Lecture 21: Advanced POS Taggers, Trees

Ling 1330/2330 Intro to Computational Linguistics
Na-Rae Han, 11/12/2024

Overview

- Building POS taggers:
  - N-gram tagger
  - Hidden Markov Model (HMM) tagger
  - For discussions on HMM, see Jurafsky & Martin

- Homework 7
  - Comparison with: Hidden Markov Model (HMM) tagger

- Syntactic trees
  - NLTK Ch.7 & Ch.8 ← In lecture21.html

Homework 7

- You built a **bigram tagger**
  - Backs off to a unigram tagger, which backs off to a "NN" default tagger
  - Trained and tested on the Brown corpus
  - Trained on the first 50,000 sentences = 1,039,920 words
  - Tested on the last 7340 sentences = 121,272 words

- How good is it? Can we make a better tagger?
- How well does it perform on 'cold' NN-JJ ambiguity?
- What are its strengths and limitations?

Performance

Tagger	Accuracy	Improvement
t0 ('NN' default tagger)	0.10919	n/a
t1 (unigram tagger)	0.88978	+ 0.78059
t2 (bigram tagger)	0.91116	+ 0.02138

- How to make it better?
  - Obvious candidate: build a **trigram tagger** on top.

Tagger	Accuracy	Improvement
t3 (trigram tagger)	0.91180	+ 0.00063

  - What do you notice about the amount of improvement?

🡐 As the size of *n* in your *n*-gram tagger increases, you see a smaller gain in performance improvement. Performance may even drop! (overfitting)

Performance: even better?

Tagger	Accuracy	Improvement
t0 ('NN' default tagger)	0.10919	n/a
t1 (unigram tagger)	0.88978	+ 0.78059
t2 (bigram tagger)	0.91116	+ 0.02138
t3 (trigram tagger)	0.91180	+ 0.00063

- Anything else we can try?
  - Can we do even better?
  - One simple fix: replace the default tagger ('everything's NN!!') with something more intelligent: a **regular-expression tagger**.

    ← After that, you need to rebuild your 1- 2- 3-gram taggers.

Regular expression tagger as t0

```
>>> patterns = [
        (r'.*ing$', 'VBG'),           # gerunds
        (r'.*ed$', 'VBD'),            # simple past
        (r'.*es$', 'VBZ'),            # 3rd singular present
        (r'.*\'s$', 'NN$'),               # possessive nouns
        (r'^-?[0-9]+(\.[0-9]+)?$', 'CD'),  # cardinal numbers
        (r'^[A-Z][a-z]*s$', 'NPS'),       # plural proper nouns
       (r'^[A-Z][a-z]*[^s]$', 'NP'),     # singular proper nouns
        (r'.*s$', 'NNS'),                 # plural nouns
        (r'.*', 'NN')                     # nouns (default)
      ]
>>> re_tagger = nltk.RegexpTagger(patterns)
>>> re_tagger.tag('Akbar and Jedis tweeted'.split())
[('Akbar', 'NP'), ('and', 'NN'), ('Jedis', 'NPS'), ('tweeted', 'VBD')]
```

- More sophisticated than the 'NN' default tagger!

New tagger performance

Tagger	Accuracy	Improvement
re_tagger (regex tagger)	0.19243	+ 0.08324 from t0
t1new (unigram tagger)	0.90395	+ 0.01416 from t1
t2new (bigram tagger)	0.92563	+ 0.01447 from t2
t3new (trigram tagger)	0.92634	+ 0.01454 from t3

- 1.5% overall improved performance!
- Regex tagger does a better job of handling "unseen" words than the 'NN' default tagger: ‘tweeted’, ‘Akbar’

How n-gram taggers work

- How do our n-gram taggers handle the 'cold' NN-JJ ambiguity?
- Mining the training data for instances of 'cold' as NN or JJ
  - cold/JJ vs. cold/NN in the training data: 110* vs. 8

    ➔ The unigram tagger will always pick JJ for 'cold'.
  - Considering POS_{n-1}:
    - AT cold/JJ (38) vs. cold/NN (4) ➔ JJ wins
    - JJ cold/JJ (4) vs. cold/NN (2) ➔ JJ wins
    - DT cold/JJ (3) vs. cold/NN (1) ➔ JJ wins
    - , cold/JJ (3) vs. cold/NN (1) ➔ JJ wins
  - Every POS_{n-1} in fact favors JJ for 'cold'!

    ➔ The bigram tagger too will always tag 'cold' as JJ.

* 109 sentences in cold_JJ, but there is a sentence with two instances of cold/JJ.

'cold': adjective or noun?

1. *I was very cold.* ✓
2. *January was a cold month.* ✓
3. *I had a cold.* ✗
4. *I had a severe cold.* ✗

1-4 all tagged 'JJ' by the bigram tagger (t2).

- OK, so our bigram tagger fails to treat 'cold' as a noun, *ever*.
- Does a trigram tagger do better?

'cold': adjective or noun?

1. *I was very <u>cold</u>.* ✓
2. *January was a <u>cold</u> month.* ✓
3. *I had a <u>cold</u>.* ✗
4. *I had a severe <u>cold</u>.* ✗

1-4 all tagged 'JJ' by the bigram tagger (t2).

- OK, so our bigram tagger fails to treat 'cold' as a noun, ever.
- Does a trigram tagger do better?
  - YES! On one of them: "I had a cold".

```
>>> t3.tag('I had a cold .'.split())
[('I', 'PPSS'), ('had', 'HVD'), ('a', 'AT'), ('cold', 'NN'), ('.', '.')]
>>> t3.tag('I had a severe cold .'.split())
[('I', 'PPSS'), ('had', 'HVD'), ('a', 'AT'), ('severe', 'JJ'), ('cold',
'JJ'), ('.', '.')]
```

'cold': adjective or noun?

