LING 520: Computational Analysis of English

Semester: FALL '16

Instructor: Sowmya Vajjala

Iowa State University, USA

4 October 2016

Class Outline

- Quick recap and conclusion of HMM Tagging
- Assignment 3: Discussion about Problem 1
- Transformation Based Learning
- Evaluating POS taggers
- Practice with POS taggers in NLTK

HMM tagging

HMM Tagging

- Aim: Maximize the probability of a tag sequence, given a word sequence.
  $\hat{t}_1^n = \text{argmax}_{t_1^n}(\text{P}(\text{w}_1^n|\text{t}_1^n)*\text{P}(\text{t}_1^n))$
- First assumption: probability of a word depends only on its POS tag, not on previous words or tags.
  $\Rightarrow \text{P}(\text{w}_1^n|\text{t}_1^n) \approx \pi_{i=1}^n \text{P}(\text{w}_i|\text{t}_i)$
- Second assumption: probability of a tag appearing is only dependent on previous tag instead of entire history (for Bigram taggers!)
  $\Rightarrow \text{P}(\text{t}_1^n) \approx \pi_{i=1}^n \text{P}(\text{t}_i|\text{t}_{i-1})$
- Considering these assumptions, $\hat{t}_1^n$ becomes:
  $\text{argmax}_{t_1^n} \pi_{i=1}^n(\text{P}(\text{w}_i|\text{t}_i)*\text{P}(\text{t}_i|\text{t}_{i-1}))$

What does this mean in normal language?

- This is the equation we have:
  $\hat{t}_1^n = \text{argmax}_{t_1^n} \pi_{i=1}^n (\text{P}(\text{w}_i|\text{t}_i)*\text{P}(\text{t}_i|\text{t}_{i-1}))$
- $(\text{P}(\text{w}_i|\text{t}_i) = \text{Count}(\text{t}_i, w_i)/\text{Count}(\text{t}_i)$
  $\Rightarrow \text{P}(\text{good}|\text{ADJ}) = \text{C}(\text{ADJ,good})/\text{C}(\text{ADJ})$
- $\text{P}(\text{t}_i|\text{t}_{i-1}) = \text{Count}(\text{t}_{i-1}, t_i)/\text{Count}(\text{t}_{i-1})$ (Same as in language models)
  $\Rightarrow \text{P}(\text{NN}|\text{DT}) = \text{C}(\text{DT,NN})/\text{C}(\text{DT})$
- This is HMM tagging. The "Hidden" here refers to the tag sequence, because that is something that we cannot observe in the input which is only a sequence of words.
- If we use HMM for speech recognition, "observed" state is speech sequence, hidden state is word sequence.

