

# Math 444: Final Project

Iris Zhang

April 26, 2020

## Introduction

For this project, we use the seeds dataset from UCI Machine Learning repository. In the dataset, we have three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each. For each sample, seven attributes – area, perimeter, compactness, length, width, asymmetry coefficient, length of kernel groove – are measured. Here since the elements in the dataset are randomly selected, we use first 35 elements for each wheat to be the training set and the remaining will be considered as the testing set. We will develop tree classifier and use LDA classifier that we developed in midterm project to classify each data point and compare the performances of two classifiers.

## Tree classifier

### Concepts

Let  $R$  be a rectangle that represents a data space. Define  $n(R)$  be the number of data points in  $R$  and  $n_j(R)$  be the number of data points in  $R$  that belongs to the class  $j$ . For this dataset, we have three classes, so  $1 \leq j \leq 3$ . Then, we have  $n(R) = \sum_{j=1}^3 n_j(R)$ .

The frequencies (probabilities) of each class appearing in  $R$  is calculated by  $p_j(R) = \frac{n_j(R)}{n(R)}$ .

The majority class is defined by  $c(R) = \operatorname{argmax} \{p_j(R) | 1 \leq j \leq 3\}$ , which is the class representing the majority in  $R$ . We will assign every vector in that rectangle  $R$  to the class  $c(R)$ .

To quantitatively describe the impurity, we have the following measurements:

The misclassification error will be calculated as  $m(R) = \sum_{j \neq c(R)} p_j(R) = 1 - p_{c(R)}(R)$

The Gini index is defined as  $g(R) = \sum_{j=1}^3 p_j(R)(1 - p_j(R)) = 1 - \sum_{j=1}^3 p_j(R)^2$

The impurity change is defined as  $\Delta g = g(R) - \frac{n(R_L)}{n(R)}g(R_L) - \frac{n(R_R)}{n(R)}g(R_R)$

To split a given rectangle, we will need to choose a split index  $j$  and a split value  $s$ . We will compute the optimal split value  $s_i$  for each attribute  $i$  by solving  $s_i = \operatorname{argmax} \{\Delta g_i(s)\}$ ,  $\Delta g_i^* = \Delta g_i(s_i)$ . Then, we can choose the optimal split index  $i^*$  such that  $i^* = \operatorname{argmax} \{\Delta g_i^* | 1 \leq i \leq n\}$ . So, the optimal split value  $s^*$  can be obtained by  $s^* = s_{i^*}$ .

Then, we can obtain  $R = R_L \cup R_R$  where  $R_L = \{x \in \mathbb{R} : x_j \leq s\}$  and  $R_R = \{x \in \mathbb{R} : x_j > s\}$ .

A leaf is a rectangle that has no children. The leaves are classified to two types: pure leaves and mixed leaves. A leaf is a pure leaf if  $g(R) = 0$  and a leaf is a mixed leaf if  $g(R) > 0$ .

# Helper Functions

In order to build our tree classifier, we have some helper functions.

## ClassDistr

This function is used to calculate the frequencies  $p_j(R)$ , majority class  $c(R)$ , and misclassification error  $m(R)$  for a given rectangle  $R$ .

The Matlab code is attached in Appendix I.

## Splitting

This function is used to determine the optimal splitting rectangles by calculating the optimal split value  $s$  and optimal split index  $j$ .

As described above, we will compute optimal split value  $s_i$  for each attribute. So, first we need to sort the data points in the given rectangle for each coordinate as  $x_i^{j_1} \leq x_i^{j_2} \leq \dots \leq x_i^{j_q}$ . Then, the discrete split values producing different splits are  $\sigma_t = \frac{1}{2}(x_i^{j_t} + x_i^{j_{t+1}})$  where  $1 \leq t \leq q - 1$ . So,  $s_i = \min \{\sigma_t\}$ . The Matlab code is attached in Appendix II.

