Using Decision Trees for Classification

Dr. Akinul Islam Jony

Assistant Professor

Dept. of Computer Science
Faculty of Science & Technology
American International University – Bangladesh

akinul@aiub.edu

Outline

- Introduction to Decision Rules and Decision Trees
- The TDIDT Algorithm
- Types of Reasoning: Deduction, Abduction and Induction

Objectives & Outcomes

- To get familiar with Decision Rules and Decision Trees
- To understand the TDIDT (Top-Down Induction of Decision Trees) algorithm for inducing classification rules via the intermediate representation of a decision tree
- To distinguish three types of reasoning: deduction, abduction and induction

Decision Rules and Decision Trees

- In many fields, large collections of examples, possibly collected for other purposes, are readily available.
- Automatically generating classification rules (often called *decision rules*) for such tasks has proved to be a realistic alternative to the standard Expert System approach of eliciting the rules from experts.
- In many (but not all) cases decision rules can conveniently be fitted together to form a *tree* structure.

Decision Rules and Decision Trees

- In this lecture we look at a widely-used method of constructing a model from a dataset in the form of a *decision tree* or (equivalently) a set of *decision rules*.
- It is often claimed that this representation of the data has the advantage compared with other approaches of being meaningful and easy to interpret.

Decision Trees: The Golf Example

- The Golf Example dataset is that of a golfer who decides whether or not to play each day on the basis of the weather.
- It shows the results of two weeks (14 days) of observations of weather conditions and the decision on whether or not to play.

Outlook	Temp (°F)	Humidity (%)	Windy	Class
sunny	75	70	true	play
sunny	80	90	true	don't play
sunny	85	85	false	don't play
sunny	72	95	false	don't play
sunny	69	70	false	play
overcast	72	90	true	play
overcast	83	78	false	play
overcast	64	65	true	play
overcast	81	75	false	play
rain	71	80	true	don't play
rain	65	70	true	don't play
rain	75	80	false	play
rain	68	80	false	play
rain	70	96	false	play

Data for the Golf Example

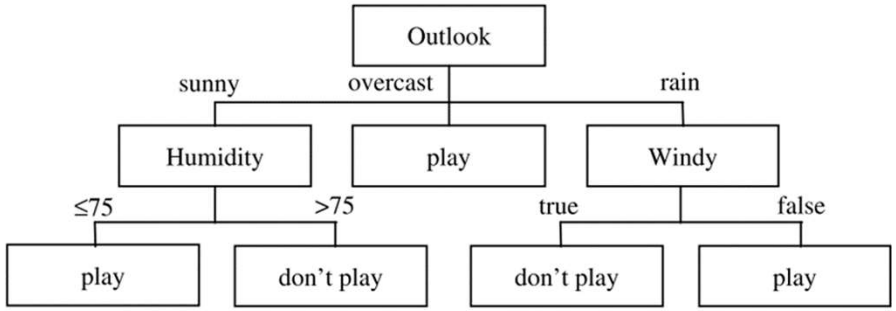
Classes
play, don't play
Outlook
sunny, overcast, rain
Temperature
numerical value
Humidity
numerical value
Windy
true, false

Decision Trees: The Golf Example

- Assuming the golfer is acting consistently, what are the rules that determine the decision whether or not to play each day?
- If tomorrow the values of

  Outlook: sunny,
  Temperature: 74°F,
  Humidity: 77%, and
  Windy: false,

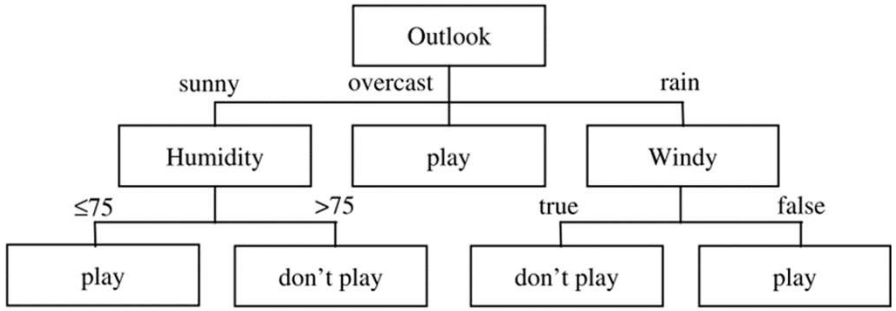
  what would the decision be?
- One way of answering this is to construct a ***decision tree*** (as shown in the figure).
- We call this decision tree (shown in the figure) a *hypothetical decision tree*.



