EECE5644: Assignment #3

Due on March 30, 2020

Professor Deniz Erdogmus

Anja Deric

GitHub Repo: https://github.com/AnjaDeric/MachineLearning

Problem 1

Generating Data
Before starting the problem, data sets were first generated to be used throughout the assignment. Four different sets were generated: 3 training sets with 100, 500 and 1000 samples and labels, and 1 test set with 10000 samples and labels. Each data set consisted of 3-class multi-ring data. A sample distribution can be seen in Figure 1 below.

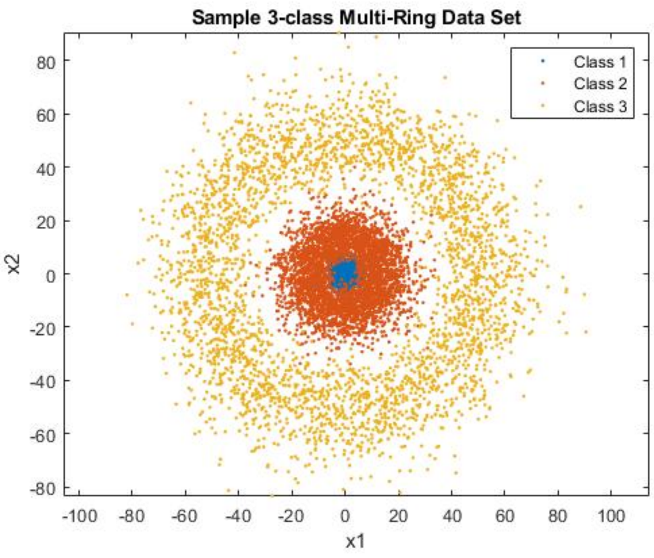


Figure 1: Sample Data Distribution for 3-class Multi-Ring Data Set

Once generated, the same data sets were used throughout the entire assignment in order to stay consistent and generate legitimate and accurate results.

Problem 1 continued on next page. . .

Neural Network Model Training
For problem 1 of the exam, data sets were used to train a 2-layer neural network to approximate class label posteriors. Each neural network trained for this problem consisted of 2 fully-connected layers of adaptive weights, followed by a soft-max layer. For each trial, the number of perceptrons in the network was varied between 1 and 6, with the activation function being varied between the sigmoid/logistic function and a soft-ReLu function. By testing each combination of perceptrons and activation functions, the most successful combination could be established in order to achive best possible results. Performance in this assignment was simply defined as the proportion of correct classifications made by the model.
The overall process used to train the networks was as follows:

1. Split training set for 10-fold cross validation using the cvpartition() function in Matlab (this function automatically makes 10 training sets, each with their own subset of testing points)
2. For each of the 10 smaller training sets, train the neural network model for 1-6 perceptrons, and both activation functions (each smaller training set is thus used to train 12 different models)
3. Test and record the model performance using the test points of the smaller training sets
4. Compare performance across all 10 test sets to determine the best combination of perceptrons and activation functions.
5. Once the best combination is found, train the full original training set (with 100, 500, and 1000 samples) for that particular combination 10 times so as to minimize the chance of getting trapped in a local minimum.
6. Record the performance of each new model, as well as the parameters for each trial.
7. Across the 10 models, find the best performing model and use those parameters to test the original 10,000 sample training set on the final model.
8. Record the final performance on the 10,000 sample set for later comparison.

The procedure described above was repeated for all 3 training sets (100, 500, and 1000 samples). Matlab code for this portion of the assignment can be found in Appendix B.

Based on the results collected, the 100, 500, and 1000 sample training sets led to a performance level of 79.50%. 83.87%, and 88.67%, respectively. Figure 2 shows this performance summary in Matlab.

