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

 ©Copyrighted material

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		

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} ./.

 ©Copyrighted material

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"

 ©Copyrighted material

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

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

 ©Copyrighted material

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
⋮			

 ©Copyrighted material

Frequency: counting words in corpora

Example: Collocations for synonym selection

w	$\#(\text{strong}, w)$	w	$\#(\text{powerful}, w)$
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

 ©Copyrighted material

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:

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

 ©Copyrighted material

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

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

 ©Copyrighted material

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.

 ©Copyrighted material

Add-one smoothing

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

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

- After add-one smoothing:

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

  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.

 ©Copyrighted material

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,

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

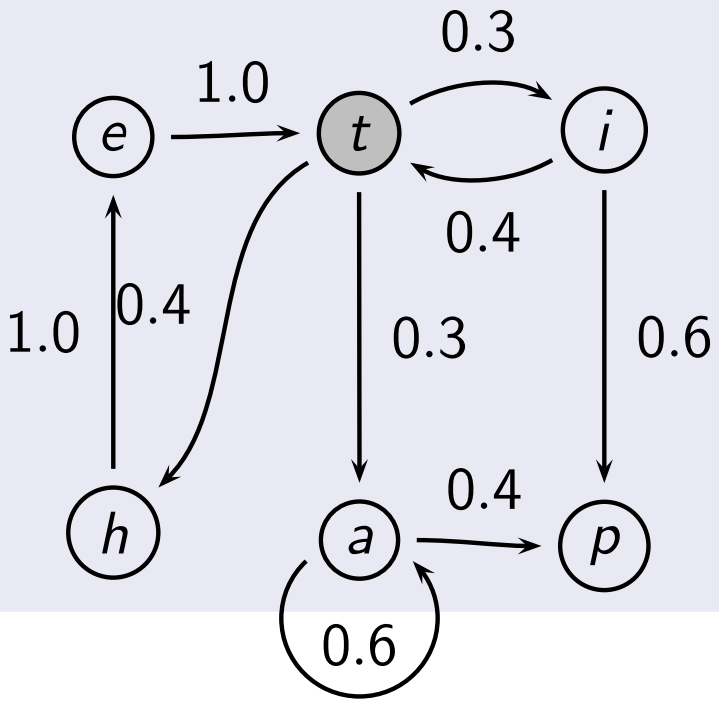
    where $\Sigma_{i=1}^{N} \pi_i = 1$
- Alternatively, Markov chains can be specified as weighted automata.

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)

 ©Copyrighted material

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,

$$P(X|\mu) = \pi_{X_1} a_{X_1 X_2} a_{X_2 X_3} \cdots a_{X_T X_{T+1}}$$

- Similarly, from the definition of the model,

$$\begin{aligned} P(O|X,\mu) &= \Pi_{t=1}^{T} P(o_t|X_t, X_{t+1}, \mu) \\ &= b_{X_1 X_2 o_1} b_{X_2 X_3 o_2} \cdots b_{X_T X_{T+1} o_T} \end{aligned}$$

- Putting it all together,

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

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

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

 ©Copyrighted material

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:

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

 ©Copyrighted material

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**

 ©Copyrighted material

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.

 ©Copyrighted material

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

 ©Copyrighted material

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*.

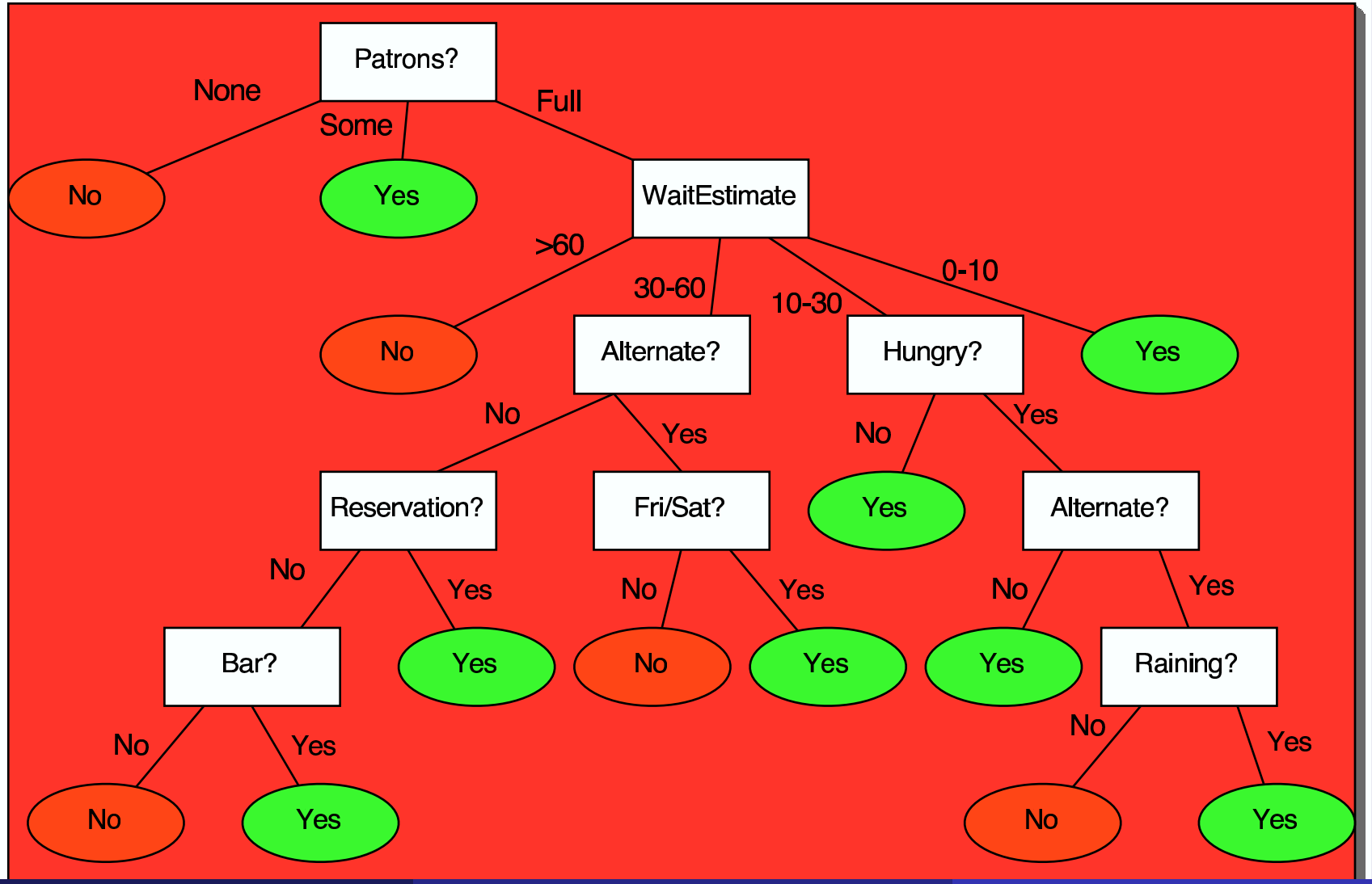
Decision trees

- 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.

