LABORATORY EXPERIENCE #5 (Hierarchical Trees)

Introduction

Hierachical Trees are a category of clustering/classification techniques that allow to build a hierarchy of clusters/classes starting from a given dataset. The assignment requires to implement both **agglomerative clustering** (a bottom-up approach of hierarchical clustering) and **hierarchical classification** on data belonging to a collection of measurements on patients affected by Chronic Kidney disease. The input dataset is public and available at https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease.

Data preparation

As the original dataset contains also nominal values, it is necessary to convert this values to numbers in order to allow Matlab to process the data. The procedure to do it is described in the assignment of the lab.
Then, we also isolate the last column of the dataset matrix, that stores the results of classification given by the medical doctor, that divide the patients affected by Chronic Kidney disease and the healthy patients. This column vector will be used to verify the correctness of the clustering/classification.

Algorithms implementation and results

The first task of the laboratory experience requires to perform **agglomerative clustering**, that starting from a set of N observations, evaluates the N*(N-1)/2 distances between them and agglomerates the two observations at minimum distance. Then, repeats the procedure on the N-1 objects performing agglomeration between observations and linkage between intermediate clusters, until a single cluster containing all the data is reached.
The method for evaluating the distance between observations and clusters can be arbitrarily chosen, and in this case study Euclidean distance was used for observations and minimum linkage for clusters. Matlab provides a sequence of functions to build the hierarchical tree, which output is a **dendrogram**.

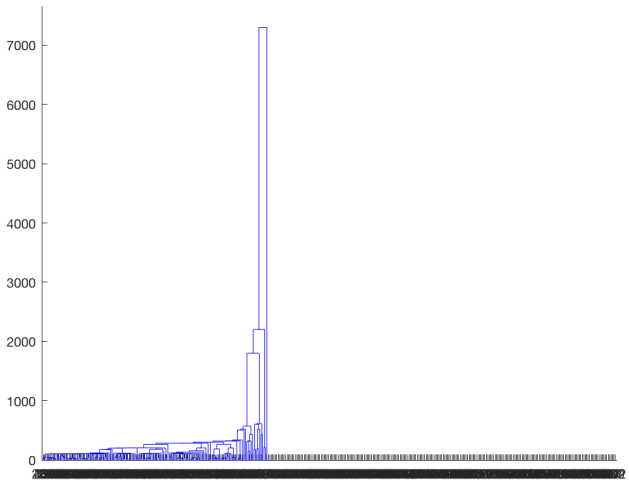

Figure 1: Dendrogram

The advantage of hierarchical clustering is that we don't have a priori to decide the number of clusters, but we can "cut" the clustering tree (dendrogram) at will. In this case we decide to extract two clusters, that will hopefully trace the decision made by the medical doctor. The resulting clustering happens to be quite good, with a strike rate of 80% and good values in both sensitivity and specificity.

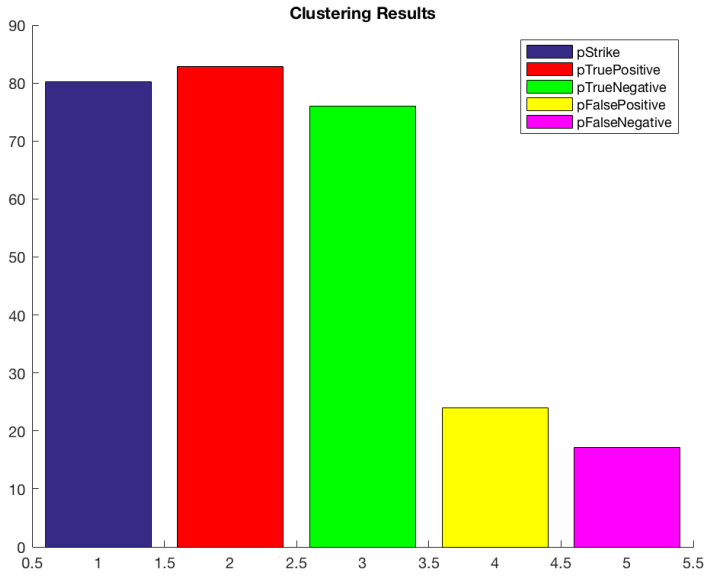

Figure 2: Performances of clustering

An attempt to measure the compactness of the clusters has been performed through the evaluation of the sum of squared errors (SSE), but the presence of many NaN values distributed among all the features invalidated the process.

