_{i=1}^N x_i p(\ell|x_i, \Theta^g)}{\sum_{i=1}^N p(\ell|x_i, \Theta^g)}$$

$$\Sigma_\ell^{new} = \frac{\sum_{i=1}^N p(\ell|x_i, \Theta^g)(x_i - \mu_\ell^{new})(x_i - \mu_\ell^{new})^T}{\sum_{i=1}^N p(\ell|x_i, \Theta^g)}$$

ROC curve for class posterior parameters trained with different datasets

Below is the ROC curve that illustrates performance of the ERM classifier applied to class posterior models trained with different datasets. The upper bound for classifier performance is also present in the below figure.



Effect of training on error performance of the trained gaussian models

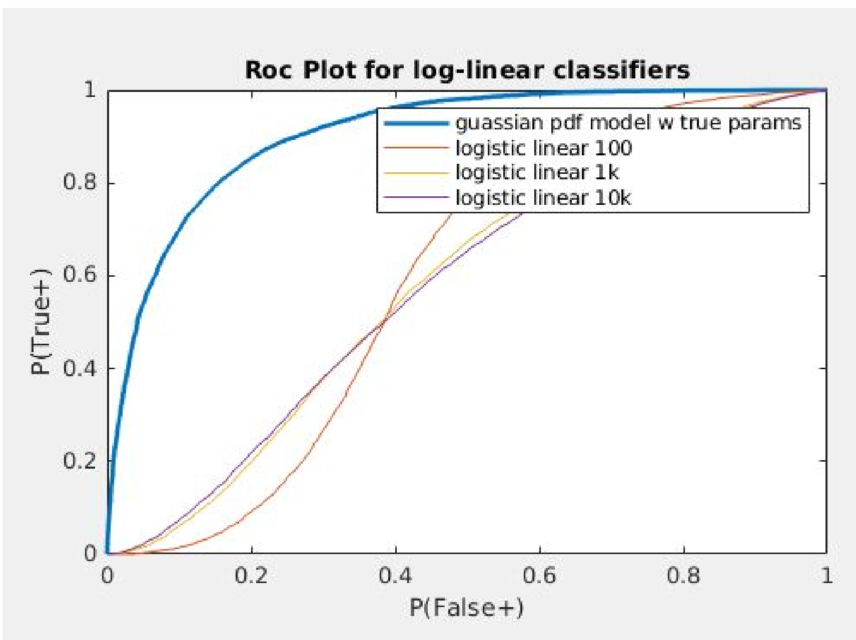
The below table shows the effect of additional data on trained model performance. Since the model chosen matches the original model for the generated dataset D_Valid it is understandable that all trained models perform close to the upper bound with performance saturating at the 1k dataset. Fewer samples (D_100) results in worse error performance but not by much.

Training Dataset	Perror
D_100	0.1784
D_1k	0.1729
D_10k	0.1729

Part 3

ROC Logistic-Linear-Model

Below is the ROC curve that illustrates the performance of the log-linear model. Given the nature of the random vector X class 0 and class 1 are not linearly separable and thus the performance of the linear model suffers.

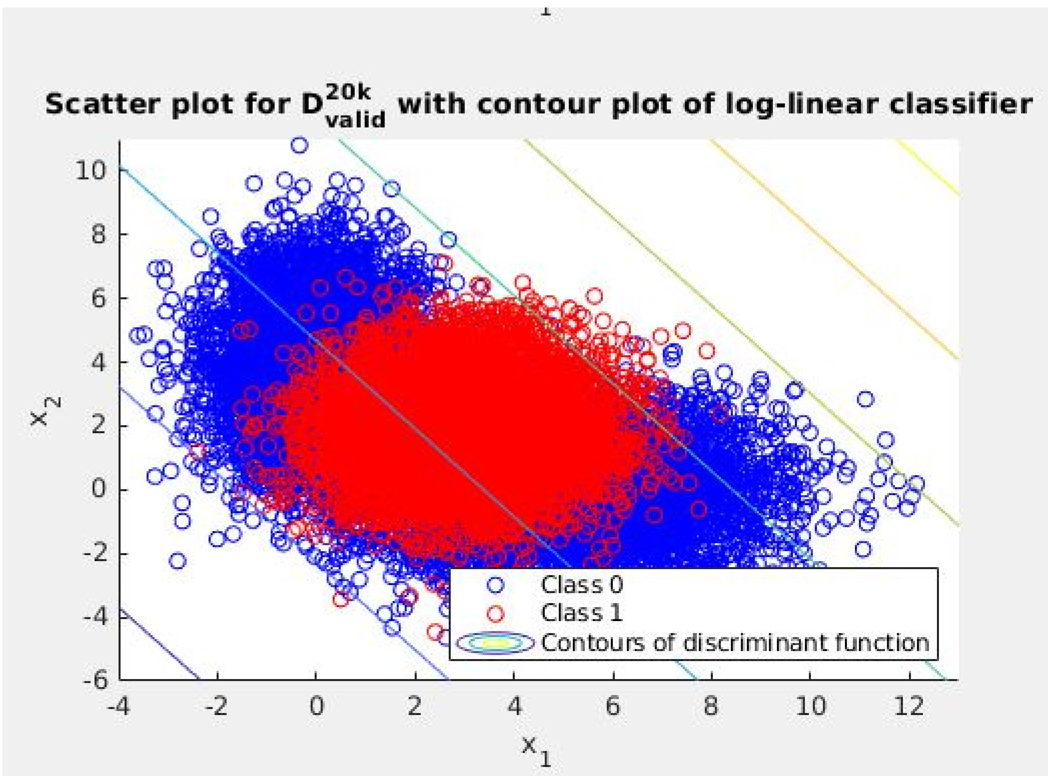


Error performance Logistic-Linear-Model

Regardless of the amount of training data, performance of the linear model is abysmal across the board which is logical given the limitations of the model.