1. *I was very cold.*
2. *January was a cold month.*
3. *I had a cold.*
4. *I had a severe cold.*

"HVD AT cold/NN" has a *higher* count than "HVD AT cold/JJ" in training data.

- OK, so our bigram tagger fails to treat 'cold' as a noun, ever.
- Does a trigram tagger do better?
  - YES! On one of them: "I had a cold".

```
>>> t3.tag('I had a cold .'.split())
[('I', 'PPSS'), ('had', 'HVD'), ('a', 'AT'), ('cold', 'NN'), ('.', '.')]
>>> t3.tag('I had a severe cold .'.split())
[('I', 'PPSS'), ('had', 'HVD'), ('a', 'AT'), ('severe', 'JJ'), ('cold', 
'JJ'), ('.', '.')]
```

So: three POS tags

Sentence examples
I failed to do <u>so</u>. It wasn't <u>so</u>. I was happy, but <u>so</u> was my enemy.
<u>So</u>, how was the exam? They rushed <u>so</u> they can get good seats. She failed, <u>so</u> she must re-take the exam.
That was <u>so</u> incredible. Wow, <u>so</u> incredible. The prices fell <u>so</u> fast.

So: three POS tags

Sentence examples	POS	traits
I failed to do so. It wasn't so. I was happy, but so was my enemy.	**RB** (Adverb)	Modifies a verb.
So, how was the exam? They rushed so they can get good seats. She failed, so she must re-take the exam.	**CS** (Subordinating conjunction)	Clausal adverb; starts a subordinate clause.
That was so incredible. Wow, so incredible. The prices fell so fast.	**QL** (Qualifier)	Aka 'intensifier'; modifies following adjective or adverb.

Which were more frequent in Jane Austen? The Bible?

n-gram tagger: limitations?

*I was very <u>cold</u> **.***	*I failed to do <u>so</u> **.***
*January was a <u>cold</u> **month.***	*She failed the exam, <u>so</u> **she** …*
*I had a <u>cold</u> **.***	*That was <u>so</u> **incredible**.*
*I had a severe <u>cold</u> **.***	*Wow, <u>so</u> **incredible**.*

- Q: Does it matter at all what comes AFTER 'cold'? 'so'?
  - NOT unless you make your tagger work the <u>opposite direction</u>.

    ← But then, it won't be able to use the left-hand side context!
- In general, an n-gram tagger **makes a decision for a given word, one at a time**, in a **single direction**.
- It **commits to every decision it makes as it proceeds**. It cannot go back on it after seeing more context.
- It does **NOT optimize for global POS tag assignment**.

Global optimization of tags

- n-gram taggers do NOT optimize for global (sentence-wide) POS tag assignment.
- More sophisticated probabilistic sequential taggers do.
  - ➔ HMM taggers, CRF taggers, …

that is so cold

DT
BEZ
QL
JJ

***n*-gram tagger**

that is so cold

DT	BEZ	QL	JJ
DT	BEZ	QL	NN
DT	BEZ	CS	JJ
DT	BEZ	CS	NN
	…		

Which is most likely?

HMM tagger

Evaluating a tagger

- But how good is "good"? 90%? 95%? 98%...?
- We need to establish a **baseline**.
  - A good unigram tagger can already achieve 90-91% (!)
  - Bigram/trigram … taggers should show a better performance.
- How about a **ceiling**?
  - ← Agreement between human annotators are said to top out at ~97%.
  - ← Therefore, trained taggers cannot be expected to perform better than that.

Advanced POS taggers

- Rule-based taggers

- Transformation-based taggers (Brill tagger)
  - ←NLTK book focuses on it; we will skip it

- Hidden-Markov Model (HMM) taggers
  - ← These use more sophisticated probabilistic techniques.

Probabilistic sequence models

- Generally, POS tagging can be viewed as a **sequence labeling task**.
  - input: Colorless green ideas sleep furiously
  - labels: JJ JJ NNS VBP RB

*Penn Treebank tagset.

- **Probabilistic sequence models** allow integrating uncertainty over multiple, interdependent classifications and collectively determine the most likely **GLOBAL assignment**.

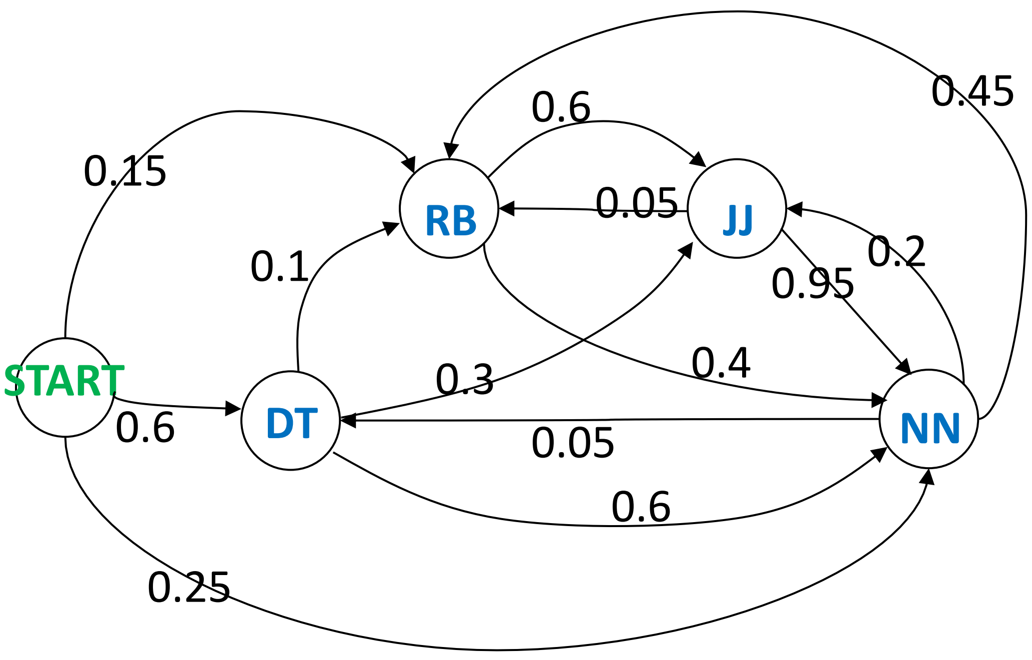
- Well-known models:
  - **Hidden Markov Model (HMM)**
  - **Conditional Random Field (CRF)**

