CS442/542b: Artificial Intelligence II
Prof. Olga Veksler

Lecture 9

NLP: Language Models

Many slides from: Joshua Goodman, L. Kosseim, D. Klein

Outline

- Why we need to model language
- Probability background
  - Basic probability axioms
  - Conditional probability
  - Bayes' rule
- n-gram model
- Parameter Estimation Techniques
  - MLE
  - Smoothing

Why Model Language?

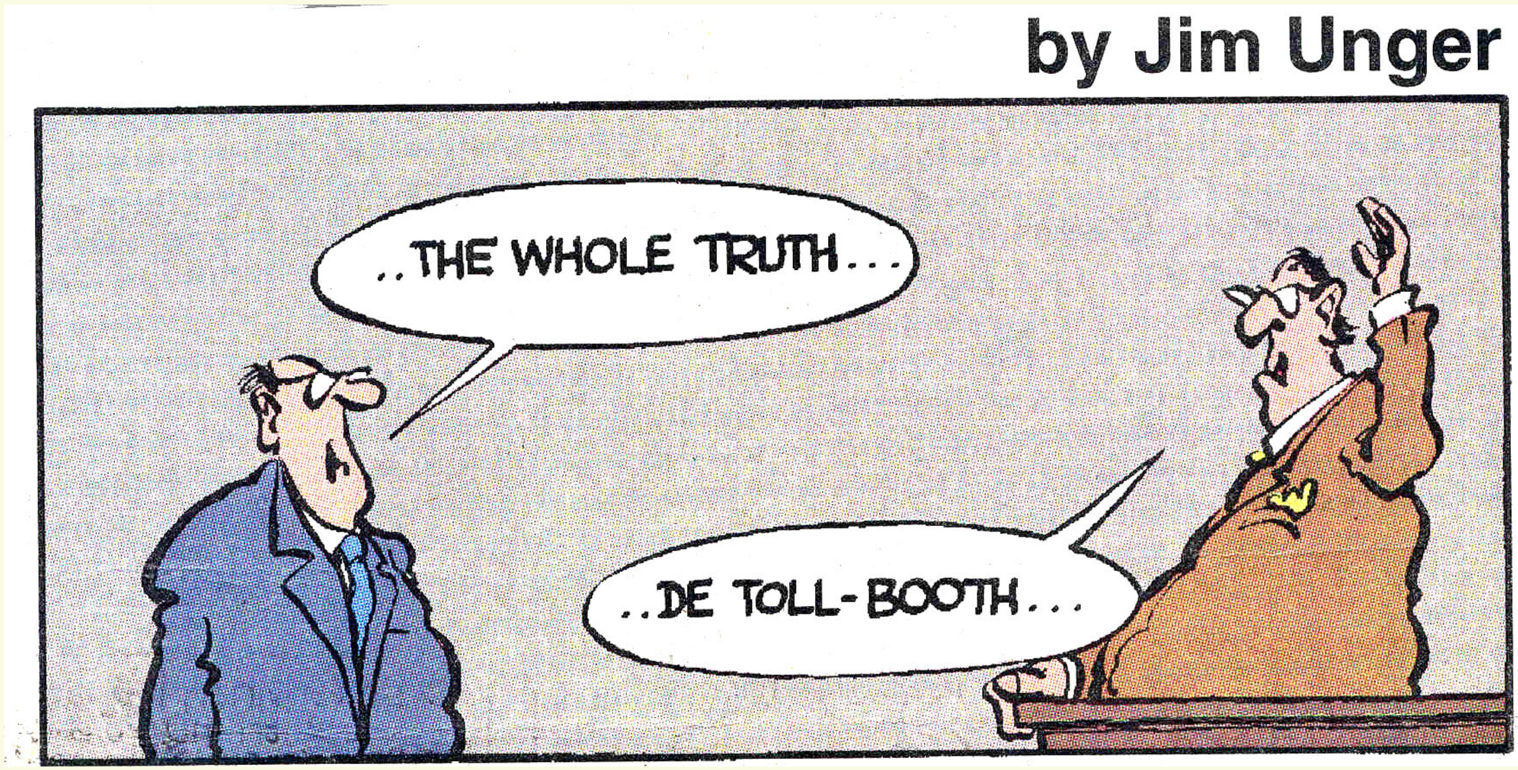
- Some sequences of words are more likely to be a good English sentence than others
- Want a probability model P s.t.

  P(unlikely sentence) < P(likely sentence)
- Useful in
  - Spell checker: "I think there are OK" vs. "I think they are OK"
  - Speech recognition: "lie cured mother" vs "like your mother"
  - Optical character recognition: "thl cat" vs. "the cat"
  - Machine translation: "*On voit Jon à la télévision*"
    - Jon appeared in TV
    - In Jon appeared TV
    - Jon appeared on TV
  - lots of other applications
  - In all of the above cases, we chose the sentence with higher probability according to the model P

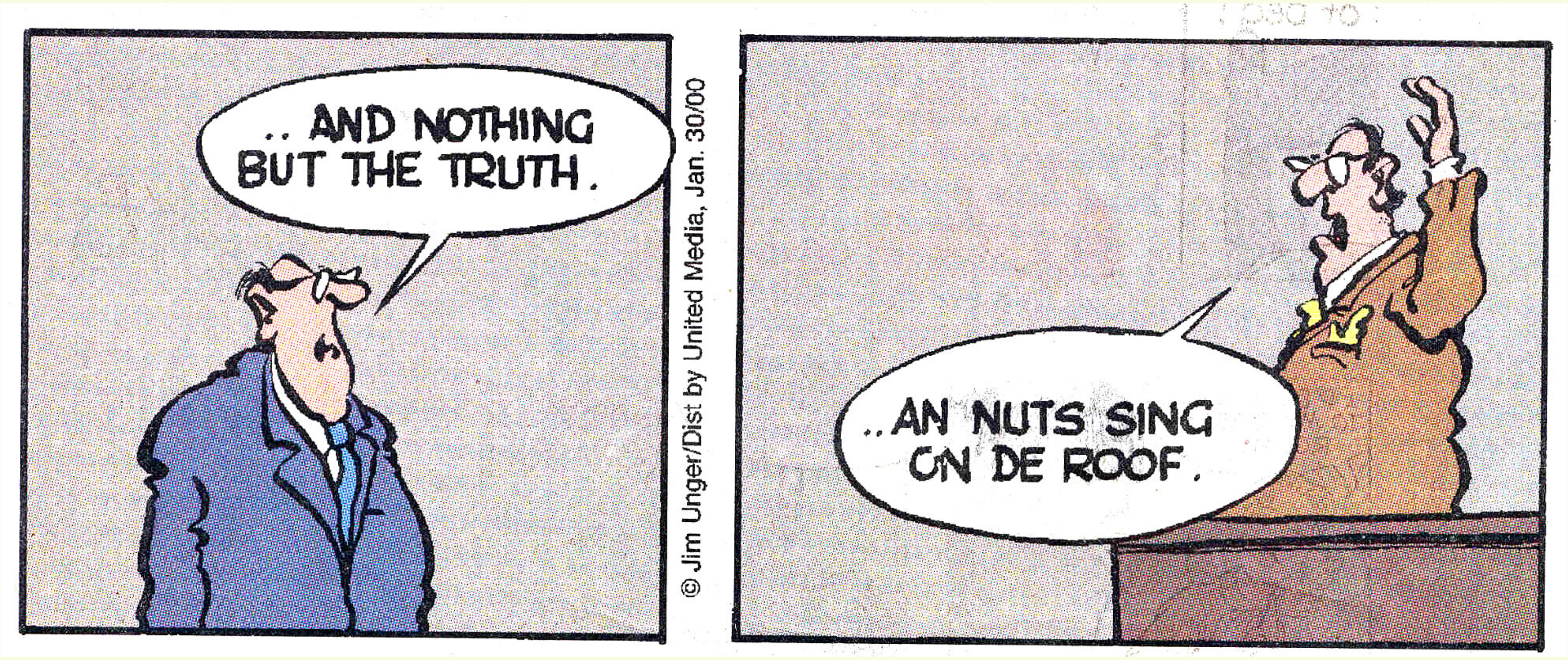
Language Model for Speech Recognition

Slides 2-7, from Joshua Goodman's slides
research.microsoft.com/~joshuago/lm-tutorial-public.ppt

