
Advanced Natural Language Processing

Lecture 6

Part-of-speech Tagging with Hidden Markov Models

Philipp Koehn

4 October 2011



Parts of Speech

- **Open class words** (or content words)
 - nouns, verbs, adjectives, adverbs
 - mostly content-bearing: they refer to objects, actions, and features in the world
 - *open* class, since there is no limit to what these words are, new ones are added all the time ([email](#), [website](#)).
- **Close class words**
 - pronouns, determiners, prepositions, connectives, ...
 - there is a limited number of these
 - mostly functional: to tie the concepts of a sentence together

Parts of Speech (2)

- There are about 30-100 parts of speech
 - distinguish between names and abstract nouns?
 - distinguish between plural noun and singular noun?
 - distinguish between past tense verb and present tense word?
- Identifying the parts of speech is a first step towards syntactic analysis

Ambiguous words

- For instance: like
 - verb: I like the class.
 - preposition: He is like me.
- Another famous example: Time flies like an arrow
- Most of the time, the local context disambiguated the part of speech

Part-of-speech tagging

- Task: Given a text of English, identify the parts of speech of each word
- Example
 - Input: Word sequence
Time flies like an arrow
 - Output: Tag sequence
Time/NN flies/VB like/P an/DET arrow/NN
- What will help us to tag words with their parts-of-speech?

Relevant knowledge for POS tagging

- The word itself
 - Some words may only be nouns, e.g. *arrow*
 - Some words are ambiguous, e.g. *like, flies*
 - Probabilities may help, if one tag is more likely than another
- Local context
 - two determiners rarely follow each other
 - two base form verbs rarely follow each other
 - determiner is almost always followed by adjective or noun

Bayes rule

- We want to find the best part-of-speech tag sequence T for a sentence S :

$$\operatorname{argmax}_T p(T|S)$$

- Bayes rule gives us:

$$p(T|S) = \frac{p(S|T) p(T)}{p(S)}$$

- We can drop $p(S)$ if we are only interested in argmax_T :

$$\operatorname{argmax}_T p(T|S) = \operatorname{argmax}_T p(S|T) p(T)$$

Decomposing the model

- The mapping $p(S|T)$ can be decomposed into

$$p(S|T) = \prod_i p(w_i|t_i)$$

- $p(T)$ could be called a *part-of-speech language model*, for which we can use an n-gram model (bigram):

$$p(T) = p(t_1) p(t_2|t_1) p(t_3|t_2) \dots p(t_n|t_{n-1})$$

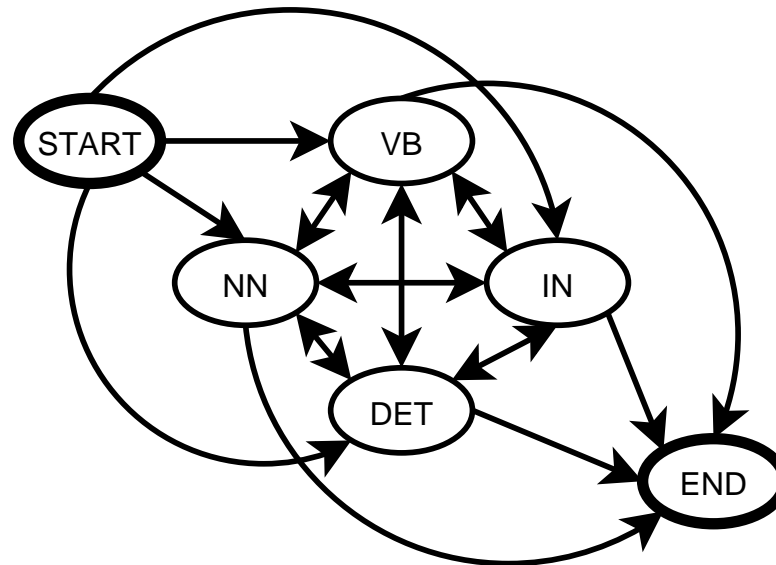
- We can estimate $p(S|T)$ and $p(T)$ with maximum likelihood estimation (and maybe some smoothing)

Hidden Markov Model (HMM)

- The model we just developed is a **Hidden Markov Model**
- Elements of an HMM model:
 - a set of states (here: the tags)
 - an output alphabet (here: words)
 - initial state (here: beginning of sentence)
 - state transition probabilities (here: $p(t_n|t_{n-1})$)
 - symbol emission probabilities (here: $p(w_i|t_i)$)

Graphical representation

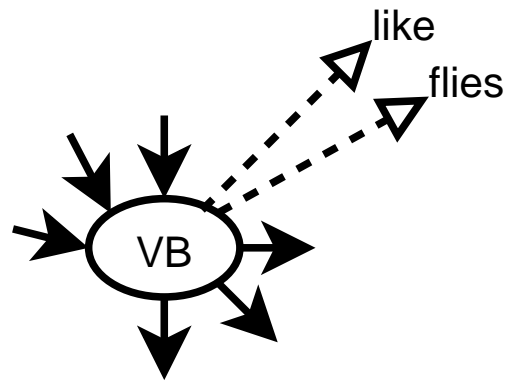
- When tagging a sentence, we are walking through the state graph:



- State transition probabilities: $p(t_n|t_{n-1})$

Graphical representation (2)

- At each state we emit a word:



- Symbol emission probabilities: $p(w_i|t_i)$

Search for the best tag sequence

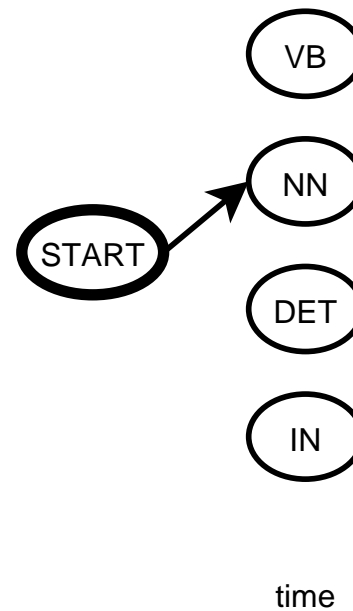
- We have defined a model, but how do we use it?
 - given: word sequence
 - wanted: tag sequence
- If we consider a specific tag sequence, it is straight-forward to compute its probability

$$p(S|T) p(T) = \prod_i p(w_i|t_i) p(t_i|t_{i-1})$$

- Problem: if we have on average c choices for each of the n words, there are c^n possible tag sequences, maybe too many to efficiently evaluate

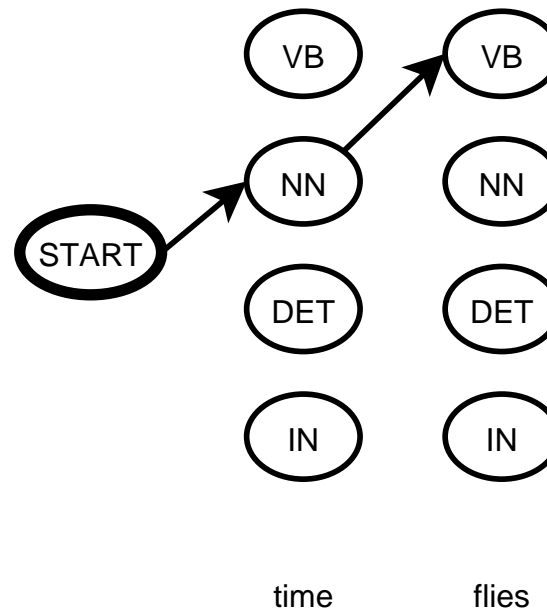
Walking through the states

- First, we go to state NN to emit *time*:



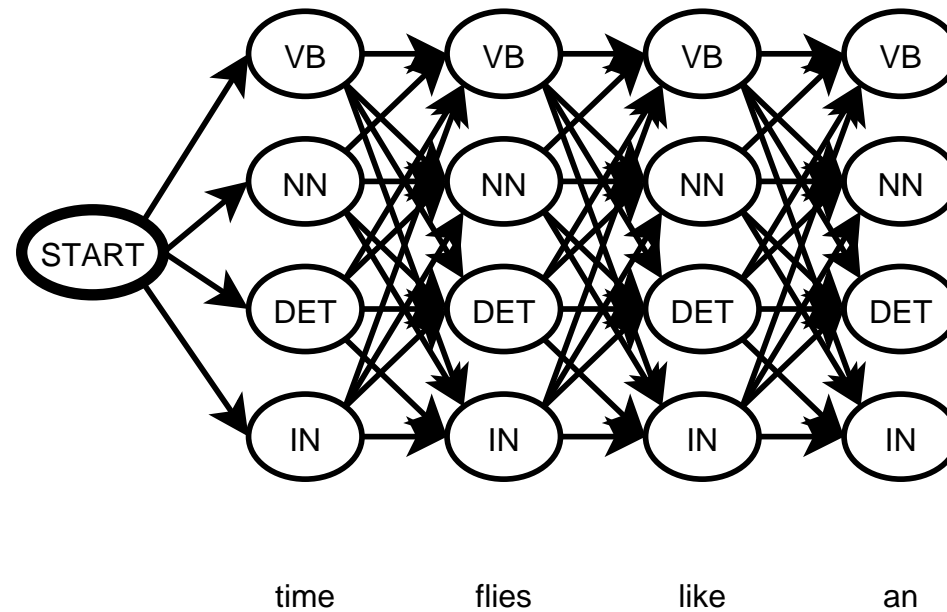
Walking through the states (2)

- Then, we go to state VB to emit *flies*:



Walking through the states (3)

- Of course, there are many possible paths:



Viterbi algorithm

- Intuition: Since state transition out of a state only depend on the current state (and not previous states), we can record for each state the optimal path
- We record:
 - cheapest cost to state j at step s in $\delta_j(s)$
 - backtrace from that state to best predecessor $\psi_j(s)$
- Stepping through all states at each time steps allows us to compute
 - $\delta_j(s + 1) = \max_{1 \leq i \leq N} \delta_i(s) p(t_j|t_i) p(w_s|t_j)$
 - $\psi_j(s + 1) = \operatorname{argmax}_{1 \leq i \leq N} \delta_i(s) p(t_j|t_i) p(w_s|t_j)$
- Best final state is $\operatorname{argmax}_{1 \leq i \leq N} \delta_i(S + 1)$, we can backtrack from there

Other tagging tasks

- A number of problems can be framed as tagging problems:
- **BaseNP chunking:** for text processing purposes it is useful to detect base noun phrases that correspond to concepts, e.g. [department of defense](#)
- **Named entity recognition:** it may also be useful to find names of persons, organizations, etc. in the text, e.g. [Tony Blair](#)
- **Accent restoration:** When keyboards lack the proper keys, it is common to not write the accents in Spanish or French. We may want to restore them.
- **Case restoration:** If we just get lowercased text, we may want to restore proper casing, e.g. [the river Thames](#)

BaseNP chunking

- Task: find basic noun phrases (facilitates parsing, information extraction)
- Example: [the student] said [the exam question] is hard
- Three tags
 - B = beginning of baseNP
 - I = continuing baseNP (internal)
 - O = other word
- Example: the/B student/I said/O the/B exam/I question/I is/O hard/O
- Tagging task: assign tags (B, I, O) to each word