ECE 8527 Homework Number 4: Markov Processes, HMMS and Estimation

Andrew Powell

May 9, 2014

1. Create N random sequences of length 100 for each of these models:

$$\omega_1 : \pi_1 = \{0.33, 0.33, 0.34\} \; A_1 = \begin{bmatrix} 0.500 & 0.250 & 0.250 \\ 0.125 & 0.750 & 0.125 \\ 0.250 & 0.250 & 0.500 \end{bmatrix} \quad B_1 = \begin{bmatrix} 0.750 & 0.125 & 0.125 \\ 0.500 & 0.250 & 0.250 \\ 0.250 & 0.250 & 0.500 \end{bmatrix}$$
$$\omega_2 : \pi_2 = \{0.25, 0.50, 0.25\} \; A_2 = \begin{bmatrix} 0.900 & 0.050 & 0.050 \\ 0.050 & 0.900 & 0.050 \\ 0.050 & 0.050 & 0.900 \end{bmatrix} \quad B_2 = \begin{bmatrix} 0.500 & 0.250 & 0.250 \\ 0.125 & 0.750 & 0.125 \\ 0.333 & 0.333 & 0.334 \end{bmatrix} \tag{1}$$

By convention, assume the output symbols L, M, and H correspond to the discrete symbols. Treat each of these two sets as your training sets. Re-seed your random number generator (if applicable) and generate M random sequences of length 100 for your test data—again generating M sequences for each class.

(a) Plot the likelihood of the training of the training data given the models as a function of the number of Baum-Welch training iterations (using only the training sets). Comment on convergence of this plot. Select a reasonable value for the remaining tasks.

For each of the two specified classes, ω_1 and ω_2, how the data's likelihood given the model $P(D|\theta)$ changes with the number of iterations i needed to generate the model θ is shown in Figure 1.

To help with the explanation of the results, the following explains the important notation. D refers to the data. The data D, of course, contains a sequence of the output symbols emitted from the hidden states. θ refers to the model used to generate the data and is also associated with one of the two classes ω. The number of BW iterations i, initial state vector π, the transitional matrix A, the observation matrix B, and as well as other unmentioned parameters are all a part of the model θ. The subscripts proceeding any of the aforementioned symbols refer to the particular class associated with the symbol. For instance, D_1 refers to data associated with class ω_1, and θ_2 refers to model of the class ω_2.

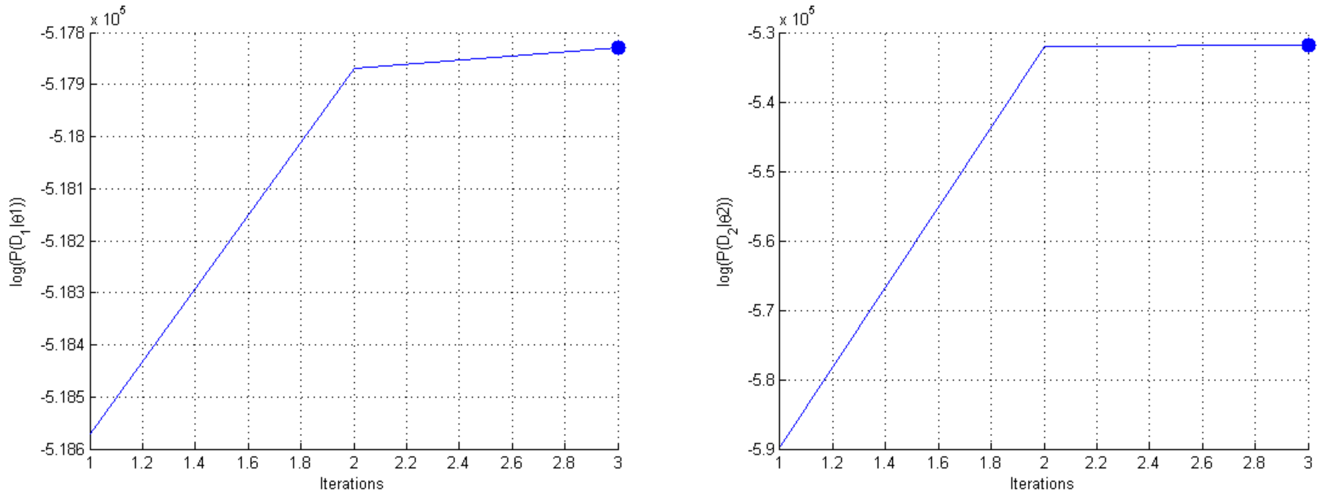
As specified for this problem, the transitional matrices, A_1 and A_2 are utilized to create the training and test data. For every iteration i, A_{train} and B_{train} is generated from the BW algorithm. $\log(P(D|\theta))$ is also generated for each iteration of the BW algorithm.

It is very important to mention the initial guesses—i.e. $A_{initial}$, $B_{initial}$, and $\pi_{initial}$—are "randomized". Namely, $A_{initial}$ and $B_{initial}$ are generated as stochastic matrices whose elements are randomly selected, whereas $\pi_{initial}$ is generated as a normalized vector whose elements are initially chosen at random, prior to the normalization. From much experimentation, it is discovered setting all the elements of the initial guesses to .333 causes the two different implementations of the BW algorithm to fail and only return the initial guesses as roughly the trained results θ_{train}—i.e. A_{train}, B_{train}, and π_{train}.

The implementation of the BW algorithm utilized for Homework 4's simulation is from Kevin Murphy's HMM MATLAB toolbox. Several other functions related to hidden Markov models (HMMs) are also called from Murphy's toolbox. It is also worth mentioning the solutions for Homework 4 were once carried out with MATLAB's implementation of the BW algorithm and other HMM-related tools from its Statistics toolbox. The reason for switching to Murphy's toolbox was because it was thought the implementation from MATLAB's Statistics toolbox was erroneous. However, it was soon discovered the issue was with the initial guesses, not the Statistics toolbox. The reason for sticking with Murphy's toolbox is the toolbox is much easier to use and there are closer sources for getting assistance (i.e. Amir).

As shown in Figure 1, the particular simulation developed for generating the likelihoods $\log(P(D|\theta_2))$ for the number of iterations i of the BW algorithm only goes up to 3 iterations. 3 iterations is also the value of i chosen for the rest of Homework 4. The reason? 3 iterations is actually all the Murphy's implementation of the BW algorithm needs to converge for N sequences of training data, each of which is 100 symbols in length and where $N = 5 \times 10^3$. Indeed, the function that executes the BW algorithm always stop at 3 iterations when the tolerance indicating convergence is reached.

Figure 1: $\log(P(D_1|\theta_1))$ versus i (on left) and $\log(P(D_2|\theta_2))$ versus i (on right)
$N = 5 \times 10^3$



(b) Set $M = 100$, and plot the probability of error for classifying the test data as a function of N (the amount of training data). Do this using an ML approach—for each test vector, compute the likelihood it could have produced by the model, and choose the model which has the greater likelihood. Justify your results.