Markov model (Markov chain)

- A **finite state machine** with **probabilistic state transitions**.
- Makes Markov assumption that the next state only depends on the current state and is independent of previous history.
- **Hidden** Markov Model (HMM): the states (POS tags) are in fact hidden from the view; the only observable events are the sequence of emitted symbols (words).

Simple Markov Model for POS

- Given **DT** as the current POS, what's the likelihood of POS_{n+1}:
  - NN ('the question')
  - JJ ('the happy girl')
  - RB ('the very happy girl')



*NOTE: this machine is incomplete!

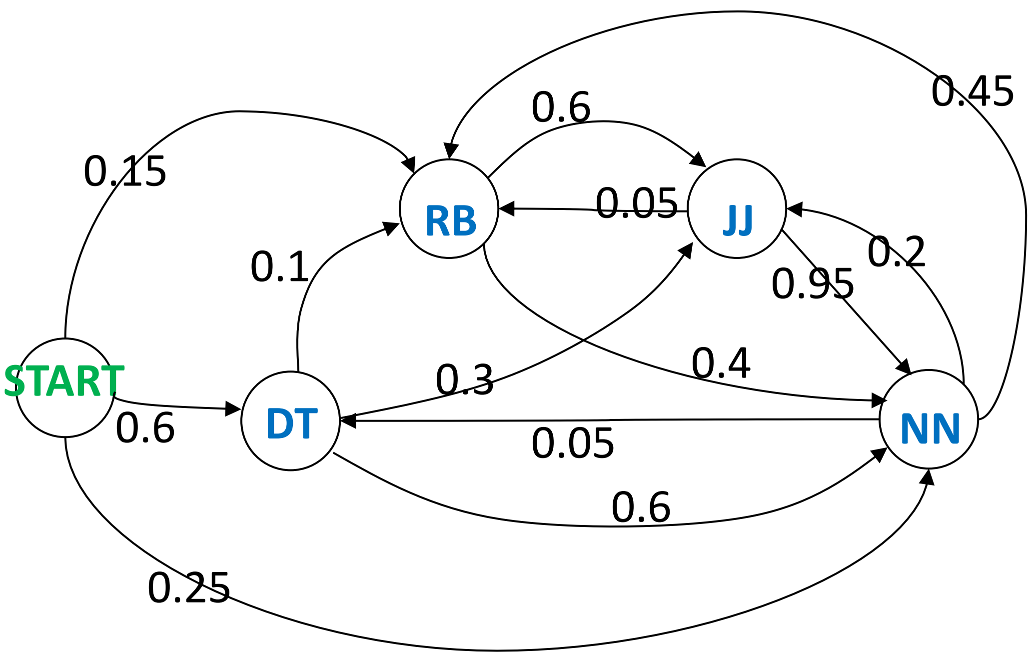
Which tag sequence is most likely:

- `DT NN JJ`
- `NN JJ RB`
- `RB JJ NN`
- `DT JJ NN`

*Penn Treebank tagset.

Simple Markov Model for POS

- Given **DT** as the current POS, what's the likelihood of POS_{n+1}:
  - NN ('the <u>question</u>')
  - JJ ('the <u>happy</u> girl')
  - RB ('the <u>very</u> happy girl')

