

11-711 Algorithms for NLP

Part-of-Speech Tagging

Reading:

James Allen, NLU, Ch.7: 7.1-7.4

Jurafsky and Martin, SLP: Ch 8

Optional: Manning and Schuetze, SNLP, Ch. 10

Statistical Models for Natural Language Analysis

- Statistical models can be used as a substitute for detailed symbolic semantic and pragmatic models of language
- Easier to apply:
 - Once the statistical model has been defined, statistics can often be extracted (trained) automatically
 - Availability of large language corpora makes this attractive and feasible
 - The models are based on underlying knowledge - often require “supervised” training
 - Can often be combined with “pure” knowledge based methods - covers “the gaps”
- Such models can be applied at various levels of language analysis:
 - Part-of-Speech Tagging
 - Parse-tree Disambiguation using probabilistic CFGs
 - Semantic disambiguation using selectional restrictions

Part-of-Speech Tagging

- Most syntactic grammars describe sentence structure from the level of POS (articles, nouns, verbs, prepositions, etc.)
- Many words have multiple possible POS (“*can*”: *VB*, *VBP*, *NN*)
- For a given input sentence, how do we determine the correct sequence of POS tags?
- The *meaning* of the sentence determines the correct tagging
⇒ a “chicken and egg” problem
 - Consider multiple POS for each word during parsing - may pass along the ambiguity
 - Select *a single* POS tagging prior to parsing, using various statistical methods

Part-of-Speech Tagging

- Simple Method: Pick the most likely tag for each word
 - The probabilities can be estimated from a tagged corpus
 - Assumes independence between tags
 - Works pretty good (usually a word tag accuracy $> 90\%$)
 - But not good enough - one in ten words wrong!

Information Sources for POS Tagging

Two main sources of information:

- The distribution of tags for the word in isolation:

$$P(t|w)$$

- Syntagmatic information - some POS sequences are much more common than others due to syntactic constraints of the language
 - Example: “*a new play*”: AT JJ NN versus AT JJ VBP
 - We must find ways of capturing this type of information via appropriate probability models
- We would like to have a statistical model that would be able to take into account both types of information in a principled way

POS Tagging with Markov Models

The basis for all Markov Models for POS Tagging:

- We look for $t_1 t_2 \dots t_n$ that maximizes $P(t_1 t_2 \dots t_n | w_1 w_2 \dots w_n)$
- Using Bayes Rule:

$$P(t_1 t_2 \dots t_n | w_1 w_2 \dots w_n) = \frac{P(w_1 w_2 \dots w_n | t_1 t_2 \dots t_n) * P(t_1 t_2 \dots t_n)}{P(w_1 w_2 \dots w_n)}$$

- We need to find $t_1 t_2 \dots t_n$ that maximizes the numerator
- Two main independence assumptions:

– Words are independent of each other:

$$P(w_1 w_2 \dots w_k | t_1 t_2 \dots t_n) = \\ P(w_1 | t_1 t_2 \dots t_n) * P(w_2 | t_1 t_2 \dots t_n) * \dots * P(w_n | t_1 t_2 \dots t_n)$$

– The probability of a word depends only on its tag:

$$P(w_i | t_1 t_2 \dots t_n) = P(w_i | t_i)$$

POS Tagging with a Simple Markov Model

A probability model for $P(t_1 t_2 \dots t_n)$:

- A breakdown of the joint probability:

Recall that $P(A, B) = (P(A|B) * P(B))$, thus:

$$P(t_1 t_2 \dots t_n) = P(t_n | t_1 t_2 \dots t_{n-1}) * P(t_{n-1} | t_1 t_2 \dots t_{n-2}) * \dots * P(t_2 | t_1) * P(t_1)$$

- The simple Markov Model “limited horizon” principle:

$$P(t_i | t_1 t_2 \dots t_{i-1}) = P(t_i | t_{i-1})$$

- Thus, using the Markov Model we get that:

$$P(t_1 t_2 \dots t_n) = P(t_n | t_{n-1}) * P(t_{n-1} | t_{n-2}) * \dots * P(t_2 | t_1) * P(t_1)$$

With a simple Markov Model, we need to find the POS sequence $t_1 t_2 \dots t_n$ that

maximizes $\prod_{i=1}^n P(w_i | t_i) * P(t_i | t_{i-1})$

Two main questions:

- How do we obtain (train) these probabilities?
- How do we find the sequence of tags that maximizes the joint product?

