Minería de datos

U2 Entendimiento de datos

Héctor Maravillo

Section 1

Knowledge representation

Input

The **input** of a learning methods takes the form of:

- **Concepts**. A **concept description** refers to the entity or idea that is intended to be learned. This should be **intelligible**, meaning that it can be understood and communicable, and **operational**, that is, it can be applied to actual examples.
- **Instances**. Each instance is an **individual**, independent **example** of the concept to be learned.
- **Attributes**. Each instance is characterized by the values of attributes that measure different aspects of the instance.

Datasets

Typically, a **dataset** in machine learning is represented as a matrix of instances versus attributes, which in database terms is a single relation, or a **flat file**.

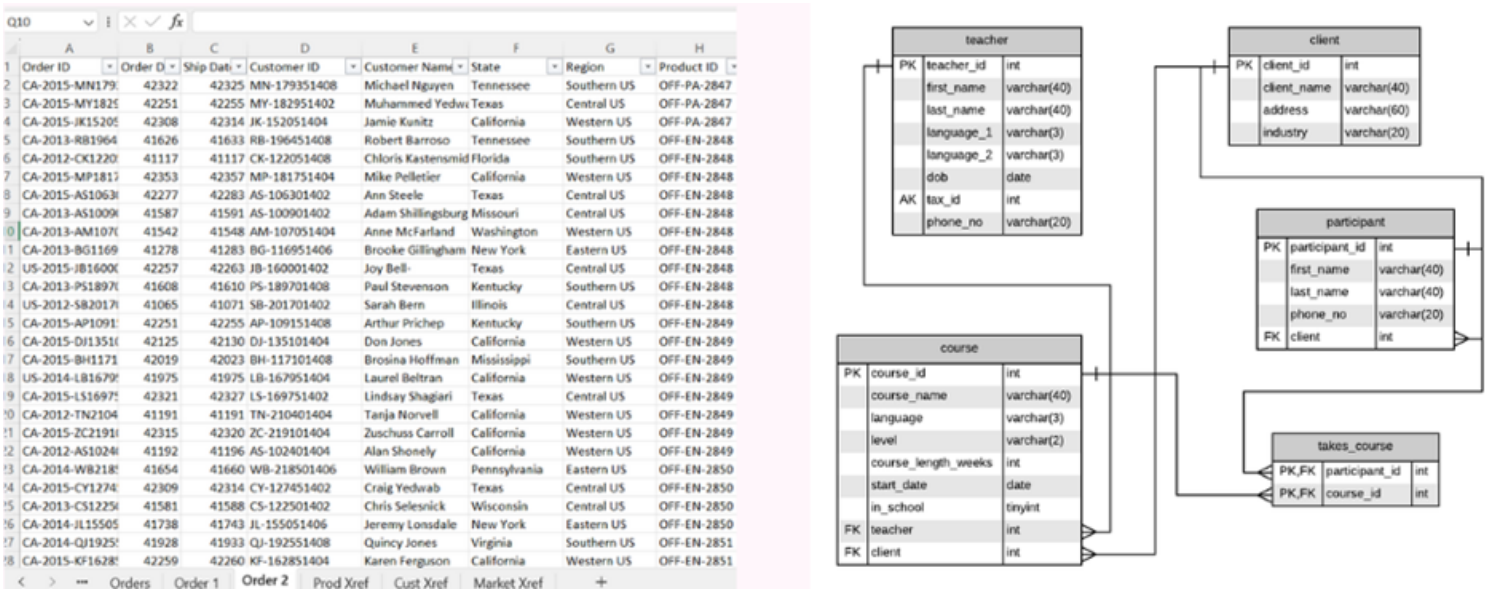
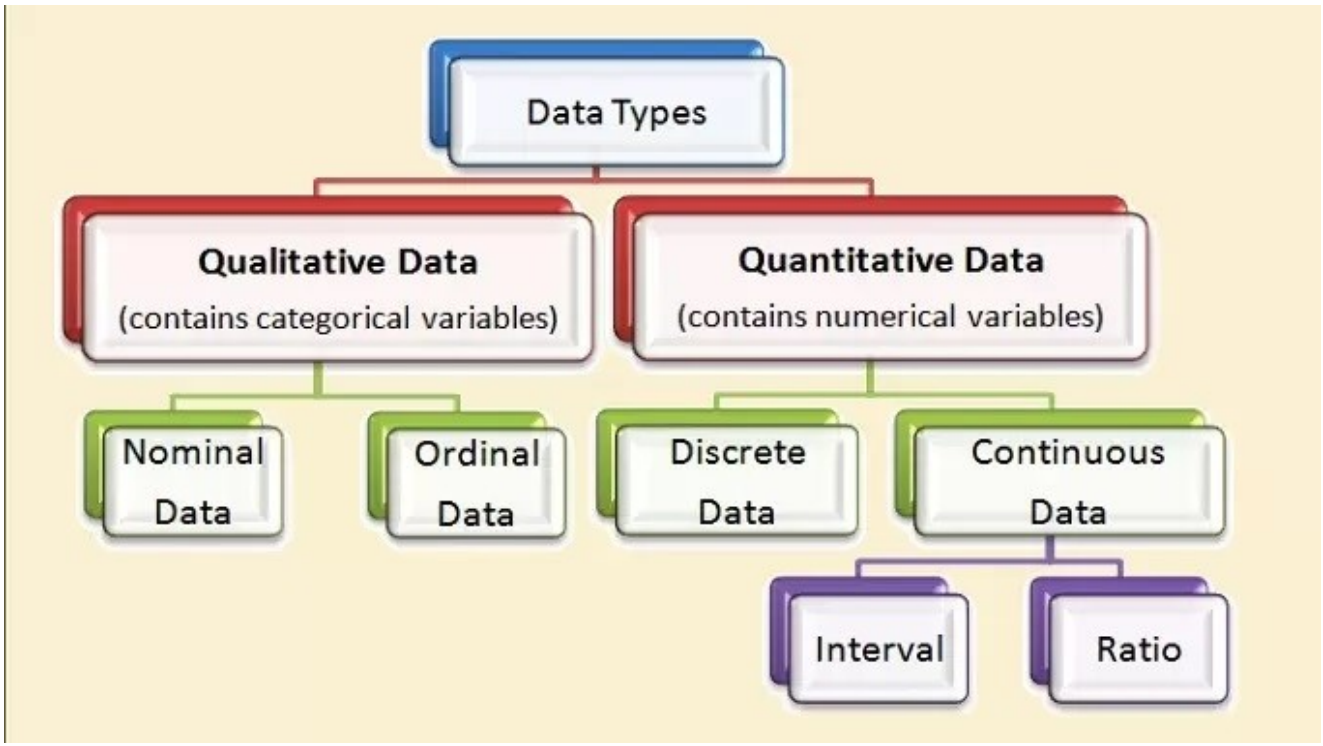


