Statistical and Learning Methods in Natural Language Processing

Ido Dagan
dagan@macs.biu.ac.il

Shuly Wintner
shuly@cs.haifa.ac.il

Spring 2004

Supervised learning techniques

- Decision trees
- K Nearest Neighbors
- Perceptron
- Naïve Bayes

All are instances of techniques for a general problem: classification.

Classification

Problem: Given a universe of objects and a pre-defined set of of classes, or categories, assign each object to its correct class.

Examples:

Problem	Objects	Categories
Tagging	words in context	POS tag
WSD	words in context	word sense
PP attachment	sentences	parse trees
Language identification	text	language
Text categorization	text	topic

Text categorization

We focus on a specific problem, text categorization.

Problem: Given a document and a pre-defined set of topics, assign the document to one or more topics.

Typical sets of topic categories: Reuters; Yahoo; etc.

Typical categories: "mergers and acquisitions"; "crude oil"; "earning reports"; etc.

Classification

Formal setting for statistical classification problems:

- A training set is given containing a set of objects, each labeled by one or more classes;
- The training set is encoded via a data representation model. Typically, each object in the training set is represented as a pair $(\vec{x}, c)$, where $\vec{x} \in R^n$ is a vector of measurements and c is a category label;
- A model class, which is a parameterized family of classifiers, is defined. A training procedure selects one classifier from this family (training).
- The classifier is evaluated (testing).

Classification

Example: text categorization

- The training set: a collection of text documents, each labeled by one or more topic categories
- Data representation: each document is associated with a feature vector
- Training: the parameters of a model class are set
- Testing: for binary classification, recall and precision.

Decision trees

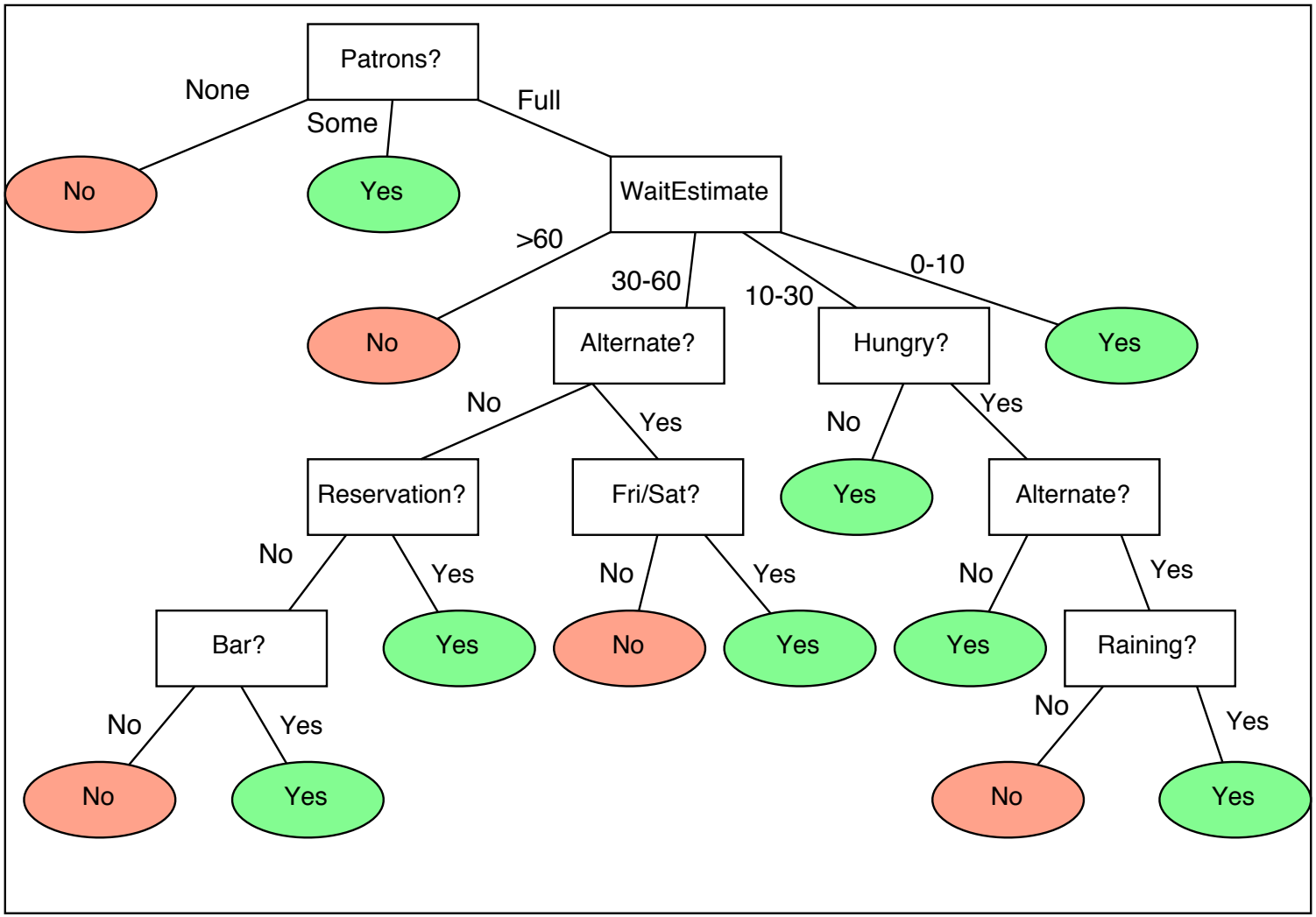
A decision tree takes as input an object described by a set of properties, and outputs a yes/no decision. Decision trees therefore represent Boolean functions.