Decision Tree for the Golf Example

Decision Trees: The Golf Example

- In order to determine the decision (classification) for a given set of weather conditions from the *decision tree*, first look at the value of ***Outlook***.
- There are three possibilities.
  - If the value of ***Outlook*** is **sunny**, next consider the value of ***Humidity***. If the value is less than or equal to **75** the decision is ***play***. Otherwise the decision is ***don't play***.
  - If the value of ***Outlook*** is **overcast**, the decision is ***play***.
  - If the value of ***Outlook*** is **rain**, next consider the value of ***Windy***. If the value is **true** the decision is ***don't play***, otherwise the decision is ***play***.
- Note that the value of *Temperature* is never used.



Decision Tree for the Golf Example

Decision Trees: Terminology

- There is a universe of ***objects*** (people, houses etc.), each of which can be described by the values of a collection of its ***attributes***.
- Attributes with a finite (and generally fairly small) set of values, such as sunny, overcast and rain, are called ***categorical***.
- Attributes with *numerical* values, such as *Temperature* and *Humidity*, are generally known as ***continuous***.
- We will distinguish between a specially-designated categorical attribute called the *classification* and the other attribute values and will generally use the term 'attributes' to refer only to the latter.

Decision Trees: Terminology

- Descriptions of a number of *objects* are held in tabular form in a ***training set***.
- Each row of the *training set* comprises an ***instance***, i.e. the (non-classifying) attribute values and the classification corresponding to one object.
- The aim is to develop ***classification rules*** from the data in the training set. This is often done in the implicit form of a ***decision tree***.
- *Training set* and *Decision tree* are equivalent in the sense that for each of the *instances* the given values of all *attributes* in the *training set* will lead to the identical *classifications*.

Decision Trees: Terminology

- A *decision tree* is created by a process known as ***splitting on the value of attributes*** (or just ***splitting on attributes***), i.e. testing the value of an attribute such as *Outlook* and then creating a branch for each of its possible values.
- In the case of *continuous* attributes the test is normally whether the value is ‘less than or equal to’ or ‘greater than’ a given value known as the ***split value***.
- The splitting process continues until each branch can be labelled with just one classification.
- *Decision trees* have two different functions: *data compression* and *prediction*.

Decision Trees: Terminology

- However, the *decision tree* is more than an equivalent representation to the *training set*.
- *Decision tree* can be used to predict the values of other instances not in the *training set*.
- It is important to stress that this 'decision' is only a prediction, which may or may not turn out to be correct. *There is no infallible way to predict the future!*
- So the *decision tree* can be viewed as not merely equivalent to the original *training set* but as a generalisation of it which can be used to predict the classification of other instances. These are often called ***unseen instances*** and a collection of them is generally known as a ***test set*** or an ***unseen test set***, by contrast with the original *training set*.

Decision Trees: Terminology

- *Decision tree* consists of a number of *branches*, each ending with a *leaf node* labelled with one of the valid classifications.
- Each branch comprises the route from the *root node* (i.e. the top of the tree) to a *leaf node*.
- A node that is neither the *root* nor a *leaf node* is called an *internal node*.

The *degrees* Dataset

- The training set of the *degrees* dataset shows the results of students for five subjects coded as *SoftEng*, *ARIN*, *HCI*, *CSA* and *Project* and their corresponding degree classifications (i.e. *Class*), which are either FIRST or SECOND.
- There are 26 instances.
- What determines who is classified as FIRST or SECOND?

