Part-of-speech Tagging & Hidden Markov Model Intro

Lecture #10

Introduction to Natural Language Processing

CMPSCI 585, Fall 2004

University of Massachusetts Amherst

Andrew McCallum

Today's Main Points

- PA#2 commentary and stories
- Part-of-speech tagging
  - What is it? Why useful?
- (Statistical) recipe for NLP problems
- Hidden Markov Models
  - Definition
  - Generative Model
  - Dynamic programming with Viterbi algorithm

Administration

- Both HW#3 and PA#3 going out today.
- HW#3 due first.
- We will have a quiz on HMMs, etc before the midterm. "Practice midterm" :-)

Grammatical categories: parts-of-speech

- Nouns: people, animals, concepts, things
- Verbs: expresses action in the sentence
- Adjectives: describe properties of nouns
- The { sad / intelligent / green / fat / … } one is in the corner.

"Substitution test"

The Part-of-speech Tagging Task

Input: `the lead paint is unsafe`

Output: `the/Det lead/N paint/N is/V unsafe/Adj`

- Uses:
  - text-to-speech (how do we pronounce "lead"?)
  - can differentiate word senses that involve part of speech differences (what is the meaning of "interest")
  - can write regexps like `Det Adj* N*` over the output (for filtering collocations)
  - can be used as simpler "backoff" context in various Markov models when too little is known about a particular history based on words instead.
  - preprocessing to speed up parser (but a little dangerous)
  - tagged text helps linguists find interesting syntactic constructions in texts ("ssh" used as a verb)

Tagged Data Sets

- Brown Corpus
  - Designed to be a representative sample from 1961
    - news, poetry, …
  - 87 different tags
- Claws5 "C5"
  - 62 different tags
- Penn Treebank
  - 45 different tags
  - Most widely used currently

Part-of-speech tags, examples

PART-OF-SPEECH	TAG	EXAMPLES
Adjective	JJ	happy, bad
Adjective, comparative	JJR	happier, worse
Adjective, cardinal number	CD	3, fifteen
Adverb	RB	often, particularly
Conjunction, coordination	CC	and, or
Conjunction, subordinating	IN	although, when
Determiner	DT	this, each, other, the, a, some
Determiner, postdeterminer	JJ	many, same
Noun	NN	aircraft, data
Noun, plural	NNS	women, books
Noun, proper, singular	NNP	London, Michael
Noun, proper, plural	NNPS	Australians, Methodists
Pronoun, personal	PRP	you, we, she, it
Pronoun, question	WP	who, whoever
Verb, base present form	VBP	take, live

Closed, Open

- Closed Set tags
  - Determiners
  - Prepositions
  - …
- Open Set tags
  - Noun
  - Verb

Why is this such a big part of NLP?

Input: `the lead paint is unsafe`

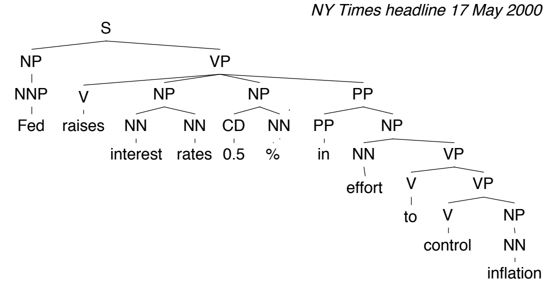
Output: `the/Det lead/N paint/N is/V unsafe/Adj`

- The first statistical NLP task
- Been done to death by different methods
- Easy to evaluate (how many tags are correct?)
- Canonical finite-state task
  - Can be done well with methods that look at local context
  - (Though should "really" do it by parsing!)

Ambiguity in Language

`Fed raises interest rates 0.5%`
`in effort to control inflation`

NY Times headline 17 May 2000



Part of speech ambiguities

Part-of-speech ambiguities

		VB					
	VBZ	VBZ	VBZ				
NNP	NNS	NNS	NNS	CD	NN		
Fed	raises	interest	rates	0.5	%	in effort to control inflation	

Degree of Supervision

- Supervised: Training corpus is tagged by humans
- Unsupervised: Training corpus isn't tagged
- Partly supervised: Training corpus isn't tagged, but you have a dictionary giving possible tags for each word
- We'll start with the supervised case and move (in later classes) to decreasing levels of supervision.