## Cost\_complexity

This function is used to calculate the misclassification cost of the rectangle weighted by size, which is measured to determine the effectiveness of classification of a node. The Matlab code is included below.

```
function [cost] = Cost_complexity(R,n,C)
weight = length(R.I)/n;
[~,~,r] = ClassDistr(C,R.I);
cost = weight*r;
end
```

# Growing a Tree

## Initialization

We let  $R_0$  be the rectangle that contains all the data points. Also, we assign the set of pure leaves  $\mathcal{R}_p = \emptyset$  and the set of mixed leaves  $\mathcal{R}_m = \{R_0\}$ .

## Iteration

We pick a rectangle  $R \in \mathcal{R}_m$  and apply Splitting function to calculate the optimal split value and index so that we have  $R = R_L \cup R_R$ . Then, we assign new leaves and check whether the leaves are pure or mixed. The Matlab codes for determining which type the new (left) leaf is are shown below where the `r_left` is the misclassification error calculated by applying `ClassDistr` function.

```

if r_left ==0
    % pure leaf
    PureNodes = [PureNodes, count+1];
else
    % mixed leaf
    MixedNodes = [MixedNodes, count+1];
end

```

Then, we remove the rectangle  $R$  from  $\mathcal{R}_m$  and update the number of nodes in the current tree. We repeat this procedure until  $\mathcal{R}_m = \emptyset$ .

## Pruning a Tree

Since the  $T_{max}$  we obtained in previous step might be too large and hence have the tendency to misclassify, we need to prune the tree. Pruning a tree is done by deleting the children of a non-leaf node and making it a leaf. After growing the tree, all the leaves are pure and all non-leaf nodes are mixed. Overall, we need to balance the misclassification cost and the complexity in the pruning procedure. In other words, we need to minimize the cost-complexity function, which is defined by The cost complexity is defined by  $\rho_\alpha(T) = \rho(T) + \alpha|T|$ . We attached the Matlab codes for growing a tree in Appendix III and pruning a tree in Appendix IV.

## Initialization

We create a cell  $T$  to store all the pruned trees and set  $T_0 = T_{max}$ , and let  $k = 0$ .

## Iteration

We compute  $\alpha(\bar{R})$  for all non-leaf nodes of tree  $T_k$  by  $\alpha(\bar{R}) = \frac{v(\bar{R})r(\bar{R}) - \sum_{\mathcal{L}'} v(R)r(R)}{|T'| - 1}$  where  $v(R) = \frac{n(R)}{n(R_0)}$ ,  $\mathcal{L}' = \mathcal{L}(T')$  = the leaves of the subtree with root  $\bar{R}$ ,  $|T'|$  = the number of leaves of the subtree with root  $\bar{R}$ .

Then, we will have  $\alpha_{k+1} = \min \{\alpha(\bar{R})\}$  so that we can prune the tree  $T_k$  at nodes corresponding to  $\alpha_{k+1}$  and we can get the pruned tree  $T_{k+1}$ . We repeat this procedure until  $T_k = \{R_0\}$ .

During each iteration, before pruning, we first have a successor matrix  $A$  and a genealogy matrix  $G$  to indicate whether one node is a successor of another node. The matrix  $A$  is defined by  $A_{k,j} = 1$  if and only if  $R(j)$  is a child of  $R(k)$  and matrix  $G$  is defined by  $G_{k,j} = 1$  if and only if  $R(j)$  is a successor of  $R(k)$  where  $G = A + A^2 + \dots + A^L$  that gives the list of nodes that need to be removed.

## Results

Now we have all pruned trees stored in cell  $T$  and we use specificity to determine the optimal pruned trees. By using 105 training data points and 105 testing data points, we have  $T_{max}$  as Figure 1. The first column is the frequencies of each class appearing in each rectangle while the number of data points in each rectangle can be found in the last column.