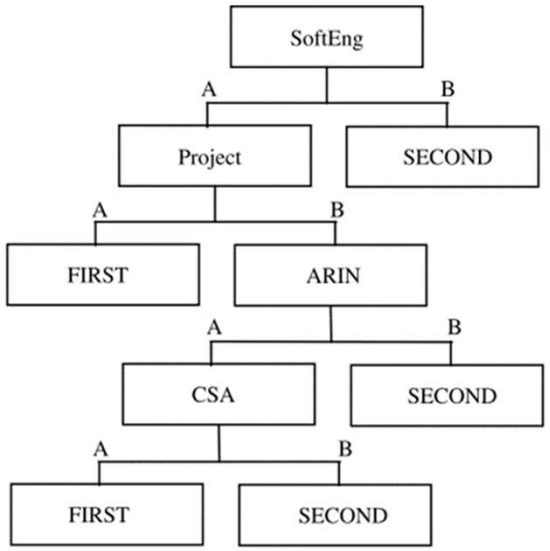
SoftEng	ARIN	HCI	CSA	Project	Class
A	B	A	B	B	SECOND
A	B	B	B	A	FIRST
A	A	A	B	B	SECOND
B	A	A	B	B	SECOND
A	A	B	B	A	FIRST
B	A	A	B	B	SECOND
A	B	B	B	B	SECOND
A	B	B	B	B	SECOND
A	A	A	A	A	FIRST
B	A	A	B	B	SECOND
B	A	A	B	B	SECOND
A	B	B	A	B	SECOND
B	B	B	B	A	SECOND
A	A	B	A	B	FIRST
B	B	B	B	A	SECOND
A	A	B	B	B	SECOND
B	B	B	B	B	SECOND
A	A	B	A	A	FIRST
B	B	B	A	A	SECOND
B	B	A	A	B	SECOND
B	B	B	B	A	SECOND
B	A	B	A	B	SECOND
A	B	B	B	A	FIRST
A	B	A	B	B	SECOND
B	A	B	B	B	SECOND
A	B	B	B	B	SECOND

Classes
FIRST, SECOND
SoftEng
A,B
ARIN
A,B
HCI
A,B
CSA
A,B
Project
A,B

The *degrees* Dataset

The *degrees* Dataset

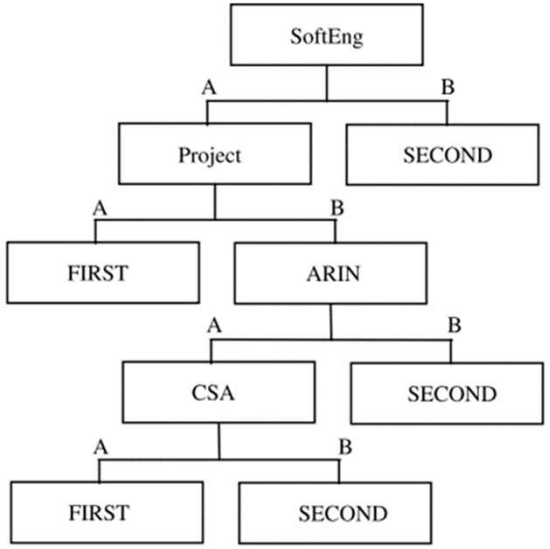
- The figure shows a possible *decision tree* corresponding to the *degrees* training set.
- We can think of the *root node* as corresponding to the original training set.
- All other *nodes* correspond to a subset of the training set.
- At the *leaf nodes* each instance in the subset has the same classification.
- There are five leaf nodes and hence five branches.



Decision Trees for the *degrees* Dataset

The *degrees* Dataset

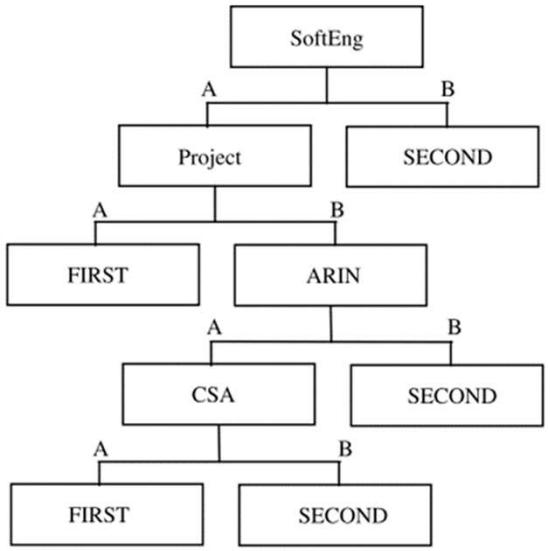
- Each *branch* corresponds to a *classification rule*. The five classification rules can be written in full as:
  - IF SoftEng = A AND Project = A
    THEN Class = FIRST
  - IF SoftEng = A AND Project = B AND ARIN = A AND CSA = A
    THEN Class = FIRST
  - IF SoftEng = A AND Project = B AND ARIN = A AND CSA = B
    THEN Class = SECOND
  - IF SoftEng = A AND Project = B AND ARIN = B
    THEN Class = SECOND
  - IF SoftEng = B
    THEN Class = SECOND