Figure 2 shows how the probability of error (or the error rate) $P(e)$ as a function of the N number of training data sequences used to train the model θ. The $P(e)$ as a functions of N is computed for the test data sequences produced by the models θ_1

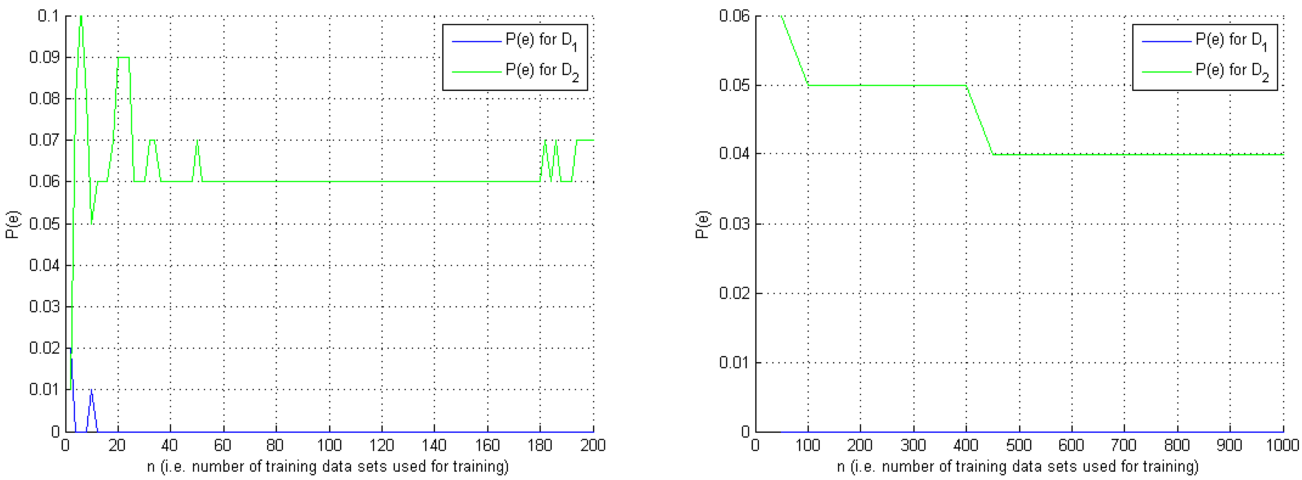
and θ_2.

The $P(e)$ as a function is obtained as follows. The beginning set of steps are each computed for each N. The models θ_{train} are first trained from N sets of training data sequences D previously generated from their respective models θ. The probability of the model for each test data sequence $P(\theta|D)$ is calculated for both of the trained models θ_{train}. $P(\theta|D)$ can be viewed as the probability of a class ω if the particular test data sequence D is given—which is the unscaled posterior probability $P(\omega|D)$, where D is viewed as a feature vector. The classes ω to which each test data sequence D may potentially belong thus are chosen based on having the largest posterior $P(\omega|D)$ for the particular test data sequence D.

Once the number of errors for each value of N and each test data sequence D is known, the error rate $P(e)$ is finally determined by dividing each error count (i.e. the number of errors) by M, the total number test data sequences generated from each of the two models θ.

The results shown in Figure 2 appear correct; both error rates $P(e)$ appear to go to 0 when the N number of training data sequences goes to infinity.

Figure 2: $P(e)$ versus N, where $P(e)$ is the probability of error and N is the number of training data sequences (please note, for the right plot, the N number of training data sets is actually increasing in steps of 50)



2. Choose a reasonable value of N and M, and repeat 1(b) using HMMs with a different number of states. Plot the probability of error as function of the number of states over the range $[1, 10]$. Can you infer the number of "underling states" in the model from this plot? Explain.

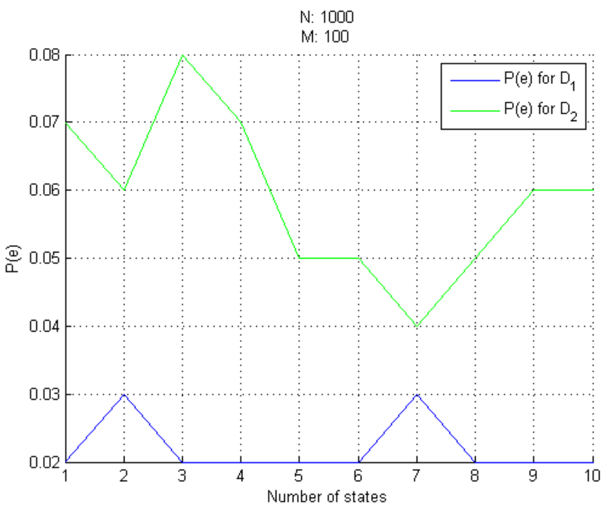
   Figures 3 and 4 contain plots of the error rate $P(e)$ as a function of the trained model's number of hidden states, which is expressed by the number of rows and columns the trained transitional matrix A_{train} has. The number of rows the observation matrix B_{train} has also changes to the number of hidden states; however, since the number of unique states observed from the training data sequences does not change, the number of columns the observation matrix B_{train} has remains the same.

The error rate $P(e)$ determined for this problem's simulation is determined with a similar approach as explained in Problem 1c. The only difference is the number of hidden states of the trained θ_{train} is varied for the range [1, 10], instead of the number training data sequences.

Each plot in the Figures 3 and 4 is the result of running the simulation for 3 separate trials. Similar to the results seen in the solution to Problem 1b, the $P(e)$ calculated over the test data sequences originally generated from the second model θ_2 is always larger than the $P(e)$ calculated over the test data sequences originally generated from the first model θ_1. The Another observation is the general shape of the $P(e)$ calculated over the second model's test data sequences; the $P(e)$ appears to have a more concave shape. Due to random nature of the simulation, however, the seemingly concave nature of the $P(e)$ calculated over the second model's test data sequences could easily be due to chance, rather than an actual trend.

As for inferring the number of hidden states from the original models θ, the trend appears to be the $P(e)$ peaks when the trained model's number of hidden states is equal to original model's number of hidden states. Another simulation (not shown) revealed this observations could have been another coincidence. However, is being able to determine the precise number of hidden states absolutely necessary? If the error rate $P(e)$ is optimized up to the point before the trained θ_{train} starts to over-generalize, then however many hidden states of the trained model should be sufficient.

Figure 3: $P(e)$ versus the trained model's number of hidden states



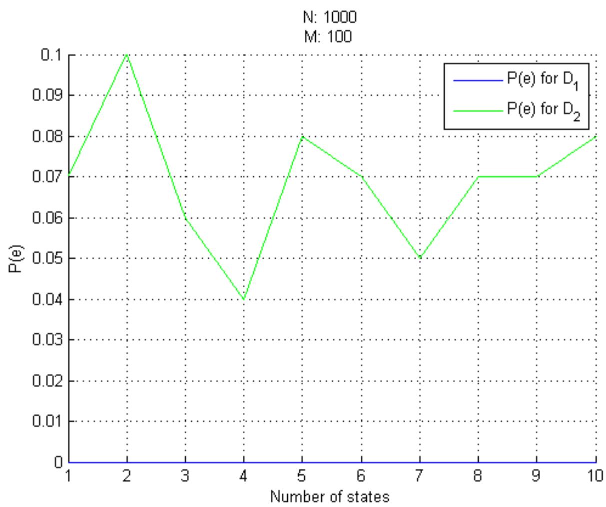
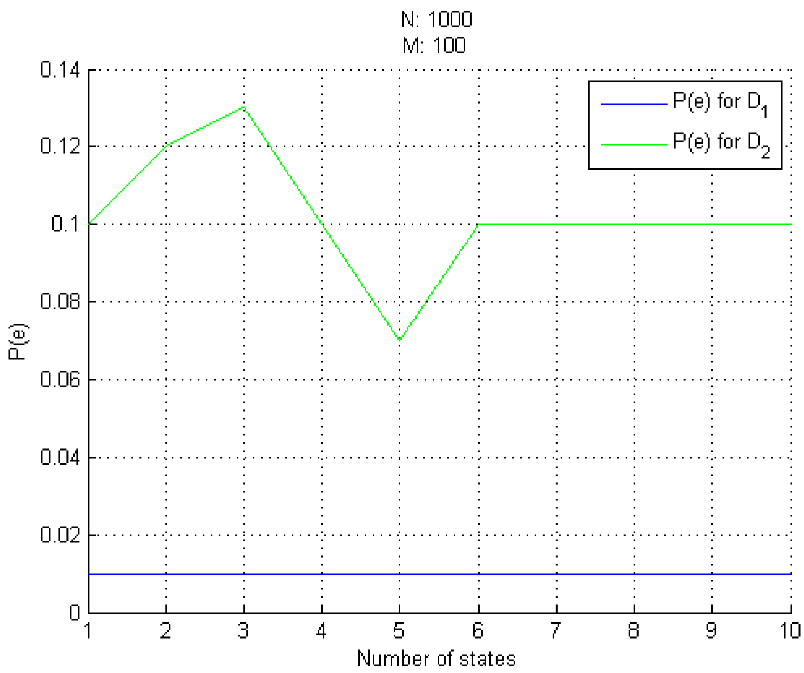


