Lecture 7: POS Tagging

Dan Klein – UC Berkeley

Statistical NLP
Spring 2008

One basic kind of linguistic structure: syntactic word classes

Open class (lexical)

Verbs

Adjectives

Proper

Common

Main

Adverbs

Nouns

Determiners

Conjunctions

Pronouns

Closed class (functional)

Modals

Prepositions

Particles

POS Tagging

IBM
tut / cats

one

registered

see

can

had

yellow

slowly

to

and

or

the

he

its

Numbers

122,312

hit

is

Fed raises interest rates 0.5 percent

HMMs

We want a model of sequences \( s \) and observations \( w \)

\[
P(s; w) = \prod_i P(s_i|s_{i-1})P(w_i|s_i)
\]

Assumptions:

- States are tag n-grams
- Usually a dedicated start and end state / word
- Tag/state sequence is generated by a markov model
- Words are chosen independently, conditioned only on the tag/state
- These are totally broken assumptions: why?

Why POS Tagging?

Useful in and of itself

- Text-to-speech: record, lead
- Lemmatization: saw[v] → see, saw[n] → saw
- Quick-and-dirty NP-chunk detection: grep {JJ | NN}* {NN | NNS}

Useful as a pre-processing step for parsing

- Less tag ambiguity means fewer parses
- However, some tag choices are better decided by parsers

HMMs

Fed raises interest rates 0.5 percent

Part-of-Speech (English)

- Parts-of-Speech (English)
- Open class (lexical)
- Verbs
- Adjectives
- Proper
- Common
- Main
- Adverbs
- Nouns
- Determiners
- Conjunctions
- Pronouns
- Closed class (functional)
- Modals
- Prepositions
- Particles

Part-of-Speech Ambiguity

Example

Fed raises interest rates 0.5 percent

- Two basic sources of constraint:
  - Grammatical environment
  - Identity of the current word

- Many more possible features:
  - … but we won’t be able to use them for a while

Important:

- We want a model of sequences \( s \) and observations \( w \)

\[
P(s; w) = \prod_i P(s_i|s_{i-1})P(w_i|s_i)
\]

Assumptions:

- States are tag n-grams
- Usually a dedicated start and end state / word
- Tag/state sequence is generated by a markov model
- Words are chosen independently, conditioned only on the tag/state
- These are totally broken assumptions: why?

Why POS Tagging?

Useful in and of itself

- Text-to-speech: record, lead
- Lemmatization: saw[v] → see, saw[n] → saw
- Quick-and-dirty NP-chunk detection: grep {JJ | NN}* {NN | NNS}

Useful as a pre-processing step for parsing

- Less tag ambiguity means fewer parses
- However, some tag choices are better decided by parsers

HMMs

We want a model of sequences \( s \) and observations \( w \)

\[
P(s; w) = \prod_i P(s_i|s_{i-1})P(w_i|s_i)
\]

Assumptions:

- States are tag n-grams
- Usually a dedicated start and end state / word
- Tag/state sequence is generated by a markov model
- Words are chosen independently, conditioned only on the tag/state
- These are totally broken assumptions: why?
Transitions and Emissions

- Transitions \( P(s|s') \) encode well-formed tag sequences.
- In a bigram tagger, states = tags.
- In a trigram tagger, states = tag pairs.

Estimating Transitions

- Use standard smoothing methods to estimate transitions:
  \[ P(t_3 | t_0, t_2) = \lambda_1 P(t_3 | t_0, t_2) + \lambda_2 P(t_3 | t_2) + (1 - \lambda_1 - \lambda_2) P(t_3) \]
- Can get a lot fancier (e.g., KN smoothing), but in this case it doesn’t buy much.
- One option: encode more into the state, e.g., whether the previous word was capitalized (Brants 00).

