Empirical methods in NLP

- Some history
- The underlying motivation
- The current state-of-the-art
- A few application examples

Empirical methods in NLP: applications

Example: Segmentation

Problem: Given a word w, find a sequence of morphemes $m_1, \ldots, m_k$ such that $w = m_1 \cdots m_k$.

im|possible, in|credible, ir|regular, ir|resistable, in|finite, in|dependent, ...

ink, imply, Iran,...

resist|able, comfort|able, ed|ible, incred|ible, imposs|ible, ...

table, stable, ...

More complex cases: segmenting sentences to words in Asian languages.

Empirical methods in NLP: applications

Example: POS tagging

Problem: Given a text where each word is associated with all its possible parts of speech, determine the most likely POS for the word with respect to its context.

who	PRON(int), PRON(rel)
can	AUX, V(inf), N(sg)
it	EXPLETIVE, PRON(3sg)
be	V(inf)
?	PUNC

Empirical methods in NLP: applications

Example: POS tagging

Russian/JJ and/CC Ukrainian//NNP energy/NN officials/NNS
reached/VBD a/DT deal/NN Wednesday/NNP morning/NN ,/,
ending/VBG a/DT dispute/NN over/IN the/DT price/NN of/IN
Russian/JJ natural/JJ gas/NN that/WDT caused/VBD
shortages/NNS in/IN the/DT Ukraine//NNP and/CC
throughout/IN Western/NNP Europe/NNP ./.

Empirical methods in NLP: applications

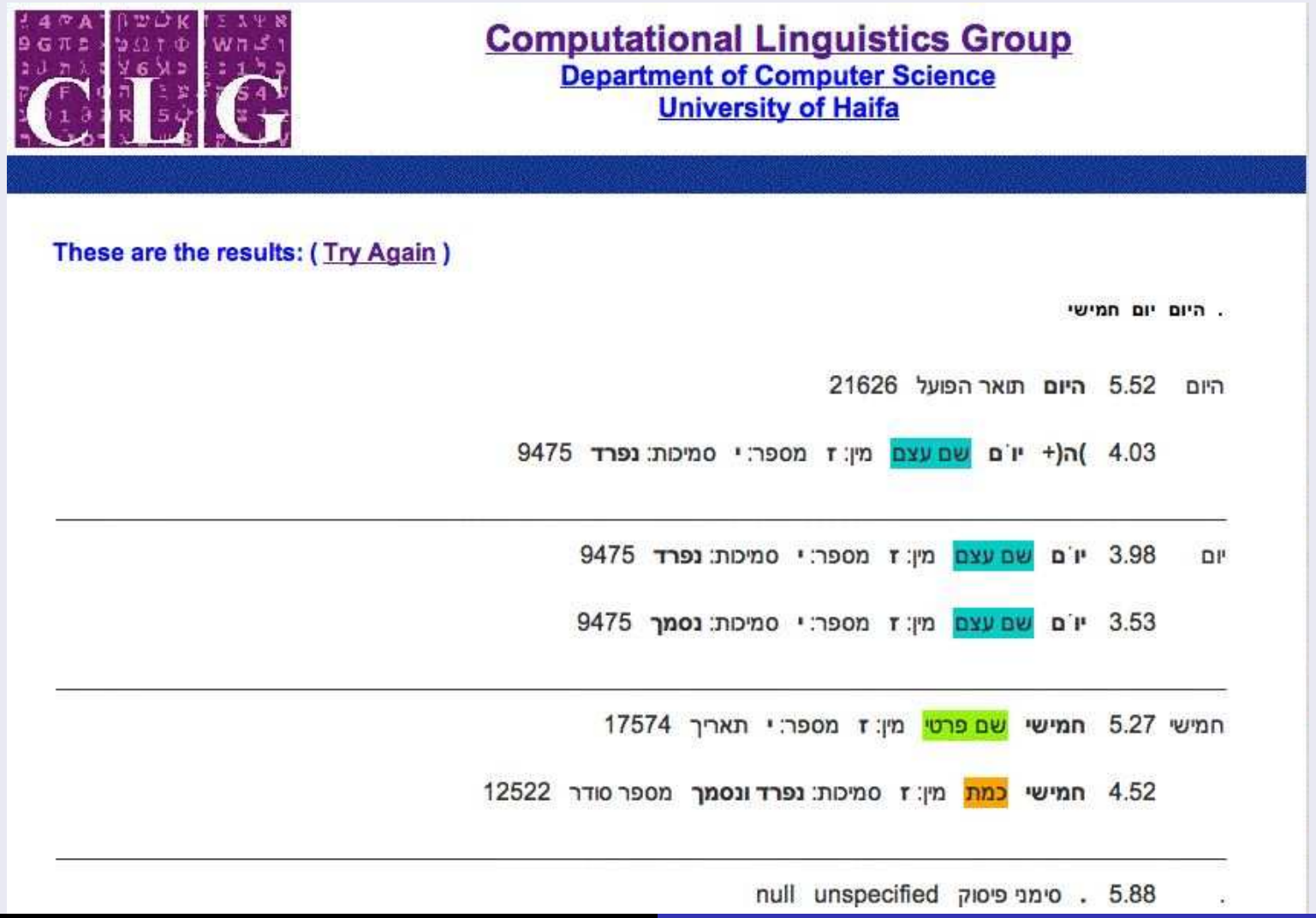
Example: Morphological disambiguation

A generalization of Part-of-speech Tagging

Problem: Given a text where each word is associated with all its possible morphological analyses, determine the most likely analysis for the word with respect to its context.

Empirical methods in NLP: applications

Example: Morphological disambiguation



Computational Linguistics Group
Department of Computer Science
University of Haifa

These are the results: (Try Again)

היום יום חמישי .