R × specificity × T					
1x17 struct with 6 fields					
...	p	j	s	left	right
1	[0.3333,0.3333,0.3333]	1	12.7100	2	3 1x105 do...
2	[0.0303,0,0.9697]	6	1.8080	4	5 1x33 dou...
3	[0.4722,0.4861,0.0417]	7	5.3775	6	7 1x72 dou...
4	[1,0,0]	NaN	NaN	NaN	NaN 24
5	[0,0,1]	NaN	NaN	NaN	NaN 1x32 dou...
6	[0.9697,0,0.0303]	6	5.6145	8	9 1x33 dou...
7	[0.0513,0.8974,0.0513]	1	14.8650	10	11 1x39 dou...
8	[1,0,0]	NaN	NaN	NaN	NaN 1x32 dou...
9	[0,0,1]	NaN	NaN	NaN	NaN 143
10	[0,0,1]	NaN	NaN	NaN	NaN [141,142]
11	[0.0541,0.9459,0]	7	5.5755	12	13 1x37 dou...
12	[1,0,0]	NaN	NaN	NaN	NaN 10
13	[0.0278,0.9722,0]	1	16.7000	14	15 1x36 dou...
14	[0.3333,0.6667,0]	1	16.5800	16	17 [9,81,101]
15	[0,1,0]	NaN	NaN	NaN	NaN 1x33 dou...
16	[0,1,0]	NaN	NaN	NaN	NaN [81,101]
17	[1,0,0]	NaN	NaN	NaN	NaN 9
18					

Figure 1:  $T_{max}$  composed of 17 nodes

To be more clear, we construct the tree in Figure 2 according to the nodes in Figure 1.

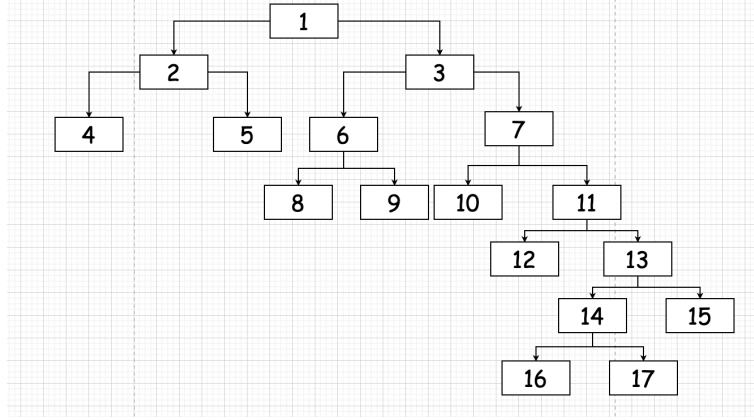


Figure 2: Construction of tree (the number  $i$  in the graph indicate the rectangle  $R_i$ ,  $1 \leq i \leq 17$ )

It turns out that attributes  $x_1$  (area),  $x_6$  (asymmetry coefficient), and  $x_7$  (length of kernel groove) are more informative since all the splits are done in those three coordinates. To illustrate the split as rectangles and also visualize the performance, we plot projections of the splits in the  $(x_1, x_6)$  plane and the splits in the  $(x_1, x_7)$  plane and, see Figure 3 and Figure 4, respectively,