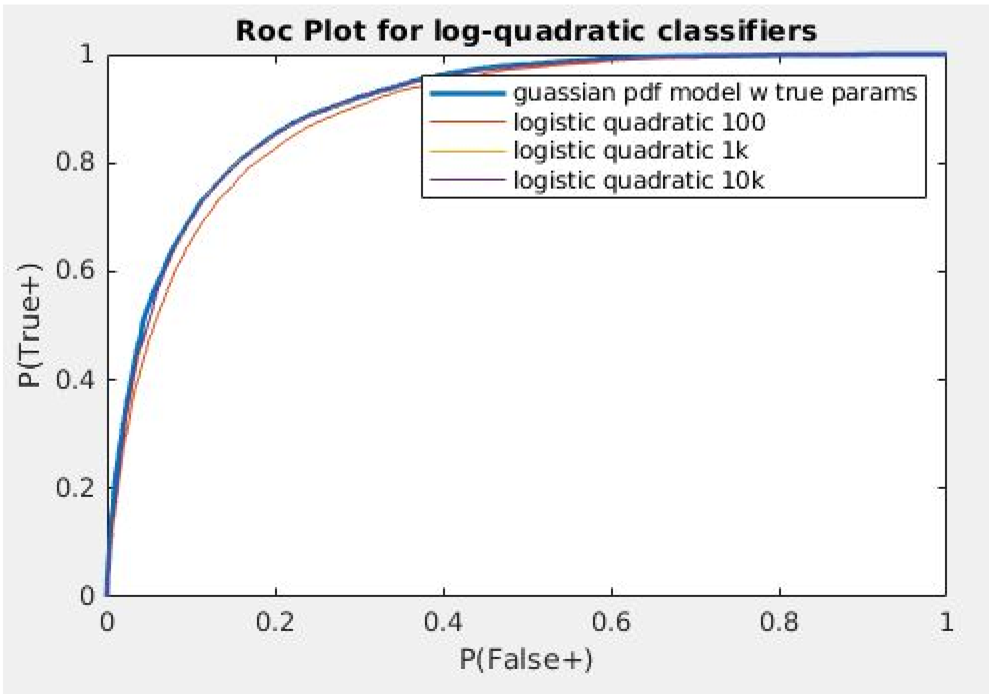
Training Dataset	Perror
D_100	0.3889
D_1k	0.4053
D_10k	0.4053

Below is the scatter plot for the data plus the linear decision boundary produced by the linear model.



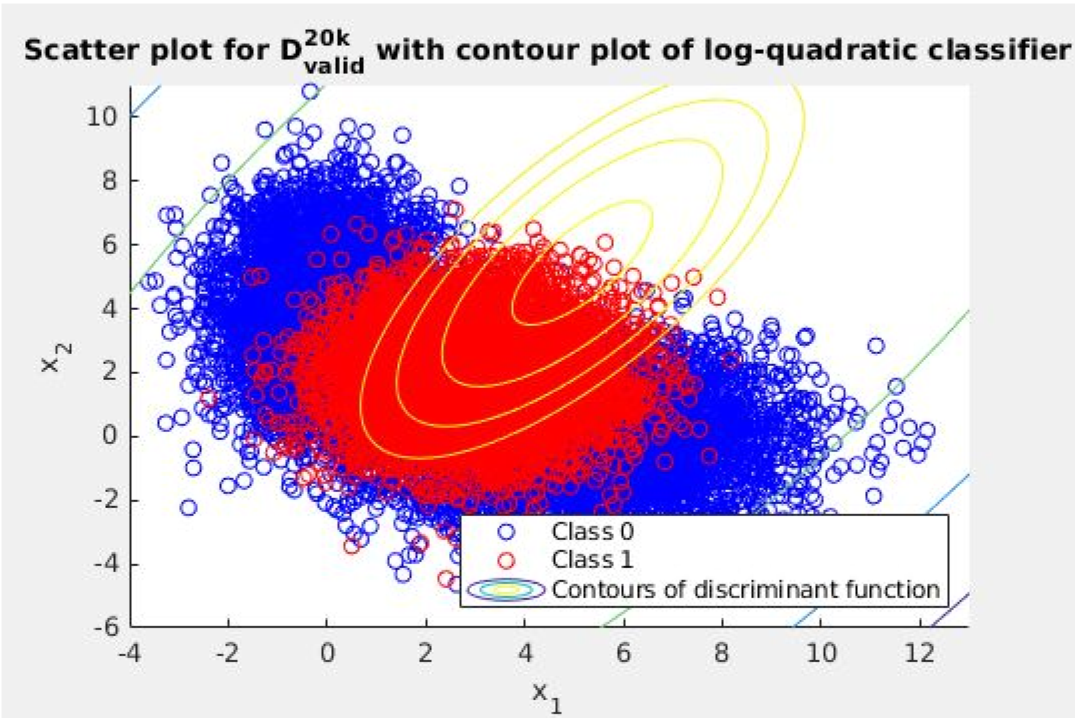
ROC Logistic-quadratic-Model

Below is the ROC curve that illustrates the performance of the log-linear model. Performance is comparable to the models in Part-2. Unlike the models in part-2 no amount of training data will enable the quadratic model to asymptotically approximate the theoretically optimal classification rule as indicated by the slight reduction in model performance for the 10k dataset.



Training Dataset	Perror
D_100	0.1844
D_1k	0.1727
D_10k	0.1734

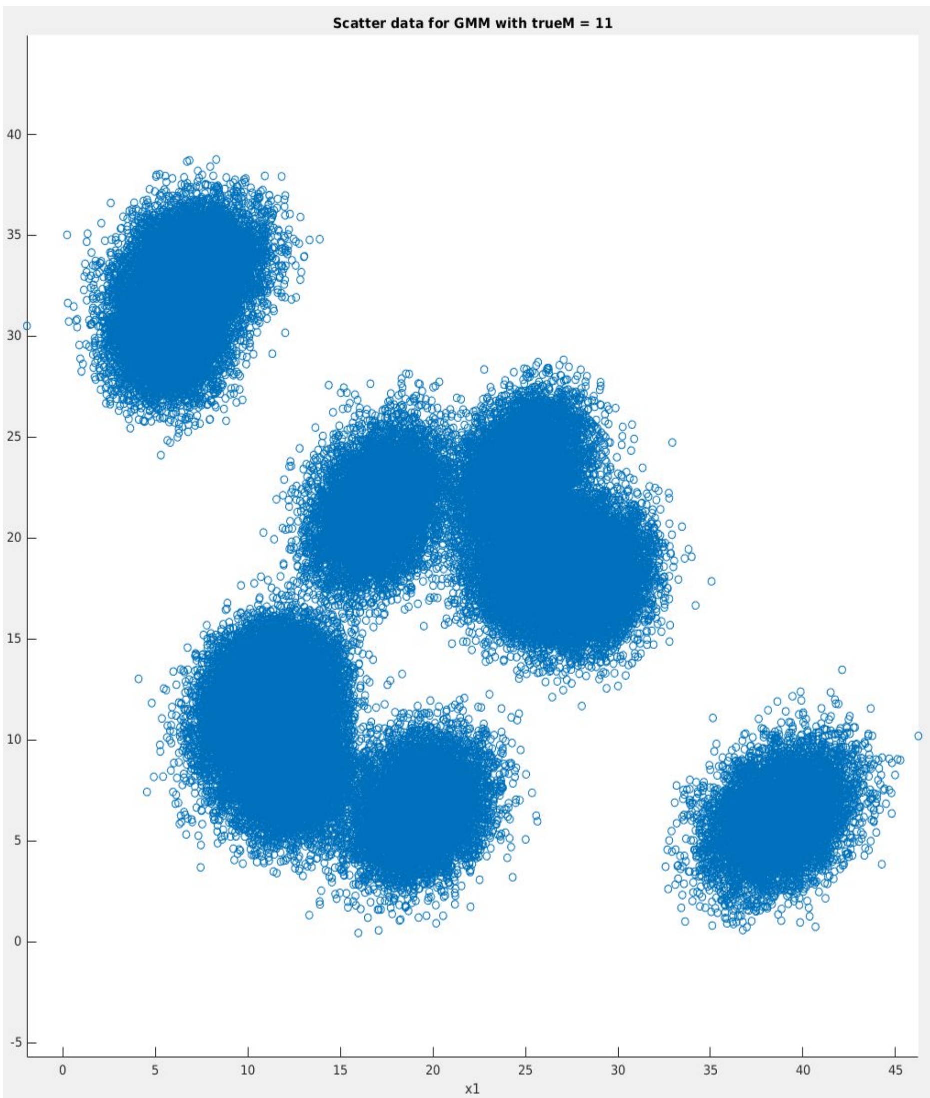
Below is the scatter plot for the data plus the quadratic decision boundary produced by the logistic-quadratic model.