היום 5.52 **היום** תואר הפועל 21626

4.03)ה(+ **יו"ם** שם עצם מין: ז מספר: י סמיכות: **נפרד** 9475

יום 3.98 **יו"ם** שם עצם מין: ז מספר: י סמיכות: **נפרד** 9475

3.53 **יו"ם** שם עצם מין: ז מספר: י סמיכות: **נסמך** 9475

חמישי 5.27 **חמישי** שם פרטי מין: ז מספר: י תאריך 17574

4.52 **חמישי** כמת מין: ז סמיכות: **נפרד ונסמך** מספר סודר 12522

. 5.88 . סימני פיסוק unspecified null

Empirical methods in NLP: applications

Example: Shallow parsing

Problem: Given a sentence, segment it into phrases such that no two phrases overlap.

Example (from http://pi0657.kub.nl/cgi-bin/tstchunk/demo.pl):

[NP Russian/JJ and/CC Ukrainian//NNP energy/NN officials/NNS NP] [VP reached/VBD VP] [NP a/DT deal/NN NP] [NP Wednesday/NNP NP] [NP morning/NN NP] ./. [VP ending/VBG VP] [NP a/DT dispute/NN NP] {PNP [Prep over/IN Prep] [NP the/DT price/NN NP] PNP} {PNP [Prep of/IN Prep] [NP Russian/JJ natural/JJ gas/NN NP] PNP} [NP that/WDT NP] [VP caused/VBD VP] [NP shortages/NNS NP] {PNP [Prep in/IN Prep] [NP the/DT Ukraine//NNP NP] PNP} and/CC {PNP [Prep throughout/IN Prep] [NP Western/NNP Europe/NNP NP] PNP} ./.

Empirical methods in NLP: applications

Example: attachment

Problem: Given an ambiguous syntactic structure, determine which of the candidate structures is most likely.

The teacher [wrote [three equations] [on the board]]

The author [wrote [three novels [on the civil war]]]

Empirical methods in NLP: applications

Example: Word sense disambiguation

Problem: Given a text in which each word is associated with several senses, determine the correct sense in the context of each of the words.

brilliant:

- of surpassing excellence; "a brilliant performance"
- brainy: having or marked by unusual and impressive intelligence; "a brilliant solution to the problem"
- characterized by grandeur; "Versailles brilliant court life"
- bright: having striking color; "brilliant tapestries";
- full of light; shining intensely; "a brilliant star"; "brilliant chandeliers"
- bright: clear and sharp and ringing; "the brilliant sound of the trumpets"

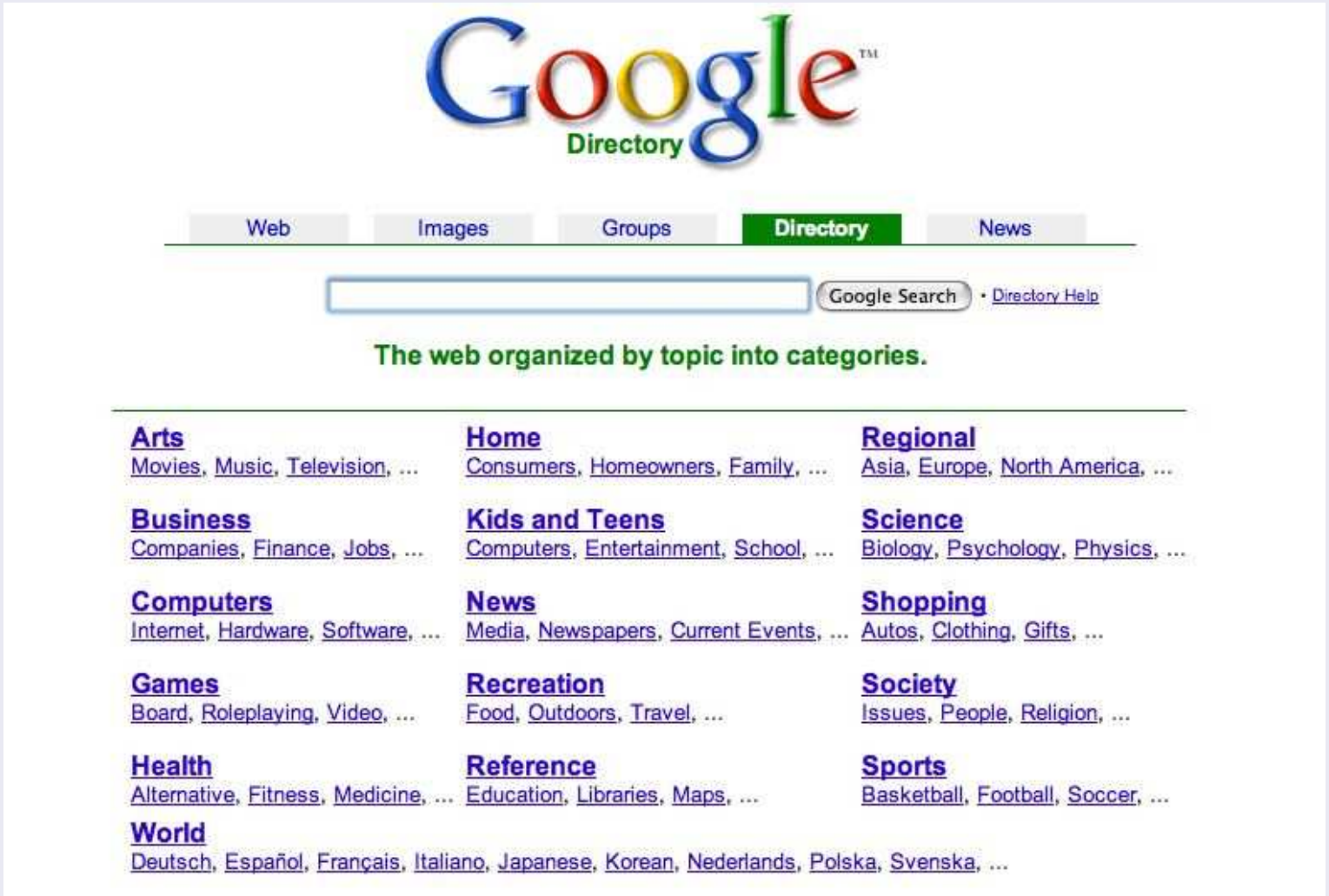
Empirical methods in NLP: applications

Example: Text categorization

Problem: Given a document and a (hierarchical) classification of "topics", determine which topics are addressed by the document.