Figure: Flat file vs. relational database

Attributes

The **value** of an attribute for a particular instance is a **measurement** of the quantity to which the attribute refers.

There is a clear distinction between attributes that are **numerical** (sometimes called **continuous**) and those that are **nominal**.



Attributes

Statistics text often introduce four *levels of measurements*:

- **Nominal** attributes take on values in a prespecified, finite set of possibilities. The values themselves serve just as **labels** or names.
- **Ordinal** quantities are ones that make it possible to **rank order** the categories. However, although there is a notion of **ordering**, there is no notion of distance.
- **Interval** quantities have values that are not only ordered but also measured in **fixed and equal units**.
- **Ratio** quantities are ones for which the measurement method inherently defines a **zero point**.

Knowledge representation

In the usual supervised machine learning, the learning algorithm receives as **input** the **background knowledge** and the **learning data**. It then searches its **hypothesis space** and returns the final hypothesis as a result.

It is therefore necessary to **efficiently represent** the background knowledge, the input data and the hypothesis space in order to allow for efficient usage and generation of new knowledge.

Knowledge representation

Different kinds of knowledge are represented in different ways.

Most frequently, the following types of **knowledge representation** are used:

- Propositional calculus.
- First-order predicate calculus.
- Discriminant and regression functions.
- Probability distribution.

Propositional calculus

In mathematical logic, **propositional calculus** is a formal deduction system, whose atomic formulae are **propositional variables**.

Propositional calculus is frequently used for representation of:

- Learning examples in classification and regression problems
- Hypotheses in symbolic learning of classification and regression rules.

Representation with attributes

An **attribute** (feature) is a **variable** with a specified **domain** (a set of allowed values). It represents some property of an example.

