Introduction to Computational Linguistics: Smoothing

Sharon Goldwater
(with some slides from Philipp Koehn)

13 July 2015

School of informatics

Recap: ngrams and smoothing

- One way to estimate $P(\vec{w})$: assume words only depend on fixed context
  - n-gram model: sentence prob is product of each word's prob conditioned on $n-1$ previous words.
  - i.e., $P(\vec{w}) = \prod P(w_i|w_{i-n+1} \dots w_{i-1})$.
- But even for smallish n, we might not see all possible n-grams in our corpus, or see very few occurrences.
- *Smoothing* methods try to better estimate probabilities for infrequent events: move probability mass from seen to unseen.

Recap: smoothing methods

- Two simple smoothing methods:
  - Add-one: simplest choice, but steals way too much mass from seen items.
  - Add-α: somewhat better; optimize α on held-out data.
- These may actually be good enough when not many distinct items in "vocabulary", e.g. n-gram models over
  - *characters* or *phonemes* (to model phonotactics)
  - *parts of speech* (syntactic categories): coming up
- But not good for *word* n-grams.
- This lecture: a taster of other methods for smoothing word probabilities.

First, notation: adjusted counts

- Previously, we estimated probabilities based on actual counts

$$P_{\text{ML}} = \frac{c}{n}$$

- Then, we changed the formula to estimate smoothed probabilities

$$P_{+\alpha} = \frac{c+\alpha}{n+\alpha v}$$

- Another view: we adjusted the counts c

$$P_{+\alpha} = \frac{c^*}{n} \quad \Rightarrow \quad c^* = n\, P_{+\alpha} = (c+\alpha)\,\frac{n}{n+\alpha v}$$

Good-Turing Smoothing

- Adjust actual counts c to expected counts c^* with formula

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

  - N_c number of n-grams that occur exactly c times in corpus
  - N_0 total number of unseen n-grams

Good-Turing for 2-Grams in Europarl

Count	Count of counts	Adjusted count	Test count
c	N_c	c^*	t_c
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927
20	4,546	19.54487	18.95948

t_c are average counts of n-grams in test set that occurred c times in corpus

Good-Turing justification: 0-count items

- Estimate the probability that the next observation is previously unseen (i.e., will have count 1 once we see it)

$$P(\text{unseen}) = \frac{N_1}{n}$$

This part uses MLE!

- Divide that probability equally amongst all unseen events

$$P_{\text{GT}} = \frac{1}{N_0}\frac{N_1}{n} \quad \Rightarrow \quad c^* = \frac{N_1}{N_0}$$

Good-Turing justification: 1-count items

- Estimate the probability that the next observation was seen once before (i.e., will have count 2 once we see it)

$$P(\text{once before}) = \frac{2N_2}{n}$$

- Divide that probability equally amongst all 1-count events

$$P_{\text{GT}} = \frac{1}{N_1}\frac{2N_2}{n} \quad \Rightarrow \quad c^* = \frac{2N_2}{N_1}$$

- Same thing for higher count items

Problems with Good-Turing

- Assumes we know the vocabulary size (no unseen words) [but see J&M 4.3.2]
- Doesn't allow "holes" in the counts (if $N_i > 0$, $N_{i-1} > 0$) [but see J&M 4.5.3]
- Applies discounts even to high-frequency items [but see J&M 4.5.3]
- Divides shifted probability mass evenly between all items of same frequency.

Remaining problem

- In given corpus, suppose we never observe
  - Scottish beer drinkers
  - Scottish beer eaters
- If we build a trigram model smoothed with Add-α or G-T, which example has higher probability?

Remaining problem

- Previous smoothing methods assign equal probability to all unseen events.
- Better: use information from lower order n-grams (shorter histories).
  - beer drinkers
  - beer eaters
- Two ways: **interpolation** (discussed here) and **backoff** (see J&M).

Interpolation

- Higher and lower order n-gram models have different strengths and weaknesses
  - high-order n-grams are sensitive to more context, but have sparse counts
  - low-order n-grams consider only very limited context, but have robust counts
- So, combine them: $P_I(w_3|w_1,w_2) = \lambda_1 P_1(w_3) + \lambda_2 P_2(w_3|w_2) + \lambda_3 P_3(w_3|w_1,w_2)$
  - Note that $\sum_i \lambda_i = 1$. These interpolation parameters can be optimized on held-out data.

Do our smoothing methods work here?

Example from MacKay and Bauman Peto (1994):

> Imagine, you see, that the language, you see, has, you see, a frequently occurring couplet, 'you see', you see, in which the second word of the couplet, see, follows the first word, you, with very high probability, you see. Then the marginal statistics, you see, are going to become hugely dominated, you see, by the words you and see, with equal frequency, you see.

