Machine Learning and Neural Computation
Coursework 2

Peter Sarvari - 27/11/2017

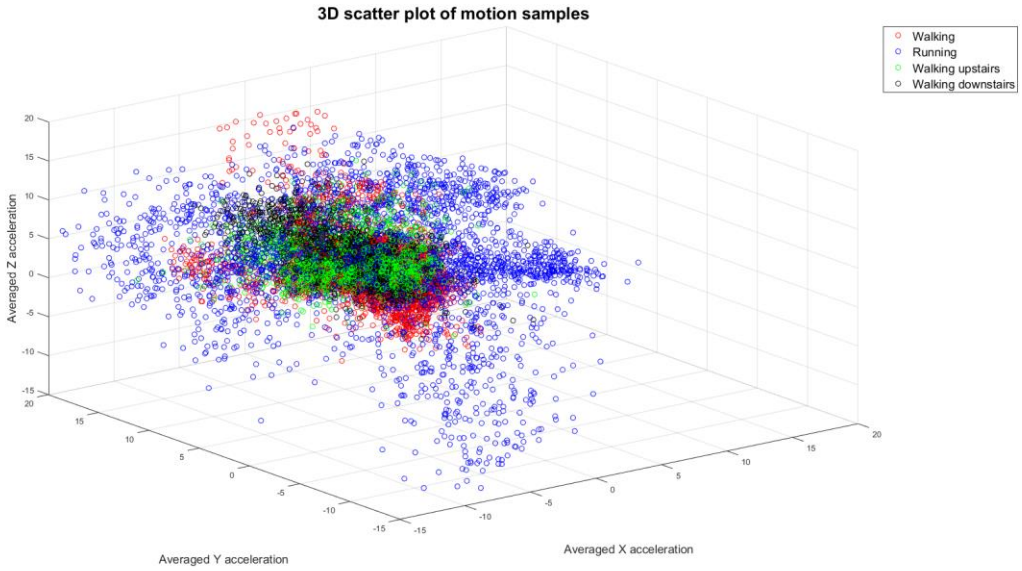
00987075

Ps5714@ic.ac.uk

MEng Bioengineering

Appendix – Introduction

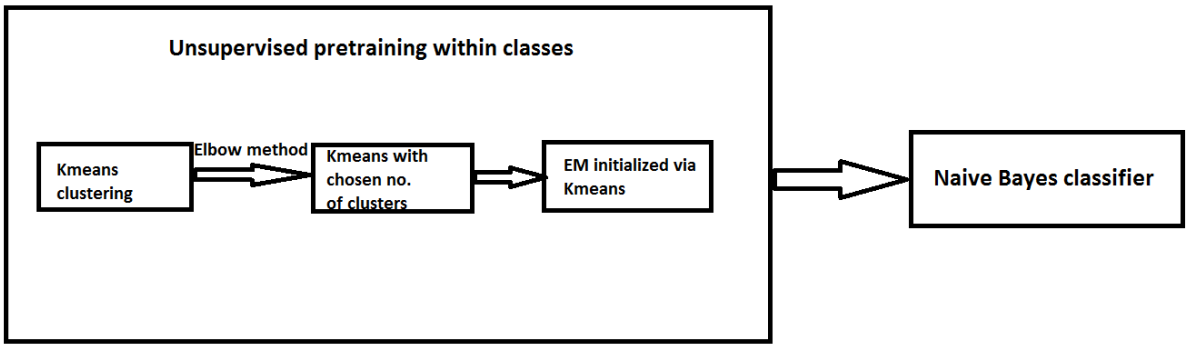
We are asked to classify samples of 4 classes (walking, running, walking upstairs and walking downstairs), which occupy the space the following way:



It seems a hard task, no natural decision boundary is really visible. In the first part, we are using a given neural network structure and tune its parameters to obtain better accuracy on the test set. Note that in the classification pipeline several methods could have been changed: for example, rather than taking the average of the acceleration time-series, the average of the integral of the time-series (velocity) could have been used, which is likely to separate all activities better for an obvious reason. Other advanced neural network settings that could have been used include glorot initialization, batch normalization, ReLu activation function (except for the last layer) and Pollack-Ribiere optimization with exact line search (fmincg) for more effective minimization, regularization (2-norm or dropout) and model ensembles for better generalization and vectorization for faster training. Even if we decided to stick to gradient descent (as we do in First Part, Question 1/a), other than manipulating the decrease and minimum value of the learning rate, we could have used momentum or Nesterov momentum and manipulate the momentum hyperparameter. Furthermore, we could have specified learning rate separately for each weight depending on how big its gradient is: a great example in RSMprop. Please also note that the accuracy we obtain on the test set is not the accuracy we should expect on an independent (unseen) set, since we use it to tune hyperparameters (although, since the test data is quite big, this bias should be small).

