MATH444 Final Project

Jiasen Zhang

Problem 1 and 2

In the first 2 parts I load the data and reduce the gray scale to 6 levels. It's really enough to get good results.

Problem 3 and 4

Compute 3 GLCMs using 3 pairs of offset parameters: (2,0), (0,2) and (2,2). Stack their columns into vecotrs and put together so that the dimensionality is reduced to 108. Then compute the first 4 principal components to reduce the dimensionality to 4.

The different pairs of pricinpal components are plotted in Figure 1. The corresponding colors of the classes are shown in Table 1. We can see even with a drastic dimensionality reduction, the classes are separated well. It looks like the green points and purple points are more difficult to separate, but in fact most green points just stay around rather than mix with purple points.

Classes	1	2	3	4	5
Color	Red	Blue	Green	Purple	Black

Table 1

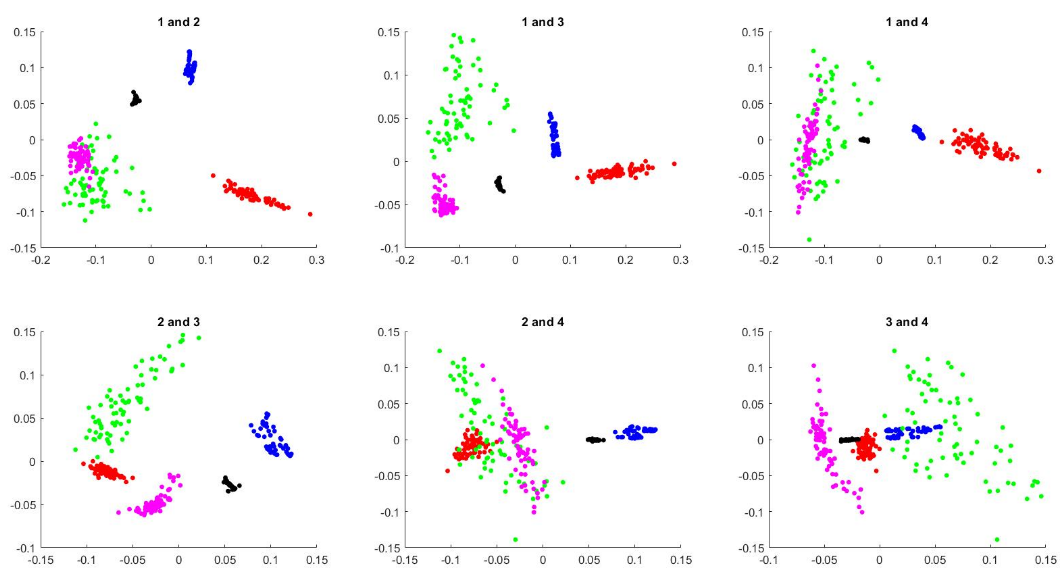

Figure 1: The first 4 principal components.

Problem 5

Using the same parameters and gray scale, compute the first 3 LDA separators and project the principal components onto these directions. As shown in Figure 2, the 5 classes are separated really well. It reflects that LDA generally performs better than PCA by making use of annotations. It also indicates that these images can be separated well with knowning the annotations, it makes sure that classification tree is suitable for these images.

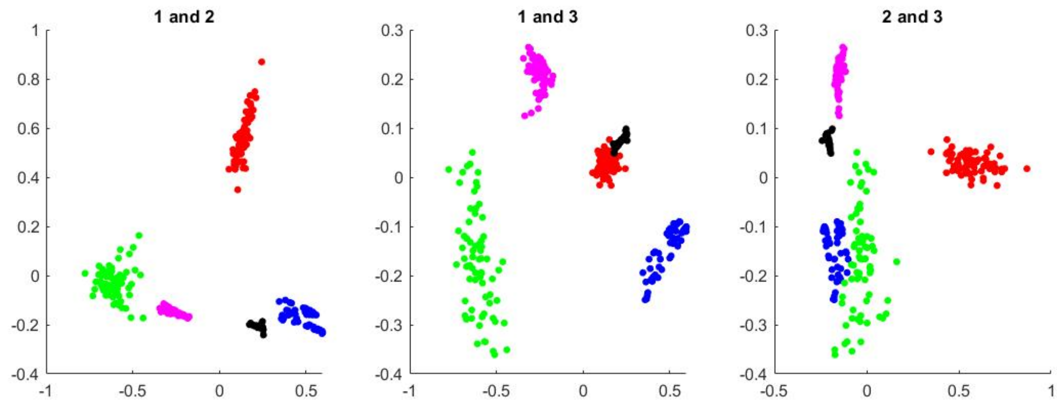

Figure 2: The first 3 LDA components (6 levels).

