{::nomarkdown}
html { min-height:100%; margin-bottom:1px; } html body { height:100%; margin:0px; font-family:Arial, Helvetica, sans-serif; font-size:10px; color:#000; line-height:140%; background:#fff none; overflow-y:scroll; } html body td { vertical-align:top; text-align:left; }
h1 { padding:0px; margin:0px 0px 25px; font-family:Arial, Helvetica, sans-serif; font-size:1.5em; color:#d55000; line-height:100%; font-weight:normal; } h2 { padding:0px; margin:0px 0px 8px; font-family:Arial, Helvetica, sans-serif; font-size:1.2em; color:#000; font-weight:bold; line-height:140%; border-bottom:1px solid #d6d4d4; display:block; } h3 { padding:0px; margin:0px 0px 5px; font-family:Arial, Helvetica, sans-serif; font-size:1.1em; color:#000; font-weight:bold; line-height:140%; }
a { color:#005fce; text-decoration:none; } a:hover { color:#005fce; text-decoration:underline; } a:visited { color:#004aa0; text-decoration:none; }
p { padding:0px; margin:0px 0px 20px; } img { padding:0px; margin:0px 0px 20px; border:none; } p img, pre img, tt img, li img, h1 img, h2 img { margin-bottom:0px; }
ul { padding:0px; margin:0px 0px 20px 23px; list-style:square; } ul li { padding:0px; margin:0px 0px 7px 0px; } ul li ul { padding:5px 0px 0px; margin:0px 0px 7px 23px; } ul li ol li { list-style:decimal; } ol { padding:0px; margin:0px 0px 20px 0px; list-style:decimal; } ol li { padding:0px; margin:0px 0px 7px 23px; list-style-type:decimal; } ol li ol { padding:5px 0px 0px; margin:0px 0px 7px 0px; } ol li ol li { list-style-type:lower-alpha; } ol li ul { padding-top:7px; } ol li ul li { list-style:square; }
.content { font-size:1.2em; line-height:140%; padding: 20px; }
pre, code { font-size:12px; } tt { font-size: 1.2em; } pre { margin:0px 0px 20px; } pre.codeinput { padding:10px; border:1px solid #d3d3d3; background:#f7f7f7; } pre.codeoutput { padding:10px 11px; margin:0px 0px 20px; color:#4c4c4c; } pre.error { color:red; }
@media print { pre.codeinput, pre.codeoutput { word-wrap:break-word; width:100%; } }
span.keyword { color:#0000FF } span.comment { color:#228B22 } span.string { color:#A020F0 } span.untermstring { color:#B20000 } span.syscmd { color:#B28C00 }
.footer { width:auto; padding:10px 0px; margin:25px 0px 0px; border-top:1px dotted #878787; font-size:0.8em; line-height:140%; font-style:italic; color:#878787; text-align:left; float:none; } .footer p { margin:0px; } .footer a { color:#878787; } .footer a:hover { color:#878787; text-decoration:underline; } .footer a:visited { color:#878787; }
table th { padding:7px 5px; text-align:left; vertical-align:middle; border: 1px solid #d6d4d4; font-weight:bold; } table td { padding:7px 5px; text-align:left; vertical-align:top; border:1px solid #d6d4d4; }
Contents
Tai Duc Nguyen - HW1 - 09/30/2019
HW CX 2.7
HW CX 2.8
Using the data generation procedures from textbook CX 2.3 Page 80 with slight modifications
Using the Bayes classifier algorithm from textbook CX 2.5 Page 81 with slight modifications
Using the Euclidean classifier algorithm from textbook CX 2.6 Page 82 with slight modifications
Calculate the probability density value (using equation from page 30, class's slides ac_1_classifier_bayes_1.ppt)
Using the Euclidean classifier algorithm from textbook CX 2.8 Page 82-83 with modifications
Tai Duc Nguyen - HW1 - 09/30/2019

```
seed = 0
randn('seed',seed);
```

```
seed =

     0
```

