Language Models

Instructor: Rada Mihalcea

Note: some of the material in this slide set was adapted from an NLP course taught by Bonnie Dorr at Univ. of Maryland

Language Models

A language model

- an abstract representation of a (natural) language phenomenon.
- an approximation to real language

Statistical models

- predictive
- explicative

Claim

A useful part of the knowledge needed to allow letter/word predictions can be captured using simple statistical techniques.

Compute:

- probability of a sequence
- likelihood of letters/words co-occurring

Why would we want to do this?

- Rank the likelihood of sequences containing various alternative hypotheses
- Assess the likelihood of a hypothesis

Outline

- **Applications of language models**
- Approximating natural language
- The chain rule
- Learning N-gram models
- Smoothing for language models
- Distribution of words in language: Zipf's law and Heaps law

Why is This Useful?

Speech recognition
Handwriting recognition
Spelling correction
Machine translation systems
Optical character recognizers

Handwriting Recognition

Assume a note is given to a bank teller, which the teller reads as I have a gub. (cf. Woody Allen)

NLP to the rescue ….

- gub is not a word
- gun, gum, Gus, and gull are words, but gun has a higher probability in the context of a bank

Real Word Spelling Errors

They are leaving in about fifteen *minuets* to go to her house.

The study was conducted mainly *be* John Black.

Hopefully, all *with* continue smoothly in my absence.

Can they *lave* him my messages?

I need to *notified* the bank of….

He is trying to *fine* out.

For Spell Checkers

Collect list of commonly substituted words

piece/peace, whether/weather, their/there ...

Example:

"On Tuesday, the whether ..."

"On Tuesday, the weather ..."

Other Applications

- Machine translation
- Text summarization
- Optical character recognition

Outline

- Applications of language models
- **Approximating natural language**
- The chain rule
- Learning N-gram models
- Smoothing for language models
- Distribution of words in language: Zipf's law and Heaps law

Letter-based Language Models

Shannon's Game

Guess the next letter:

Letter-based Language Models

Shannon's Game

Guess the next letter:

W

Letter-based Language Models

Shannon's Game

Guess the next letter:

Wh

Letter-based Language Models

Shannon's Game

Guess the next letter:

Wha

Letter-based Language Models

Shannon's Game

Guess the next letter:

What

Letter-based Language Models

Shannon's Game

Guess the next letter:

What d

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

What

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

What do

Letter-based Language Models

Shannon’s Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

What do you

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

What do you think

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

What do you think the

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

What do you think the next

Letter-based Language Models

Shannon's Game

Guess the next letter:

What do you think the next letter is?

Guess the next word:

What do you think the next word is?

Approximating Natural Language Words

zero-order approximation: letter sequences are independent of each other and all equally probable:

xfoml rxkhrjffjuj zlpwcwkcy ffjeyvkcqsghyd

Approximating Natural Language Words

first-order approximation: letters are independent, but occur with the frequencies of English text:

ocro hli rgwr nmielwis eu ll nbnesebya th eei alhenhtppa oobttva nah

Approximating Natural Language Words

second-order approximation: the probability that a letter appears depends on the previous letter

on ie antsoutinys are t inctore st bes deamy achin d ilonasive tucoowe at teasonare fuzo tizin andy tobe seace ctisbe

Approximating Natural Language Words

third-order approximation: the probability that a certain letter appears depends on the two previous letters

in no ist lat whey cratict froure birs grocid pondenome of demonstures of the reptagin is regoactiona of cre

Approximating Natural Language Words

Higher frequency trigrams for different languages:

English:	THE, ING, ENT, ION
German:	EIN, ICH, DEN, DER
French:	ENT, QUE, LES, ION
Italian:	CHE, ERE, ZIO, DEL
Spanish:	QUE, EST, ARA, ADO

Language Syllabic Similarity

Anca Dinu, Liviu Dinu

Languages within the same family are more similar among them than with other languages

How similar (sounding) are languages within the same family?