Estimating Emissions

- Emissions are trickier.
  \[ P(s, w) = \prod P(s_j | s_{j-1}) P(w_j | s_j) \]
- Words we’ve never seen before.
- Words which occur with tags we’ve never seen.
- One option: break out the Good-Turing smoothing.
- Issue: words aren’t black boxes:
  \[ 343,127.23 \Rightarrow \text{Minteria reintroducibly} \]
- Another option: decompose words into features and use a maxent model along with Bayes’ rule.
  \[ P(w | t) = P_{\text{maxent}}(t | w)P(w) / P(t) \]

Better Features

- Can do surprisingly well just looking at a word by itself:
  - Word: the → DT
  - Lowercased word: importantly: importantly → RB
  - Prefixes: unfathomable: un- → JJ
  - Suffixes: importantly: -ly → RB
  - Capitalization: Meridian: CAP → NNP
  - Word shapes: 35-year: d-x → JJ

- Then build a maxent (or whatever) model to predict tag.
- Maxent \( P(t|w) \): 93.7% / 82.6%

Disambiguation

- Given these two multinomials, we can score any word / tag sequence pair.
- In principle, we’re done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence).

Fed raises interest rates 0.5 percent.

\[ P(\text{NNP}|<,>) \cdot P(\text{Fed}|\text{NNP}) \cdot P(\text{VBZ}|\text{NNP},>) \cdot P(\text{raises}|\text{VBZ}) \cdot P(\text{NN}|\text{VBZ},\text{NNP}) \ldots \]

\[ \log P = -23 \quad \log P = -29 \quad \log P = -27 \]
Finding the Best Trajectory

- Too many trajectories (state sequences) to list
- Option 1: Beam Search
  - A beam is a set of partial hypotheses
  - Start with just the single empty trajectory
  - At each derivation step:
    - Consider all continuations of previous hypotheses
    - Discard most, keep top k, or those within a factor of the best, or some combination
- Beam search works relatively well in practice
  - ... but sometimes you want the optimal answer
  - ... and you need optimal answers to validate your beam search

The Path Trellis

- Represent paths as a trellis over states
- Each arc \((s_i \rightarrow s_{i+1})\) is weighted with the combined cost of:
  - Transitioning from \(s_i\) to \(s_{i+1}\) (which involves some unique tag \(t\))
  - Emitting word \(i\) given \(t\)
- Each state path (trajectory):
  - Corresponds to a derivation of the word and tag sequence pair
  - Corresponds to a unique sequence of part-of-speech tags
  - Has a probability given by multiplying the arc weights in the path

HMM Trellis

The Viterbi Algorithm

- Dynamic program for computing
  \[
  \delta_i(s) = \max_{s_{i-1}} \left( P(s_{i-1} \rightarrow s, w_{i-1} \ldots w_i) \right)
  \]
- The score of a best path up to position \(i\) ending in state \(s\)
  \[
  \delta_0(s) = \begin{cases} 1 & \text{if } s = \text{start} \\ 0 & \text{otherwise} \end{cases}
  \]
- \(\delta_i(s) = \max P(s_i | s') P(w_i | s') \delta_{i-1}(s')\)
- Also store a backtrace
  \[
  \psi_i(s) = \arg \max P(s_i | s') P(w_i | s') \delta_{i-1}(s')
  \]
- Memoized solution
- Iterative solution

So How Well Does It Work?

- Choose the most common tag
  - 90.3% with a bad unknown word model
  - 93.7% with a good one
- TnT (Brants, 2000):
  - A carefully smoothed trigram tagger
  - Suffix trees for emissions
  - 96.7% on WSJ text (SOA is ~97.2%)
- Noise in the data
  - Many errors in the training and test corpora
  - Probably about 2% guaranteed error from noise (on this data)

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - Maxent \(P(t|w)\): 93.7% / 82.6%
  - TnT (HMM++): 96.2% / 86.0%
  - MEMM tagger: 96.9% / 86.0%
  - Cyclic tagger: 97.2% / 89.0%
  - Upper bound: ~98%
What’s Next for POS Tagging

- Better features!
  - We could fix this with a feature that looked at the next word
  - We could fix this by linking capitalized words to their lowercase versions
- Solution: maximum entropy sequence models (next class)
- Reality check:
  - Taggers are already pretty good on WSJ journal text…
  - What the world needs is taggers that work on other text!