HW CX 2.7

```
clear; close all;
```

```
% Prepare the variables:
N = 10000;

m1 = [1;1];
m2 = [4;4];
m3 = [8;1];
m = [m1 m2 m3];

S1 = eye(2)*2;
S2 = S1;
S3 = S1;
S(:,:,1) = S1; S(:,:,2) = S2; S(:,:,3) = S3;

P_x5 = [1/3;1/3;1/3];
P_x5_ = [0.8;0.1;0.1];

% Generate data set X5 and X5_ and class assignments y5 and y5_
[X5, y5] = generate_gauss_classes(m, S, P_x5, N);
[X5_, y5_] = generate_gauss_classes(m, S, P_x5_, N);

% Classify data set
z5_bayes = bayes_classifier(m,S,P_x5,X5);
z5_bayes_ = bayes_classifier(m,S,P_x5_,X5_);
z5_eu = euclid_classifier(m,X5);
z5_eu_ = euclid_classifier(m,X5_);

% Calculate error for the bayes classifiers
error_x5_bayes = z5_bayes-y5;
error_x5_bayes(error_x5_bayes > 0) = 1; error_x5_bayes(error_x5_bayes < 0) = 1;
error_x5_bayes = sum(error_x5_bayes)/N

error_x5_bayes_ = z5_bayes_-y5_;
error_x5_bayes_(error_x5_bayes_ > 0) = 1; error_x5_bayes_(error_x5_bayes_ < 0) = 1;
error_x5_bayes_ = sum(error_x5_bayes_)/N

% Calculate error for the euclidean classifiers
error_x5_eu = z5_eu-y5;
error_x5_eu(error_x5_eu > 0) = 1; error_x5_eu(error_x5_eu < 0) = 1;
error_x5_eu = sum(error_x5_eu)/N

error_x5_eu_ = z5_eu_-y5_;
error_x5_eu_(error_x5_eu_ > 0) = 1; error_x5_eu_(error_x5_eu_ < 0) = 1;
error_x5_eu_ = sum(error_x5_eu_)/N
```

HW CX 2.8

```
clear; close all;

% Prepare the variables:
N = 1000;

m1 = [1;1];
m2 = [8;6];
```

```
m3 = [13;1];
m = [m1 m2 m3];

S1 = eye(2)*6;
S2 = S1;
S3 = S1;
S(:,:,1) = S1; S(:,:,2) = S2; S(:,:,3) = S3;

P = [1/3;1/3;1/3];

% Generate data set X, Z and class assignments y_x, y_z
[X, y_x] = generate_gauss_classes(m, S, P, N);
[Z, y_z] = generate_gauss_classes(m, S, P, N);

% Run KNN classifier with k=1 and k=11
z_knn_1 = k_nn_classifier(Z,y_z,1,X);
z_knn_11 = k_nn_classifier(Z,y_z,11,X);

% Calculate error for the KNN classifiers
error_knn_1 = z_knn_1-y_x;
error_knn_1(error_knn_1 > 0) = 1; error_knn_1(error_knn_1 < 0) = 1;
error_knn_1 = sum(error_knn_1)/N

error_knn_11 = z_knn_11-y_x;
error_knn_11(error_knn_11 > 0) = 1; error_knn_11(error_knn_11 < 0) = 1;
error_knn_11 = sum(error_knn_11)/N
```

Using the data generation procedures from textbook CX 2.3 Page 80 with slight modifications

```
function [X,y] = generate_gauss_classes(m,S,P,N)
[~,c]=size(m);
X=[];
y=[];
for j=1:c
% Generating the [p(j)*N)] vectors from each distribution
t=mvnrnd(m(:,j),S(:,:,j),fix(P(j)*N));
% The total number of points may be slightly less than N
% due to the fix operator
X=[X; t];
y=[y ones(1,fix(P(j)N))j];
end
end
```

Using the Bayes classifier algorithm from textbook CX 2.5 Page 81 with slight modifications

```
function z = bayes_classifier(m,S,P,X)
[~,c]= size(m); % c=no. of classes
[N,~]=size(X);  % N=no. of vectors
t=zeros(c,1);
z=zeros(1,N);
for i=1:N
for j=1:c
t(j)=P(j)prob_density_value(X(i,:),m(:,j)',S(:,:,j));
```

```
end
% Determining the maximum quantity Pip(x|wi)
[~,z(i)]=max(t);
end
end
```