HMM - Example

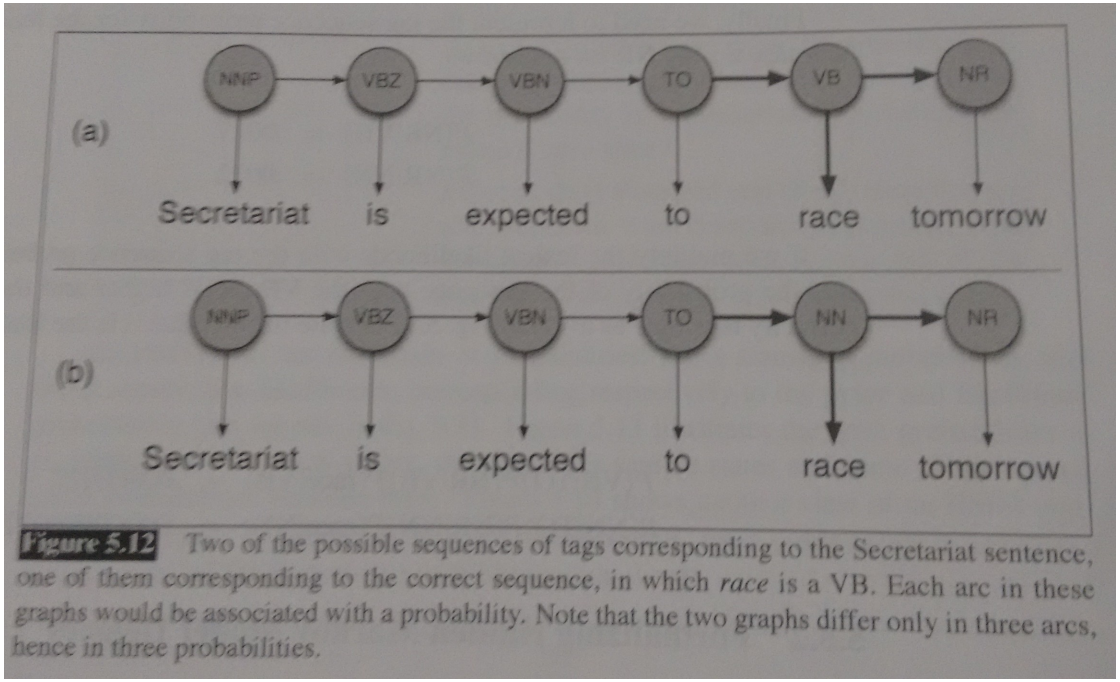


Figure 5.12 Two of the possible sequences of tags corresponding to the Secretariat sentence, one of them corresponding to the correct sequence, in which *race* is a VB. Each arc in these graphs would be associated with a probability. Note that the two graphs differ only in three arcs, hence in three probabilities.

- Only difference between the two tag sequence probabilities:
  1. In sentence a), we estimate P(VB|TO)*P(NR|VB)*P(race|VB)
  2. In sentence b), we estimate P(NN|TO)*P(NR|NN)*P(race|NN) instead.

HMM - What we need while training a model

- Tag transition probabilities (e.g., P(NN|ADJ), P(VB|NN) etc. for a bigram model)
- Observation likelihoods (e.g., P(race|VB), P(I|PRP) etc. for a bigram model)

- Once you build information about all these probabilities, you will just use these as a basis to tag a new sequence.

Viterbi Algorithm for HMM

- The task of determining which sequence of output is the most probable for input sequence is called decoding.
- Viterbi algorithm is the most commonly used decoding algorithm for HMMs.
- This is also a form of dynamic programming (breaking up the problem into smaller problems and solving them one by one)
- Viterbi algorithm provides an optimal solution of using all those tag-transition probabilities and observation likelihoods and finds the tag sequence which satisfies the maximum likelihood requirement to observation sequence.

Note: Implementation details are beyond the scope of this course. Please read Chapter 5 and 6 in J&M if you are interested to know more (and can handle more math)

Assignment 3, Q1 - Overview and Hints

- As mentioned on thursday, I want you to implement a trigram tagger of the kind: P(Tag2|Tag1,Word2). Not a HMM. If you want, you can implement a HMM instead.
- What you need for this non-HMM trigram tagger: First, using the training data, you should create a file that can store this kind of counts: C(Tag1,Word2,Tag2)
- The second program should be able to read these counts, and then, look at each word in the text sentence, grab the necessary counts, pick the trigram with maximum counts.
- Questions: How do we estimate probability for the first POS tag?? What do we do when we do not see a Trigram we want??