 ©Copyrighted material

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)

 ©Copyrighted material

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

 ©Copyrighted material

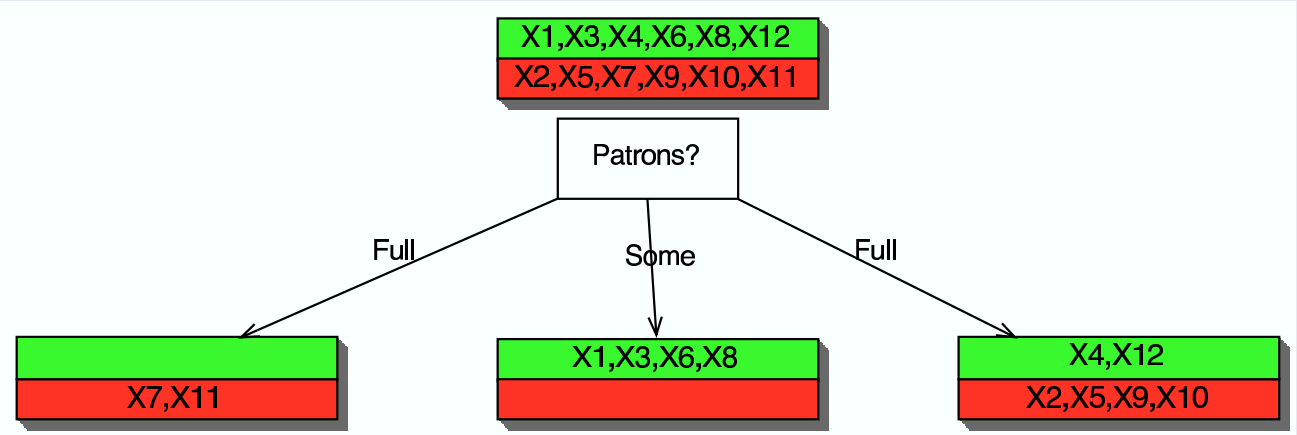
Inducing decision trees from examples

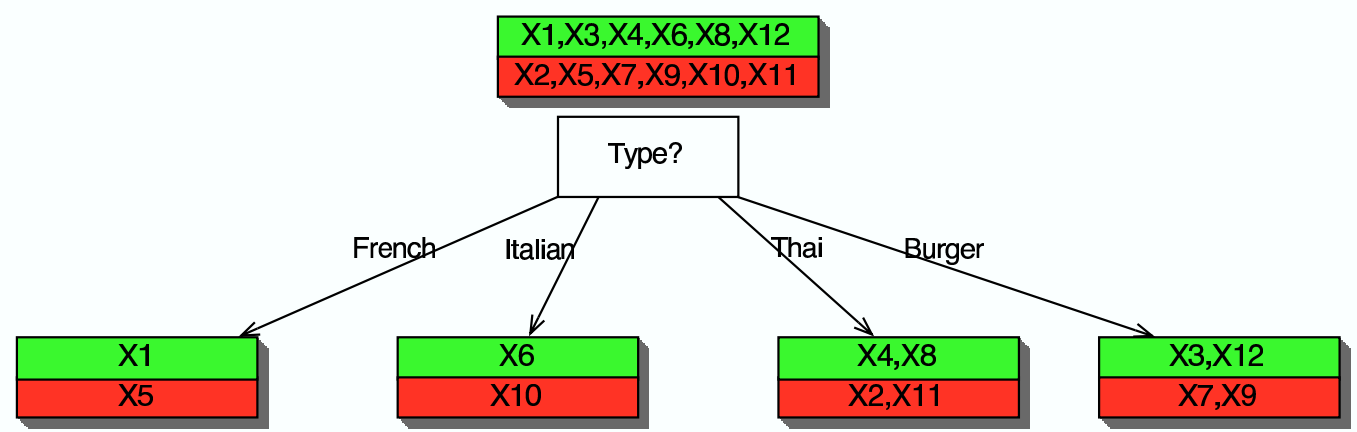
- How to find a decision tree that agrees with the training set?
- This is always possible, since a tree can consist of a unique path from root to leaf for each example. However, such a tree does not generalize to other examples.
- An induced tree must not only agree with all the examples, but also be 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.