Figure 4: $P(e)$ versus the trained model's number of hidden states

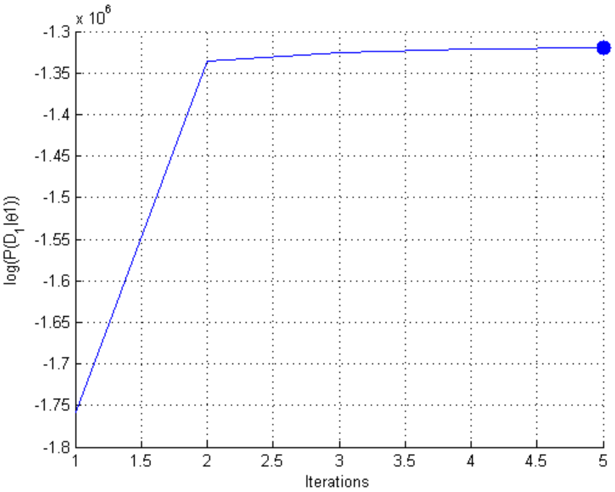


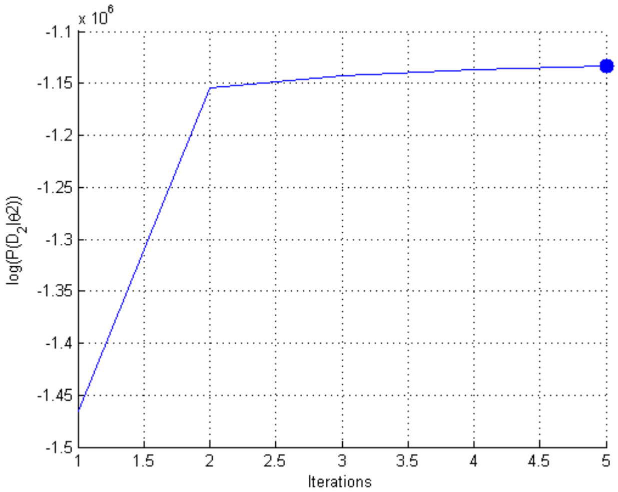
3. Repeat problem 1, but replace the discrete emission distributions with multivariate Gaussian distributions. Assume a mean vector of dimension 2, two Gaussian distributions per state, and use the mean and covariance parameters. Also experiment with the number of Gaussian mixtures. Plot the probability of error as a function of the number of mixtures components allocated to each state (using the same number of mixtures per state).

   The solutions to problem 3 are presented with the following format. The solutions listed under 3a and 3b correspond to the questions asked in problems 1a and 1b, except the models θ instead have multivariate Gaussian mixtures emitted from their hidden states. The Gaussian mixtures' parameters—mean vector μ , covariance matrix Σ, coefficient matrix c—emitted are not presented within the text of this document, but can be found with the rest of the MATLAB source code shown in this document's appendix. The solution to problem 3c contains the results to the simulation developed to determine the error rate $P(e)$ as a function of the number of Gaussian distributions emitted from each of the trained model's hidden states.

   (a) Figure 5 contains two plots of the error rate $P(e)$ as the number of iterations i for training with the BW algorithm changes. As aforementioned, each plot corresponds to one of the two classes ω and their respective models θ. As shown in the plots, the likelihood of the training data D when the model θ that produced the data D is given converges quickly after 2 iterations. The number of iterations i chosen for the rest of problem 3 is 3, seeing as likelihood $\log(P(D|\theta))$ does not change much after 3 iterations and more iterations causes the training to last longer.

Figure 5: $\log(P(D_1|\theta_1))$ versus i (on left) and $\log(P(D_2|\theta_2))$ versus i (on right) $N = 5 \times 10^3$



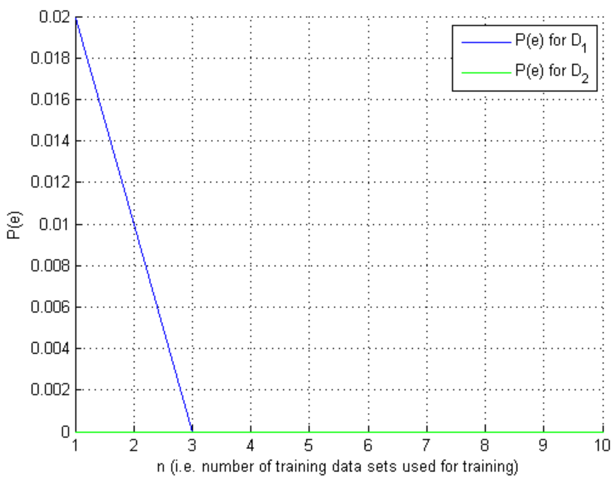


(b) Figure 6 and Figure 7 together contain a number of plots, each of which graphically display the $P(e)$ as the number of training data sequences is increased for each time a trained model θ_{train} is generated with the BW algorithm. Each plot represents the results of running the simulation once.

Interestingly enough, the error rate $P(e)$ determined from the test data D_1 produced from class 1's model θ_1 is seemingly greater than the test data D_2 produced from class 2's model θ_2. The assumption is the transitional matrix A_{train} trained from class 1's training data D_1 is a closer approximation to class 1's transitional matrix A than the transitional matrix A_{train} trained from class 2's training data D_2 is to class 2's transitional matrix A.

Overall, the results make perfect sense; the more training data inputted into the training process, the better the resultant classifier, as demonstrated by the error rate $P(e)$ dropping to zero the number of training data sequences is increased.