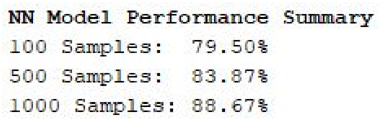

```
NN Model Performance Summary
100 Samples:  79.50%
500 Samples:  83.87%
1000 Samples: 88.67%
```

Figure 2: Neural Network Performance Summary

Additionally, for all 3 data sets, the sigmoid function outperformed the soft-ReLu activation function. In terms of perceptrons, the models with the best performance had 6, 5, and 6 perceptrons for 100, 500, and 1000 samples, respectively.

As expected, the performance of each model improved with an increasing number of samples, since more data was available for the model to use towards finding the optimal parameters. Although the overall performance of most models was good (achieved >85%), it was surprising that no model reached the 90% mark. In the next section, the EM (GMM) model will be discussed, which in my implementation achieved >95% accuracy. Thus it is possible that Matlab's fminsearch function did limit the performance of the models.

Problem 1 continued on next page. . .

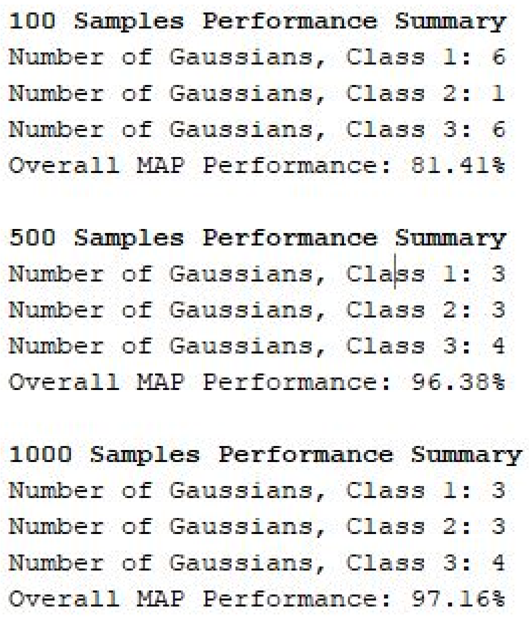
Fminsearch not only limits the number of iterations (which can be increased, but is still limited), but it is also a local minimizer- meaning that it is not difficult to get stuck in a local minimum. My implementation for this problem attempted to avoid that by re-initializing parameters every time and training even the final model 10 times. However, this did not seem to make a significand difference in performance. Another possibility is that stochastic gradient descent might've given better results for this particular assignment, but my implementation did not use that particular algorithm.

Problem 2

For the second problem on this exam, the same data sets from problem 1 were used to train Gaussian Mixture Models (GMMs) instead. The overall training and validation procedure for this problem was as follows:

1. Split each training set into 3 sets based on original class labels
2. Estimate class priors based on the number of samples in each set
3. Run the individual class data through the EM algorithm 100 times, recording which order model is picked each time
4. For the EM algortihm itself, use Matlab's built in fitgmdist() function with 5000 iterations, 10 repretitions per data set, and 6 models (mixture of 1-6 Gaussians). For each trial, use 10-fold cross-validation to split the full data set into 10 smaller sets- all of which are used to train the model. Use log-likelihood to pick model order for each trial.
5. After the best model order is picked for all 3 classes and all 100 experiments, pick the model that was most frequently selected for each class.
6. Based on the model order, fit a GMM distribution to each class and record it's negative log-likelihood. Repeat this step 10 times for each class in order to find best possible model.
7. After generating 10 models for each class, pick the model with the minimum log-likelihood as the best Gaussian Mixture Model for that particular class.
8. To test the model, use the 10,000 sample test set and decide on class labels for each data point by fitting the GMM model to the data and multiplying it with the class prior estimates from step 2.
9. Calculate performance by counting the number of correct classifications.