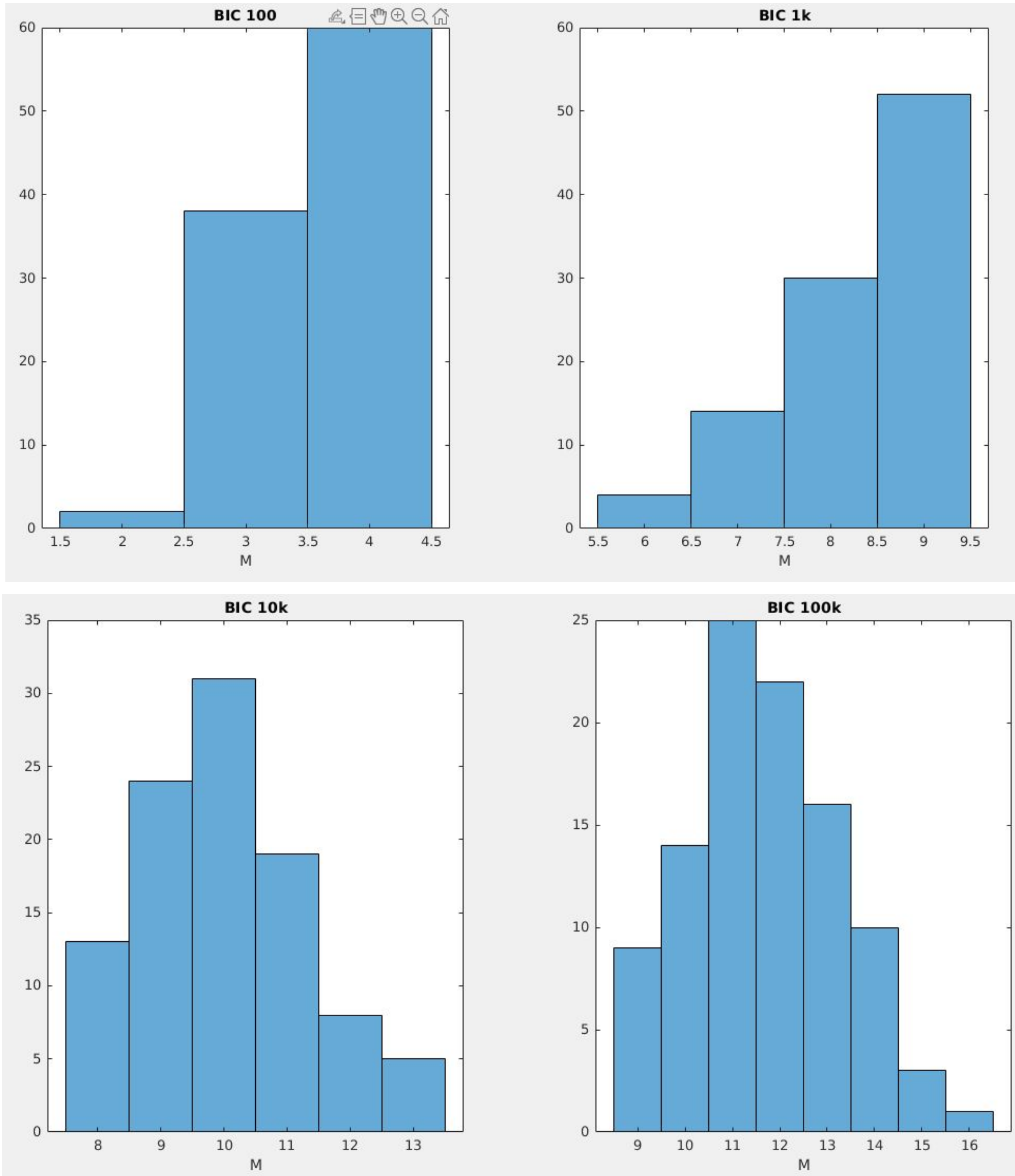
Q2

BIC

Initially I misread the requirements of the data and assumed that that the means of the gaussian mixtures do not have to be on the same axis as illustrated in the below scatter plot. Additionally, I assumed the experiments were run on the same dataset. So the below BIC and Kfold histograms pertain to the same dataset. This was corrected but the results acquired were odd and I ran out of time to fix it.

