Part-of-Speech Tagging

L545

Spring 2013

POS Tagging Problem

- Given a sentence W1…Wn and a tagset of lexical categories, find the most likely tag T1..Tn for each word in the sentence
- Example

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

  People/NNS continue/VBP to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/JJ space/NN
- Note that many of the words may have unambiguous tags
  - But enough words are either ambiguous or unknown that it's a nontrivial task

More Details of the Problem

- How ambiguous?
  - Most words in English have only one Brown Corpus tag
    - Unambiguous (1 tag) 35,340 word types
    - Ambiguous (2- 7 tags) 4,100 word types = 11.5%
      - 7 tags: 1 word type "still"
  - But many of the most common words are ambiguous
    - Over 40% of Brown corpus tokens are ambiguous
- Obvious strategies may be suggested based on intuition
    - to/TO race/VB
    - the/DT race/NN
    - will/MD race/NN
  - This leads to hand-crafted rule-based POS tagging (J&M, 5.4)
- Sentences can also contain unknown words for which tags have to be guessed: Secretariat/NNP

Example English Part-of-Speech Tagsets

- Brown corpus - 87 tags
  - Allows compound tags
    - "I'm" tagged as PPSS+BEM
      - PPSS for "non-3rd person nominative personal pronoun" and BEM for "am, 'm"

- Others have derived their work from Brown Corpus
  - LOB Corpus: 135 tags
  - Lancaster UCREL Group: 165 tags
  - London-Lund Corpus: 197 tags.
  - BNC – 61 tags (C5)
  - PTB – 45 tags

- Other languages have developed other tagsets

PTB Tagset (36 main tags + punctuation tags)

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NP	Proper noun, singular
NPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PP	Personal pronoun
PP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Typical Problem Cases

- Certain tagging distinctions are particularly problematic

- For example, in the Penn Treebank (PTB), tagging systems do not consistently get the following tags correct:
  - NN vs NNP vs JJ, e.g., *Fantastic*
    - somewhat ill-defined distinctions
  - RP vs RB vs IN, e.g., *off*
    - pseudo-semantic distinctions
  - VBD vs VBN vs JJ, e.g., *hated*
    - non-local distinctions

POS Tagging Methods

- Two basic ideas to build from:
  - Establishing a simple baseline with unigrams
  - Hand-coded rules

- Machine learning techniques:
  - Supervised learning techniques
  - Unsupervised learning techniques

- We'll only provide an overview of the methods
  - Many of the details will be left to L645

A Simple Strategy for POS Tagging

- Choose the most likely tag for each ambiguous word, independent of previous words
  - i.e., assign each token the POS category it occurred as most often in the training set
  - e.g., race – which POS is more likely in a corpus?

- This strategy gives you 90% accuracy in controlled tests
  - So, this "unigram baseline" must always be compared against

Example of the Simple Strategy

- Which POS is more likely in a corpus (1,273,000 tokens)?

	NN	VB	Total
race	400	600	1000

- P(NN|race) = P(race&NN) / P(race) by the definition of conditional probability

  - P(race) ≅ 1000/1,273,000 = .0008
  - P(race&NN) ≅ 400/1,273,000 =.0003
  - P(race&VB) ≅ 600/1,273,000 = .0005

- And so we obtain:

  - P(NN|race) = P(race&NN)/P(race) = .0003/.0008 =.375
  - P(VB|race) = P(race&VB)/P(race) = .0004/.0008 = .625

Hand-coded rules

- Two-stage system:
  - Dictionary assigns all possible tags to a word
  - Rules winnow down the list to a single tag
    - Sometimes, multiple tags are left, if it cannot be determined
- Can also use some probabilistic information
- These systems can be highly effective, but they of course take time to write the rules.
  - We'll see an example later of trying to automatically learn the rules (transformation-based learning)

Hand-coded Rules: ENGCG System

- Uses 56,000-word lexicon which lists parts-of-speech for each word (using two-level morphology)
- Uses up to 3,744 rules, or constraints, for POS disambiguation