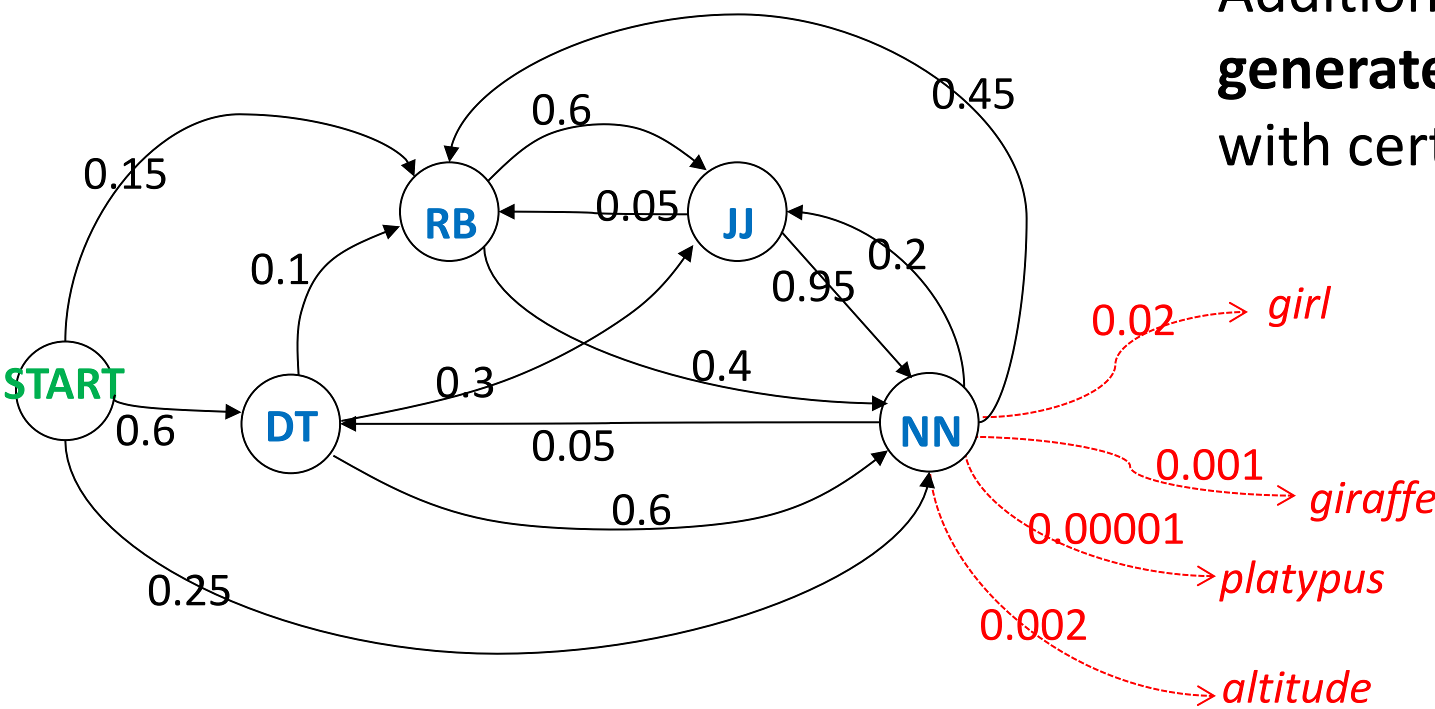



```
DT  NN  JJ
    = 0.6*0.6*0.2 = 0.072
NN  JJ  RB
    = 0.25*0.2*0.05 = 0.0025
RB  JJ  NN
    = 0.15*0.6*0.95 = 0.0855
DT  JJ  NN
    = 0.6*0.3*0.95 = 0.171
```

Where can we get **transition probability**? CORPUS.

What about words?

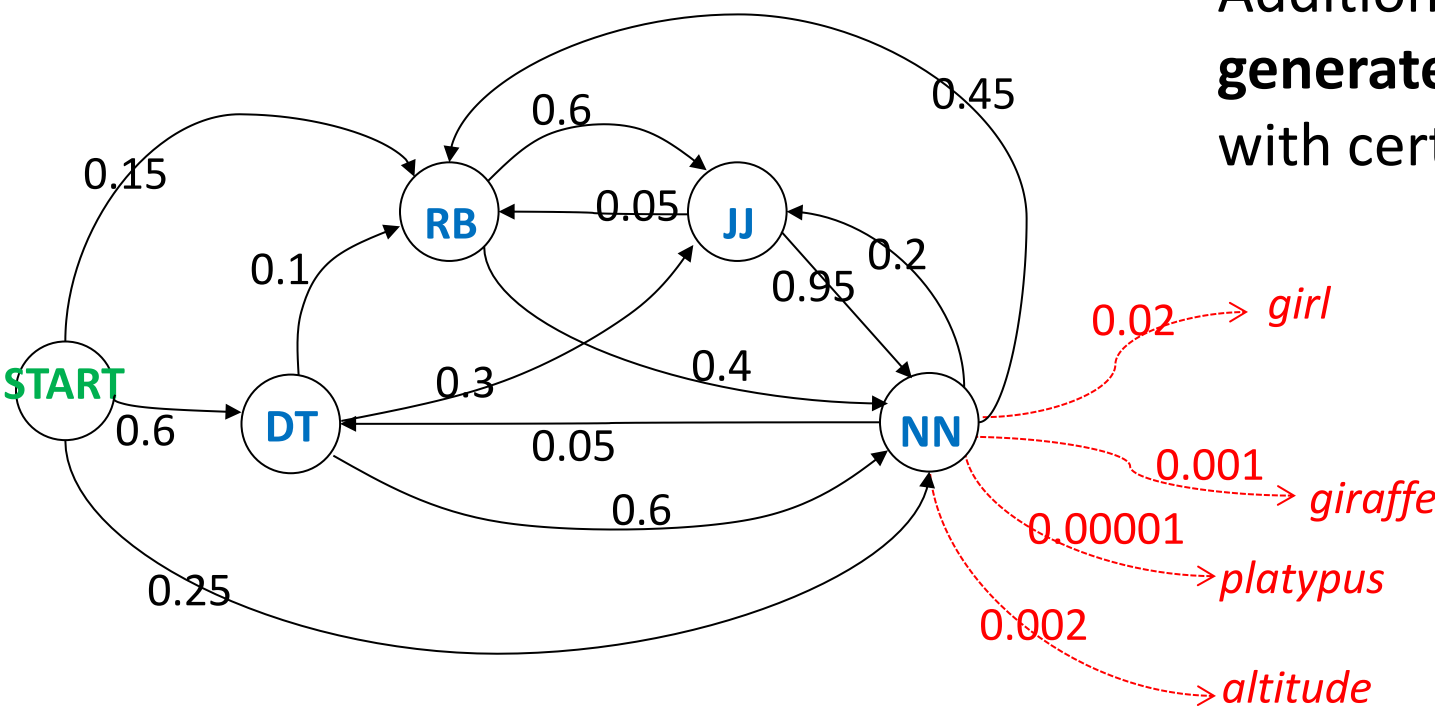
- So, DT JJ NN is a highly probable tag sequence, but ultimately the overall probability should also be about the *word sequence*:
  - *the happy girl, the stupendous giraffe, a bright/bad cold*



- Additionally, each state **generates tokens** (words) with certain **probability**.

