CS 181: Natural Language Processing

Lecture 7: PoS Tagging

Kim Bruce
Pomona College
Spring 2008

Disclaimer: Slide contents borrowed from many sources on web!

PoS Taggers

- Rule-Based Tagger - English Two Level Analysis ✔ *Done last time*
- Stochastic Tagger: Hidden Markov Model
- Transformation-based Tagger

Stochastic Taggers

- Based on probability of tag occurring, given other info.
- Requires training corpus.
- No probabilities for words not in corpus.
- Use distinct testing corpus.
- Simplest: choose most frequent tag associated w/word in training corpus.

General Recipe

- Data: Decide notation, representation
- Problem: Write down in notation
- Model: Make assumptions & define parametric model
- Inference: How to search through possible answers for best answers?
- Learning: How to estimate parameters
- Implementation: Engineering trade-offs for efficient implementation.

HMM Tagger

- Find tag sequence t_1^n to maximize $P(t_1^n | w_1^n)$.

  $$\hat{t}_1^n = \underset{t_1^n}{argmax}\ P(t_1^n|w_1^n)$$

- Using Bayes' rule:

  $$\hat{t}_1^n = \underset{t_1^n}{argmax}\frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

- Ignore denominator -- always same
- Still too complex ...

Simplify

- Assume probability of word depends only on its own tag:

  $$P(w_1^n|t_1^n) \equiv \prod_{i=1}^{n} P(w_i|t_i)$$

- Bigram assumption:

  $$P(t_1^n) \equiv \prod_{i=1}^{n} P(t_i|t_{i-1})$$

- Thus $\hat{t}_1^n \equiv \underset{t_1^n}{argmax} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$
- Makes it finite state!

Examples

- Secretariat is expected to race tomorrow.
- Consider two possible taggings for entire sentence:
  - Secretariat/NNP is/BEZ expected/VBZ to/TO race/VB tomorrow/NR
  - Secretariat/NNP is/BEZ expected/VBZ to/TO race/NN tomorrow/NR
- If use formulas, only differ on few terms
  - if race is VB: P(race | VB), P(VB | TO), P(NN | VB)
  - if race is NN: P(race | NN) P(NN | TO), P(NN | NN)

Data from Brown Corpus

- *Estimate P(V | T) as C(TV)/C(T)*
- Tagging race as VB:
  - P(VB | TO) = .83
  - P(race | VB) = .00012
  - P(NR | VB) = .0027

  = .00000027
- Tagging race as NN:
  - P(NN | TO) = .00047
  - P(race | NN) = .0057
  - P(NR | NN) = .0012

  = .00000000032

Freq w/Simplified Tags

Bigram(Ti, Tj)	Count(i, i + 1)	Prob(Tj|Ti)
<s>,ART	213	0.71
<s>,N	87	0.29
ART,N	633	1
N,V	358	0.32
N,N	108	0.10
N,P	366	0.33
V,N	134	0.37
V,ART	194	0.54
P,ART	226	0.62
P,N	140	0.38

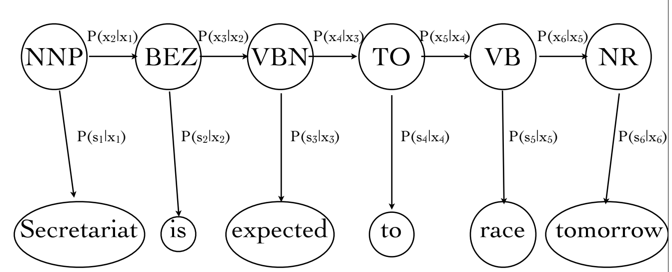
Lexical Generation

P(an | ART)	0.36
P(an | N	0.001
P(flies | N)	0.076
P(flies | V)	0.076
P(time | N)	0.0663
P(time | V)	0.012
P(arrow | N)	0.076
P(like | N)	0.012
P(like | V)	0.10
P(like | P)	0.068

Trigrams Even Better

- RB (adverb) VBD (past) *versus* RB VBN (past participle)
- Looking two back helps with "clearly marked"
  - "Is clearly marked":
    P(BEZ RB VBN) > P(BEZ RB VBD)
  - "He clearly marked":
    P(PN RB VBD) > P(PN RB VBN)
- Usual problems with sparse data ...