Assignment 3, Q1 - Overview and Hints

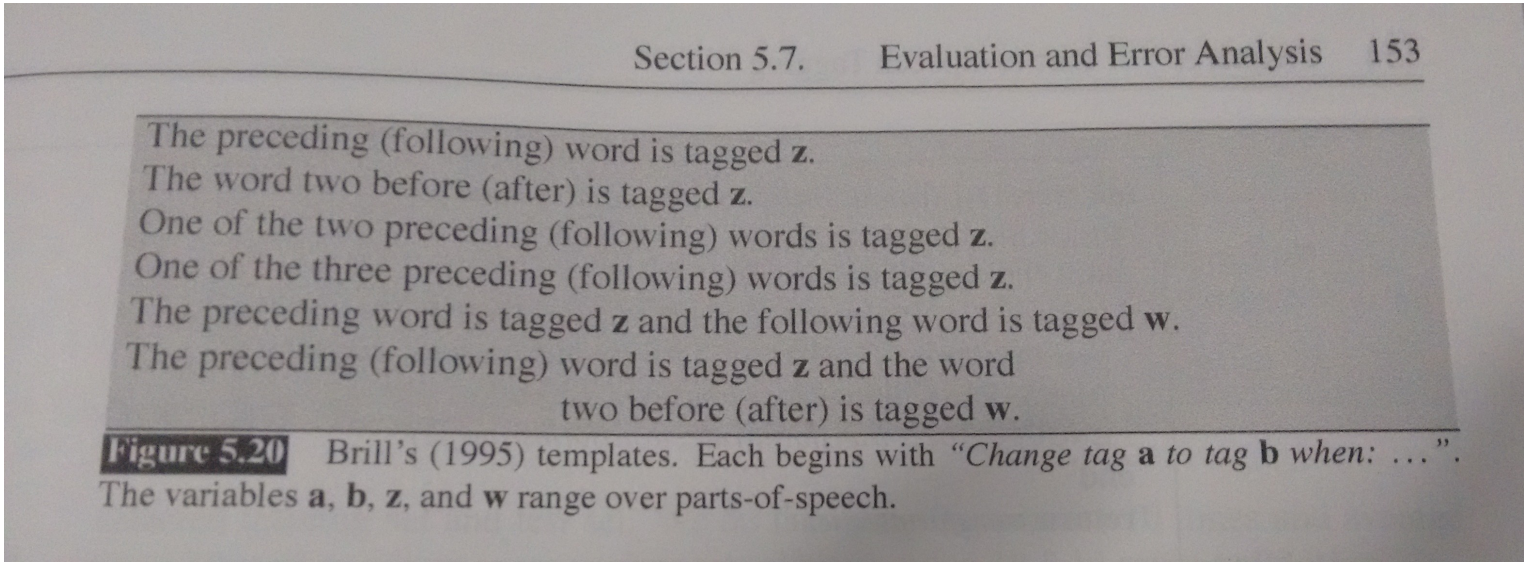
- As mentioned on thursday, I want you to implement a trigram tagger of the kind: P(Tag2|Tag1,Word2). Not a HMM. If you want, you can implement a HMM instead.
- What you need for this non-HMM trigram tagger: First, using the training data, you should create a file that can store this kind of counts: C(Tag1,Word2,Tag2)
- The second program should be able to read these counts, and then, look at each word in the text sentence, grab the necessary counts, pick the trigram with maximum counts.
- Questions: How do we estimate probability for the first POS tag?? What do we do when we do not see a Trigram we want??
- A small example of how it can look (without Erk's code, without NLTK.).

Transformation based tagging

Tranformation Based Learning

- TBL is a machine learning approach which relies both on the creation of rules, and inducing such rules automatically from a large annotated corpus.
  (Annotation here refers to POS tags.)
- TBL occurs in three steps in the training phase:
  1. Stage 1: Label every word with its most likely tag.
  2. Stage 2: Automatically induce a set of rules based on pre-defined templates, to improve tagging from Step 1. (More on this in the next slide!)
  3. Stage 3: Use these rules from Step 2 to re-tag sentences from Step 1. Repeat the rule induction and retagging process until there is no significant improvement in Tagger accuracy between two repeats.