HMM: transition (POS) + generation (word)

- So, DT JJ NN is a highly probable tag sequence, but ultimately the overall probability should also be about the *word sequence*:
  - *the happy girl, the stupendous giraffe, a bright/bad cold*



- Additionally, each state **generates tokens** (words) with certain **probability**.

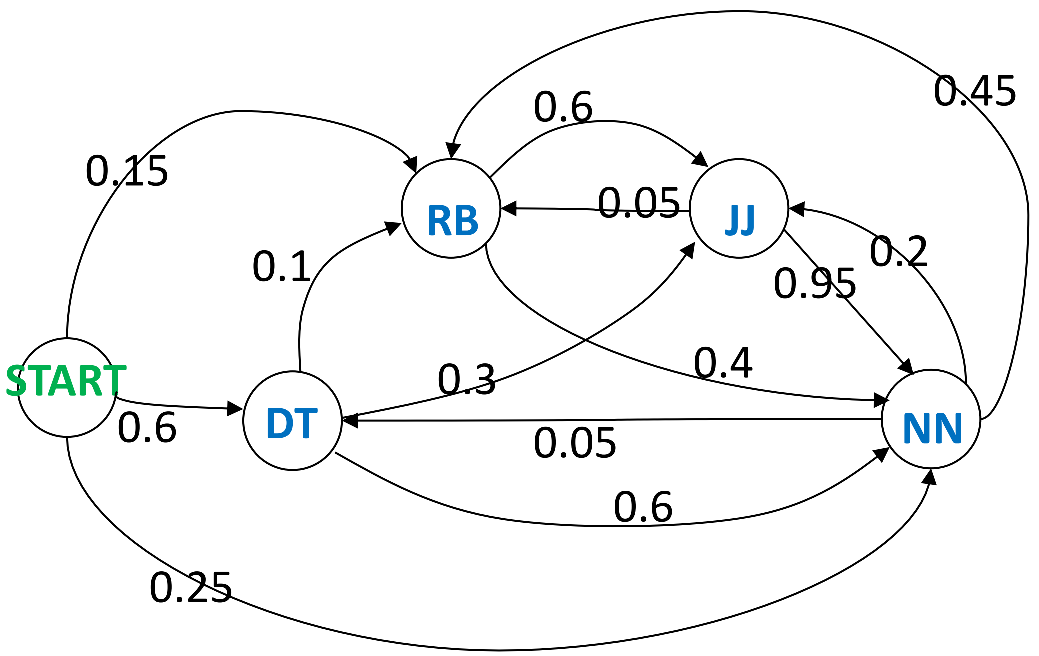
How to get **generation probability**? Corpus.

HMM: in a nutshell

- POS tagging using a HMM means:
  - Given the word sequence $w_1\ w_2\ w_3\ \ldots\ w_n$
  - Find the tag sequence $T_1\ T_2\ T_3\ \ldots\ T_n$ such that the probability of the particular word sequence occurring with the tag sequence is maximized.
  - $\arg\max_{T_1 T_2 T_3 \ldots Tn} p(T_1\ T_2\ T_3\ \ldots\ Tn,\ w_1\ w_2\ w_3\ \ldots\ wn)$
  - Algorithms exist that effectively compute this. (We will not get into them.)

HMM is built on *probabilistic* FSA

- Given DT as the current tag, what's the likelihood of:
  - NN ('the question')
  - JJ ('the happy girl')
  - RB ('the very happy girl')



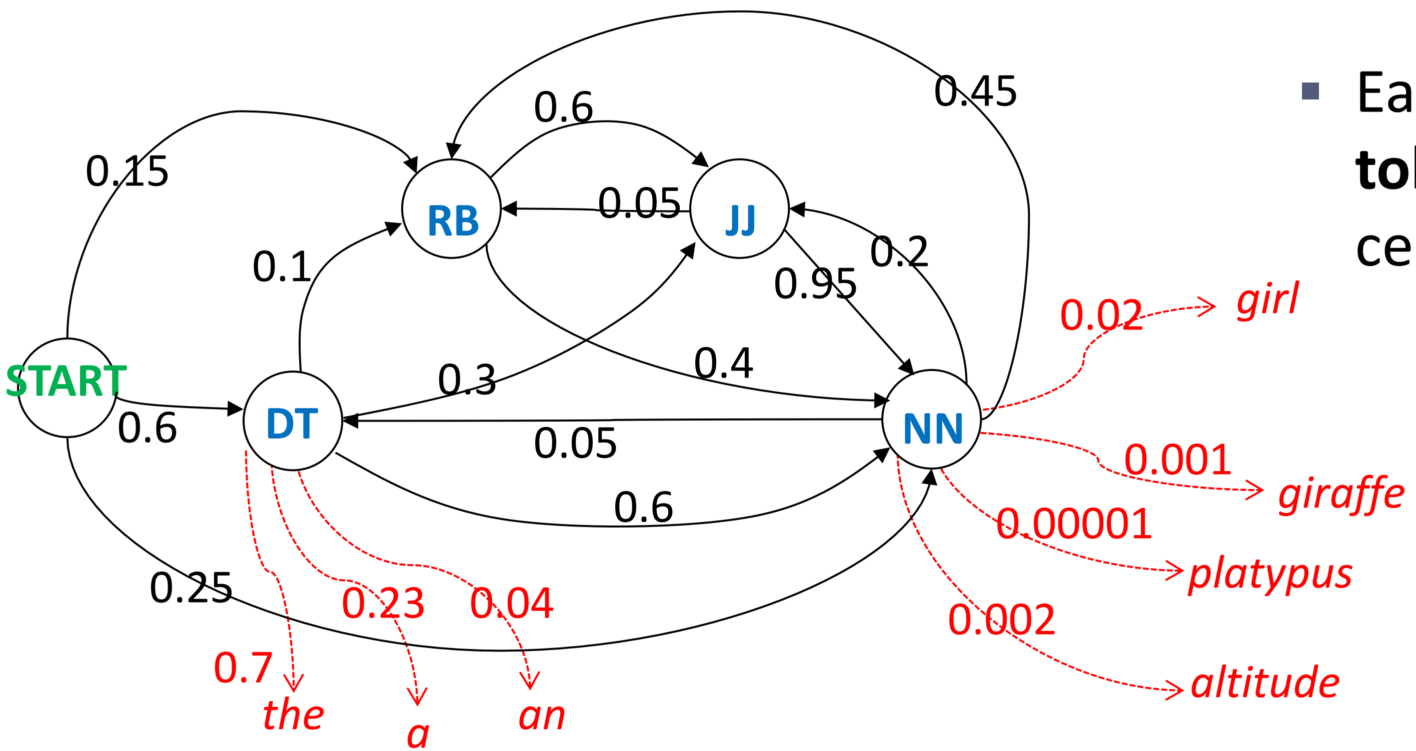
HMM's POS tag transition model is a ***probabilistic* FSA**! (with no arc labels)