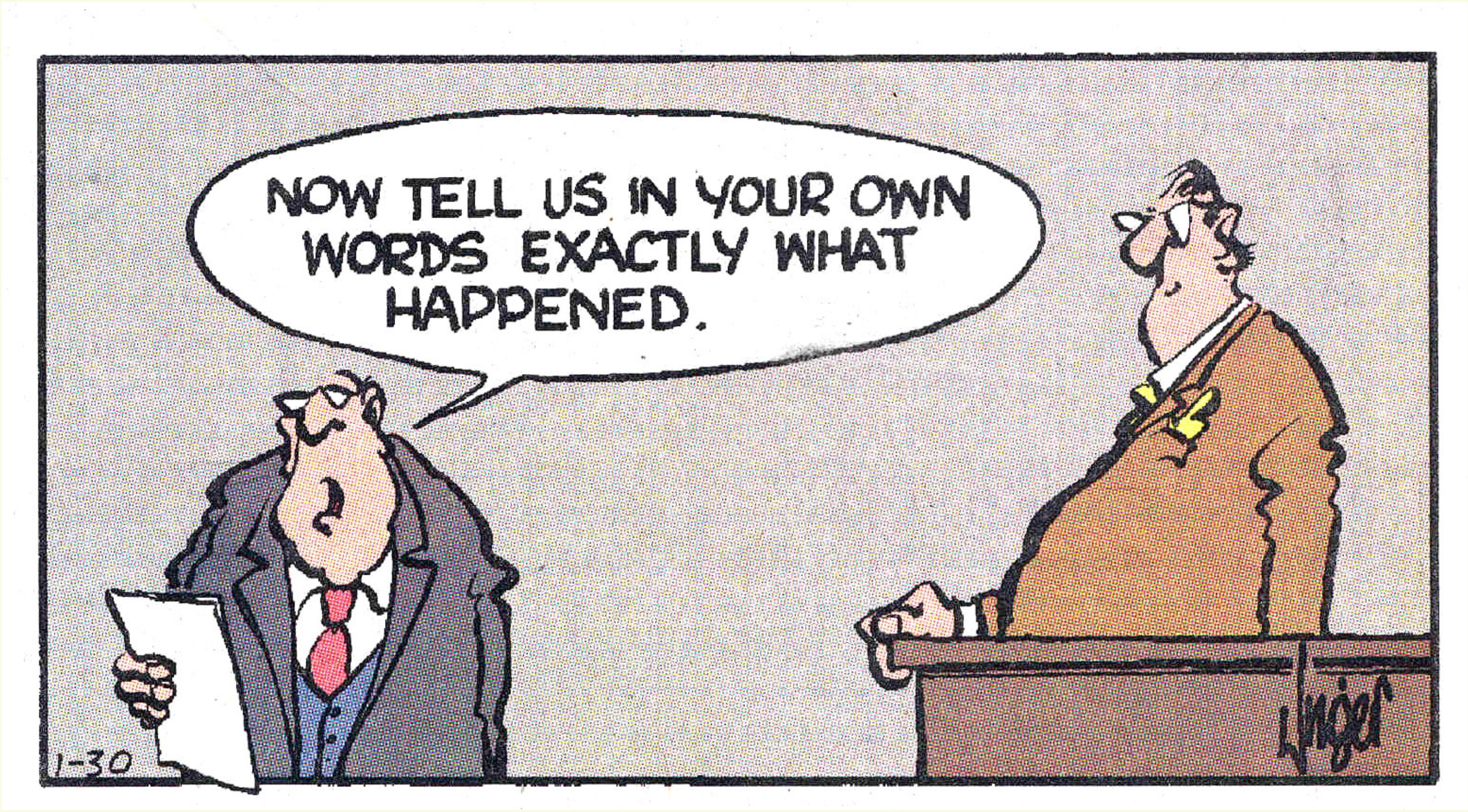
Language Model for Speech Recognition



Language Model for Speech Recognition



Language Model for Speech Recognition

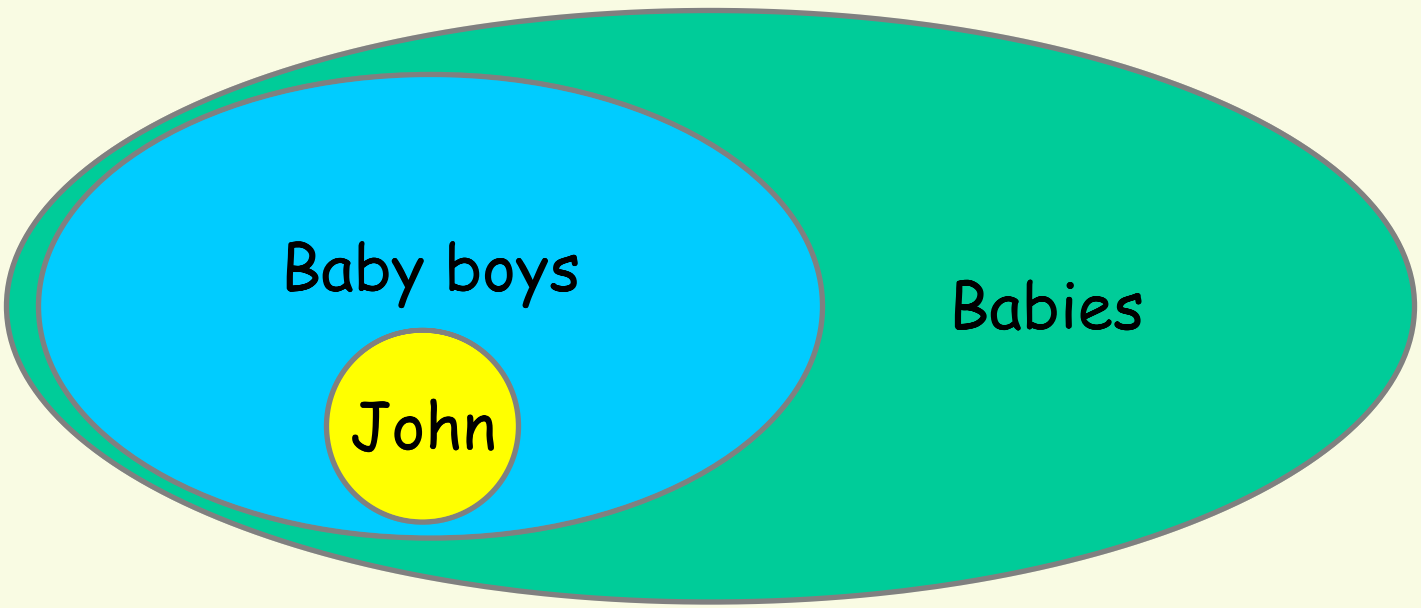


What is a Language Model?

- A language model is a probability distribution over word/character sequences
- We would like to find a language model P s.t.
  - P(“And nothing but the truth”) $\approx$ 0.001
  - P(“And nuts sing on the roof”) $\approx$ 0.000000001

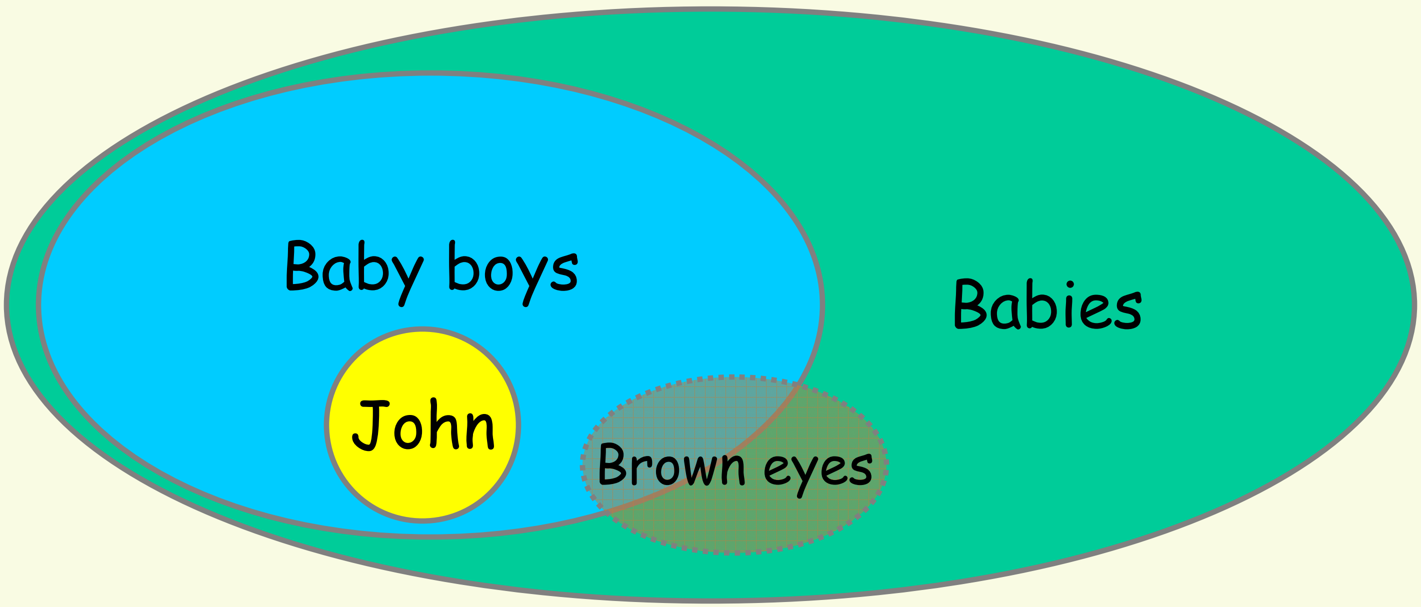
Basic Probability

- P(X) means probability that X is true
  - P(baby is a boy) = 0.5 (1/2 of all babies are boys)
  - P(baby is named John) = 0.001 (1 in1000 babies is named John)



Joint probabilities

- P(X,Y) means probability that X and Y are both true, for example:
  P(brown eyes, boy) = (number of all baby boys with brown eyes)/(total number of babies)

