HMM
and
Part of Speech Tagging

Adam Meyers

New York University

Outline

- Parts of Speech Tagsets
- Rule-based POS Tagging
- HMM POS Tagging
- Homework Assignment

Part of Speech Tags Standards

- There is no standard set of parts of speech that is used by all researchers for all languages.
- The most commonly used English tagset is that of the Penn Treebank at the University of Pennsylvania:
  - http://repository.upenn.edu/cgi/viewcontent.cgi?article=1603&context=cis_reports
  - Page 5 of: http://aclweb.org/anthology/J/J93/J93-2004.pdf
    - Provides list
- To map several POS tagsets to each other, see Table 1 in:
  - http://nlp.cs.nyu.edu/meyers/Annotation%20Compatibility%20Working%20Group%20Report%202006.html
- POS tagsets:
  - Assume Particular Tokenizations, e.g., ***Mary's*** → ***Mary + 's***
  - Distinguish inflections: e.g., *eat*/**VB**, *eat*/VBP, *eats*/**VBZ**, *ate*/**VBD**
  - Different instances of the same string can have different tags
    - *She wants to eat*/**VB**; *They eat*/**VBP**. *He eats*/**VBZ**, *Those are good eats*/**NNS**
- **Annotators & POS taggers assign tags to each token in a sentence, no exceptions**

The Penn Treebank II POS tagset

- Verbs: VB, VBP, VBZ, VBD, VBG, VBN
  - base, present-non-3rd, present-3rd, past, -ing, -en
- Nouns: NNP, NNPS, NN, NNS
  - proper/common, singular/plural (singular includes mass + generic)
- Adjectives: JJ, JJR, JJS (base, comparative, superlative)
- Adverbs: RB, RBR, RBS, RP (base, comparative, superlative, particle)
- Pronouns: PRP, PP$ (personal, possessive)
- Interogatives: WP, WP$, WDT, WRB (compare to: PRP, PP$, DT, RB)
- Other Closed Class: CC, CD, DT, PDT, IN, MD
- Punctuation: **# $. , : ( ) “ ” '' ' `**
- Weird Cases: FW(***deja vu***), SYM (***@***), LS (***1, 2, a, b***), TO (***to***), POS(***'s, '***), UH (***no, OK, well***), EX (***it/there***)
- Newer tags: HYPH, PU

Part of Speech Tagging

- POS taggers assign 1 POS tag to each input token
  - *The*/**DT** *silly*/**JJ** *man*/**NN** *is*/**VBZ** *a*/**DT** *professor*/**NN** ./**PU**
- Different ways of breaking down POS tagging:
  - Use separate "tokenizer", program that divides string into list of tokens – POS tagger processes output
  - Incorporate tokenizer into POS tagger
- Different ways of breaking down parsing:
  - Use separate POS tagger – output of tagger is input to parser
  - Assign POS tags as part of parsing (assumed previously)
- Accurate POS tagging is "easier" than accurate parsing
  - POS tags may be sufficient information for some tasks

Some Tokenization Rules for English

- 1) Divide at spaces and hyphens.
- 2) Divide before punctuation that is followed by: a space or the end of the line
  - Define punctuation as any non-letter/non-number:
    - ***`!@#$%^&*()-_+={[}]\|:;'”<,>.?/***
  - Punctuation followed by a space, other punctuation, or at the end of line should be separated from words:
    - ***...and he left.”) → and he left . ”)***
- 3) Break off the following as separate tokens when followed by a space or end of line:
  - ***'s***, ***n't***, ***'d***, ***'ve***, ***'m***, ***'ll***, ***'re***, … (a short list)
- 4) Abbreviations are exceptions to rule 2:
  - Period after abbreviations should not be separate from words
    - Most cases covered by list of 100 items (or if sentence end is known)
  - Final periods are not duplicated after abbreviations (consistency issues)
    - These periods serve 2 functions simultaneously (argument for duplication)
    - These periods occupy a single character position
      - argument against duplication – difficulty with calculating character offsets