Syllabic based similarity

Syllable Ranks

Gather the most frequent words in each language in the family;

Syllabify words;

Rank syllables;

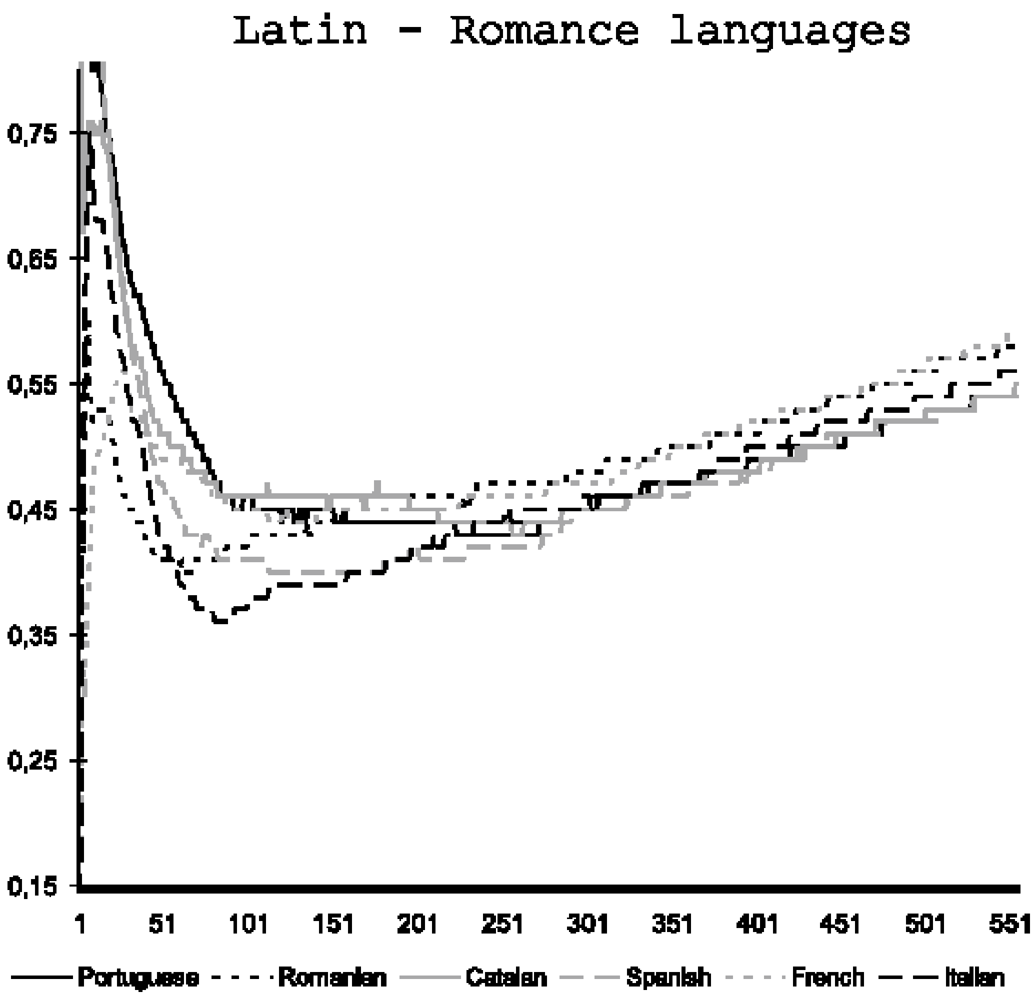
Compute language similarity based on syllable rankings;