Second Part – Classification with generative model – Methods

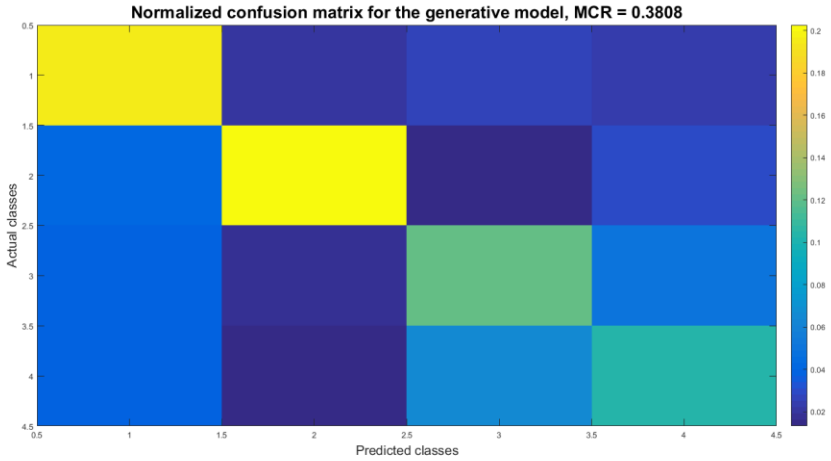
We use Expectation-Maximization within the classes to find the parameters in the Gaussian Mixture model. We then use that mixture model and a Naïve Bayes classifier to gain the probabilities of data points belonging to different classes, as suggested by Dr Aldo Faisal during the lectures. We initialize the centres of the Gaussians using the kmeans clustering algorithm, which is itself initialized multiple times so that we avoid local minima. We choose the number of clusters in the kmeans via the "elbow" method (see Bishop). The methods are summarized on the diagram below.



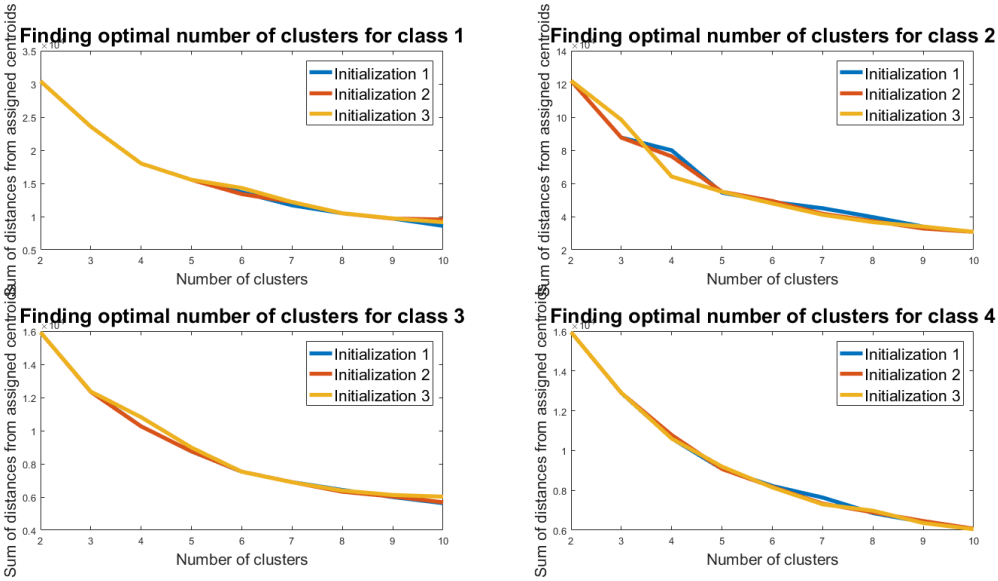
Second Part – Classification with generative model – Results

Our generative model *outperformed* the Multilayer Perceptron model on the given dataset (accuracy = 61.9% vs 60.3% for MLP). In addition, the vectorised generative model is *much faster* than the MLP model (about 26 times: 15'' vs 6'30''). Furthermore in this case we do not tune any hyperparameters on the test set, so this *prediction is actually valid* in the case of the generative model. We report the results on both all the 4 classes and the binary classification (running vs. walking upstairs).

Confusion matrix for the multiclass classification with the generative model:



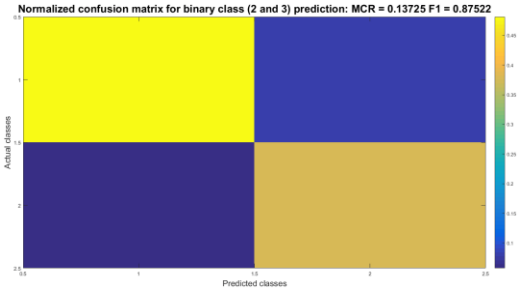
The reason we chose 4 clusters for class 1 and 4 and 3 clusters for class 2 and 3 (elbow method):



Note that obviously the elbow method is quite a bit subjective. Here, for the second and third class we picked 3 clusters, but from the binary classification diagram (Appendix), we would have picked 4 and 6 clusters for them (which actually yields even better prediction). There is no perfect solution. If we were aiming for a better result, we could do 10 initializations, take the minimum of each sum of squared distance among the 10 initializations, connect them and look for the elbow. Then cross-validate for the possible elbows to find the one that yields the highest accuracy on the cross validation set and then report accuracy on an unseen set (test set).

In case of the binary classification, the generative model is able to yield almost the same prediction as the MLP (0.27% difference in accuracy). The diagram in the Appendix illustrates why it is reasonable to pick 7 clusters in class 1 (original class 2) and 6 clusters in class 2 (original class 3).