Representation with attributes

Representation with attributes is defined by

- A set of l attributes
$$A = \{A_0, \ldots, A_a\}$$
- For each discrete attribute A_i: a set of allowable values
$$V_i = \{V_1, \ldots, V_m\}$$
- For each continuous attribute A_i: an interval of allowable values
$$V_i = [\min_i, \max_i]$$

Representation with attributes

- The dependent (target) variable is by definition represented with A_0:
  - In **classification** problems A_0 is a discrete attribute (Class C with values $\{C_1, \ldots, C_{m_0}\}$).
  - In **regression** problems A_0 is a continuous attribute.
- Each example is described with an attribute vector

$$t_j = (v_{1,j}, v_{2,j}, \ldots, v_{a,j})$$

its value of the dependent variable is given by

$$c_j = v_{0,j}$$

- The set of learning examples is a set of attribute vectors

$$T = \{t_1, \ldots, t_n\}$$

Classification and regression rules

Classification and regression rules are usually made of two parts:

- A conditional part (**antecedent**). Usually, it is a conjunction of propositions T_i.
- An unconditional part (**consequent**)). It includes the proposition T_0:

$$T_{i_1} \wedge T_{i_2} \wedge \ldots \wedge T_{i_l} \Rightarrow T_0$$

Rule representation can be generalized by allowing other logical operators in the antecedent. They may include disjunction, negation, equivalence, and others.

Classification and regression rules

The propositions are formed like

$$T_i = (A_i \in V_i^{'})$$

where $V_i^{'}$ is

- A subset of allowed values for

$$A_i : V_i^* \subset V_i$$

  if the attribute A_i is discrete.
- A sub-interval of the interval of the allowed values for the attribute

$$A_i : V_i^* \subset [min_i^*, max_i^*], \quad min_i^* \geq \min_i, \quad max_i^* \leq max_i$$

- In regression problems, the interval in the consequential proposition is sometimes reduced to a single point

$$min_0^* = max_i^*$$

- In classification problems, the consequential proposition may include only one value:

$$|V_0^*| = 1$$

Association rules

Association rules are a generalization of classification rules to problems where not only the target variable (class), but also the associations between attributes are interest.

The consequential part of the rule therefore does not include the proposition about the target variable T_0, but an arbitrary conjunction of propositions:

$$T_{i_1} \wedge T_{i_2} \wedge \ldots \wedge T_{i_l} \Rightarrow T_{j_1} \wedge T_{j_2} \wedge \ldots \wedge T_{j_k}$$

In practice, $k > 2$ is only rarely used.

Hypothesis space

In symbolic rule learning the hypothesis space $\mathscr{H}$ is defined as the power set of all possible rule.

The following properties of hypotheses are important:

- **Consistency:** A hypothesis H is **consistent** if for each rule $r \in H$ it holds that it correctly predicts the class value for all learning examples covered by the antecedent of the rule.
- **Completeness:** a hypothesis H is **complete** if every learning example is covered by at least one rule $r \in H$.

Section 2

Decision trees

Decision trees

A **decision tree** is a special case of a set of decision rules.

A decision tree consists of **internal nodes** -corresponding to attributes edges- corresponding to subsets of attribute values, and **terminal nodes** (leaves) corresponding to class labels.