Sentence Boundaries

- Most POS taggers assume sentence divisions
- Sample sentence splitting rules:
  - End sentence after . ? !, possibly others (:;...) Begin quotes are part of next sentence. End quotes are part of previous sentence.
  - But post-abbreviation (inc, co, ...) periods are ambiguous
    - next character is lowercase – not sentence end
    - next character is uppercase or number – possible sentence end
- Most POS taggers assume sentence boundaries are given.
- Multiple sentences within quotes are assumed separate.
  - <S> ***She said, "This is the way things are.*** </S>
  - <S> ***This is this.***</S> <S> ***That is that."***</S>
- **Sentence Splitting is a potentially good Final Project Topic**
  - **http://www.aclweb.org/anthology/C12-2096**
  - **http://aclweb.org/anthology/D18-2012**
  - **https://github.com/google/sentencepiece**

Rule-based POS Tagger

- Method
  - Assign lists of potential POS tags to each word based on dictionary
  - Manual rules for Out of Vocabulary (OOV) words
    - Ex: Non-initial capital → NNP; ends in S → VBZ|NNS; default → NN|JJ; etc.
  - Apply hand-written constraints until each word has only one possible POS
- Sample Constraints:
  - 1) DT cannot immediately precede a verb
  - 2) No verb can immediately precede a tensed verb: VBZ, VBP, VBD
    - Untensed: VB (base form), VBN & VBG (past & present participles)
- Example:
  - The/DT book/{NN|VB|VBP} is/VBZ on/IN the/DT table{NN|VB|VBP}
  - The/DT book/NN is/VBZ on/IN the/DT table/NN
    - DT cannot precede VB or VBP
    - VBZ cannot be preceded by VB or VBP

Probability

- Estimate of probability of future event based on past observations

$$P(event) = \frac{\text{num of events}}{\text{num of trials}}$$

- Conditional Probability: probability of X given Y

$$P(X|Y) = \frac{P(X,Y)}{P(Y)}$$

- Examples relating to POS tags (previous examples with word N-grams):
  - Out of 200 ***DT*** tags, 150 of them are tagging the word ***the***
    - If a word is tagged ***DT***, there is a 75% chance that word is ***the***
    - Example of likelihood probability
  - The POS after a ***DT*** is ***NN*** 120 times and ***JJ*** 60 times:
    - A word following ***DT*** is
      - 120/200 = 60% likely to be a singular noun (***NN***)
      - 60/200 = 30% likely to be a base adjective (***JJ***)
    - Examples of transition probability (probabilty of tag NN or JJ, given previous tag DT)

More Math Terminology

- N instances of a variable looked at individually:

  X_1^n is the same as $\{X_1, X_2, X_{3,} \ldots, X_n\}$ in sequence
- The product of instances of X from 1 to n

  $$\prod_{i=1}^{n} P(X_i)$$
- Max = the maximum number in a set
- Argmax = the choice of variable values that maximizes a formula (example in 2 slides)

Probabilistic Models of POS tagging

- For tokens $w_1, \ldots, w_n$, find the most probable corresponding sequence of possible tags $t_1, \ldots, t_n$
  - We assume that *probable* means something like "most frequently observed in some manually tagged corpus of words".
- Penn Treebank II (a common training corpus)
  - 1 million words from the Wall Street Journal
  - Tagged for POS (and other attributes)
- The specific sequence (sentence) is not in the training corpus
  - Therefore the actual "probability" is 0
  - Common practice: estimate probability given assumptions, e.g.,
    - Assume that we can estimate probability of whole tag sequence by multiplying simpler probabilities, e.g., sequences of 2 consecutive tags

Probabilistic Assumptions of HMM Tagging

- $\hat{t} = \underset{t_1^n}{argmax}\, P\left(t_1^n | w_1^n\right)$
  - Choose the tag sequence of length n that is most probable given the input token sequence
- Bayes Rule:
  - $P\left(x | y\right) = \frac{P\left(y | x\right) P\left(x\right)}{P\left(y\right)}$
  - Way to derive the probability of x given y when you know: the probability of ***y*** given ***x***, the probability of ***x*** and the probability of ***y***
- Applying Bayes Rule to Tag Probability
  - $\hat{t} = \underset{t_1^n}{argmax}\, \frac{P\left(w_1^n | t_1^n\right) P\left(t_1^n\right)}{P\left(w_1^n\right)}$