Decision Trees for the *degrees* Dataset

The *degrees* Dataset

- The order in which we write the rules generated from a decision tree is arbitrary, so the five rules could be rearranged without any change to the predictions the ruleset will make on unseen instances.
  - IF SoftEng = A AND Project = B AND ARIN = A AND CSA = B
    THEN Class = SECOND
  - IF SoftEng = B
    THEN Class = SECOND
  - IF SoftEng = A AND Project = A
    THEN Class = FIRST
  - IF SoftEng = A AND Project = B AND ARIN = B
    THEN Class = SECOND
  - IF SoftEng = A AND Project = B AND ARIN = A AND CSA = A
    THEN Class = FIRST



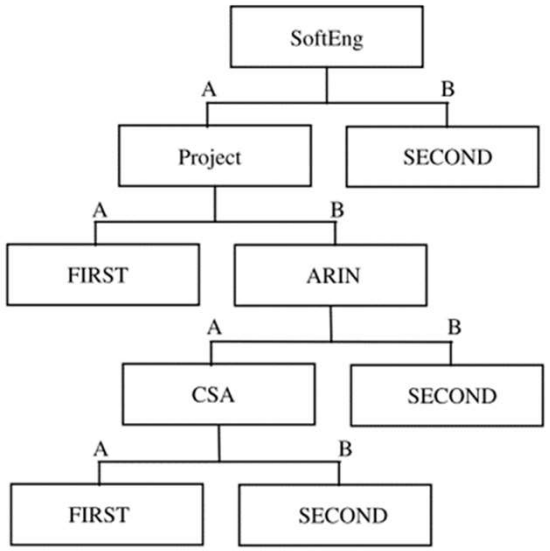
Decision Trees for the *degrees* Dataset

The *degrees* Dataset

- The left-hand side of each rule (known as the *antecedent*) comprises a number of *terms* joined by the logical AND operator.
- Each term is a simple test on the value of a categorical attribute (e.g. SoftEng = A, in the *degrees* example) or a continuous attribute (e.g. Humidity > 75, in the *Golf* example).
- A set of rules of this kind is said to be in *Disjunctive Normal Form* (DNF).
- The individual rules are sometimes known as *disjuncts*.

The *degrees* Dataset

- Looking at this decision trees for the *degrees* example in terms of *data compression*, the decision tree can be written as five decision rules with a total of 14 terms, an average of 2.8 terms per rule.



Decision Trees for the *degrees* Dataset

The *degrees* Dataset

- Each instance in the original degrees training set could also be viewed as a rule, for example
  - IF SoftEng = A AND ARIN = B AND HCI = A AND CSA = B AND Project = B THEN Class = SECOND

SoftEng	ARIN	HCI	CSA	Project	Class
A	B	A	B	B	SECOND

- In this case, there are 26 such rules, one per instance, each with five terms, making a total of 130 terms.
- Even for this very small training set, the reduction in the number of terms requiring to be stored from the training set (130 terms) to the decision tree (14 terms) is almost 90%.

The *degrees* Dataset

- For practical use, the rules can easily be simplified to an equivalent nested set of IF . . . THEN . . . ELSE rules, with even more compression.

```
if (SoftEng = A) {
   if (Project = A) Class = FIRST
   else {
      if (ARIN = A) {
         if (CSA = A) Class = FIRST
         else Class = SECOND
      }
      else Class = SECOND
   }
}
else Class = SECOND
```

The TDIDT Algorithm

- Decision trees are widely used as a means of generating classification rules because of the existence of a simple but very powerful algorithm called **TDIDT** (Top-Down Induction of Decision Trees).
- This has been known since the mid-1960s and has formed the basis for many classification systems, two of the best-known being **ID3** and **C4.5**, as well as being used in many commercial data mining packages.
- The method produces decision rules in the implicit form of a decision tree. Decision trees are generated by repeatedly splitting on the values of attributes. This process is known as *recursive partitioning*.
- In the standard formulation of the TDIDT algorithm there is a training set of instances. Each instance corresponds to a member of a universe of objects, which is described by the values of a set of categorical attributes. (The algorithm can be adapted to deal with continuous attributes)

The TDIDT Algorithm

- The basic algorithm can be given in just a few lines as shown below:

TDIDT: BASIC ALGORITHM

IF all the instances in the training set belong to the same class
THEN return the value of the class
ELSE (a) Select an attribute A to split on^{+}
(b) Sort the instances in the training set into subsets, one for each value of attribute A
(c) Return a tree with one branch for each *non-empty* subset, each branch having a descendant subtree or a class value produced by applying the algorithm recursively

$^{+}$ Never select an attribute twice in the same branch

The TDIDT Algorithm

The TDIDT Algorithm

- At each non-leaf node an attribute is chosen for splitting. This can potentially be any attribute, except that the same attribute must not be chosen twice in the same branch.
- This restriction is entirely innocuous (safe), e.g. in the branch corresponding to the incomplete rule as below, it is not permitted to choose *SoftEng* or *Project* as the next attribute to split on, but as their values are already known there would be no point in doing so.

  IF SoftEng = A AND Project = B . . . . . .

- However this harmless restriction has a very valuable effect. Each split on the value of an attribute extends the length of the corresponding branch by one term, but the maximum possible length for a branch is **M** terms where there are **M** attributes. Hence the algorithm is guaranteed to terminate.

The TDIDT Algorithm

- There is one important condition which must hold before the TDIDT algorithm can be applied.
- This is the *Adequacy Condition*: no two instances with the same values of all the attributes may belong to different classes.
- This is simply a way of ensuring that the training set is consistent.

The TDIDT Algorithm

- A major problem with the TDIDT algorithm, which is not apparent at first sight, is that it is *underspecified*. The algorithm specifies '**Select an attribute A to split on**' but no method is given for doing this.
- Provided the *adequacy condition* is satisfied the algorithm is guaranteed to terminate and any selection of attributes (even random selection) will produce a decision tree, provided that an attribute is never selected twice in the same branch.
- This *under-specification* may seem desirable, but many of the resulting decision trees (and the corresponding decision rules) will be of little, if any, value for predicting the classification of unseen instances.

The TDIDT Algorithm

- Thus some methods of selecting attributes may be much more useful than others.
- Making a good choice of attributes to split on at each stage is crucial to the success of the **TDIDT** approach.

Types of Reasoning

- The automatic generation of decision rules from examples is known as *rule induction* or *automatic rule induction*.
- Generating decision rules in the implicit form of a decision tree is also often called *rule induction*, but the terms ***tree induction*** or ***decision tree induction*** are sometimes preferred.
- Logicians distinguish between different types of reasoning.
  - Deduction
  - Abduction
  - Induction

Types of Reasoning: Deduction

- The most familiar type of reasoning is *deduction*, where the conclusion is shown to follow necessarily from the truth of the *premises*, for example

All Men Are Mortal
John is a Man

Therefore John is Mortal

- If the first two statements (the *premises*) are true, then the conclusion must be true.
- This type of reasoning is entirely reliable but in practice rules that are 100% certain (such as '**all men are mortal**') are often not available.

Types of Reasoning: Abduction

- A second type of reasoning is called *abduction*. An example of this is

All Dogs Chase Cats
Fido Chases Cats

Therefore Fido is a Dog

- Here the conclusion is consistent with the truth of the *premises*, but it may not necessarily be correct. Fido may be some other type of animal that chases cats, or perhaps not an animal at all.
- Reasoning of this kind is often very successful in practice but can sometimes lead to incorrect conclusions.

Types of Reasoning: Induction

- A third type of reasoning is called *induction*. This is a process of generalisation based on repeated observations.

> After many observations of x and y occurring together, learn the rule
> **if** x **then** y

- For example,
  - if I see 1,000 dogs with four legs I might reasonably conclude that "if x is a dog then x has 4 legs" (or more simply "all dogs have four legs").
  - This is induction.
- The decision trees derived from the *golf* and *degrees* datasets are of this kind. They are generalised from repeated observations (the instances in the training sets) and we would expect them to be good enough to use for predicting the classification of unseen instances in most cases, but they may not be infallible (foolproof).

Exercises

- What is the *adequacy condition* on the instances in a training set?
- What are the most likely reasons for the condition not to be met for a given dataset?
- What is the significance of the *adequacy condition* to automatic rule generation using the TDIDT algorithm?
- What happens if the basic TDIDT algorithm is applied to a dataset for which the *adequacy condition* does not apply?

Reference

- Max Bramer, “Chapter 4: Using Decision Trees for Classification”, *Principles of Data Mining* (4th Edition).