ADV-that rule

Given input "that" (ADV/PRON/DET/COMP)

If (+1 A/ADV/QUANT) #next word is adj, adverb, or quantifier

(+2 SENT_LIM) #and following word is a sentence boundary

(NOT -1 SVOC/A) #and the previous word is not a verb like

#*consider* which allows adjs as object complements

Then eliminate non-ADV tags

Else eliminate ADV tag

Machine Learning

- Machines can learn from examples
  - Learning can be supervised or unsupervised

- Given training data, machines analyze the data, and learn rules which generalize to new examples
  - Can be sub-symbolic (rule may be a mathematical function) e.g., neural nets
  - Or it can be symbolic (rules are in a representation that is similar to representation used for hand-coded rules)

- In general, machine learning approaches allow for more tuning to the needs of a corpus, and can be reused across corpora

1. TBL: A Symbolic Learning Method

- A method called error-driven Transformation-Based Learning (TBL) (Brill algorithm) can be used for symbolic learning
  - The rules (actually, a sequence of rules) are learned from an annotated corpus
  - Performs about as accurately as other statistical approaches

- Can have better treatment of context compared to HMMs (later)
  - rules which use the next (or previous) POS
    - HMMs just use P(Ti| Ti-1) or P(Ti| Ti-2 Ti-1)
  - rules which use the previous (next) word
    - HMMs just use P(Wi|Ti)

Rule Templates

- Brill's method learns transformations which fit different templates
  - Template: Change tag X to tag Y when previous word is W
    - Transformation: NN → VB when previous word = *to*
  - Change tag X to tag Y when next tag is Z
    - NN → NNP when next tag = NNP
  - Change tag X to tag Y when previous 1st, 2nd, or 3rd word is W
    - VBP → VB when one of previous 3 words = *has*
- The learning process is guided by a small number of templates (e.g., 26) to learn specific rules from the corpus
- Note how these rules sort of match linguistic intuition

Brill Algorithm (Overview)

- Assume you are given a training corpus G (for gold standard)
- First, create a tag-free version V of it … then do steps 1-4
- Notes:
  - As the algorithm proceeds, each successive rule covers fewer examples, but potentially more accurately
  - Some later rules may change tags changed by earlier rules

1. Initial-state annotator: Label every word token in V with most likely tag for that word type from G.
2. Consider every possible transformational rule: select the one that leads to the most improvement in V using G to measure the error
3. Retag V based on this rule
4. Go back to 2, until there is no significant improvement in accuracy over previous iteration

Error-driven method

- How does one learn the rules?
- The TBL method is error-driven
  - The rule which is learned on a given iteration is the one which reduces the error rate of the corpus the most, e.g.:
  - Rule 1 fixes 50 errors but introduces 25 more → net decrease is 25
  - Rule 2 fixes 45 errors but introduces 15 more → net decrease is 30

  → Choose rule 2 in this case
- We set a stopping criterion, or threshold → once we stop reducing the error rate by a big enough margin, learning is stopped

Brill Algorithm (More Detailed)

- 1. Label every word token with its most likely tag (based on lexical generation probabilities).
- 2. List the positions of tagging errors and their counts, by comparing with "truth" (T)
- 3. For each error position, consider each instantiation I of X, Y, and Z in Rule template.
  - If Y=T, increment improvements[I], else increment errors[I].
- 4. Pick the I which results in the greatest error reduction, and add to output
  - VB NN PREV1OR2TAG DT improves on 98 errors, but produces 18 new errors, so net decrease of 80 errors
- 5. Apply that I to corpus
- 6. Go to 2, unless stopping criterion is reached

Most likely tag:

P(NN|race) = .98

P(VB|race) = .02

Is/VBZ expected/VBN to/TO race/**NN** tomorrow/NN

Rule template: *Change a word from tag X to tag Y when previous tag is Z*

Rule Instantiation for above example: NN VB PREV1OR2TAG TO

Applying this rule yields:

Is/VBZ expected/VBN to/TO race/**VB** tomorrow/NN

Example of Error Reduction

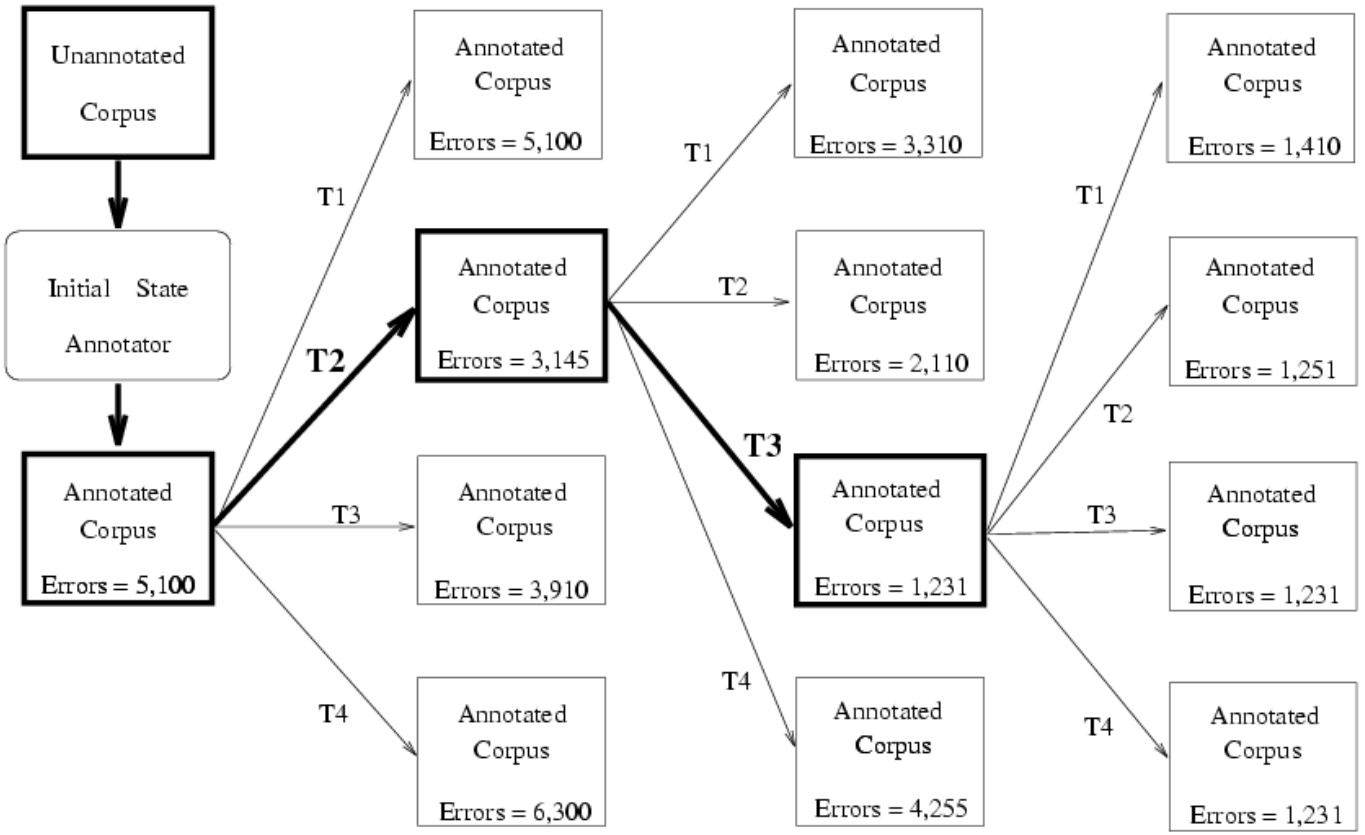


Figure 2

From Eric Brill (1995):
Computational Linguistics, 21, 4, p. 7

Rule ordering

- One rule is learned with every pass through the corpus.
  - The set of final rules is what the final output is
  - Unlike HMMs, such a representation allows a linguist to look through and make more sense of the rules