Using the elbow method, we could have also picked 4 clusters for class 1, but with 7 clusters we got slightly better result (86.275% instead of 84.72%). The confusion matrix is the following:

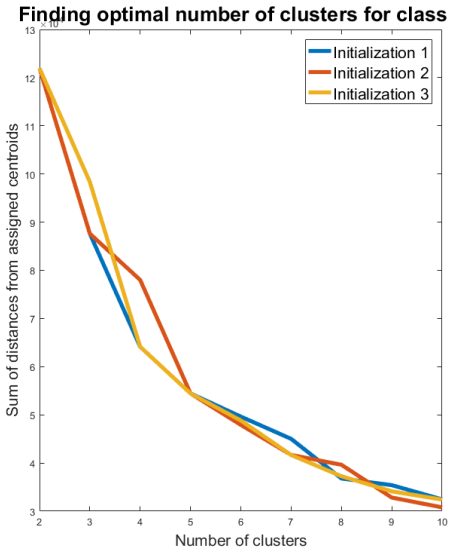


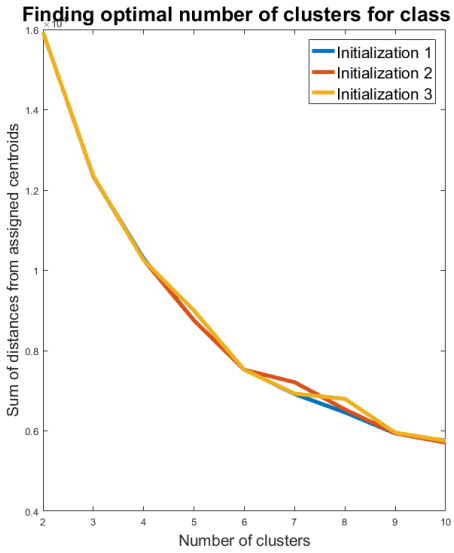
The generative model in the case has 7*3 plus 6*3 parameters associated with the centres of the Gaussians in each class. It has 7*6 plus 6*6 parameters associated with the covariance matrices of the Gaussians (Note: the covariance matrix is a 3*3 matrix in this case, but since it is symmetric, we only really control 6 elements in each matrix) and 7 plus 6 parameters associated with mixing coefficients of the Gaussians. This yields 130 parameters. Note that in the binary case, the MLP has 3*10 + 10*7 + 7*2 = 114 parameters.

Discussion and Conclusion

We have given 3 reasons (better and valid prediction, faster) why the generative model is better in the multiclass classification case. In the binary case, the results from the two models are almost the same and in the strict sense, both prediction accuracies are not valid, since hyperparameters have been tuned on the test set (although since the test set is quite large the effect of this bias should be small). The generative model is much faster in the binary case as well, however, we have to note that should the dimension of the data increase, the complexity of the generative model is expected to increase at a higher rate than that of the MLP. Other disadvantage of our generative model is that it involves a manual subjective decision (elbow method) in the training process. It is also important to note that MLP does not make any assumption on the data distribution, whereas the generative model assumes that the data is Gaussian distributed.