Figure 6: $P(e)$ versus N, where $P(e)$ is the probability of error and N is the number of training data sequences



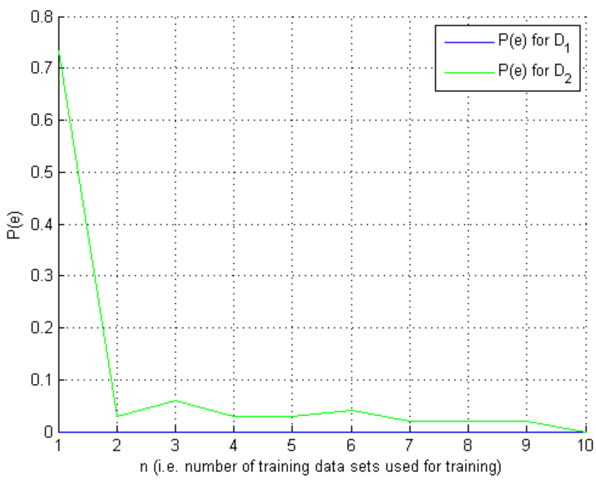
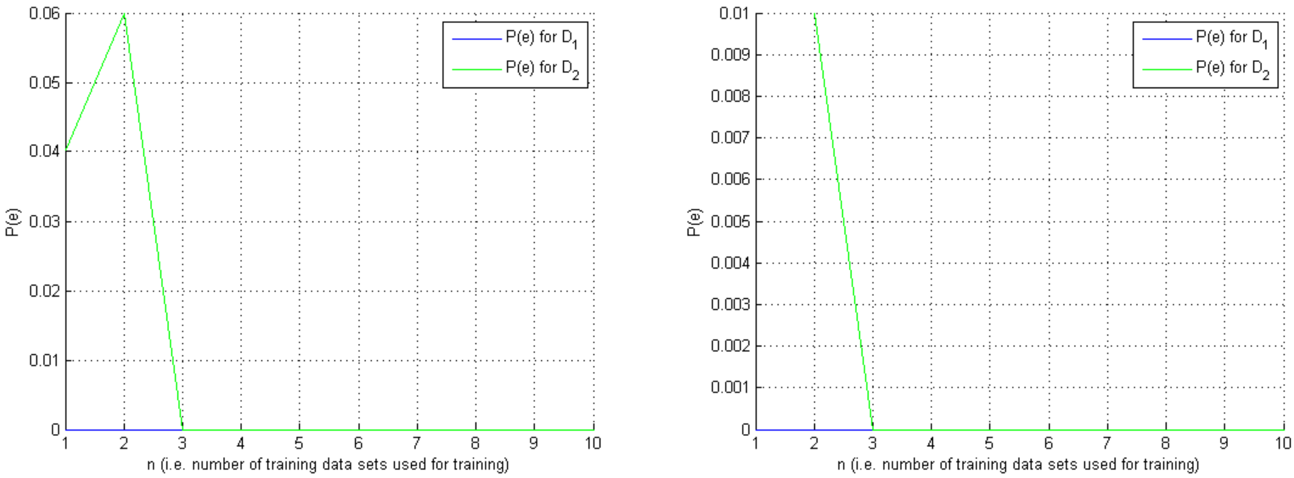


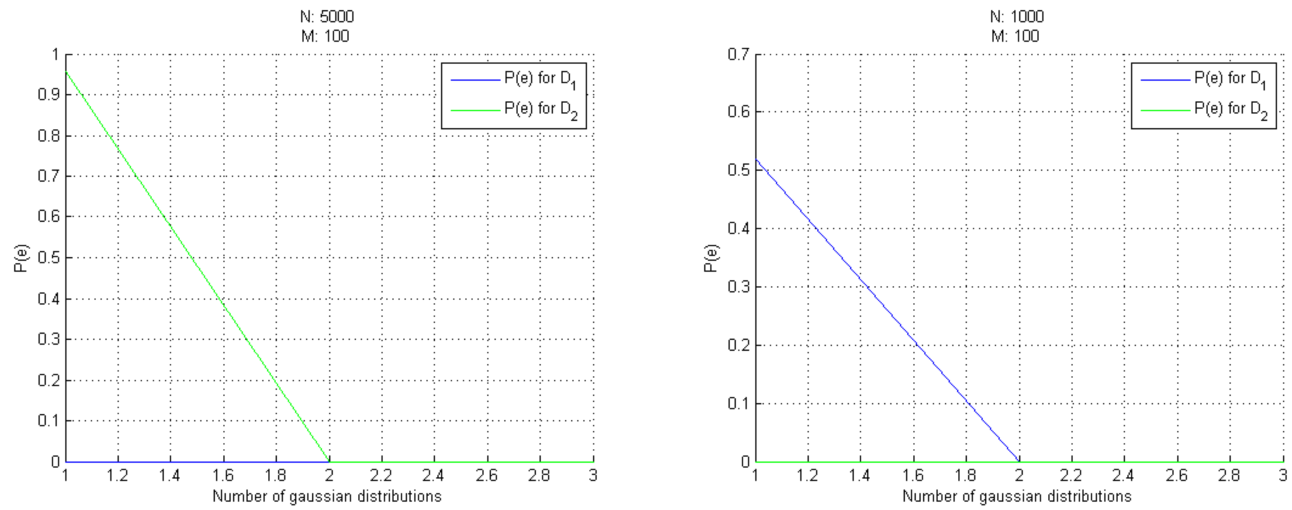
Figure 7: $P(e)$ versus N, where $P(e)$ is the probability of error and N is the number of training data sequences



(c) Similar to how the results are determine in the solutions for problems 3a and 3b, the error rate $P(e)$ is determined by Bayesian decision theory on each test data sequence taken from a particular set of test data sequences. The two sets of interested of course are the set generated from the first model and the set generated from the second model. However, in the context of this solution, the number of randomly generated Gaussian distributions per hidden state for each of the trained models is the independent variable, rather than the number of iterations or number of hidden states.

The number of randomly generated Gaussian distributions per hidden state is by simply initializing the BW algorithm with a new initial model θ_{train}. Please refer to source code that implements to the simulation for more information on how the observations are initialized.

Figure 8: $P(e)$ versus the number of randomly generated 2-dimensional Gaussian distributions emitted from each of the trained model's hidden states



4. Plot the computation time required to train the models of Problem 3 as a function of the number of training sequences N and the number of iterations of training. Similarly, plot

the computation time as a function of the number of test sequences M. Explain whether these plots match your theoretical predictions for computational complexity.

Before the results shown in Figures 9 and 10 to this problem are discussed, the assumptions made about the problem are first explained. The computational time necessary to train the models is interpreted as the amount time it takes for the BW algorithm to run (obviously). However, considering the test data sequences have nothing to do with the training, it is assumed the "computation time as a function of the number of test sequences M" refers to time taken to determine each posterior probability $P(\omega|D)$ needed for the classification of each test data sequence D. What's more, it is assumed the the time complexity of the function called to determine the posterior is the time complexity of the Forward/Viterbi algorithm, which is $O(N^2T)$, where N is the number of hidden states and T is the length of each data sequence. The theoretical time complexity of the BW algorithm is also $O(N^2T)$.

Unfortunately, due to the limited amount of time to complete Problem 4, the only work done for this problem is the generation of the numerical data and then building the plots to display the data. Figure 9 shows how the computation time changes with respect to the number of training sequences and the number of iterations, with the BW algorithm. In all cases, it is easy to notice the linear increase when the number of training sequences increase and the number of iterations increase. Because the parameters of the time complexity do not get changed, the number of hidden states N and the length of each set of sequences T, a linear increase makes perfect sense for the amount of time the algorithm should take to complete. In essence, the time complexity is simply being scaled by the number of training sequences and the number of iterations.

The same idea is applicable for Figure 10, in which the time for calculating the posterior probability is shown as a function of the number of test data sequences. Again, the time complexity's parameters are not changed, so only a linear increase is possible.