If I reduce the gray scale to 4 levels, the result is good too, as shown in Figure 3.

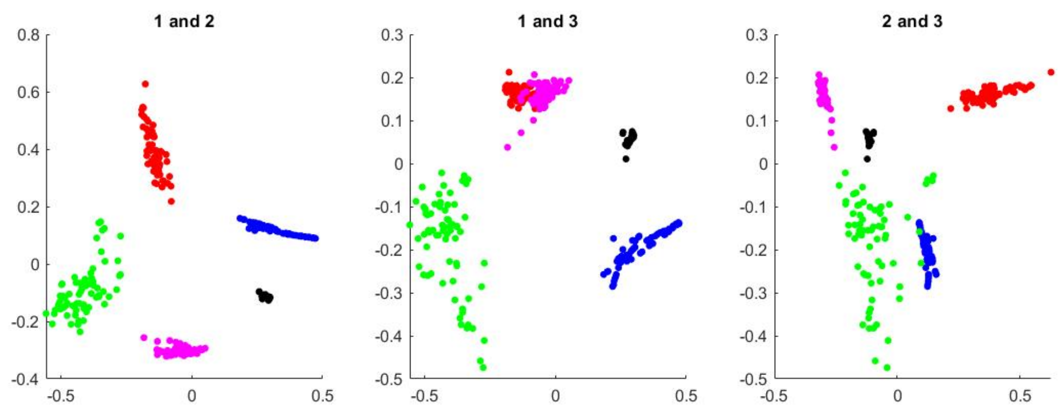

Figure 3: The first 3 LDA components (4 levels).

Problem 6

With the PCA results, select the first 200 data as the training set and generate the maximal tree, the structure is shown in Figure 4. There are 27 intervals totally with 14 pure leaves.

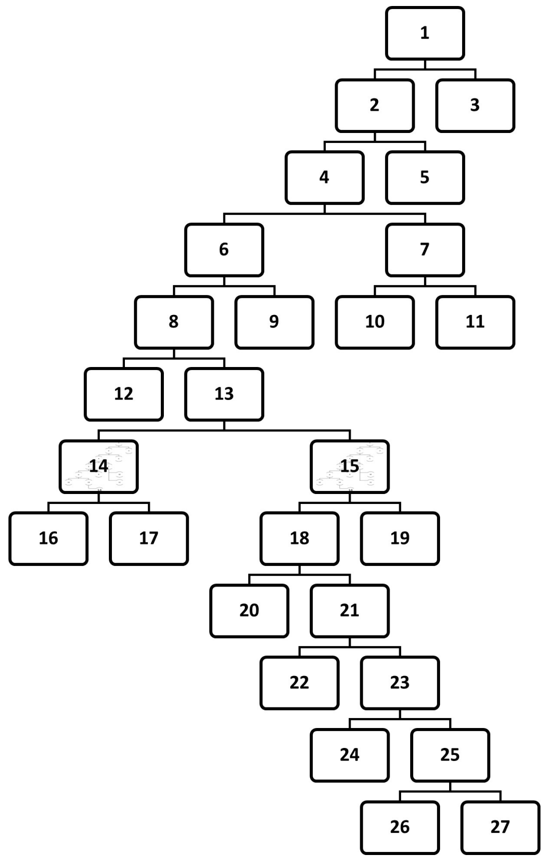

Figure 4: The maximal tree.

Next prune the tree. The times of pruning is 4. The pruned nodes and α are shown in Table 2. We can see α is an increasing sequence. The numbers of nodes are shown in Figure 4. In the 4th row, I test the trees with testing data and compute the rates of success. From the table we can see the maximal tree and that pruned for one time have the highest rate of success 0.89. Because the pruned tree has lower complexity, it can be the optimal tree (the tree pruned for one time).