Each internal node in the tree corresponds to a test of the value of one of the properties, and the branches from the node are labeled with the possible values of the test.

Each leaf node in the tree specifies the Boolean value to be returned if that leaf is reached.

Decision trees

Example: A decision tree for deciding whether to wait for a table


Patrons?
None
Some
Full
No
Yes
WaitEstimate
>60
30-60
10-30
0-10
No
Alternate?
Hungry?
Yes
No
Yes
No
Yes
Reservation?
Fri/Sat?
Yes
Alternate?
No
Yes
No
Yes
No
Yes
Bar?
Yes
No
Yes
Yes
Raining?
No
Yes
No
Yes
No
Yes

Decision trees

How to find an appropriate data representation model?

This is an art by itself, and feature engineering is a major task.

Data representation

For the "waiting for a table" problem:

Alternate: Is there an alternative restaurant nearby?

Bar: Does the restaurant have a bar to wait in?

Fri/Sat: Is it a weekend?

Hungry: Are we hungry?

Patrons: How many people are in the restaurant? (none/some/full)

Price: The restaurant's price range ($/$$/$$$)

Raining: Is it raining outside?

Data representation

Reservation: Do we have a reservation?

Type: The type of restaurant (Thai/French/Italian/Burger)

WaitEstimate: Estimated wait time (0-10/10-30/30-60/>60)

Inducing decision trees from examples

An example is described by the values of the features and the category label (the classification of the example).

In binary classification tasks, examples are either positive or negative.

The complete set of example is called the training set.

Inducing decision trees from examples

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Figure 18.5 Examples for the restaurant domain.

Inducing decision trees from examples

How to find a decision tree that agrees with the training set?

This is always possible, since a tree can consist of a unique path from root to leaf for each example. However, such a tree does not generalize to other examples.

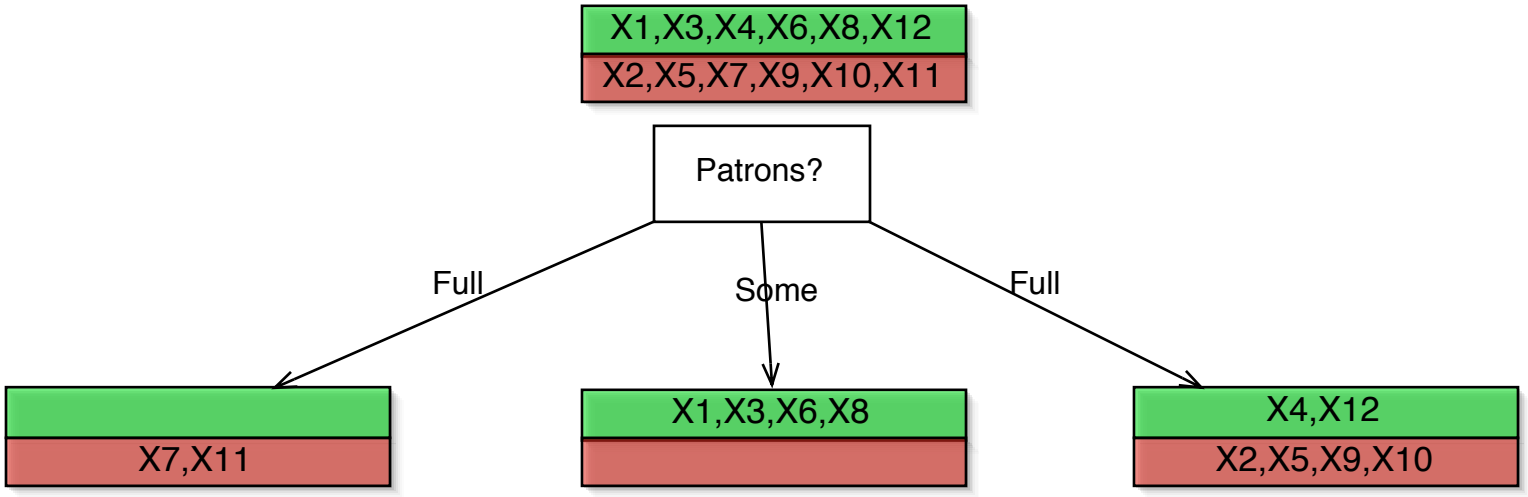
An induced tree must not only agree with all the examples, but also be concise. Unfortunately, finding the smallest tree is an intractable problem.

