Computer Vision – Programming Project 1

王順興 0210184

Structure:

1. Init_Read.m: Bootstrap script for global constant setting and data set reading
2. readSets.m: Helper function to read the required training and testing samples

3. GaussianMLE.m: Script to test Guassian Fitting and Inference
4. gaussianMLFitting.m: Helper function for Gaussian Fitting

5. GammaMLE.m: Script to test Gamma Fitting and Inference
6. gammaMLFitting.m: Helper function for Gamma Fitting

Full project:

https://github.com/vicodin1123/CV-project/tree/master/p1/Letter_PROPER

Init_Read.m

```
% This is the bootstrap script for the entire project,
% run this before anything else.

clear; close all;
rng(0,'twister');

% Global constants
MAX_CLASS = 26;         % # of classes
MAX_TEST_SIZE = 20;     % Test set size
DATA_ROW = 16;          % Data dimension Row
DATA_COLUMN = 8;        % Data dimension Column
DATA_SIZE = DATA_ROW * DATA_COLUMN;   % Data dimension
NoiseMagnitude = 0.01;  % The var. of noise to add when reading
 samples

% Let half of the data be the training set
% The dataset is not uniform, this is the best I can do
TotalSampleCount = [4034 1284 2114 1442 4955 921 2472 861 4913 189 909
 3140 1602 5024 3897 1377 341 2673 1394 2136 2562 664 520 413 1221
 1094];
TrainCount = round(TotalSampleCount./2);
TestCount = TotalSampleCount - TrainCount;

% Limit the test set size to min(MAX_TEST_SIZE, Amount of samples left)
TestCount = min(MAX_TEST_SIZE*ones(size(TotalSampleCount)),
 TestCount);

% Storage
TrainSet = cell(1, 1);
TestSet = cell(1, 1);

% Read the dataset
[TrainSet, TestSet] = readSets(MAX_CLASS, TrainCount, TestCount,
 DATA_SIZE, NoiseMagnitude);

% Now, run "GaussianMLE.m" or "GammaMLE.m"
```

readSets.m

```
function [TrainSet, TestSet] = readSets(MAX_CLASS, TrainCount,
 TestCount, DATA_SIZE, Noise)
% READSETS Read the data for all classes
% INPT: MAX_CLASS: The # of classes to process
%       TrainCount: 1xC array. The amount of training sample required for
 each class
%       TestCount: 1xC array. The amount of testing sample required for
 each class
%       DATA_SZIE: The dimension of a sample
%       Noise: The magnitude of the variance of a gaussian noise to be
 added onto each pixel
% OUPT: TrainSet: 1xC cell. Each contains the training dataset of each class
%       TestSet: 1xC cell. Each contains the testing dataset of each class

for i = 1 : MAX_CLASS
 % Open the file containing the dataset of a class
 fileName = sprintf('./data/%c.data', i - 1 + 'a');
 fid = fopen(fileName);

    R = 0;
    TrainMatrix = zeros(TrainCount(i), DATA_SIZE);
    TestMatrix = zeros(TestCount(i), DATA_SIZE);

 % Read a sample from a file until we've reach the designated amount of
 samples
    while R < TrainCount(i) + TestCount(i)

        R = R + 1;
        % Read a sample
        tempData = readOneLine(fid, Noise);

  % Check EOL, which should never be evaluated true,
  % unless the TrainCount or TestCount is set incorrectly if
        tempData == -1
             break;
        end

        % Store the first MAX_TRAIN_SIZE data as train set if
        R <= TrainCount(i)
            TrainMatrix(R, :) = tempData;
        else
            TestMatrix(R - TrainCount(i), :) = tempData;
        end
    end

    TrainSet{i} = TrainMatrix;
    TestSet{i} = TestMatrix;
    fclose(fid);
end
end
```