Hidden Markov Model



All Tags at Once

- HMM is probabilistic transducer: Probs on transitions, probs of outputs on states.
- Components:
  - Q = set of states
  - A = transition probability matrix, a_{ij} = probability of going from state i to state j.
  - O = observations from vocabulary V
  - B = sequence of observation equivalences, $b_i(o_t)$ represents prob. of o_t generated from state i.
  - q_0, q_F = start and final states

Hidden States

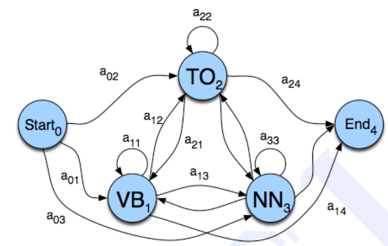


Figure 5.13 The Markov chain corresponding to the hidden states of the HMM. The *A* transition probabilities are used to compute the prior probability.

Prior Probabilities

Hidden States

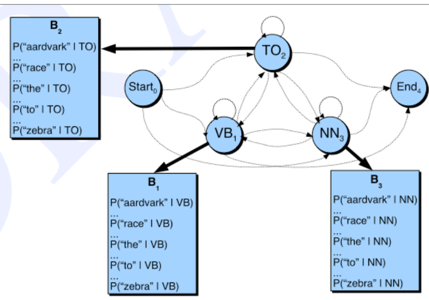


Figure 5.14 The *B* observation likelihoods for the HMM in the previous figure. Each state (except the non-emitting Start and End states) is associated with a vector of probabilities, one likelihood for each possible observation word.

Likelihood Probabilities

Goal

- Find path through FST that emits all words in sentence & maximizes probabilities.
- Path gives tagging.
- Harder than previous tasks as states are hidden.
- Try Greedy algorithm (always maximize as proceed), but won't always work!
  - *The old man the boat.*
- Backtracking leads to dynamic programming

Viterbi Algorithm

- Input: HMM as constructed by training set, input sentence.
- Returns tagging of sentence
- Builds table w/row for each state (tag) and column for each word of sentence.

```
def Viterbi(wd,HMM:(a,b)) ret best-path
  T = len(wd), N = num states of HMM
  create prob. matrix viterbi[N+1,T]
  for each state s from 1 to N do   // initialize
    viterbi[s,1] = a[0,s]*b[s,wd[1]]
    backptr[s,1] = 0
  for each time step t from 2 to T do  // iterate
    viterbi[s,t] = max viterbi[s',t-1]*a[s',s]*b[s,wd[t]]
                        for s'=1 to N
    backptr[s,t] = s' making the max
  viterbi[qF,T] = max viterbi[s',t-1]*a[s',s]   // finalize
                     for s'=1 to N
  backptr[qF,T] = s' making the max
  return path from following backptr.
```

Intuition of Viterbi

- Each time encounter new word go down the column looking at each possible state
- Look at paths to it from all rows of prev. column and calculate probabilities.
- Record the max and how it got there.
- In final state, no word emitted, just take max of prev column * prob of transition

Predicting weather

- Jason Eisner of Johns Hopkins kept a careful diary of how many ice cream cones he ate every day.
- Based on the diary, and his long term records of ice cream eating, we would like to determine the weather, based on the number of cones he ate.

Predicting Weather from Ice Cream

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.7	0.1	
p(2|…)	0.2	0.2	
p(3|…)	0.1	0.7	
p(C|…)	0.8	0.1	0.5
p(H|…)	0.1	0.8	0.5
p(STOP|…)	0.1	0.1	0

Predictions

ice creams

weather

	2	3	3	1	1
H	0.1	0.056	0.03136	0.0025088	0.000200704
C	0.1	0.008	0.00064	0.0021952	0.001229312

v[X,t+1] = MAX(v[H,t]*P(X|H),V[C,t]*P(X|C))*P(#|X)
for X = H or C

Spread sheet: icecreamPredWeather.xls

Drawbacks

- Bigrams not as accurate, go with trigrams
- Sparse data!
- Back up to bigram or unigram if fails
- Can also train to find best linear combination.

Any Questions?