- The rules are learned iteratively & must be applied in an iterative fashion.
  - At one stage, it may make sense to change NN to VB after *to*
  - But at a later stage, it may make sense to change VB back to NN in the same context, e.g., if the current word is *school*

Example of Learned Rule Sequence

- 1. NN VB PREVTAG TO
  - to/TO race/NN->VB
- 2. VBP VB PREV1OR20R3TAG MD
  - might/MD vanish/VBP-> VB
- 3. NN VB PREV1OR2TAG MD
  - might/MD not/RB reply/NN -> VB
- 4. VB NN PREV1OR2TAG DT
  - the/DT great/JJ feast/VB->NN
- 5. VBD VBN PREV1OR20R3TAG VBZ
  - He/PP was/VBZ killed/VBD->VBN by/IN Chapman/NNP

Handling Unknown Words

- Can also use the Brill method to learn how to tag unknown words
- Instead of using surrounding words and tags, use affix info, capitalization, etc.
  - Guess NNP if capitalized, NN otherwise.
  - Or use the tag most common for words ending in the last 3 letters.
  - etc.
- TBL has also been applied to some parsing tasks

Example Learned Rule Sequence for Unknown Words

Change tag:

1. From **common noun** to **plural common noun** if the word has suffix **-s**[12]
2. From **common noun** to **number** if the word has character **.**
3. From **common noun** to **adjective** if the word has character **-**
4. From **common noun** to **past participle verb** if the word has suffix **-ed**
5. From **common noun** to **gerund or present participle verb** if the word has suffix **-ing**
6. To **adjective** if adding the suffix **-ly** results in a word
7. To **adverb** if the word has suffix **-ly**
8. From **common noun** to **number** if the word **$** ever appears immediately to the left
9. From **common noun** to **adjective** if the word has suffix **-al**
10. From **noun** to **base form verb** if the word **would** ever appears immediately to the left.

Insights on TBL

- TBL takes a long time to train, but is relatively fast at tagging once the rules are learned
- The rules in the sequence may be decomposed into non-interacting subsets, i.e., only focus on VB tagging (need to only look at rules which affect it)
- In cases where the data is sparse, the initial guess needs to be weak enough to allow for learning
- Rules become increasingly specific as you go down the sequence.
  - However, the more specific rules generally don't overfit because they cover just a few cases

2. HMMs: A Probabilistic Approach

- What you want to do is find the "best sequence" of POS tags T=T1..Tn for a sentence W=W1..Wn.
  - (Here T1 is pos_tag(W1)).

  find a sequence of POS tags T that maximizes P(T|W)
- Using Bayes' Rule, we can say

  P(T|W) = P(W|T)*P(T)/P(W)
- We want to find the value of T which maximizes the RHS

  → denominator can be discarded (same for every T)

➔ Find T which maximizes

P(W|T) * P(T)

- Example: He will race
- Possible sequences:
  - He/PRP will/MD race/NN
  - He/PRP will/NN race/NN
  - He/PRP will/MD race/VB
  - He/PRP will/NN race/VB
- W = W1 W2 W3

  = He will race
- T = T1 T2 T3
  - Choices:
    - T= PRP MD NN
    - T= PRP NN NN
    - T = PRP MD VB
    - T = PRP NN VB

Independence Assumptions

- Assume that current event is based only on previous n-1 events (for a bigram model, it's based only on previous 1 event)
- $P(T1 \ldots Tn) \cong \Pi_{i=1,\, n} P(Ti|\, Ti\text{-}1)$
  - *assumes that the event of a POS tag occurring is independent of the event of any other POS tag occurring, except for the immediately previous POS tag*
    - *From a linguistic standpoint, this seems an unreasonable assumption, due to long-distance dependencies*
- $P(W1 \ldots Wn \mid T1 \ldots Tn) \cong \Pi_{i=1,\, n} P(Wi|\, Ti)$
  - *assumes that the event of a word appearing in a category is independent of the event of any surrounding word or tag, except for the tag at this position.*