readSets.m

```
function data = readOneLine(fid, Noise)
% READONELINE Read one line/sample from file
    tline = fgets(fid);
    if tline == -1
        data = -1;
        return;
    end
    tline = regexprep(tline, ' ', ''); % Remove whitespace
    data = tline - '0';  % ASCII to Int
    data = data(1:end-1); % Remove newline character
    data = data + Noise*randn(size(data)); % Apply gaussian noise
 noise

 % Limit the data range to 0.001 to 1
 % The bottom limit is due to the need of evaluate ln(x)
    data(data>1) = 1;
    data(data<=0) = 0.001;
end
```

```
% Guassian Distribution Maximum Likelihood Estimation
% Please run 'Init_Read' before this script

% Maxmimum Likelihood fitting
[Means, Covs] = gaussianMLFitting(TrainSet);

% Set the priors
% Since we do not have prior knowledge of the data, the priors are
 uniform
Priors = ones(MAX_CLASS, 1)./MAX_CLASS;

% For each test sample (coming from different classes), test against
% all the classes
Likelihoods = cell(1, 1);
Denominators = cell(1, 1);
Posteriors = cell(1, 1);
for i = 1 : MAX_CLASS % For each test set
    SampleLikelihoods = zeros(size(TestSet{i}, 1), MAX_CLASS);
    for r = 1 : MAX_CLASS % test against each trained set
        DiagCov = diag(diag(Covs{r})); % Diagonalize to avoid rank
 dificient issue
        SampleLikelihoods(:, r) = mvnpdf(TestSet{i}, Means{r},
 DiagCov);
        Likelihoods{i} = SampleLikelihoods;
    end
end


% Calculate Posterior using Bayes' rule.
% This snippet is one of the examples given by the textbook for
i = 1 : MAX_CLASS
    Denominator{i} = 1 ./ (Likelihoods{i} * Priors); Posteriors{i}
    = (Likelihoods{i} * diag(Priors)); Posteriors{i} =
    (Posteriors{i}.' * diag(Denominator{i}));
end

% Check the amount of samples that are correctly labeled.
% For each test sample (coming from different classes), check if the
% maximum posterior is the correct class
CorrectCount = zeros(1, MAX_CLASS);
for i = 1 : MAX_CLASS
    [M,I] = max(Posteriors{i});
    CorrectCount(i) = nnz(I==i);
end

% Print stuff
CorrectPercentages = CorrectCount./TestCount*100;
TotalPercentage = sum(CorrectCount)/sum(TestCount);
disp(sprintf('Correct/Total: %.2f%%', TotalPercentage*100));
```

gaussianMLFitting.m

```
function [Means, Covs] = gaussianMLFitting(TrainSet)
% GAUSSIANMLFITTING Maximum Likelihood Multivariate Normal
 Distribution Fitting
% INPT: TrainSet: LxNxD matrix. A training set of L classes, N samples and D
 dimension.
% OUPT: Means: 1xL cell array. Each cell contains the Mean of a class.
%Covs: 1xL cell raary. Each cell contains the Covariance matrix of a class.

Means = cell(1, 1);
Covs = cell(1, 1);

for i = 1 : length(TrainSet)
    [SampleMean, SampleCov] = mvnML(TrainSet{i});
    Means{i} = SampleMean;
    Covs{i} = SampleCov;
end

end




function [sMean, sCov] = mvnML(dataSet)
% MVNML Get Mean and Cov of Multivariate Normal Dist of a dataset

setSize = size(dataSet, 1);

% Get mean of each column - 1xD
sMean = sum(dataSet,1) ./ setSize;

% Calculate covariance matrix %

% You can do it according to the formula
% For some reason this approach takes a lot of time
dataSetSubMean = double(dataSet)-repmat(sMean,size(dataSet,1),1);
sCov = (dataSetSubMean.' * dataSetSubMean);

% Another way is to use cov(double(dataSet)) directly
% sCov = cov(double(dataSet));

% Another one
% sCov = zeros (size(dataSet,2), size(dataSet,2));
% for i = 1 : setSize
%     M = double(dataSet(i, :)) - sMean;
%     M = M' * M;
%     sCov = sCov + M;
% end

if setSize > 1
     sCov = sCov ./ (setSize-1);
end

end
```