Using the Euclidean classifier algorithm from textbook CX 2.6 Page 82 with slight modifications

```
function z = euclid_classifier(m,X)
[~,c]=size(m); % c=no. of classes
[N,~]=size(X); % N=no. of vectors/
t=zeros(c,1);
z=zeros(1,N);
for i=1:N
for j=1:c
t(j)=sqrt((X(i,:)-m(:,j)')(X(i,:)-m(:,j)')');
end
% Determining the maximum quantity Pip(x|wi)
[~,z(i)]=min(t);
end
end
```

```
error_x5_bayes =
```

    0.0711

```
error_x5_bayes_ =
```

    0.0436

```
error_x5_eu =
```

    0.0711

```
error_x5_eu_ =
```

    0.0708

Calculate the probability density value (using equation from page 30, class's slides ac_1_classifier_bayes_1.ppt)

```
function res = prob_density_value(X,m,S)
[b,~]=size(m);
res = 1/((2*pi)^(b/2)*det(S)^(1/2))exp(-1/2(X-m)inv(S)(X-m)');
end
```

Using the Euclidean classifier algorithm from textbook CX 2.8 Page 82-83 with modifications

```
function z=k_nn_classifier(Z,v,k,X)
[~,N1]=size(Z);
[N,~]=size(X);
c=max(v); % The number of classes
% Computation of the (squared) Euclidean distance
% of a point from each reference vector
z=zeros(1,N);
for i=1:N
dist=sum(((repmat(X(i,:),N,1)-Z).^ 2),2);
%Sorting the above distances in ascending order
[~,n]=sort(dist);
% Counting the class occurrences among the k-closest
% reference vectors Z(:,i)
refe=zeros(1,c); %Counting the reference vectors per class
for q=1:k
class=v(n(q));
refe(class)=refe(class)+1;
end
[~,z(i)]=max(refe);
end
end
```

```
error_knn_1 =
```

```
0.1050
```

```
error_knn_11 =
```

```
0.0700
```

Published with MATLAB® R2018a
<!--
SOURCE BEGIN
%% Tai Duc Nguyen - HW1 - 09/30/2019
seed = 0 randn('seed',seed);
%% HW CX 2.7
clear; close all;
% Prepare the variables: N = 10000;