Pruning steps	0	1	2	3	4
Pruned nodes	N/A	7, 23	18	6	1
α	0	0.005	0.0125	0.015	0.0183
Rate of success	0.89	0.89	0.88	0.76	0.17

Table 2

Problem 7

The structure of the optimal tree is shown in Figure 5. Use it to classify the rest of 50 images, the rate of success is 0.88. The confusion matrix is:

$$\begin{bmatrix} 9 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 4 & 6 & 0 \\ 0 & 0 & 2 & 0 & 10 \end{bmatrix}$$

In the matrix we can see the class 3 and 4 are difficult to separate, this result corresponds to Figure 1, in which the class 3 and 4 stay really close.

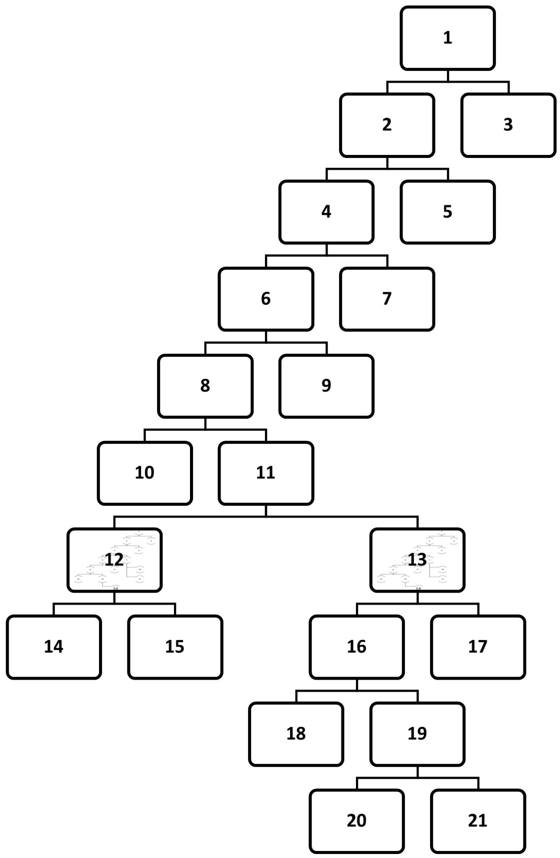

Figure 5: The optimal tree.