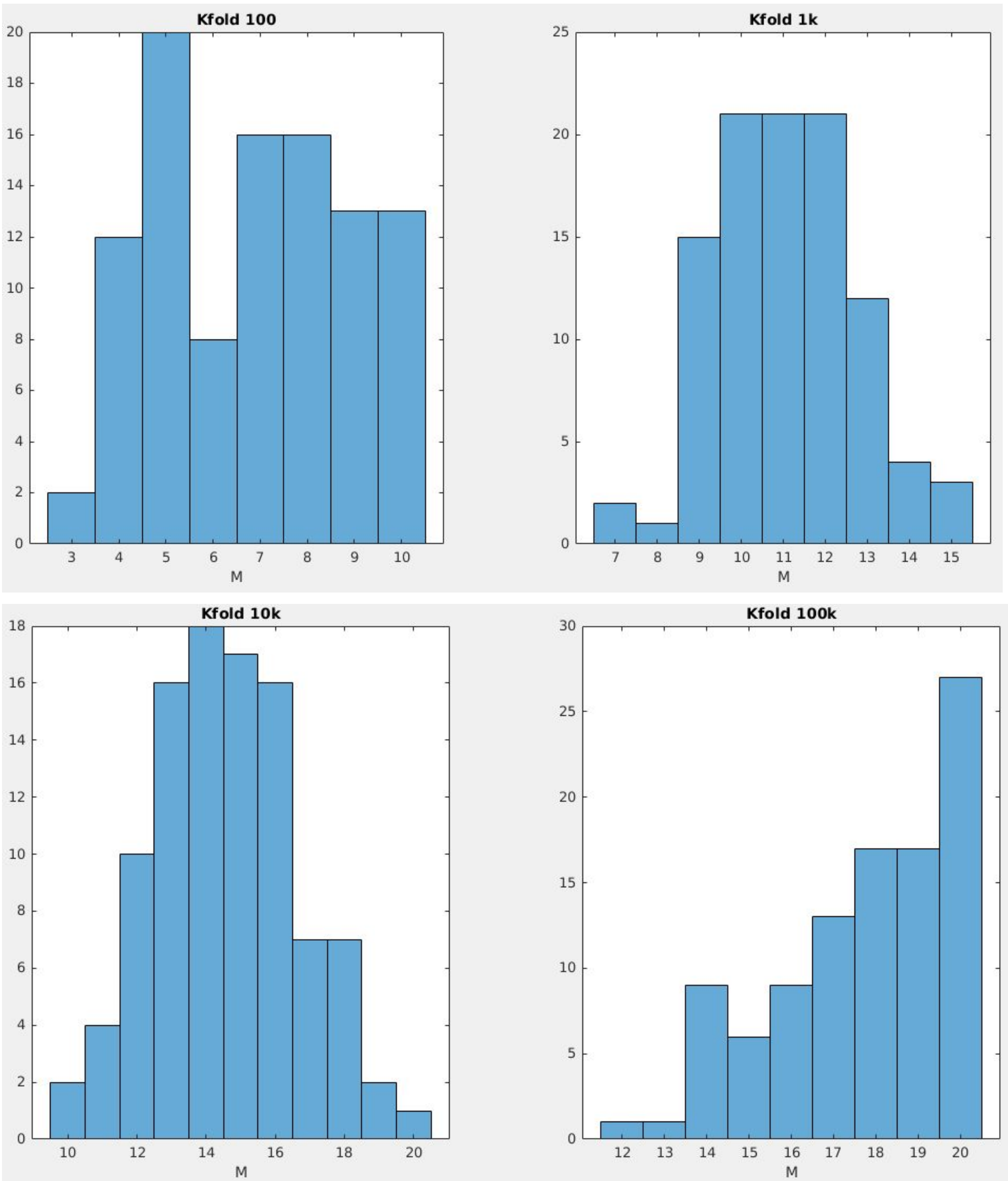


Under the same data assumption BIC benefitted with more data by eventually converging to the actual true M value for the dataset.

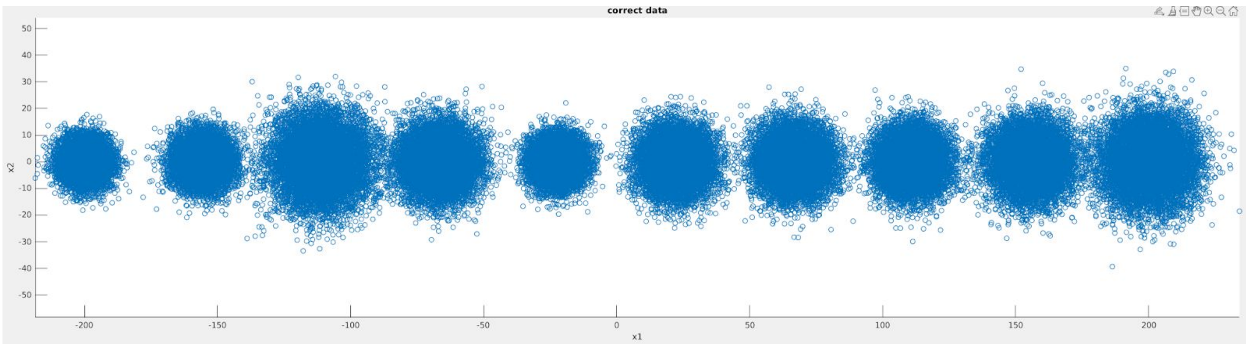


Kfold

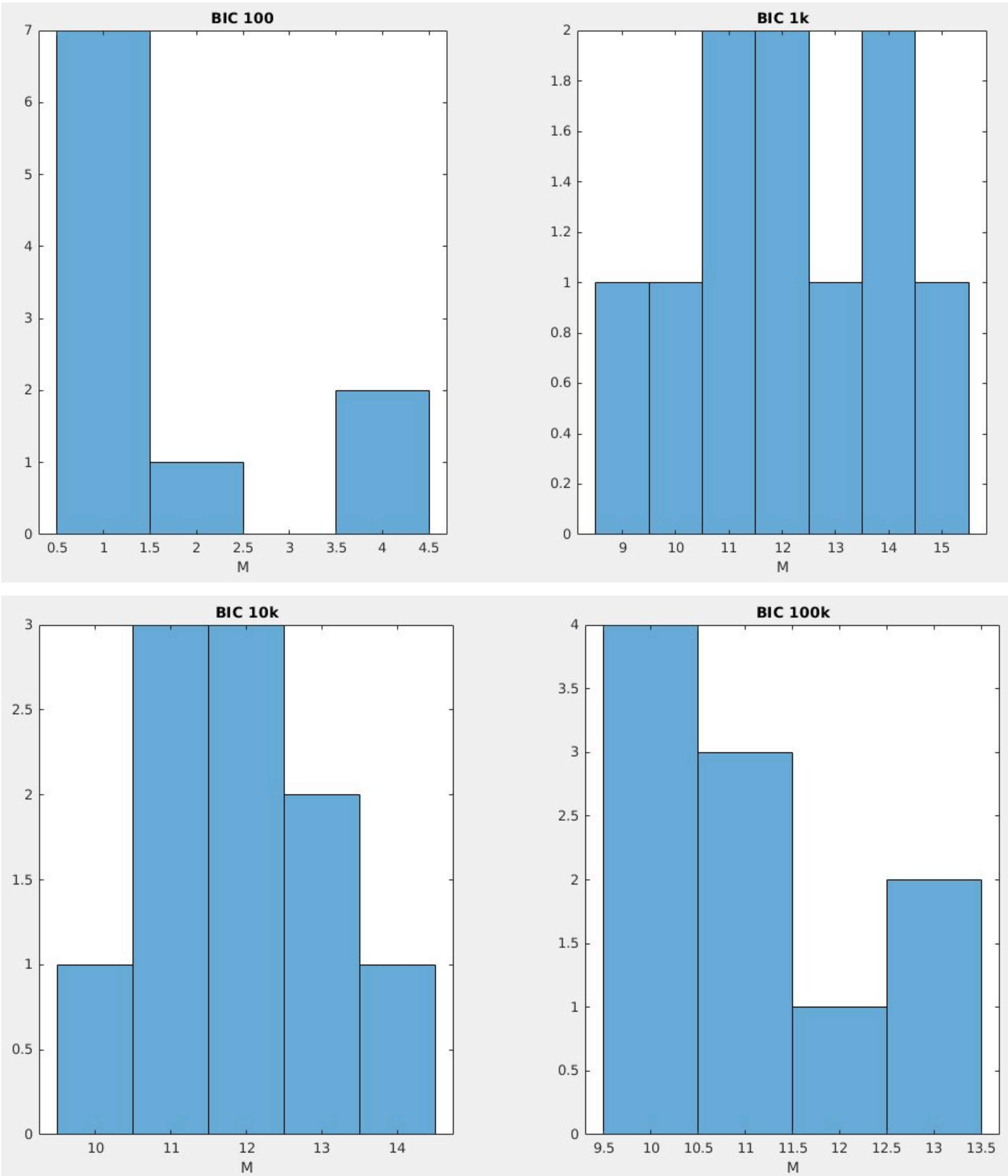
Kfold unfortunately started favouring higher model orders with more data. This trend was discussed in class. Kfold will not penalize higher model orders unless they are detrimental to data log likelihood so regularization in the EM algorithm may fix this. Regardless, if that is not the case then there is likely a bug in the code. K = 5 was chosen for the below run.