- $P(\text{see})$ and $P(\text{you})$ are both high, but *see* nearly always follows *you*.
- So $P(\text{see}|novel)$ should be much lower than $P(\text{you}|novel)$.

Diversity of histories matters!

- A real example: the word York
  - fairly frequent word in Europarl corpus, occurs 477 times
  - as frequent as foods, indicates and providers
  - → in unigram language model: a respectable probability
- However, it almost always directly follows New (473 times)
- So, in unseen bigram contexts, York should have low probability
  - lower than predicted by unigram model used in interpolation/backoff.

Kneser-Ney Smoothing

- Kneser-Ney smoothing takes diversity of histories into account
- Count of distinct histories for a word

$$N_{1+}(\bullet w_i) = |\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|$$

- Recall: maximum likelihood estimation of unigram language model

$$P_{ML}(w) = \frac{c(w_i)}{\sum_{w_i} c(w_i)}$$

- In Kneser-Ney smoothing, replace raw counts with count of histories

$$P_{KN}(w_i) = \frac{N_{1+}(\bullet w)}{\sum_{w_i} N_{1+}(\bullet w_i)}$$

Kneser-Ney in practice

- Original version used backoff, later "modified Kneser-Ney" introduced using interpolation (Chen and Goodman, 1998).
- Fairly complex equations, but until recently the best smoothing method for word n-grams.
- See Chen and Goodman for extensive comparisons of KN and other smoothing methods.
- KN (and other methods) implemented in language modeling toolkits like SRILM (classic), KenLM (good for really big models), OpenGrm Ngram library (uses finite state transducers), etc.

Bayesian interpretations of smoothing

- We started off by asking: What is the best choice of θ given the data d that we saw?

$$P(\theta|d) \propto P(d|\theta)P(\theta)$$

- MLE ignored $P(\theta)$, and we had to introduce smoothing.
- It turns out that many smoothing methods are mathematically equivalent to forms of **Bayesian estimation**, i.e., the use of non-uniform priors!
  - Add-α smoothing: Dirichlet prior
  - Kneser-Ney smoothing: Pitman-Yor prior

See MacKay and Bauman Peto (1994); (Goldwater, 2006, pp. 13-17); Goldwater et al. (2006); Teh (2006).

Are we done with smoothing yet?

We've considered methods that predict rare/unseen words using

- Uniform probabilities (add-α, Good-Turing)
- Probabilities from lower-order n-grams (interpolation, backoff)
- Probability of appearing in new contexts (Kneser-Ney)

What's left?

Word similarity

- Two words with $C(w_1) \gg C(w_2)$
  - salmon
  - swordfish
- Can $P(\text{salmon}|\text{caught two})$ tell us something about $P(\text{swordfish}|\text{caught two})$?
- n-gram models: no.

Word similarity in language modeling

- Early version: class-based language models (J&M 4.9.2)
  - Define classes c of words, by hand or automatically
  - $P_{CL}(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$ (an HMM)
- Recent version: **distributed** language models
  - Current models have better perplexity than MKN.
  - Ongoing research to make them more efficient.
  - Examples: Recursive Neural Network LM (Mikolov et al., 2010), Log Bilinear LM (Mnih and Hinton, 2007) and extensions.

Distributed word representations

- Each word represented as high-dimensional vector (50-500 dims)

E.g., salmon is $[0.1, 2.3, 0.6, -4.7, \dots]$

- Similar words represented by similar vectors

E.g., swordfish is $[0.3, 2.2, 1.2, -3.6, \dots]$

Distributed word representations

- Each word represented as high-dimensional vector (50-500 dims)

E.g., salmon is $[0.1, 2.3, 0.6, -4.7, \dots]$

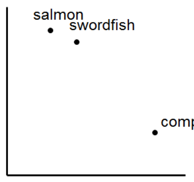
- Similar words represented by similar vectors

E.g., swordfish is $[0.3, 2.2, 1.2, -3.6, \dots]$

- Representations can be thought of as feature vectors.
  - A bit like representing phonemes as vectors of their distinctive features, or vowel sounds as vectors of formant values.
  - But unlike phoneme/acoustic feature vectors, the dimensions/values are determined automatically, may not be easily interpretable.

