Lecture 4: Smoothing, Part-of-Speech Tagging

Ivan Titov
Institute for Logic, Language and Computation
Universiteit van Amsterdam

Language Models from Corpora

We want a model of sentence probability $P(w_1, \ldots, w_n)$ for all word sequences $w_1, \ldots, w_n$ over the vocabulary V:

$$P(w_1, \ldots, w_n) = P(w_1) \prod_{i=2}^{n} P(w_i | w_1, \ldots, w_{i-1})$$

Tasks to do:

- Estimate $P(w_1)$
- Estimate probabilities $P(w_i | w_1, \ldots, w_{i-1})$ for all $w_1, \ldots, w_i$!

What to do in order to estimate these probabilities?

- Bucketing histories: Markov models
- Smoothing techniques against sparse-data

Markov Assumption and N-grams

Limited history: There is a fixed finite k such that for all w_1^{i+1}:

$$P(w_{i+1}|w_1,\ldots,w_i) \approx P(w_{i+1}|w_{i-k},\ldots,w_i)$$

For $k \geq 0$

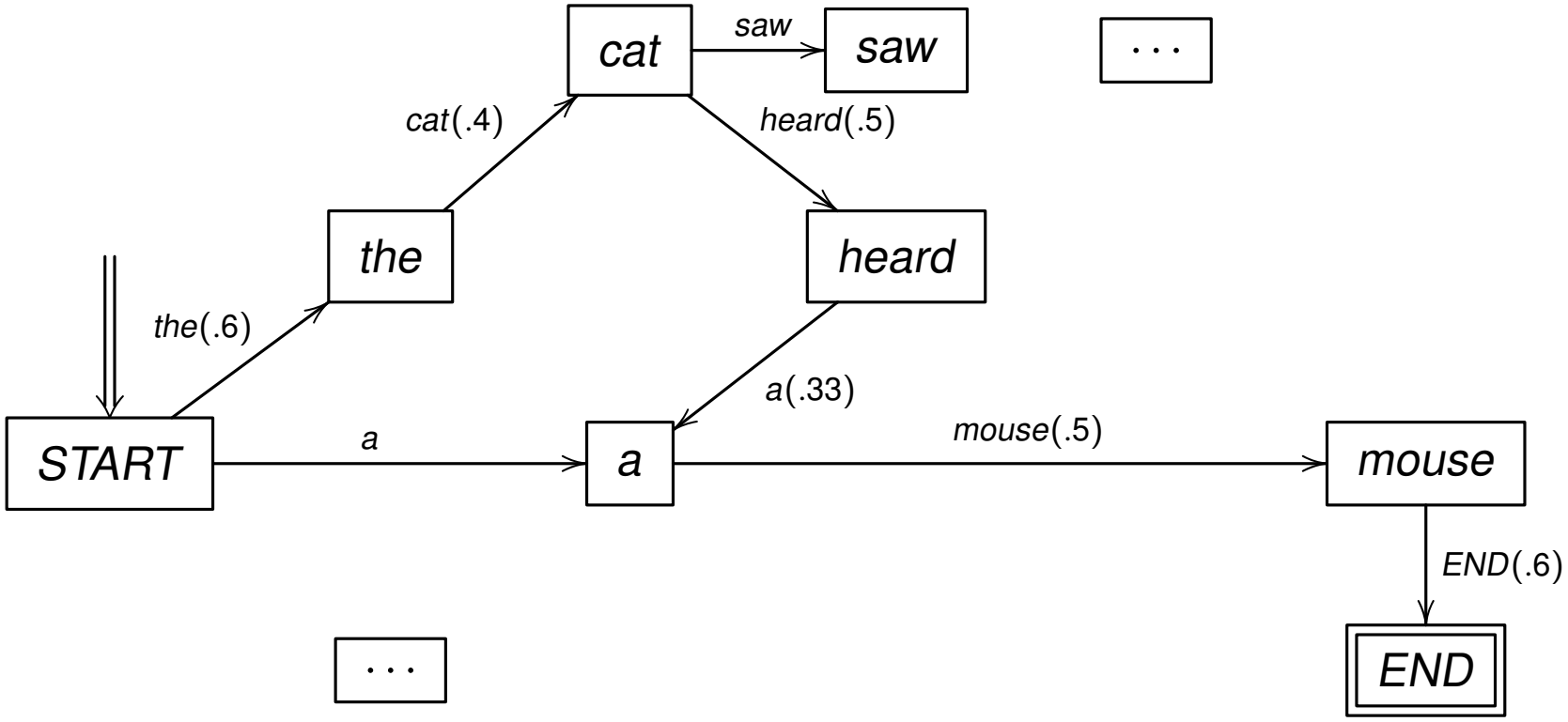
$$P(w_i \mid w_1,\ldots w_{i-k}\ldots,w_{i-1}) \approx P(w_i \mid w_{i-k},\ldots,w_{i-1})$$

