北京航空航天大學
BEIHANG UNIVERSITY

Pattern Recognition and Machine Learning Experiment Report

院（系）名称	自动化科学与电气工程学院
专业名称	模式识别与智能系统
学生姓名	邓颖
学号	12031156
任课老师	秦曾昌

2015 年 5 月 25 日

3 Decision Tree Learning for Classification

3.1 Introduction

Decision tree induction is one of the simplest and yet most successful learning algorithms. A decision tree (DT) consists of internal and external nodes and the interconnections between nodes are called branches of the tree. An internal node is a decision-making unit to decide which child nodes to visit next depending on different possible values of associated variables. In contrast, an external node also known as a leaf node, is the terminated node of a branch. It has no child nodes and is associated with a class label that describes the given data. A decision tree is a set of rules in a tree structure, each branch of which can be interpreted as a decision rule associated with nodes visited along this branch.

3.2 Principle and Theory

Decision trees classify instances by sorting them down the tree from root to leaf nodes. This tree-structured classifier partitions the input space of the data set recursively into mutually exclusive spaces. Following this structure, each training data is identified as belonging to a certain subspace, which is assigned a label, a value, or an action to characterize its data points. The decision tree mechanism has good transparency in

that we can follow a tree structure easily in order to explain how a decision is made. Thus interpretability is enhanced when we clarify the conditional rules characterizing the tree.

Entropy of a random variable is the average amount of information generated by observing itsvalue. Consider the random experiment of tossing a coin with probability of heads equal to 0.9, so that P(Head) = 0.9 and P(Tail) = 0.1. This provides more information than the case where P(Head) = 0.5 and P(Tail) = 0.5. Entropy is used to evaluate randomness in physics, where a large entropy value indicates that the process is very random. The decision tree is guided heuristically according to the information content of each attribute. Entropy is used to evaluate the information of each attribute; as a means of classification. Suppose we have m classes, for a particular attribute, we denoted it by pi by the proportion of data which belongs to class Ci where i = 1, 2, … m.

The entropy of this attribute is then:

$$\text{Entropy} = \sum_{i=1}^{m} -p_i \cdot log_2 p_i$$

We can also say that entropy is a measurement of the impurity in a collection of training examples: larger the entropy, the more impure the data is. Based on entropy, information Gain (IG) is used to measure the effectiveness of an attribute as a means of discriminating between classes.

$$\text{IG}(\text{S}, \text{A}) = \text{Entropy}(\text{S}) - \sum_{v \in Value(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where all examples *S* is divided into several groups (i.e. S_v for $v \in Values(\text{A})$) according to the value of *A*. It is simply the expected reduction of entropy caused by partitioning the examples according to this attribute.

3.3 Objective

The goals of the experiment are as follows:

(1) To understand why we use entropy-based measure for constructing a decision tree.

(2) To understand how Information Gain is used to select attributes in the process of building a decision tree.

(3) To understand the equivalence of a decsion tree to a set of rules.

(4) To understand why we need to prune the tree sometimes and how can we prune? Based on what mesure we prune a decsion tree.

(5) To understand the concept of Soft Decsion Treees and why they are important extensions to classical decision trees.

3.4 Contents

Stage 1:

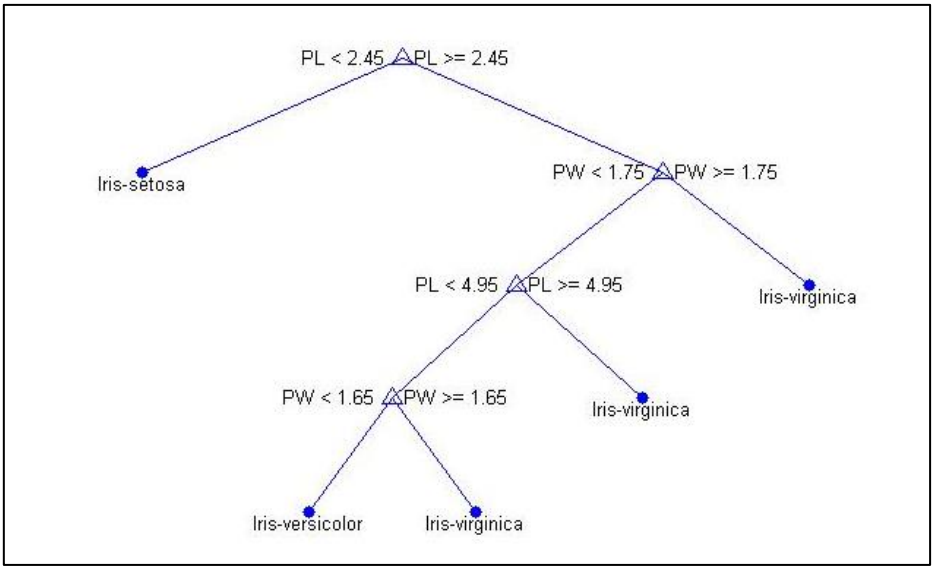
(1) The information entropy is calculated by the formula
$\text{Entropy} = \sum_{\text{i}=1}^{\text{m}} -\text{p}_\text{i} \cdot \log_2 \text{p}_\text{i}$.

(2) The Iris Dataset has four attributes: SL,SW,PL,PW
and three classes: Iris-setosa(class1), Iris-versicolor(class2), Iris-virginica(class3)
The most informative attribute is the one with the largest information gain. By the implementing the code, I get the nodes as below:

root	edge1	node1	edge2	node3	edge3	node4	edge4	leaf
PL	≤ 1.9	class1						
	≥ 3.0	PW	≥ 1.8	class3				
			≤ 1.7	PL	≥ 5.0	class2		
					≤ 4.9	PW	≤ 1.6	class2
							≥ 1.7	class3

The program codes are attached to the end of this report.

(3) To test the reliability of the self-written code, I run the instructions in the matlab's toolbox to build the following decision tree:



By comparing the tree chart with the picture, it can be found that the attributes and thresholds of the nodes in two trees are basically consistent. So we can roughly conclude that the program codes are in line with the question.

Stage 2:

(1) My thought to discretize the datasets with continuous attributes or mixed attributes: First, sort each attribute separately in order; second, find the turning points where the classes change in the new array. Then work out the corresponding information gains. At last find the largest value, the matching turning point is the threshold to split the attribute values.

After a reference to some papers, I attain a more professional approach to deal with this problem, and it`s concluded as follows: First, define a space of discretization models. The parameters of a specific discretization are the number

of intervals, the bounds of the intervals and the class frequencies in each interval. Then, we define a prior distribution on this model space. Finally, we derive an evaluation criterion of discretizations, which is a direct application of the Bayesian approach for the discretization model space and its prior distribution.

(2) A typical way to ensure both compactness and performance of a decision tree is to prune it. That means growing the tree until each node contains a small number of instances, then use pruning to remove nodes that do not provide additional information. And the pruning can be measured by many parameters such as reduced error and cost complexity, each of which can determine an optimal tree. Taking the reduced error pruning for example, it's starting at the leaves while each node is replaced with its most popular class. If the prediction accuracy is not affected then the change is kept.

(3) In the classical decision tree each sample belongs totally to one class, but a soft approach enables a sample to be seen as several portions. Samples outside the boundary zones are treated as normal while the others are handled with soft classification. For the ones in boundary zones, the probability density of a sample become the predictor variable and the known class proportions of it become the target variable. With this method we can construct a tree using the dataset and the result is what we want.

(4) Compare to the Naïve Bayes, the advantages of the decision tree learning:

i. It's simple to understand and interpret;
ii. It has value even with little hard data;
iii. It allows the addition of new possible scenarios;
iv. It's flexible to be combined with other decision techniques.