 ©Copyrighted material

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

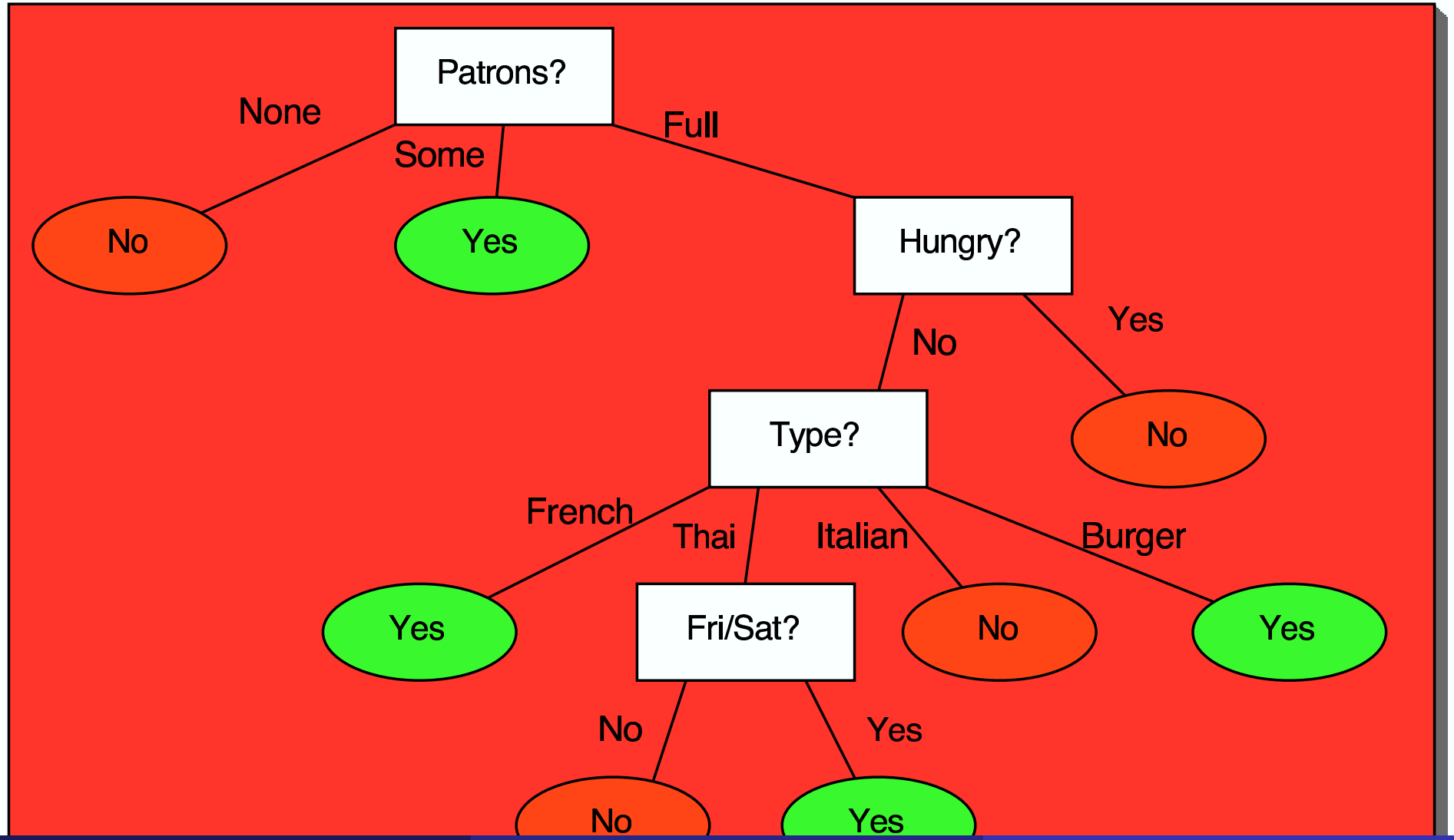
The algorithm: Decide which attribute to use as the first test in the tree. After the first feature splits up the examples, each outcome is a new decision tree learning problem in itself, with fewer examples and one fewer feature. For this recursive problem:

- 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.

 ©Copyrighted material

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.

 ©Copyrighted material

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.

 ©Copyrighted material

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 $\Sigma_{i=1}^{K} w_i x_i^j > \theta$.

 ©Copyrighted material

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

 ©Copyrighted material

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

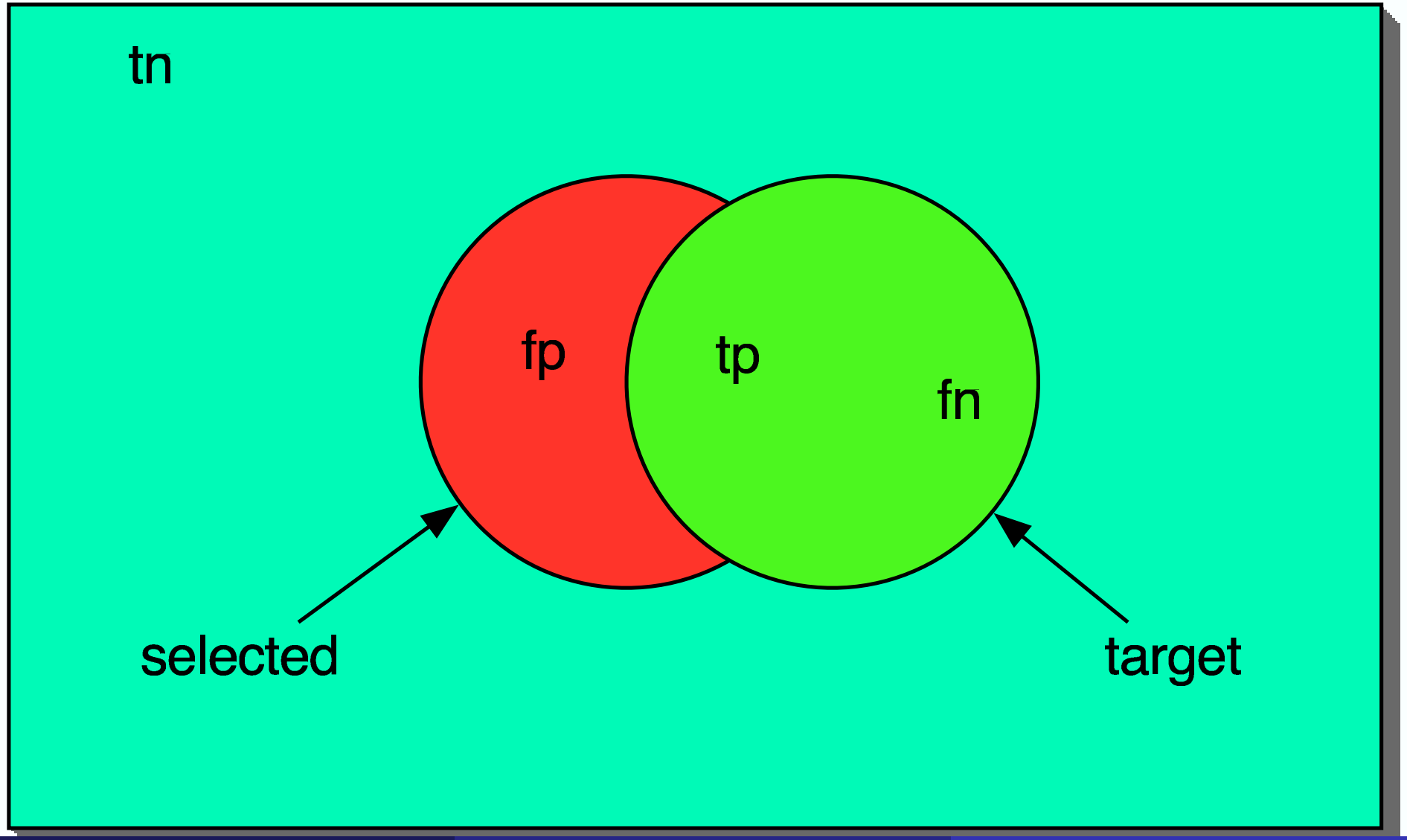
- Borrowing from Information Retrieval, empirical NLP systems are usually evaluated using the notions of **precision** and **recall**.
- Example: Text categorization. A set of documents is given of which a subset is in a particular category (say, "earnings"). The system classifies some other subset of 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

 ©Copyrighted material

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:

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

 ©Copyrighted material

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:

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

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

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

 ©Copyrighted material

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.