Which tag sequence is most likely:

- `DT NN JJ`
- `NN JJ RB`
- `RB JJ NN`
- `DT JJ NN`

HMM: transition (POS) + generation (word)

- HMM combines POS tag sequence probability (DT → JJ → NN → ...) and the probability of certain words occurring with a POS (given DT tag, 'the' is 0.7 likely, and 'a' 0.23...)



- Each state **generates tokens** (words) with a certain **probability**.

Markov model (Markov chain)

- A **finite state machine** with **probabilistic state transitions**.
- Makes Markov assumption that the next state only depends on the current state and is independent of previous history.
- **Hidden** Markov Model (HMM): the states (POS tags) are in fact hidden from the view; the only observable events are the **sequence of emitted symbols** (words).

NLTK's HMM package is `nltk.tag.hmm`

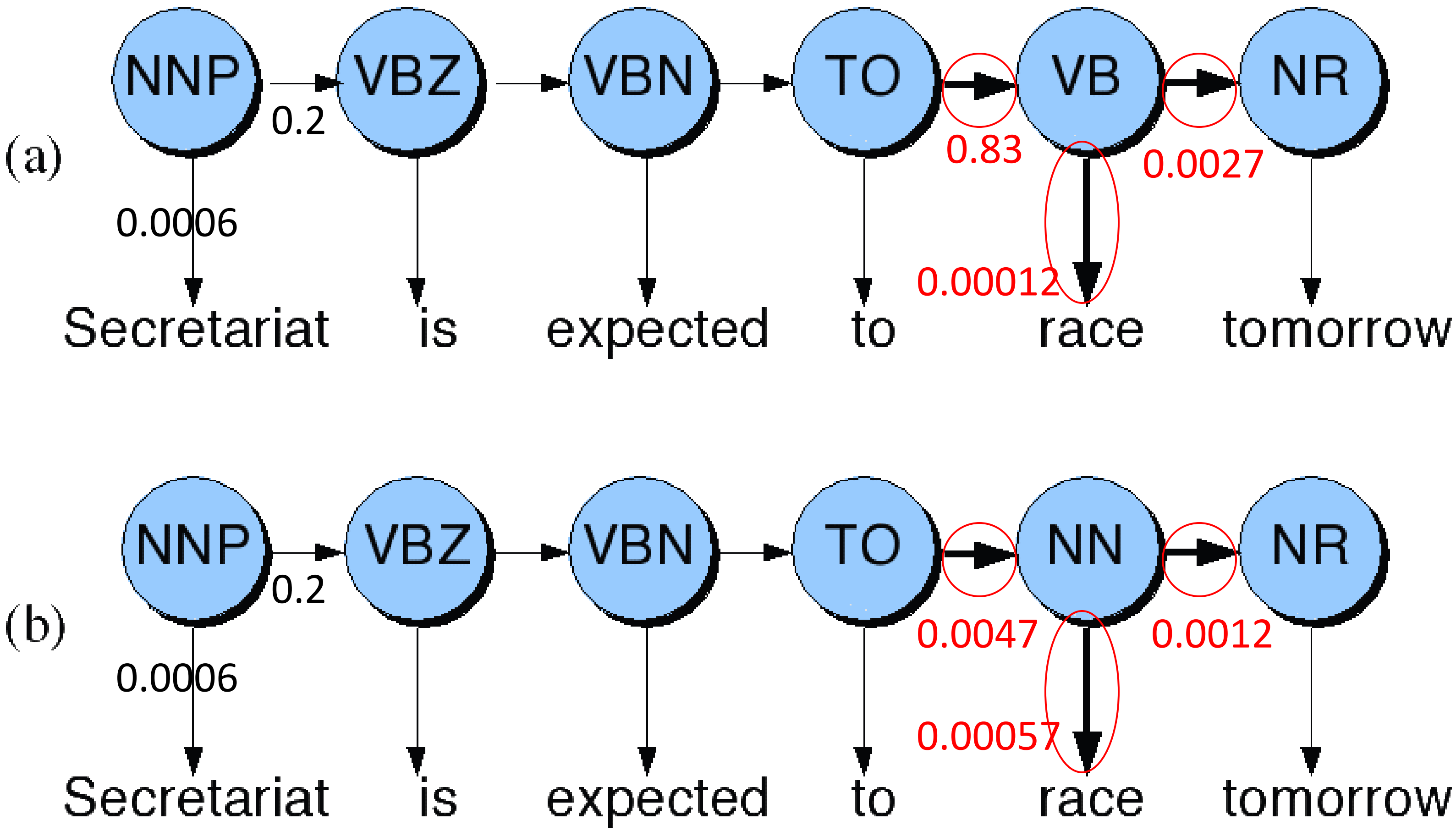
The NLTK book does not cover HMM.
For details, see J&M.