The disadvantages:

i. The classifying results are easily to be influenced by external disturbance;
ii. Calculations can get very complex particularly if many values are uncertain or if many outcomes are linked;
iii. For data including categorical variables with different number of levels, information gain in decision trees has preference for those attributes with more levels.

3.5 Experience

Through this experiment I get a deeper hold of the theory and implementation of decision tree, at the time grasp the differences between the use of ID3 and C4.5. The main problem came up in the procedure of writing the codes, hence costing me a lot of time. But finally it was overcame and I learn to use struct in matlab. What's more, by referring to some material I gain a rough knowledge of several methods to discretize datasets and prune the tree.

3.6 Codes

```
%maintree
clear all;
clc;
[S1,S2,S3,S4,type]=textread('data.txt','%f,%f,%f,%f,%d');
global T index;
index=0;
data=[S1,S2,S3,S4];
propertyName={'sl','sw','pl','pw'};
decisionTreeModel=decisionTree(data,type,propertyName);

%计算信息熵
function infentropy=CEntropy(property)
   infentropy=0;
   totalLength=length(property);
   itemList=unique(property);
   pNum=length(itemList);
   for i=1:pNum
       itemLength=length(find(property==itemList(i)));
       pItem=itemLength/totalLength;
       infentropy=infentropy-pItem*log2(pItem);
   end
end

%建立决策树模型
function decisionTreeModel=decisionTree(data,type,propertyName)
   global rootNode;  %定义全局变量
   global Node;

   rootNode=struct('NodeName',[]);
   Node=struct('fatherNodeName',[],'EdgeProperty',[],'NodeName',[]);
   rootIndex=CalcuteNode(data,type);
   dataRowIndex=setdiff(1:length(propertyName),rootIndex);
   rootNode.NodeName=propertyName(rootIndex);
```

```
    propertyName(rootIndex)=[];
    rootData=data(:,rootIndex);
    sonEdge=unique(rootData);

    for i=1:length(sonEdge)
    edgeDataIndex=find(rootData==sonEdge(i));
    BuildTree(rootNode.NodeName,sonEdge(i),data(edgeDataIndex,dataRowI
    ndex),type(edgeDataIndex,:),propertyName);
    end

    model.rootNode=rootNode;
    model.Node=Node;
    decisionTreeModel=model;
end

%返回最大信息增益对应的属性号
function [NodeIndex]=CalcuteNode(data,type)
    LargeEntropy=CEntropy(type);
    [m,n]=size(data);
    EntropyGain=LargeEntropy*ones(1,n);

    for i=1:n
        pData=data(:,i);
        itemList=unique(pData);
        for j=1:length(itemList)

    itemIndex=find(pData==itemList(j));
    EntropyGain(i)=EntropyGain(i)-length(itemIndex)/m*CEntropy(type(it
    emIndex));
        end
    end
    [~,NodeIndex]=max(EntropyGain);
end

%确定决策树的枝干
function BuildTree(fatherNodeName,edge,data,label,propertyName)
    global Node;

    k=length(Node)+1;
    Node(k).fatherNodeName=fatherNodeName;
    Node(k).EdgeProperty=edge;
    if length(unique(label))==1
        Node(k).NodeName=label(1);
        return;
    end

    sonIndex=CalcuteNode(data,label);
```

```
    dataRowIndex=setdiff(1:length(propertyName),sonIndex);
    Node(k).NodeName=propertyName(sonIndex);
    propertyName(sonIndex)=[];
    sonData=data(:,sonIndex);
    sonEdge=unique(sonData);

    for i=1:length(sonEdge)
    edgeDataIndex=find(sonData==sonEdge(i));

    BuildTree(Node(k).NodeName,sonEdge(i),data(edgeDataIndex,dataRowIn
    dex),label(edgeDataIndex,:),propertyName);
    end
end
```