Example Analysis: the Romance Family

Syllables in Romance languages

Language	The percentage covered by the first ··· syllables						No. syllables	
	100	200	300	400	500	561	type	token
Latin	72%	86%	92%	95%	98%	100%	561	3922
Romanian	63%	74%	80%	84%	87%	90%	1243	6591
Italian	75%	85%	91%	94%	96%	97%	803	7937
Portuguese	69%	84%	91%	95%	97%	98%	693	6152
Spanish	73%	87%	93%	96%	98%	99%	672	7477
Catalan	62%	77%	84%	88%	92%	93%	967	5624
French	48%	61%	67%	72%	76%	78%	1738	5691

Latin-Romance Languages Similarity



servus

servus

ciao

Outline

- Applications of language models
- Approximating natural language
- **The chain rule**
- Learning N-gram models
- Smoothing for language models
- Distribution of words in language: Zipf's law and Heaps law

Terminology

Sentence: unit of written language
Utterance: unit of spoken language
Word Form: the inflected form that appears in the corpus
Lemma: lexical forms having the same stem, part of speech, and word sense
Types (V): number of distinct words that might appear in a corpus (vocabulary size)
Tokens (N_T): total number of words in a corpus
Types seen so far (T): number of distinct words seen so far in corpus (smaller than V and N_T)

Word-based Language Models

A model that enables one to compute the probability, or likelihood, of a sentence S, P(S).

Simple: Every word follows every other word w/ equal probability (0-gram)

- Assume |V| is the size of the vocabulary V
- Likelihood of sentence S of length n is = $1/|V| \times 1/|V| \ldots \times 1/|V|$
- If English has 100,000 words, probability of each next word is 1/100000 = .00001

Word Prediction: Simple vs. Smart

- Smarter: probability of each next word is related to word frequency (unigram)
  - Likelihood of sentence S = $P(w_1) \times P(w_2) \times \ldots \times P(w_n)$
  - Assumes probability of each word is independent of probabilities of other words.

- Even smarter: Look at probability *given* previous words (N-gram)
  - Likelihood of sentence S = $P(w_1) \times P(w_2|w_1) \times \ldots \times P(w_n|w_{n-1})$
  - Assumes probability of each word is dependent on probabilities of other words.

Chain Rule

Conditional Probability

$P(A_1,A_2) = P(A_1) \cdot P(A_2|A_1)$

The Chain Rule generalizes to multiple events

$P(A_1, \ldots, A_n) = P(A_1)\, P(A_2|A_1)\, P(A_3|A_1,A_2) \ldots P(A_n|A_1 \ldots A_{n-1})$

Examples:

P(the dog) = P(the) P(dog | the)

P(the dog bites) = P(the) P(dog | the) P(bites| the dog)

Relative Frequencies and Conditional Probabilities

- Relative word frequencies are better than equal probabilities for all words
  - In a corpus with 10K word types, each word would have P(w) = 1/10K
  - Does not match our intuitions that different words are more likely to occur (e.g. the)
- Conditional probability more useful than individual relative word frequencies
  - dog may be relatively rare in a corpus
  - But if we see barking, P(dog|barking) may be very large

For a Word String

In general, the probability of a complete string of words $w_1^n = w_1 \ldots w_n$ is

$$P(w_1^n)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1..w_2)\ldots P(w_n|w_1\ldots w_{n-1})$$

$$= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

But this approach to determining the probability of a word sequence is not very helpful in general – gets to be computationally very expensive

Markov Assumption

How do we compute $P(w_n|w_1^{n-1})$?

- Trick: Instead of P(rabbit|I saw a), we use P(rabbit|a).
  - This lets us collect statistics in practice
  - A bigram model: P(the barking dog) =
    - P(the|<start>)P(barking|the)P(dog|barking)

Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past

- Specifically, for N=2 (bigram):
  - $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}); w_0 = \text{<start>}$

Order of a Markov model: length of prior context

- bigram is first order, trigram is second order, …

Counting Words in Corpora

What is a word?

- e.g., are cat and cats the same word?
- September and Sept?
- zero and oh?
- Is seventy-two one word or two? AT&T?
- Punctuation?

How many words are there in English?

Where do we find the things to count?