Simplifying Assumptions for HMMs

- Simplification: Drop the denominator
  - Denominator is same for all the tag sequences (the word sequence is given)
  - $\hat{t} = \underset{t_1^n}{argmax} P(w_1^n | t_1^n) P(t_1^n)$
  - For each tag sequence calculate the product of:
    - The probability of the word sequence given the tag sequence (**likelihood**)
    - The probability of the tag sequence (**prior probability**)
  - Still too hard
- 2 simplifying assumptions make it possible to estimate the probability of tag sequences given word sequences:
  - 1) If the probability of a word is only dependent on its own POS tag,
    - $P(w_1^n | t_1^n) \approx \prod_{i=1}^{n} P(w_i | t_i)$
  - 2) If the probability of a POS tag is only dependent on the previous POS tag,
    - $P(t^n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1})$
- The result of these assumptions: $\hat{t} \approx \underset{t_1^n}{argmax} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$
- Note: B & E represent tag before/after sentence
- HMM taggers are fast and achieve accuracy scores of about 93-95%

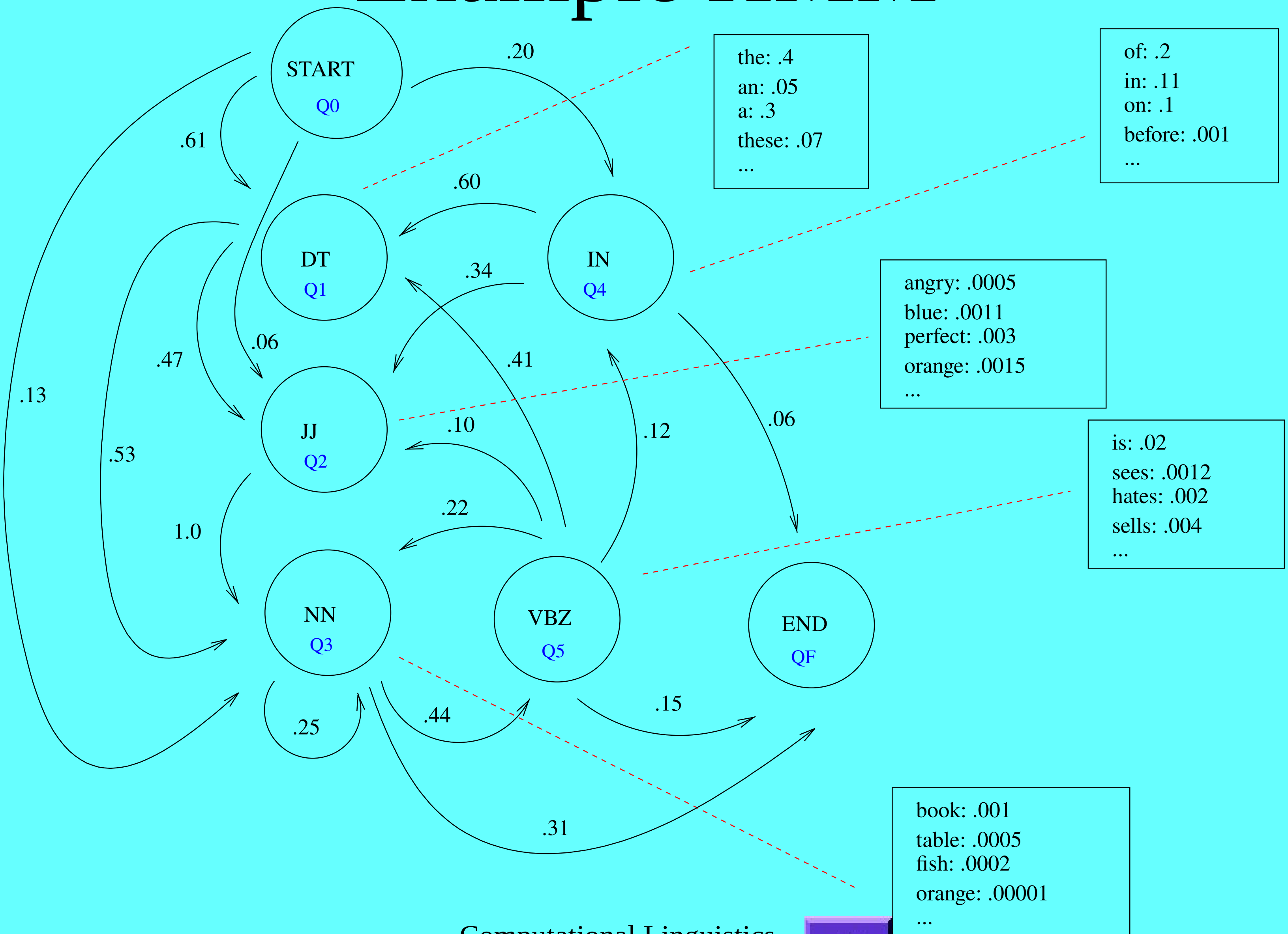
Estimating Probability of $\hat{t}$

- We assume that: $\hat{t} \approx \underset{t_1^n}{argmax} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$
- Acquire frequencies from a training corpus:
  - Word Frequency with given POS
    - suppose ***book*** occurs 14 times in a corpus: 10 times (.001) as **NN (**there are 10000 instances of **NN** in the corpus); 3 times (.003) as **VBP** (the corpus has 1000 **VBP**s**)**, and 1 instance of book (.005) as **VB (**the corpus has 500 **VB**s).
  - Given the previous tag, how often does each tag occur
    - suppose **DT is** followed by **NN** 80,000 times (.53), **JJ** 30,000 times (.2), **NNS** 20,000 times (.13), **VBN** 3,000 (.02) times, … out of a total of 150,000 occurrences of **DT**
- All possible tags for sequence:
  - *The*/**DT** *book*/**{NN|VB|VBP}** *is*/**VBZ** *on*/**IN** *the*/**DT** *table*/**{NN|VB|VBP}**