How to estimate the probabilities?

Estimation: k^{th}-order Markov Model

$$P(w_i \mid w_{i-k},\ldots,w_{i-1}) = \frac{Count(w_{i-k},\ldots,w_{i-1},w_i)}{\sum_{w\in V} Count(w_{i-k},\ldots,w_{i-1},\ w)}$$

Bigram derivations



Likelihood:

P_{bigram} (“ the cat heard a mouse ”) $= .6 \times .4 \times .5 \times .33 \times .5 \times .6 = 0.12$

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. “the cat sees the mouse”, ..., “Tom sees Jerry”, “the mouse sees Jerry”

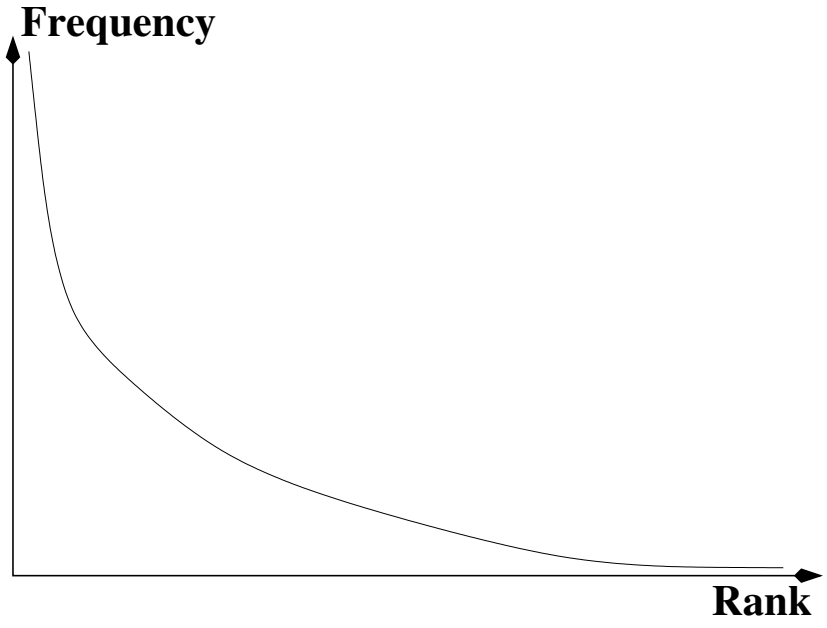
Problem: $P(\text{“Jerry sees the mouse”}) = 0$, why?

Data: text does not contain bigrams $\langle START, Jerry \rangle$ and $\langle Jerry, sees \rangle$

Zeros: are called “unseen events”,

Question: the above sentence should have a non-zero probability, how do we estimate its probability?

Zipf's law



- In a given corpus, a few very frequent words, but very many infrequent words.
- Having a resonably sized corpus is important, but we always need smoothing.
- Smoothing technique used has a large effect on the performance of any NLP system.

Smoothing techniques for Markov Models

General Idea

Find a way to fill the gaps in counts of events in corpus C.

- Take care not to change original distribution too much.
- Fill the gaps only as much as needed: as corpus grows larger, there will be less gaps to fill.