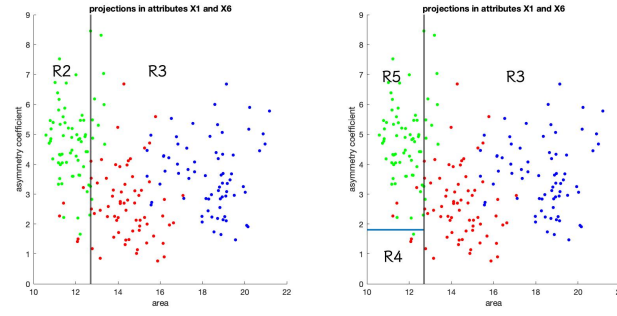


Figure 3: projections in  $(x_1, x_6)$  plane

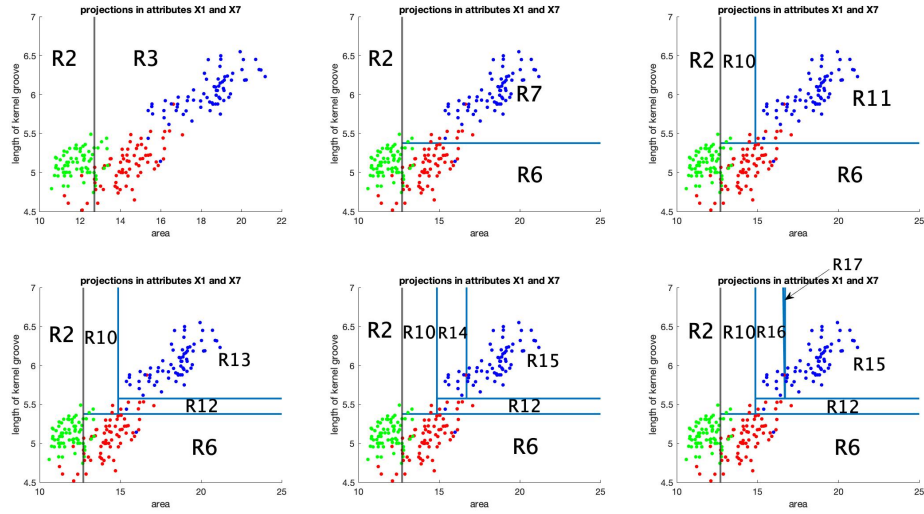


Figure 4: projections in  $(x_1, x_7)$  plane

Then, we get eight pruned trees by deleting children, see Figure 5.  $T\{1\}$  is obtained by pruning two children of  $T_{max}$  that we get in Figure 1, and so on.

	1	2	3	4	5	6	7	8
1	1x15 str...	1x13 str...	1x11 str...	1x9 struct	1x7 struct	1x5 struct	1x3 struct	1x1 struct
2								
3								
4								

Figure 5: Pruned trees

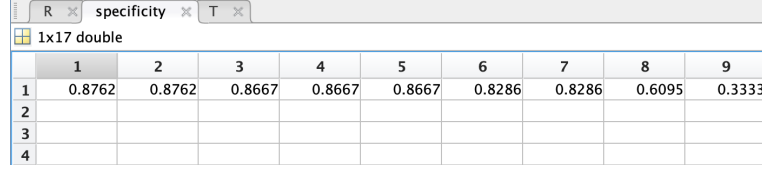
Then, we calculate the specificity values of the testing data points classified by each tree, see Figure 6. A snippet of Matlab code of computing specificity values is shown below:

```
specificity=zeros(1,length(T));
for i=1:length(T)
    num_correct=0;
    for j=1:length(I_test)
```

```

        if classify(i,j)==C(I_test(j))
            num_correct=num_correct+1;
        end
    end
    specificity(i)=num_correct/length(I_test);
end

```



	1	2	3	4	5	6	7	8	9
1	0.8762	0.8762	0.8667	0.8667	0.8667	0.8286	0.8286	0.6095	0.3333
2									
3									
4									

Figure 6: specificity values

The first value is the specificity of  $T_{max}$  and the remaining are the specificity values of the pruned trees in the order of the trees shown in Figure 5. We can see that the pruning procedure does not improve the performance of the classifier. This is because there is almost no overfitting problem after we grow the tree. So, the improvement after some "effective" pruning will be small. And if we over-pruning the tree, the specificity value will decrease.

## LDA classifier

We will apply the LDA classifier here to the seeds dataset with the same training and testing dataset as the ones used in tree classifier. The snippet of my Matlab codes is attached below.

```

[V,D]= LDA(training_data, training_label, 3);
Z=cell(1,3);
for i=1:3
    Z{i}=V'*training_data(:,training_label==(i));
end
c=cell(1,3);
for i=1:3
    c{i}=(1/size(Z{i},2)) * sum(Z{i},2);
end
I_lda=zeros(1,105);
for i=1:105
    I_lda(i)=LDA_classifier(testing_data(:,i), V', c, 2);
end

```

Similar, in order to compare those two classifiers we developed, we also calculate the specificity value for LDA classifier and we obtain that the value is 0.6667 for LDA classifier.

## Conclusion

Comparing to the highest specificity value we have from tree classifier, the one for  $T_{max}$ , which is 0.8762, with the specificity value for LDA classifier, which is 0.6667, we can see that the performance of tree classifier is much better than the LDA classifier.