- Hypothetical probabilities for highest scoring tag sequence:
  - *The*/**DT** *book*/**NN** *is*/**VBZ** *on*/**IN** *the*/**DT** *table*/**NN**
  - *The*/**DT=.4,** *book*/**NN=.001,** *is*/**VBZ=.02,** *on*/**IN=.1,** *the*/**DT=.4**, *table*/**NN=.0005,**
  - **B DT = .61, DT NN = .53, NN VBZ =.44, VBZ IN = .12, IN DT = .05, DT NN = .53 NN E .31**
  - $\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) = (.4\times.61)(\mathbf{.001\times.53})(.02\times.44)(.1\times.12)(.4\times.05)(.005\times.53)(1\times.31) \approx 2.4\times10^{-13}$

Defining an HMM

- A Weighted Finite-state Automaton (WFSA)
  - Each transition arc is associated with a probability
  - The sum of all arcs outgoing from a single node is 1
- Markov chain is a WFSA in which an input string uniquely determine path through the Automaton
- Hidden Markov Model (HMM) is a slightly different case because some information (previous POS tags) is unknown (or hidden)
- HMM consists of the following:
  - **Q** = set of states: $\mathbf{q_0}$ (start state), …, $\mathbf{q_F}$ (final state)
  - **A** = transition probability matrix of **n X n** probabilities of transitioning between any pair of **n** states (**n** = **F**+1). Called: ***prior probability*** or ***transition probability*** of a tag sequence
  - **O** = sequence of **T** observations (**words**) from a vocabulary **V**
  - **B** = sequence of observation likelihoods (probability of observation generated at state) – Called ***likelihood*** (of word sequence given tag sequence)**,** aka ***emission probability***

Example HMM



START
Q0
DT
Q1
JJ
Q2
NN
Q3
IN
Q4
VBZ
Q5
END
QF
.20
.61
.13
.60
.34
.47
.53
.06
.41
.10
.12
.06
.22
1.0
.25
.44
.15
.31
the: .4
an: .05
a: .3
these: .07
...
of: .2
in: .11
on: .1
before: .001
...
angry: .0005
blue: .0011
perfect: .003
orange: .0015
...
is: .02
sees: .0012
hates: .002
sells: .004
...
book: .001
table: .0005
fish: .0002
orange: .00001
...