Empirical methods in NLP: applications

Example: Text categorization



Empirical methods in NLP: roadmap

- Probabilistic models
  - Basic probability theory
  - Bayes Rule
- Collocations, N-grams and the use of corpora
  - N-grams
  - Normalization
  - Maximum-likelihood estimation
  - Data sparseness, smoothing and backoff
- Markov Models
  - Weighted automata and Markov chains
  - Hidden Markov Models
  - decoding and the forward algorithm
  - The Viterbi algorithm
  - Parameter estimation

Empirical methods in NLP: roadmap

- Classification in general
  - Problem representation
  - Training
  - Evaluation
- Classification methods
  - Decision trees
  - Memory-based learning (KNN)
  - Perceptron

Empirical methods in NLP: roadmap

- Spell checking
  - The noisy channel model
  - Bayesian methods
  - Minimum edit distance
- Part of speech tagging
  - HMMs
- Text categorization
- Chunking as a classification task
  - Chunking, shallow parsing, argument detection
  - Transformation-based learning
  - Sequential inference

Basic probability theory

- Probability theory deals with the likelihood of events, where likelihood is established through experiments (trials)
- An event is a subset of the sample space
- A probability distribution distributes a probability mass of 1 throughout the sample space
- Conditional probability:

$$P(A \cap B) = P(B)P(A|B) = P(A)P(B|A)$$

Basic probability theory

- Chain rule:

$$P(A_1 \cap \cdots \cap A_n) = P(A_1)P(A_2|A_1)P(A_3|A_1 \cap A_2) \cdots P(A_n|\cap_{i=1}^{n-1} A_i)$$

- Bayes theorem:

$$P(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A|B)P(B)}{P(A)}$$

- Partition: if B_i is a partition of A, i.e., $A = \cup_i B_i$ and the B_i's are disjoint, then

$$P(A) = \Sigma_i P(A|B_i)P(B_i)$$

N-grams

- Can we guess the next word in a sequence?
- I'd like to make a collect
- Why is this important?
  - Speech recognition, hand-writing recognition, spell checking, machine translation, ...
  - Analytical methods usually fail.
- **N-gram** models predict the next word using the previous $N-1$ words
- In general, this is called **language modeling**.

N-grams

- The use of corpora in natural language processing
- Collocations

Example: Collocations

strong tea
weapons of mass destruction
by and large
אף על פי כן
יצא מגדרו

- Limited compositionality
- Usually very hard to define analytically, but much easier with corpora
- Applications: linguistic research, parsing, generation, machine translation, ...

Frequency: counting words in corpora

- The simplest method for finding collocations is by counting words in a corpus
- However, most bi-grams are uninteresting.

Example: Bi-grams in a corpus

$\#(w_1, w_2)$	w_1	w_2
80,871	of	the
58,841	in	the
26,430	to	the
⋮		
11,428	New	York
10,007	he	said
9,775	as	a
⋮		

Frequency: counting words in corpora

- Solution: apply part-of-speech filtering
- Looking only for the patterns *noun-noun* and *adjective-noun*:

Example: Bi-grams in a corpus

$\#(w_1, w_2)$	w_1	w_2	POS
11428	New	York	A N
7261	United	States	A N
5412	Los	Angeles	N N
3301	last	year	A N
3191	Saudi	Arabia	N N
2699	last	week	A N
2514	vice	president	A N
⋮			
2106	President	Bush	N N
⋮			

Frequency: counting words in corpora

Example: Collocations for synonym selection

w	#(strong, w)	w	#(powerful, w)
support	50	force	13
safety	22	computers	10
sales	21	position	8
opposition	19	men	8
showing	18	computer	8
sense	18	man	7
message	15	symbol	6
defense	14	military	6
gains	13	machine	6
evidence	13	country	6

Simple N-gram models

- Computing the probability of a sequence of words $w = w_1, \ldots, w_n$
- Using the chain rule:

$$\begin{aligned} P(w) &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_n|w_1, \ldots, w_{n-1}) \\ &= \Pi_{k=1}^{n} P(w_k|w_1, \ldots, w_{k-1}) \end{aligned}$$

- However, to compute $P(w_k|w_1, \ldots, w_{k-1})$ reliably we need a huge corpus, and will usually run into **data sparseness** problems
- Solution: make the simplifying assumption that $P(w_k|w_1, \ldots, w_{k-1}) = P(w_k|w_{k-1})$
- Markov chains and higher-order Markov models.

Simple N-gram models: normalization

- For bi-grams:

$$\begin{aligned} P(w_n|w_{n-1}) &= \frac{\#(w_{n-1}w_n)}{\sum_w \#(w_{n-1}w)} \\ &= \frac{\#(w_{n-1}w_n)}{\#(w_{n-1})} \end{aligned}$$

- For general N-grams:

$$P(w_n|w_{n-N+1},\ldots,w_{n-1}) = \frac{\#(w_{n-N+1},\ldots,w_{n-1}w_n)}{\#(w_{n-N+1},\ldots,w_{n-1})}$$

Maximum-likelihood estimation

- **Maximum-likelihood estimation**:

$$P_{ML}(w) = \frac{\#(w)}{N}$$

  where N is the size of the training corpus
- If the observed data are fixed and the space of all possible assignments within a certain distribution is considered, then the maximum likelihood estimate is the choice of parameter values which gives the highest probability to the training corpus

Maximum-likelihood estimation

Example: Maximum-likelihood estimates

Assume a trigram model (using two preceding words to predict the next word). Assume that the two preceding words are comes across. In a given corpus, there were 10 instances of comes across, 8 of which were followed by as, one by more and one by a. The MLE is then

$$\begin{aligned} P(\text{as}) &= 0.8 \\ P(\text{more}) &= 0.1 \\ P(\text{a}) &= 0.1 \\ P(w) &= 0 \text{ for all other words } w \end{aligned}$$

Smoothing

- N-gram models are trained on a (finite) corpus, and hence necessarily some (perfectly grammatical) N-grams are never observed
- It would be useful to assign non-zero probabilities to N-grams which are not observed in the training corpus
- This is usually done by distributing some of the probability mass differently
- **Smoothing**: re-evaluating the probabilities assigned to zero-probability and low-probability events.