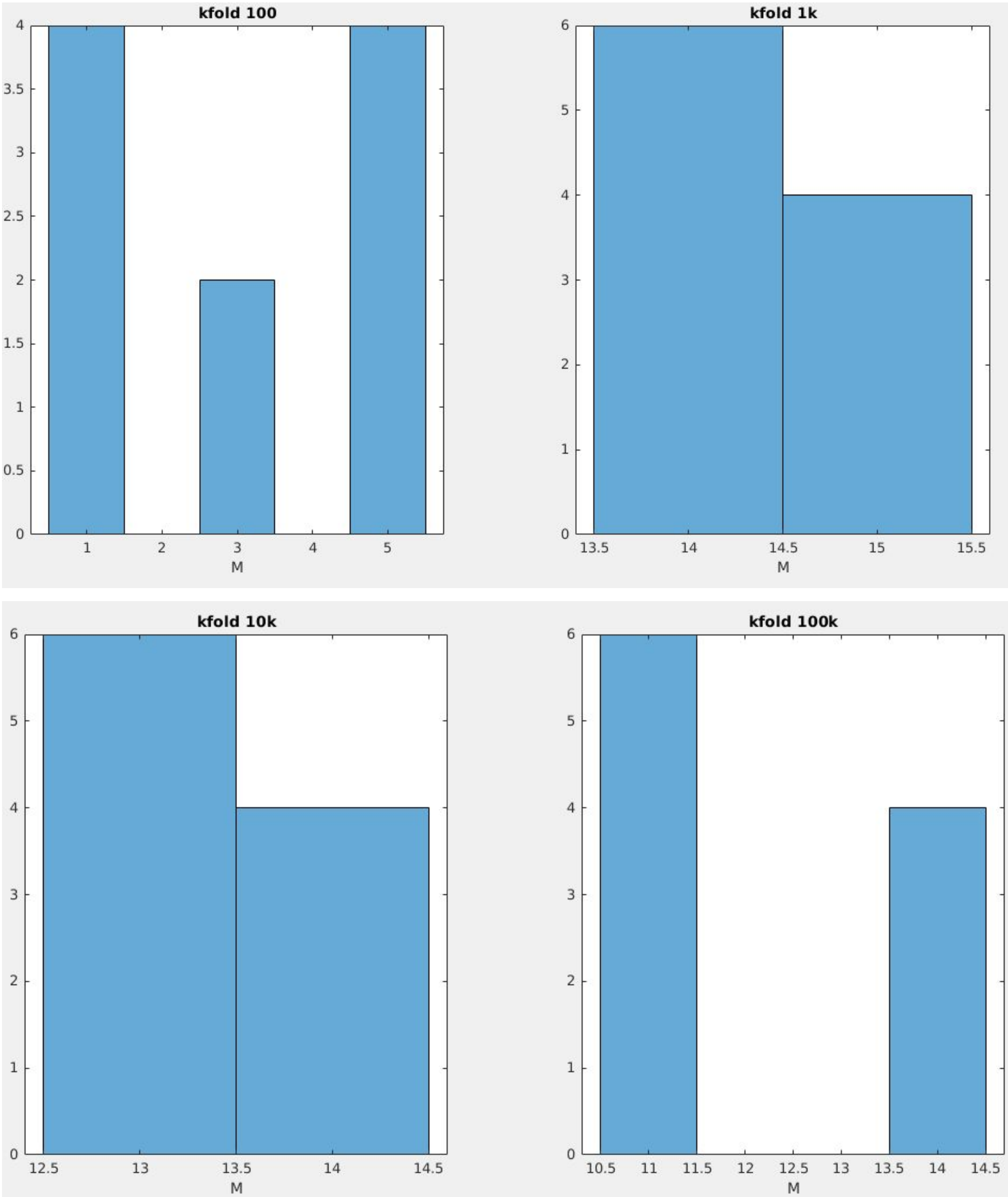
An additional 10 experiments were run on the 10 different datasets with distributions similar to the below scatter plot with 10 mixture components (as expected by the assignment) for both kfold and BIC. Only 10 additional experiments were run because by the time I realized my mistake it was too late to rerun more than 10 experiments. Additionally, the internal matlap fitgmmdist function was used with 1 replicate and with a regularization value = 1e-1.



BIC results remained consistent with the above with more data resulting in a model order closer to the true model order being selected (10).



Kfold results improved for the 10 experiments with more data resulting in a model order close to the true one being selected. At kfold 100k a model order of 11 was chosen more consistently as the best model order. Given the low k value chosen for this experiment (K=5), and the low experiment count. The below results may be valid and indicative of bug free code.