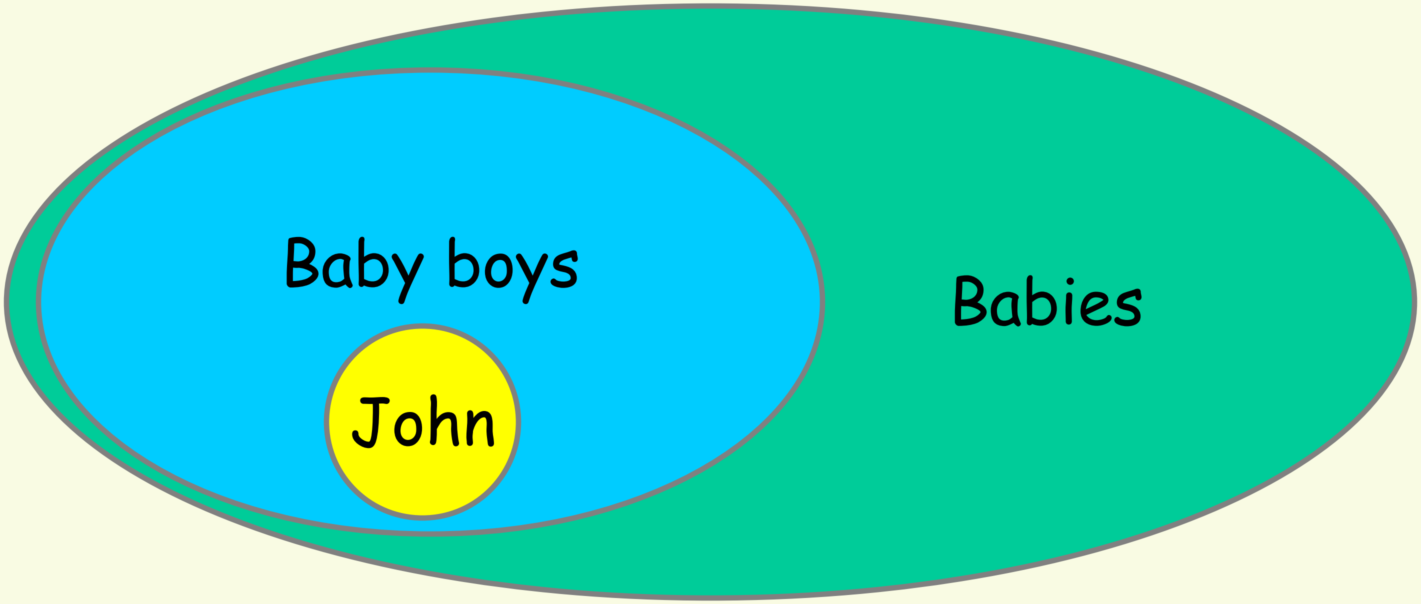


Conditional Probability

- P(X|Y) = P(X, Y) / P(Y)

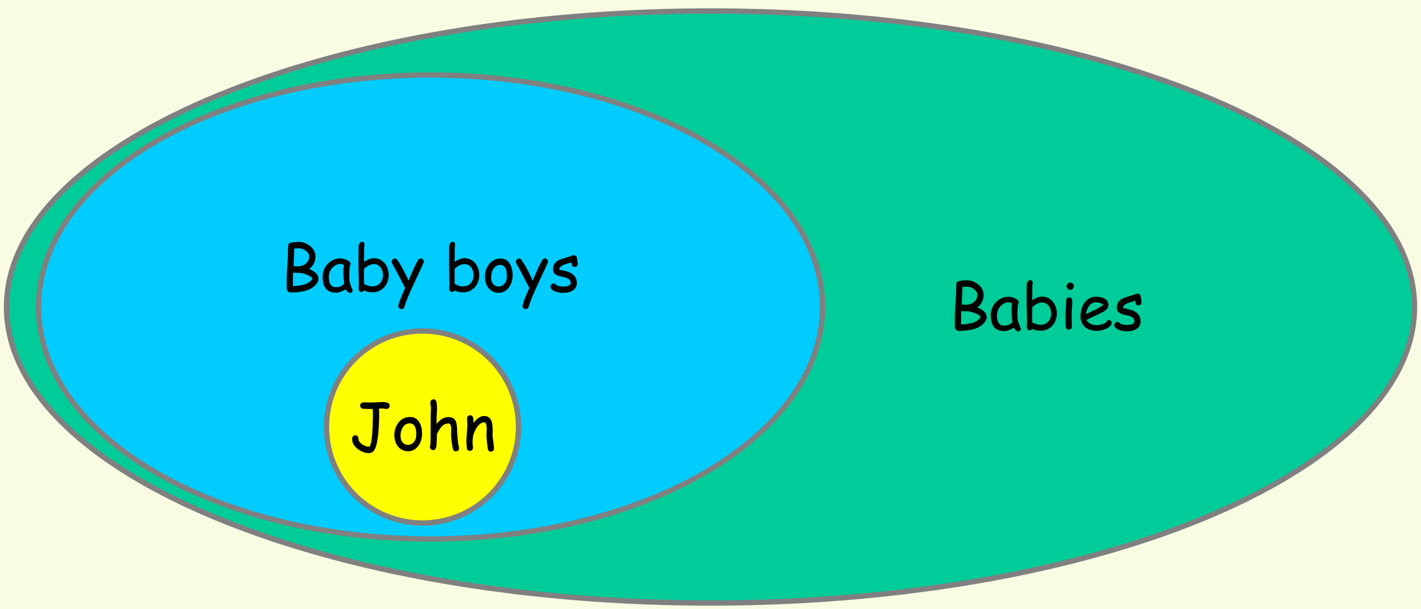
  P(baby is named John | baby is a boy) =

$$\frac{\text{P(baby is named John, baby is a boy)}}{\text{P(baby is a boy)}} = \frac{0.001}{0.5} = 0.002$$



Conditional probability

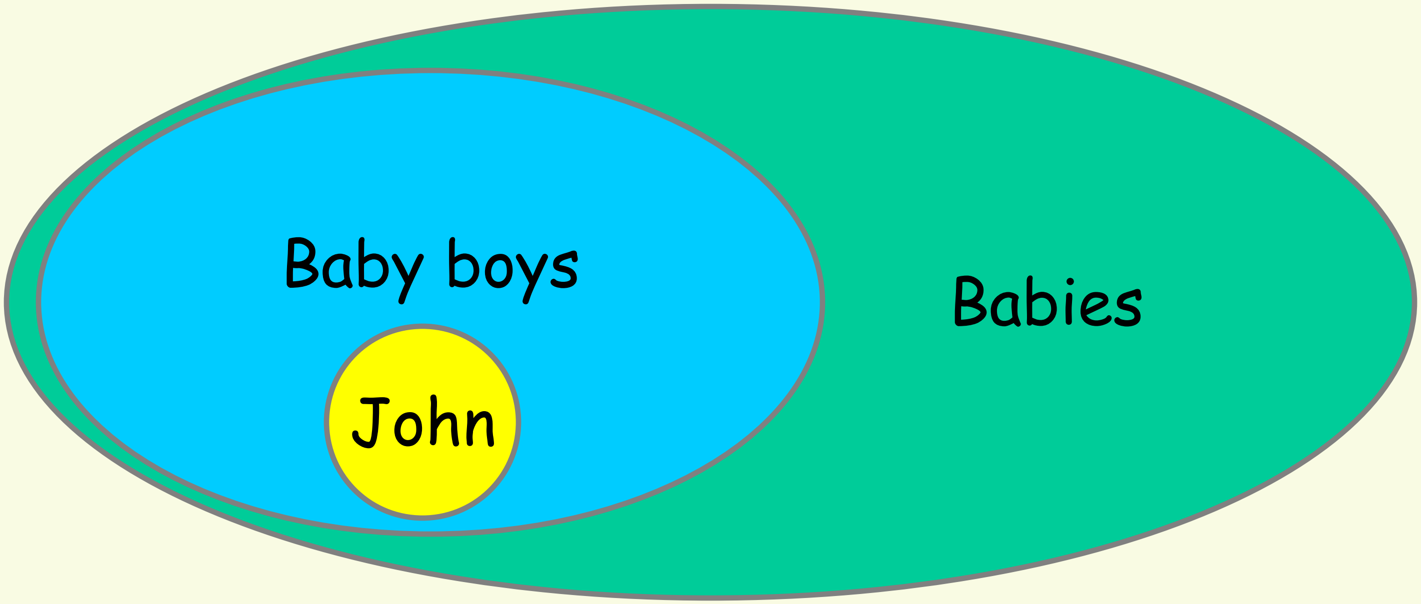
- P(X|Y) means probability that X is true when we already know Y is true
  - P(baby is named John | baby is a boy) = 0.002
  - P(baby is a boy | baby is named John) = 1



Bayes Rule

- Bayes rule: $P(X \mid Y) = \dfrac{P(Y \mid X)P(X)}{P(Y)}$

$$P(named\ John \mid boy) = \frac{P(boy \mid named\ \ John)P(named\ John)}{P(boy)}$$



Speech Recognition Example

$$\overbrace{P(\text{word sequence} \mid \text{acoustics})}^{\text{very hard to model}} = \frac{\overbrace{P(\text{acoustics} \mid \text{word sequence})}^{\text{reasonably easy to model}} \times \overbrace{P(\text{word sequence})}^{\text{from language model}}}{\underbrace{P(\text{acoustics})}_{\text{usually don't need this}}}$$

Language Modeling

- Let V be the set of words, V={a, apple,..,zoo}
- A sentence X is a sequence of words in V, for example S = "John went to the zoo"
- We need to learn the probability distribution P from the training data s.t.

$$P(S) \geq 0 \qquad\qquad \sum_{all\ sentences\ S} P(S) = 1$$

Language Modeling

- In our case, events will be sequences of words, for example "an apple fell"
- P("an apple fell") is the probability of the joint event that
  - the first word in a sequence is "an"
  - the second word in a sequence is "apple"
  - the third word in a sequence is "fell"
- P(fell | an apple) should be read as probability that the third word in a sequence is "fell" given that the previous 2 words are "an apple"

How Language Models work

- Hard to compute $P(\textit{and nothing but the truth})$
- Step 1: Decompose probability using conditional probability:

$$
\begin{aligned}
&P(\textit{and nothing but the truth}) = \\
&= P(\textit{truth} \mid \textit{and nothing but the})P(\textit{and nothing but the}) = \\
&= P(\textit{truth} \mid \textit{and nothing but the})P(\textit{the} \mid \textit{and nothing but}) \times \\
&\times P(\textit{and nothing but}) = \\
&= P(\textit{truth} \mid \textit{and nothing but the})P(\textit{the} \mid \textit{and nothing but}) \times \\
&\times P(\textit{but} \mid \textit{and nothing})P(\textit{and nothing}) = \\
&= P(\textit{truth} \mid \textit{and nothing but the})P(\textit{the} \mid \textit{and nothing but}) \times \\
&\times P(\textit{but} \mid \textit{and nothing})P(\textit{nothing} \mid \textit{and})P(\textit{and})
\end{aligned}
$$