Add-one smoothing

- Add one to all counts before normalizing
- For unigram probabilities, using a corpus C of size N:

$$P(w) = \frac{\#(w)}{\sum_{w' \in C} \#(w')} = \frac{\#(w)}{N}$$

- After add-one smoothing:

$$P(w) = \frac{\#(w) + 1}{\sum_{w' \in C} \#(w') + 1} = \frac{\#(w) + 1}{N + V}$$

  where V is the size of the vocabulary.
- For bi-gram probabilities:

$$P(w_n | w_{n-1}) = \frac{\#(w_{n-1} w_n) + 1}{\#(w_{n-1}) + V}$$

Backoff

- Suppose we want to estimate $P(w_n|w_{n-1}w_{n-2})$ but we have no examples of the trigram $w_{n-2}w_{n-1}w_n$.
- **Backoff** methods resort to lower-order models: estimate $P(w_n|w_{n-1}w_{n-2})$ as $P(w_n|w_{n-1})$
- For the trigram case:

$$\hat{P}(w_i|w_{i-2}w_{i-1}) = \begin{cases} P(w_i|w_{i-2}w_{i-1}) & \text{if } \#(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha_1 P(w_i|w_{i-1}) & \text{if } \#(w_{i-2}w_{i-1}w_i) = 0 \\ & \text{and } \#(w_{i-1}w_i) > 0 \\ \alpha_2 P(w_i) & \text{otherwise} \end{cases}$$

- Some models combine backoff with smoothing.

Hidden Markov Models

- Let $X = (X_1, X_2, \ldots)$ be a sequence of random variables. Think of the sequence as a random variable in different points in time
- Assume that the value of the variable is taken from a finite domain $S = \{s_1, \ldots, s_N\}$ of *states*
- X is a **Markov chain** if:

  limited horizon

  $$P(X_{t+1} = s_k | X_1, \ldots, X_t) = P(X_{t+1} = s_k | X_t)$$

  time invariant

  $$P(X_{t+1} = s_k | X_t) = P(X_2 = s_k | X_1)$$

- The probability of X being in state s_k at time $t + 1$ depends only on the value of X in time t. In particular, it is independent of previous values of X or of the time t.

Hidden Markov Models

- A Markov chain can be characterized by:
  - A transition matrix,

$$a_{ij} = P(X_{t+1} = s_j | X_t = s_i)$$

    where $a_{ij} > 0$ for all i, j and $\Sigma_{j=1}^{N} a_{ij} = 1$ for all i; and
  - the initial probabilities,

$$\pi_i = P(X_1 = s_i)$$

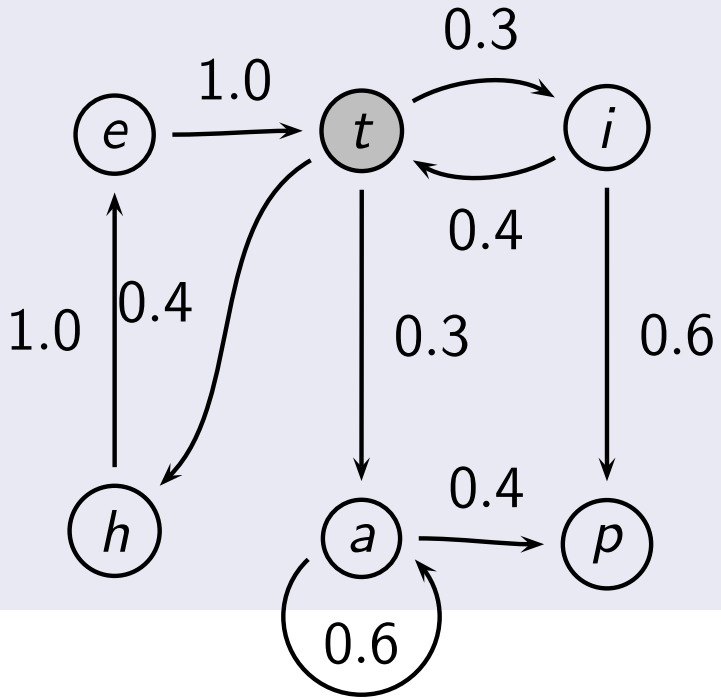
    where $\Sigma_{i=1}^{N} \pi_i = 1$
- Alternatively, Markov chains can be specified as weighted automata.

Weighted automata

- Like a standard finite-state automaton/transducer, with weights on the edges
- The sum of the weights of all edges leaving some node must be 1
- Each path in the automaton is thus assigned a weight
- **Markov chains**, or **visible** Markov models: a weighted automaton in which the input sequence uniquely determines the path (i.e., unambiguous). The state sequence can thus be taken as output
- **Hidden Markov Models** (HMMs): we don't know the state sequence the model passes through, but only some probabilistic function of it.

Weighted automata

Example: Weighted automaton



Markov chains

- Given a Markov chain, the probability of a sequence of states can be calculated directly from the model
- It is the product of the probabilities that occur on the arcs (or in the stochastic matrix):

$$\begin{aligned} P(X_1, \ldots, X_T) &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\cdots P(X_T|X_1, \ldots, X_{T-1}) \\ &= P(X_1)P(X_2|X_1)P(X_3|X_2)\cdots P(X_T|X_{T-1}) \\ &= \pi_{X_1}\Pi_{t=1}^{T-1} a_{X_t X_{t+1}} \end{aligned}$$

Hidden Markov Models

Definition: An HMM is a tuple (Q, Σ, Π, A, B) where:

- Q is a set of states
- Σ is the (output) alphabet
- $\Pi = \{\pi_i \mid i \in Q\}$ is the probability of starting at state $i \in Q$
- $A = \{a_{ij} \mid i, j \in Q\}$ is the state transition probability
- $B = \{b_{ij\sigma} \mid i, j \in Q \text{ and } \sigma \in \Sigma\}$ is the symbol emission probability.