Appendix

Q1 Code

```
close all, clear all

% define dataset parameters
row = 3; col = 2;
scatter_plot_index_gaussian = 1;
roc_gaussian_plot_index = 2;
scatter_plot_index_logistic_linear = 3;
roc_logistic_linear_plot_index = 4;
scatter_plot_index_logistic_quadratic = 5;
roc_logistic_quadratic_plot_index = 6;

parameters.alpha{1} = [0.5, 0.5];
parameters.mu{1} = [5 0; 0 4]';
parameters.Sigma{1}(:,:,1) = [4 0;0 2];
parameters.Sigma{1}(:,:,2) = [1 0;0 3];

parameters.alpha{2} = 1;
parameters.Sigma{2}(:,:,1) = [2 0;0 2];
parameters.mu{2} = [3 2];

parameters.priors = [0.6 0.4];

% generate datasets
data.train_100 = generateGMMData(parameters, 100);
data.train_1k = generateGMMData (parameters, 1000);
data.train_10k = generateGMMData(parameters, 10000);
[data.valid_20k, gmmtrue] = generateGMMData(parameters, 20000);
fn = fieldnames(data);

% gaussian w true pdf
tau_true = log(parameters.priors(1)/parameters.priors(2));
a = likelyhood_ratio(gmmtrue, data.valid_20k.features);
results.gaussian.truepdf = ERMeval(descriminantScores.truepdf, data.valid_20k.labels,
tau_true);
results.gaussian.roc.truepdf = ROCcurve(descriminantScores.truepdf,data.valid_20k.labels);

figure(1)
subplot(row, col, scatter_plot_index_gaussian);
```

```
gmm_scatter_plot(data.valid_20k);
contour_plot_gaussian(data.valid_20k, 500, gmmtrue, tau_true);
title('Scatter plot for D_{valid}^{20k} with contour plot of optimum classifier');

% gaussian w training

for k=1:numel(fn)-1
    display(['fitting gmm parameters from ', fn{k}]);
    gmm.(fn{k}) = estimate_pdf_params(data.(fn{k}));
    display(['evaluating descriminant scores ', fn{k}]);
    descriminantScores.gaussian.(fn{k}) = likelyhood_ratio(gmm.(fn{k}), data.valid_20k.features);
    results.gaussian.roc.(fn{k}) =
ROCcurve(descriminantScores.gaussian.(fn{k}),data.valid_20k.labels);
end

figure(1)
subplot(row, col, roc_gaussian_plot_index);
plot(results.gaussian.roc.truepdf.pfp, results.gaussian.roc.truepdf.ptp), hold on
plot(results.gaussian.roc.truepdf.min_pfp, results.gaussian.roc.truepdf.min_ptp, '*r'), hold on
plot_roc(fn, results.gaussian.roc);
legend('guassian pdf model w true params', '\gamma_{opt}', ...
    'gaussian 100', 'gaussian 1k', 'gaussian 10k');
title('Roc Plot for gaussian classifiers');

% logistic
% linear training
for k=1:numel(fn)-1
    display(['fitting logistic-linear-function-based model parameters from ', fn{k}]);
    z = linear_feature_representation_for_logistic_fn(data.(fn{k}).features);
    [theta.linear.(fn{k}),cost] = fit_logistic_fn(data.(fn{k}), z);

    display(['evaluating descriminant scores ', fn{k}]);
    z = linear_feature_representation_for_logistic_fn(data.valid_20k.features);
    descriminantScores.logistic.linear.(fn{k}) = likelyhood_ratio_logistic(theta.linear.(fn{k}), z);
    results.logistic.linear.roc.(fn{k}) =
ROCcurve(descriminantScores.logistic.linear.(fn{k}),data.valid_20k.labels);
end


subplot(row, col, scatter_plot_index_logistic_linear);
gmm_scatter_plot(data.valid_20k);
contour_plot_logistic(data.valid_20k, @linear_feature_representation_for_logistic_fn, 500,
theta.linear.train_10k);
title('Scatter plot for D_{valid}^{20k} with contour plot of log-linear classifier');
```

```
subplot(row, col, roc_logistic_linear_plot_index);
plot(results.gaussian.roc.truepdf.pfp, results.gaussian.roc.truepdf.ptp, 'LineWidth',2), hold on
plot_roc(fn, results.logistic.linear.roc);
legend('guassian pdf model w true params', ...
   'logistic linear 100', 'logistic linear 1k', 'logistic linear 10k');
title('Roc Plot for log-linear classifiers');

% quadritic training
for k=1:numel(fn)-1
   display(['fitting logistic-quadratic-function-based model parameters from ', fn{k}]);
   z = quadratic_feature_representation_for_logistic_fn(data.(fn{k}).features);
   [theta.linear.(fn{k}),cost] = fit_logistic_fn(data.(fn{k}), z);

   display(['evaluating descriminant scores ', fn{k}]);
   z = quadratic_feature_representation_for_logistic_fn(data.valid_20k.features);
   descriminantScores.logistic.(fn{k}) = likelyhood_ratio_logistic(theta.linear.(fn{k}), z);
   results.logistic.quadratic.roc.(fn{k}) =
ROCcurve(descriminantScores.logistic.(fn{k}),data.valid_20k.labels);
end

subplot(row, col, scatter_plot_index_logistic_quadratic);
gmm_scatter_plot(data.valid_20k);
contour_plot_logistic(data.valid_20k, @quadratic_feature_representation_for_logistic_fn, 500,
theta.linear.train_10k);
title('Scatter plot for D_{valid}^{20k} with contour plot of log-quadratic classifier');

subplot(row, col, roc_logistic_quadratic_plot_index);
plot(results.gaussian.roc.truepdf.pfp, results.gaussian.roc.truepdf.ptp, 'LineWidth',2), hold on
plot_roc(fn, results.logistic.quadratic.roc);
lgnd = legend('guassian pdf model w true params', ...
   'logistic quadratic 100', 'logistic quadratic 1k', 'logistic quadratic 10k');
title('Roc Plot for log-quadratic classifiers');

% FUNCTIONS
function contour_plot_gaussian(grid_data, gridSize, gmmtrue, tau_true)
x1Grid = linspace(floor(min(grid_data.features(1,:))),ceil(max(grid_data.features(1,:))),gridSize);
x2Grid = linspace(floor(min(grid_data.features(2,:))),ceil(max(grid_data.features(2,:))),gridSize);