How Language Models work

- Consider

  P(computer | Instead of working every day, I would like to play on my)
- Probability that the word “computer” follows words “Instead of working every day, I would like to play on my” is intuitively almost the same as probability that the word “computer” follows words “play on my”
- The probability of the next word depends mostly on the few previous words

"Shannon Game" (Shannon, 1951)

"I am going to make a collect …"

- Predict the next word/character given the *n-1* previous words/characters.
- Human subjects were shown 100 characters of text and were asked to guess the next character
- As context increases, entropy decreases
  - the smaller the entropy => the larger the probability of predicting the next letter

Context	0	1	2	3
Entropy (H)	4.76	4.03	3.21	3.1

- But only a few words is enough to make a good prediction on the next word, in most cases
- Evidence that we only need to look back at n-1 previous words

n-grams

- n-gram model: the probability of a word depends only on the n-1 previous words (the history)

$$P(w_k | w_1 w_2 \ldots w_{k-1}) = P(w_k | w_{k+1-n} \ldots w_{k-1})$$

- This called **Markov Assumption**: only the closest n words are relevant:
  - Unigram: previous words do not matter
  - Bigram: only the previous one word matters
  - Trigram: only the previous two words matter

Example: The Trigram Approximation

- Assume each word depends only on the previous two words
  - three words total
  - tri means three
  - gram means writing

- P("the|… whole truth and nothing but") ≈
  P("the|nothing but")
- P("truth|… whole truth and nothing but the") ≈
  P("truth|but the")

The Trigram Approximation

- After decomposition we have:

$$P(\textit{and nothing but the truth}) =$$
$$= P(\textit{truth} \mid \cancel{\textit{and nothing}}\ \textit{but the}) P(\textit{the} \mid \cancel{\textit{and}}\ \textit{nothing but}) \times$$
$$\times P(\textit{but} \mid \textit{and nothing}) P(\textit{nothing} \mid \textit{and}) P(\textit{and})$$

- Using trigram approximation:

$$P(\textit{and nothing but the truth}) \approx$$
$$\approx P(\textit{truth} \mid \textit{but the}) P(\textit{the} \mid \textit{nothing but}) \times$$
$$\times P(\textit{but} \mid \textit{and nothing}) P(\textit{nothing} \mid \textit{and}) P(\textit{and})$$

  - Intuition: probability of each sentence is approximated as a product of probabilities of each individual word
    - Where probability of each individual word is conditioned on the previous two words

Trigrams, continued

- How do we find all the probabilities?
  - P(nextWord | prevWord2 PrevWord1)
  - These probabilities are usually called "parameters"
- Get real text, and start counting!
  - Let C1 be the count of how many times the phrase "nothing but the" occurred in the training corpus
  - Let C2 be the count of how many times the phrase "nothing but" occurred in the training corpus

$$P(the \mid nothing\ but) = \frac{P(nothing\ but\ the)}{P(nothing\ but)} \approx \frac{C1}{C2}$$

Trigrams, continued

$$P(\textit{and nothing but the truth}) =$$
$$= P(\textit{truth} \mid \textit{but the})P(\textit{the} \mid \textit{nothing but}) \times$$
$$\times P(\textit{but} \mid \textit{and nothing})P(\textit{nothing} \mid \textit{and})P(\textit{and})$$

- The approximation to P(and nothing but the truth)

$$\approx \frac{C_{\textit{but the truth}}}{C_{\textit{but the}}} \frac{C_{\textit{nothing but the}}}{C_{\textit{nothing but}}} \frac{C_{\textit{and nothing but}}}{C_{\textit{and nothing}}} \frac{C_{\textit{and nothing}}}{C_{\textit{and}}} \frac{C_{\textit{and}}}{N}$$

  - where N is the number of words in our training text

Bigrams

- first-order Markov models

$$P(w_n | w_{n-1})$$

- Can construct V-by-V matrix of probabilities/frequencies
- V = size of the vocabulary we are modeling

2nd word

1st word

	a	**an**	**apple**	**…**	**zoo**	**zucchini**
a	0	0	0		8	5
an	0	0	20		0	0
apple	0	0	0		1	3
…	…	…	…		…	…
zoo	0	2	0		0	0
zucchini	0	0	3		0	0

Problems with n-grams

- *"the large green ______ ."*
  - "*mountain*"? "*tree*"?
- *"Sue swallowed the large green ______ ."*
  - "*pill*"? "*broccoli*"?

- Knowing that Sue "*swallowed*" helps narrow down possibilities
- But, how far back do we look?

Which n-gram to use?

- example: for a vocabulary of 20,000 words
  - number of bigrams = 400 million ($20\ 000^2$)
  - number of trigrams = 8 trillion ($20\ 000^3$)
  - number of four-grams = 1.6×10^{17} ($20\ 000^4$)
- number of n-grams is exactly the number of parameters we have to learn, that is for bigrams we need to learn P(word1 word2) for any combination of word1 and word 2 from vocabulary of size V
- However, our training data has fixed size of N words, therefore in our training data here are
  - N-1 bigram samples
  - N-2 trigram samples
  - N-3 fourgram samples
- As we go from n-gram to (n+1) gram, number of parameters to learn grows a lot, but the number of training samples does not increase. Big problem!
  - For reliable estimates, the more parameters we need to learn, the more training samples we need

Unigram vs. bigram Illustration

- For reliable estimates, the more parameters we need to learn, the more training samples we need to have
- Suppose we have a text of 10,000 words. We have a reasonable amount of data to produce unigrams, that is probabilities of individual words, P(“a”),P(“to”), etc., are high and P(“zombee”), P(“gene”) are low
- However, we do not have enough data to estimate bigrams for example:
  - P(“a table”), P(“to ride”), P(“can draw”), even though these word sequences are quite likely, in a text of 10,000 words we may not have seen them
  - Need a much larger text for bigrams

Which n-gram to use? Reliability vs. Discrimination

- larger n:
  - **greater discrimination**: more information about the context of the specific instance
  - but **less reliability**:
    - Our model is too complex, that is has too many parameters
    - Cannot estimate parameters reliably from limited data (data sparseness)
      - too many chances that the history has never been seen before
      - our estimates are not reliable because we have not seen enough examples
- smaller n:
  - **less discrimination**, not enough history to predict next word very well, our model is not so good
  - but **more reliability**:
    - more instances in training data, better statistical estimates of our parameters
- Bigrams or trigrams are used in practice

Text generation with n-grams

- n-gram model trained on 40 million words from WSJ (wall street journal)
- Start with random word and generate next word according to the n-gram model
- Unigram:
  - *Months the my and issue of year foreign new exchange's September were recession exchange new endorsed a acquire to six executives.*
- Bigram:
  - *Last December through the way to preserve the Hudson corporation N.B.E.C. Taylor would seem to complete the major central planner one point five percent of U.S.E. has already old M. X. corporation of living on information such as more frequently fishing to keep her.*
- Trigram:
  - *They also point to ninety point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions.*

From [Jurafsky and Martin, 2000] , Ch. 4

Reducing number of Parameters

- with a 20 000 word vocabulary:
  - bigram needs to store 400 million parameters
  - trigram needs to store 8 trillion parameters
  - using a language model > trigram is impractical
- to reduce the number of parameters, we can:
  - do stemming (use stems instead of word types)
    - help = helps = helped
  - group words into semantic classes
    - {Monday,Tuesday,Wednesday,Thursday,Friday} = one word
  - seen once --> same as unseen
  - ...

Statistical Estimators

- How do we estimate parameters (probabilities of unigrams, bigrams, trigrams)?
  - Using statistical estimators
- Maximum Likelihood Estimation (MLE)
  - we have already seen this, has major problems due to data sparsness
- Smoothing
  - Add-one -- Laplace
  - Add-delta -- Lidstone’s & Jeffreys-Perks’ Laws (ELE)
  - Good-Turing
- Combining Estimators
  - Simple Linear Interpolation
  - General Linear Interpolation

Maximum Likelihood Estimation

- We have already seen this
- Let $C(w_1 \ldots w_n)$ be the frequency of n-gram $w_1 \ldots w_n$

$$P_{MLE}(w_n \mid w_1 \ldots w_{n-1}) = \frac{C(w_1 \ldots w_n)}{C(w_1 \ldots w_{n-1})}$$

- Has the name “Maximum Likelihood” because the parameter values it gives lead to highest probability of the **training** corpus
- However, we are interested in good performance on **testing** data

Example 1

- in a training corpus, we have 10 instances of "*come across*"
  - 8 times, followed by "*as*"
  - 1 time, followed by "*more*"
  - 1 time, followed by "*a*"
- so we have:
  - $P_{MLE}(\text{as} \mid \text{come across}) = \frac{C(\text{come across as})}{C(\text{come across})} = \frac{8}{10}$
  - $P_{MLE}(\text{more} \mid \text{come across}) = 0.1$
  - $P_{MLE}(\text{a} \mid \text{come across}) = 0.1$
  - $P_{MLE}(X \mid \text{come across}) = 0$ where $X \neq$ "*as*", "*more*", "*a*"

Example 2

P(on|eat) = .16 P(some|eat) = .06 P(British|eat) = .001 … P(I|<s>) = .25 P(I'd|<s>) = .06 …	P(want|I) = .32 P(would|I) = .29 P(don't|I) = .08 … P(to|want) = .65 P(a|want) = .5 …	P(eat|to) = .26 P(have|to) = .14 P(spend|to)= .09 … P(food|British) = .6 P(restaurant|British) = .15 …