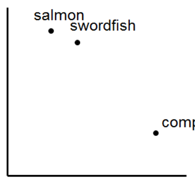
Learning the representations

- Goal: learn word representations (**embeddings**) such that words that behave similarly end up near each other in high-dimensional space.
- 2-dimensional example:



Learning the representations

- Goal: learn word representations (**embeddings**) such that words that behave similarly end up near each other in high-dimensional space.
- 2-dimensional example:



- Machine learning methods (e.g. neural networks) are used to learn embeddings.
- Embeddings seem to encode both semantic and syntactic similarity (using different dimensions) (Mikolov et al., 2013).

Other Topics in Language Modeling

Many active research areas, some more linguistically interesting than others.

- Modeling issues:
  - Morpheme-based language models
  - Syntactic language models
  - Domain adaptation: when only a small domain-specific corpus is available
- Implementation issues:
  - Speed: both to train, and to use in real-time applications like translation and speech recognition.
  - Disk space and memory: espcially important for mobile devices

Summary

- We can estimate sentence probabilities by breaking down the problem, e.g. by instead estimating n-gram probabilities.
- Longer n-grams capture more linguistic information, but are sparser.
- Different smoothing methods capture different intuitions about how to estimate probabilities for rare/unseen events.
- still lots of work on how to improve these models.

References

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Center for Research in Computing Technology, Harvard University.

Goldwater, S. (2006). *Nonparametric Bayesian Models of Lexical Acquisition*. PhD thesis, Brown University.

Goldwater, S., Griffiths, T. L., and Johnson, M. (2006). Interpolating between types and tokens by estimating power-law generators. In *Advances in Neural Information Processing Systems 18*, pages 459–466, Cambridge, MA. MIT Press.

MacKay, D. and Bauman Peto, L. (1994). A hierarchical Dirichlet language model. *Natural Language Engineering*, 1(1).

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751.

Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM.

Teh, Y. W. (2006). A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 985–992, Syndney, Australia.

Introduction to Computational Linguistics: Part-of-speech tagging

Sharon Goldwater
(based on slides from Philipp Koehn)

13 July 2015

School of informatics

What is part of speech tagging?

Given a string, identify parts of speech (syntactic categories):

This is a simple sentence
⇓
This/DET is/VB a/DET simple/ADJ sentence/NOUN

- First step towards syntactic analysis
- Taggers can help linguists analyze phenomena of interest
- Hidden Markov models for tagging illustrate important mathematical/computationa concepts

Other tagging tasks in NLP

A number of problems can be framed as tagging (sequence labelling) problems:

- **Named entity recognition:** it may also be useful to find names of persons, organizations, etc. in the text, e.g. Barack Obama
- **Information field segmentation:** Given specific type of text (classified advert, bibiography entry), identify which words belong to which "fields" (price/size/#bedrooms, author/title/year)
- **Prosodic marking:** In speech synthesis, decide which words/syllables have stress or intonational changes, e.g. You're going. vs You're going?

Parts of Speech

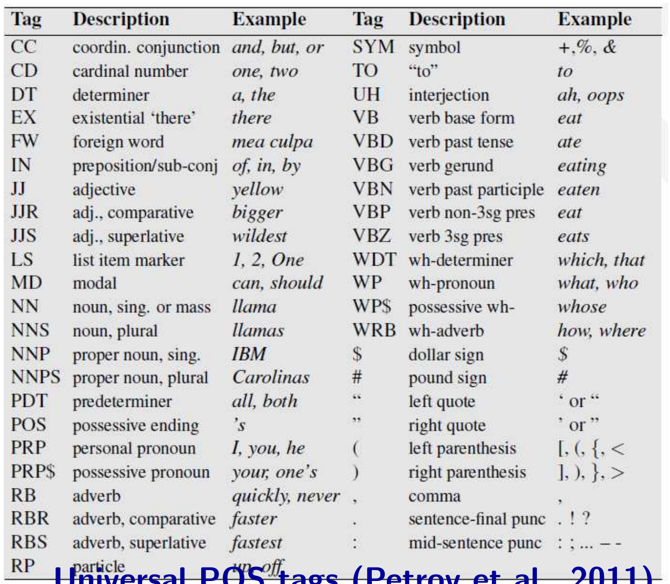
- **Open class words** (or content words)
  - nouns, verbs, adjectives, adverbs
  - mostly content-bearing: they refer to objects, actions, and features in the world
  - *open* class, since there is no limit to what these words are, new ones are added all the time (email, website).
- **Closed class words**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

How many parts of speech?