Naive smoothing: Adding λ method (1)

Events: Set E of possible events, e.g. bigrams over V: $E = (V \times V)$

Data: $e_1, \ldots e_N$ (data size is N events)

Counting: Event (bigram) e occurs $C(e)$ times in Data

Relative Frequency estimate: $P_{rf}(e) = \frac{C(e)}{N}$

Add $0 < \lambda \leq 1$: for all $e \in E$ (bigrams) change $C(e)$ and $P_{rf}(e)$

$$\hat{C}(e) = C(e) + \lambda$$

$$P_\lambda(e) = \frac{C(e) + \lambda}{N + \lambda|E|}$$

Add λ method (2)

Example: Bigram Model

$$P_\lambda(w_i|w_{i-1}) = \frac{\lambda + c(w_{i-1}w_i)}{\sum_w(\lambda + c(w_{i-1}, w))}$$

Advantages: very simple and easy to apply

Disadvantges: Method performs poorly (see Chen & Goodman):

- All unseen events receive same probability! Is that OK?
- All events upgraded by λ! Is that OK?

Good-Turing method

Intuition Use the counts of things you have seen *once* to estimate the count of things not seen.
i.e. use n-grams with frequency 1 to re-estimate the frequency of zero counts.

Suppose we have data with total count of events being N:

Standard notation:

r	=	frequency of event e
n_r	=	number of events e with frequency r
		$n_r = \|\{e \in E \| Count(e) = r\}\|$
N_r	=	total frequency of events occuring exactly r times
		$N_r = r \times n_r$

Observation: $N = \sum_{r=1}^{\infty} N_r$ $\qquad\qquad$ $N_0 = 0$

What we want: To recalculate the frequency r of an event (r^*)

Good-Turing Estimates

The Good-Turing probability estimate for events with frequency r:

$$P_{GT}(\alpha) = \frac{r+1}{N}\frac{E[n_{r+1}]}{E[n_r]} \approx \frac{r+1}{N}\frac{n_{r+1}}{n_r} = \frac{1}{N} \times (r+1) \times \frac{n_{r+1}}{n_r}$$

We can think of this, as assigning frequency of r^* to events appearing r times:

$$r^* = (r+1) \times \frac{n_{r+1}}{n_r}$$

n_r : number of events with freq. r

n_{r+1} : number of events with freq. $r+1$

Properties of Good-Turing

Preservation: Total number of counts is preserved:

$$N = \sum_{r=1}^{\infty} r n_r = \sum_{r=0}^{\infty} (r+1) n_{r+1} = \sum_{r=0}^{\infty} n_r r^*$$

Discounting: Total freq. for non-zero events is discounted

$$N_0 = n_0 \times 0^* = n_0 \times (1 \times \frac{n_1}{n_0}) = n_1$$

Zero freq. events

$$P_0 = \frac{r^*}{N} = \frac{0*}{N} = \frac{n_1}{N}$$

Zero events: No explicit method for redistributing N_0 among zero events!

Redistribute the reserved mass (N_0) uniformly among zero events?

Estimating n_r from Data

The values n_r from data could be sparse themselves!

GT Good suggests to estimate $E[n_r] \approx S(n_r)$ for some smoothing function S over the values n_r.

SGT William Gale suggests the Simple Good-Turing where function S is linear $S(log(n_r)) = a + b\ log(r)$ and fitted by logistic regression such that $b < -1$ (to guarantee that $r^* < r$).

Available the implementation by Geoffrey Sampson on the web.

Problem: Good-Turing says nothing about how to redistribute probability mass over unseen-events!

Why Uniform Redistribution is Not Good

Given some data, we observe that

- Frequency of trigram $\langle gave, the, thing \rangle$ is zero
- Frequency of trigram $\langle gave, the, think \rangle$ is also zero
- Uniform distribution over unseen events means:

$$P(thing|gave, the) = P(think|gave, the)$$

Does that reflect our knowledge about English use?

$$P(thing|gave, the) > P(think|gave, the)$$

How do we achieve this?