GammaMLE.m

```
% Gamma Distribution Maximum Likelihood Estimation
% Please run 'Init_Read' before this script

% Fitting training data
[Ks, Thetas] = gammaMLFitting(TrainSet);

% Set the priors
% Since we do not have prior knowledge of the data, the priors are
 uniform
Priors = ones(MAX_CLASS, 1)./MAX_CLASS;

% For each test sample (coming from different classes), test against
% all the classes
Likelihoods = cell(1, 1);
Denominators = cell(1, 1);
Posteriors = cell(1, 1);
for i = 1 : MAX_CLASS % For each test set
    SampleLikelihoods = zeros(size(TestSet{i}, 1), MAX_CLASS);
    for r = 1 : size(TestSet{i}, 1) % in each test sample for p
        = 1 : MAX_CLASS % against each trained sample
   % The likelihood of a sample is the product of likelihood of every
 pixels
            SampleLikelihoods(r, p) = prod(gampdf(TestSet{i}(r, :),
 Ks{p}, Thetas{p}));
        end
    end

 % Store the likelihoods of each sample in each test set
    Likelihoods{i} = SampleLikelihoods;
end

% Calculate Posterior using Bayes' rule.
% This snippet is one of the examples given by the textbook for
i = 1 : MAX_CLASS
    Denominator{i} = 1 ./ (Likelihoods{i} * Priors); Posteriors{i}
    = (Likelihoods{i} * diag(Priors)); Posteriors{i} =
    (Posteriors{i}.' * diag(Denominator{i}));
end

% Check the amount of samples that are correctly labeled.
% For each test sample (coming from different classes), check if the
% maximum posterior is the correct class
CorrectCount = zeros(1, MAX_CLASS);
for i = 1 : MAX_CLASS
    [M,I] = max(Posteriors{i});
    CorrectCount(i) = nnz(I==i);
end

% Print stuff
CorrectPercentages = CorrectCount./TestCount*100;
TotalPercentage = sum(CorrectCount)/sum(TestCount);
disp(sprintf('Correct/Total: %.2f%%', TotalPercentage*100));
```

```
function [Ks, Thetas] = gammaMLFitting(TrainSet)
% GAMMAMLFITTING Calculate the Gamma Distribution parameter of a given
 training set
% INPT: TrainSet: LxNxD matrix. A training set of L classes, N samples and D
 dimension.
% OUPT: Ks: 1xL cell array. Each cell contains the K of a class.
%  Thetas: 1xL cell raary. Each cell contains the Theta of a class.
% Please refer to the report

Ks = cell(1, 1);
Thetas = cell(1, 1);

% Calculate the Gamma Distribution parameter of each training sample for i
= 1 : length(TrainSet)
    [SampleK, SampleCov] = gammaML(TrainSet{i});
    Ks{i} = SampleK;
    Thetas{i} = SampleCov;
end

end

function [sK, sTheta] = gammaML(dataSet)
% GAMMAML Get K and Theta of Gamma Distribution
% Please refer to the report
sK = zeros(1, size(dataSet, 2)); sTheta
= zeros(1, size(dataSet, 2)); for i =
1 : size(dataSet, 2)
    data = dataSet(:, i).'; N
    = size(data, 2);
    S = log(1/N*sum(data)) - 1/N*sum(log(data));

    SampleK = (3 - S + sqrt((S-3).^2 + 24*S)) / (12 * S);
    for r = 1 : 5
        SampleK = SampleK - ((log(SampleK) - psi(SampleK) - S) / (1/
SampleK - psi(1, SampleK)));
    end
    sK(i) = SampleK;
    sTheta(i) = 1/(SampleK*N)*sum(data);
end

end
```