In the table above , <s> is the beginning of the sentence

P_{MLE} (I want to eat British food) =

P(I|<s>) x P(want|I) x P(to|want) x P(eat|to) x P(British|eat) x P(food|British)

= .25 x .32 x .65 x .26 x .001 x .6

= .000008

In Practice

- product of probabilities … numerical underflow for long sentences
- so instead of multiplying the probabilities, we add the log of the probabilities
  - log(A*B*C*D)=log(A)+log(B)+log(C)+log(D)

P_{MLE} (I want to eat British food) =

P(I|<s>) x P(want|I) x P(to|want) x P(eat|to) x P(British|eat) x P(food|British)

= .25 x .32 x .65 x .26 x .001 x .6

= .000008

log[P_{MLE} (*I want to eat British food)*]

= log(P(I|<s>)) + log(P(want|I)) + log(P(to|want)) + log(P(eat|to)) + log(P(British|eat)) + log(P(food|British))

= log(.25) + log(.32) + log(.65) + log (.26) + log(.001) + log(.6)

= -11.722

Conditional probability vs probability of an n-gram

$$\text{we know that } P(w_n \mid w_1 \ldots w_{n-1}) = \frac{P(w_1 \ldots w_n)}{P(w_1 \ldots w_{n-1})}$$

$$\text{so from now on, we will simply try to estimate}: P(w_1 \ldots w_n)$$

$$P_{MLE}(w_n \mid w_1 \ldots w_{n-1}) = \frac{C(w_1 \ldots w_n)}{C(w_1 \ldots w_{n-1})} \quad \text{e.g. } P(\text{as} \mid \text{come across})$$

$$P_{MLE}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)}{N} \quad \text{e.g. } P(\text{come across as})$$

$$\text{where } N = \text{number of training instances (total number of ngram tokens)}$$

Common words in Tom Sawyer

Word	Freq.	Use
the	3332	determiner (article)
and	2972	conjunction
a	1775	determiner
to	1725	preposition, verbal infinitive marker
of	1440	preposition
was	1161	auxiliary verb
it	1027	(personal/expletive) pronoun
in	906	preposition
that	877	complementizer, demonstrative
he	877	(personal) pronoun
I	783	(personal) pronoun
his	772	(possessive) pronoun
you	686	(personal) pronoun
Tom	679	proper noun
with	642	preposition

but words in NL have an uneven distribution…

Most Words are Rare

Word Frequency	Frequency of Frequency
1	3993
2	1292
3	664
4	410
5	243
6	199
7	172
8	131
9	82
10	91
11-50	540
51-100	99
> 100	102

- most words are rare
  - 3993 (50%) word types appear only once
  - they are called **happax legomena** (*read only once*)
- but common words are **very** common
  - 100 words account for 51% of all tokens (of all text)

Problem with MLE: Data Sparseness

- Got trigram "nothing but the" in training corpus, but not trigram "and nuts sing"
- Therefore we estimate P("and nuts sing") = 0
- Any sentence which has "and nuts sing" will have probability 0
  - We want P("and nuts sing") to be small, but not 0!
- if a trigram never appears in training corpus, probability of sentence containing this trigram is 0
- MLE assigns a probability of zero to unseen events …
- probability of an n-gram involving unseen words will be zero!
- but … most words are rare
- so n-grams involving rare words are even more rare… data sparseness

Problem with MLE: data sparseness

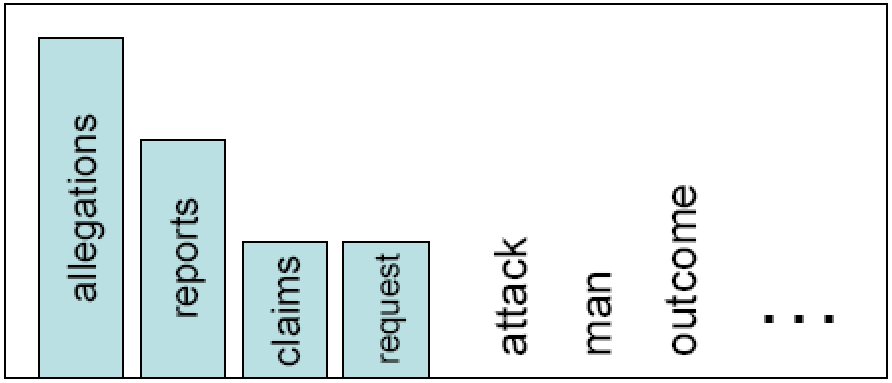
- in (Balh et al 83)
  - training with 1.5 million words
  - 23% of the trigrams from another part of the same corpus were previously unseen.
- in Shakespeare’s work
  - out of all possible bigrams, 99.96% were never used
- So MLE alone is not good enough estimator

Discounting or Smoothing

- MLE is usually unsuitable for NLP because of the sparseness of the data
- We need to allow for possibility of seeing events not seen in training
- Must use a **Discounting** or **Smoothing** technique
- Decrease the probability of previously seen events to leave a little bit of probability for previously unseen events

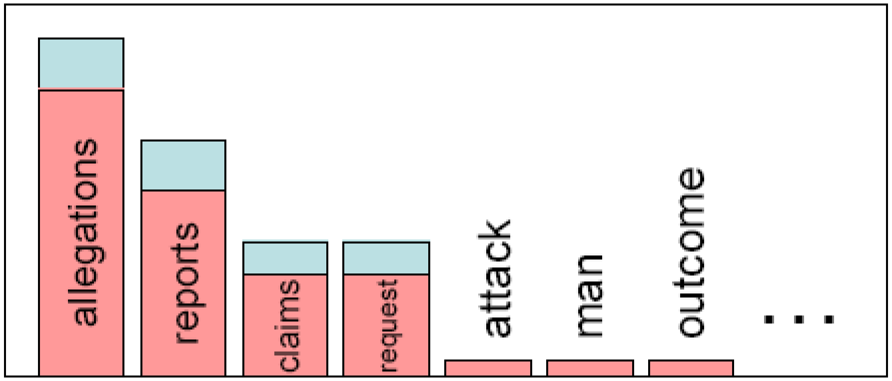
One Solution: Smoothing

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



- Solution: smoothing
  - decrease the probability of previously seen events
  - so that there is a little bit of probability mass left over for previously unseen events

Many smoothing techniques

- Add-one
- Add-delta
- Good-Turing smoothing
- Many other methods we will not study...

Add-one Smoothing (Laplace's Law)

- The idea is to give a little bit of the probability space to unseen events
- Pretend we have seen every n-gram at least once
- Intuitively we appended all possible n-grams at the end of our training data.
- Example with bigrams:

$$\underbrace{\boxed{\text{real data}}}_{\textit{N bigrams}}\underbrace{\boxed{\text{fake data}}}_{\textit{all possible bigrams}}$$

- If our training data has N ngrams, then the "new" size is N+B, where B is the number of all possible ngrams. If there are V words then
  - B= V*V for bigrams
  - B=V*V*V for trigrams
  - etc.
- Now $P_{Add1}(w_1\, w_2 \ldots w_n) = \dfrac{C(w_1\, w_2 \ldots w_n) + 1}{N + B}$

Add-One: Example

unsmoothed bigram **counts**:

2nd word (columns); 1st word (rows)

	I	want	to	eat	Chinese	food	lunch	…	Total
I	8	1087	0	13	0	0	0		N(I)=3437
want	3	0	786	0	6	8	6		N(want)=1215
to	3	0	10	860	3	0	12		N(to)=3256
eat	0	0	2	0	19	2	52		N(eat)=938
Chinese	2	0	0	0	0	120	1		N(Chinese)=213
food	19	0	17	0	0	0	0		N(food)=1506
lunch	4	0	0	0	0	1	0		N(lunch)=459
…									N=10,000

unsmoothed bigram **probabilities**:

	I	want	to	eat	Chinese	food	lunch	…	Total
I	.0008	.1087	0	.0013	0	0	0		
want	.0003	0	.0786	0	.0006	.0008	.0006		
to	.0003	0	.001	.086	.0003	0	.0012		
eat	0	0	.0002	0	.0019	.0002	.0052		
Chinese	.0002	0	0	0	0	.012	.0001		
food	.0019	0	.0017	0	0	0	0		
lunch	.0004	0	0	0	0	.0001	0		
…									N=10,000

Add-one: Example (con't)

add-one smoothed bigram counts:

	I	*want*	*to*	*eat*	*Chinese*	*food*	*lunch*	…	*Total*
I	~~8~~ 9	~~1087~~ 1088	1	14	1	1	1		~~3437~~ N(I) + V = 5053
want	~~3~~ 4	1	787	1	7	9	7		N(want) + V = 2831
to	4	1	11	861	4	1	13		N(to) + V = 4872
eat	1	1	23	1	20	3	53		N(eat) + V = 2554
Chinese	3	1	1	1	1	121	2		N(Chinese) + V = 1829
food	20	1	18	1	1	1	1		N(food) + V = 3122
lunch	5	1	1	1	1	2	1		N(lunch) + V = 2075
…									~~N= 10,000~~ $N+V^2 = 10{,}000 + (1616)^2 = 2{,}621{,}456$

add-one bigram probabilities:

	I	*want*	*to*	*eat*	*Chinese*	*food*	…
I	.0000034 (9/2621456)	.00041	.00000038	.0000053	.00000038	.00000038	
want	.0000015	.00000038	.0003	.00000038	.0000027	.0000034	
to	.0000015	.00000038	.000004	.0046	.0000015	.00000038	
eat	.00000038	.00000038	.0000088	.00000038	.0000076	.0000011	
…							