Hidden Markov Models

- Linguists know both these assumptions are incorrect!
  - But, nevertheless, statistical approaches based on these assumptions work pretty well for part-of-speech tagging

- In particular, with Hidden Markov Models (HMMs)
  - Very widely used in both POS-tagging and speech recognition, among other problems
  - A Markov model, or Markov chain, is just a weighted Finite State Automaton

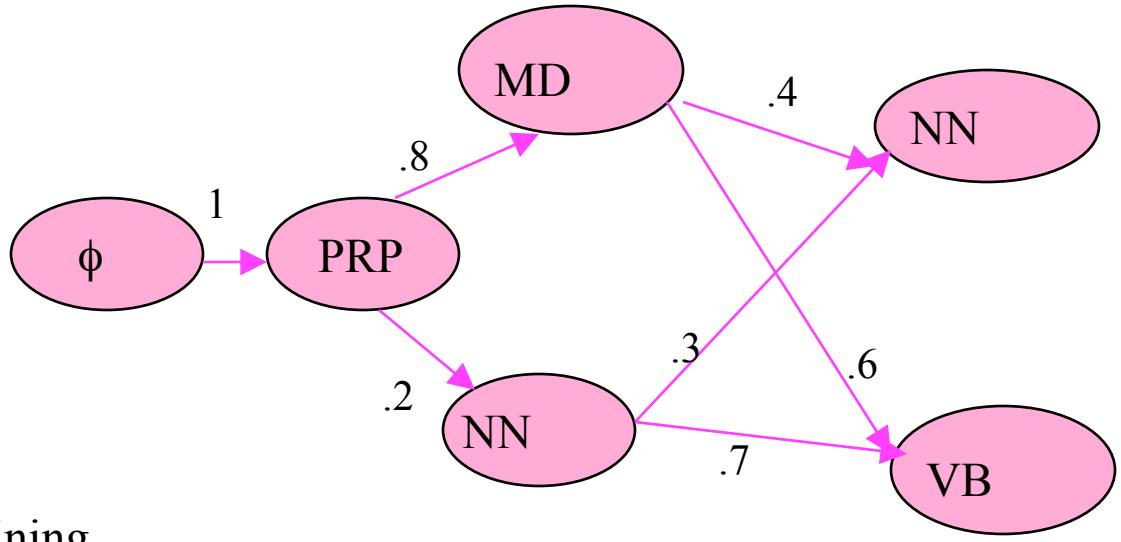
POS Tagging Based on Bigrams

- Problem: Find T which maximizes P(W | T) * P(T)
  - Here W=W1..Wn and T=T1..Tn
- Using the bigram model, we get:
  - Transition probabilities (prob. of transitioning from one state/tag to another):
    - $P(T1 \ldots Tn) \cong \Pi_{i=1,\, n} P(Ti|Ti\text{-}1)$
  - Emission probabilities (prob. of emitting a word at a given state):
    - $P(W1 \ldots Wn \mid T1 \ldots Tn) \cong \Pi_{i=1,\, n} P(Wi|\, Ti)$
- So, we want to find the value of T1..Tn which maximizes:

$$\Pi_{i=1,\, n} \quad P(Wi|\, Ti) \quad * \quad P(Ti|\, Ti\text{-}1)$$

Using POS bigram probabilities: transitions

- $P(T1 \ldots Tn) \cong \Pi_{i=1,\,n} P(Ti|Ti-1)$
- Example: He will race
- Choices for T=T1..T3
  - T= PRP MD NN
  - T= PRP NN NN
  - T = PRP MD VB
  - T = PRP NN VB
- POS bigram probs from training corpus can be used for P(T)
- P(PRP-MD-NN)=1*.8*.4 =.32