As with the previous problem, this procedure was repeated for all 3 training sets, and validated on the 10,000 sample test set. Figure 3 below summarizes results for problem 2.



```
100 Samples Performance Summary
Number of Gaussians, Class 1: 6
Number of Gaussians, Class 2: 1
Number of Gaussians, Class 3: 6
Overall MAP Performance: 81.41%

500 Samples Performance Summary
Number of Gaussians, Class 1: 3
Number of Gaussians, Class 2: 3
Number of Gaussians, Class 3: 4
Overall MAP Performance: 96.38%

1000 Samples Performance Summary
Number of Gaussians, Class 1: 3
Number of Gaussians, Class 2: 3
Number of Gaussians, Class 3: 4
Overall MAP Performance: 97.16%
```

Figure 3: Gaussian Mixture Models Performance Summary

Problem 2 continued on next page. . .

As can be observed with these results, the more samples were used for training, the better the performance of the model was. This is the same conclusion derived from the previous problem. The results here show that the with the lower number of samples, the algorithm tended to choose 6-component GMMs for 2 of the classes, most likely due to individual points being far apart. The more samples were added, the lower this number got. For 1000 samples, the GMMs selected were 3 and 4-component ones. Comparing this to the results fron the previous problem, GMMs in this case have far better performance, with 500- and 1000-sample GMMs achieving $> 95\%$ accuracy. As explained in the previous problem, this result can most likely be attributed to the neural network algorithm not performing to its full capacity, which would make the two models more competitive.

Appendix A: Main File

```
% Question 3 | Take Home Exam #3
% Anja Deric | March 30, 2020
clear all; close all; clc;

%% Generating Data for Questions 1 & 2
[train100_data,train100_labels] =  generateMultiringDataset(3,100);
[train500_data,train500_labels] =  generateMultiringDataset(3,500);
[train1k_data,train1k_labels] =  generateMultiringDataset(3,1000);
[test_data,test_labels] = generateMultiringDataset(3,10000);

%% Question 1 - Neural Network Training
% Train and validate neaural netrwork for each data set
% For each set, report performance for 10 trials and the average

% Train and validate 100 sample data set
NNperf100 = q1_train_and_val(train100_data,train100_labels,...
    test_data,test_labels)

% Train and validate 500 sample data set
NNperf500 = q1_train_and_val(train500_data,train500_labels,...
    test_data,test_labels)

% Train and validate 1000 sample data set
NNperf1k = q1_train_and_val(train1k_data,train1k_labels,...
    test_data,test_labels)

%% Question 2 - GMM + MAP Classifier
% Train and valiade each data set using the GMM + EM algorithm
% For each set, report on performance and model order selection

% Train and validate 100 sample data set
[GMMperf100,GMMorder100] = q2_train_and_val(train100_data,...
    train100_labels,100,test_data,test_labels);
GMM_perf_summary(100,GMMperf100,GMMorder100);

% Train and validate 500 sample data set
[GMMperf500,GMMorder500] = q2_train_and_val(train500_data,...
    train500_labels,500,test_data,test_labels);
GMM_perf_summary(500,GMMperf500,GMMorder500);

% Train and validate 1000 sample data set
[GMMperf1k,GMMorder1k] = q2_train_and_val(train1k_data,...
    train1k_labels,1000,test_data,test_labels);
GMM_perf_summary(1000,GMMperf1k,GMMorder1k);

%% Functions
function GMM_perf_summary(samples,perf,best)
```

```
    fprintf('<strong>%i Samples EM/GMM Summary</strong>\n',samples);
    fprintf('Number of Gaussians, Class 1: %i\n',best(1));
    fprintf('Number of Gaussians, Class 2: %i\n',best(2));
    fprintf('Number of Gaussians, Class 3: %i\n',best(3));
    fprintf('Overall MAP Performance: %.2f%%\n\n',perf*100);
end

function [data,labels] = generateMultiringDataset(numberOfClasses,
    numberOfSamples)

    C = numberOfClasses;
    N = numberOfSamples;
    % Generates N samples from C ring-shaped
    % class-conditional pdfs with equal priors

    % Randomly determine class labels for each sample
    thr = linspace(0,1,C+1); % split [0,1] into C equal length intervals
    u = rand(1,N); % generate N samples uniformly random in [0,1]
    labels = zeros(1,N);
    for l = 1:C
        ind_l = find(thr(l)<u & u<=thr(l+1));
        labels(ind_l) = repmat(l,1,length(ind_l));
    end

    a = [1:C].^3; b = repmat(2,1,C); % parameters of the Gamma pdf needed
        later
    % Generate data from appropriate rings
    % radius is drawn from Gamma(a,b), angle is uniform in [0,2pi]
    angle = 2*pi*rand(1,N);
    radius = zeros(1,N); % reserve space
    for l = 1:C
        ind_l = find(labels==l);
        radius(ind_l) = gamrnd(a(l),b(l),1,length(ind_l));
    end

    data = [radius.*cos(angle);radius.*sin(angle)];
end
```