Outline

- Applications of language models
- Approximating natural language
- The chain rule
- **Learning N-gram models**
- Smoothing for language models
- Distribution of words in language: Zipf's law and Heaps law

Simple N-Grams

An N-gram model uses the previous N-1 words to predict the next one:

$P(w_n \mid w_{n-N+1}\, w_{n-N+2 \ldots}\, w_{n-1})$

unigrams: P(dog)

bigrams: P(dog | big)

trigrams: P(dog | the big)

quadrigrams: P(dog | chasing the big)

Using N-Grams

Recall that

N-gram: $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$

Bigram: $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$

For a bigram grammar

P(sentence) can be approximated by multiplying all the bigram probabilities in the sequence

Example:

P(I want to eat Chinese food) =

P(I | <start>) P(want | I) P(to | want) P(eat | to)

P(Chinese | eat) P(food | Chinese)

A Bigram Grammar Fragment

Eat on	.16	Eat Thai	.03
Eat some	.06	Eat breakfast	.03
Eat lunch	.06	Eat in	.02
Eat dinner	.05	Eat Chinese	.02
Eat at	.04	Eat Mexican	.02
Eat a	.04	Eat tomorrow	.01
Eat Indian	.04	Eat dessert	.007
Eat today	.03	Eat British	.001

Additional Grammar

<start> I	.25	Want some	.04
<start> I'd	.06	Want Thai	.01
<start> Tell	.04	To eat	.26
<start> I'm	.02	To have	.14
I want	.32	To spend	.09
I would	.29	To be	.02
I don't	.08	British food	.60
I have	.04	British restaurant	.15
Want to	.65	British cuisine	.01
Want a	.05	British lunch	.01

Computing Sentence Probability

P(I want to eat British food) = P(I|<start>) P(want|I) P(to|want) P(eat|to) P(British|eat) P(food|British) = .25×.32×.65×.26×.001×.60 = .000080

vs.

P(I want to eat Chinese food) = .00015

Probabilities seem to capture "syntactic" facts, "world knowledge"

- eat is often followed by a NP
- British food is not too popular

N-gram models can be trained by counting and normalization

N-grams Issues

Sparse data
- Not all N-grams found in training data, need smoothing

Change of domain
- Train on WSJ, attempt to identify Shakespeare – won't work

N-grams more reliable than (N-1)-grams
- But even more sparse
  - Generating Shakespeare sentences with random unigrams...
    - Every enter now severally so, let
  - With bigrams...
    - What means, sir. I confess she? then all sorts, he is trim, captain.
  - Trigrams
    - Sweet prince, Falstaff shall die.

N-grams Issues

Determine reliable sentence probability estimates

- should have smoothing capabilities (avoid the zero-counts)
- apply back-off strategies: if N-grams are not possible, back-off to (N-1) grams

P("And nothing but the truth") ≈ 0.001

P("And nuts sing on the roof") ≈ 0

Bigram Counts

	I	Want	To	Eat	Chinese	Food	lunch
I	8	1087	0	13	0	0	0
Want	3	0	786	0	6	8	6
To	3	0	10	860	3	0	12
Eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
Food	19	0	17	0	0	0	0
Lunch	4	0	0	0	0	1	0

Bigram Probabilities: Use Unigram Count

Normalization: divide bigram count by unigram count of first word.

I	Want	To	Eat	Chinese	Food	Lunch
3437	1215	3256	938	213	1506	459

Computing the probability of I I

$P(I|I) = C(I\ I)/C(I) = 8 / 3437 = .0023$

A bigram grammar is an VxV matrix of probabilities, where V is the vocabulary size

Learning a Bigram Grammar

The formula

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

is used for bigram "parameter estimation"

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Training and Testing

Probabilities come from a training corpus, which is used to design the model.