Go to Ralph's Viterbi Demo for ***Fish Sleep***

Viterbi Algorithm for HMM

Observed_Words = $\mathbf{w_1 \ldots w_T}$

- States = $\mathbf{q_o, q_1 \ldots q_{N,} q_F}$

A = **N** × **N** matrix such that $\mathbf{a_{i,j}}$ is the probability of the transition from $\mathbf{q_i}$ to $\mathbf{q_j}$

B = lookup table such that $\mathbf{b_i(w_t)}$ is the probability that POS **i** is realized as word **t**

viterbi = (**N**+2) × **T** matrix # columns are states, rows are words

backpointer = (**N**+2) × **T** matrix # highest scoring previous cells for viterbi

for states **q** from 1 to **N**: ## **BEGINNING**

initialize **viterbi[q,1]** to $\mathbf{a_{0,q}} * \mathbf{b_q(w_1)}$ # score transition 0→q given w_1

initialize **backpointer[q,1]** to 0 (start state)

for word **w** from 2 to **T**: ## **Middle**

for state **q** from 1 to **N**: # for T-1 × N (w,q) pairs

viterbi[q,w] ← $\max_{q'=1}^{N} viterbi[q',t-1] * a_{q',q} * b_q(w_t)$ # score = maximum previous * prior * likelihood

- **backpointer[q,w]** ← $\max_{q=1}^{N} viterbi[q,T] * a_{q,qF}$ # backpointer = maximum previous

viterbi[qF,T] ← # **END** score = maximum previous * prior * likelihood

backpointer[qF,T] ← # backpointer = maximum previous

- return(best_path) # derive by following backpointers from (qF,T) to q_0

Walk Through: The orange is on the table. (ignoring period)

$1 * .4 * .61 * .00001 * .53 * .02 * .33 * .1 * .12 * .4 * .6 * .54 * .0005 * .33 * 1 = 2.19 * 10^{-15}$

	0	**1:** ***The***	**2:orange**	**3:** ***is***	**4:** ***on***	**5:** ***the***	**6:** ***table***	**7**
Start	1							
DT		.4 * .61				.4*.6		
IN					.1*.12			
JJ			.0015 * .47					
VBZ				.02*.44				
NN			.00001 * .53				.53*.0005	
End								.33 * 1

Comments on Viterbi Trace

- The transitions from B to ***the*** and the transition from period (.) to E are necessary parts of the process.
  - Scores for first column: transitions from 0 to each possible state given: ***the***
    - The probability of reaching Q1 matching the first item on the tape (***the***) will be .4 X .61 = .244 (this is also the only possibility)
  - The transition from period (.) to E will have a high probability.
  - Likelihood that period (.) has POS **.** is very high (possibly 100%).
- The word ***orange*** is more likely to be **JJ** then **NN**
  - But the path in the chart **B DT JJ** leads to a dead end

Unknown (OOV) Words

- Possibility 1
  - Assume all POS tags have the same probability (e.g., 1/1000)
  - In effect, only use transitions to predict the correct tag
- Possibility 2
  - Use morphology (prefixes, suffixes), orthography (uppercase/lowercase), hyphenation
- Possibility 3:
  - Words occurring once in corpus = instances of UNKNOWN_WORD
  - Distribution of UNKNOWN_WORD used for OOV words
- Possibility 4: Some combination
  - Example: divide UNKNOWN_WORD into morphological classes like UNKNOWN_WORD_ENDING_IN_S

Homework

- http://cs.nyu.edu/courses/spring23/CSCI-UA.0480-057/homework3.html
- **Guidance on Program – Next few slides**