It is also worth noting MATLAB explains setting the affinity to a single processor is recommended for determining the most accurate timing results, since running simulations normally implies the performance is optimized to run faster. 9 10

Figure 9: Training time as a function of the number of training sequences and number of iterations

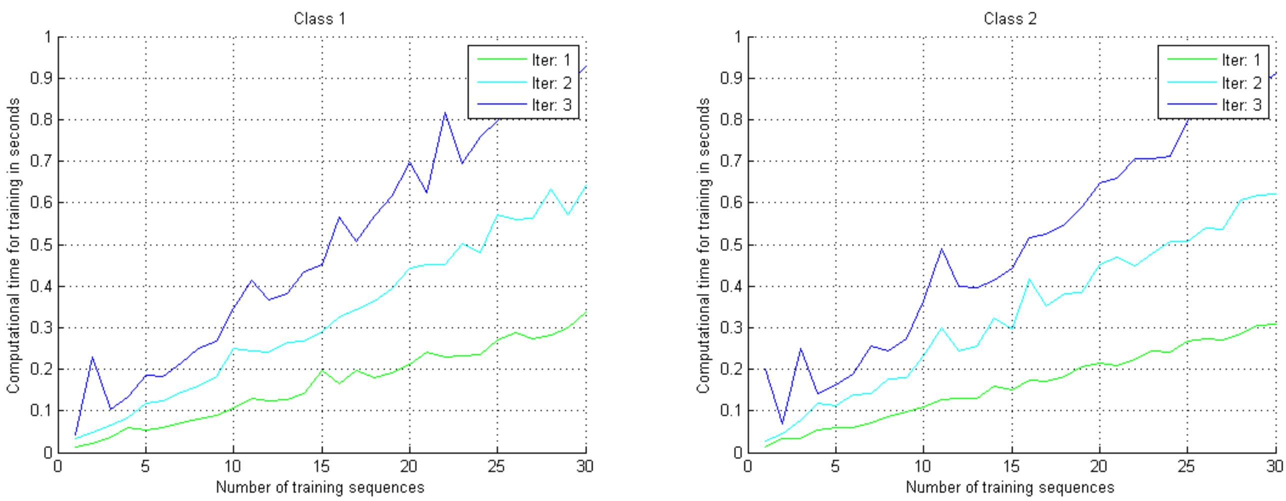
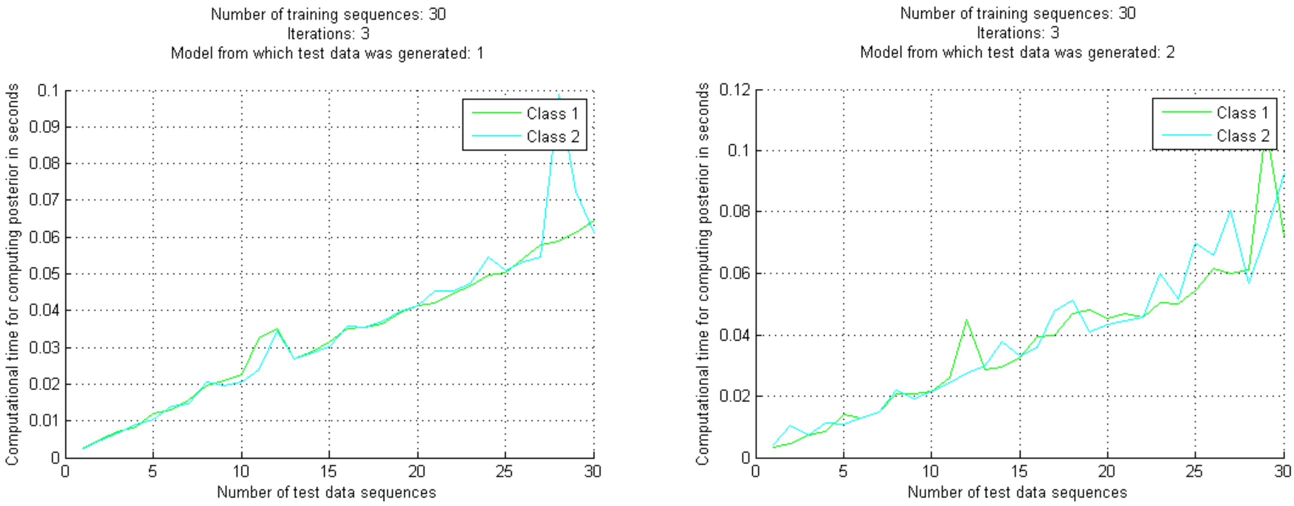


Figure 10: Time for determining the posterior probability $P(\omega|D)$ versus number of test data sequences