- This might be achieved if we look at bigrams!
  $P(thing|the)$ vs. $P(think|the)$

Katz's Backoff

In order to re-distribute reserved mass, use a lower-order distribution

Discount and backoff Suppose $C(w_{i-1}w_i) = r$:

$$C_{katz}(w_{i-1}w_i) = \begin{cases} d_r \times r & \text{if } r > 0 \\ \alpha(w_{i-1}) \times P_{ML}(w_i) & \text{if } r = 0 \end{cases}$$

Discounting all non-zero counts are discounted by $1 - d_r$

Redistribution of reserved mass over zero-events is proportional to their lower-order distribution, for bigrams this is unigrams, for trigrams this is bigrams etc.

Computing d_r

Efficiency: discount only events occuring $1 \leq r \leq k$: e.g. $k = 5$,

Constraint 1: Discounts should be proportional to discounts of GT, i.e. there exists some constant μ such that $1 - d_r = \mu(1 - \frac{r^*}{r})$

Constraint 2: The total mass of zero-events is the same as for GT GT assign to all zero-events: $n_0 0^* = n_0(\frac{n_1}{n_0}) = n_1$:

$$\sum_{r=1}^{k} n_r \times [1 - d_r] \times r = n_1$$

Solution: The above system of equations has the solution:

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}$$

Solution for α

Solution for $\alpha(w_{i-1})$ is through enforcing the constraint that total counts are not changed:
$\sum_{w_i} C_{katz}(w_{i-1} w_i) = \sum_{w_i} C(w_{i-1} w_i)$

See the Chen and Goodman tech report for details.

Smoothing of Markov Model

Smoothing an $(n+1)$ model gives us an n-gram model.

$$P^*(w_{n+1}|w_1,\ldots,w_n) = \frac{Count^*(w_1,\ldots,w_n,w_{n+1})}{\sum_{w\in V} Count^*(w_1,\ldots,w_n,w)}$$

For many smoothing methods:

$$\sum_{w\in V} Count^*(w_1,\ldots,w_n,w) \neq Count^*(w_1,\ldots,w_n)$$

Smoothing by linear interpolation: Jelinek-Mercer

(Also called finite mixture models)

Interpolate a bigram with a unigram model using some $0 \leq \lambda \leq 1$

$$P_{interp}(w_{i-1}w_i) = \lambda P_{ML}(w_{i-1}w_i) + (1 - \lambda)P_{ML}(w_i)$$

Estimation: data for P_{ML} and held-out data for estimating the λ's
What would happen if use the training data to estimate λ?

Algorithm: Baum-Welch/Forward-Backward for Hidden Markov Models searching for a Maximum-Likelihood estimate of the λ.

More on this when we learn HMMs!