Matlab Code

Problem 1-7

The code has 7 parts and two subroutines.

```
clear all;clc;
% Math444 Final Project
% Jiasen Zhang

load TestImages.mat;
p = 350;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2: Reduce the gray scales of the images
k = 6; % number of gray scales
unit = 1/k;
X = ceil(X/unit);
X(X==0)=1;
%imagesc(reshape(X(:,2),128,128));colormap(gray);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 3: Three GLCMs
GLCM = zeros(3*k^2,p);
mu=2;
nu=0;
for n=1:p
    C = reshape(X(:,n),128,128);
    C_shift = NaN(128,128);
    C_shift(1:128-mu,1:128-nu) = C(1+mu:128,1+nu:128);
    G = zeros(k,k);
    for i=1:k
        Ii = (C==i);
        for j=1:k
            Ij = (C_shift==j);
            G(i,j) = sum(sum(Ii.*Ij));
        end
    end
    G = G/(sum(sum(G))); % normalize G
    GLCM(1:k^2,n) = reshape(G/3,k^2,1);
end
mu=0;
nu=2;
for n=1:p
    C = reshape(X(:,n),128,128);
    C_shift = NaN(128,128);
    C_shift(1:128-mu,1:128-nu) = C(1+mu:128,1+nu:128);
    G = zeros(k,k);
    for i=1:k
        Ii = (C==i);
        for j=1:k
            Ij = (C_shift==j);
            G(i,j) = sum(sum(Ii.*Ij));
        end
    end
    G = G/(sum(sum(G))); % normalize G
    GLCM(k*k+1:2*k^2,n) = reshape(G/3,k^2,1);
end
mu=2;
nu=2;
for n=1:p
    C = reshape(X(:,n),128,128);
    C_shift = NaN(128,128);
    C_shift(1:128-mu,1:128-nu) = C(1+mu:128,1+nu:128);
    G = zeros(k,k);
```

```
    for i=1:k
        Ii = (C==i);
        for j=1:k
            Ij = (C_shift==j);
            G(i,j) = sum(sum(Ii.*Ij));
        end
    end
    G = G/(sum(sum(G))); % normalize G
    GLCM(2*k*k+1:3*k^2,n) = reshape(G/3,k^2,1);
end
% Compute the first 4 principal components
GLCMbar = mean(GLCM,2);
GLCMc = GLCM - GLCMbar; % centered data
[u,¬,¬] = svd(GLCMc);
z = u(:,1:4)'*GLCMc;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 4: plot using different colors or the same color
nplot = 1;
figure(1);
for i=1:3
    for j=i+1:4
        subplot(2,3,nplot);
        scatter(z(i,I==1),z(j,I==1),'r','k.','SizeData',200); hold on;
        scatter(z(i,I==2),z(j,I==2),'b','k.','SizeData',200);
        scatter(z(i,I==3),z(j,I==3),'g','k.','SizeData',200);
        scatter(z(i,I==4),z(j,I==4),'m','k.','SizeData',200);
        scatter(z(i,I==5),z(j,I==5),'k','k.','SizeData',200);
        title(strcat(num2str(i),{32},'and',{32},num2str(j)));
        nplot = nplot+1;
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 5: Compute the first 3 LDA separators
[zLDA,¬] = lda(5, z, I);
nplot=1;
figure(2);
for i=1:2
    for j=i+1:3
        subplot(1,3,nplot);
        scatter(zLDA(i,I==1),zLDA(j,I==1),'r','k.','SizeData',200); hold on;
        scatter(zLDA(i,I==2),zLDA(j,I==2),'b','k.','SizeData',200);
        scatter(zLDA(i,I==3),zLDA(j,I==3),'g','k.','SizeData',200);
        scatter(zLDA(i,I==4),zLDA(j,I==4),'m','k.','SizeData',200);
        scatter(zLDA(i,I==5),zLDA(j,I==5),'k','k.','SizeData',200);
        title(strcat(num2str(i),{32},'and',{32},num2str(j)));
        nplot = nplot+1;
    end
end


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Training set and testing set
I_train = I(1:200);
I_test = I(201:300);
I0 = I(301:end);
z_train = z(:,1:200);
z_test = z(:,201:300);
z0 = z(:,301:end);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 6: Classification tree: using z_train
% initialize the tree
% structure R
```

```
R(1).idx = 1:200; % index of data in the intervel
R(1).data = z_train;
R(1).n = 1; % number of R(1)
R(1).split_idx = [];
R(1).split_value = [];
R(1).leftR = []; % left child
R(1).rightR = []; % right child
[R(1).g, R(1).c] = gini(I_train(R(1).idx)); % gini index and classification
Lpure = [];
Lmixed = [1];
m = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 6.1: Build a maximal tree
while(1)
% pick one index from Lmixed randomly:
pick = Lmixed(1);
Lmixed = Lmixed(2:end);

% find split index and value of R(pick)
[j, s] = optimal(R(pick).idx, I_train, z_train);
R(pick).split_idx = j;
R(pick).split_value = s;

% build idxleft and idxright, generate m+1 and m+2
R(pick).leftR = m+1;
R(pick).rightR = m+2;
dataj = R(pick).data(j,:);
idx_left = find(dataj≤s);
idx_right = find(dataj>s);
%
R(m+1).idx = R(pick).idx(idx_left);
R(m+1).data = R(pick).data(:,idx_left);
R(m+1).n = m+1;
R(m+2).idx = R(pick).idx(idx_right);
R(m+2).data = R(pick).data(:,idx_right);
R(m+2).n = m+2;

% decide if new leaves are pure or mixed
[R(m+1).g, R(m+1).c] = gini(I_train(R(m+1).idx));
if R(m+1).g==0
    Lpure = [Lpure,m+1];
else
    Lmixed = [Lmixed,m+1];
end
[R(m+2).g, R(m+2).c] = gini(I_train(R(m+2).idx));
if R(m+2).g==0
    Lpure = [Lpure,m+2];
else
    Lmixed = [Lmixed,m+2];
end
m=m+2;

% If all leaves are pure, stop
if isempty(Lmixed)==1
    break;
end

end
Rmax = R;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 6.2: Prune the tree
% initial succesor matrix A
A = zeros(m,m);
for i=1:m
    if ¬isempty(R(i).leftR)
        A(i,R(i).leftR)=1;
```

```
        A(i,R(i).rightR)=1;
    end
end
alpha0 = [0]; % save alpha
T(1,:) = R; % put the maximal tree into T
nNodes = m;
numLeaf = []; % number of leaves of each pruned tree

while(m>1)
%for nn=1:4
% update Lpure and Lmixed
Lpure = [];
Lmixed = [];
for i=1:m
    if isempty(R(i).leftR)==1 % if it's a leaf
        if R(i).g==0
            Lpure = [Lpure, i];
        else