Appendix – Elbow method in the binary classification case





Appendix – TrainClassifierX.m

```
function parameters = TrainClassifierX (train_input_data,
train_input_labels)

%% Written by Peter Sarvari, 2017
% Imperial College, London, ID: 00987075

% This is needed because in the code I assume that class labels start from
% one
if min(train_input_labels) == 0
    train_input_labels = train_input_labels + 1;
    disp('Samples have been relabelled so that the labels start from 1!
Press enter to acknowledge!')
    pause;
end

%close all

load Activities.mat
maxiterkmeans = 100; %maximum iterations allowed in the kmeans algorithm
train_data_class = cell(1,length(unique(train_input_labels)));
%length(unique(train_input_labels)) is the number of classes

for class = 1:length(unique(train_input_labels))
    %stores train data separately for each class
    train_data_class{class} =
train_input_data(train_input_labels==class,:);
end



%% Kmeans with Elbow Method - Pipeline Step 1

% totalsumofsquares = zeros(length(unique(train_input_labels)),9);
% %stores the sum of (squared) distances of the data points
```

```
% %from their assigned cluster centres for each class and each possible
% %maximum cluster option (2-10 clusters, see below)
% for class = 1:length(unique(train_input_labels)) %do for all classes
%     for init = 1:3 %see the result for 3 different initializations
%         for clusters = 2:10 %I investigate this region only
%             %choose the initial centroids randomly from data points
%             [centroids, ~] = kMeansInitCentroids(train_data_class{class},
clusters);
%             [classes, mu, totalsumofsquares(class,clusters-1)] =
fastkmeans(train_data_class{class}, centroids, maxiterkmeans);
%             %totalsumofsquares
%             subplot(length(unique(train_input_labels))/2,2,class)
%         end
%         plot(2:10, totalsumofsquares(class,:), 'LineWidth',4);
%         hold on
%         ylabel('Sum of distances from assigned centroids', 'FontSize',
16);
%         xlabel('Number of clusters', 'FontSize', 16);
%         title(['Finding optimal number of clusters for class ',
num2str(class)], 'FontSize', 20);
%         legend({'Initialization 1', 'Initialization 2', 'Initialization
3'}, 'FontSize', 16);
%     end
% end
%
% pause;

%% Minimizing squared distance given a number of clusters - Pipeline Step 2
% From elbow method we chose 4, 3, 3, 4 for class 1, 2, 3, 4, respectively
% These were determined from Pipeline Step 1 (now commented out)

totalsumofsquares = zeros(length(unique(train_input_labels)),10);
% storing the totalsumofsquares (defined in Step 1) for each class and
% each of the 10 random initialization
mu = cell(length(unique(train_input_labels)),10);
% cluster centres for each class and for each initialization
mu_best = cell(length(unique(train_input_labels)),1);
% stores the cluster centres for each class that resulted in the smallest
% totalsumofsquares among the 10 initializations (multiple initialization
% is implemented to avoid local minima)

for class = 1:length(unique(train_input_labels))
    for init = 1:10

        %Need to rewrite if new dataset accoding to results from Part I!!

        if class == 1 || class == 4
            clusters = 4; %4 for 2-class, 7 for binary
        else
            clusters = 3; %3 for 4-class, 6 for binary
        end

        %End of need to rewrite if new dataset

        %choose the initial centroids randomly from data points
        [centroids, ~] = kMeansInitCentroids(train_data_class{class},
clusters);
        %implement kmeans using function defined below
        [~, mu{class,init}, totalsumofsquares(class,init)] =
fastkmeans(train_data_class{class}, centroids, maxiterkmeans);
```

```
        [~, ix] = min(totalsumofsquares, [], 2);
        mu_best{class} = mu{class, ix};
    end
end

%% EM - Pipeline Step 3

%Parameters of the EM algorithm: mean, covariance matrix and mixing
%coefficients
mus = cell(1,length(unique(train_input_labels)));
covariances = cell(1,length(unique(train_input_labels)));
coeffs = cell(1,length(unique(train_input_labels)));

for class = 1:length(unique(train_input_labels))

    % Iinitializing EM parameters for each class separately
    covars =
zeros(size(mus{class},1),size(mus{class},1),size(mus{class},2));
    % Centroids are initialized from Kmeans result above
    mus{class} = mu_best{class};
    for n = 1:size(mus{class},1)
        %size(mus{class},1) is the number of clusters
        covars(:,:,n) = eye(size(mus{class},2));
        %size(mus{class},2) is the number of dimensions
    end
    coeffs{class} = 1/size(mus{class},1) * ones(size(mus{class},1), 1);
    %Coefficients are initialized so that they are uniform for the
    %Gaussians and they are also normalized
    covariances{class} = covars;

    disp('initial log likelihood is:');
    maxiter = 1000; %Maximum iteration for the EM algorithm
    p = zeros(maxiter+1, 1);
    %Calculating the log likelihood (see Bishop)
    p(1) = logLikelihoodGaussianMixture(coeffs{class}, (mus{class})',
covariances{class}, train_data_class{class});
    p(1) %displaying initial log likelihood

    for iter=1:maxiter
        %Calculating the responsibilities of each Gaussian for the data
        %points (see Bishop). Note that mus matrix had to be transposed
        %because of the specification of the responsibilities function
        %E-step:
        gamma = responsibilities(coeffs{class}, (mus{class})',
covariances{class}, train_data_class{class});
        %M-step
        combined_params = MaximizeProbability(train_data_class{class},
gamma);
        %Unwrapping the parameters
        coeffs{class} = combined_params{1};
        mus{class} = (combined_params{2})';
        covariances{class} = combined_params{3};
        %Showing number of iterations
        iter
        %Calculating and displaying new log likelihood to see how algorithm
        %converges
        p(iter+1) = logLikelihoodGaussianMixture(coeffs{class},
(mus{class})', covariances{class}, train_data_class{class});