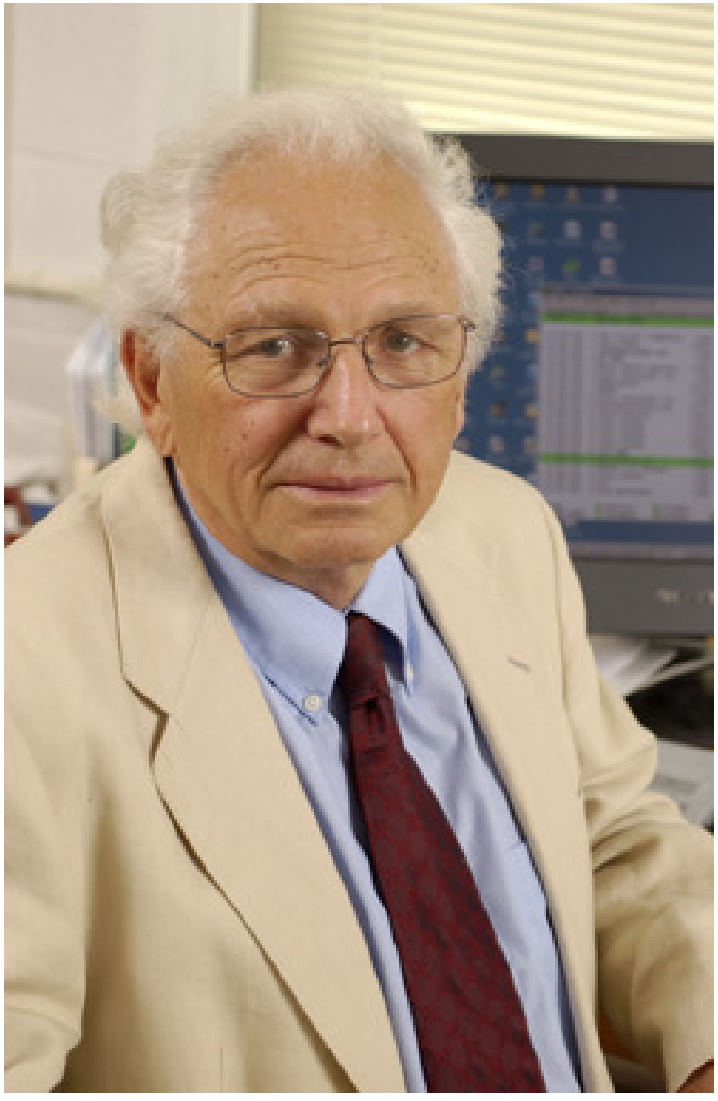
Summary of Smoothing Techniques

- *n*-gram statistics suffer from sparseness in text as *n* grows
- There are smoothing methods against sparseness
- Add λ is too simple
- Good-Turing only for reserving mass for zero-events
- Katz allows redistribution according to a lower order distribution
- Interpolation combines lower order distributions

Smoothing N-gram Models

1. Add λ method
2. Good-Turing method
3. Discounting: Katz' backoff
4. Interpolation: Jelinek-Marcer

Extra Reference: Joshua Goodman and Stanly Chen. *An empirical study of smoothing techniques for language modeling*. Tech. report TR-10-98, Harvard University, August 1998. `http://research.microsoft.com/~joshuago/`



- Born 1932 (Vilém, Czechoslovakia) from Jewish father and Catholic mother;
- Father died in Theresienstadt; mother & children survived in Prague.
- Electrical Engineering City College of New York, 1962 PhD MIT (under Robert Fano; Shannon's department)
- 1957 met Milena Tabolova (arrived 1961, studied linguistics at MIT), together attended Chomsky's classes.
- 1962-1972 Cornell, focus Information Theory
- 1972-1993, IBM Watson Research Center: applied noisy channel model to speech recognition & machine translation
- 1993-2010, Johns Hopkins
- died 2010.

Shannon (1956): “Information theory has, in the last few years, become something of a scientific bandwagon. Starting as a technical tool for the communication engineer, it has received an extraordinary amount of publicity in the popular as well as the scientific press... Our fellow scientists in many different fields, attracted by the fanfare and by the new avenues opened to scientific analysis, are using these ideas in their own problems. Applications are being made to biology, psychology, linguistics, fundamental physics, economics, the theory of organization, and many others... Although this wave of popularity is certainly pleasant and exciting for those of us working in the field, it carries at the same time an element of danger... It will be all too easy for our somewhat artificial prosperity to collapse overnight when it is realized that the use of a few exciting words like information, entropy, redundancy, do not solve all our problems.”

- Pierce and Caroll, ALPAC report (1956) persuaded the U.S. government to stop funding machine translation (MT) research.
- “Certainly, language is second to no phenomenon in importance. The new linguistics presents an attractive as well as an extremely important challenge.”
- Machine translation is field for “mad inventors or untrustworthy engineers”
- Pierce (1959, JASA): “We are safe in asserting that speech recognition is attractive to money. The attraction is perhaps similar to the attraction of schemes for turning water into gasoline, extracting gold from the sea, curing cancer, or going to the moon.
- Chomsky (1957): “Probabilistic models give no insight into basic problems of syntactic structure”
- Pierce: “Funding artificial intelligence is real stupidity”

In this climate, Jelinek pioneered statistical models for speech recognition and machine translation, reconciling Shannon's and Chomsky's contributions.

- In 1976 (Proceedings of the IEEE) "Continuous Speech Recognition by Statistical Methods"
- "eventually his view of the problem entirely reshaped the landscape and set the foundations for virtually all modern speech recognition systems."
- "in his 20 years at IBM, he and his team invented nearly all of the key components that you need to build a high performance speech recognition system including phone-based acoustic models, N-gram language models, decision tree clustering, and many more"
- 2000 (with Chelba) "Structured language modeling for speech recognition": PCFGs as language models.