Spelling errors

- Assumption: the only spelling errors are a single insertion, deletion, substitution or transposition:

Example: Spelling errors

insertion: the $\longrightarrow$ ther

deletion: the $\longrightarrow$ th

substitution: the $\longrightarrow$ thw

transposition: the $\longrightarrow$ hte

- The **noisy channel model**: the surface form is an instance of the lexical form which has been passed through a noisy communication channel.
- Spelling error detection has to restore the original form from the noisy instance.

Spelling errors

Example: Spelling errors

Error	Correction	Correct	Error	Position	Type
acress	actress	t	ϵ	2	deletion
acress	cress	ϵ	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	ϵ	s	5	insertion
acress	acres	ϵ	s	4	insertion

Bayesian inference

- Spelling correction as a *classification* problem: given an *observation* (misspelled word), determine which of a set of *classes* (correctly spelled words) it belongs to.
- Given a vocabulary V and an observation O, the (estimated) correct word $\hat{w}$ is:

$$\hat{w} = \arg\max_{w \in V} P(w|O)$$

- The problem: how to (directly) compute $P(w|O)$.

Bayesian inference

- Bayes rule:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

- Hence,

$$\begin{aligned}
\hat{w} &= \arg\max_{w \in V} P(w|O) \\
&= \arg\max_{w \in V} \frac{P(O|w)P(w)}{P(O)} \\
&= \arg\max_{w \in V} P(O|w)P(w)
\end{aligned}$$

because $P(O)$ is independent of w, and we are maximizing over all words.

- $P(O|w)$ is the **likelihood**, and $P(w)$ is the **prior probability**.

Spelling errors

- Let O be a misspelled word and V a set of corrections. Then the most likely correction is:

$$\hat{w} = \arg\max_{w \in V} P(O|w)P(w)$$

- The prior probability of each correction, $P(w)$, can be estimated from a corpus by counting how many times w occurs in the corpus $\#(w)$ and normalizing by the size of the corpus, N:

$$P(w) \approx \frac{\#(w)}{N}$$

- Zero counts can cause problems, and hence we **smooth**:

$$P(w) = \frac{\#(w) + 0.5}{N + 0.5V}$$

 ©Copyrighted material

Spelling errors: estimating the prior

Example: Prior probabilities

In a particular corpus of $N = 44$ million words, the following data were observed:

w	$\#(w)$	P(w)
actress	1343	0.0000315
cress	0	0.000000014
caress	4	0.0000001
access	2280	0.000058
across	8436	0.00019
acres	2879	0.000065

Spelling errors: estimating the likelihood

- How to estimate $P(O|w)$, the probability of a typo given the correct word?
- The exact probability depends on various factors (who the typist is, etc.)
- Factors which can be estimated include the identity of the letters (e.g., m is substituted for n because their pronunciation is similar and because they are next to each other on the keyboard) and on context (because they are pronounced similarly, they occur in similar contexts).
- A simplification: using a **confusion matrix** which specifies the number of times one letter was substituted for another in a corpus of errors.

 ©Copyrighted material

Spelling errors: estimating the likelihood

Example: Confusion matrices

$\text{del}(x, y)$: The number of times the characters xy were typed as x

$\text{ins}(x, y)$: The number of times the character x was typed as xy

$\text{sub}(x, y)$: The number of times the character x was typed as y

$\text{trans}(x, y)$: The number of times the characters xy were typed as yx.

$$P(O|w) = \begin{cases} \frac{\text{del}(w_{p-1}, w_p)}{\text{count}(w_{p-1} w_p)} & \text{if deletion} \\ \frac{\text{ins}(w_{p-1}, O_p)}{\text{count}(w_{p-1})} & \text{if insertion} \\ \frac{\text{sub}(O_p, w_p)}{\text{count}(w_p)} & \text{if substitution} \\ \frac{\text{trans}(w_p, w_{p+1})}{\text{count}(w_p w_{p+1})} & \text{if transposition} \end{cases}$$

 ©Copyrighted material

Spelling errors: putting everything together

Example: Ranking of candidate corrections for acress

w	$\#(w)$	$P(w)$	$P(O\|w)$	$P(w)P(O\|w)$	%
actress	1343	0.0000315	0.000117	3.69^{-9}	37
cress	0	0.000000014	0.00000144	2.02^{-14}	0
caress	4	0.0000001	0.00000164	1.64^{-13}	0
access	2280	0.000058	0.000000209	1.21^{-11}	0
across	8436	0.00019	0.0000093	1.77×10^{-9}	18
acres	2879	0.000065	0.0000321	2.09×10^{-9}	21
acres	2879	0.000065	0.0000342	2.22×10^{-9}	23

Results: acres (normalized percentage of 45%), actress (37%).

Minimum edit distance

- Motivation: The previous (over-simplifying) method assumed that each word had only a single error.
- In general, the problem is that of finding the *distance* between two strings.
- **Minimum edit distance**: the minimum number of operations (insert, delete or substitute) needed to transform one string into another.
- **Levenshtein distance**: each operation has the same cost. Variant: insertions and deletions cost 1, substitutions not allowed.
- Variant: assign a cost to each instance of the operations, e.g., using confusion matrices.

Minimum edit distance

Example: Computing minimum edit distance

For two strings, s and t,

$$dist(i,j) = \begin{cases} 0 \text{ if } i = j = 0 \\ i \text{ if } j = 0 \\ j \text{ if } i = 0 \\ \min \left\{ \begin{array}{l} dist(i-1,j) + ins\text{-}cost(t_i), \\ dist(i-1,j-1) + subst\text{-}cost(s_j, t_i) \\ dist(i,j-1) + del\text{-}cost(s_j) \end{array} \right\} \text{ otherwise} \end{cases}$$

Part of speech tagging

- The problem: given a sentence $O = o_1, \ldots, o_n$, assign to each word o_i a correct part of speech (POS) t_i
- Resources:

  Tagset a set of POS tags

  Lexicon a list of words with associated possible POS tags

  Training data a corpus where each word is correctly tagged
- Tagsets for English vary, but most have 40–150 tags
- Why is this task important?
- POS tagging for languages with complex morphology...