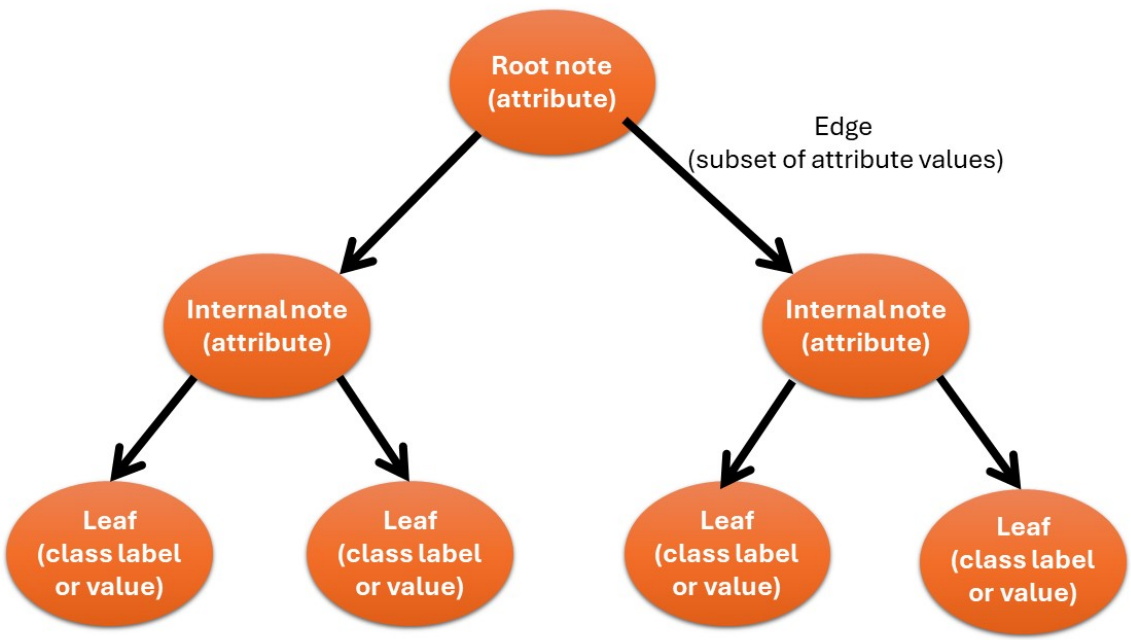


Figure: Schematic representation of a decision or regression tree.

Decision trees

Each path starting in a tree root (topmost node) and ending in a tree leaf corresponds to a **decision rule**.

The conditions (pairs of attribute names and corresponding sets of attribute values) evaluated in each internal node are conjunctively connected.

Decision tree learning is one of the most widely used and practical methods for **inductive inference**. It is a method for approximating discrete-valued functions that is **robust to noisy data** and capable of learning disjunctive expressions.

Regression trees

If leaf labels are continuous, such a tree is called a **regression tree**.

Knowledge representation with decision or regression trees can be generalized by evaluating in each node an **arbitrary function of several attributes** instead of a single attribute. For example, in regression trees terminal nodes (leaves) often consist of linear functions of subsets of attributes.

Appropriate problems for decision tree learning

Decision tree learning is generally best suited to problems with the following characteristics:

- Instances are represented by **attribute-valued pairs**. real-valued attributes as well.
- The target function has **discrete output values**.
- Decision tree learning methods are **robust to errors**: in classification of the training examples and in the attribute values.
- The training data may contain **missing attribute values**.

ID3 Algorithm

ID3 Algorithm is an algorithm for **empirical learning**, developed by Ross Quinlan in 1986. Its main task is constructing a decision tree.

The central choice in the **ID3 algorithm** is selecting which attribute to test at each note in the tree.

To select the attribute that is most useful for classifying examples is used the **information gain**, that measures how well a given attribute separates the training examples according to their target classification.

Learning from examples

Let U denote the set of examples. Any subset C of U may be considered as a **concept** to be learned.

An example from C is called a **positive example** for C, an example from $\bar{C} = U - C$ is called a **negative example** for C.

A concept C is typically represented by a unique value of a variable d, called a **decision**. Thus each value w of a decision d describes a unique concept C.

The task of learning from examples is to learn the **description** D of the concept C.

Learning from examples

The description D learned from concept C may be expressed by a **set of rules** or a **decision tree**. The description D, characterizing the set C, is easier to comprehend for humans, and can be used by other computer programs, such as expert systems.

Two important properties of a description D:

- **Completeness**: For any example e from concept C there exists a description D such that D describes e.
- **Consistency**: For every example e from concept C there are not two different descriptions D and $D^{'}$ such that both D and $D^{'}$ describe e.

Learning from examples

Say that the original **production rule** is

$$C_1 \wedge C_2 \wedge \ldots \wedge C_j \rightarrow A$$

where $C_1, C_2, \ldots, C_j$ are **conditions** and A is an **action**.

Usually conditions are presented as ordered pairs (a, v), where a is an attribute and v is its value, and action A is presented as an ordered pair (d, w), where d is a decision and w is its value. Thus (d, w) describes some concept C.