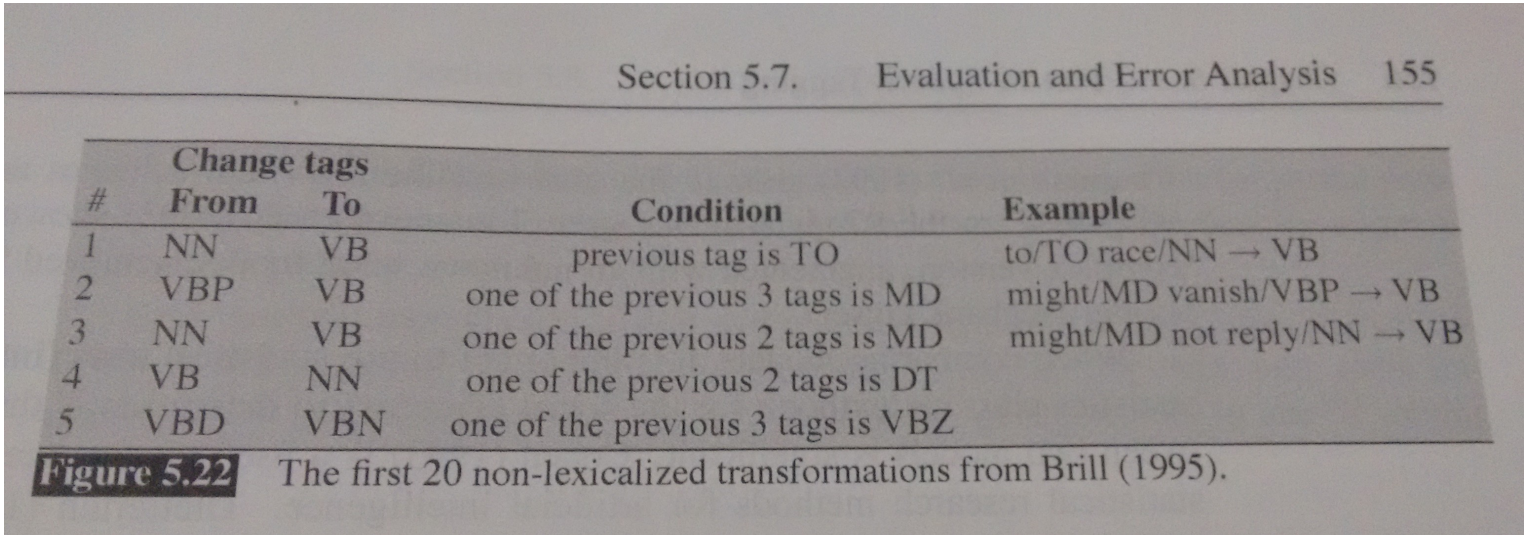
Brill (1995)'s templates



The preceding (following) word is tagged **z**.
The word two before (after) is tagged **z**.
One of the two preceding (following) words is tagged **z**.
One of the three preceding (following) words is tagged **z**.
The preceding word is tagged **z** and the following word is tagged **w**.
The preceding (following) word is tagged **z** and the word two before (after) is tagged **w**.

Figure 5.20 Brill's (1995) templates. Each begins with *"Change tag* **a** *to tag* **b** *when: ..."*. The variables **a**, **b**, **z**, and **w** range over parts-of-speech.

Brill Tagger - Learned Rules



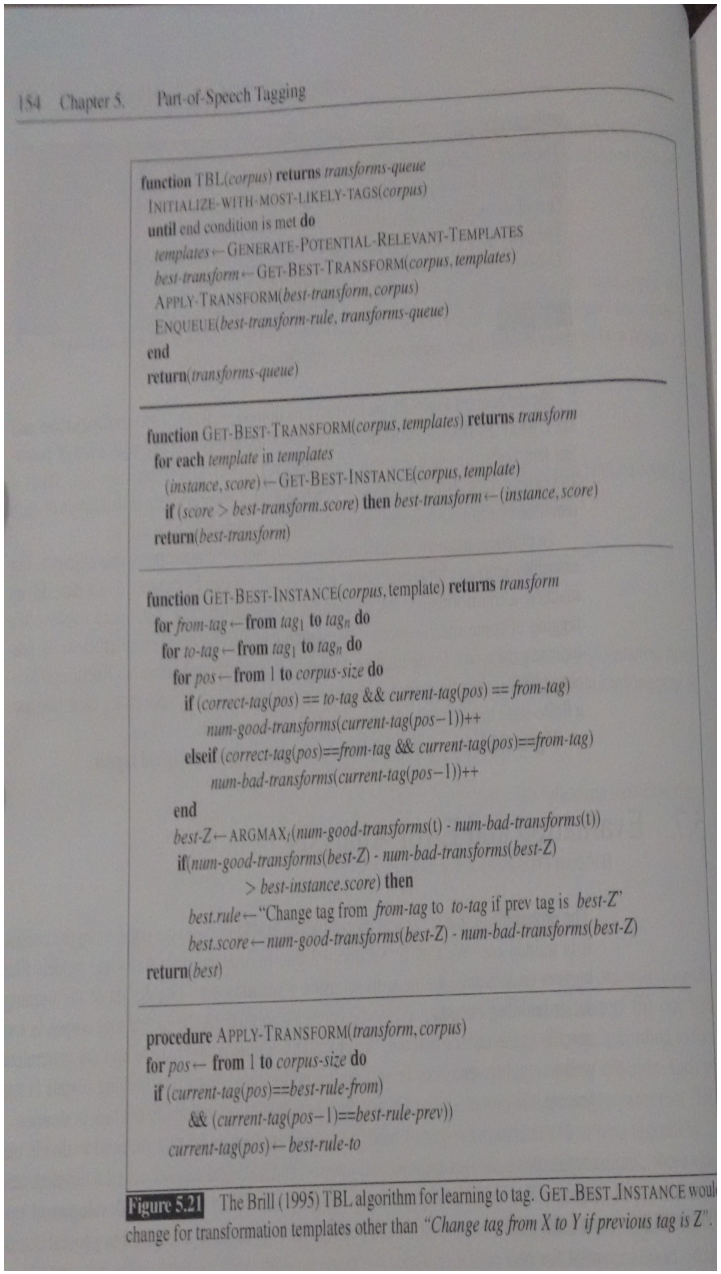
Section 5.7. Evaluation and Error Analysis 155