- overly narrow corpus: probabilities don't generalize
- overly general corpus: probabilities don't reflect task or domain

A separate test corpus is used to evaluate the model, typically using standard metrics

- held out test set
- cross validation
- evaluation differences should be statistically significant

Outline

- Applications of language models
- Approximating natural language
- The chain rule
- Learning N-gram models
- **Smoothing for language models**
- Distribution of words in language: Zipf's law and Heaps law

Smoothing Techniques

Every N-gram training matrix is sparse, even for very large corpora (Zipf's law)

Solution: estimate the likelihood of unseen N-grams

Add-one Smoothing

Add 1 to every N-gram count

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

$$P(w_n|w_{n-1}) = [C(w_{n-1}w_n) + 1] / [C(w_{n-1}) + V]$$

Add-one Smoothed Bigrams

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

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

$$P'(w_n|w_{n-1}) = [C(w_{n-1}w_n)+1]/[C(w_{n-1})+V]$$

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1

	I	want	to	eat	Chinese	food	lunch
I	.0018	.22	.00020	.0028	.00020	.00020	.00020
want	.0014	.00035	.28	.00035	.0025	.0032	.0025
to	.00082	.00021	.0023	.18	.00082	.00021	.0027
eat	.00039	.00039	.0012	.00039	.0078	.0012	.021
Chinese	.0016	.00055	.00055	.00055	.00055	.066	.0011
food	.0064	.00032	.0058	.00032	.00032	.00032	.00032
lunch	.0024	.00048	.00048	.00048	.00048	.00096	.00048

Other Smoothing Methods: Good-Turing

Imagine you are fishing

You have caught 10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel.

How likely is it that next species is new? 3/18

How likely is it that next is tuna? Less than 2/18

Smoothing: Good Turing

How many species (words) were seen once? Estimate for how many are unseen.

All other estimates are adjusted (down) to give probabilities for unseen

$$p_0 = \frac{n_1}{N}$$

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

Smoothing: Good Turing Example

10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel.

How likely is new data (p_0).

Let n_1 be number occurring once (3), N be total (18). $p_0=3/18$

How likely is eel? 1^*

$n_1 = 3$, $n_2 = 1$

$1^* = 2 \times 1/3 = 2/3$

$P(eel) = 1^* / N = (2/3)/18 = 1/27$

Back-off Methods

Notice that:

- N-grams are more precise than (N-1)grams (remember the Shakespeare example)
- But also, N-grams are more sparse than (N-1) grams

How to combine things?

- Attempt N-grams and back-off to (N-1) if counts are not available
- E.g. attempt prediction using 4-grams, and back-off to trigrams (or bigrams, or unigrams) if counts are not available

Outline

- Applications of language models
- Approximating natural language
- The chain rule
- Learning N-gram models
- Smoothing for language models
- **Distribution of words in language: Zipf's law and Heaps law**

Text properties (formalized)

Sample word frequency data

Frequent Word	Number of Occurrences	Percentage of Total
the	7,398,934	5.9
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1.0
The	1,144,860	0.9
that	1,066,503	0.8
said	1,027,713	0.8

Frequencies from 336,310 documents in the 1GB TREC Volume 3 Corpus
125,720,891 total word occurrences; 508,209 unique words

Zipf's Law

Rank (r): The numerical position of a word in a list sorted by decreasing frequency (f).

Zipf (1949) "discovered" that:

$$f \cdot r = k \quad (\text{for constant } k)$$

If probability of word of rank r is p_r and N is the total number of word occurrences:

$$p_r = \frac{f}{N} = \frac{A}{r} \quad \text{for corpus indp. const. } A \approx 0.1$$