Appendix B: Question 1 Code

```
function best_perf = q1_train_and_val(all_train_data,all_train_labels,...
    all_test_data,all_test_labels)

    % Create K-fold cross valiadation sets
    kfold_split = cvpartition(length(all_train_data),'KFold',10);
    max_perceptrons = 6;
    total_act_funcs = 2;

    % Arrays to store performance for each activ. function + num perceptrons
    accuracy_relu = zeros(kfold_split.NumTestSets,max_perceptrons);
    accuracy_sig = zeros(kfold_split.NumTestSets,max_perceptrons);

    % Test out each perceptron and activation function combination for all 10
    % test sets created for 10-fold cross validation
    for act_func = 1:total_act_funcs
        for perceptrons = 1:max_perceptrons
            for test_set = 1: kfold_split.NumTestSets

                % Get train and test data and labels for each test set
                train_index = kfold_split.training(test_set);
                test_index = kfold_split.test(test_set);
                train_data = all_train_data(:,find(train_index));
                test_data = all_train_data(:,find(test_index));
                train_labels = all_train_labels(find(train_index));
                test_labels = all_train_labels(find(test_index));

                % Train the NN and get performance measures for each combo
                if act_func == 1
                    accuracy_sig(test_set,perceptrons) = ...
                        mleMLPwAWGN(train_data,test_data,perceptrons,...
                        train_labels,test_labels,act_func);
                elseif act_func == 2
                    accuracy_relu(test_set,perceptrons) = ...
                        mleMLPwAWGN(train_data,test_data,perceptrons,...
                        train_labels,test_labels,act_func);
                end
            end
        end
    end

    % Find num of perceptrons and activation function combination that gives
    % best accuracy
    [~, perceptron_sig] = max(mean(accuracy_sig,1));
    [~, perceptron_relu] = max(mean(accuracy_relu,1));
    best_perceptron = max([perceptron_sig perceptron_relu]);
    best_activation = find(best_perceptron == [perceptron_sig perceptron_relu
        ],1);
    % Train and test NN on full data set with ideal perceptrons and act. func.
    test_runs = 10; final_perf = zeros(1,test_runs);
    for run = 1:test_runs
        [final_perf(run), final_params(run)] = mleMLPwAWGN(all_train_data,
            all_train_data,...
             best_perceptron,all_train_labels,all_train_labels,best_activation)
                ;
    end

    % Train the 10,000 sample training set with the parameters from the
    % best performing model
    best_perf = find(final_perf == max(final_perf));
    best_params = final_params(best_perf);
    result_labels = mlpModel(all_test_data,best_params,best_activation);

    % Calculate and report the performance of the model
    result_label = vec2ind(result_labels);
    incorrect = sum(result_label ~= all_test_labels);
    best_perf = (length(all_test_labels) - incorrect)/length(all_test_labels);
end

function [accuracy, best_params] = mleMLPwAWGN(train_data, test_data,...