POS bigram probs

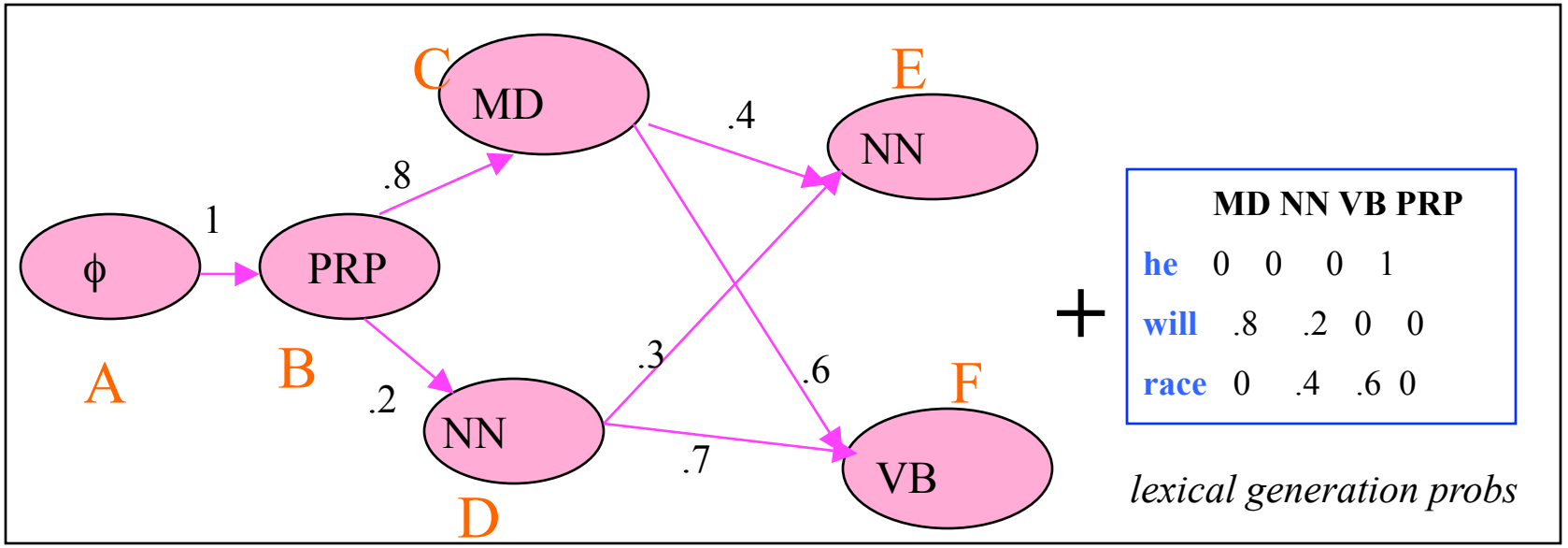
C|R	**MD**	**NN**	**VB**	**PRP**
MD		.4	.6	
NN		.3	.7	
PRP	.8	.2		
ϕ			1	