The basic idea behind the Decision-Tree-Learning algorithm is to test the most important feature first. By "most important" we mean the one that makes the most difference to the classification of an example. This way we hope to get the correct classification with a small number of tests, thereby generating a smaller tree with shorter paths.

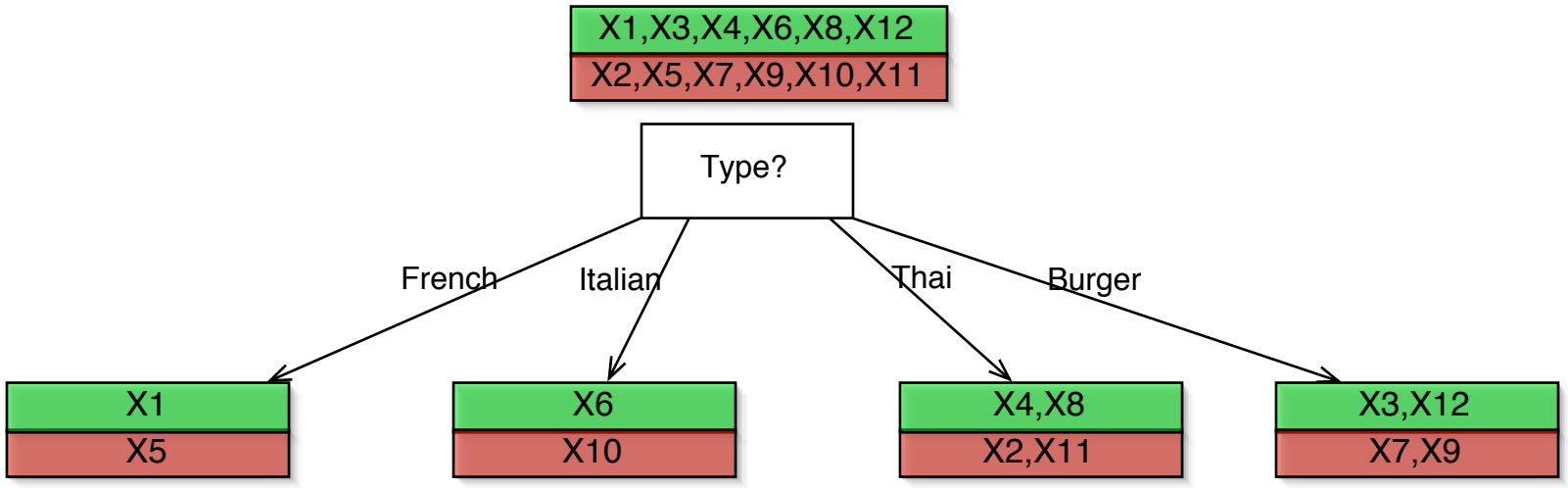
Inducing decision trees from examples

Suppose we start with the set of examples shown above.

Consider the contribution of each of the two features **Patrons** and **Type**.