            Lmixed = [Lmixed, i];
        end
    end
end
% genealogy matrix G
AA = A;
G = zeros(m,m);
while(sum(sum(AA))>0)
    G = G+AA;
    AA = A*AA;
end
% find non-leaf intervals for the tree
Jleaf = [];
Jnonleaf = [];
for i=1:m
    if sum(G(i,:))==0
        Jleaf = [Jleaf,i];
    else
        Jnonleaf = [Jnonleaf,i];
    end
end
numLeaf = [numLeaf,length(Jleaf)];
% compute alpha for each nonleaf interval
alpha = zeros(1,length(Jnonleaf));
for i=1:length(Jnonleaf)
    j0 = Jnonleaf(i);
    % frequency multiply misclassification rate: v*r
    if R(j0).c == R(R(j0).leftR).c
        vr = length(R(R(j0).rightR).idx)/length(R(1).idx);
    elseif R(j0).c == R(R(j0).rightR).c
        vr = length(R(R(j0).leftR).idx)/length(R(1).idx);
    end
    % compute # of pure leaves for each subtree
    successor = find(G(j0,:));
    %nL = sum(ismember([R(successor).n],Lpure))+sum(ismember([R(successor).n],Lmixed));
    nL = sum(ismember(successor,Lpure))+sum(ismember(successor,Lmixed));
    % v and r of leaves
    nominator = vr;
    for j=1:length(successor)
        if isempty(R(successor(j)).leftR) % If it's a leaf
            vrL = sum(I_train(R(successor(j)).idx)≠R(successor(j)).c)/length(R(1).idx);
            nominator = nominator - vrL;
        end
    end
    % compute alpha
    alpha(i) = nominator/(nL-1);
end
% choose smallest alpha and prune the corresponding subtree
alpha0 = [alpha0,min(alpha)];
nPrune = Jnonleaf(alpha==min(alpha))
```

```
nKeep = find(sum(G(nPrune,:),1)==0);
R = R(nKeep);
A = A(nKeep,nKeep);
% m, and save the pruned tree into T
m = length(nKeep);
nNodes = [nNodes,m];
for ell=1:m
    if sum(A(ell,:))==0
        R(ell).split_idx = [];
        R(ell).split_value = [];
        R(ell).leftR = [];
        R(ell).rightR = [];
    else
        i_1 = find(A(ell,:)==1,1,'first');
        i_2 = find(A(ell,:)==1,1,'last');
        R(ell).leftR = i_1;
        R(ell).rightR = i_2;
    end
end
T(end+1,1:m) = R;
end
numLeaf = [numLeaf,1];
alpha0



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 6.3: choose the tree with testing data
[numTree,¬] = size(T);
rate = zeros(1,numTree);
for n=1:numTree % for each tree
    % classification of testing data
    R = T(n,1:nNodes(n));
    class = zeros(1,length(I_test)); % result of classification
    for k=1:length(I_test) % for each data vector in the testing data
        i=1;
        while(isempty(R(i).leftR)==0) % if it's not a leaf
            j = R(i).split_idx;
            s = R(i).split_value;
            if z_test(j,k)≤s
                i = R(i).leftR;
            elseif z_test(j,k)>s
                i = R(i).rightR;
            end
        end
        class(k) = R(i).c;
    end
    rate(n) = sum(class==I_test)/length(I_test);
%     confusionM = zeros(5,5);
%     for k=1:length(I_test)
%         confusionM(class(k),I_test(k)) = confusionM(class(k),I_test(k))+1;
%     end

end
rate



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 7: Test the tree using the rest of data
n=find(rate==max(rate));
n=n(end)
R = T(n,1:nNodes(n));
class = zeros(1,length(I0)); % result of classification
for k=1:length(I0) % for each data vector in the testing data
    i=1;
    while(isempty(R(i).leftR)==0) % if it's not a leaf
        j = R(i).split_idx;
```

```
        s = R(i).split_value;
        if z0(j,k)≤s
            i = R(i).leftR;
        elseif z0(j,k)>s
            i = R(i).rightR;
        end
    end
    class(k) = R(i).c;

end
confusionM = zeros(5,5);
for k=1:length(I0)
    confusionM(class(k),I0(k)) = confusionM(class(k),I0(k))+1;
end
rate = sum(class==I0)/length(I0)
confusionM

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function of optimal splitting
function [split_idx, split_value] = optimal(I, C, X)
% optimal splitting
% I = R(k).idx
% C = full annotation
% X = full data matrix

q = length(I);
RX = X(:,I); % data of I
RC = C(I); % annotation of I
giniRC = gini(RC);
%split_idx = zeros(size(X,1),1);
split_value = zeros(size(X,1),1);
dg = zeros(size(X,1),1);
dgn = zeros(1,q-1);
sigma=zeros(1,q-1);
% for each attribute
for n=1:size(X,1) % n=1:4
    % sort RX with attribute n
    [sortRX, idx] = sort(RX(n,:));
    RC = RC(idx);
    % for each data of attribute n
    for j=1:q-1
        % compute q-1 split values
        sigma(j) = (sortRX(j)+sortRX(j+1))/2;
        % compute Rr and Rl from RC
        idx1=(sortRX≤sigma(j));
        idx2=(sortRX>sigma(j));
        Rl = RC(idx1);
        Rr = RC(idx2);
        % compute q-1 impurity change
        dgn(j) = giniRC - length(Rl)*gini(Rl)/q - length(Rr)*gini(Rr)/q;
    end
    temp = sigma(dgn==max(dgn));
    % split value and gini index for attribute n=1:4
    split_value(n) = (temp(1)+temp(end))/2;
    temp = dgn(dgn==max(dgn));
    dg(n) = (temp(1)+temp(end))/2;
end

% find the largest impurity change
temp = find(dg==max(dg));
temp = temp(1);
split_idx = temp;
split_value = split_value(temp);

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function of gini index
function [g,c] = gini(annotation)
% g = gini index
% c = classification by majority vote
P = zeros(5,1);
g = 1;
for j=1:5
    P(j) = sum(annotation==j)/length(annotation); % frequency
    g = g - P(j)^2;
end
c = find(P == max(P)); % classification by majority vote
c = c(unidrnd(length(c)));

end
```