Training the Bigram Models

- Calculate estimated probabilities based on frequency counts from a corpus
- Smooth the estimated probabilities to avoid anomalies due to data sparseness

Finding the Maximum Likelihood Tag Sequence

We use the Viterbi search algorithm:

- Incremental left-to-right search using dynamic programming
- $\delta_i(j)$ is the probability of the most likely sequence of tags that ends with word i having the tag t_j
- $\psi_{i+1}(j)$ will contain the tag t of word i that corresponds to $\delta_{i+1}(j)$
- The induction of the search:
 - Find probability of most likely sequence that ends with word $i + 1$ having tag j , by considering all possible tags for previous word i :
$$\delta_{i+1}(j) = \max_{1 \leq k \leq T} [\delta_i(k) * P(t^j | t^k) * P(w_{i+1} | t^j)]$$
 - For specific tag t^k that resulted in the above max, set: $\psi_{i+1}(j) = t^k$
- Start with $\delta_1(Period) = 1$, $\delta_1(t) = 0$ for all $t \neq PERIOD$
- At the end, find the tag t for which $\delta_{n+1}(t)$ is the greatest
- Reconstruct the most likely sequence of tags by backtracking:
$$X_{n+1} = t \quad \text{for } i = n \text{ to } 1 \quad X_i = \psi_{i+1}(X_{i+1})$$

Further Issues with Markov Model Tagging

Unknown Words:

- Unknown words are a problem, since we don't have the required probabilities $P(w|t)$.
- Solution: estimate the likelihood of unknown words for various POS from a large corpus

Using higher order Markov Models:

- using a trigram model for POS captures more context and is thus potentially a better probability model
- However, data sparseness is much more of a problem
- The Viterbi search for trigrams is more complex
- trigrams are not always better than bigrams

Smoothing of the Probabilities

- Data sparseness is always a problem when estimating probabilities based on a corpus of data
- The “adding one” smoothing technique: add a count of one to all events, so that there are no zero probabilities
- linear interpolation methods can compensate for data sparseness with higher order models. A common method is interpolating trigrams, bigrams and unigrams:

$$P(t_i|t_1t_2\dots t_{i-1}) = \lambda_1P_1(t_i) + \lambda_2P_2(t_i|t_{i-1}) + \lambda_3P_3(t_i|t_{i-1}t_{i-2})$$

- interpolated models can be used in conjunction with a Viterbi search

Tag-by-Tag versus Overall Max Likelihood

- The method we described previously finds the overall most likely sequence of tags
- Alternatively, we can find the most likely tag for each word independently, still using the Markov Models
- This involves summing probabilities over all sequences in which the word has a specific tag and picking the tag with the highest probability
- In practice this turns out to be not much better or worse than the joint overall max likelihood method

Transformation-Based POS Tagging

- First proposed and developed by Eric Brill (1995)
- Idea: we learn (from a tagged corpus) a sequence of transformations to improve the tagging of an initial basic tagger
- The transformations are in the form of rules: “if the following triggering conditions match, then change tag X to tag Y”
- This is more of a symbolic (rather than statistical) method
- Main advantage - the transformation rules can use much more specific information and context in determining whether they should be applied
- Two main components to the approach:
 - constructing the set of valid possible transformation triggering conditions
 - The learning algorithm

Transformation Trigger Environments

- The transformation itself is kept simple: “replace tag t1 with tag t2”
- The inventory of transformations is a function of the types of possible triggers for applying a transformation
- Most common types of triggers:
 - “Tag X occurs/doesn’t-occur in one of the previous/next Y positions”
 - “Word X occurs/doesn’t-occur in one of the previous/next Y positions”
 - Morphological conditions on preceding/following words
 - Combinations (multiple conditions) of the above
- The space of possible combinations of trigger environments is huge
- Linguistic intuition has significantly influenced the choices of how to limit this space to a reasonable set of candidate transformations

Applying the Transformations

- The learning produces an *ordered* sequence of transformations
- Applying the method on new unseen data works as follows:
 1. Tag the input with the basic tagger - Brill's tagger starts with simply the most likely tag for each word
 2. Scan the ordered list of transformations and find the *first* transformation rule that applies.
 3. Apply the transformation (Brill uses delayed rather than immediate effect)
 4. Repeat, until no rules apply.

The Learning Algorithm

- A “greedy” search algorithm for finding the rule that most reduces the overall number tagging errors in the training corpus:
 1. Create the collection of all possible transformation rules
 2. Create an initial tagged corpus C using the initial tagger
 3. Calculate the current overall number of tagging errors $E(C)$
 4. Apply each possible transformation U to the entire corpus and for each, calculate the number of tagging errors in the transformed corpus $E(U(C))$
 5. Select V , the transformation for which $E(V(C))$ is minimal
 6. If $E(C) - E(V(C)) < \epsilon$ then halt
 7. Add V to the end of the sequence of transformation rules
 8. Replace the corpus C with $V(C)$, and goto (4)

Further Issues

- A major advantage to the TBL approach is that it can be trained in an “unsupervised” fashion - without a correctly tagged training corpus
- Brill achieved a tagging accuracy of 95.6% with unsupervised learning
- The TBL method is very robust to over-training
- The runtime transformation rule system can be sped up significantly by the construction of an equivalent single joint finite-state transducer
- Transformation-based learning has been applied successfully to “correcting” the output of other NLP components and processes, such as parsers, and disambiguation modules