Notes on the numbers

AddOne probability of an unseen bigrams

$$= \frac{1}{N+B} = \left(\frac{1}{22{,}000{,}000 + 74{,}674{,}306{,}760}\right) = 1.33875\text{x}10^{-11}$$

Total probability mass given to unseen bigrams

= number unseen bigrams x prob of each unseen bigram

$$= 74{,}671{,}100{,}000 \text{ x } 1.33875\text{x}10^{-11}$$

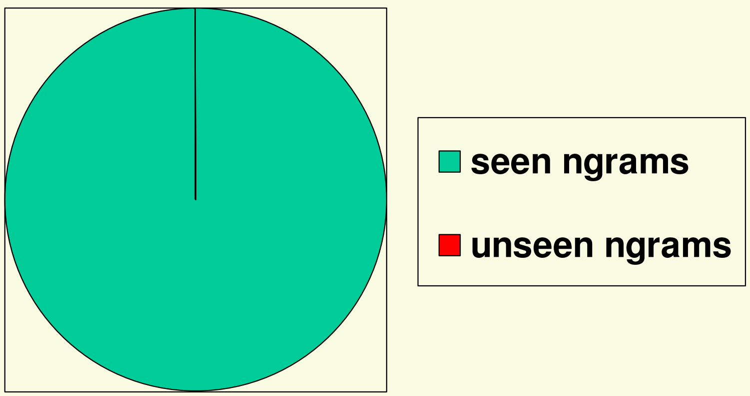
$$\approx 99.96 \text{ !!!!}$$

Problem with add-one smoothing

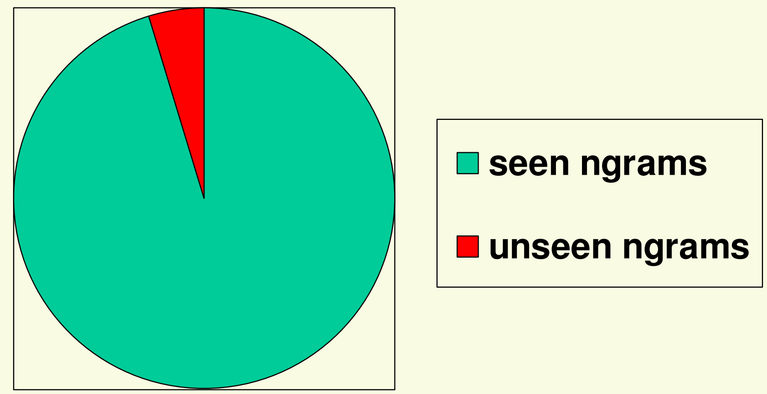
- each individual unseen n-gram is given a low probability
- but there is a huge number of unseen n-grams
  - Adding a little of probability over a huge number of unseen events gives too much probability mass to all unseen events
- Instead of giving small portion of probability to unseen events, most of the probability space is given to unseen events

Problem with add-one smoothing

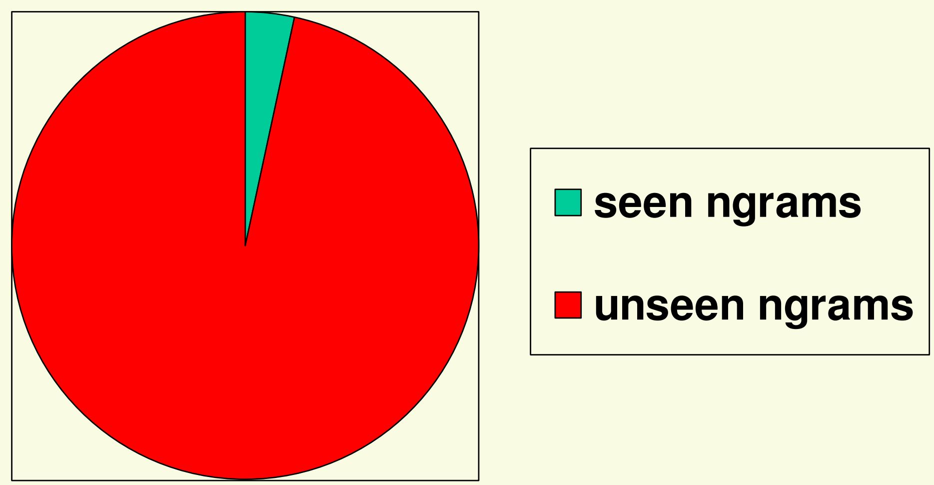
MLE



want something like this



get this with add1 smoothing



Problem with add-one smoothing

- Data from the AP from (Church and Gale, 1991)
  - Corpus of 44,000,000 bigram tokens, 22,000,000 for training
    - Vocabulary of 273,266 words, i.e. 74,674,306,760 possible bigrams
  - 74,671,100,000 bigrams were unseen
  - frequency is the number of occurrences per 22,000,000 samples
    - To get probability, divide frequency by 22,000,000
  - each unseen bigram was given a <u>frequency</u> of 0.000295

num. of times appeared in training corpus

Freq.observed in testing corpus

Add-one smoothed freq. given to testing corpus

f_{MLE}	$f_{empirical}$	$f_{add\text{-}one}$
0	0.000027	0.000295
1	0.448	0.000589
2	1.25	0.000884
3	2.24	0.001180
4	3.23	0.001470
5	4.21	0.001770

too high

too low

Add-delta smoothing (Lidstone's law)

- instead of adding 1, add some other (smaller) positive value δ

$$P_{AddD}(w_1\, w_2 \ldots w_n) = \frac{C\,(w_1\, w_2 \ldots w_n) + \delta}{N + \delta\, B}$$

- most widely used value for δ = 0.5
- if δ =0.5, Lidstone's Law is called:
  - the **Expected Likelihood Estimation** (ELE)
  - or the **Jeffreys-Perks** Law

$$P_{ELE}(w_1\, w_2 \ldots w_n) = \frac{C\,(w_1\, w_2 \ldots w_n) + 0.5}{N + 0.5\, B}$$

- better than add-one, but still not very good

Smoothing: Good Turing

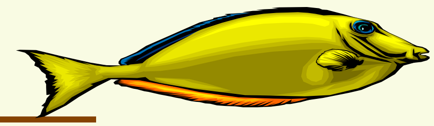
- Imagine you are fishing
  - You have bass, carp, cod, tuna, trout, salmon, eel, shark, tilapia, etc. in the sea
- You have caught 10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel
- How likely is it that next species is new?
  - roughly 3/18, since 18 fish total, 3 unique species
- How likely is it that next is tuna? Less than 2/18
  - 2 out of 18 are tuna, but we have to give some “room” to the new species that we may catch in the future
- Say that there are 20 species of fish that we have not seen yet (bass, shark, tilapia,….)
- The probability of any individual unseen species is $\frac{3}{18 \cdot 20}$
- P(shark)=P(tilapia)= $\frac{3}{18 \cdot 20}$

Smoothing: Good Turing

- How many species (n-grams) were seen once?
  - Let N_1 be the number species (n-grams) seen once
- Use it to estimate for probability of unseen species
  - Probability of new species (new n-gram) is N_1/N
- Let N_0 be the number of unseen species (unseen n-grams). Spreading around the mass equally for unseen n-grams, the probability of seeing any individual unseen species (unseen n-gram) is

$$\frac{N_1}{N \cdot N_0}$$

Smoothing: Good Turing



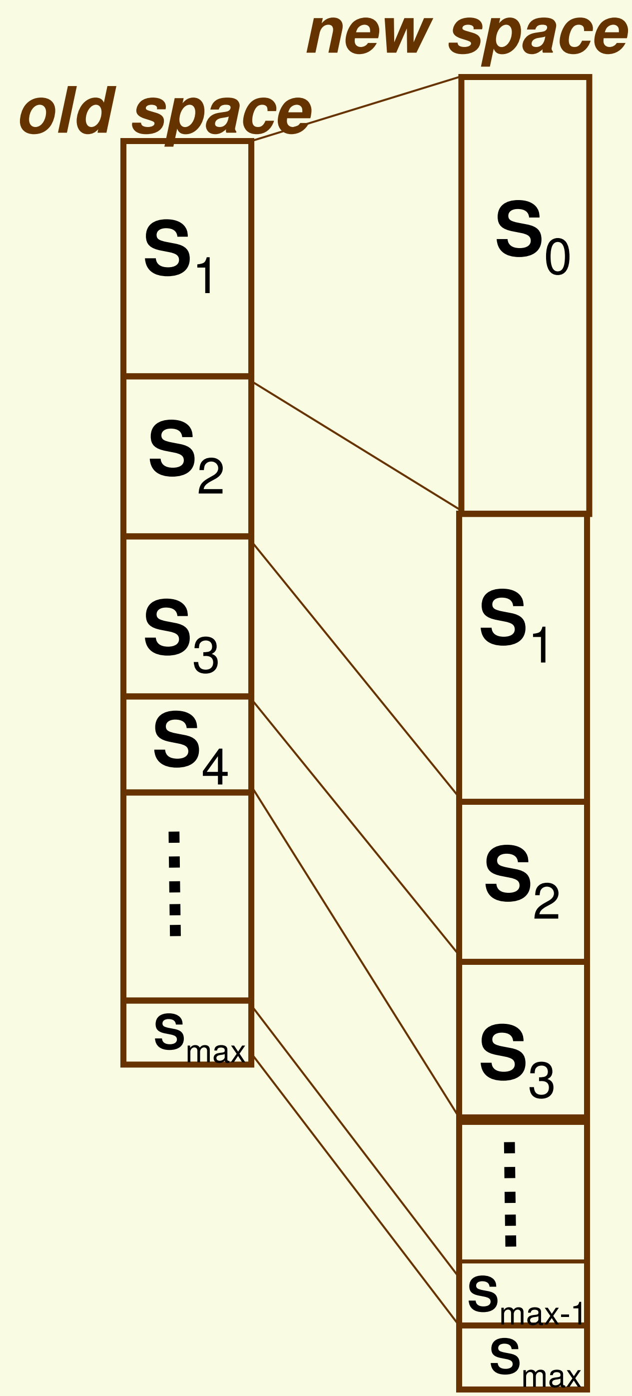
- Back to fishing: you have caught 10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel; 20 species unseen
- How likely is it that next species is new? 3/18
  - The probability of any individual unseen fish is $\frac{3}{18 \cdot 20}$
- What is the new probability of catching a trout?
  - Should be lower than 1/18th to make room for unseen fish
  - Idea:
    - if we catch another trout, trout will occur with the rate of 2
    - According to our data, that is the probability of fish with rate 2 (occurring 2 times). Tuna occurs 2 times, so probability is 2/18
    - Now spread the probability of 2/18 over all species which occurred only once – 3 species
    - The probability of catching a fish which occurred 1 time already is $\frac{2}{18 \cdot 3}$