 ©Copyrighted material

POS tagging

Example: The Penn Treebank Tagset

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Part of speech tagging

Example: POS ambiguity in two corpora

English		Hebrew	
#Tags	#word types	#analyses	#tokens
1	35,340	1	17,106
2	3,760	2	7,753
3	264	3	4,619
4	61	4	3,272
5	12	5	1,512
6	2	6	648
7	1	7	388
		8	307
		9	128
		10–12	78

 ©Copyrighted material

Markov Model POS tagging

- Assumptions:

  limited horizon: the tag of a word depends only on the tag of the previous word

  time invariant: this dependency does not change over time

  For example, if a pronoun has a probability p to occur after an auxiliary verb in the beginning of a sentence, then this probability does not change in the rest of the sentence.
- Plausibility?
- Notation: subscripts refer to positions in the sentence and in the corpus; superscripts refer to word types in the lexicon and tag types in the tagset.

Markov Model POS tagging

- The states of the Markov model are tags; each time the computation leaves a state, a word is emitted
- The maximum likelihood estimates of some tag t^k following a tag t^j are estimated from the tags' relative frequencies:

$$P(t^k|t^j) = \frac{\#(t^j t^k)}{\#(t^j)}$$

  This constitutes the values of the transition probabilities a_{ij}
- The probability of a word being emitted by a particular state (tag) via maximum likelihood estimation:

$$P(w^l|t^j) = \frac{\#(w^l : t^j)}{\#(t^j)}$$

  This constitutes the values of the emission probabilities b_{ijk}.

 ©Copyrighted material

Markov Model POS tagging

- The best tagging $t_{1..n}$ for a sentence $w_{1..n}$ is:

$$\begin{aligned}\arg\max_{t_{1..n}} P(t_{1..n}|w_{1..n}) &= \arg\max_{t_{1..n}} \frac{P(w_{1..n}|t_{1..n})P(t_{1..n})}{P(w_{1..n})} \\ &= \arg\max_{t_{1..n}} P(w_{1..n}|t_{1..n})P(t_{1..n})\end{aligned}$$

- Assuming (wrongly!) that words are independent of each other, and that a word's identity depends only on its tag,

$$\begin{aligned}P(w_{1..n}|t_{1..n}) &= \Pi_{i=1}^{n} P(w_i|t_{1..n}) \\ &= \Pi_{i=1}^{n} P(w_i|t_i)\end{aligned}$$

- By partitioning and the assumption of limited horizon,

$$\begin{aligned}P(t_{1..n}) &= P(t_n|t_{1..n-1})P(t_{n-1}|t_{1..n-2})\cdots P(t_2|t_1) \\ &= P(t_n|t_{n-1})P(t_{n-1}|t_{n-2})\cdots P(t_2|t_1)\end{aligned}$$

Markov Model POS tagging

- The best tagging $t_{1..n}$ for a sentence $w_{1..n}$ is:

$$\arg\max_{t_{1..n}} P(t_{1..n}|w_{1..n}) \quad = \quad \Pi_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

- A direct evaluation would require an exponential number of multiplications
- Use the Viterbi algorithm for classification:
  - $\delta[t, i]$ is the probability of being at state (tag) i at time (word) t
  - $\Psi[t+1, i]$ is the most likely state (tag) at time (word) t given that at time (word) t we are in state (tag) i

 ©Copyrighted material

Markov Model POS tagging

Example: Viterbi classification

Initialization:

$\delta[., 1] = 1.0$; $\delta[t, 1] = 0.0$ for $t \neq .$;

Induction:

for $i = 1$ to n

for all tags t^j

$\delta[t^j, i+1] = \max_{1 \leq k \leq T}(\delta[t^k, i] P(w_{i+1}|t^j) P(t^j|t^k))$;

$\Psi[t^j, i+1] = \arg\max_{1 \leq k \leq T}(\delta[t^k, i] P(w_{i+1}|t^j) P(t^j|t^k))$;

Termination and path read-out:

$t_{n+1} = \arg\max_{1 \leq j \leq T} \delta[j, n+1]$;

for $j = n$ downto 1

$t_j = \Psi[t_{j+1}, j+1]$;

$P(t_1, \ldots, t_n) = \max_{1 \leq j \leq T} \delta[t^j, n+1]$;

 ©Copyrighted material

Shallow parsing

- **Shallow parsing** consists of identifying the main components of sentences and their heads and determining syntactic relationships among them.
- Problem: Given an input string $O = \langle o_1, \ldots, o_n \rangle$, a **phrase** is a consecutive substring $\langle o_i, \ldots, o_j \rangle$. The goal is, given a sentence, to identify all the phrases in the string.
- A secondary goal is to tag the phrases as Noun Phrase, Verb Phrase etc.
- An additional goal is to identify relations between phrases, such as subject–verb, verb–object etc.
- Question: How can this problem be cast as a classification problem?

 ©Copyrighted material

Text chunking

- Text chunking involves dividing sentences into non-overlapping segments on the basis of fairly simple superficial analysis.
- This is a useful and relatively tractable precursor to full parsing, since it provides a foundation for further levels of analysis, while still allowing complex attachment decisions to be postponed to a later phase.

Deriving chunks from treebank parses

- Annotation of training data can be done automatically based on the parsed data of the Penn Tree Bank
- Two different chunk structure tagsets: one bracketing non-recursive "base NPs", and one which partitions sentences into non-overlapping N-type and V-type chunks

 ©Copyrighted material

"Base NP" chunk structure

- The goal of the "base NP" chunks is to identify essentially the initial portions of non-recursive noun phrases up to the head, including determiners but not including postmodifying prepositional phrases or clauses.
- These chunks are extracted from the Treebank parses, basically by selecting NPs that contain no nested NPs.
- The handling of conjunction follows that of the Treebank annotators as to whether to show separate baseNPs or a single baseNP spanning the conjunction.
- Possessives are treated as a special case, viewing the possessive marker as the first word of a new baseNP, thus flattening the recursive structure in a useful way.

"Base NP" chunk structure

Example: "Base NP" chunk structure

$[_N$ The government $_N]$ has $[_N$ other agencies and instruments $_N]$ for pursuing $[_N$ these other objectives $_N]$.
Even $[_N$ Mao Tse-tung $_N]$ $[_N$'s China $_N]$ began in $[_N$ 1949 $_N]$ with $[_N$ a partnership $_N]$ between $[_N$ the communists $_N]$ and $[_N$ a number $_N]$ of $[_N$smaller , non-communist parties $_N]$.