        p(iter+1)
        if p(iter+1) < (p(iter) + 1) %stopping criterion, can be changed
```

```
            %but small enough compared to initial p (10^4)
            break
        end
    end
end

%% Priors - Part of Naive Bayes classification, Pipeline Step 4

%Wrapping parameters into the cell array called "parameters"
parameters = cell(1,4);
parameters{1} = coeffs;
parameters{2} = mus;
parameters{3} = covariances;
%Calculating prior probability of classes based on the training labels
prior = zeros(1,length(unique(train_input_labels)));
for class = 1:length(unique(train_input_labels))
    prior(class) =
sum(train_input_labels==class)/length(train_input_labels);
end
parameters{4} = prior;

end

function [class_vec, mean_vec, totalsumofsquares] = fastkmeans(vec,
initial_means, maxiterkmeans)
%Kmeans algorithm implementation
%Cluster 1 is first row vector in initial_means
mean_vec = initial_means;
mean_minus = zeros(size(initial_means));
index = 0;
K = size(initial_means, 1);
while ~isequal(mean_minus, mean_vec);
    mean_minus = mean_vec;
    res_matrix = bsxfun(@plus, sum(vec.^2, 2), bsxfun(@minus,
(sum(mean_vec.^2, 2))', 2*vec*mean_vec'));
    %Vectorization: I want (SAMPLEi - MUx)^2,
    %I calculate SAMPLEi^2+MUx^2-SAMPLEi*MUx
    [sumofsquares, ix] = min(res_matrix, [], 2);
    class_vec = ix; %assign the data points to the cluster centre closest
    %to them
    totalsumofsquares = sum(sumofsquares);
    for class = 1:K
        mean_vec(class,:) = mean(vec(class_vec==class, :), 1);
    end
    index = index + 1
    if index > maxiterkmeans
        break
    end
    %Note for me:
    %NaN was caused by centroids being the same
    %because multiple same entries in original dataset
    %pause
end

end

function [centroids, randidx] = kMeansInitCentroids(X, K)
%KMEANSINITCENTROIDS This function initializes K centroids that are to be
%used in K-Means on the dataset X
%idea from Prof. Andrew Ng's Coursera course
```

```
%   centroids = KMEANSINITCENTROIDS(X, K) returns K initial centroids to be
%   used with the K-Means on the dataset X

% Initialize the centroids to be random examples
% Randomly reorder the indices of examples
randidx = randperm(size(X, 1));
% Take the first K examples as centroids
centroids = X(randidx(1:K), :);

end

function p = logLikelihoodGaussianMixture(coeffs, mus, covars, x)
%mus are the means of the K multivariate Gaussians (D*K), where D is the
%Diension and K is the number of Gaussians
%coeffs are the mixing coefficients, size is K*1
%covars are the covariance matrices of the K multivariate Gaussians (D*D*K)
%x are the samples matrix (M*D), where M is the number is samples
%see Bishop on the calculation of the log likelihood

p = 0;
tempo = 0;

for sample = 1:size(x, 1)
    for cluster_number = 1:size(mus, 2)
        %size(x(sample,:))
        %size(mus(:,cluster_number))
        %size(mus, 2)
        %size(coeffs(cluster_number))
        tempo = tempo +
coeffs(cluster_number)*multivariateGaussian(x(sample,:),
mus(:,cluster_number), covars(:,:,cluster_number));
    end
    p = p + log(tempo);
    tempo = 0;
end

end

function gamma = responsibilities(coeffs, mus, covars, x)
%coeffs are the mixing coefficients
%mus are the means of the k multivariate Gaussians (D*K), where D is the
%Diension and K is the number of Gaussians
%covars are the covariance matrices of the K multivariate Gaussians (D*D*K)
%x are the samples matrix (M*D), where M is the number is samples
%gamma has size M*K and stores the posteriors (responsibilities) for a
%sample and a particular Gaussian

%See Page 438, Bishop, Eq (9.23)


temp = zeros(size(x,1), size(mus, 2));
for cluster_number = 1:size(mus, 2)
    temp(:, cluster_number) =
coeffs(cluster_number)*multivariateGaussian(x, mus(:, cluster_number),
covars(:,:,cluster_number));
end

denominator = sum(temp, 2);
```

```
gamma = zeros(size(x,1), size(mus,2));

for sample = 1:size(x,1)
    for cluster_number = 1:size(mus,2)
        gamma(sample, cluster_number) = temp(sample,
cluster_number)/denominator(sample);
    end
end

end

function combined_params = MaximizeProbability(x, gamma)
%Algorithm outlined in Bishop, page 439

mus = zeros(size(x,2), size(gamma, 2));
coeffs = zeros(size(gamma,2), 1);
covars = zeros(size(x,2), size(x,2), size(gamma,2));
for cluster_number = 1:size(gamma,2)
    Nk = sum(gamma(:,cluster_number)); %Eq (9.27)
    %Eq (9.24)
    mus(:,cluster_number) = (1/Nk * sum(bsxfun(@times, x,
gamma(:,cluster_number)),1))';
    %Eq (9.26)
    coeffs(cluster_number) = Nk/size(x,1);
    %Eq (9.25)
    temp = zeros(size(x,2), size(x,2));
    for sample = 1:size(x,1)
        temp = temp + gamma(sample, cluster_number)*((x(sample,:))'-
mus(:,cluster_number))*(x(sample,:)-(mus(:,cluster_number))');
    end
    covars(:,:,cluster_number) = 1/Nk * temp;
end
%Wrap parameters into cell array called "combined_params"
combined_params = cell(3,1);
combined_params{1} = coeffs;
combined_params{2} = mus;
combined_params{3} = covars;
end

function p = multivariateGaussian(X, mu, Sigma2)
%    p = MULTIVARIATEGAUSSIAN(X, mu, Sigma2) Computes the probability
%    density function of the examples X under the multivariate gaussian
%    distribution with parameters mu and Sigma2. If Sigma2 is a matrix, it
is
%    treated as the covariance matrix. If Sigma2 is a vector, it is treated
%    as the \sigma^2 values of the variances in each dimension (a diagonal
%    covariance matrix)
%    X should have dimension n by k, where n is the number of samples and k
%    is the number of features (base functions)
%    Normalized distribution and vectorized implementation

k = length(mu); %Dimension of the Gaussian

if (size(Sigma2, 2) == 1) || (size(Sigma2, 1) == 1)
    Sigma2 = diag(Sigma2);
end

X = bsxfun(@minus, X, mu(:)'); %(:) makes mu a column vector so it does not
matter if input is row or column vector
p = (2 * pi) ^ (- k / 2) * det(Sigma2) ^ (-0.5) * ...
```

```
    exp(-0.5 * sum(bsxfun(@times, X * pinv(Sigma2), X), 2));

end
```