Smoothing: Good Turing

- In general, let **r** be the rate with which an n-gram occurs in the training data
  - Rate is the same thing as count
  - Example: if training data is {“a cow”, “a train”, “a cow”, “do as”, “to go”, “let us”,”to go”}, then the rate of “a cow” is 2 and the rate of “let us” is 1
- If an n-gram occurs with rate **r**, we used to get its probability as
  - **r**/**N**, where **N** is the size of the training data
  - We need to lower all the rates to make room for unseen n-grams
- In general, the number of n-grams which occur with rate **r**+1 is smaller than the number of grams which occur with rate **r**
- Idea: take the portion of probability space occupied by n-grams which occur with rate **r**+1 and divide it among the n-grams which occur with rate **r**

Smoothing: Good Turing

- Let S_r be the n-grams that occur r times in the training data
- Proportion of probability space occupied by n-grams in S_r in the new space = proportion of probability space occupied by n-grams in S_{r+1} in the new space
  - Spread evenly among all n-grams in S_r
- Note no space left for n-grams in S_{max} , has to be fixed



Smoothing: Formula for Good Turing

- N_r be the number different n-grams that we saw in the training data exactly ***r*** times
  - Example: if training data is {“a cow”, “a train”, “a cow”, “do as”, “to go”, “let us”,”to go”}, then $N_1 = 3$ and $N_2 = 2$
  - In notation on previous slide, rN_r is the size of S_r
- Probability for any n-gram with rate r is estimated from the space occupied by n-grams with rate r+1
- Let N be the size of the training data.The probability space occupied by n-grams with rate r+1 is:

$$\frac{(r+1)N_{r+1}}{N}$$

- Spread this mass evenly among n-grams with rate r, there are N_r of them

$$\frac{(r+1)N_{r+1}}{N \cdot N_r}$$

- That is for a n-gram x that occurs r times, Good Turing estimate of probability is

$$P_{GT}(x) = (r+1)\frac{N_{r+1}}{N \cdot N_r}$$

Smoothing: Good Turing

$$P_{GT}(w_1 \ldots w_n) = \frac{1}{N} \cdot \underbrace{\frac{(r+1)N_{r+1}}{N_r}}_{r^*}$$

- Another way of looking at Good-Turing:

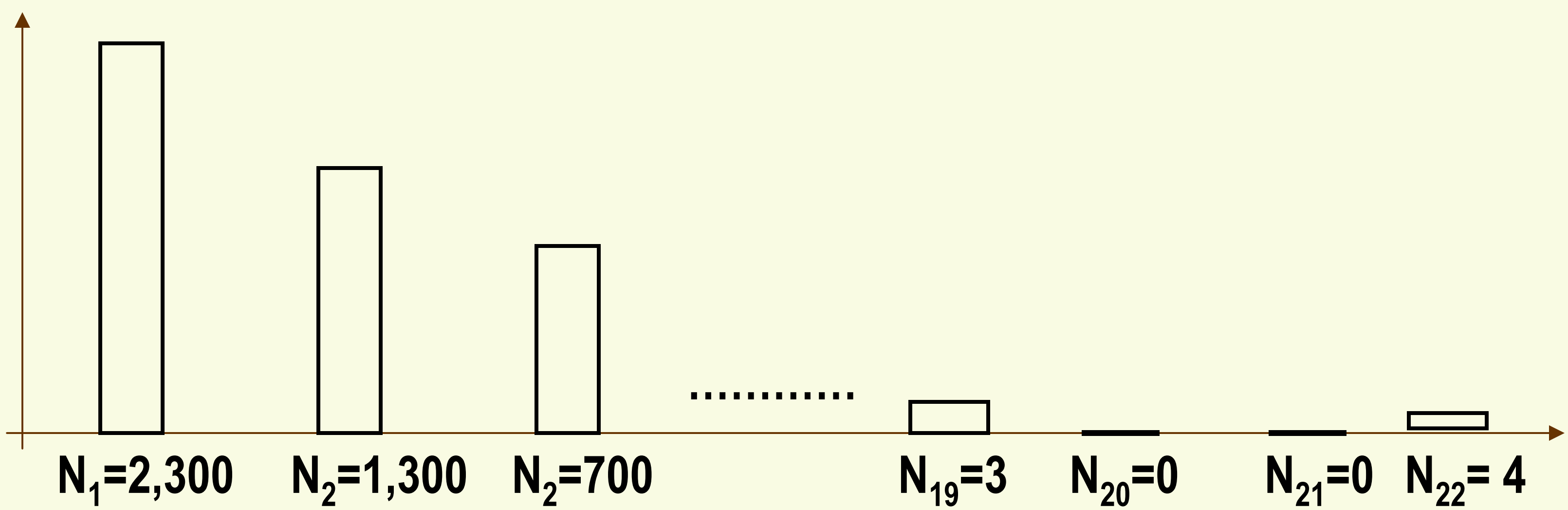
$$P_{MLE}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)}{N} = \frac{r}{N}$$

  - $P_{MLE}(w_1 \ldots w_n) = 0$ for rate $r = 0$, need to increase it
    - at the expense of decreasing the rate of observed nGrams
  - if $r = 0$, new r^* should be larger
  - if $r \neq 0$, new r^* should be smaller

- This is exactly what Good-Turing does
  - For $r = 0$, $r^* = \frac{N_1}{N_0} > r$
  - For $r > 0$, $r^* = \frac{(r+1)N_{r+1}}{N_r}$
  - most likely $r^* < r$ since usually N_{r+1} is significantly less than N_r

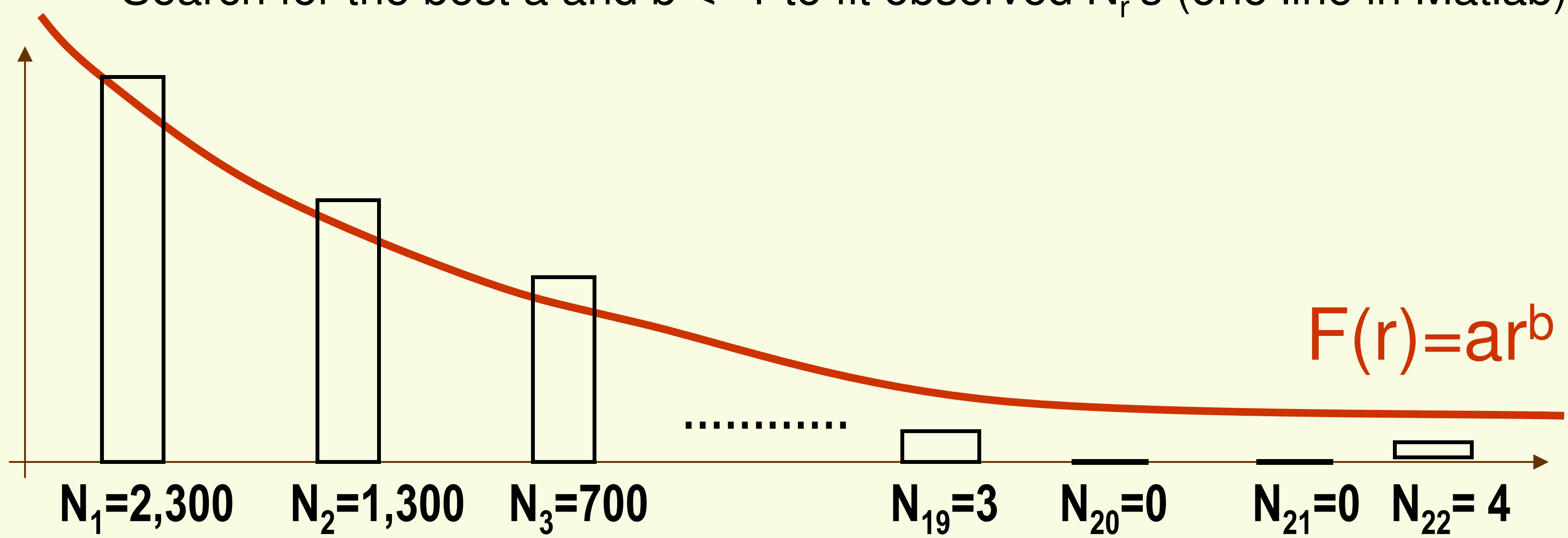
Smoothing: Fixing Good Turing

- That is for an n-gram x that occurs r times, Good Turing estimate of probability is $$P_{GT}(x) = (r+1)\frac{N_{r+1}}{N \cdot N_r}$$
- This works well except for high values of r
  - For high values of r, N_r is not reliable estimate of the number of n-grams that occur with rate r
  - In particular, for the most frequent r it completely fails since $N_{r+1}=0$
- The problem is that N_r is unreliable for high values of r