Inducing decision trees from examples



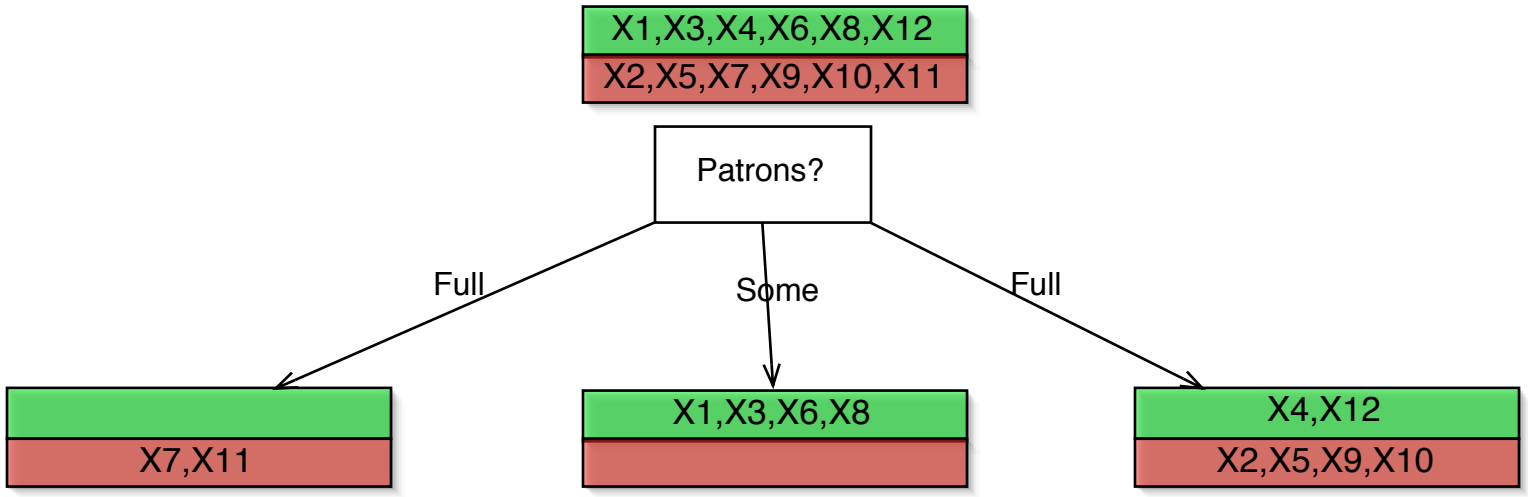
Hence "Patrons?" is a better feature than "Type?".

An algorithm for inducing decision trees

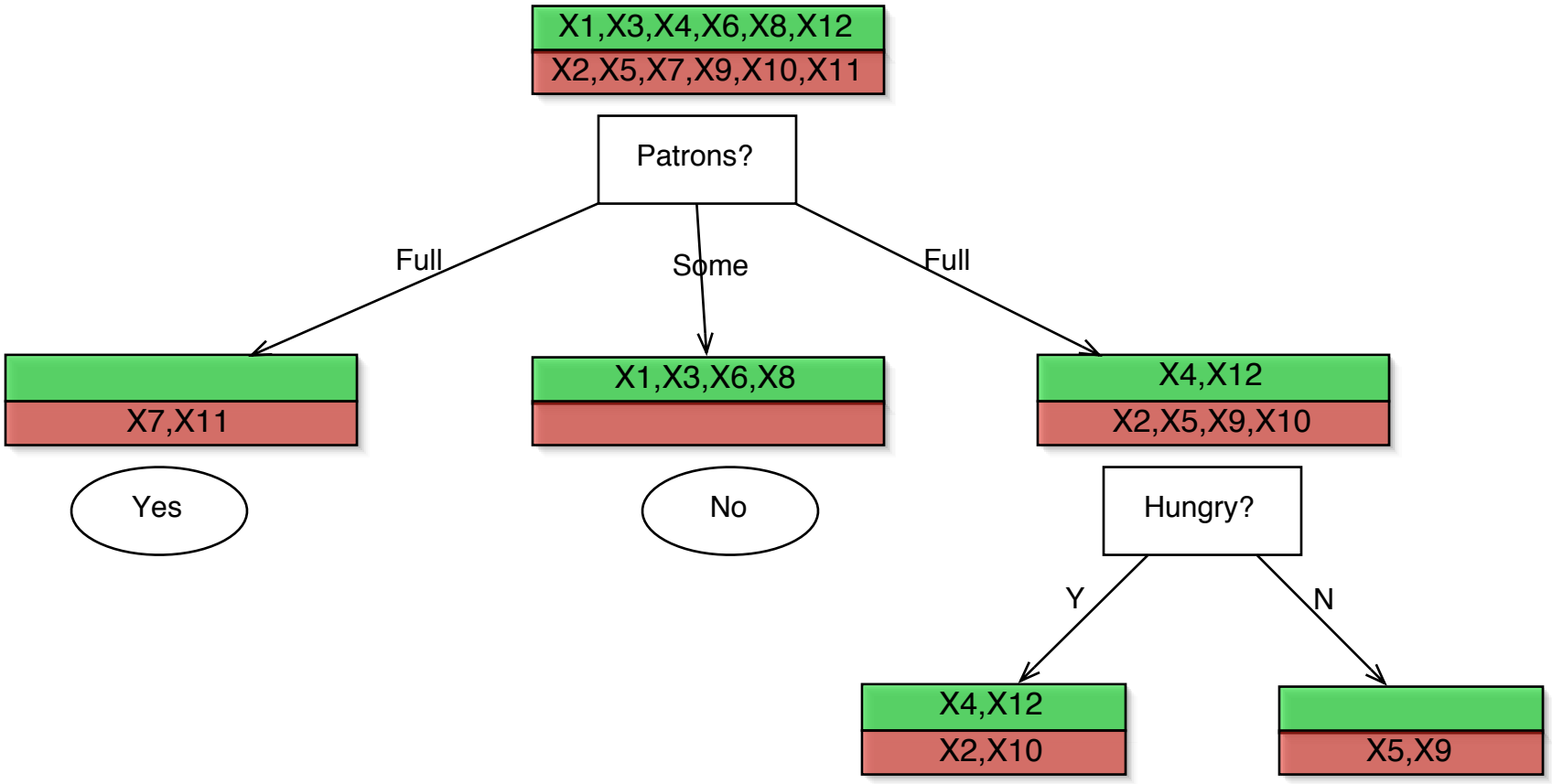
The algorithm: Decide which attribute to use as the first test in the tree. After the first feature splits up the examples, each outcome is a new decision tree learning problem in itself, with fewer examples and one fewer feature. For this recursive problem:

- if there are both positive and negative examples, choose the best attribute to split them.
- If all remaining examples are positive (or all negative), the answer is "yes" ("no").
- if no examples are left, return a default value.
- if no features are left (noise in the data), use a majority vote.

Inducing decision trees from examples



Inducing decision trees from examples



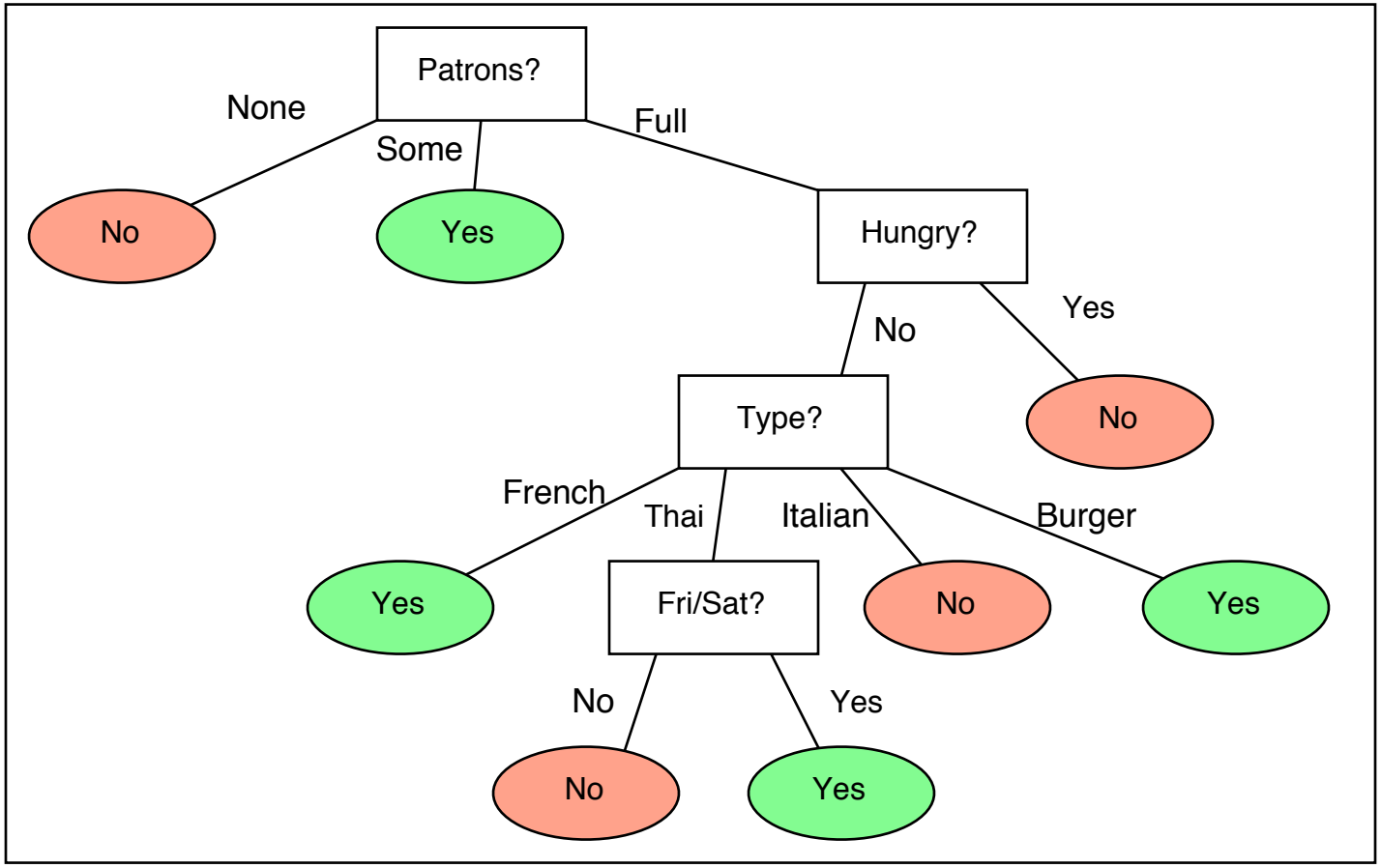
An algorithm for inducing decision trees

```
function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
  inputs: examples, set of examples
          attributes, set of attributes
          default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MAJORITY-VALUE(examples)
  else
      best ← CHOOSE-ATTRIBUTE(attributes, examples)
      tree ← a new decision tree with root test best
      for each value v_i of best do
          examples_i ← {elements of examples with best = v_i}
          subtree ← DECISION-TREE-LEARNING(examples_i, attributes − best,
                                           MAJORITY-VALUE(examples))
          add a branch to tree with label v_i and subtree subtree
      end
      return tree
```