The symbol emitted at time t depends on the states at times t and $t + 1$ (arc-emission HMM).

Hidden Markov Models

- Notation:
  - $O = (o_1, \ldots, o_T)$, where $o_i \in \Sigma$: the *observation*
  - $\mu = (\Pi, A, B)$: the *model*
  - $X = (X_1, \ldots, X_{T+1})$, where $X_i \in Q$: the *state sequence*
- The three fundamental questions for HMM:
  1. Given a model μ, how to compute the likelihood of some observation O, $P(O|\mu)$? (decoding)
  2. Given a model μ and an observation O, which state sequence $(X_1, \ldots, X_{T+1})$ best explains the observation? (classification)
  3. Given an observation sequence O, which model μ best explains the observed data? (parameter estimation)

HMM: decoding

- Given an HMM and an observation O, **decoding** is the process of finding the probability of O
- By partition,

$$P(O|\mu) = \Sigma_X P(O|X,\mu)P(X|\mu)$$

- By definition of X,

$$P(X|\mu) = \pi_{X_1} a_{X_1X_2} a_{X_2X_3} \cdots a_{X_TX_{T+1}}$$

- Similarly, from the definition of the model,

$$\begin{aligned} P(O|X,\mu) &= \Pi_{t=1}^{T} P(o_t|X_t, X_{t+1}, \mu) \\ &= b_{X_1X_2o_1} b_{X_2X_3o_2} \cdots b_{X_TX_{T+1}o_T} \end{aligned}$$

- Putting it all together,

$$P(O|\mu) = \Sigma_{X_1 \cdots X_{T+1}} \pi_{X_1} \Pi_{t=1}^{T} a_{X_tX_{t+1}} b_{X_tX_{t+q}o_t}$$

HMM: decoding

- The **forward** algorithm: a dynamic programming implementation of decoding
- Given a model μ and an observation O, the forward algorithm computes $P(O|\mu)$
- $forward[t, i]$ is the probability of being in state i after seeing the first t observations
- This is the sum of all the probabilities of the paths that lead to state i
- Cashing:

$$forward[t, i] = P(o_1 o_2 \cdots o_{t-1}, X_t = i|\mu)$$

HMM: decoding

- $forward[t, i] = P(o_1 o_2 \cdots o_{t-1}, X_t = i|\mu)$
- Initialization: for all i, $1 \leq i \leq N$,

$$forward[1, i] = \pi_i$$

- Induction: for all t, $1 \leq t \leq T$ and all j, $1 \leq j \leq N$,

$$forward[t + 1, j] = \Sigma_{i=1}^{N} forward[t, i] a_{ij} b_{ijo_t}$$

- Finally,

$$P(O|\mu) = \Sigma_{i=1}^{N} forward[T + 1, i]$$

HMM: decoding

Results can also be cashed working backwards through time.

- $backward[t, i] = P(o_t \cdots o_{T+1}, X_t = i|\mu)$
- Initialization: for all i, $1 \leq i \leq N$,

$$backward[T + 1, i] = 1$$

- Induction: for all t, $1 \leq t \leq T$ and all j, $1 \leq j \leq N$,

$$backward[t, i] = \Sigma_{j=1}^{N} backward[t + 1, j] a_{ij} b_{ijo_t}$$

- Finally, $P(O|\mu) = \Sigma_{i=1}^{N} \pi_i backward[1, i]$
- It can even be shown that

$$P(O|\mu) = \Sigma_{i=1}^{N} backward[t, i] forward[t, i]$$

HMM: classification

- Given an observation sequence, which is the most likely state sequence which generated this observation?
- $$\hat{X} = \arg\max_X P(X|O,\mu) = \arg\max_X P(X,O|\mu)$$
- The **Viterbi** algorithm: use dynamic programming
- For each state, stored the probability of the most probable path leading to this state:

$$\delta[t,j] = \max_{X_1 \cdots X_{t-1}} P(X \cdots X_{t-1}, o_1 \cdots o_{t-1}, X_t = j|\mu)$$

Also, store the node $\Psi[t+1,j]$ of the incoming arc that led to this most probable path.

The Viterbi algorithm

- Initialization: for all j, $1 \leq j \leq N$,

$$\delta[1, j] = \pi_j$$

Induction: for all j, $1 \leq j \leq N$,

$$\delta[t+1, j] = \max_{1 \leq i \leq N} \delta[t, i] a_{ij} b_{ijo_t}$$

and

$$\Psi[t+1, j] = \arg\max_{1 \leq i \leq N} \delta[t, i] a_{ij} b_{ijo_t}$$

- Finally, backtrack by working from the end backwards:

$$\begin{aligned} \hat{X}_{T+1} &= \arg\max_{1 \leq i \leq N} \delta[T+1, i] \\ \hat{X}_t &= \Psi[t+1, \hat{X}_{t+1}] \\ P(\hat{X}) &= \max_{1 \leq i \leq N} \delta[T+1, i] \end{aligned}$$

HMM: parameter estimation

- Given an observation sequence O, which model $\mu = (\Pi, A, B)$ best explains the observation?
- Using maximum likelihood estimation, this is:

$$\arg\max_{\mu} P(O|\mu)$$

- There is no analytic way to choose μ to maximize $P(O|\mu)$
- However, it can be locally maximized by an iterative hill-climbing algorithm:
  - Choose some model (perhaps in random)
  - Calculate the probability of O given this model
  - Observing the calculation, select the state transitions and symbol emissions that were used most
  - increase the probability of those and choose a revised model that gives a higher probability to the observed sequence
- This method is referred to as **training** the model and requires **training data**

Classification: a general framework

Several supervised machine 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.