Smoothing: Fixing Good Turing

- The problem is that N_r is unreliable for high values of r
- Solution 1:
  - use P_{GT} for low values of r, say for r < 10
  - For n-grams with higher rates, use P_{MLE} which is reliable for higher values of r, that is $P_{MLE}(w_1 \ldots w_n) = C(w_1 \ldots w_n)/N$
- Solution 2:
  - Smooth out N_r's by fitting a power law function $F(r)=ar^b$ (with b < -1) and use it when N_r becomes unreliable.
    - Search for the best a and b < -1 to fit observed N_r's (one line in Matlab)



Smoothing: Fixing Good Turing

- Probabilities will not add up to 1, whether using Solution 1 or Solution 2 from the previous slide
- Have to renormalize all probabilities so that they add up to 1
  - Could renormalize all n-grams
  - Usually we renormalize only the n-grams with observed rates higher than 0
  - Suppose the total space for unseen n-grams is 1/20
  - renormalize the weight of the seen n-grams so that the total is 19/20

Good Turing vs. Add-One

$r = f_{MLE}$	$f_{empirical}$	f_{Lap}	f_{GT}
0	0.000027	0.000137	0.000027
1	0.448	0.000274	0.446
2	1.25	0.000411	1.26
3	2.24	0.000548	2.24
4	3.23	0.000685	3.24
5	4.21	0.000822	4.22
6	5.23	0.000959	5.19
7	6.21	0.00109	6.21
8	7.21	0.00123	7.24
9	8.26	0.00137	8.25

Simple Example

$$P_{GT}(n-gram\ occuring\ \ r\ \ times) = (r+1)\frac{N_{r+1}}{N \cdot N_r}$$

- Vocabulary is {a,b,c}
- Possible bigrams: {aa,ab,ba,bb,ac,bc,ca,cb,cc}
- Corpus: b a b a a c b c a c a c
  - Observed bigrams are {ba, ab, ba, aa, ac, cb, bc, ca, ac, ca, ac}
- Unobserved bigrams: bb,cc
- Observed bigram frequencies:
  - ab: 1, aa: 1,cb: 1, bc: 1, ba: 2, ca: 2, ac: 3
- N_0=2, N_1=4, N_2=2, N_3=1, N = 11
- Will use GT probabilities up to and including r = 2
- Probability estimations:
  - Use Good-Turing: P(bb)=P(cc)= (0+1)*(N_1/(N*N_0))=4/(11*2)=2/11
  - Use Good-Turing: P(ab)=P(aa)=P(cb)=P(bc)= (1+1)*(N_2/(N*N_1))=1/11
  - Use Good-Turing: P(ba)=P(ca)= (2+1)*(N_3/(N*N_2))=3/22
  - Use MLE: P(ac) = 3/11

Simple Example Continued

- Finally renormalize
- Before renormalization:
  - P'(bb)=P(cc)= 2/11
  - P'(ab)=P'(aa)=P'(cb)=P'(bc)= 1/11
  - P'(ba)=P'(ca)=3/22
  - P'(ac) = 3/11
    - I put P'(…) to indicate that the things above are not true probabilities, since they don't add up to 1
- Renormalize only the weight of seen bigrams ab,aa,cb,bc,ba,ca,ac and their total weight should be 1-[P'(bb)+P'(cc)] =7/11
  - P'(ab)+P'(aa)+P'(cb)+P'(bc)+P'(ba)+P'(ca)+P'(ac) = 10/11
  - Multiply through by constant (11/10)*(7/11)=7/10
  - New probabilities are:
    - P(bb)=P(cc)= 2/11
      - did not want to change these
    - P(ab)=P(aa)=P(cb)=P(bc)= (1/11)*(7/10)=7/110
    - P(ba)=P(ca)=(3/22)*(7/10)=21/220
    - P(ac)=(3/11)*(7/10)=21/110

Simple Example Continued

- Can also renormalize weights in a simpler manner
  - I asked you to do this for your assignment, to simplify your life!
- Before renormalization:
  - $P'(bb)=P'(cc)= 2/11 = P'_0$
  - $P'(ab)=P'(aa)=P'(cb)=P'(bc)= 1/11= P'_1$
  - $P'(ba)=P'(ca)=3/22= P'_2$
  - $P'(ac) = 3/11= P'_3$
- Simply renormalize all “probabilities” P’ to add to 1
  - (1) find their sum; (2) Divide each one by the sum
- For efficiency, you want to add them up based on the rates, since nGrams with the same rate have the same probability
  - Set S_r contain all nGrams that were observed r times, N_r is size of S_r
  - $S_0 = \{bb,cc\}$, $S_1 = \{ab,aa,cb,bc\}$, $S_2 = \{ba,ca\}$, $S_3 = \{ac\}$
  - $sum = P'_0N_0+P'_1N_1+P'_2N_2+P'_3N_3=(2/11)*2+(1/11)*4+(3/22)*2+(3/11)=14/11$
- New probabilities are:
  - $P(bb)=P(cc)= (2/11)/(14/11)=2/14 = P_0$
  - $P(ab)=P(aa)=P(cb)=P(bc)= (1/11)/(14/11)=1/14= P_1$
  - $P(ba)=P(ca)=(3/22)/(14/11)= 3/28= P_2$
  - $P(ac) = (3/11)/(14/11)=3/14= P_3$

Simple Example Continued

- Let us calculate P(abcab) using our model
- In general, when you need to use nGram approximation of $P(w_1 w_2 w_3 w_4 \ldots w_k)$
- after applying the law of conditional probability many many times you get

$$P(w_1 w_2 w_3 w_4 \ldots w_k) \approx$$

$$\approx P(w_1)P(w_2 / w_1)P(w_3 / w_1 w_2)\ldots P(w_n / w_1 w_2 \ldots w_{n-1})\ldots P(w_k / w_{k-n} w_{k-1})$$

- P(abcab) ≈ P(a) * P(b|a) * P(c|b) * P(a|c) * P(b|a)

Simple Example Continued

- probabilities are (using the first case of normalization):
  - P(bb)=P(cc)= 2/11
  - P(ab)=P(aa)=P(cb)=P(bc)= 7/110
  - P(ba)=P(ca)=21/220
  - P(ac)=21/110
- Let us calculate P(abcab) using our model
- We will need probabilities for unigrams a,b,c, which we can compute using MLE estimator:
  - P(a) = 5/12, P(b) = 3/12, P(c)=4/12
  - since a occurs 5 times, b occurs 3 times, and c occurs 4 times in our corpus consisting of 12 unigrams
- P(abcab) ≈ P(a) * P(b|a) * P(c|b) * P(a|c) * P(b|a) =

$$= P(a)\frac{P(ab)}{P(a)}\frac{P(bc)}{P(b)}\frac{P(ca)}{P(c)}\frac{P(ab)}{P(a)} =$$

$$= \frac{5}{12} \cdot \frac{7}{110(5/12)} \cdot \frac{7}{110(3/12)} \cdot \frac{21}{220(4/12)} \cdot \frac{7}{110(5/12)}$$

Combining Estimators

- Assume we have never seen the bigrams
    - *journal of* $P_{unsmoothed}(of \mid journal) = 0$
    - *journal from* $P_{unsmoothed}(from \mid journal) = 0$
    - *journal never* $P_{unsmoothed}(never \mid journal) = 0$
  - all models so far will give the same probability to all 3 bigrams
- but intuitively, "*journal of*" is more probable because...
  - *"of"* is more frequent than "*from*" & "*never*"
  - unigram probability P(*of*) > P(*from*) > P(*never*)

Combining Estimators (con't)

- observation:
  - unigram model suffers less from data sparseness than bigram model
  - bigram model suffers less from data sparseness than trigram model
  - …
- so use a lower model to estimate probability of unseen n-grams
- if we have several models of how the history predicts what comes next, we can combine them in the hope of producing an even better model

Simple Linear Interpolation

- Solve the sparseness in a trigram model by mixing with bigram and unigram models
- Also called:
  - linear interpolation
  - finite mixture models
  - deleted interpolation
- Combine linearly

$$P_{li}(w_n|w_{n-2},w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2},w_{n-2})$$

  - where $0 \leq \lambda_i \leq 1$ and $\Sigma_i \lambda_i = 1$

Other applications of LM

- Author / Language identification
- hypothesis: texts that resemble each other (same author, same language) share similar characteristics
  - In English character sequence "*ing*" is more probable than in French

- Training phase:
  - pre-classified documents (known language/author)
  - construct the language model for each document class separately
- Testing phase:
  - evaluation of unknown text (comparison with language model)

Example: Language identification

- bigram of characters
  - characters = 26 letters (case insensitive)
  - possible variations: case sensitivity, punctuation, beginning/end of sentence marker, …

Example: Language identification

1. **Train an language model for English:**

	A	B	C	D	…	Y	Z
A	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
B	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
C	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
D	0.0042	0.0014	0.0014	0.0014	…	0.0014	0.0014
E	0.0097	0.0014	0.0014	0.0014	…	0.0014	0.0014
…	…	…	…	…	…	…	0.0014
Y	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
Z	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014

2. **Train a language model for French**
3. **Evaluate probability of a sentence with LM-English & LM-French**
4. **Highest probability -->language of sentence**

Spam/Ham Classification

- Can do the same thing for ham/spam emails
- Construct character based model for ham/spam separately
- For new email, evaluate its character sequence using spam character model and ham character model
- Highest probability model wins
- This is approach was the best one on our assignment 1 data, as presented in a workshop where the data comes from