Appendix

```
function Homework4Script

close all;

% references
%
% Kevin Murphy's HMM MATLAB toolbox:
% http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html

% These are the parameters configured for the MATLAB script. It has been
% observed randomly selecting the initial transition, observation, and
% priors produces the best results with the BW (i.e. EM) algorithm
N = 1e3;                            % number of random sequence for training data
M = 100;                            % number of random sequences for test data
lTr = 100;                          % length of training data
lTe = 100;                          % length of test data
symbols = {'L','M','H'};            % symbols (aren't really used)
eA = mk_stochastic(rand(3,3));      % initial guess for A
eB = mk_stochastic(rand(3,3));      % initial guess for B
eBg = generateGaussianParameters(3, 2, 2); % initial guess for B (gaussian)
ei = normalise(rand(3,1));          % initial guess for initial
global globalData;                  % global data is declared as a struct

% Functions that are used in the script
n2s = @(value)num2str(value);
m2s = @(mat)mat2str(mat);
getMu = @(B)B{1};
getSigma = @(B)B{2};
getMixmat = @(B)B{3};
function [B] = setGaussianParameters(mu, sigma, mixmat)
    B = {mu, sigma, mixmat};
end
function [B, local] =  generateGaussianParameters(...
        nHiddenStates, nMixtures, nFeatures)
    local = struct;
    local.T = 50;
    local.nex = 50;
```

```
    local.data = randn(nFeatures,local.T,local.nex);
    [local.mu, local.sigma] = mixgauss_init(nHiddenStates*nMixtures, ...
        reshape(local.data, [nFeatures local.T*local.nex]), 'full');
    local.mu = reshape(local.mu, [nFeatures nHiddenStates nMixtures]);
    local.sigma = reshape(local.sigma, [nFeatures nFeatures nHiddenStates nMixtures]);
    local.coefficient = mk_stochastic(rand(nHiddenStates,nMixtures));
    B = setGaussianParameters(local.mu, local.sigma, local.coefficient);
end
function [mm, local, i] = createMM(initial, A, B, varargin)
    mm =struct( ...
        'initial', initial, 'A', A, 'B', B, ...
        'initialtrain', [], 'Atrain', [], 'Btrain', [], ...
        'trainingData', cell(1), 'trainingDataStates', cell(1), ...
        'testData', cell(1), 'testDataStates', cell(1), ...
        'GaussianOutput', false, ...
        'pDGM', [], ...
        'states', cell(1), ...
        'iterations', 3);
    local = struct;
    for i = 1:2:numel(varargin);
        local.arg = varargin{i};
        local.value = varargin{i+1};
        if strcmpi('Mu', local.arg)
            mm.GaussianOutput = true;
            local.mu = local.value;
        elseif strcmpi('Sigma', local.arg)
            mm.GaussianOutput = true;
            local.sigma = local.value;
        elseif strcmpi('Coefficient', local.arg)
            mm.GaussianOutput = true;
            local.mixmat = local.value;
        else error('Unrecognizable Input');
        end
    end
    if mm.GaussianOutput
        mm.B = setGaussianParameters( ...
            local.mu, local.sigma, local.mixmat);
    end
end
function [mm, arg] = createMMdata(mm, varargin)
    for arg = varargin
        if strcmpi('TrainingData', arg)
            [mm.trainingData, mm.trainingDataStates] = ...
                createMMdataNest(mm, lTr, N);
        elseif  strcmpi('TestData', arg)
            [mm.testData, mm.testDataStates] = createMMdataNest(mm, lTe, M);
        else
            error('Unrecognizable Input');
        end
    end
    function [data, states] = createMMdataNest(mm, l, count)
        if mm.GaussianOutput
            [data, states] = mhmm_sample(l, count, mm.initial, mm.A, ...
                getMu(mm.B), getSigma(mm.B), getMixmat(mm.B));
        else
            [data, states] = dhmm_sample(mm.initial, mm.A, mm.B, count, l);
        end
    end
end
function [mm, time, local, i] = trainMMdata(mm, varargin)
    local = struct;
    local.A = eA;
    if mm.GaussianOutput, local.B = eBg;
```

```
    else local.B = eB; end
    local.initial = ei;
    local.data = mm.trainingData;
    local.iter = mm.iterations;
    local.recordTime = false;
    for i = 1:2:numel(varargin)
        local.arg = varargin{i};
        local.value = varargin{i+1};
        if strcmpi('A', local.arg), local.A = local.value;
        elseif strcmpi('Initial', local.arg), local.initial = local.value;
        elseif strcmpi('B', local.arg), local.B = local.value;
        elseif strcmpi('Data', local.arg), local.data = local.value;
        elseif strcmpi('Iter', local.arg), local.iter = local.value;
        elseif strcmpi('Time', local.arg), local.recordTime = true;
        else error('Unrecognizable Input');
        end
    end
    if local.recordTime, tic; end
    if mm.GaussianOutput
        [mm.pDGM, mm.initialtrain, mm.Atrain, ...
            local.mu, local.sigma, local.mixmat] = mhmm_em( ...
            local.data, local.initial, local.A, ...
            getMu(local.B), getSigma(local.B), getMixmat(local.B), ...
            'max_iter', local.iter);
        mm.Btrain = setGaussianParameters( ...
            local.mu, local.sigma, local.mixmat);
    else
        [mm.pDGM, mm.initialtrain, mm.Atrain, mm.Btrain] = dhmm_em( ...
            local.data, local.initial, local.A, local.B, ...
            'max_iter', local.iter);
    end
    if local.recordTime, time = toc; end
end
function [pMGD, time, local, i] = getPMGD(mm, varargin)
    local = struct;
    local.data = mm.testData;
    local.initial = mm.initialtrain;
    local.A = mm.Atrain;
    local.B = mm.Btrain;
    local.recordTime = false;
    for i = 1:2:numel(varargin)
        local.arg = varargin{i};
        local.value = varargin{i+1};
        if strcmpi('Data', local.arg), local.data = local.value;
        elseif strcmpi('Initial', local.arg), local.initial = local.value;
        elseif strcmpi('A', local.arg), local.A = local.value;
        elseif strcmpi('B', local.arg), local.B = local.value;
        elseif strcmpi('Time', local.arg), local.recordTime = true;
        else error('Unrecognizable Input');
        end
    end
    if local.recordTime, tic; end
    if mm.GaussianOutput
        pMGD = mhmm_logprob(local.data, local.initial, local.A, ...
            getMu(local.B), getSigma(local.B), getMixmat(local.B));
    else
        pMGD = dhmm_logprob(local.data, local.initial, local.A, local.B);
    end
    if local.recordTime, time = toc; end
end

% These are the declarations for the models
while true
```

```
% discrete stuff
initial1 = [0.33, 0.33, 0.34];
A1 = [.500 .250 .250
      .125 .750 .125
      .250 .250 .500];
B1 = [.750 .125 .125
      .500 .250 .250
      .250 .250 .500];

initial2 = [0.25, 0.50, .25];
A2 = [.900 .050 .050
      .050 .900 .050
      .050 .050 .900];
B2 = [.500 .250 .250
      .125 .750 .125
      .333 .333 .334];

mms = [createMM(initial1, A1, B1)
       createMM(initial2, A2, B2)];
mmsSize = numel(mms);

% gaussian stuff
mu1 = zeros(2, 3, 2);
sigma1 = zeros(2, 2, 3, 2);
coefficient1 = [0.50 0.50
                0.90 0.10
                0.75 0.25];
mu1(:, 1, 1) = [0.50 0.50];
sigma1(:, :, 1, 1) = [1.00 0.25
                      0.25 0.50];
mu1(:, 1, 2) = [0.75 0.25];
sigma1(:, :, 1, 2) = [1.00 0.50
                      0.50 0.25];
mu1(:, 2, 1) = [0.90 0.10];
sigma1(:, :, 2, 1) = [1.00 0.75
                      0.75 1.00];
mu1(:, 2, 2) = [0.10 0.90];
sigma1(:, :, 2, 2) = [1.00 0.25
                      0.25 1.00];