Example: Classification tasks

Problem	Objects	Categories
POS 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

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

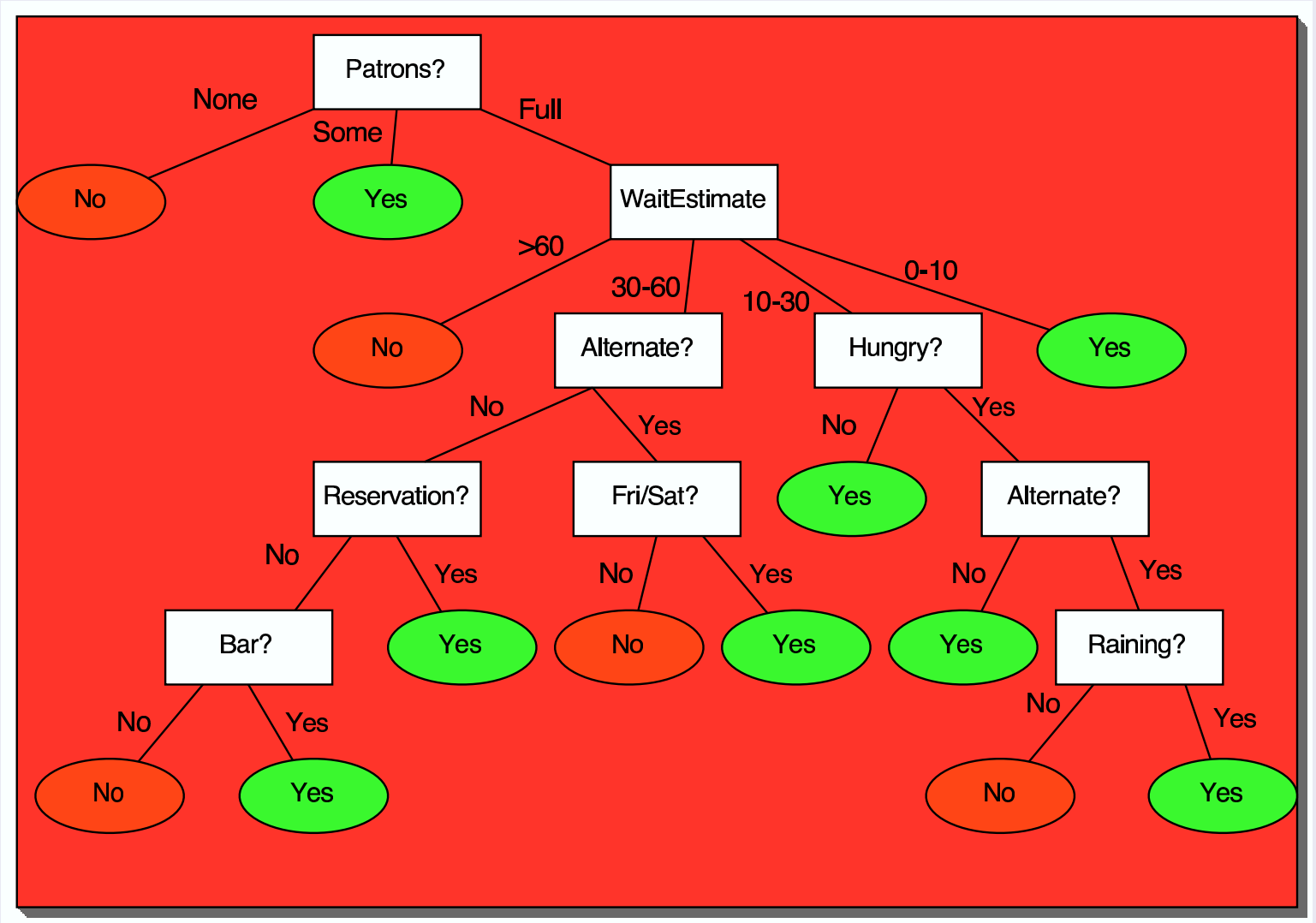
testing: for binary classification, *recall* and *precision*.

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



Decision trees

How to find an appropriate data representation model? This is an art by itself, and **feature engineering** is a major task.

Example: Data representation

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?
Price:	The restaurant's price range ($/$$/$$$)
Raining:	Is it raining outside?
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: Inducing decision trees from examples

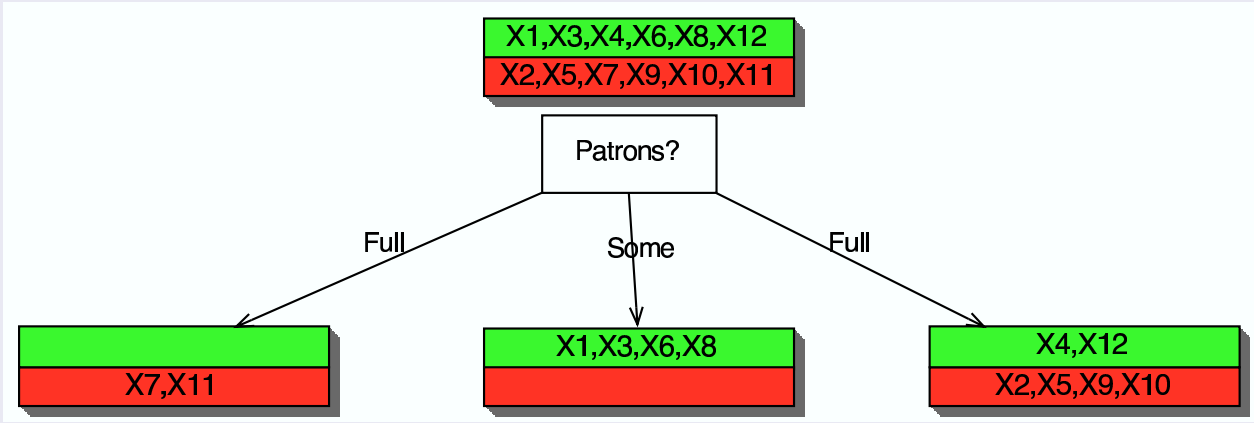
	Alt	Bar	Wknd	Hungry	Pat	$	Rain	Res	Type	Wait	Goal
1	+	-	-	+	Some	$$$	-	+	French	10	Yes
2	+	-	-	+	Full	$	-	-	Thai	60	No
3	-	+	-	-	Some	$	-	-	Burger	10	Yes
4	+	-	+	+	Full	$	-	-	Thai	30	Yes
5	+	-	+	-	Full	$$$	-	+	French	>60	No
6	-	+	-	+	Some	$$	+	+	Italian	10	Yes
7	-	+	-	-	None	$	+	-	Burger	10	No
8	-	-	-	+	Some	$$	+	+	Thai	10	Yes
9	-	+	+	-	Full	$	+	-	Burger	>60	No
10	+	+	+	+	Full	$$$	-	+	Italian	30	No
11	-	-	-	-	Some	$	-	-	Thai	10	No
12	+	+	+	+	Full	$	-	-	Burger	60	Yes