Implement Simple version of training stage first

- Data 2 fields (separated by tab): word and POS
  - Start of file = begin of sentence
  - Blank line = begin and end of sentence
  - End of file = end of sentence
- Make 2 hash tables of hash tables (e.g., Python dictionaries of dictionaries)
  1. POS → table of frequencies of words that occur with that POS
     - Example: likelihood['DT'] → {'the':1500,'a':200,'an':100, …}
     - Hash table of POSs with each value a hash table from words to frequencies
  2. STATE → table of frequencies of following states
     - Example: Transition['Begin_Sent'] → {'DT':1000,'NNP':500,'VB':200, …}
     - Example: Transition['DT'] → {'NN':500,'NNP:'200,'VB':30,,…}
     - Hash table of states with each a value a hash table from states to frequencies
     - States = Begin_Sent, End_Sent and all POSs
     - Alternative to dictionary of dictionaries: 2 dimensional array:
       - Each dimension should be 2 + number of different POS tags
- Go through the data one line at a time
  - Record frequencies for both 1 and 2
  - Loop thru hash table and convert frequencies into probabilities
    - freq/total = probability

Simple Version of Transducer

- Make a 2 dimensional array (or equivalent)
  - columns represent tokens at positions in the text
    - 0 = start of sentence
    - N = Nth token (word punctuation at position N)
    - Length+1 = end of sentence
  - rows represent S states: the start symbol, the end symbol and all possible POS (NN, JJ, ...)
  - cells represent the likelihood that a particular word is at a particular state
- Traverse the chart as per the algorithm (fish sleep slides, etc.)
  - For all states at position 1, multiply transition probability from Start (position 0) by likelihood that word at position 1 occurs in that state. Choose highest score for each cell.
  - For n from 2 to N (columns)
    - for each cell [n,s] in column n and each state [n-1,s'] in column n-1:
    - get the product of:
      - likelihood that token n occurs in state s
      - the transition probability from s' to s
      - the score stored in [n-1,s']
    - At each position [n,s], record the max of the s scores calculated

Calculating Probabilities

- The probability of each transition to state ***N*** for token ***T*** is assumed to be the product of 3 factors
  - Probability that state ***N*** occurs with token ***T***
    - There is 100% chance that the start state will be at the beginning of the sentence
    - There is 100% chance that the end state will be at the end of the sentence
    - If a token was observed in the training corpus, look up probability from table
    - For Out of Vocabulary words, there are several strategies
      - Simple strategy (for first implementation): 1/1000 or 100% divided by number of states or any fraction that is the same for all POS
      - Other strategies are a separate discussion
  - Probability that state N occurs given previous state
    - Look up in table, calculate for every possible previous state
  - Highest Probability of previous state (calculate for each previous state)
- For each new state, choose the highest score (this is the bigram model)
- Choose the POS tag sequence resulting in the highest score in the end state

OOV Strategies from slide 21

- Default (use until other parts of program are debugged)
  - Assume all POS tags have the same probability (e.g., 1/1000)
  - In effect, only use transitions to predict the correct tag
- Morphology
  - Use prefixes, suffixes, uppercase/lowercase, hyphenation, to predict POS classes of OOV words
  - Assign "made up" values based on these features?
  - Perhaps hard-code unusual punctuation based on Penn Treebank specs
- Compute probability of UNKNOWN_WORD
  - Treat words occurring once in training collectively as UNKNOWN_WORD
    - don't record them separately (recalculate likelihood table)
  - UNKNOWN_WORD probability used for OOV words by transducer
- Combination:
  - UNKNOWN_ending_in_s, UNKNOWN_ending_in_ed, UNKNOWN_with_capital_letter, ...

How you Might Improve your Score

- Do error analysis on development corpus – base changes on your findings
  - Example: Are there errors for punctuation (which should be almost unambiguous)?
- Implement a trigram algorithm
  - See Jurafsky and Martin (p. 149)
  - 4-gram is a waste of time for this size corpus
- Note: A clever OOV system contributes more to score than trigam
- Manual rule system using constraints, e.g., slide 7.
  - For words with frequency>1, assume the disjunction of observed labels is possible
  - Rule out possibilities according to constraints
  - Run this and compare results with HMM system
  - Figure out way of combining results with HMM based on error analysis
    - Voting, weighted combinations, etc.

Grading

- Your grade 1–11 is based on:
  – Meeting the format constraints (1 point)
  – Your accuracy score
    - 95% → 10 points, 93% → 9 points
    - 92% → 8 points, 91% → 7 points
    - 90% → 6 points, 85% → 5 points
- You should include a short write-up of what you did, so it is easier to evaluate.