[a, b] = meshgrid(x1Grid,x2Grid);

meshfeatures = [a(:)';b(:)'];
discriminantScoreGridValues = log(pdf(gmmtrue{2}, meshfeatures')) - log(pdf(gmmtrue{1},
meshfeatures')) - log(tau_true);
```

```
minDSGV = min(discriminantScoreGridValues);
maxDSGV = max(discriminantScoreGridValues);
discriminantScoreGrid = reshape(discriminantScoreGridValues,gridSize,gridSize);

contour(x1Grid,x2Grid,discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDSGV]
); % plot equilevel contours of the discriminant function
lgnd = legend('Class 0','Class 1', 'Contours of discriminant function');
lgnd.Location = 'southeast';
end

function contour_plot_logistic(grid_data, x_rep, gridSize, theta)
x1Grid = linspace(floor(min(grid_data.features(1,:))),ceil(max(grid_data.features(1,:))),gridSize);
x2Grid = linspace(floor(min(grid_data.features(2,:))),ceil(max(grid_data.features(2,:))),gridSize);

[a, b] = meshgrid(x1Grid,x2Grid);

meshfeatures = [a(:)';b(:)'];
z = x_rep(meshfeatures);
discriminantScoreGridValues = log(logistic_model(theta,z)) - log((1-logistic_model(theta,z)));

minDSGV = min(discriminantScoreGridValues);
maxDSGV = max(discriminantScoreGridValues);
discriminantScoreGrid = reshape(discriminantScoreGridValues,gridSize,gridSize);

contour(x1Grid,x2Grid,discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDSGV]
); % plot equilevel contours of the discriminant function
lgnd = legend('Class 0','Class 1', 'Contours of discriminant function');
lgnd.Location = 'southeast';
end

function lgnd = plot_roc(fieldnames, results)
   fn = fieldnames;
   for k=1:numel(fn)-1
      plot(results.(fn{k}).pfp,results.(fn{k}).ptp), hold on
   end
   xlabel('P(False+)'),ylabel('P(True+)'), title('ROC Curves'),
   lgnd = legend;
end

function z = linear_feature_representation_for_logistic_fn(features)
   N = size(features, 2);
   z=[ones(N,1)  features'];
end
```

```
function z = quadratic_feature_representation_for_logistic_fn(features)
   N = size(features, 2);
   z=[ones(N,1) features(1,:)' features(2,:)' (features(1,:).^2)' (features(1,:).*features(2,:))'
(features(2,:).^2)'];
end

function [theta,cost] = fit_logistic_fn(data, z)
   options = optimset('MaxIter',10000, 'MaxFunEvals', 10000);
   N = size(data.features, 2);
   theta_init = zeros(1, size(z,2));
   [theta,cost]=fminsearch(@(t)(costfunc(t, z, data.labels, N)),theta_init,options);
end

function gmm = estimate_pdf_params(data)
   options = statset('MaxIter',10000);
   gmm{1} = fitgmdist(data.features(:, find(data.labels==0))',2,'Replicates',30,'Options',options);
   mu = mean(data.features(:, find(data.labels==1)),2)';
   sigma = cov(data.features(:, find(data.labels==1))');
   alpha = 1;
   gmm{2} = gmdistribution(mu,sigma,alpha);
end

function descriminantScores = likelyhood_ratio(gmm, features)
   descriminantScores = log(pdf(gmm{2}, features')./pdf(gmm{1}, features'))';
end

function descriminantScores = likelyhood_ratio_logistic(theta, features)
   descriminantScores = log(logistic_model(theta,features)./(1-logistic_model(theta,features)))';
end

function gmm_scatter_plot(data)
scatter(data.features(1, data.labels==0), data.features(2, data.labels==0), 'b'), hold on
scatter(data.features(1, data.labels==1), data.features(2, data.labels==1), 'r'), hold on
lgnd = legend('Class 0','Class 1');
title('Data and their true labels'),
xlabel('x_1'), ylabel('x_2'),
end

function [data, gmtrue] = generateGMMData(parameters, N)
   data.labels = (rand(1,N)>=parameters.priors(1));
   dim = size(parameters.mu{1}, 1);
   data.features = zeros(dim,N);
   for l = 1:size(parameters.priors,2)
      gmtrue{l} = gmdistribution(parameters.mu{l},parameters.Sigma{l},parameters.alpha{l});
```

```
        ind{l} = find(data.labels==l-1);
        data.features(:,ind{l}) = random(gmtrue{l},size(ind{l},2))';
    end
end

function [results] = ROCcurve(discriminantScores,labels)
[sortedScores,~] = sort(discriminantScores,'ascend');
thresholdList = [min(sortedScores)-eps,(sortedScores(1:end-1)+sortedScores(2:end))/2,
max(sortedScores)+eps];
ptp = zeros(1,length(thresholdList));
pfp = zeros(1,length(thresholdList));
perror = zeros(1,length(thresholdList));
parfor i = 1:length(thresholdList)
    tau = thresholdList(i);
    result = ERMeval(discriminantScores, labels, tau);
    ptp(i) = result.ptp;
    pfp(i) = result.pfp;
    perror(i) = result.perror;
end
results.ptp = ptp;
results.pfp = pfp;
results.perror = perror;
[results.min_perror, min_ind] = min(perror);
results.min_threshold = thresholdList(min_ind(1));
results.min_ptp = ptp(min_ind(1));
results.min_pfp = pfp(min_ind(1));
end

function results = ERMeval(discriminantScores, labels, tau)
decisions = (discriminantScores >= tau);
results.ptp = length(find(decisions==1 & labels==1))/length(find(labels==1));
results.pfp = length(find(decisions==1 & labels==0))/length(find(labels==0));
results.perror = sum(decisions~=labels)/length(labels);
end

function cost=costfunc(theta,x,label,N)
        h = logistic_model(theta,x);
        cost=(-1/N)*(label*log(h)+(1-label)*log(1-h));
end

function h = logistic_model(theta,x)
    h=1./(1+exp(-x*theta'));
end
```