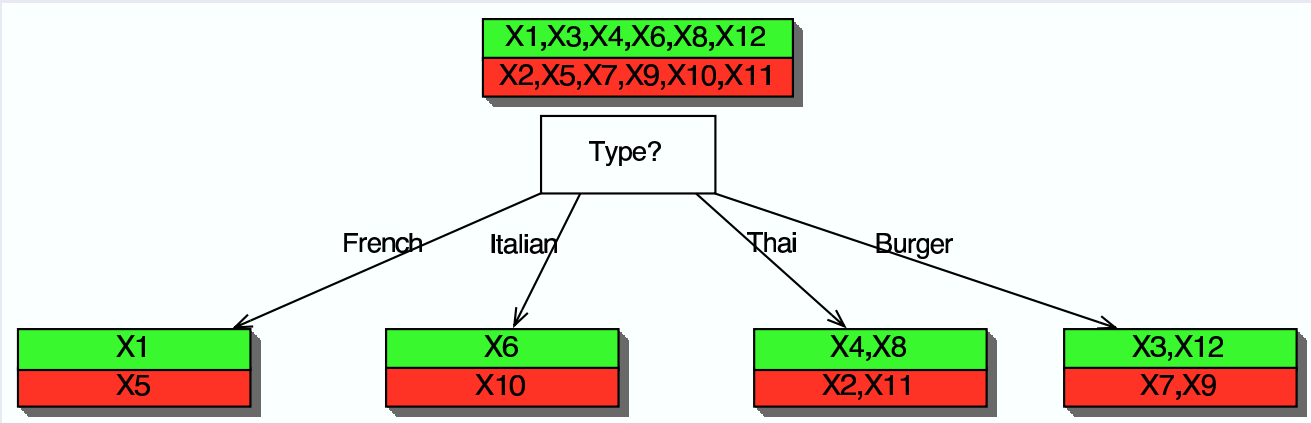
Inducing decision trees from examples

- 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

Example: The contribution of the features **Patrons** and **Type**





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

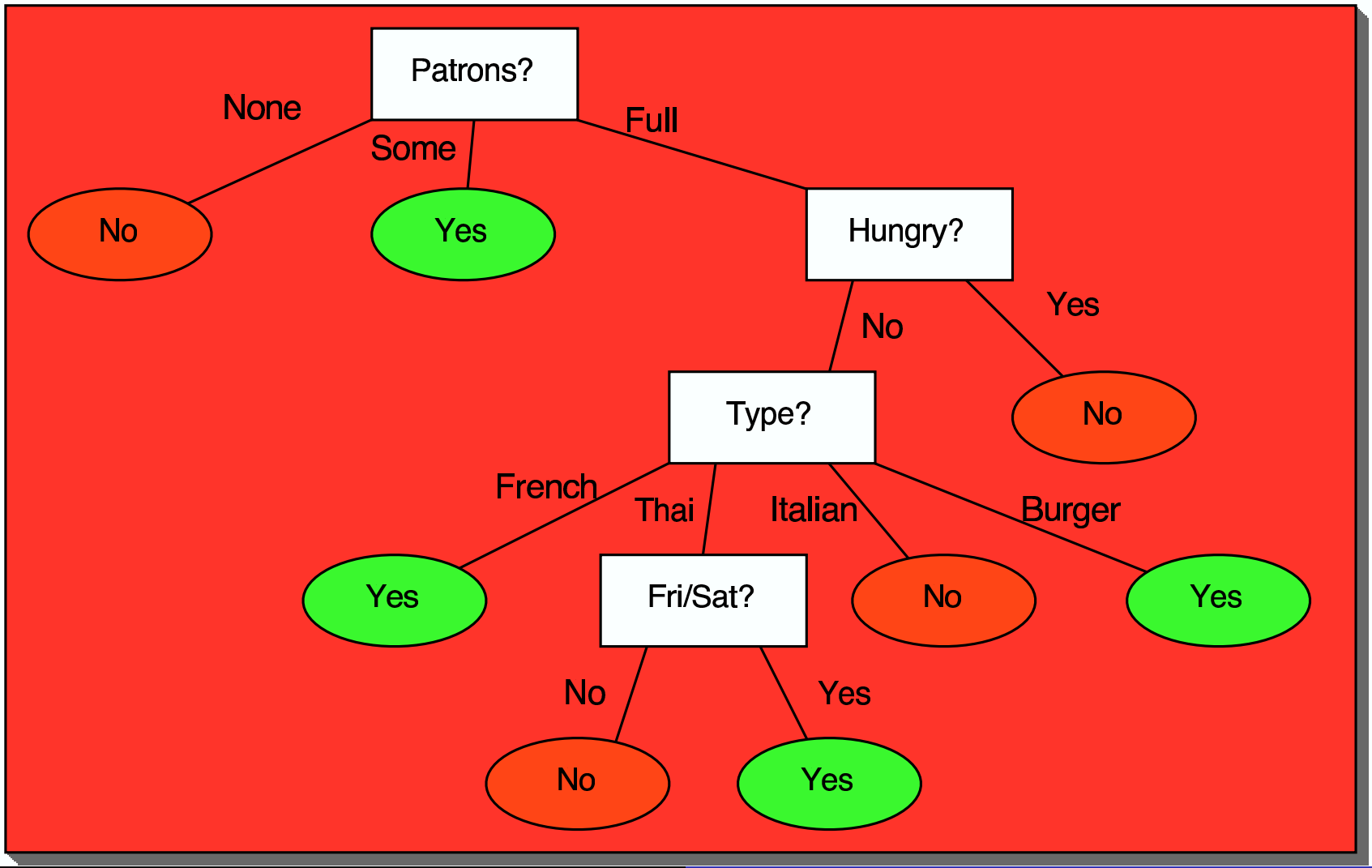
An algorithm for inducing decision trees

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.