    perceptrons, train_label, test_label, act_func)
    % Maximum likelihood training of a 2-layer MLP assuming AWGN

    % Determine/specify sizes of parameter matrices/vectors
    nX = 2;      % input dimensions
    nY = 3; % number of classes
    sizeParams = [nX;perceptrons;nY];

    % True NN parametes
    X = train_data;
    paramsTrue.A = 0.3*rand(perceptrons,nX);
    paramsTrue.b = 0.3*rand(perceptrons,1);
    paramsTrue.C = 0.3*rand(nY,perceptrons);
    paramsTrue.d = 0.3*rand(nY,1);
    Y = full(ind2vec(train_label));
    vecParamsTrue = [paramsTrue.A(:);paramsTrue.b;paramsTrue.C(:);paramsTrue.d
        ];

    % Initialize model parameters
    params.A = 0.3*rand(perceptrons,nX);
    params.b = 0.3*rand(perceptrons,1);
    params.C = 0.3*rand(nY,perceptrons);
    params.d = mean(Y,2);
    vecParamsInit = [params.A(:);params.b;params.C(:);params.d];

    % Optimize model using fminsearch
    vecParams = fminsearch(@(vecParams)(objectiveFunction(X,Y,...
        sizeParams,vecParams,act_func)),vecParamsInit);

    % Visualize model output for training data
    params.A = reshape(vecParams(1:nX*perceptrons),perceptrons,nX);
    params.b = vecParams(nX*perceptrons+1:(nX+1)*perceptrons);
    params.C = reshape(vecParams((nX+1)*perceptrons+1:(nX+1+nY)*perceptrons),
       nY,perceptrons);
    params.d = vecParams((nX+1+nY)*perceptrons+1:(nX+1+nY)*perceptrons+nY);
    best_params = params;
    H = mlpModel(test_data,params,act_func);

    % Calculate accuracy based on number of misclassifications
    result_label = vec2ind(H);
    incorrect = sum(result_label ~= test_label);
    accuracy = (length(test_label) - incorrect)/length(test_label);
end

function objFncValue = objectiveFunction(X,Y,sizeParams,vecParams,act_func)
    % Function to be optimized by neaural net
    N = size(X,2);
    nX = sizeParams(1);
    nPerceptrons = sizeParams(2);
    nY = sizeParams(3);
    params.A = reshape(vecParams(1:nX*nPerceptrons),nPerceptrons,nX);
    params.b = vecParams(nX*nPerceptrons+1:(nX+1)*nPerceptrons);
    params.C = reshape(vecParams((nX+1)*nPerceptrons+1:(nX+1+nY)*nPerceptrons)
        ,nY,nPerceptrons);
    params.d = vecParams((nX+1+nY)*nPerceptrons+1:(nX+1+nY)*nPerceptrons+nY);
    H = mlpModel(X,params,act_func); % neural net model
    objFncValue = sum(-sum(Y.*log(H),1),2)/N;
end

function H = mlpModel(X,params,act_func)
    % Neutal Network Model
    N = size(X,2);                              % number of samples
    nY = size(params.d);                      % number of outputs
    U = params.A*X + repmat(params.b,1,N);  % u = Ax + b, x \in R^nX, b,u \in
       R^nPerceptrons, A \in R^{nP-by-nX}
    Z = activationFunction(U,act_func);      % z \in R^nP, using nP instead of
       nPerceptons
    V = params.C*Z + repmat(params.d,1,N);  % v = Cz + d, d,v \in R^nY, C \in
       R^{nY-by-nP}
    H = exp(V)./repmat(sum(exp(V),1),nY,1); % softmax nonlinearity for second/
       last layer
end

function out = activationFunction(in, act_func)
    % Possible Activation Functions
    if act_func == 1
        out = 1./(1+exp(-in)); % sigmoid
    elseif act_func == 2
        out = in./sqrt(1+in.^2); % ISRU
    end
end
```