m1 = [1;1]; m2 = [4;4]; m3 = [8;1]; m = [m1 m2 m3];
S1 = eye(2)*2; S2 = S1; S3 = S1; S(:,:,1) = S1; S(:,:,2) = S2; S(:,:,3) = S3;
P_x5 = [1/3;1/3;1/3]; P_x5_ = [0.8;0.1;0.1];
% Generate data set X5 and X5_ and class assignments y5 and y5_ [X5, y5] = generate_gauss_classes(m, S, P_x5, N); [X5_, y5_] = generate_gauss_classes(m, S, P_x5_, N);
% Classify data set z5_bayes = bayes_classifier(m,S,P_x5,X5); z5_bayes_ = bayes_classifier(m,S,P_x5_,X5_); z5_eu = euclid_classifier(m,X5); z5_eu_ = euclid_classifier(m,X5_);
% Calculate error for the bayes classifiers error_x5_bayes = z5_bayes-y5; error_x5_bayes(error_x5_bayes > 0) = 1; error_x5_bayes(error_x5_bayes < 0) = 1; error_x5_bayes = sum(error_x5_bayes)/N
error_x5_bayes_ = z5_bayes_-y5_; error_x5_bayes_(error_x5_bayes_ > 0) = 1; error_x5_bayes_(error_x5_bayes_ < 0) = 1; error_x5_bayes_ = sum(error_x5_bayes_)/N
% Calculate error for the euclidean classifiers error_x5_eu = z5_eu-y5; error_x5_eu(error_x5_eu > 0) = 1; error_x5_eu(error_x5_eu < 0) = 1; error_x5_eu = sum(error_x5_eu)/N
error_x5_eu_ = z5_eu_-y5_; error_x5_eu_(error_x5_eu_ > 0) = 1; error_x5_eu_(error_x5_eu_ < 0) = 1; error_x5_eu_ = sum(error_x5_eu_)/N
%% HW CX 2.8
clear; close all;
% Prepare the variables: N = 1000;
m1 = [1;1]; m2 = [8;6]; m3 = [13;1]; m = [m1 m2 m3];
S1 = eye(2)*6; S2 = S1; S3 = S1; S(:,:,1) = S1; S(:,:,2) = S2; S(:,:,3) = S3;
P = [1/3;1/3;1/3];
% Generate data set X, Z and class assignments y_x, y_z [X, y_x] = generate_gauss_classes(m, S, P, N); [Z, y_z] = generate_gauss_classes(m, S, P, N);
% Run KNN classifier with k=1 and k=11 z_knn_1 = k_nn_classifier(Z,y_z,1,X); z_knn_11 = k_nn_classifier(Z,y_z,11,X);
% Calculate error for the KNN classifiers error_knn_1 = z_knn_1-y_x; error_knn_1(error_knn_1 > 0) = 1; error_knn_1(error_knn_1 < 0) = 1; error_knn_1 = sum(error_knn_1)/N
error_knn_11 = z_knn_11-y_x; error_knn_11(error_knn_11 > 0) = 1; error_knn_11(error_knn_11 < 0) = 1; error_knn_11 = sum(error_knn_11)/N
%% Using the data generation procedures from textbook CX 2.3 Page 80 with slight modifications function [X,y] = generate_gauss_classes(m,S,P,N) [~,c]=size(m); X=[]; y=[]; for j=1:c % Generating the [p(j)*N)] vectors from each distribution t=mvnrnd(m(:,j),S(:,:,j),fix(P(j)*N)); % The total number of points may be slightly less than N % due to the fix operator X=[X; t]; y=[y ones(1,fix(P(j)*N))*j]; end end
%% Using the Bayes classifier algorithm from textbook CX 2.5 Page 81 with slight modifications function z = bayes_classifier(m,S,P,X) [~,c]= size(m); % c=no. of classes [N,~]=size(X); % N=no. of vectors t=zeros(c,1); z=zeros(1,N); for i=1:N for j=1:c t(j)=P(j)*prob_density_value(X(i,:),m(:,j)',S(:,:,j)); end % Determining the maximum quantity Pi*p(x|wi) [~,z(i)]=max(t); end end
%% Using the Euclidean classifier algorithm from textbook CX 2.6 Page 82 with slight modifications function z = euclid_classifier(m,X) [~,c]=size(m); % c=no. of classes [N,~]=size(X); % N=no. of vectors/ t=zeros(c,1); z=zeros(1,N); for i=1:N for j=1:c t(j)=sqrt((X(i,:)-m(:,j)')*(X(i,:)-m(:,j)')'); end % Determining the maximum quantity Pi*p(x|wi) [~,z(i)]=min(t); end end
%% Calculate the probability density value (using equation from page 30, class's slides ac_1_classifier_bayes_1.ppt) function res = prob_density_value(X,m,S) [b,~]=size(m); res = 1/((2*pi)^(b/2)*det(S)^(1/2))*exp(-1/2*(X-m)*inv(S)*(X-m)'); end
%% Using the Euclidean classifier algorithm from textbook CX 2.8 Page 82-83 with modifications function z=k_nn_classifier(Z,v,k,X) [~,N1]=size(Z); [N,~]=size(X); c=max(v); % The number of classes % Computation of the (squared) Euclidean distance % of a point from each reference vector z=zeros(1,N); for i=1:N dist=sum(((repmat(X(i,:),N,1)-Z).^ 2),2); %Sorting the above distances in ascending order [~,n]=sort(dist); % Counting the class occurrences among the k-closest % reference vectors Z(:,i) refe=zeros(1,c); %Counting the reference vectors per class for q=1:k class=v(n(q)); refe(class)=refe(class)+1; end [~,z(i)]=max(refe); end end

SOURCE END

--> {😕}