mu1(:, 3, 1) = [0.70 0.30];
sigma1(:, :, 3, 1) = [1.00 0.01
                      0.01 1.00];
mu1(:, 3, 2) = [0.30 0.70];
sigma1(:, :, 3, 2) = [0.50 0.10
                      0.10 .25];

mu2 = zeros(2, 3, 2);
sigma2 = zeros(2, 2, 3, 2);
coefficient2 = [0.90 0.10
                0.10 0.90
                0.50 0.50];
mu2(:, 1, 1) = [0.10 0.10];
sigma2(:, :, 1, 1) = [1.00 0.75
                      0.75 1.00];
mu2(:, 1, 2) = [0.25 0.25];
sigma2(:, :, 1, 2) = [1.00 0.50
                      0.50 0.25];
mu2(:, 2, 1) = [0.35 0.35];
sigma2(:, :, 2, 1) = [0.75 0.40
                      0.40 0.25];
mu2(:, 2, 2) = [0.45 0.65];
sigma2(:, :, 2, 2) = [0.25 0.01
```

```
                              0.01 0.25];
mu2(:, 3, 1) = [0.55 0.85];
sigma2(:, :, 3, 1) = [1.00 0.25
                      0.25 1.00];
mu2(:, 3, 2) = [0.65 0.95];
sigma2(:, :, 3, 2) = [0.50 0.10
                      0.10 .25];

mmgs = [createMM(initial1, A1, [], ...
        'Mu', mu1, 'Sigma', sigma1, 'Coefficient', coefficient1)
        createMM(initial2, A2, [], ...
        'Mu', mu2, 'Sigma', sigma2, 'Coefficient', coefficient2)];
mmgsSize = numel(mmgs);

break;
end

% Generate data based on discrete observations
while false

% The first step is to generate all the data. 'createMMdata' and several
% other functions are actually user-defined functions that abstract some of
% the lower-level details and the functions from Kevin Murphy's HMM MATLAB
% toolbox (i.e. the toolbox Amir recommended).
for i = 1:mmsSize

    % Create the sequences of training and test data.
    mms(i) = createMMdata(mms(i), 'TrainingData', 'TestData');
end

break;
end

% Test stuff Dr. Picone had me do in order to verify whether or not the
% trained transition and observation matrices were converging
while false

    iters = [1e2];
    Ns = [1e2, 1e3, 1e4];

    disp(char(['iters: ' m2s(iters)], ...
        ['Ns: ' m2s(Ns)]));

    for i = 2:mmsSize
        disp(['Class : ' n2s(i)]);
        A = mms(i).A
        B = mms(i).B
        for iter = iters
            for n = Ns
                mms(i) = trainMMdata(mms(i), ...
                    'Data', mms(i).trainingData(1:n,:), ...
                    'Iter', iter);
                disp(['iter: ' n2s(iter)]);
                disp(['N: ' n2s(n)]);
                Atrain = mms(i).Atrain
                Btrain = mms(i).Btrain
            end
        end
        disp(char('----', '----'));
    end

    break;
end
```

```
% Script for Problem 1a
while false

% Set up the iteration vector. For the sake of saving time, I am left this
% vector very small. Moreover, I found the BW algorithm usually converged
% within the default tolerance in 3 iterations.
iters = 1:3;

% The actions contained within the for-loop are done for each model in the
% structure array 'mms'
for i = 1:mmsSize
    mm = mms(i);

    % Determine likelihood of the data given the model for each iteration
    likelihoodVersusIter = zeros(2, numel(iters));
    for iter = iters
        mm = trainMMdata(mm, 'Iter', iter);
        likelihoodVersusIter(:,iter == iters) = [iter; mm.pDGM(end)];
    end

    % Find the maximum of the likelihood to find a reasonable iterations.
    [~, maxIndex] = max(likelihoodVersusIter(2,:));
    mm.iterations = likelihoodVersusIter(1, maxIndex);

    % Plot results
    figure
    hold on
    plot(likelihoodVersusIter(1,:), likelihoodVersusIter(2,:));
    plot(likelihoodVersusIter(1,maxIndex),...
        likelihoodVersusIter(2,maxIndex), ...
        '.','MarkerSize',30);
    xlabel('Iterations');
    ylabel(['log(P(D_' num2str(i) '|\theta' num2str(i) '))']);
    grid on
    hold off

    mms(i) = mm;
end

break;
end

% Script for Problem 1b
while false

% Parameters and data
Ns = 1:100;
Ms = 1:M;
errors = zeros(mmsSize, numel(Ns));

disp(['Now onto determining probability of error as a function of the ' ...
    'number of training data sequences used for training.']);

for i = 1:mmsSize

    % The number of incorrectly assigned classes are determined for every
    % value of the variable 'n'. 'n' causes the number of sequences for
    % training to increase.
    for n = Ns

        disp([n2s(n) ' sets of training data are being used for training']);
        pMGDs = zeros(mmsSize, numel(Ms));
```

```

        % Calculated the models and then determine the number of errors.
        for k = 1:mmsSize

            % Train the new models based on 'n' amount of data
            disp(['Class ' n2s(k) ' is being trained.']);
            mms(k) = trainMMdata(mms(k),'Data', mms(i).trainingData(1:n,:));
            disp(['Class ' n2s(k) ' is done being trained.']);

            % Determine the posteriors for each of M test data sequences
            for m=Ms
                pMGDs(k, m==Ms) = getPMGD(mms(k), ...
                    'Data', mms(k).testData(m==Ms,:));
            end
        end

        % Use the Maximum A Posteriori approach (i.e. Maximum Likelihood
        % Classfication) in order to determine the number of errors.
        [~, MAPClassSelections] = max(pMGDs);
        errors(i, n==Ns) = sum(MAPClassSelections ~= i);
        disp(['There are ' n2s(errors(i, n==Ns)) ...
            ' error(s) with class ' n2s(i)]);
    end
end

% Determine the error rates for the sets of test data
errorRates = errors/numel(Ms);

% Plot the results
figure
hold on
colors = ['b', 'g'];
legendValues = cell(mmsSize, 1);
for i = 1:mmsSize
    plot(Ns, errorRates(i,:), colors(i));
    legendValues{i} = ['P(e) for D_' n2s(i)];
end
xlabel('n (i.e. number of training data sets used for training)');
ylabel('P(e)');
legend(legendValues);
grid on
hold off

break;
end

% Script for Problem 2
while false

nstates = 1:20;
Ms = 1:M;
errors = zeros(mmsSize, numel(nstates));

disp(['Now onto determining probability of error as a function of the ' ...
    'number of states used for training.']);

for i = 1:mmsSize
    for nstate = nstates

        disp([n2s(nstate) ' states used for training.']);
        pMGDs = zeros(mmsSize, numel(Ms));

        % Calculated the models and then determine the number of errors.
```

```
        for k = 1:mmsSize

            % Create a new model with initial guesses and train against
            % the  data
            disp(['Class ' n2s(k) ' is being trained.']);
            mms(k) = trainMMdata(mms(k), ...
                'Initial', normalise(rand(nstate,1)), ...
                'A', mk_stochastic(rand(nstate,nstate)), ...
                'B', mk_stochastic(rand(nstate,size(mms(k).B,2))));
            disp(['Class ' n2s(k) ' is done being trained.']);

            % Determine the posteriors for each of M test data sequences
            for m=Ms
                pMGDs(k, m==Ms) = getPMGD(mms(k), ...
                    'Data', mms(i).testData(m==Ms,:));
            end
        end

        % Use the Maximum A Posteriori approach (i.e. Maximum Likelihood
        % Classfication) in order to determine the number of errors.
        [~, MAPClassSelections] = max(pMGDs);
        errors(i, nstate==nstates) = sum(MAPClassSelections ~= i);
        disp(['There are ' n2s(errors(i, nstate==nstates)) ...
            ' error(s) with class ' n2s(i)]);
    end
end

% Determine the error rates for the sets of test data
errorRates = errors/numel(Ms);

% Plot the results
figure
hold on
colors = ['b', 'g'];
legendValues = cell(mmsSize, 1);
for i = 1:mmsSize
    plot(nstates, errorRates(i,:), colors(i));
    legendValues{i} = ['P(e) for D_' n2s(i)];
end