The result on the 12 examples

Example: Decision tree



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.

Decision trees for text categorization

Example: A 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.
- 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:

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

- 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 $s_i^j = round(10 \times \frac{1+\log 8}{1+\log 59}) = 7$.

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: 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.

Memory-based learning

- 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?

K-Nearest Neighbors (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.

Memory-based learning: 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.

Perceptron

- **Perceptrons** are a simple example of *hill-climbing* (or *gradient-descent*) algorithms.
- The idea is to try to optimize a function of the data that computes a goodness criterion (such as error rate).
- Data are again represented as numeric vectors. The goal is to learn a *weight vector* $\vec{w}$ and a *threshold* θ such that the weight vector multiplied by the example vector is greater than the threshold for positive examples and lower than the threshold for negative ones.
- In other words, for an example $\vec{x}^j$, the classifier returns "yes" iff $\sum_{i=1}^{K} w_i x_i^j > \theta$.

Training a perceptron

Example: Training perceptrons

$\vec{w} = 0$

$\theta = 0$

while not converged do

for all elements $\vec{x}^j$ in the training set do

if $\vec{x}^j \cdot \vec{w} > \theta$ then $d := 1$ else $d := 0$

if class$(\vec{x}^j = d)$ then continue

else if class$(\vec{x}^j = 1)$ and $d = 0$ then

$\theta := \theta - 1;\ \vec{w} = \vec{w} + \vec{x}^j$

else if class$(\vec{x}^j = 0)$ and $d = 1$ then

$\theta := \theta + 1;\ \vec{w} = \vec{w} - \vec{x}^j$

Perceptron: summary

- Perceptrons are guaranteed to converge when the class of examples is **linearly separable**.
- Several tasks cannot be classified using linear models; sometimes a transformation to another space is useful (e.g., with SVM).
- Perceptrons are useful for simple classification tasks but cannot cope with more complex ones which abound in NLP.

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).

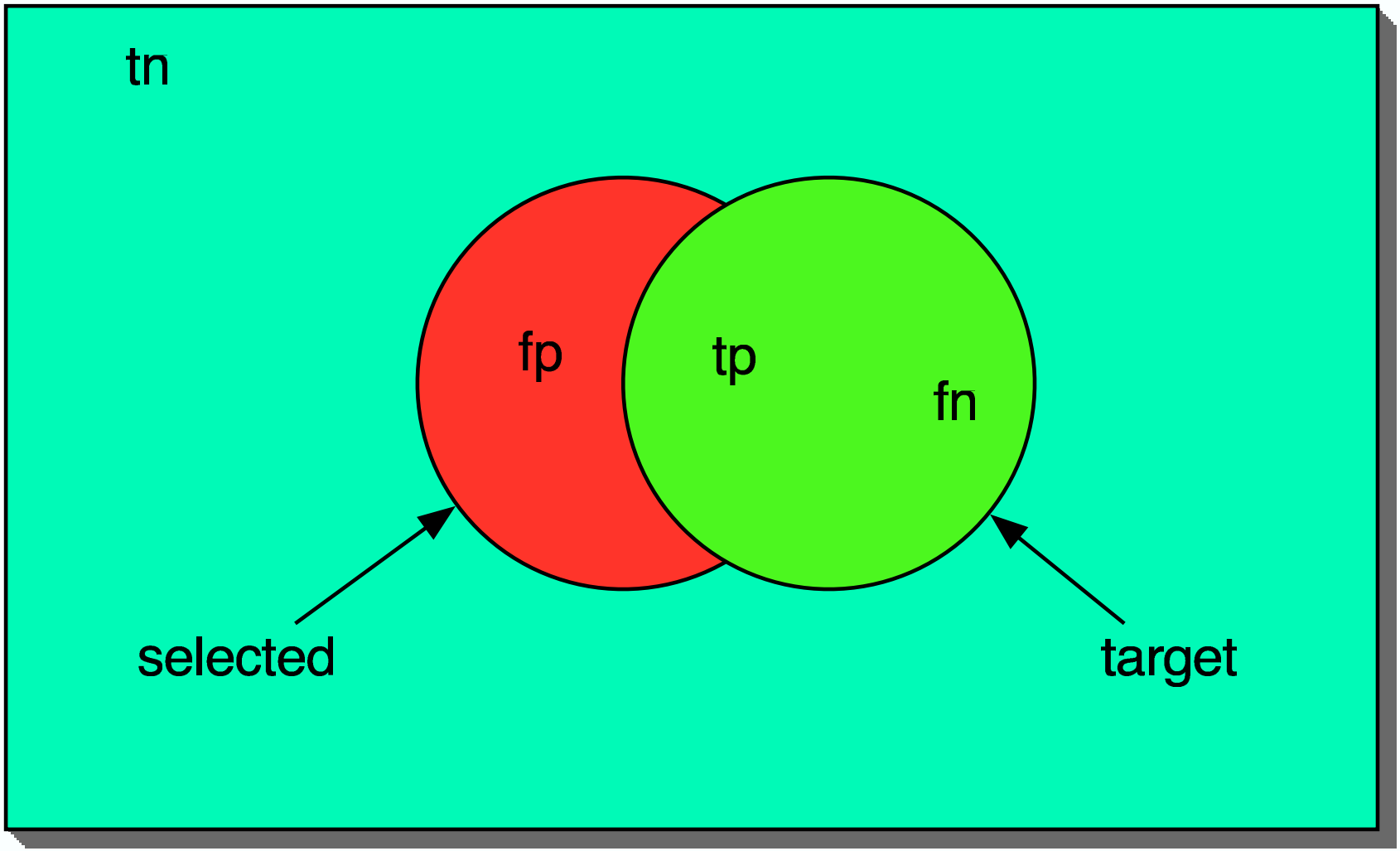
Evaluation

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

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

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

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

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.