#	Change tags From	To	Condition	Example
1	NN	VB	previous tag is TO	to/TO race/NN → VB
2	VBP	VB	one of the previous 3 tags is MD	might/MD vanish/VBP → VB
3	NN	VB	one of the previous 2 tags is MD	might/MD not reply/NN → VB
4	VB	NN	one of the previous 2 tags is DT	
5	VBD	VBN	one of the previous 3 tags is VBZ	

Figure 5.22 The first 20 non-lexicalized transformations from Brill (1995).

Note: In pure rule-based tagging, these rules will be created by linguists, based on their knowledge of language

Brill's algorithm



154 Chapter 5. Part-of-Speech Tagging

```
function TBL(corpus) returns transforms-queue
 INITIALIZE-WITH-MOST-LIKELY-TAGS(corpus)
 until end condition is met do
  templates ← GENERATE-POTENTIAL-RELEVANT-TEMPLATES
  best-transform ← GET-BEST-TRANSFORM(corpus, templates)
  APPLY-TRANSFORM(best-transform, corpus)
  ENQUEUE(best-transform-rule, transforms-queue)
 end
 return(transforms-queue)
```

```
function GET-BEST-TRANSFORM(corpus, templates) returns transform
 for each template in templates
  (instance, score) ← GET-BEST-INSTANCE(corpus, template)
  if (score > best-transform.score) then best-transform ← (instance, score)
 return(best-transform)
```

```
function GET-BEST-INSTANCE(corpus, template) returns transform
 for from-tag ← from tag_1 to tag_n do
  for to-tag ← from tag_1 to tag_n do
   for pos ← from 1 to corpus-size do
    if (correct-tag(pos) == to-tag && current-tag(pos) == from-tag)
        num-good-transforms(current-tag(pos−1))++
    elseif (correct-tag(pos)==from-tag && current-tag(pos)==from-tag)
        num-bad-transforms(current-tag(pos−1))++
   end
   best-Z ← ARGMAX_t(num-good-transforms(t) - num-bad-transforms(t))
   if(num-good-transforms(best-Z) - num-bad-transforms(best-Z)
          > best-instance.score) then
    best.rule ← "Change tag from from-tag to to-tag if prev tag is best-Z"
    best.score ← num-good-transforms(best-Z) - num-bad-transforms(best-Z)
 return(best)
```

```
procedure APPLY-TRANSFORM(transform, corpus)
 for pos ← from 1 to corpus-size do
  if (current-tag(pos)==best-rule-from)
      && (current-tag(pos−1)==best-rule-prev))
   current-tag(pos) ← best-rule-to
```

Figure 5.21 The Brill (1995) TBL algorithm for learning to tag. GET_BEST_INSTANCE would change for transformation templates other than *"Change tag from X to Y if previous tag is Z"*.

To know more on TBL based tagging

- Brill's 1995 paper
  http://www.aclweb.org/anthology/J95-4004
- Brill Tagger implementation in NLTK:
  1. http://www.nltk.org/_modules/nltk/tag/brill.html
  2. http://www.nltk.org/_modules/nltk/tag/brill_trainer.html

Evaluating POS taggers

Evaluating POS taggers -1

Intrinsic Evaluation

- Tagging accuracy is the measure to evaluate the performance of a POS tagger.
- Usually, there is a gold-standard test set which is used as a bench mark to compare all POS taggers that are developed for a given language.
- If your tagger identifies 90% of tags in this test set correctly, your tagger's accuracy is 90%.
- English POS taggers currently report up to 97-98% accuracy with that gold standard test set.
- Is 97-98% good?