Information gain

Let a be the attribute with values $a_1, a_2, \ldots, a_l$ and let d be a decision with values $d_1, d_2, \ldots, d_k$. Then the **information gain** is defined as

$$I(a \to d) = H(d) - H(d|a)$$

where $H(d)$ is the **entropy** of decision d:

$$H(d) = -\sum_{i=1}^{k} p(d_i) \log p(d_i) \tag{1}$$

$$H(d|a) = \sum_{j=1}^{l} p(a_j) H(d|a_j) \tag{2}$$

$$= -\sum_{j=1}^{l} p(a_j) \sum_{i=1}^{k} p(d_i|a_j) \log p(d_i|a_j) \tag{3}$$

The **entropy** $H(X)$ of a discrete variable random X is defined by

$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

The log is to the base 2 and entropy is expressed in **bits**. In this case, the entropy can be interpreted as the minimum numbers of bits of information needed to encode the classification of an arbitrary member of S.

Note that entropy is a **functional** of the distribution of X. It does not depend on the actual values taken by the random variable X, but only on the probability.

Entropy

If the base of the logarithm is b, we denote the entropy as $H_b(X)$.

If the base of the logarithm is e, the entropy is measured in **nats**. Unless other specified, we will take all the logarithms to **base 2**.

Note that

$$H_b(X) = \log_b a H_a(X)$$

For calculations, we use the convention that $0 \log 0 = 0$. Adding terms of zero probability does not change the entropy.

ID3 Algorithm

At this point, the corresponding decision tree, created by **ID3**, has the root, labeled by the attribute a and outgoing arches from the root, each such arch corresponds to a value a_j of the attribute.

The set of all examples with the same value a_j of attribute a consists of a new set S of examples.

When all members of S are members of the same concept C, they do not need to be further partitioned, so S is a label of the leaf of the tree. However, when S contains examples from at least two different concepts, the node of the tree labeled by S is the root of a new subtree.

An attribute to be a label for the root of this new subtree is selected among remaining attributes again on the basis of the **maximum** of **information gain** criterion.

Example

	Attributes			*Decision*
	Temperature	**Headache**	**Nausea**	**Flu**
1	high	yes	no	yes
2	very_high	yes	yes	yes
3	normal	no	no	no
4	high	yes	yes	yes
5	high	no	yes	no
6	normal	yes	no	no
7	normal	no	yes	no

Figure: Example of the decision table.