Current Performance

Input: `the lead paint is unsafe`

Output: `the/Det lead/N paint/N is/V unsafe/Adj`

- Using state-of-the-art automated method, how many tags are correct?
  - About 97% currently
  - But baseline is already 90%
    - Baseline is performance of simplest possible method:
    - Tag every word with its most frequent tag
    - Tag unknown words as nouns

Recipe for solving an NLP task

Input: `the lead paint is unsafe` Observations

Output: `the/Det lead/N paint/N is/V unsafe/Adj` Tags

1) **Data**: Notation, representation
2) **Problem**: Write down the problem in notation
3) **Model**: Make some assumptions, define a parametric model (often generative model of the data)
4) **Inference**: How to search through possible answers to find the best one
5) **Learning**: How to estimate parameters
6) **Implementation**: Engineering considerations for an efficient implementation

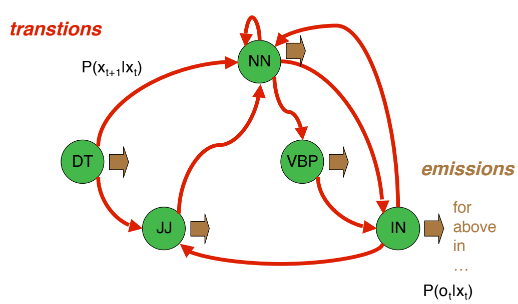
Work out several alternatives on the board…

(Hidden) Markov model tagger

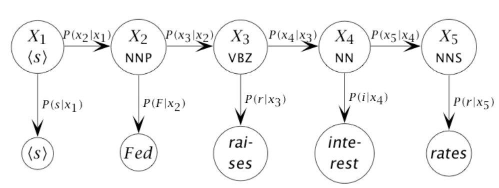
- View sequence of tags as a Markov chain. Assumptions:
  - Limited horizon $P(x_{t+1}|x_1, ...x_t) = P(x_{t+1}|x_t)$
  - Time invariant (stationary) $P(x_{t+1}|x_t) = P(x_2|x_1)$
  - We assume that a word's tag only depends on the previous tag (limited horizon) and that his dependency does not change over time (time invariance)
  - A state (part of speech) generates a word. We assume it depends only on the state.

$$P(o_t|x_1, ...x_T, o_1, ...o_{t-1}) = P(o_t|x_t)$$

HMM as Finite State Machine



HMM as Bayesian Network



- Top row is unobserved states, interpreted as POS tags
- Bottom row is observed output observations (words)

Applications of HMMs

- NLP
  - Part-of-speech tagging
  - Word segmentation
  - Information extraction
  - Optical Character Recognition (OCR)
- Speech recognition
  - Modeling acoustics
- Computer Vision
  - gesture recognition
- Biology
  - Gene finding
  - Protein structure prediction
- Economics, Climatology, Communications, Robotics…

Probabilistic Inference in an HMM

Three fundamental questions for an HMM:

1) Compute the probability of a given observation sequence, when tag sequence is hidden (language modeling)
2) Given an observation sequence, find the most likely hidden state sequence (tagging) **DO THIS NOW**
3) Given observation sequence(s) and a set of states, find the parameters that would make the observations most likely (parameter estimation)

(One) Standard HMM formalism

- *(X, O, x_s, A, B) are all variables. Model* μ = *(A, B)*
- *X* is state sequence of length T; *O* is observation seq.
- x_s is a designated start state (with no incoming transitions). (Can also be separated into π as in book.)
- *A* is matrix of transition probabilities (each row is a conditional probability table (CPT)
- *B* is matrix of output probabilities (vertical CPTs)

$$P(X, O|\mu) = \prod_{t=1}^{T} a[x_t|x_{t-1}]\ b[o_t|x_t]$$

- HMM is a probabilistic (nondeterministic) finite state automaton, with probabilistic outputs (from vertices, not arcs, in the simple case)
- Book describes more complex "outputs on arcs" formulation.