Appendix C: Question 2 Code

```
function [performance,best_GMM_order] = q2_train_and_val(train_data,...
    train_labels,train_size,test_data,test_labels)

    % Separate training data by class labels
    class1_data = train_data(:,find(train_labels==1));
    class2_data = train_data(:,find(train_labels==2));
    class3_data = train_data(:,find(train_labels==3));

    % Estimate class priors from labels
    est_alpha = [length(class1_data), length(class2_data), ...
        length(class3_data)]./train_size;

    % Run EM algorith for 100 experiments to determine best model order for
    % each class
    num_experiments = 100;
    num_GMM_picks = zeros(3,6);
    for class_num = 1:3
        for experiment = 1:num_experiments
            % EM algorithm with 10-fold cross validation to determine ideal
            % number of components
            if class_num == 1
                GMM_pick = cross_val(class1_data);
            elseif class_num == 2
                GMM_pick = cross_val(class2_data);
            else
                GMM_pick = cross_val(class3_data);
            end
            % Keep track of how many times each model order is selected
            num_GMM_picks(class_num,GMM_pick) = num_GMM_picks(class_num,
                GMM_pick)+1;
        end
    end

    % Pick model order for each class based on maximum num of selections
    [~,best_GMM_order] = max(num_GMM_picks',[],1);

    %Run 10 trials finding best gmm for each class label
    num_experiments = 10;
        for experiment = 1:num_experiments
        likelihood = zeros(3,num_experiments);
        max_iter = statset('MaxIter',5000);
        for class_num = 1:3
            if class_num == 1
                GMModel = fitgmdist(class1_data',best_GMM_order(1),...
                    'regularizationValue',1e-10,'Options', max_iter);
            elseif class_num == 2
                GMModel = fitgmdist(class2_data',best_GMM_order(2),...
                    'regularizationValue',1e-10,'Options', max_iter);
            else
                GMModel = fitgmdist(class3_data',best_GMM_order(3),...
                    'regularizationValue',1e-10,'Options', max_iter);
            end
            all_GMModels{class_num,experiment} = GMModel;
            likelihood(class_num,experiment) = GMModel.NegativeLogLikelihood;

        end
    end

    [~,best_model] = min(likelihood',[],1);
    best_GMModels =[all_GMModels(1,best_model(1)),...
        all_GMModels(2,best_model(2)),all_GMModels(3,best_model(3))];

    % Use test data to get probabilty of each class label for each sample
    label1_prob = pdf(best_GMModels{1},test_data')*est_alpha(1);
    label2_prob = pdf(best_GMModels{2},test_data')*est_alpha(2);
    label3_prob = pdf(best_GMModels{3},test_data')*est_alpha(3);

    % Pick class label that is most likely out of 3
    label_prob = [label1_prob label2_prob label3_prob];
    [~, label_guess] = max(label_prob,[],2);

    % Calculate performance based on number of correct guesses
    performance = sum(test_labels' == label_guess)/size(test_labels',1);
end

function best_GMM = cross_val(data)
    % Performs EM algorithm to estimate parameters and evaluete performance
    % on each data set B times, with 1 through M GMM models considered

    B = 10; M = 6;          % repetitions per data set; max GMM considered
    perf_array= zeros(B,M); % save space for performance evaluation

    kfold_split = cvpartition(length(data),'KFold',10);
    for b = 1:B
            % Pull out test and training data based on 10-fold K cross-val
        train_index = kfold_split.training(b);
        train_data = data(:,find(train_index));

        for m = 1:M
           % Run EM algorithm to estimate parameters
           max_iter = statset('MaxIter',5000);
           GMModel = fitgmdist(train_data',m,'regularizationValue',1e-10, ...
                'Options',max_iter);
           % Record likelihood of model fit for this combination
           perf_array(b,m) = GMModel.BIC;
        end
    end

    % Calculate average performance for each M and find best fit model
    avg_parf = sum(perf_array,1)./B;
    best_GMM = find(avg_parf == min(avg_parf));
end
```