Q2 Code

```
% Generates N samples from a specified Gaussian Mixture PDF
% then uses EM algorithm to estimate the parameters along
% with BIC to select the model order, which is the number
% of Gaussian components for the model.

function [data_trueMs, data_truegmm, data_bestMs, data_bestGMMs, data_dataset] =
HW_2_fitgmdist(method, jobidx)

dim = 2;
maxExp = 10;
%generate data samples
dataset_count = 4;
% method = 0;

% Evaluate BIC for candidate model orders
tic
for datasetIdx = 1:dataset_count
    N = 10^(1+datasetIdx);

    maxM = floor(N^(1/2)); % arbitrarily selecting the maximum model using this rule
    if(maxM > 15)
        maxM = 15;
    end
    bestMs = zeros(1, maxExp);
    bestGMMs = {};
    dataset_name = num2str(N)
    trueMs = zeros(1,maxExp);
    dataset = zeros(N,dim,maxExp);
    parfor experiment = 1:maxExp
        display(experiment); %#ok<*NOPTS>
        jobidx
        trueMs(experiment) = 10;
        rng shuffle;
        [dataset(:,:,experiment), truegmm{experiment}] =
generateRandGMMData(trueMs(experiment), dim, N);
        if method(1) == 1
            [bestMs(experiment), bestGMMs{experiment}] =
runBICexperiment(dataset(:,:,experiment), N, dim, maxM);
        elseif method(1) == 0
            [bestMs(experiment), mu, Sigma, alpha] =
runkfoldexperiment_fitgmdist(dataset(:,:,experiment), 10, N, dim, maxM);
```

```
            bestGMMs{experiment} = gmdistribution(mu', Sigma, alpha);
        end
    end
    data_trueMs.(['N', dataset_name]) = trueMs;
    data_truegmm.(['N', dataset_name]) = truegmm;
    data_bestMs.(['N', dataset_name]) = bestMs;
    data_bestGMMs.(['N', dataset_name]) = bestGMMs;
    data_dataset.(['N', dataset_name]) = dataset;
end
toc
    if method(1) == 1
    save(['BIC_', num2str(jobidx)]);
    elseif method(1) == 0
    save(['kfold_', num2str(jobidx)]);
    end
end



function g = evalGaussian(x,mu,Sigma)
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
invSigma = inv(Sigma);
C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(invSigma*(x-repmat(mu,1,N))),1);
g = C*exp(E);
end

function [bestM, bestGMM] = runBICexperiment(dataset, N, dim, maxM)
    nSamples = dim*N;
    nParams = zeros(1,maxM);
    BIC = zeros(1,maxM);
    neg2logLikelihood = zeros(1,maxM);
    for M = 1:maxM
        nParams(1,M) = (M-1) + dim*M + M*(dim+nchoosek(dim,2));
        options = statset('MaxIter',10000);
        gm{M} = fitgmdist(dataset,M,'Replicates',1, 'RegularizationValue',1e-10,'Options',options);
%       [mu, Sigma, alpha] = fitgmm(dim, N, M, dataset);
%        gm{M} = gmdistribution(mu', Sigma, alpha);
        neg2logLikelihood(1,M) = -2*sum(log(pdf(gm{M},dataset)));
        BIC(1,M) = neg2logLikelihood(1,M) + nParams(1,M)*log(nSamples);
    end
    [~,bestM] = min(BIC);
    bestGMM = gm{bestM};
```

```
end


function [mu, Sigma, alpha] = fitgmm(dim, N, M, dataset)

   delta = 1e-2; % tolerance for EM stopping criterion
   regWeight = 1e-10; % regularization parameter for covariance estimates

   % Initialize the GMM to randomly selected samples
   x = dataset';
   alpha = ones(1,M)/M;
   shuffledIndices = randperm(N);
   mu = x(:,shuffledIndices(1:M)); % pick M random samples as initial mean estimates
   [~,assignedCentroidLabels] = min(pdist2(mu',x'),[],1); % assign each sample to the nearest
mean
   Sigma = zeros(dim, dim, M);
   temp = zeros(M, N);
   SigmaNew = zeros(dim, dim, M);
   for m = 1:M % use sample covariances of initial assignments as initial covariance estimates
      %alpha(1,m) = find(assignedCentroidLabels==m)/N;
      Sigma(:,:,m) = cov(x(:,assignedCentroidLabels==m)') + regWeight*eye(dim,dim);
   end

   total_delta = 1e8;
   while total_delta > delta
      for l = 1:M
         temp(l,:) = repmat(alpha(l),1,N).*evalGaussian(x,mu(:,l),Sigma(:,:,l));
      end
      temp_sum = repmat(sum(temp, 1),M,1);
      plgivenx = temp./temp_sum;
      alphaNew = mean(plgivenx,2)';
      w = plgivenx./repmat(sum(plgivenx,2),1,N);
      muNew = x*w';
      for l = 1:M
         v = x-repmat(muNew(:,l),1,N);
         u = repmat(w(l,:),dim,1).*v;
         SigmaNew(:,:,l) = u*v' + regWeight*eye(dim,dim); % adding a small regularization term
      end
      Dalpha = sum(abs(alphaNew-alpha));
      Dmu = sum(sum(abs(muNew-mu)));
      DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));
      total_delta = Dalpha+Dmu+DSigma(1,1,1);
      alpha = alphaNew; mu = muNew; Sigma = SigmaNew;
   end
```

```
end

function A = generateSPDmatrix(n)
% A = rand(n,n); % generate a random n x n matrix
% A = A*A';
% A = A + n*eye(n);
A = (rand(1,1)*200).*eye(n);
end

function [data, gmdist] = generateRandGMMData(Order, dim, N)
    mu = [linspace(-200,200,Order); zeros(1,Order)];
    sigma = zeros(dim,dim,Order);
    for m = 1:Order
        sigma(:,:,m) = (rand(1,1)*100).*eye(dim);
    end
    gmdist = gmdistribution(mu',sigma,ones(1,Order)/Order);
    data = random(gmdist,N);
end


function [bestM, mu, Sigma, alpha] = runkfoldexperiment_fitgmdist(dataset, K, N, dim, maxM)
partitions_idx_start = ceil(linspace(0,N,K+1));
indPartitionLimits = zeros(K, 2);
for k = 1:K
    indPartitionLimits(k,:) = [partitions_idx_start(k)+1,partitions_idx_start(k+1)];
end
loglikelyood_M = zeros(1,maxM);
options = statset('MaxIter',10000);
for M = 1:maxM
    loglikelyhood = zeros(1,K);
    for k= 1:K
        indValidate = indPartitionLimits(k,1):indPartitionLimits(k,2);
        x = dataset';
        xValidate = x(:, indValidate); % Using folk k as validation set
        if k == 1
            indTrain = indPartitionLimits(k+1,1):N;
        elseif k == K
            indTrain = 1:indPartitionLimits(k-1,2);
        else
            indTrain = [1:indPartitionLimits(k-1,2),indPartitionLimits(k+1,1):N];
        end
        xTrain = x(:, indTrain); % using all other folds as training set
        Ntrain = length(indTrain);
        gm = fitgmdist(xTrain',M,'Replicates',10, 'RegularizationValue',1e-1,'Options',options);
```

```
%        [mu, Sigma, alpha] = fitgmm(dim, Ntrain, M, xTrain');
      loglikelyhood(k) = sum(log(pdf(gm, xValidate')));
   end
   loglikelyood_M(M) = mean(loglikelyhood);
end
[~, bestM] = max(loglikelyood_M);
finalgm = fitgmdist(dataset,bestM,'Replicates',10, 'RegularizationValue',1e-1,'Options',options);
mu = finalgm.mu';
Sigma = finalgm.Sigma;
alpha = finalgm.ComponentProportion;

% [finalgm.mu, finalgm.Sigma, finalgm.alpha] = fitgmm(dim, N, bestM, dataset);
end

function [mu, Sigma, alpha] = fitgmm(dim, N, M, dataset)

   delta = 1e-2; % tolerance for EM stopping criterion
   regWeight = 1e-10; % regularization parameter for covariance estimates

   % Initialize the GMM to randomly selected samples
   x = dataset';
   alpha = ones(1,M)/M;
   shuffledIndices = randperm(N);
   mu = x(:,shuffledIndices(1:M)); % pick M random samples as initial mean estimates
   [~,assignedCentroidLabels] = min(pdist2(mu',x'),[],1); % assign each sample to the nearest
mean
   Sigma = zeros(dim, dim, M);
   temp = zeros(M, N);
   SigmaNew = zeros(dim, dim, M);
   for m = 1:M % use sample covariances of initial assignments as initial covariance estimates
      %alpha(1,m) = find(assignedCentroidLabels==m)/N;
      Sigma(:,:,m) = cov(x(:,assignedCentroidLabels==m)') + regWeight*eye(dim,dim);
   end

   total_delta = 1e8;
   while total_delta > delta
      for l = 1:M
         temp(l,:) = repmat(alpha(l),1,N).*evalGaussian(x,mu(:,l),Sigma(:,:,l));
      end
      temp_sum = repmat(sum(temp, 1),M,1);
      plgivenx = temp./temp_sum;
      alphaNew = mean(plgivenx,2)';
      w = plgivenx./repmat(sum(plgivenx,2),1,N);
```

```
        muNew = x*w';
        for l = 1:M
            v = x-repmat(muNew(:,l),1,N);
            u = repmat(w(l,:),dim,1).*v;
            SigmaNew(:,:,l) = u*v' + regWeight*eye(dim,dim); % adding a small regularization term
        end
        Dalpha = sum(abs(alphaNew-alpha));
        Dmu = sum(sum(abs(muNew-mu)));
        DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));
        total_delta = Dalpha+Dmu+DSigma(1,1,1);
        alpha = alphaNew; mu = muNew; Sigma = SigmaNew;
    end
end

function gmm = evalGMM(x,alpha,mu,Sigma)
gmm = zeros(1,size(x,2));
for m = 1:length(alpha) % evaluate the GMM on the grid
    gmm = gmm + alpha(m)*evalGaussian(x,mu(:,m),Sigma(:,:,m));
end
end

function g = evalGaussian(x,mu,Sigma)
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
invSigma = inv(Sigma);
C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(invSigma*(x-repmat(mu,1,N))),1);
g = C*exp(E);
end
```