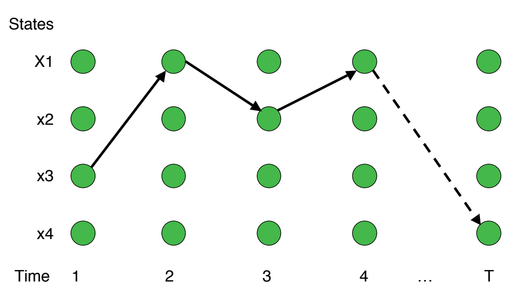
Most likely hidden state sequence

- Given O = (o_1,…,o_T) and model μ = (A,B)
- We want to find

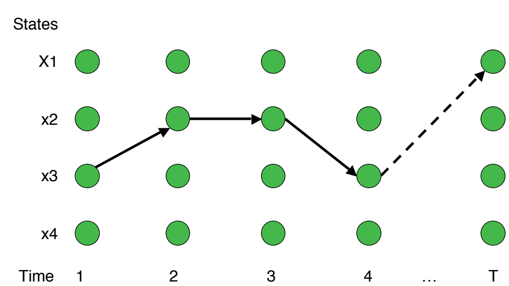
$$\arg\max_X P(X|O,\mu) = \arg\max_X \frac{P(X,O|\mu)}{P(O|\mu)} = \arg\max_X P(X,O|\mu)$$

- P(O|X, μ) = b[x_1|o_1] b[x_2|o_2] … b[x_T|o_T]
- P(X| μ) = a[x_1|x_2] a[x_2|x_3] … a[x_{T-1}|x_T]
- P(O,X| μ) = P(O|X, μ) P(X| μ)
- arg $\max_X$ P(O,X| μ) = arg max x_1, x_2,… x_T
- Problem: arg max is exponential in sequence length!

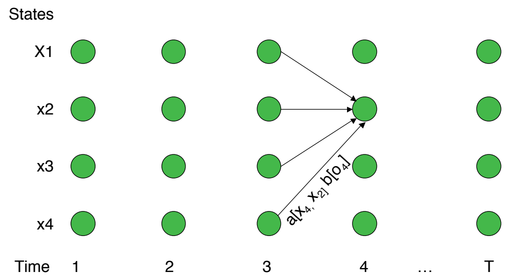
Representation for Paths: Trellis



Representation for Paths: Trellis



Representation for Paths: Trellis



$d_i(t)$ = Probability of most likely path that ends at state *i* at time *t*.

Finding Probability of Most Likely Path using Dynamic Programming

- Efficient computation of max over all states
- Intuition: Probability of the first *t* observations is the same for all possible *t+1* length sequences.
- Define forward score:

$$\delta_i(t) = \max_{x_1 \ldots x_{t-1}} P(o_1 o_2 \ldots o_t, x_1 \ldots x_{t-1}, x_t = i | \mu)$$

$$\delta_j(t+1) = \max_{i=1..N} \delta_i(t) a[x_j|x_i]\ b[o_{t+1}|x_j]$$

- Compute it recursively from the beginning
- (Then must remember best paths to get arg max.)

Finding the Most Likely State Path with the Viterbi Algorithm

[Viterbi 1967]

- Used to efficiently find the state sequence that gives the highest probability to the observed outputs
- Maintains two dynamic programming tables:
  - The probability of the best path (max)

$$\delta_j(t+1) = \max_{i=1..N} \delta_i(t) a[x_j|x_i]\ b[o_{t+1}|x_j]$$

  - The state transitions of the best path (arg)

$$\psi_j(t+1) = \arg \max_{i=1..N} \delta_i(t) a[x_j|x_i]\ b[o_{t+1}|x_j]$$

- Note that this is different from finding the most likely tag for each time *t*!

Viterbi Recipe

- Initialization

$$\delta_j(0) = 1 \text{ if } x_j = x_s. \quad \delta_j(0) = 0 \text{ otherwise.}$$

- Induction

$$\delta_j(t+1) = \max_{i=1..N} \delta_i(t) a[x_j|x_i]\ b[o_{t+1}|x_j]$$

  Store backtrace

$$\psi_j(t+1) = \arg \max_{i=1..N} \delta_i(t) a[x_j|x_i]\ b[o_{t+1}|x_j]$$

- Termination and path readout

$$\hat{x}_T = \arg \max_{i=1..N} \delta_i(T)$$

$$\hat{x}_t = \psi_{\hat{x}_{t+1}}(t+1)$$

Probability of entire best seq.

$$P(\hat{X}) = \max_{i=1..N} \delta_i(T)$$