LDA function

```
function [z,Q] = lda(k, X, I)
% Jiasen Zhang LDA

% input
% k = number of clusters
% X = data
% I = partition

% output
% z = LDA reduced data
% Q = seperating direction

[n,p]=size(X);

c=mean(X,2); % global mean
ck = zeros(n,k); % cluster means
S_w = zeros(n,n);
for L=1:k
    % cluster means
    ck(:,L) = mean(X(:,I==L),2);
    % compute S_L and S_w
    X_LC = X(:,I==L) - ck(:,L);
    % S_L = X_LC*X_LC';
    S_w = S_w + X_LC*X_LC';
end

% compute S_b
X_bar = zeros(n,p);
for j=1:p
    L = I(j); % cluster
    X_bar(:,j) = ck(:,L);
end
S_b = (X_bar-c)*(X_bar-c)';

% get matrix A
d1 = max(eig(S_w));
tau = 1e-16; % if not positive definite, change to 1e-16
S_we = S_w + tau*d1*d1*eye(n);
K = chol(S_we);
A = inv(K)'*S_b/K;

% nQ largest eigenvectors of A and solve Q
nQ = 4;
Q = zeros(n,nQ);
[v,d]=eig(A);
d = diag(d)/sum(diag(d)); % proportion of trace
```

```matlab
%fprintf('Proportion of trace:\n');
for j=1:nQ
    maxindex = find(d==max(d)); % find the maximum eigenvalue
    Q(:,j) = K\v(:,maxindex);
    %fprintf('%.8f\n',d(maxindex)); % show proportions of first k-1 eigenvalues
    d(maxindex)=0;
end

% LDA reduced data
z = Q'*X;
end
```