Appendix – ClassifyX.m

```
function [pred_class, norm_results] = ClassifyX(input, parameters)

%% Written by Peter Sarvari, 2017
% Imperial College, London, ID: 00987075

%% Naive Bayes Classification - Pipeline Step 4

results = zeros(size(input,1), length(parameters{1}));
%length(parameters{1}) is the number of classes

for classes = 1:length(parameters{1})
    temp = zeros(size(input,1), 1);
    %length(parameters{1}{classes}) is the number of clusters within each
    %class
    for clusters = 1:length(parameters{1}{classes})
        %Vectorized implementation of Eq (9.7) in Bishop, page 430
        temp = temp + parameters{1}{classes}(clusters) *
multivariateGaussian(input, parameters{2}{classes}(clusters,:),
parameters{3}{classes}(:,:,clusters)); %likelihood
    end
    results(:, classes) = temp * parameters{4}(classes); %posterior
end
norm_results = bsxfun(@rdivide, results, sum(results, 2)); %divide by p(x)
%or in other words, normalize the result
[~, ix] = max(results, [], 2); %The predicted class is the one with the
%highest posterior, p(class|x)
pred_class = ix;

end

function p = multivariateGaussian(X, mu, Sigma2)
%    p = MULTIVARIATEGAUSSIAN(X, mu, Sigma2) Computes the probability
%    density function of the examples X under the multivariate gaussian
%    distribution with parameters mu and Sigma2. If Sigma2 is a matrix, it
is
%    treated as the covariance matrix. If Sigma2 is a vector, it is treated
%    as the \sigma^2 values of the variances in each dimension (a diagonal
%    covariance matrix)
%    X should have dimension n by k, where n is the number of samples and k
%    is the number of features (base functions)
%    Normalized distribution and vectorized implementation

k = length(mu); %Dimension of the Gaussian

if (size(Sigma2, 2) == 1) || (size(Sigma2, 1) == 1)
    Sigma2 = diag(Sigma2);
end

X = bsxfun(@minus, X, mu(:)'); %(:) makes mu a column vector so it does not
matter if input is row or column vector
p = (2 * pi) ^ (- k / 2) * det(Sigma2) ^ (-0.5) * ...
    exp(-0.5 * sum(bsxfun(@times, X * pinv(Sigma2), X), 2));
```

```
end
```