Partitioning chunks

- In the partitioning chunk experiments, the prepositions in prepositional phrases are included with the object NP up to the head in a single N-type chunk.
- The handling of conjunction again follows the Treebank parse.
- The portions of the text not involved in N-type chunks are grouped as chunks termed V-type, though these "V" chunks include many elements that are not verbal, including adjective phrases.
- Again, the possessive marker is viewed as initiating a new N-type chunk.

Partitioning chunks

Example: Partitioning chunks

$[_N$ Some bankers $_N]$ $[_V$ are reporting $_V]$ $[_N$ more inquiries than usual $_N]$ $[_N$ about CDs $_N]$ $[_N$ since Friday $_N]$.
$[_N$ Indexing $_N]$ $[_N$ for the most part $_N]$ $[_V$ has involved simply buying $_V]$ $[_V$ and then holding $_V]$ $[_N$ stocks $_N]$ $[_N$ in the correct mix $_N]$ $[_V$ to mirror $_V]$ $[_N$ a stock market barometer $_N]$.

Encoding chunking as a tagging problem

- Each word carries both a part-of-speech tag and also a "chunk tag" from which the chunk structure can be derived.
- In the baseNP experiments, the chunk tag set is $\{I, O, B\}$, where words marked I are *inside* some baseNP, those marked O are *outside*, and the B tag is used to mark the leftmost item of a baseNP which immediately follows another baseNP.
- In the partitioning experiments, the chunk tag set is $\{BN, N, BV, V, P\}$, where BN marks the first word and N the succeeding words in an N-type group while BV and V play the same role for V-type groups.

 ©Copyrighted material

Encoding chunking as a tagging problem

- Encoding chunk structure with tags attached to words (rather than inserting bracket markers between words) limits the dependence between different elements of the encoded representation.
- While brackets must be correctly paired in order to derive a chunk structure, it is easy to define a mapping that can produce a valid chunk structure from any sequence of chunk tags; the few hard cases that arise can be handled locally.
- For example, in the baseNP tag set, whenever a B tag immediately follows an O, it must be treated as an I.
- In the partitioning chunk tag set, wherever a V tag immediately follows an N tag without any intervening BV, it must be treated as a BV.

 ©Copyrighted material

Transformation-based learning for Chunking

- Transformational learning begins with some initial "baseline" prediction, which here means a baseline assignment of chunk tags to words.
- Reasonable suggestions for baseline heuristics after a text has been tagged for part-of-speech might include assigning to each word the chunk tag that it carried most frequently in the training set, or assigning each part-of-speech tag the chunk tag that was most frequently associated with that part-of-speech tag in the training.
- Testing both approaches, the baseline heuristic using part-of-speech tags turned out to do better.

 ©Copyrighted material

Rule templates

- Rules can refer to words and to POS tags. Up to three words to the left and right of the target word, and up to two POS tags to the left and right of the target can be addressed.
- A set of 100 rule templates, obtained by the cross product of 20 word-patterns and 5 tag-patterns, was used.
- Then, a variant of Brill's TBL algorithm was implemented.

Results

- BaseNP chunks:

Training	Recall	Precision
Baseline	81.9%	78.2%
50K	90.4%	89.8%
100K	91.8%	91.3%
200K	92.3%	91.8%

- Partitioning chunks:

Training	Recall	Precision
Baseline	60.0%	47.8%
50K	86.6%	85.8%
100K	88.2%	87.4%
200K	88.5%	87.7%

Memory-based shallow parsing

- Shallow parsing consists of discovering the main constituents of sentences (NPs, VPs, PPs) and their heads, and determining syntactic relationships (like subjects, objects or adjuncts) between verbs and heads of other constituents.
- This is an important component of text analysis systems in applications such as information extraction and summary generation.

Memory-based learning: reminder

- A memory-based learning algorithm constructs a classifier for a task by storing a set of examples.
- Each example associates a feature vector (the problem description) with one of a finite number of classes (the solution).
- Given a new feature vector, the classifier extrapolates its class from those of the most similar feature vectors in memory.
- The metric defining similarity can be automatically adapted to the task at hand.

Organization

- Syntactic analysis is carved up into a number of classification tasks.
- These can be segementation tasks (e.g., deciding whether a word or tag is the beginning or the end of an NP) or disambiguation tasks (e.g., deciding whether a chunk is the subject, object or neither).
- Output of one module (e.g., POS tagging or chunking) is used as input by other modules (e.g., syntactic relation assignment).

Algorithms and implementation

- All the experiments use TiMBL.
- Two variants of MBL are used:

  IB1-IG: The distance between a test item and a memory item is the number of features on which they disagree. The algorithm uses information gain to weigh the cost of mismatches. Classification speed is linear in the number of training instances times the number of features.

  IGTree: A decision tree is created with features as tests, ordered according to information gain of features. Classification speed is linear in the number of features times the average branching factor of the tree, which is bound by the average number of values per feature.

 ©Copyrighted material

Experiments

- Two series of experiments:
  - Memory-based NP and VP chunking
  - Subject/object detection using the chunker

Chunking as a tagging task

- Each word is assigned a tag which indicates whether it is inside or outside a chunk:

I_NP	inside a baseNP
O	outside both a baseNP and a baseVP
B_NP	inside a baseNP, but the preceding word is in another baseNP
I_VP	inside a baseVP
B_VP	inside a baseVP, but the preceding word is in another baseVP

- Since baseNPs and baseVPs are non-overlapping and non-recursive, these five tags suffice to unambiguously chunk a sentence.

Tagging example

Example:

[$_{NP}$ Pierre$_{I_NP}$ Vinken$_{I_NP}$ $_{NP}$] ,$_{O}$ [$_{NP}$ 61$_{I_NP}$ years$_{I_NP}$ $_{NP}$] old$_{O}$,$_{O}$ [$_{VP}$ will$_{I_VP}$ join$_{I_VP}$ $_{VP}$] [$_{NP}$ the$_{I_NP}$ board$_{I_NP}$ $_{NP}$] as$_{O}$ [$_{NP}$ a$_{I_NP}$ nonexecutive$_{I_NP}$ director$_{I_NP}$ $_{NP}$] [$_{NP}$ Nov.$_{B_NP}$ 29$_{I_NP}$ $_{NP}$] .

Chunking as a tagging task: experiments

- The features for the experiments are the word and the POS tag (as provided by the Penn Tree Bank) of two words to the left, the target word and one word to the right.
- The baseline is computed with IB1-IG, using as features only the focus word/POS.