Figure 18.7 The decision tree learning algorithm.

The result on the 12 examples



Decision trees for text categorization

In order to set a data representation model we must understand what documents look like.

Assume that the task is to classify Reuters documents to the class "earnings". That is, given a Reuters document, the classifier must determine whether its topic is "earnings" or not.

An example Reuters document

```
<REUTERS TOPICS="YES" NEWID="2005">
<DATE> 5-MAR-1987 09:22:57.75</DATE>
<TOPICS><D>earn</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<TEXT>&#2;
<TITLE>NORD RESOURCES CORP &lt;NRD> 4TH QTR NET</TITLE>
<DATELINE>    DAYTON, Ohio, March 5 -   </DATELINE>
<BODY>Shr 19 cts vs 13 cts
    Net 2,656,000 vs 1,712,000
    Revs 15.4 mln vs 9,443,000
    Avg shrs 14.1 mln vs 12.6 mln
    Shr 98 cts vs 77 cts
    Net 13.8 mln vs 8,928,000
    Revs 58.8 mln vs 48.5 mln
    Avg shrs 14.0 mln vs 11.6 mln
    NOTE: Shr figures adjusted for 3-for-2 split paid Feb 6, 1987.
 Reuter &#3;</BODY></TEXT>
</REUTERS>
```

Data representation

For the text categorization problem: use words which are frequent in "earnings" documents.

The 20 most representative words of this category are: vs, mln, cts, loss, 000, profit, dlrs, pct, net etc.

Data representation

Each document j is represented as a vector of $K = 20$ integers, $\vec{x_j} = \langle s_1^j, \ldots, s_K^j \rangle$, where:

$$s_i^j = \mathit{round}\left(10 \times \frac{1 + \log \mathit{tf}_i^j}{1 + \log l^j}\right)$$

where tf_i^j is the number of occurrences of word i in document j, and l^j is the length of document j. s_i^j is set to 0 if word i does not occur in document j.

In the above example, whose length is 59, the word vs occurs 8 times, hence $s_i^j = \mathit{round}(10 \times \frac{1+\log 8}{1+\log 59}) = 7$.

Training procedure

The model class is decision trees; the data representation is 20-integer vectors; now the training procedure has to be determined.

The splitting criterion is the criterion used to determine which feature should be used for splitting next, and which values this feature should split on.

The idea: split the objects at a node to two piles in the way that gives maximum information gain. Use information theory to measure information gain.

The stopping criterion is used to determine when to stop splitting.

Decision trees

Example: Reuters topic category "earnings"