- Both linguistic and practical considerations
- Corpus annotators decide. Distinguish between
  - proper nouns (names) and common nouns?
  - singular and plural nouns?
  - past and present tense verbs?
  - auxiliary and main verbs?
  - etc

English POS tag sets

- Usually 40-100 tags
- Brown corpus (87 tags)
  - One of the earliest large corpora collected for computational linguistics (1960s)
  - A **balanced** corpus: different genres (fiction, news, academic, editorial, etc)
- Penn Treebank corpus (45 tags)
  - First large corpus **annotated** (tagged by hand) with POS and full syntactic trees (1992)
  - Possibly the most-used corpus in NLP
  - Contains only text from the Wall Street Journal (WSJ)



Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	$
NNPS	proper noun, plural	*Carolinas*	#	pound sign	#
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

J&M Fig 5.6: Penn Treebank POS tags

POS tags in other languages

- Morphologically rich languages often have compound morphosyntactic tags

  `Noun+A3sg+P2sg+Nom` (J&M, p.196)

- Hundreds or thousands of possible combinations
- Predicting these requires more complex methods than what we will discuss

Universal POS tags (Petrov et al., 2011)

- A move in the other direction
- Simplify the set of tags to lowest common denominator across languages
- Map existing annotations onto universal tags

  {VB, VBD, VBG, VBN, VBP, VBZ, MD} ⇒ VERB

- Allows interoperability of systems across languages
- Promoted by Google and others

Universal POS tags (Petrov et al., 2011)

NOUN (nouns)
VERB (verbs)
ADJ (adjectives)
ADV (adverbs)
PRON (pronouns)
DET (determiners and articles)
ADP (prepositions and postpositions)
NUM (numerals)
CONJ (conjunctions)
PRT (particles)
'.' (punctuation marks)
X (a catch-all for other categories such as abbreviations or foreign words)

Why is automatic POS tagging hard?

The usual reasons!

- Ambiguity:

glass of water/NOUN	vs.	water/VERB the plants	
lie/VERB down	vs.	tell a lie/NOUN	
wind/VERB down	vs.	a mighty wind/NOUN	(homographs)

  How about time flies like an arrow?

- Sparse data:
  - Words we haven't seen before (at all, or in this context)
  - Word-Tag pairs we haven't seen before

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

A probabilistic model for tagging

Let's define a new generative process for sentences.

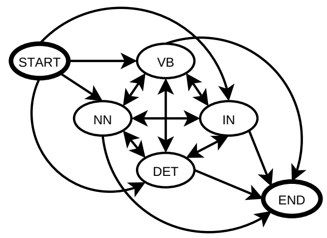
- To generate sentence of length n:

```
Let $t_0 =$<s>
For $i = 1$ to $n$
    Choose a tag conditioned on previous tag: $P(t_i|t_{i-1})$
    Choose a word conditioned on its tag: $P(w_i|t_i)$
```

- So, model assumes:
  - Each tag depends only on previous tag: a bigram model over tags.
  - Words are independent given tags

Probabilistic finite-state machine

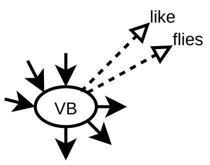
- One way to view the model: sentences are generated by walking through **states** in a graph. Each state represents a tag.



- Prob of moving from state s to s' (**transition probability**): $P(t_i = s'|t_{i-1} = s)$

Probabilistic finite-state machine

- When passing through a state, emit a word.



- Prob of emitting w from state s (**emission probability**): $P(w_i = w|t_i = s)$

What can we do with this model?

- Simplest thing: if we know the parameters (tag transition and word emission probabilities), can compute the probability of a tagged sentence.

- Let $S = w_1 \ldots w_n$ be the sentence and $T = t_1 \ldots t_n$ be the corresponding tag sequence. Then

$$p(S,T) = \prod_{i=1}^{n} P(t_i|t_{i-1})P(w_i|t_i)$$

Example: computing joint probability $P(S,T)$

What's the probability of this tagged sentence?

This/DET is/VB a/DET simple/JJ sentence/NN

Example: computing joint probability $P(S,T)$

What's the probability of this tagged sentence?

This/DET is/VB a/DET simple/JJ sentence/NN

- First, add begin- and end-of-sentence markers <s> and </s>. Then:

$$\begin{aligned} p(S,T) &= \prod_{i=1}^{n} P(t_i|t_{i-1})P(w_i|t_i) \\ &= P(\text{DET}|\text{<s>})P(\text{VB}|\text{DET})P(\text{DET}|\text{VB})P(\text{JJ}|\text{DET})P(\text{NN}|\text{JJ})P(\text{</s>}| \\ &\quad \cdot P(\text{This}|\text{DET})P(\text{is}|\text{VB})P(\text{a}|\text{DET})P(\text{simple}|\text{JJ})P(\text{sentence}|\text{NN}) \end{aligned}$$

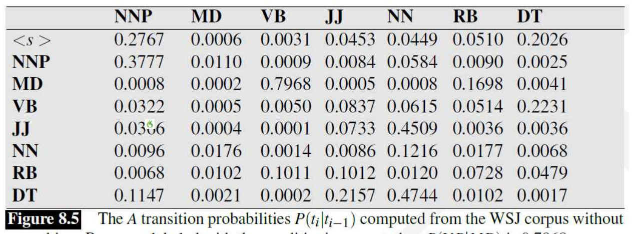
- OK. But now we need to plug in probabilities... from where?