Sequence-Free Tagging?

- What about looking at a word and it’s environment, but no sequence information?
  - Add in previous / next word
  - Previous / next word shapes
  - Occurrence pattern features
  - Crude entity detection
  - Phrasal verb in sentence?
  - Conjunctions of these things
- All features except sequence: 96.6% / 86.8%
- Uses lots of features: > 200K
- Why isn’t this the standard approach?

Feature Templates

- We’ve been sloppy:
  - Features: \(<w_0=\text{future}, t_0=\text{JJ}>\)
  - Feature templates: \(<w_0, t_0>\)
- In maxent taggers:
  - Can now add edge feature templates:
    - \(<t_{i-1}, t_i>\)
    - \(<t_{i-1}, t_i, t_{i+1}>\)
  - Also, mixed feature templates:
    - \(<t_i, w_0, t_0>\)

Maxent Taggers

- One step up: also condition on previous tags
  \[ P(t_i|w) = \prod_i P_{ME}(t_i|w, t_{i-1}, t_{i-2}) \]
  - Train up \(P(t_i|w, t_{i-1}, t_{i-2})\) as a normal maxent problem, then use to score sequences
  - This is referred to as a maxent tagger [Ratnaparkhi 96]
  - Beam search effective! (Why?)
  - What’s the advantage of beam size 1?

Decoding

- Decoding maxent taggers:
  - Just like decoding HMMs
  - Viterbi, beam search, posterior decoding
- Viterbi algorithm (HMMs):
  \[ \delta_i(s) = \arg \max_{s'} P(s|s')P(w_{i-1}|s)\delta_{i-1}(s') \]
- Viterbi algorithm (Maxent):
  \[ \delta_i(s) = \arg \max_{s'} P(s|s', w)\delta_{i-1}(s') \]
**TBL Tagger**

- [Brill 95] presents a transformation-based tagger
  - Label the training set with most frequent tags
    
    ```
    DT MD VBD VBD .
    The can was rusted .
    ```
  - Add transformation rules which reduce training mistakes
    - MD → NN: DT ______
    - VBD → VBN: VBD ______
  - Stop when no transformations do sufficient good
  - Does this remind anyone of anything?
  - Probably the most widely used tagger (esp. outside NLP)
  - … but not the most accurate: 96.6% / 82.0 %

**EngCG Tagger**

- English constraint grammar tagger
  - [Tapanainen and Voutilainen 94]
  - Something else you should know about
  - Hand-written and knowledge driven
  - “Don’t guess if you know” (general point about modeling more structure!)
  - Tag set doesn’t make all of the hard distinctions as the standard tag set (e.g. JJ/NN)
  - They get stellar accuracies: 98.5% on their tag set
  - Linguistic representation matters…
  - … but it’s easier to win when you make up the rules

**CRF Taggers**

- Newer, higher-powered discriminative sequence models
  - CRFs (also voted perceptrons, M3Ns)
  - Do not decompose training into independent local regions
  - Can be deathly slow to train – require repeated inference on training set
  - Differences tend not to be too important for POS tagging
  - However: one issue worth knowing about in local models
    - “Label bias” and other explaining away effects
    - Maxent taggers’ local scores can be near one without having both good “transitions” and “emissions”
    - This means that often evidence doesn’t flow properly
    - Why isn’t this a big deal for POS tagging?

**Domain Effects**

- Accuracies degrade outside of domain
  - Up to triple error rate
  - Usually make the most errors on the things you care about in the domain (e.g. protein names)

- Open questions
  - How to effectively exploit unlabeled data from a new domain (what could we gain?)
  - How to best incorporate domain lexica in a principled way (e.g. UMLS specialist lexica, ontologies)