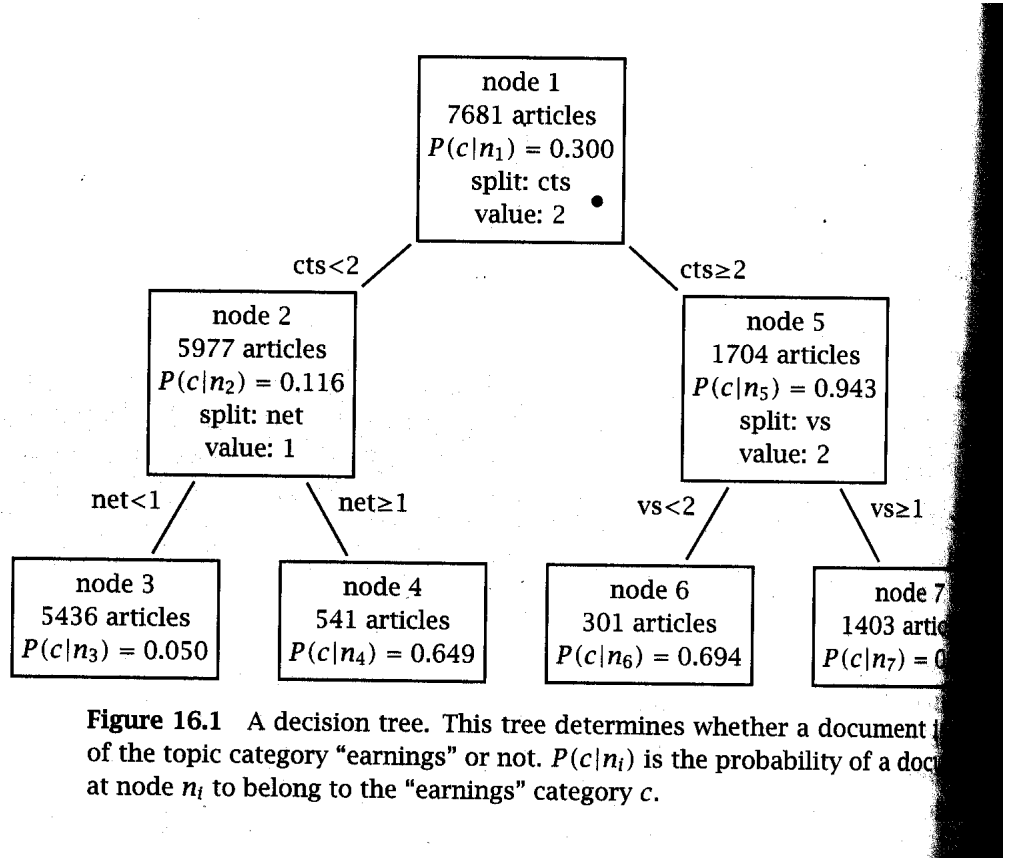


Figure 16.1 A decision tree. This tree determines whether a document i of the topic category "earnings" or not. $P(c|n_t)$ is the probability of a doc at node n_t to belong to the "earnings" category c.

Decision trees – summary

Decision trees are useful in non-trivial classification tasks (for simple tasks, simpler methods are available).

They are attractive because they can be very easily interpreted.

Their main drawback is that they sometimes make poor generalizations, since they split the training set into smaller and smaller subsets.

K-Nearest Neighbors (KNN)

A memory-based classification method. The basic idea: store all the training set examples in memory. To classify a new object, find the training example closest to it and return the class of this nearest example.

Problems: How to measure similarity? How to break ties?

KNN

KNN is a simple algorithm that stores all available examples and classifies new instances of the example language based on a similarity measure.

If there are n features, all vectors are instances of R^n. The distance $d(\vec{x_1}, \vec{x_2})$ of two example vectors x_1 and x_2 can be defined in various ways. This distance is regarded as a measure for their similarity.

To classify a new instance e, the k examples most similar to e are determined. The new instance is assigned the class of the majority of the k nearest neighbors.

KNN variations

Distance measures:

- Euclidean distance:

$$d(\vec{x}, \vec{y}) = \sqrt{\Sigma_{i=1}^{n}(x_i - y_i)^2}$$

- Cosine:

$$d(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}} = \frac{\Sigma_{i=1}^{n} x_i y_i}{\sqrt{\Sigma_{i=1}^{n} x_i^2}\sqrt{\Sigma_{i=1}^{n} y_i^2}}$$

A variant of this approach calculates a *weighted* average of the nearest neighbors. Given an instance e to be classified, the weight of an example increases with increasing similarity to e.

KNN – summary

The attractiveness of KNN stems from its simplicity: if an example in the training set has the same representation as the example to be classified, its category will be assigned.

A major problem of the simple approach of KNN is that the vector distance is not necessarily suited for finding intuitively similar examples, especially if irrelevant attributes are present.

Performance is also very dependent on the right similarity metric.

Finally, computing similarity for the entire training set may take more time (and certainly more space) than determining the appropriate path of a decision tree.

Evaluation

Am important recent development in empirical NLP has been the use of rigorous standards for evaluation of systems.

The ultimate demonstration of success is showing improved performance at the application level (here, text categorization).

For various tasks, standard benchmarks are being developed (e.g., the Reuters collection for text categorization).