Factoring in lexical generation probabilities

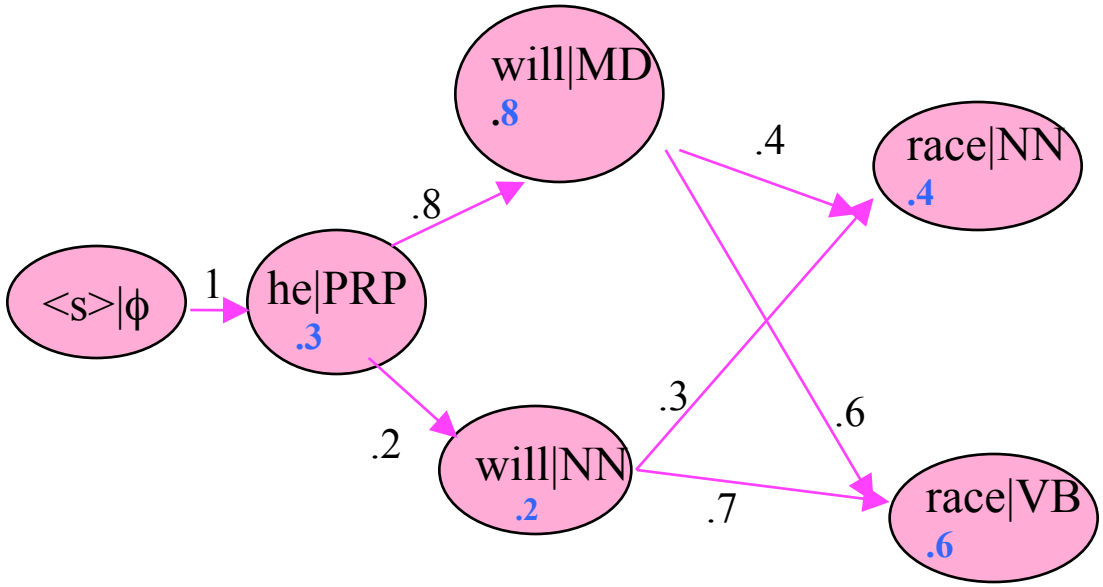
- From the training corpus, we need to find the Ti which maximizes

  $\Pi_{i=1, n}$ P(Wi| Ti) * P(Ti| Ti-1)

- So, we'll need to factor the lexical generation (emission) probabilities, somehow:



Adding emission probabilities



	MD	NN	VB	PRP
he	0	0	0	.3
will	.8	.2	0	0
race	0	.4	.6	0

lexical generation probs

C|R	MD	NN	VB	PRP
MD		.4	.6	
NN		.3	.7	
PP	.8	.2		
ϕ			1	

pos bigram probs

Dynamic Programming

- In order to find the most likely sequence of categories for a sequence of words, we don't need to enumerate all possible sequences of categories.

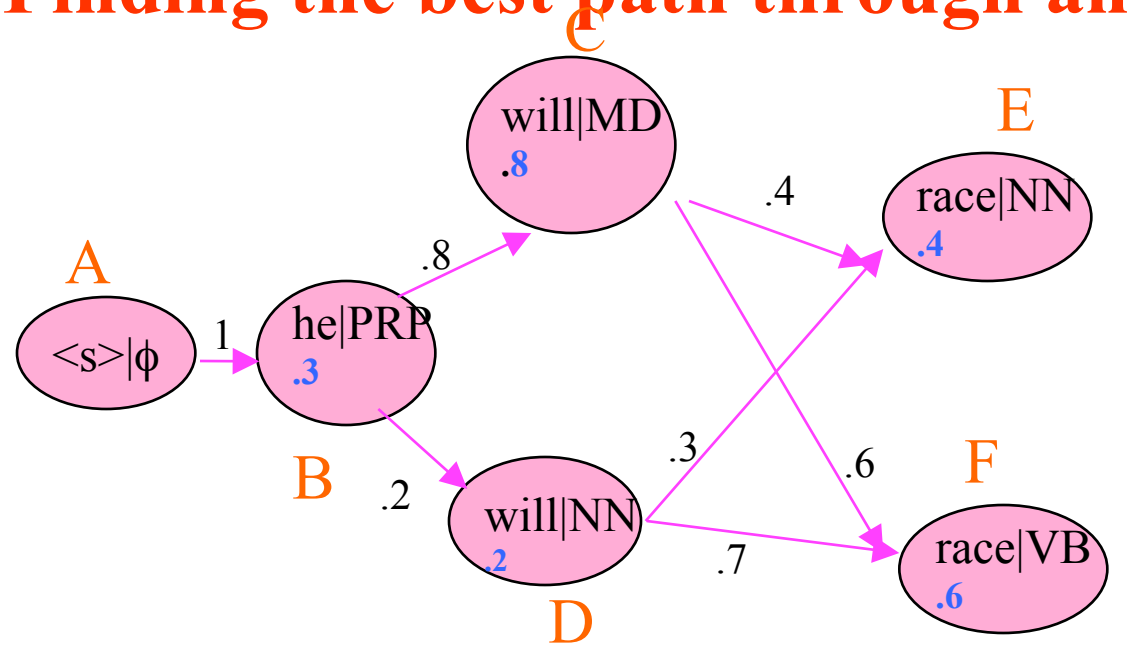
- Because of the Markov assumption, if you keep track of the most likely sequence found so far for each possible ending category, you can ignore all the other less likely sequences.
  - i.e., multiple edges coming into a state, but only keep the value of the most likely path
  - This is a use of dynamic programming