However, even the tree classifier cannot perfectly classify the data points. Here we want to explore whether it is the problem of our algorithm or it is due to the dataset itself. So, we do the LDA separator on the original whole dataset with labels and we get the scatter plot and histogram of the first two directions, see Figure 7 and 8.

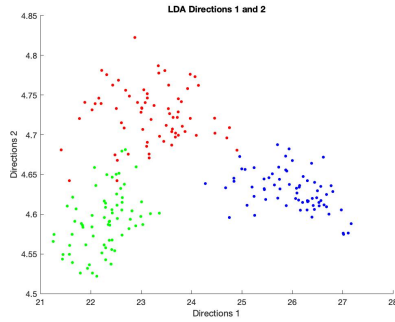


Figure 7: scatter plot of LDA direction 1 and 2

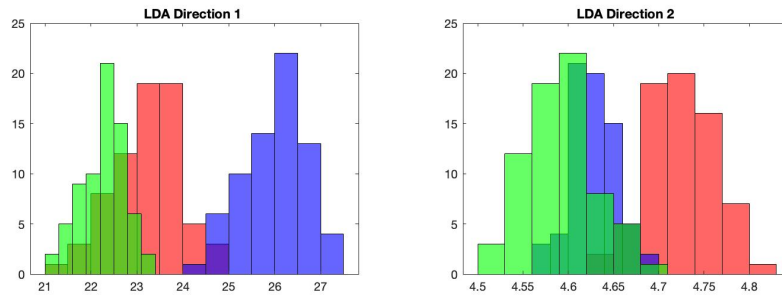


Figure 8: histogram of first two directions of LDA

We can see from the histogram plot that there are clusters but not completely separately. With the specificity value we have for the tree classifier, we believe that the classifier is doing relatively good based on the nature of this dataset.

## Appendix I : ClassDistr

```
function [p,c,r] = ClassDistr(I,C)
% number of data points in R
n = size(C,2);
% most frequent value
c = mode(I(C));
class = unique(I);
k = size(class,2);
p = zeros(1,k);
for i = 1:k
    % number of data points in R that belongs to class i
    n_i = size(find(I(C)==class(i)),2);
    p(i) = n_i/n;
end
% misclassification error
r = 1- p(c);
end
```



## Appendix II : Splitting

```
function [split_value,split_index] = Splitting(I,C,X)
n = size(X,1);
m_j = zeros(2,n);
for j = 1:n
    [x,~,~] = unique(X(j,I));
    [x_sorted,~] = sort(x,'ascend');
    p = size(x,2);
    s = zeros(1,p-1);
    m_s = zeros(1,p-1);
    for i = 1:p-1
        % median between X(i) and X(i+1)
        s(i) = 0.5*(x_sorted(i)+x_sorted(i+1));
        % left and right points of the split value
        I_left = I(find(X(j,I) <= s(i)));
        I_right = I(find(X(j,I) > s(i)));
        c_left = C(I_left);
        c_right = C(I_right);
        cmean_left = sum(c_left)/length(c_left);
        cmean_right = sum(c_right)/length(c_right);
        % mismatch of all X
        m_s(i) = sum((c_left-cmean_left).^2) + sum((c_right-cmean_right).^2);
    end
    if isempty(s)
        s = x_sorted(1);
        c = sum(C(I))/length(C(I));
        m_s(1) = sum((C(I)-c).^2);
    end
    % minimize the mismatch for each dimension
    [m_val,index]=min(m_s);
    s_star = s(index);
    m_j(:,j) = [m_val;s_star];
end
% find the best j and s
[~,split_index] = min(m_j(1,:));
split_value = m_j(2,split_index);

end
```