Evaluation

Borrowing from Information Retrieval, empirical NLP systems are usually evaluated using the notions of precision and recall.

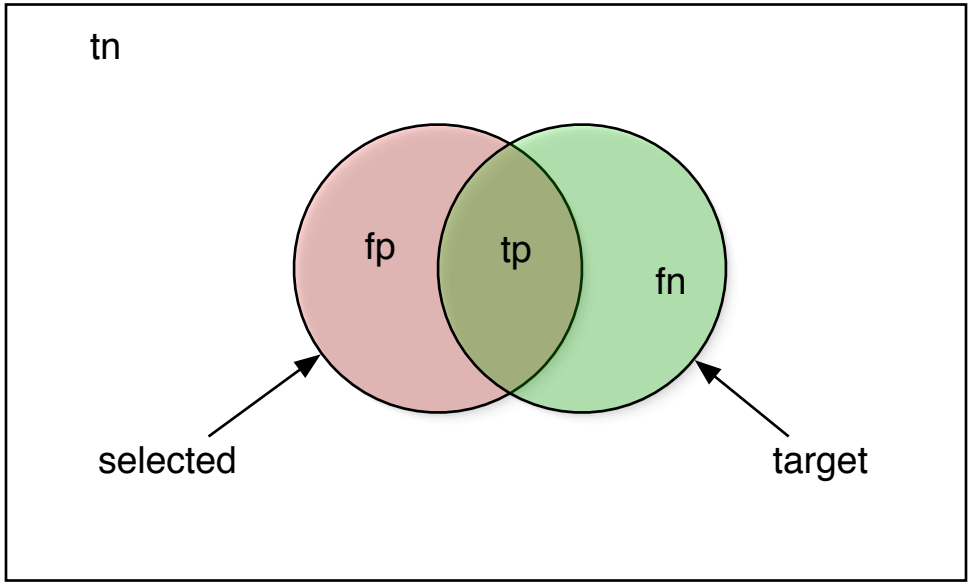
Example: for text categorization, a set of documents is given of which a subset is in a particular category (say, "earnings"). The system classifies some other subset of the documents as belonging to the "earnings" category.

The results of the system are compared with the actual results as follows:

	target	not target
selected	tp	fp
not selected	fn	tn

Evaluation

Graphically, the situation can be depicted thus:



Recall and precision

Precision is the proportion of the selected items that the system got right; in the case of text categorization it is the percentage of documents classified as "earning" by the system which are indeed "earning" documents:

$$P = \frac{tp}{tp + fp}$$

Recall is the proportion of target items that the system selected. In the case of text categorization, it is the percentage of the "earning" documents which were actually classified as "earning" by the system:

$$R = \frac{tp}{tp + fn}$$

Recall and precision

In applications like Information Retrieval, one can usually trade off recall and precision.

This tradeoff can be plotted in a precision–recall curve.

It is therefore convenient to combine recall and precision into a single measure of overall performance. One way to do it is the F-measure, defined as:

$$F = \frac{P \cdot R}{\alpha R + (1 - \alpha) P}$$

To weigh recall similarly to precision, α is set to 0.5, yielding:

$$F = \frac{2PR}{P + R}$$

Accuracy and error

Other measures of performance, perhaps more intuitive ones, are accuracy and error.

Accuracy is the proportion of items the system got right:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

whereas error is its complement:

$$\frac{fp + fn}{tp + tn + fp + fn}$$

The disadvantage of using accuracy is the observation that in most cases, the value of tn is huge, dwarfing all other numbers.

Evaluation of text categorization systems

For binary classification tasks, classifiers are typically evaluated using a table of counts:

	YES is correct	NO is correct
YES was assigned	tp	fp
NO was assigned	fn	tn

and then:

Accuracy: $\frac{tp+tn}{tp+tn+fp+fn}$

Recall: $\frac{tp}{tp+fn}$

Precision: $\frac{tp}{tp+fp}$

Evaluation of text categorization systems

When more than two categories exist, first prepare contingency tables for each category c_i, measuring c_i versus everything that is not c_i.

Then there are two options:

Macro-averaging compute an evaluation measure for each contingency table separately and average the evaluation measure over categories to get an overall measure of performance.

Micro-averaging make a single contingency table for all categories by summing the scores in each cell for all categories, then compute the evaluation measure for this large table.

Evaluation of text categorization systems

Macro-averaging gives equal weight to each category, whereas micro-averaging gives equal weight to each item. They can give different results when the evaluation measure is averaged over categories with different sizes.

Micro-averaged precision is dominated by large categories whereas micro-averaged precision gives a better sense of the quality of classification across all categories.