POS tagging: Background

What are POS tags?

- A Part of Speech (POS) tag is the *class* of a word

  Examples: Verbs (VB), Nouns (NN), Adjectives (ADJ), Adverbs (ADV), Prepositions (PP), Determiners (DT)
- A POS tag provides significant info about a word and its neighbourhood

  Example: A verb occurs in different neighbourhoods than a noun
  - Nouns are likely to be preceded by determiners, nouns, adjectives
  - Verbs are likely to be preceded by nouns, pronouns, adverbs
- POS tags capture mainly morpho-syntactic behaviour
- POS tag sets are only discrete approximations for the behaviour of words in sentences!

What are POS tags?

- POS tags are also known as word classes, morphological classes, or lexical tags
- Number of tags used by different systems/corpora/languages are different
  - Penn Treebank (Wall Street Journal Newswire): 45
  - Brown corpus (Mixed genres like fiction, biographies, etc): 87
- POS tags can be of varying granularity
  - Morphologically complex languages have very different tag-sets from English or Dutch.
  - Even for English:

He	gave	the book to the man
	VB-N-N	

He	ate	the banana.
	VB-N	

POS tags are useful

Pronunciation: Examples:

NN	CONtent	NN	OBject	NN	DIScount
ADJ	conTENT	VB	obJECT	VB	disCOUNT

Spelling correction: knowing the POS tag of a word helps.

`The house be the sea is beautiful.`
(VB be) is less likely to follow (NN house) than (PP by)

Parsing: POS tags say a lot about syntactic word behavior.

Speech recognition: POS tags are used in a language model component

Information Retrieval: important words are often (e.g.) *nouns.*

POS-tagging is often the first step in a larger NLP system.

POS tagging and ambiguity

List	the	list	.		Book	that	flight	.
VB	DT	NN	.		VB	DT	NN	.

He	thought	that	the	book	was	long	.
PRN	VB	COMP	DT	NN	VB	ADJ.	

Ambiguity:

list	VB/NN	
book	VB/NN	
following	NN/VB/ADJ	
that	DT/COMP	*I thought (COMP that) . . .*

POS tag ambiguity resolution demands “some” syntactic analysis!

Why is POS tagging hard

Ambiguity:

Most words belong to more than one word-class

- List/VB the list/NN
- He will race/VB the car
- When is the race/NN?

Some ambiguities are genuinely difficult, even for people

- particle vs. preposition
  - He talked over the deal.
  - He talked over the phone.
- past tense vs. past participle
  - The horse raced past the barn.
  - The horse raced past the barn was brown.
  - The old man the boat.

How hard is POS tagging?

Using the Brown Corpus (approx. 1 million words of mixed-genre text), a study shows that for English:

- Over 40% of all word tokens are ambiguous!
- Many ambiguities are easy to resolve: just pick the most likely tag!
- A good portion is still hard: picking most likely tag results only in 90% per word correct POS tag:

  Sentence of length n words is fully correct only in: $(0.9)^n$ %

Length	10	20	30	40	50
Precision	34%	12%	4.2%	1.5%	0.5%

Maybe we can get away with using local context?

POS-tagging is among the easiest of NLP problems

- State-of-the-art methods get about 97-98% accuracy
- Simple heuristics can take you quite far
  - 90% accuracy just by choosing the most frequent tag.
- A good portion is still hard: need to use some local context.

Current POS taggers

"Rule-based": set of rules constructed manually (e.g. ENGTWOL: CG)

Stochastic: Markov taggers, HMM taggers (and other ML techniques)

This lecture: Stochastic Taggers

- What is a stochastic tagger? What components does it contain?
- How to use Markov models?
- Hidden Markov Models for tagging.