Evaluating POS taggers -1

Intrinsic Evaluation

- Tagging accuracy is the measure to evaluate the performance of a POS tagger.
- Usually, there is a gold-standard test set which is used as a bench mark to compare all POS taggers that are developed for a given language.
- If your tagger identifies 90% of tags in this test set correctly, your tagger's accuracy is 90%.
- English POS taggers currently report up to 97-98% accuracy with that gold standard test set.
- Is 97-98% good?
- What is good depends on what is the baseline and what is the ceiling.

Evaluating POS taggers - 2

- Baseline: Choose the most likely Unigram tag for a given word.
- Ceiling: How well do humans do on this task? How much is the agreement between two humans doing manual POS tagging?
- Note: This ceiling is under the assumption that computers cannot beat humans. But there are some tasks where computers are better than humans.
- So, this ceiling is not a universal ceiling for all NLP tasks.

Evaluating POS taggers - 3

- Let us say my baseline is 80%. My POS tagger gives an accuracy fo 85%. My ceiling is 87%. Is the tagger doing well?

Evaluating POS taggers - 3

- Let us say my baseline is 80%. My POS tagger gives an accuracy fo 85%. My ceiling is 87%. Is the tagger doing well?
- Let us say my baseline is 75%. My POS tagger gives an accuracy of 85%. My ceiling is 95%. Is the tagger doing well?

Evaluating POS taggers - 3

- Let us say my baseline is 80%. My POS tagger gives an accuracy fo 85%. My ceiling is 87%. Is the tagger doing well?
- Let us say my baseline is 75%. My POS tagger gives an accuracy of 85%. My ceiling is 95%. Is the tagger doing well?
- Let us say my baseline is 80%. My POS tagger gives an accuracy of 81%. My ceiling is 100%. Is the tagger doing well?

Evaluating POS taggers - 3

- Let us say my baseline is 80%. My POS tagger gives an accuracy fo 85%. My ceiling is 87%. Is the tagger doing well?
- Let us say my baseline is 75%. My POS tagger gives an accuracy of 85%. My ceiling is 95%. Is the tagger doing well?
- Let us say my baseline is 80%. My POS tagger gives an accuracy of 81%. My ceiling is 100%. Is the tagger doing well?
- Let us say my baseline is 30%. My POS tagger gives an accuracy of 81%. My ceiling is 100%. Is the tagger doing well?

Evaluating POS taggers - 4

Extrinsic Evaluation

- Important thing to note: Having a good test-set accuracy does not mean the best tagger is always the best on any random sentence in English language. This is just a benchmark.
- It is like the grades you get in school. Just one measure of evaluation. Does not cover all aspects.
- So, although it can be used to compare different taggers, the ultimate evaluation is the extrinsic evaluation - how it works in the real setting it is meant to be used.
- If you are not developing a POS tagger, you should bother about evaluating a tagger for the application you intend to use it ..
- and not about how a tagger does on a standard test set which is not from the genre/domain etc that you want to use it in.

POS Tagging - Resources

- Chapter 5 in Jurafsky and Martin provides a good theoretical foundation. Chapter 6 has more advanced stuff about statistical modeling.
- Chapter 5 in NLTK Book provides a lot of practical information on using different POS taggers, and the training process.
- Radev's Week 7 lectures in Coursera course provide a resource you can play and replay as many times you want (unlike a regular classroom lecture)
- If you are able to finish Assignment 3, you are all set to analyse what tagger works for your purpose (if you want!), what you need if you want a customized tagger (for learner text) etc.

Practice Exercises

- Chapter 5 in NLTK book: go through the examples. Try out the various POS taggers in NLTK, and try to do some of the problems at the end.
- Go through problems in Problem Set 4 - See how many of them sound straight forward and start doing one of them.
- Explore how Regular Expressions can be useful for POS tagging.
- These slides on NLTK POS taggers is useful: https://goo.gl/O3dNl3

Next Class

- Revision of concepts so far
- Planning for a tutorial session
- Discussion about ideas for final projects