Example

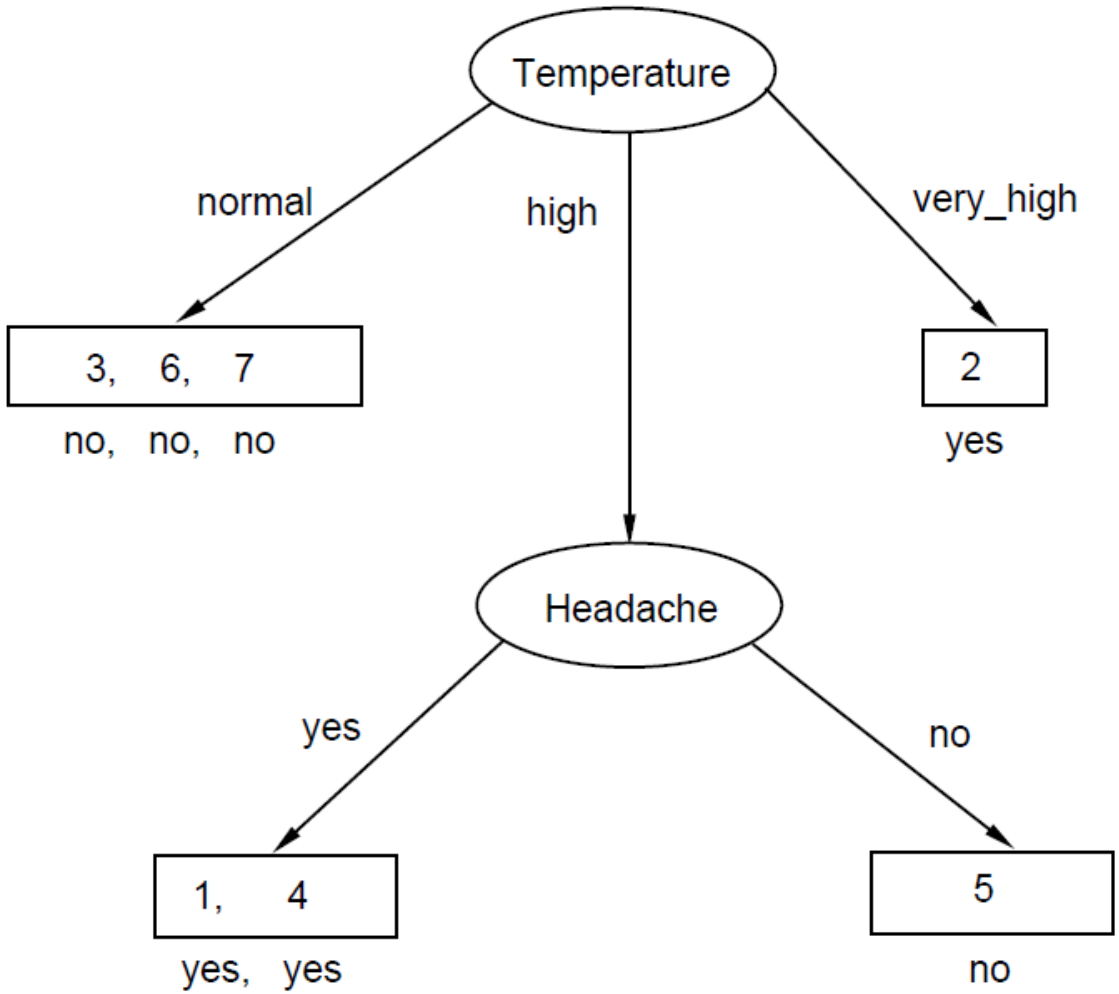


Figure: Final decision tree (Information Gain Version)

Example

The following rules can be deduced from the above decision tree:

$$
\begin{aligned}
(\mathit{Temperature}, \mathit{normal}) &\rightarrow (\mathit{Flu}, \mathit{no}) \\
(\mathit{Temperature}, \mathit{high}) \wedge (\mathit{Headache}, \mathit{yes}) &\rightarrow (\mathit{Flu}, \mathit{yes}) \\
(\mathit{Temperature}, \mathit{high}) \wedge (\mathit{Headache}, \mathit{no}) &\rightarrow (\mathit{Flu}, \mathit{no}) \\
(\mathit{Temperature}, \mathit{very\ high}) &\rightarrow (\mathit{Flu}, \mathit{yes})
\end{aligned}
$$

Gain Ratio Version

The **information gain version** of **ID3**, builds the decision tree with some bias—attributes with **greater** number of values are preferred by the algorithm.

In order to avoid this bias, another version of ID3 has been developed. This version uses another criterion for attribute selection, called gain ratio.