title(char(['N: ' n2s(N)], ['M: ' n2s(M)]));
xlabel('Number of states');
ylabel('P(e)');
legend(legendValues);
grid on
hold off

break;
end

% Generate data based on mixed gaussain observations
while true

for i = 1:mmgsSize
    mmgs(i) = createMMdata(mmgs(i), 'TrainingData', 'TestData');
end

break;
end

% Script for Problem 3a
while false

disp(['Starting script for determining the probability of error as a ' ...
```

```
    'function of the number of iterations']);

iters = 1:5;
for i = 1:mmgsSize
    disp(['On Class ' n2s(i)]);
    likelihoodVersusIter = zeros(2, numel(iters));
    for iter = iters
        disp(['Training Class ' n2s(i) ' for ' n2s(iter) ' iteration(s)']);
        mmgs(i) = trainMMdata(mmgs(i), 'Iter', iter);
        disp(['Finished training Class ' n2s(i)]);
        likelihoodVersusIter(:,iter == iters) = [iter; mmgs(i).pDGM(end)];
    end

    [~, maxIndex] = max(likelihoodVersusIter(2,:));
    mmgs(i).iterations = likelihoodVersusIter(1, maxIndex);
    disp(['Ideal number of iterations has been determine as ' ...
        n2s(mmgs(i).iterations)]);

    figure
    hold on
    plot(likelihoodVersusIter(1,:), likelihoodVersusIter(2,:));
    plot(likelihoodVersusIter(1,maxIndex),...
        likelihoodVersusIter(2,maxIndex), ...
        '.','MarkerSize',30);
    xlabel('Iterations');
    ylabel(['log(P(D_' num2str(i) '|\theta' num2str(i) '))']);
    grid on
    hold off
end

break;
end

% Script for Problem 3b
while false

% Parameters and data
Ns = 1:10;
Ms = 1:M;
errors = zeros(mmsSize, numel(Ns));

disp(['Now onto determining probability of error as a function of the ' ...
    'number of training data sequences used for training.']);

for i = 1:mmgsSize
    for n = Ns
        disp([n2s(n) ' sets of training data are being used for training']);
        pMGDs = zeros(mmgsSize, numel(Ms));
        for k = 1:mmgsSize
            disp(['Class ' n2s(k) ' is being trained.']);
            mmgs(k) = trainMMdata(mmgs(k),'Data', mmgs(k).trainingData(:,:,1:n));
            disp(['Class ' n2s(k) ' is done being trained.']);
            for m=Ms
                pMGDs(k, m==Ms) = getPMGD(mmgs(k), ...
                    'Data', mmgs(i).testData(:,:,m==Ms));
            end
        end
        [~, MAPClassSelections] = max(pMGDs);
        errors(i, n==Ns) = sum(MAPClassSelections ~= i);
        disp(['There are ' n2s(errors(i, n==Ns)) ...
            ' error(s) with class ' n2s(i)]);
    end
end
```

```
errorRates = errors/numel(Ms);

% Plot the results
figure
hold on
colors = ['b', 'g'];
legendValues = cell(mmgsSize, 1);
for i = 1:mmgsSize
    plot(Ns, errorRates(i,:), colors(i));
    legendValues{i} = ['P(e) for D_' n2s(i)];
end
xlabel('n (i.e. number of training data sets used for training)');
ylabel('P(e)');
legend(legendValues);
grid on
hold off

break;
end

% Script for Problem 3c (varying the number of gaussians in mixture)
while false


Ms = 1:M;
nstate = 3;
nMixturess = 3:6;
nFeatures = 2;

disp(['Starting to determine the error rate as a function of the ' ...
    'number of randomly generated 2D Gaussian distributions per ' ...
    'hidden state of each trained model']);

profile on

for i = 1:mmgsSize
    for nMixtures = nMixturess

        disp([n2s(nMixtures) ' gaussians per hidden state for training.']);
        pMGDs = zeros(mmgsSize, numel(Ms));

        for k = 1:mmsSize
            disp(['Class ' n2s(k) ' is being trained.']);
            mmgs(k) = trainMMdata(mmgs(k), ...
                'Initial', normalise(rand(nstate,1)), ...
                'A', mk_stochastic(rand(nstate,nstate)), ...
                'B', generateGaussianParameters(nstate,nMixtures,nFeatures));
            disp(['Class ' n2s(k) ' is done being trained.']);
            for m=Ms
                pMGDs(k, m==Ms) = getPMGD(mmgs(k), ...
                    'Data', mmgs(i).testData(:,:,m==Ms));
            end
        end
        [~, MAPClassSelections] = max(pMGDs);
        errors(i, nMixtures==nMixturess) = sum(MAPClassSelections ~= i);
        disp(['There are ' n2s(errors(i, nMixtures==nMixturess)) ...
            ' error(s) with class ' n2s(i)]);
    end
end

profile off
profile viewer
```

```
errorRates = errors/numel(Ms);

figure
hold on
colors = ['b', 'g'];
legendValues = cell(mmsSize, 1);
for i = 1:mmsSize
    plot(nMixturess, errorRates(i,:), colors(i));
    legendValues{i} = ['P(e) for D_' n2s(i)];
end
title(char(['N: ' n2s(N)], ['M: ' n2s(M)]));
xlabel('Number of gaussian distributions');
ylabel('P(e)');
legend(legendValues);
grid on
hold off

break;
end

% Script for Problem 4
while true

nstate = 3;
Ns = 1:1:30;
Ms = 1:1:30;
iters = 1:3;
timeComplexity = nstate^2*lTr;
theoreticalLineColor = 'k';
data = zeros(mmgsSize, numel(Ns), numel(Ms), numel(iters), mmgsSize, 2);

while true

disp(['Beginning to determine time as a function of number of training ' ...
    'sequences, number of test data sequences, and number of iterations']);

fprintf('i\tn\tm\titer\tk\ttT\tcT\n');
for i = 1:mmgsSize
    for n = Ns
        for m = Ms
            for iter=iters
                for k = 1:mmgsSize
                    fprintf('%d\t%d\t%d\t%d\t%d\n', i, n, m, iter, k);
                    [mmgs(k), trainTime] = trainMMdata(mmgs(k), ...
                        'Data', mmgs(k).trainingData(:,:,1:n), ...
                        'Iter', iter, ...
                        'Time', []);
                    [~, computeTime] = getPMGD(mmgs(k), ...
                        'Data', mmgs(i).testData(:,:,1:m), ...
                        'Time', []);
                    data(i, n == Ns, m == Ms, iter == iters, k, :) = ...
                        [trainTime computeTime];
                    fprintf('%d\t%d\t%d\t%d\t%d\t%g\t%g\n', i, n, m, iter, ...
                        k, trainTime, computeTime);
                end
            end
        end
    end
end

break;
end
```

```
if true, data = globalData.problem4Data;
else globalData.problem4Data = data; end

for k = 1:mmgsSize
    figure;
    hold on
    colorSet = varycolor(numel(iters));
    legendSet = cell(1, numel(iters));
    for iter = iters
        computationalTimes = reshape(data(1, :, 1, iter, k, 1), ...
            1, numel(Ns));
        unitComputationalTime = mean(computationalTimes./Ns);
        theoreticalCTx = [Ns(1) Ns(end)];
        theoreticalCTy = theoreticalCTx*unitComputationalTime;
        plot(theoreticalCTx, theoreticalCTy, theoreticalLineColor);
        plot(Ns, computationalTimes, 'Color',colorSet(iter==iters,:));
        legendSet{iter==iters} = ['Iter: ' n2s(iter)];
    end
    title(['Class ' n2s(k)]);
    xlabel('Number of training sequences');
    ylabel('Computational time for training in seconds');
    legend(legendSet);
    grid on
    hold off
end

n = find(Ns(end)==Ns);
iter = find(iters(end)==iters);
for i = 1:mmgsSize
    figure;
    hold on
    colorSet = varycolor(mmgsSize);
    legendSet = cell(1, mmgsSize);
    for k = 1:mmgsSize
        plot(Ms, reshape(data(i, n, :, iter, k, 2),1,numel(Ms)), 'Color', colorSet(k,:));
        legendSet{k} = ['Class ' n2s(k)];
    end
    title(sprintf(['Number of training sequences: %d\nIterations: '...
        '%d\nModel from which test data was generated: %d\n'], ...
        n, iter, i));
    xlabel('Number of test data sequences');
    ylabel('Computational time for computing posterior in seconds');
    legend(legendSet);
    grid on
    hold off
end

break;
end

end
```

Listing 1: MATLAB source