Appendix – Main script that generated the results of the generative model

```
%% Written by Peter Sarvari, 2017
% Imperial College, London, ID: 00987075


load Activities.mat

%% Multiclass classification

params = TrainClassifierX(train_data, train_labels);
[pred, res] = ClassifyX(test_data, params);

disp('Accuracy:')
sum(pred==test_labels)/length(test_labels)

% Confusion matrix

figure(1)

res_matrix = [test_labels, pred];
confusion = zeros(4,4);
for i = 1:size(res_matrix,1)
    y = res_matrix(i,1);
    x = res_matrix(i,2);
    confusion(y,x) = confusion(y,x) + 1;
end
accurate = trace(confusion);
all = sum(sum(confusion));
MCR = (all-accurate)/(all); %Misclassification rate
confusion = confusion/length(test_labels);
imagesc(confusion); colorbar
title(['Normalized confusion matrix for the generative model, MCR = ',
num2str(MCR)], 'FontSize', 20)
xlabel('Predicted classes', 'FontSize', 16)
ylabel('Actual classes', 'FontSize', 16)

figure(2)
%Pairwise confusion matrix from the results of the multiclass
%classification to estimate which classes can be separated the easiest
for i = 1:3
    for j = i+1:4 %iterating through all possible pairwise combinations
        if i == 1
            subplot(2,3,i+j-2)
        else
            subplot(2,3,i+j-1)
        end
        choice = confusion([i j], [i j]);
        accurate = trace(choice);
        fail = choice(1,2) + choice(2,1);
        MCR = fail/(fail+accurate); %Misclassification rate
        precision = choice(1,1)/(choice(1,1)+choice(2,1));
        recall = choice(1,1)/(choice(1,1)+choice(1,2));
        F1 = 2*precision*recall / (precision+recall); %F1-score
        imagesc(choice)
```

```
        caxis([0 0.2]) %unified color axis
        colorbar;
        title([num2str(i), '&', num2str(j), ': MCR = ', num2str(MCR) ' F1 =
' num2str(F1)], 'FontSize', 20)
        xlabel('Predicted classes', 'FontSize', 16)
        ylabel('Actual classes', 'FontSize', 16)
    end
end

pause;
%% Binary classification
%Choosing class 2 and class 3 for binary classification
train_run = train_data(train_labels==2, :);
train_walkup = train_data(train_labels==3, :);
test_run = test_data(test_labels==2, :);
test_walkup = test_data(test_labels==3, :);
train_bin_labels =
[ones(size(train_run,1),1);2*ones(size(train_walkup,1),1)];
test_bin_labels = [ones(size(test_run,1),1);2*ones(size(test_walkup,1),1)];
train_bin_data = [train_run; train_walkup];
test_bin_data = [test_run; test_walkup];

param_bin = TrainClassifierX(train_bin_data, train_bin_labels);
[pred_bin, res_bin] = ClassifyX(test_bin_data, param_bin);

disp('Accuracy:')
sum(pred_bin==test_bin_labels)/length(test_bin_labels)

% Confusion matrix

figure(3)

res_matrix = [test_bin_labels, pred_bin];
confusion = zeros(2,2);
for i = 1:size(res_matrix,1)
    y = res_matrix(i,1);
    x = res_matrix(i,2);
    confusion(y,x) = confusion(y,x) + 1;
end
confusion = confusion/size(test_bin_labels,1);
imagesc(confusion); colorbar

accurate = trace(confusion);
fail = confusion(1,2) + confusion(2,1);
MCR = fail/(fail+accurate); %Misclassification rate
precision = confusion(1,1)/(confusion(1,1)+confusion(2,1));
recall = confusion(1,1)/(confusion(1,1)+confusion(1,2));
F1 = 2*precision*recall / (precision+recall); %F1-score

title(['Normalized confusion matrix for binary class (2 and 3) prediction',
': MCR = ', num2str(MCR) ' F1 = ' num2str(F1)], 'FontSize', 20)
xlabel('Predicted classes', 'FontSize', 16)
ylabel('Actual classes', 'FontSize', 16)
```