Verb or noun?

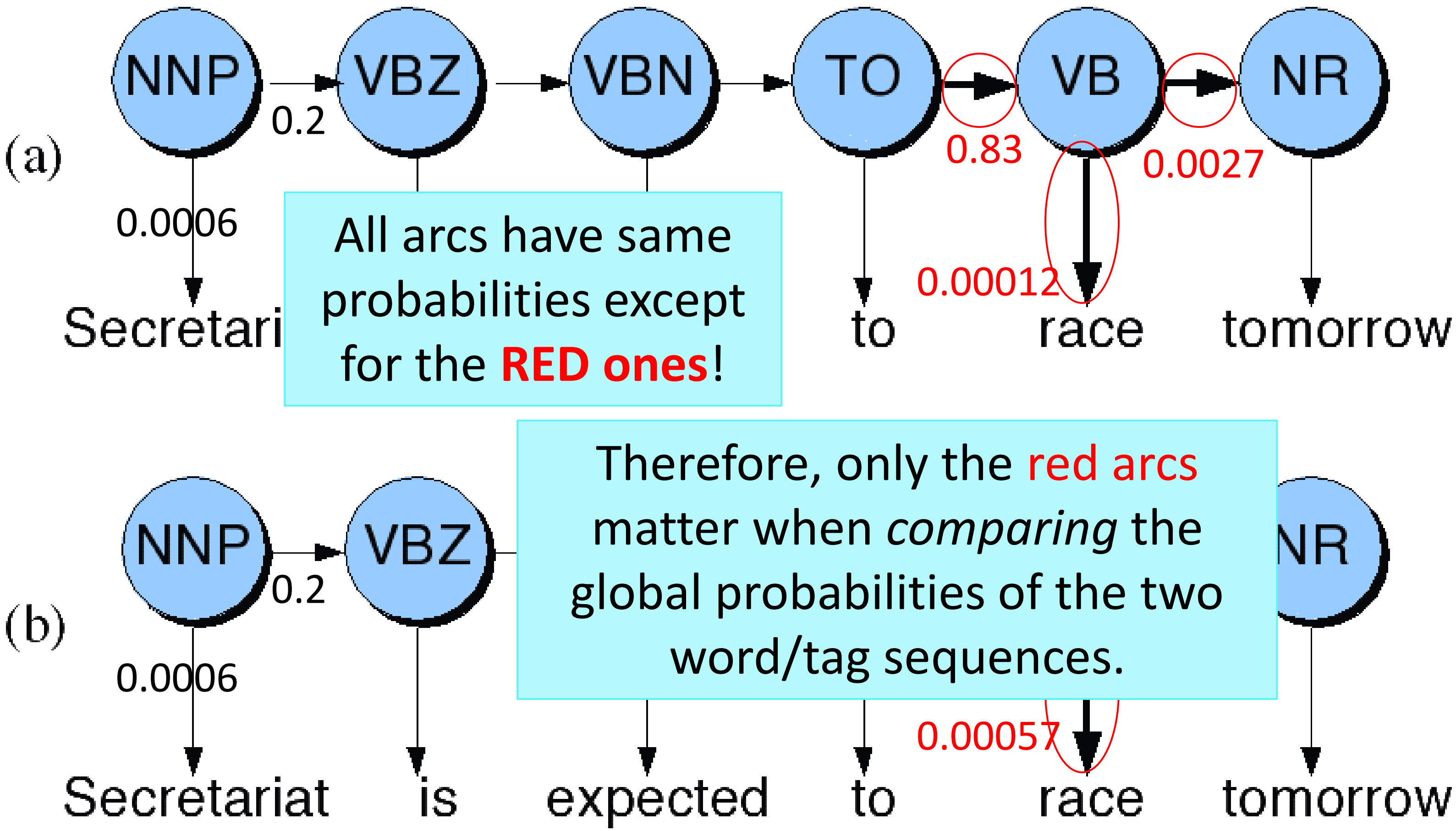
Secretariat	is	expected	to	race	tomorrow
NNP	VBZ	VBN	TO	VB NN	NR

*Penn Treebank tagset.

Resolving tag ambiguities in HMM



Resolving tag ambiguities in HMM



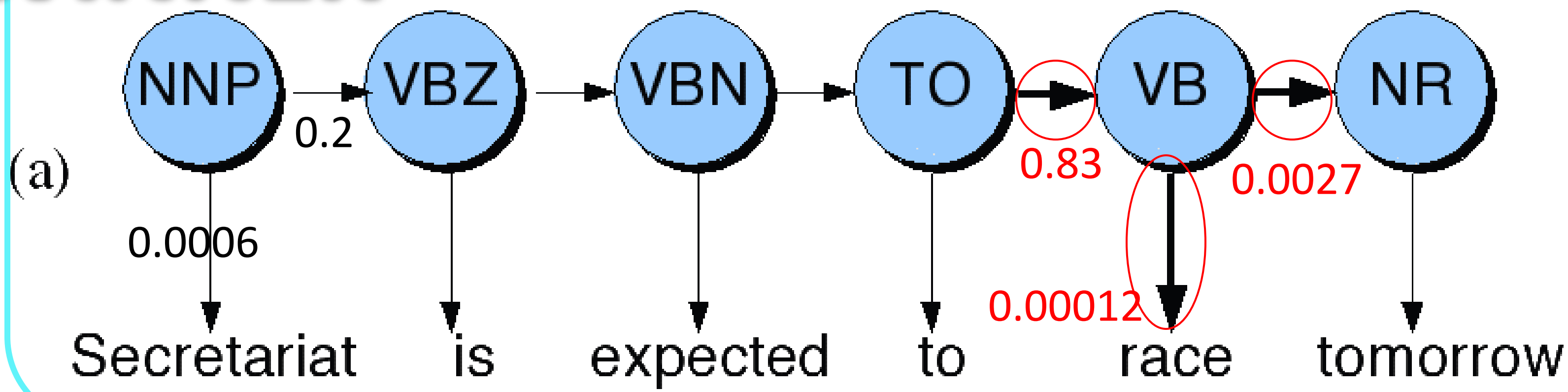
All arcs have same probabilities except for the **RED ones**!