- The algorithm to do this is called the Viterbi algorithm.

The Viterbi algorithm

1. Assume we're at state I in the HMM
   - States H1 … Hm all come into I
2. Obtain
   - the best probability of each previous state H1…Hm
   - the transition probabilities: P(I|H1), … P(I|Hm)
   - the emission probability for word w at I: P(w|I)
3. Multiple the probabilities for each new path:
   - e.g., P(Hi,I) = Best(H1)*P(I|H1)*P(w|I)
4. One of these states (H1…Hm) will give the highest probability
   - Only keep the highest probability when using I for the next state

Finding the best path through an HMM



	MD	NN	VB	PRP
he	0	0	0	.3
will	.8	.2	0	0
race	0	.4	.6	0

lexical generation probs

Viterbi algorithm

- Best(I) = Max $_{H<I}$ [Best(H)* P(I|H)]* P(w|I)
- Best(A) = 1
- Best(B) = Best(A) * P(PRP|ϕ) * P(he|PRP) = 1*1*.3=.3
- Best(C)=Best(B) * P(MD|PRP) * P(will|MD) = .3*.8*.8= .19
- Best(D)=Best(B) * P(NN|PRP) * P(will|NN) = .3*.2*.2= .012
- Best(E) = Max [Best(C)*P(NN|MD), Best(D)*P(NN|NN)] * P(race|NN) = .03
- Best(F) = Max [Best(C)*P(VB|MD), Best(D)*P(VB|NN)] * P(race|VB)= .068

Unsupervised learning

- Unsupervised learning:
  - Use an unannotated corpus for training data
  - Instead, will have to use another database of knowledge, such as a dictionary of possible tags

- Unsupervised learning use the same general techniques as supervised, but there are important differences

- Advantage is that there is more unannotated data to learn from
  - And annotated data isn't always available

Unsupervised Learning: TBL

- With TBL, we want to learn rules of patterns, but how can we learn the rules if there's no annotated data?

- Main idea: look at the distribution of unambiguous words to guide the disambiguation of ambiguous words

- Example: *the can*, where *can* can be a noun, modal, or verb
- Let's take unambiguous words from dictionary and count their occurrences after *the*
  - *the elephant*
  - *the guardian*

- Conclusion: immediately after *the*, nouns are more common than verbs or modals

Unsupervised TBL

- Initial state annotator
  - Supervised: assign random tag to each word
  - Unsupervised: for each word, list *all* tags in dictionary
- The templates change accordingly …
- Transformation template:
  - Change tag (set) X of word to tag {Y} if the previous (next) tag (word) is Z, where X is a set of 2 or more tags
  - Don't change any other tags

Error Reduction in Unsupervised Method

- Let a rule to change **X** to Y in context C be represented as Rule(**X**, Y, C).
  - Rule1: {VB, MD, NN} **NN** PREVWORD the
  - Rule2: {VB, MD, NN} VB PREVWORD the
- Idea:
  - since annotated data isn't available, score rules so as to prefer those where Y appears much more frequently in the context C than all others in **X**
    - frequency is measured by counting unambiguously tagged words
    - so, prefer {VB, MD, NN} **NN** PREVWORD the

      to {VB, MD, NN} VB PREVWORD the

      since unambiguous nouns are more common in a corpus after *the* than unambiguous verbs