Then, the lab assignment required to perform **hierarchical classification**. This time the algorithm follows a top-down approach, dividing step-by-step the dataset through a sequence of decision made on the values of the observed features. This approach allows to build a decision tree on top on which we are able to extract which are the most meaningful features to classify our data. Also in this case we can take advantage from a set of Matlab functions, that this time also need to get as input the column vector of the classes of original dataset.
In this case it would be very recommended to perform PCA before starting the classification. Unfortunately, the presence of the NaN values invalidated also this task, so the classification algorithm was ran without executing this step.
From the resulting decision tree, we can infer that 4 features are sufficient to classify our data. In particular they are, in order, feature 15 (Hemoglobin), 16 (Packed Cell Volume), 3 (Specific Gravity) and feature 4 (Albumin). Writing a short algorithm that follows the decision tree we are able to actually perform classification and to compare the results with the classes given by the medical doctor.

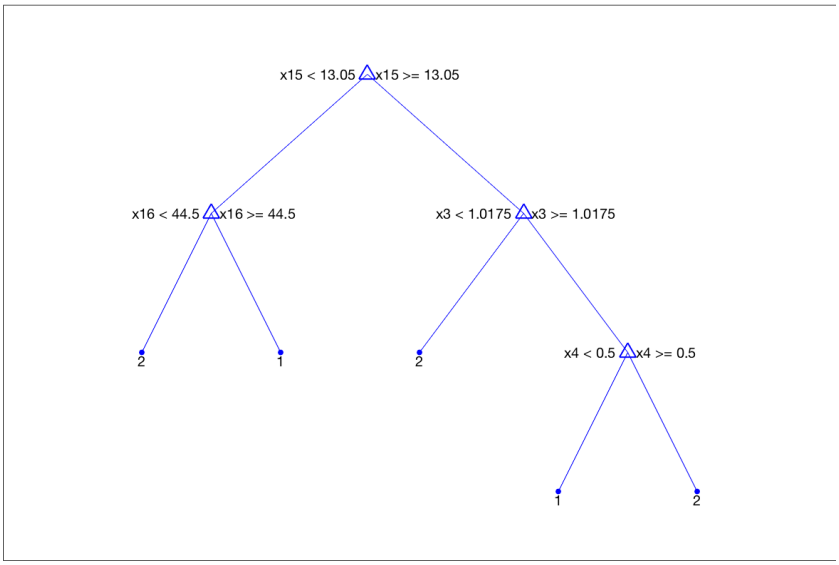

Figure 3: Decision tree

Even if PCA was not performed, the hierarchical classification happens to give better results than the agglomerative clustering, with a strike rate of 92% and both sensitivity and specificity above the 90%.

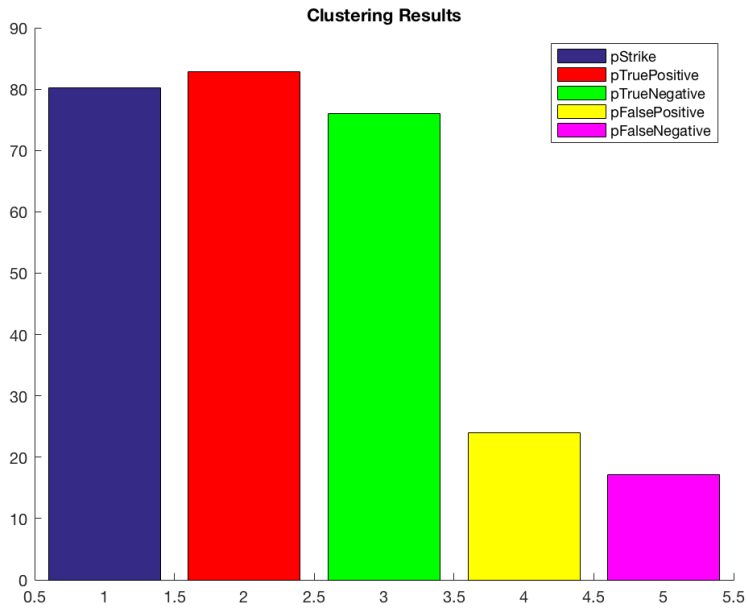

Figure 4: Performances of classification

Table of Contents

Data Preparation

```
clear all
close all
clc

load('ckd3.mat');

keylist={'normal','abnormal','present','notpresent','yes','no','good','poor','ckd'
keymap=[0,1,0,1,0,1,0,1,2,1,NaN,NaN];

ckd=chronickidneydisease;


% adjusting the format of the input data
[kR,kC] = size(ckd);
b=[];
for kr=1:kR
    for kc=1:kC
        c=strtrim(ckd(kr,kc)); % remove blanks
        check=strcmp(c,keylist);
        if sum(check)==0
            b(kr,kc)=str2num(ckd{kr,kc}); % from text to numeric
        else
            ii=find(check==1);
            b(kr,kc)=keymap(ii); % use the lists
        end
    end
end

b=b(:,1:end-1); % I mistakenly considered an empty column in the
 import phase

[N,F]=size(b);

X=b(:,1:end-1); % we don't consider the last column, that stores
 classification evaluated by the doctor
classes=b(:,end);

n_healthy=sum(classes==1);
n_ill=sum(classes==2);
```