The **gain ratio** is defined as follows

$$\frac{I(a \to d)}{H(a)}$$

Example

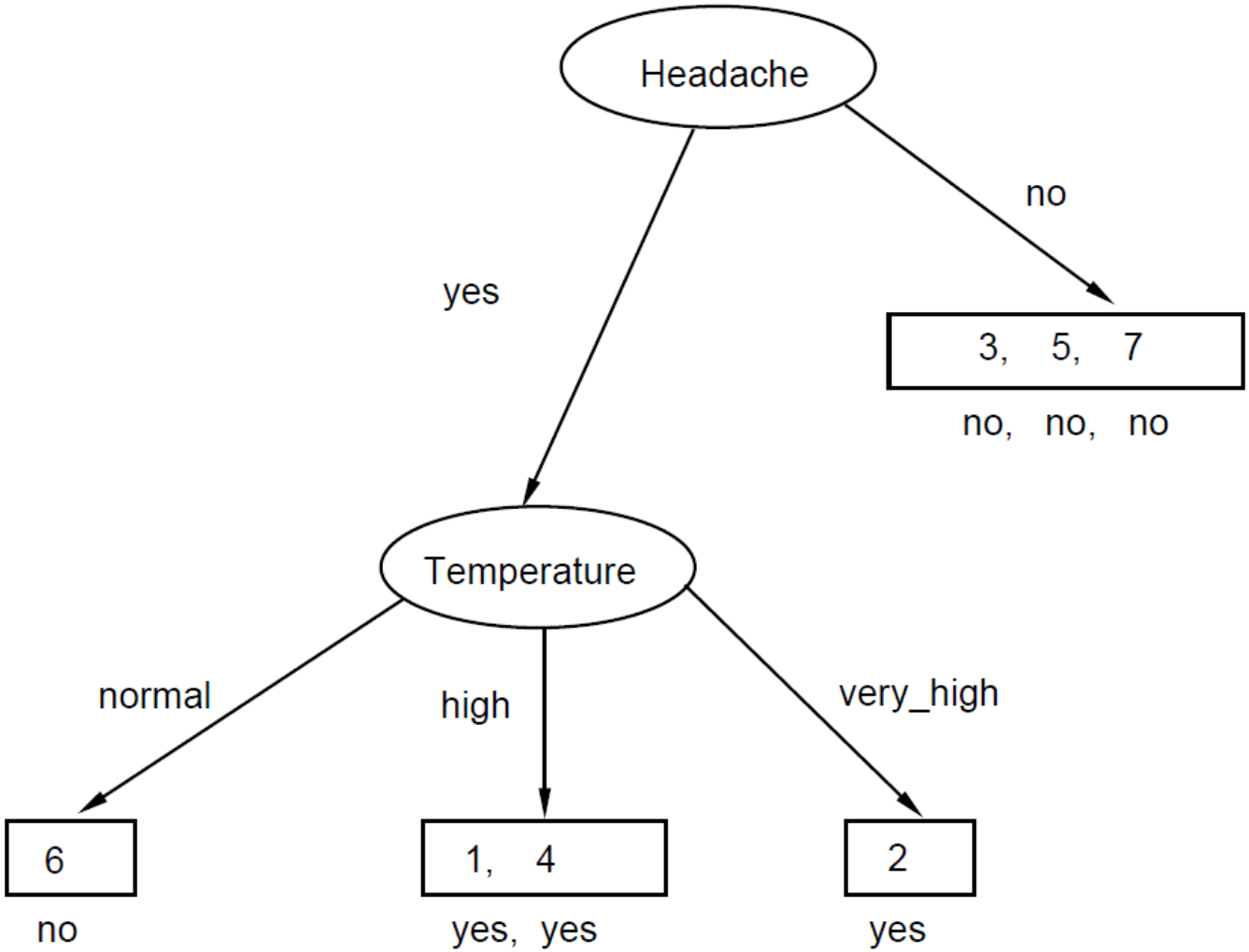


Figure: Final decision tree (Gain Ratio Version)

Example

The following rules can be deduced from the above decision tree:

$$
\begin{aligned}
(\mathit{Headache}, \mathit{yes}) \wedge (\mathit{Temperature}, \mathit{normal}) &\rightarrow (\mathit{Flu}, \mathit{no}) \\
(\mathit{Headache}, \mathit{yes}) \wedge (\mathit{Temperature}, \mathit{high}) &\rightarrow (\mathit{Flu}, \mathit{yes}) \\
(\mathit{Headache}, \mathit{yes}) \wedge (\mathit{Temperature}, \mathit{very\ high}) &\rightarrow (\mathit{Flu}, \mathit{yes}) \\
(\mathit{Headache}, \mathit{no}) &\rightarrow (\mathit{Flu}, \mathit{no})
\end{aligned}
$$

CART

CART stands for **Classification and Regression Tree**, is a decision tree algorithm applicable to both classification and regression.

CART employs the **Gini Index** for selecting the splitting features.

Intuitively, Gini Index represents the likelihood of two samples we randomly selected from data set belonging to different classes.

Given a feature set A, we select the feature with the lowest Gini index as the splitting feature.

Gini Index

If a data set T contains examples from n classes, **gini index**, $gini(T)$ is defined as

$$gini(T) = 1 - \sum_{i=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in T.

The attribute provides the **smallest** $gini(T)$ is chosen **to split** the node.

Note that Gini indices equal to 0 indicate that all the data belong to a single category.

References

- Cover, T. & Thomas, J. *Elements of Information Theory*, 2nd ed., John Wiley & Sons.
- Duda, R., Hart, P. & Stork, D. (2001) *Pattern Classification*. 2nd ed., Wiley.
- Grzynala-Busse, J. (1993). Selected Algorithms of Machine Learning from Examples. *Fundamental Informaticae*, 18, 193–207.
- Kononenko, I. & Kukar, M. *Machine Learning and Data Mining. Introduction to principles and algorithms*, Horwood Publishing, 2007.
- Mitchell, T., *Machine Learning*, McGraw-Hill, 1997.
- Witten, I. & Frank, E. *Data Mining. Practical Machine Learning Tools and Techniques*, 2nd ed., Morgan Kaufmann Publishers, 2005.