Training the model

- Given a corpus annotated with tags (e.g., Penn Treebank), we *estimate* $P(w_i|t_i)$ and $P(t_i|t_{i-1})$ using familiar methods (MLE/smoothing)

Training the model

- Given a corpus annotated with tags (e.g., Penn Treebank), we *estimate* $P(w_i|t_i)$ and $P(t_i|t_{i-1})$ using familiar methods (MLE/smoothing)



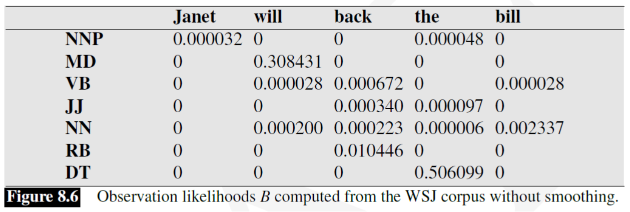
	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 8.5 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

(Fig from J&M draft 3rd edition)

Training the model

- Given a corpus annotated with tags (e.g., Penn Treebank), we *estimate* $P(w_i|t_i)$ and $P(t_i|t_{i-1})$ using familiar methods (MLE/smoothing)



	Janet	**will**	**back**	**the**	**bill**
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 8.6 Observation likelihoods *B* computed from the WSJ corpus without smoothing.

(Fig from J&M draft 3rd edition)

But... tagging?

- Normally, we want to use the model to find the best tag sequence for an *untagged* sentence.

- Thus, the name of the model: **hidden Markov model**
  - **Markov**: because of Markov assumption (tag/state only depends on immediately previous tag/state).
  - **hidden**: because we only observe the words/emissions; the tags/states are hidden (or **latent**) variables.

- FSM view: given a sequence of words, what is the most probable state path that generated them?

Hidden Markov Model (HMM)

HMM is actually a very general model for sequences. Elements of an HMM:

- a set of states (here: the tags)
- an output alphabet (here: words)
- intitial state (here: beginning of sentence)
- state transition probabilities (here: $p(t_i|t_{i-1})$)
- symbol emission probabilities (here: $p(w_i|t_i)$)

Formalizing the tagging problem

- Normally, we want to use the model to find the best tag sequence T for an *untagged* sentence S:

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

Formalizing the tagging problem

- Normally, we want to use the model to find the best tag sequence T for an *untagged* sentence S:

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

- 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:

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

Decomposing the model

- Now we need to compute $P(S|T)$ and $P(T)$ (or really, $P(S|T)P(T) = P(S,T)$).
- We already defined how!
- $P(T)$ is the state transition sequence:

$$P(T) = \prod_i P(t_i|t_{i-1})$$

- $P(S|T)$ are the emission probabilities:

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

Search for the best tag sequence

- We have defined a model, but how do we use it?
  - given: word sequence S
  - wanted: best tag sequence T^*
- For any specific tag sequence T, it is easy to compute $P(S,T) = P(S|T)P(T)$.

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

- So, can't we just enumerate all possible T, compute their probabilites, and choose the best one?

Enumeration won't work

- Suppose we have c possible tags for each of the n words in the sentence.
- How many possible tag sequences?

Enumeration won't work

- Suppose we have c possible tags for each of the n words in the sentence.
- How many possible tag sequences?
- There are c^n possible tag sequences: the number grows *exponentially* in the length n.
- For all but small n, too many sequences to efficiently enumerate.

Finding the best path

- The **Viterbi algorithm** finds this path without explicitly enumerating all paths.
- A type of **dynamic programming** (or **memoization**) algorithm: an algorithm that stores partial results in a **chart** to avoid recomputing them.
- Dynamic programming is a widely used algorithmic technique in comp ling:
  – Computing the minimum edit distance between two strings (J&M 3.11)
  – Viterbi algorithm and forward-backward algorithm for HMMs
  – Many parsing algorithms for CFGs and PCFGs (e.g., chart parsing)

References

Petrov, S., Das, D., and McDonald, R. (2011). A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*.