Results

- BaseNP chunks:

Method	Recall	Precision
Baseline words	79.7%	76.2%
Baseline POS	82.4%	79.5%
IGTree	93.1%	91.8%
IB1-IG	94.0%	93.7%

- BaseVP chunks:

Method	Recall	Precision
Baseline words	73.4%	67.5%
Baseline POS	87.7%	74.7%
IGTree	94.2%	93.0%
IB1-IG	95.5%	94.0%

 ©Copyrighted material

Subject/object detection

- Finding the subject or object of a verb is defined as a mapping from pairs of words (the verb and the head of the constituent), and a representation of their context, to a class (subject, object or neither).
- A verb can have multiple subjects (in the case of NP coordination) and a word can be the subject of more than one verb (VP coordination).
- The input is POS tagged and chunked.

Example:

[$_{NP}$ My/PRP sisters/NNS $_{NP}$] [$_{VP}$ have/VBP not/RB seen/VBN $_{VP}$] [$_{NP}$ the/DT old/JJ man/NN $_{NP}$] lately/RB ./.

- All chunks are reduced to their heads, defined as the rightmost word of a baseNP or baseVP.

Subject/object detection: features

- The distance, *in chunks*, between the verb and the head
- The number of other baseVPs between the verb and the head
- The number of commas between the verb and the head
- The verb and its POS tag
- The head and its POS tag
- The two left context and one right context words/chunks of the head, represented by the word and its POS tag

Subject/object detection: results

Finding unrestricted subjects and objects is hard:

Method	Together	Subjects	Objects
Heuristic baseline	66.2	65.2	67.7
IGTree	76.2	75.8	76.8
IB1-IG	75.6	76.5	74.0
Unanimous	77.8	77.1	79.0

 ©Copyrighted material

The use of classifiers in sequential inference

- Combination of the outcome of several classifiers in a way that provides a coherent inference that satisfies some constraints.
- Two general approaches to identifying phrase structure: *projection-based Markov models* and *constraint satisfaction with classifiers*.

 ©Copyrighted material

Identifying phrase structure

- Classifiers recognize in the input string local signals which are indicative of the existence of phrases
- Classifiers can indicate that an input symbol is **i**nside or **o**utside a string or that a symbol **o**pens or **c**loses a string.
- The open/close approach has been found more robust and is pursued here.
- The classifiers' outcomes can be combined to determine the phrase, but this combination must satisfy certain constraints for the result to be legitimate.
- Several types of constraints, such as length, order and others, can be formalized and incorporated into the two approaches studied here.

 ©Copyrighted material

Identifying phrase structure

- Two complex phrase identification tasks are defined: base NPs and Subject-Verb patterns.

Example:

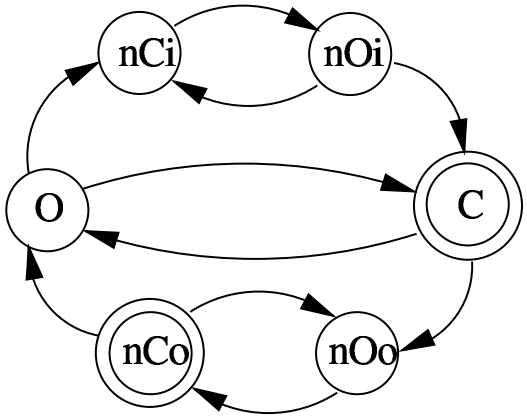
[The theory presented claims] that [the algorithm runs] and performs ...

- Two classifiers are learned for each task, predicting whether word t **o**pens or **c**loses a phrase.

 ©Copyrighted material

Identifying phrase structure

- Each classifier may output two values: open/¬open and close/¬close.
- However, for technical reasons, *three* values are output by each classifier, where the 'not' value is divided according to whether or not the word is inside a phrase.
- Consequently, the values are: **O, nOi, nOo, C, nCi, nCo**.
- The order of these values is constrained according to the following diagram:



Definitions

- The input string is $O = \langle o_1, o_2, \ldots, o_n \rangle$
- A phrase $\pi^{i,j}(O)$ is a substring $\langle o_1, o_{i+1}, \ldots, o_j \rangle$ of O
- $\pi^*(O)$ is the set of all possible phrases of O
- $\pi^{i,j}(O)$ and $\pi^{k,l}(O)$ **overlap**, denoted $\pi^{i,j}(O) \rightleftharpoons \pi^{k,l}(O)$, iff $j \geq k$ and $l \geq i$
- Given a string O and a set Y of classes of phrases, a solution to the phrase identification problem is a set $\{(\pi, y) \mid \pi \in \pi^*(O) \text{ and } y \in Y\}$ such that for all $(\pi_i, y), (\pi_j, y)$, if $i \neq j$ then $\pi_i \not\rightleftharpoons \pi_j$
- We assume that $|Y| = 1$.

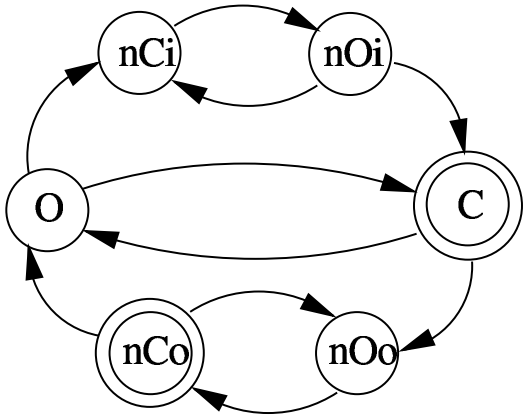
Hidden Markov Model combinator

Reminder: an HMM is a probabilistic finite state automaton consisting of

- A finite set S of states
- A set O of observations
- An initial state distribution $P_1(s)$
- A state-transition distribution $P(s|s')$ for $s, s' \in S$, and
- An observation distribution $P(o|s)$ for $o \in O, s \in S$.
- Constraints can be incorporated into the HMM by constraining the state transition probability distribution. For example, set $P(s|s') = 0$ for cases where the transition from s' to s is not allowed.

 ©Copyrighted material

Hidden Markov Model combinator

- We assume that we have local signals which indicate the state. That is, classifiers are given with states as their outcome.
- Formally, we assume that $P_t(s|o_y)$ is given, where t is a time step in the sequence.
- 

- Constraints on state transitions do not have to be stated explicitly; they can be recovered from training data.

Hidden Markov Model combinator

- Instead of estimating the observation probability $P(o|s)$ directly from training data, it is computed from the classifiers' output:

$$P_t(o_t|s) = \frac{P_t(s|o_t) \times P_t(o_t)}{P_t(s)}$$

- $$P_t(s) = \Sigma_{s' \in S} P(s|s') \times P_{t-1}(s')$$

  where $P_1(s)$ and $P(s|s')$ are the standard HMM distributions
- $P_t(o_t)$ can be treated as constant since the observation sequence is fixed for all compared sequences.
- The Viterbi algorithm can be used to find the most likely state sequence for a given observation.