## Appendix III : Growing a Tree

```
I = [1:35, 71:105, 141:175];
[n,l] = size(I);
R(1).I=I);
[p,c,r] = classproperty(C,R(1).I);
% initialization
R(1).p = p;
R(1).j = NaN;
R(1).s = NaN;
R(1).left = NaN;
R(1).right = NaN;
PureNodes = [];
MixedNodes =[1];
count = 1;
% iteration
while length(MixedNodes)>0
    % pick a rectangle and find the optimal splitting
    index = MixedNodes(1);
    I_index = R(index).I;
    X_index = seedsdata(:,I_index);
    [s,j] = splitting(I_index,C,seedsdata);
    % assign new leaves
    R(index).j = j;
    R(index).s = s;
    R(index).left = count+1;
    R(index).right = count+2;
    I_left = I_index(find(X_index(j,:)<=s));
    [p_left, c_left, r_left] = classproperty(C,I_left);
    R(count+1).I= I_left;
    R(count+1).p = p_left;
    R(count+1).j = NaN;
    R(count+1).s = NaN;
    R(count+1).left = NaN;
    R(count+1).right = NaN;
    % check if the new leaves are pure or mixed
    if r_left ==0
        % pure leaf
        PureNodes = [PureNodes, count+1];
    else
        % mixed leaf
        MixedNodes = [MixedNodes, count+1];
    end
    I_right = setdiff(I_index,I_left);
    [p_right,c_right,r_right]= classproperty(C,I_right);
    R(count+2).I= I_right;
    R(count+2).p = p_right;
    R(count+2).j = NaN;
    R(count+2).s = NaN;
    R(count+2).left = NaN;
    R(count+2).right = NaN;
    % check if the new leaves are pure or mixed
    if r_right ==0
```

```

        % pure leaf
        PureNodes = [PureNodes, count+2];
    else
        % mixed leaf
        MixedNodes = [MixedNodes, count+2];
    end
    % remove the rectangle just split from the leaf first
    MixedNodes = MixedNodes(2:end);
    count = count+2; % number of nodes in the current tree
end

```

## Appendix IV : Pruning a Tree

```
% initialization
R_current = R;
nodes_num= length(R_current);
T = cell(1,nodes_num);
count = 1;
alpha_s = zeros(1,nodes_num);

while length(R_current)~=1
    nodes_num= length(R_current);
    % define successor matrix A and genealogy matrix G
    A = zeros(nodes_num,nodes_num);
    G = zeros(nodes_num,nodes_num);
    for k = 1:nodes_num
        i = R_current(k).left;
        j = R_current(k).right;
        if ~isnan(i)
            % the node k is not a leaf
            A(k,i) = 1;
            A(k,j) = 1;
        end
    end
    G = A;
    A_next = A;
    ite = 1;
    while (norm(A_next,'fro')>0)
        % keep iterating untill the product vanishes
        A_next = A*A_next;
        G = G+A_next;
        ite = ite +1;
    end
    I_leaf = [];
    for k = 1:length(R_current)
        if sum(G(k,:))==0
            I_leaf = [I_leaf k];
        end
    end
    alpha = zeros(1,nodes_num);
    % compute alpha for all non-leaf nodes
    for i =1:nodes_num
        if sum(G(i,:))==0
            alpha(i)=inf;
        else
            % identify leaves below node i
            Ii = find(G(i,:)==1);
            leaf_j = intersect(Ii,I_leaf);
            leaf_n = length(leaf_j);
            % compute misclassification error or each leaf
            mis_node = cost_complexity(R_current(i),size(I,2),C);
            mis_leaf = 0;
            for k = 1:leaf_n
                mis_leaf = mis_leaf + cost_complexity(R_current(k),size(I,2),C);
            end
        end
    end
end
```

```

        end
        alpha(i) = (mis_node-mis_leaf)/(leaf_n-1);
    end
end
% find minimum alpha
[a_min,j_min] = min(alpha);
child = find(G(j_min,:)==1);
% prune tree with minimum alpha
R_current(j_min).left = NaN;
R_current(j_min).right = NaN;
for c=length(child):-1:1
    R_current=R_current([1:child(c)-1,child(c)+1:length(R_current)]);
end
for k = 1:length(R_current)
    for m = length(child):-1:1
        if R_current(k).left>child(m)
            R_current(k).left= R_current(k).left-1;
        end
        if R_current(k).right>child(m)
            R_current(k).right= R_current(k).right-1;
        end
    end
end
T{count} = R_current;
alpha_s(count)= a_min;
count=count+1;
end
end

```