Hierachical Clustering

```
D=pdist(X); % algorithm that evaluates the distance between each
```

```
            % measurement stored in the matrix
D_matrix=squareform(D); % the output of pdist is a row vector.
 'squareform'
                % transforms it into a matrix of distances d(i,j)

Z=linkage(D); %

K=2; % selecting the number of clusters we want to consider

    % we create the hierachical tree
T=cluster(Z,'maxclust',K);

p=0;
figure
dendrogram(Z,p)

% we compare the clustering with the classification given by doctors
perc_true=100*length(find(T==classes))/N; % 0.8025
perc_false=length(find(T~=classes))/N; % 0.1975 (1-perc_true)

n_false_negative=length(find(classes(T==1)==2));
n_false_positive=length(find(classes(T==2)==1));
n_true_negative=length(find(classes(T==1)==1));
n_true_positive=length(find(classes(T==2)==2));

p_true_positive=100*n_true_positive/n_ill; % 87.92
p_true_negative=100*n_true_negative/n_healthy; % 93.87
p_false_positive=100*n_false_positive/n_healthy; % 6.12
p_false_negative=100*n_false_negative/n_ill; % 12.07

figure
hold on
b=bar(1,perc_true);
b2=bar(2,p_true_positive,'r');
b3=bar(3,p_true_negative,'g');
b4=bar(4,p_false_positive,'y');
b5=bar(5,p_false_negative,'m');

title('Clustering Results')
legend('pStrike','pTruePositive','pTrueNegative','pFalsePositive','pFalseNegative'


% trying to evaluate the sum of the squared error, that measures the
% performance of the clustering algorithm
w1=X(find(T==1),:);
w2=X(find(T==2),:);

m_k(1,:)=mean(w1,1);
m_k(2,:)=nanmean(w2,1);

SSE_1=0;
SSE_2=0;
for i=1:size(w1,1)
    error_1(i)=norm(w1(i,:)-m_k(1,:)).^2;
```

```
    SSE_1=SSE_1+error_1(i);
end

for i=1:size(w2,1)
    error_2(i)=norm(w2(i,:)-m_k(2,:)).^2;
    SSE_2=SSE_2+error_2(i);
end

SSE=SSE_1+SSE_2;
```

Classification Tree

```
% R_x=X'*X/N;
% [U, E] = eig(R_x);
%
% P = sum(diag(E));
% percentage = 0.999; % we set the percentage of information that we
 want to keep
% new_P = percentage * P;
%
% cumulative_P = cumsum(diag(E)); % function that evaluates the
 cumulative
%                                 % sum of each element of the
 diagonal of A
% L = length(find(cumulative_P<new_P)); % determines the first L
 features
%                                 % that contribut to obtain new_P
 amount
%                                 % of "information"
%
% U_L = U(:,1:L); % we only consider the first L features
%
% Z = X * U_L;

tc=fitctree(X,classes); % function that generates the classification
 tree

view(tc); % decision tree explained in command window
view(tc,'Mode','graph'); % graphical representation of the decision
 tree

% implementation of the decision tree
for i=1:N
    if X(i,15)<13.05
        if X(i,16)<44.5
            ct_classes(i)=2;
        else
            ct_classes(i)=1;
        end
    else
        if X(i,3)<1.0175
            ct_classes(i)=2;
        else
```

```
            if X(i,4)<0.5
                ct_classes(i)=1;
            else
                ct_classes(i)=2;
            end
        end
    end
end

ct_classes=ct_classes';

perc_true_ct=100*length(find(ct_classes==classes))/N; % 0.9275
perc_false_ct=length(find(ct_classes~=classes))/N; % 0.0725 (1-
perc_true)

n_false_negative_ct=length(find(classes(ct_classes==1)==2));
n_false_positive_ct=length(find(classes(ct_classes==2)==1));
n_true_negative_ct=length(find(classes(ct_classes==1)==1));
n_true_positive_ct=length(find(classes(ct_classes==2)==2));

p_true_positive_ct=100*n_true_positive_ct/n_ill; % 90.40
p_true_negative_ct=100*n_true_negative_ct/n_healthy; % 96.66
p_false_positive_ct=100*n_false_positive_ct/n_healthy; % 3.33
p_false_negative_ct=100*n_false_negative_ct/n_ill; % 9.60

figure
hold on
b=bar(1,perc_true_ct);
b2=bar(2,p_true_positive_ct,'r');
b3=bar(3,p_true_negative_ct,'g');
b4=bar(4,p_false_positive_ct,'y');
b5=bar(5,p_false_negative_ct,'m');

title('Classification Results')
legend('pStrike','pTruePositive','pTrueNegative','pFalsePositive','pFalseNegative'
```

Published with MATLAB® R2016a