Projection based Markov Model combinator

- In standard HMMs, observations are allowed to depend only on the current state; no long-term dependencies can be modeled.
- Similarly, constraint structure is restricted by having a stationary probability distribution of a state given the previous state.
- In PMM, these limitations are relaxed by allowing the distribution of a state to depend, in addition to the previous state, on the observation.
- Formally, the independence assumption is:

$$P(s_t|s_{t-1},\ldots,s_1,o_{t-1},\ldots,o_1) = P(s_t|s_{t-1},o_t)$$

 ©Copyrighted material

Projection based Markov Model combinator

- Given an observation sequence O, the most likely state sequence S given O is obtained by maximizing

$$\begin{aligned} P(S|O) &= \Pi_{t=2}^{n}[P(s_t|s_1,\ldots,s_{t-1},o)]P_1(s_1|o) \\ &= \Pi_{t=2}^{n}[P(s_t|s_{t-1},o_t)]P_1(s_1|o_1) \end{aligned}$$

- In this model the classifiers decisions are incorporated in the terms $P(s|s',o)$ and $P_1(s|o)$. The classifiers take into account not only the current input symbol but also the previous state. The hope is that these new classifiers perform better because they are given more information.

 ©Copyrighted material

Constraint satisfaction based combinator

- A boolean constraint satisfaction problem consists of a set of n variables $V = \{v_1, \ldots, v_n\}$, each ranging over values in a domain D_i (here, 0/1).
- A constraint is a relation over a subset of the variables, defining a set of "global" possible assignments to the referred variables.
- A solution to a CSP is an assignment that satisfies all the constraints.
- The CSP formalism is extended to deal with probabilistic variables; the solution now has to minimize some cost function. Thus, each variable is associated with a cost function $c_i : D_i \rightarrow \Re$.

 ©Copyrighted material

Constraint satisfaction with classifiers

- Given an input $O = \langle o_1, \ldots, o_l \rangle$, let $V = \{v_{i,j} \mid 1 \leq i \leq j \leq l\}$. Each variable $v_{i,j}$ corresponds to a potential phrase $\pi^{i,j}(O)$.
- Associate with each variable $v_{i,j}$ a cost function $c_{i,j}$.
- Constraints can now be expressed as boolean formulae. For example, the constraint that requires that no two phrases overlap is expressed as:

$$\bigwedge_{\pi^{a,b} \rightleftharpoons \pi^{c,d}} (\neg v_{a,b} \vee \neg v_{c,d})$$

- The solution is an assignment of 0/1 to variables which satisfies the constrains and, in addition, minimizes the overall cost

$$\Sigma_{i=1}^{n} c_i(v_i)$$

Constraint satisfaction with classifiers

- In general, the corresponding optimization problem is NP-hard.
- In the special case where costs are in $[0, 1]$, a solution which is at most twice the optimal can be found efficiently.
- For the specific case of non-overlapping phrase identification, the problem can be solved efficiently using a graph representation of the constraints (the problem reduces to finding a shortest path in a weighted graph).
- What is left is determining the cost function.

 ©Copyrighted material

Constraint satisfaction with classifiers: cost function

- It can be shown that in order to maximize the number of correct phrases, each phrase has to be assigned a cost that is minus the probability of the phrase being correct:

$$c_{i,j}(v_{i,j}) = \begin{cases} -p_{i,j} & \text{if } v_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases}$$

where $p_{i,j}$ is the probability that the phrase $\pi^{i,j}$ is correct.

Constraint satisfaction with classifiers: cost function

- Assuming independence between symbols in a phrase, and assuming that the important part of a phrase are only its beginning and end words,

$$p_{i,j} = P_i^O(O) \times P_j^C(C)$$

  where $P_i^O(O)$ is the probability that the first symbol o_i in the phrase is actually the beginning of a phrase, and $P_j^C(C)$ is the probability that the last symbol o_j of the phrase is actually the end of a phrase.
- These two probabilities are supplied by the classifiers.

Results

	NP		SV	
Method	POS only	POS + word	POS only	POS + word
HMM	90.64	92.89	64.15	77.54
PMM	90.61	92.98	74.98	86.07
CSCL	90.87	92.88	85.36	90.09

Conclusion: what have we done

- Morphology
- Finite-state technology
- Syntax
- Context-free grammars
- Parsing
- The complexity of natural languages
- Unification grammars
- Empirical methods in NLP

Current work

From the Call for Papers of EACL-06:

- phonetics, phonology and morphology;
- word segmentation, tagging and chunking;
- syntax, semantics and grammars;
- pragmatics, discourse and dialogue;
- the lexicon and ontologies;
- parsing and grammatical formalisms;
- generation, text planning and summarization;
- language modeling, spoken language recognition and understanding;

Current work

- mathematical models of language;
- information retrieval, text categorisation, question answering, and information extraction;
- paraphrasing and textual entailment;
- machine learning for natural language;
- multi-lingual processing, machine translation and translation aids;
- multi-modal and natural language interfaces and dialogue systems;
- language-oriented applications, tools and resources;
- evaluation methodology.

Current work

Titles of some EACL-06 accepted papers:

- A Clustering Approach for the Nearly Unsupervised Recognition of Nonliteral Language
- A Comparison of Syntactically Motivated Word Alignment Spaces
- A Machine Learning Approach to Extract Temporal Information from Texts in Swedish and Generate Animated 3D Scenes
- A Probabilistic Answer Type Model
- Adaptive Transformation-based Learning for Improving Dictionary Tagging
- Addressee Identification in Face-to-Face Meetings

 ©Copyrighted material

Current work

- Automatic Detection of Nonreferential It in Spoken Multi-Party Dialog
- Automatic Segmentation of Multiparty Dialogue
- Automatically Constructing a Lexicon of Verb Phrase Idiomatic Combinations
- Efficient MT Evaluation Using Block Movements
- Data-driven Generation of Emphatic Facial Displays
- Exploiting Shallow Linguistic Information for Relation Extraction from Biomedical Literature
- Multi-Document Summarization of Evaluative Text
- Online Learning of Approximate Dependency Parsing Algorithms
- Parsing Arabic Dialects

Local projects

- Unification grammars: their mathematical and computational properties. Focus on grammar engineering
- Hebrew resources. Focus on morphological disambiguation
- Hebrew to English machine translation

Next semester

- Laboratory in Natural Language Processing