A generative stochastic tagger

Given a vocabulary V and POS tag set T

Input:	$w_1, \ldots, w_n$	all $w_i \in V$
Output:	$t_1, \ldots, t_n$	all $t_i \in T$
Tagger:	$P : (V \times T)^+ \rightarrow [0, 1]$	
Disambiguation:	Select	$\widehat{t_1^n} = argmax_{t_1^n} P(t_1^n \mid w_1^n)$

Partioning: Task Model & Language Model

$$argmax_{t_1^n} P(t_1^n \mid w_1^n) = argmax_{t_1^n} \frac{P(t_1^n, w_1^n)}{P(w_1^n)} = argmax_{t_1^n} P(t_1^n, w_1^n)$$

$$P(t_1^n, w_1^n) = \underbrace{P(t_1^n)} \times \underbrace{P(w_1^n \mid t_1^n)}$$

"Language model"
$P(t_1^n)$

"Task model"
$P(w_1^n \mid t_1^n)$

Independence Assumptions

Language Model

Approximate $P(t_1^n)$ using **n-gram model**

bigram $\prod_{i=1}^{n} P(t_i \mid t_{i-1})$

trigram $\prod_{i=1}^{n} P(t_i \mid t_{i-2} t_{i-1})$

Task Model

$P(w_1^n \mid t_1^n)$: Approximate by assuming that a word appears in a category independent of its neighbours.

$$P(w_1^n \mid t_1^n) = \prod_{i=1}^{n} P(w_i \mid t_i)$$

We can smooth the language model, using the smoothing techniques from earlier in this lecture.

Training: estimating the probabilities

Given a training corpus with every word tagged with its correct tag.

Using relative frequencies from training corpus

Task model:

$$P(w_i \mid t_i) = \frac{Count(\langle w_i, t_i \rangle)}{\sum_w Count(\langle w, t_i \rangle)} = \frac{Count(\langle w_i, t_i \rangle)}{Count(\langle t_i \rangle)}$$

Language model (trigram model shown here):

$$P(t_i \mid t_{i-2}, t_{i-1}) = \frac{Count(t_{i-2}, t_{i-1}, t_i)}{\sum_{t \in V} Count(t_{i-2}, t_{i-1}, t)}$$

Unknown words in the task model (1)

When a new word arrives w_u, we never have seen any tag with this word.

Hence: for all tags $t \in T$: $P(w_u \mid t) = 0$

Performance of state-of-the-art taggers depends largely on treating unknown words!

What should we do?

Use knowledge of language:

Morphology: eating, filled, trousers, Noam, naturally

POS tag classes: Closed-class versus open-class words.

Some classes such as Proper Nouns and Verbs are much more likely to generate new words than other classes e.g. prepositions, determiners.

Unknown words in the task model (2)

To capture the fact that some POS classes are more likely to generate unknown words, we can simply use Good-Turing smoothing on $P(w|t)$.

n_0: number of unknown words.
$n_1(t)$: number of words which appeared once with tag t.
$N(t)$: total number of words which appeared with tag t.
Good-Turing estimate for unknown word u:

$$P(u|t) = \frac{n_1(t)}{n_0 N(t)}$$

Unknown words: Good Turing Example

$$P(u|t) = \frac{n_1(t)}{n_0 N(t)}$$

Number of unknown words: 10

$t = NN$	$t = VBD$	$t = PREP$
$n_1(NN) = 150$	$n_1(VBD) = 100$	$n_1(PREP) = 2$
$N(NN) = 3000$	$N(VBD) = 1000$	$N(PREP) = 5000$
$P(u\|NN) = 1/200$	$P(u\|VBD) = 1/100$	$P(u \mid PREP) = 1/25000$

Unknown words in the task model (3)

- For each morphological feature: capitalization, hyphen,etc. Separately train estimators : $P(\mathit{feature} \mid \mathit{tag})$
- Assume independence and multiply.

- The model of Weischedel et al. (1993)

$$\begin{aligned} P(w, \mathit{unknown} \mid t) \quad \propto \quad & P(\mathit{unknown} \mid t) \times \\ & P(\mathit{capitalized}(w) \mid t) \times \\ & P(\mathit{suffix}(w)/\mathit{hyphen}(w) \mid t) \end{aligned}$$

This treatment of unknown words might not be suitable for "morphologically-rich" languages such as Finnish, Hebrew or Arabic

Next Lecture

- Tagging (dynamic programming)
- Recap Part A (exam preparation)
- Syntax: outlook
- No lecture on February 18