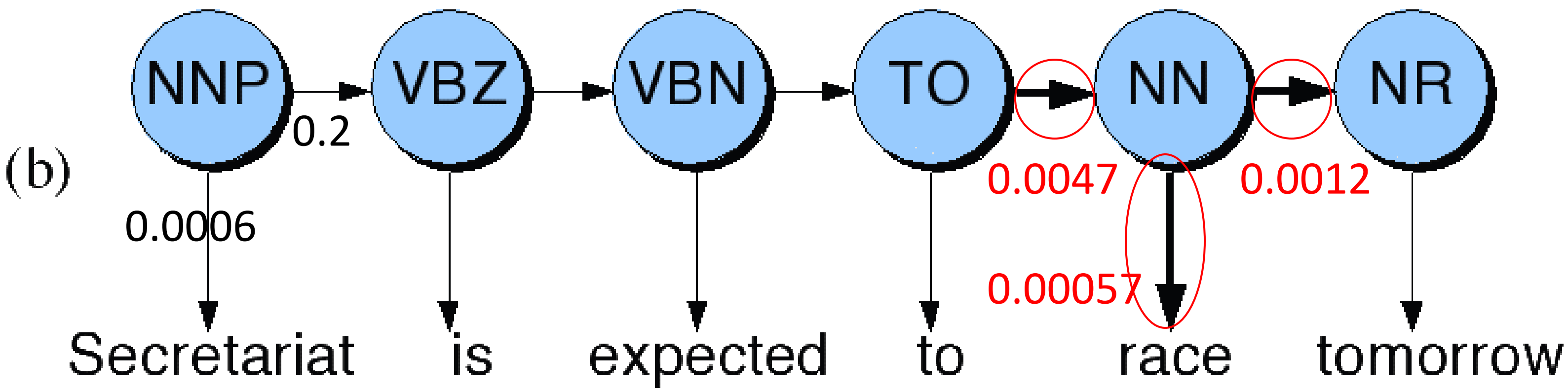
Therefore, only the red arcs matter when *comparing* the global probabilities of the two word/tag sequences.

HMM optimizes for global likelihood

WINNER

0.83*0.00012*0.0027 = **0.00000027**





0.0047*0.00057*0.0012 = **0.0000000032**

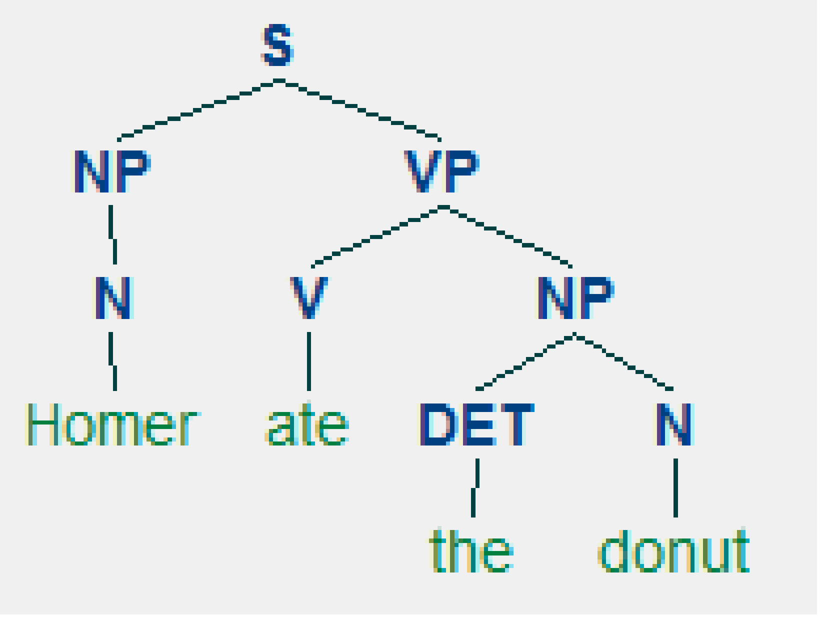
POS taggers: state-of-the-art

Below are some well-known POS taggers from various research groups:

- The Stanford POS Tagger
- CLAWS POS Tagger (uses the CLAWS tagset)
- Brill Tagger
- A list of state-of-the-art taggers on ACL web; they commonly use the Penn Treebank Wall Street Journal corpus

Syntactic trees

- Demo + lecture, in HTML document
  - https://sites.pitt.edu/~naraehan/ling1330/lecture21.html



Wrapping up

- Next class:
  - Continue with syntactic trees and parsing
  - NLTK book: 7.4.2 [Trees](), Ch.8 [Analyzing Sentence Structure]()
- Exercise 10 out
  - Getting started with trees

- Tomorrow: PyLing
  - 6pm, 2818 CL
  - About "prompt engineering", by Maya Asher

- Final exam schedule!
  - 12/12 (Thu) 4-5:50pm
  - At LMC's PC lab (G17 CL)