Appendix – run_MLP.m

```
clear all
close all
```

```
%% Load Data
%
% data - readings from the accelerometer. Each column corresponds to
% respectively X, Y and Z axis.
%
% labels - ID of the activity
% 1 - walking
% 2 - running
% 3 - walking upstairs
% 4 - walking downstairs
%
% For binary classification you should change the labels of your chosen
% activities so there are only values 1 and 2. For example if you chose to
% classify walking upstairs and downstair you should change the labels 3
% and 4 to respectively 1 and 2 for binary classification to work
% correctly. When in doubt ask GTA.

load Activities.mat


%% Configurations/Parameters

% Network's architecture.
% Each element of the vector is the number of neurons in each hidden layer.
% For example:
% [1] - 1 hidden layer with 1 neuron
% [2 3] - 2 hidden layers with 2 and 3 neurons respectively
% Default MLP architecture: [5]
nbrOfNeuronsInEachHiddenLayer = [10 7];

% Epoch - one forward pass and one backward pass of all the training
% examples.
% Maximum number of epochs.
% Default number of epochs: 500.
nbrOfEpochs_max = 500;

%% Question 1A - How does the learning rate influence the training process?
% This is classic implementation of the backpropagation-based learning
% algorithm. All the methods in the function MLP_1A are covered in the
% lecture.
% Hint: You may have noticted that the learning is relatviely slow. You can
% try to run the code with more epochs to see at which epoch do the
learning
% curves plateau. DON'T spend too much time on such experiments, as the
% process can be very time consuming. In the end you should report the
% results for 500 epochs.

% Learning rate
% Default learning rate: 0.0001
learningRate = 3*10^-3;

% Should learning rate decrease with each epoch?
enable_decrease_learningRate = 1; %1 for enable decreasing, 0 for disable
learningRate_decreaseValue = 2*10^-5; % decrease value
min_learningRate = 10^-3; % minimum learning rate

%MLP_1A modified to return train_accuracy
```

```
% [accuracy_1a, train_accuracy, best_prediction_1a] = MLP_1A(train_data,
train_labels, test_data, test_labels, nbrOfNeuronsInEachHiddenLayer,
learningRate, nbrOfEpochs_max, enable_decrease_learningRate,
learningRate_decreaseValue, min_learningRate);

% plot(1:nbrOfEpochs_max, accuracy_1a,'b')
% hold on
% plot(1:nbrOfEpochs_max, train_accuracy,'r')
% legend({'test accuracy', 'train accuracy'}, 'FontSize', 16, 'Location',
'southeast');
% %title(['Number of hidden layers is ',
num2str(length(nbrOfNeuronsInEachHiddenLayer)), ' and the respective
neurons in the layers are ', num2str(nbrOfNeuronsInEachHiddenLayer(1)), '
and ',  num2str(nbrOfNeuronsInEachHiddenLayer(2))]);
% title(['Learning rate is ', num2str(learningRate), ' and it is decreased
by ', num2str(learningRate_decreaseValue), ' at every epoch. The minimum
learning rate is set to ', num2str(min_learningRate)], 'FontSize', 20);
% xlabel('Epoch', 'FontSize', 16)
% ylabel('Accuracy', 'FontSize', 16);

% accuracy_1a - [nbrOfEpochs_max x 1] vector of accuracies obtained for
each
% of training epochs. So-called 'learning curve'.
%
% best_prediction_1a - [number of datapoints x 1] vector of predicted
classes
% for each datapoint for the epoch that yielded the best accuracy. This can
% be directly compared with target_labels containing the true labels.

%% Rest of the questions
% Note that this function does not take learning rate as an input, as the
% resilient gradient descent backpropagation (RPROP) is used in the
training
% process. The reason for using it is better overall perfomance and much
% faster runtimes.
%
% The original paper describing the method:
% Martin Riedmiller, 'Rprop -Description and Implementation Details',
% Technical Report, January 1994

%MLP_REST modified to return train_accuracy
[accuracy, train_accuracy, best_prediction] = MLP_REST(train_data,
train_labels, test_data, test_labels, nbrOfNeuronsInEachHiddenLayer,
nbrOfEpochs_max);
plot(1:nbrOfEpochs_max, accuracy,'b')
hold on
plot(1:nbrOfEpochs_max, train_accuracy,'r')
legend({'test accuracy', 'train accuracy'}, 'FontSize', 16, 'Location',
'southeast');
title(['Number of hidden layers is ',
num2str(length(nbrOfNeuronsInEachHiddenLayer)), ' and the respective
neurons in the layers are ', num2str(nbrOfNeuronsInEachHiddenLayer)],
'FontSize', 20);
xlabel('Epoch', 'FontSize', 16)
ylabel('Accuracy', 'FontSize', 16);

% accuracy - [nbrOfEpochs_max x 1] vector of accuracies obtained for each
% of training epochs. So-called 'learning curve'.
%
% best_prediction - [number of datapoints x 1] vector of predicted classes
```

```
% for each datapoint for the epoch that yielded the best accuracy. This can
% be directly compared with target_labels containing the true labels.

%% Confusion matrix

figure(2)

res_matrix = [test_labels, best_prediction];
confusion = zeros(4,4);
for i = 1:size(res_matrix,1)
    y = res_matrix(i,1);
    x = res_matrix(i,2);
    confusion(y,x) = confusion(y,x) + 1;
end
accurate = trace(confusion);
all = sum(sum(confusion));
MCR = (all-accurate)/(all); %Misclassification rate
confusion = confusion/length(test_labels);
imagesc(confusion); colorbar
title(['Normalized confusion matrix for MLP with 2 hidden layers containing
10 and 7 neurons, MCR = ', num2str(MCR)], 'FontSize', 20)
xlabel('Predicted classes', 'FontSize', 16)
ylabel('Actual classes', 'FontSize', 16)


figure(3)
%Pairwise confusion matrix from the results of the multiclass
%classification to estimate which classes can be separated the easiest
for i = 1:3
    for j = i+1:4 %iterating through all possible pairwise combinations
        if i == 1
            subplot(2,3,i+j-2)
        else
            subplot(2,3,i+j-1)
        end
        choice = confusion([i j], [i j]);
        accurate = trace(choice);
        fail = choice(1,2) + choice(2,1);
        MCR = fail/(fail+accurate); %Misclassification rate
        precision = choice(1,1)/(choice(1,1)+choice(2,1));
        recall = choice(1,1)/(choice(1,1)+choice(1,2));
        F1 = 2*precision*recall / (precision+recall); %F1-score
        imagesc(choice)
        caxis([0 0.2]) %unified color axis
        colorbar;
        title(['Class ', num2str(i), ' and ', num2str(j), ': MCR = ',
num2str(MCR) ' F1 = ' num2str(F1)], 'FontSize', 20)
        xlabel('Predicted classes', 'FontSize', 16)
        ylabel('Actual classes', 'FontSize', 16)
    end
end

%% Binary classification
%Choosing class 2 and class 3 for binary classification
train_run = train_data(train_labels==2, :);
train_walkup = train_data(train_labels==3, :);
test_run = test_data(test_labels==2, :);
test_walkup = test_data(test_labels==3, :);
train_bin_labels =
[ones(size(train_run,1),1);2*ones(size(train_walkup,1),1)];
```

```
test_bin_labels = [ones(size(test_run,1),1);2*ones(size(test_walkup,1),1)];
train_bin_data = [train_run; train_walkup];
test_bin_data = [test_run; test_walkup];

%MLP_REST modified to return train_accuracy
[accuracy, train_accuracy, best_prediction] = MLP_REST(train_bin_data,
train_bin_labels, test_bin_data, test_bin_labels,
nbrOfNeuronsInEachHiddenLayer, nbrOfEpochs_max);

figure(1)

plot(1:nbrOfEpochs_max, accuracy,'b')
hold on
plot(1:nbrOfEpochs_max, train_accuracy,'r')
legend({'test accuracy', 'train accuracy'}, 'FontSize', 16, 'Location',
'southeast');
title(['Number of hidden layers is ',
num2str(length(nbrOfNeuronsInEachHiddenLayer)), ' and the respective
neurons in the layers are ', num2str(nbrOfNeuronsInEachHiddenLayer)],
'FontSize', 20);
xlabel('Epoch', 'FontSize', 16)
ylabel('Accuracy', 'FontSize', 16);

figure(2)

res_matrix = [test_bin_labels, best_prediction];
confusion = zeros(2,2);
for i = 1:size(res_matrix,1)
    y = res_matrix(i,1);
    x = res_matrix(i,2);
    confusion(y,x) = confusion(y,x) + 1;
end
confusion = confusion/size(test_bin_labels,1);
imagesc(confusion); colorbar

accurate = trace(confusion);
fail = confusion(1,2) + confusion(2,1);
MCR = fail/(fail+accurate); %Misclassification rate
precision = confusion(1,1)/(confusion(1,1)+confusion(2,1));
recall = confusion(1,1)/(confusion(1,1)+confusion(1,2));
F1 = 2*precision*recall / (precision+recall); %F1-score

title(['Normalized confusion matrix for binary class prediction', ': MCR =
', num2str(MCR) ' F1 = ' num2str(F1)], 'FontSize', 20)
xlabel('Predicted classes', 'FontSize', 16)
ylabel('Actual classes', 'FontSize', 16)
```

Appendix – The script that generated the 3D scatter plot of the data samples

```
close all

%Script used to produce the graph in the introduction in the Appendix

data = [train_data;test_data];
label = [train_labels;test_labels];

data_label = data(label==1, :);
scatter3(data_label(:,1), data_label(:,2), data_label(:,3), 'r');
```

```
hold on

data_label = data(label==2, :);
scatter3(data_label(:,1), data_label(:,2), data_label(:,3), 'b');

data_label = data(label==3, :);
scatter3(data_label(:,1), data_label(:,2), data_label(:,3), 'g');

data_label = data(label==4, :);
scatter3(data_label(:,1), data_label(:,2), data_label(:,3), 'k');

legend({'Walking', 'Running', 'Walking upstairs', 'Walking downstairs'},
'FontSize', 14);
title('3D scatter plot of motion samples', 'FontSize', 20);
xlabel('Averaged X acceleration', 'FontSize', 14);
ylabel('Averaged Y acceleration', 'FontSize', 14);
zlabel('Averaged Z acceleration', 'FontSize', 14);
```