Zipf curve

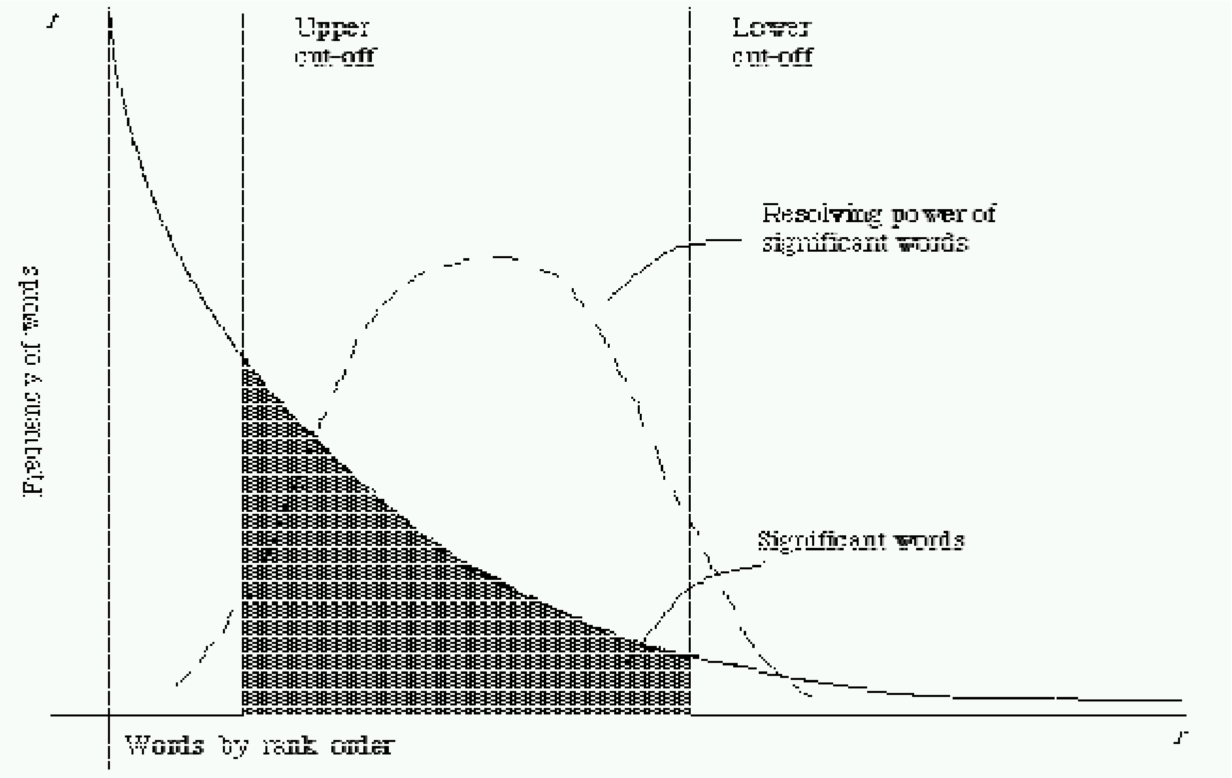


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adaped from Schultz[44] page 120)

Predicting Occurrence Frequencies

By Zipf, a word appearing n times has rank $r_n = AN/n$

If several words may occur n times, assume rank r_n applies to the last of these.

Therefore, r_n words occur n or more times and r_{n+1} words occur $n+1$ or more times.

So, the number of words appearing **exactly** n times is:

$$I_n = r_n - r_{n+1} = \frac{AN}{n} - \frac{AN}{n+1} = \frac{AN}{n(n+1)}$$

Fraction of words with frequency n is:

$$\frac{I_n}{D} = \frac{1}{n(n+1)}$$

Fraction of words appearing only once is therefore ½.

Zipf's Law Impact on Language Analysis

Good News: Stopwords will account for a large fraction of text so eliminating them greatly reduces size of vocabulary in a text

Bad News: For most words, gathering sufficient data for meaningful statistical analysis (e.g. for correlation analysis for query expansion) is difficult since they are extremely rare.

Vocabulary Growth

How does the size of the overall vocabulary (number of unique words) grow with the size of the corpus?

This determines how the size of the inverted index will scale with the size of the corpus.

Vocabulary not really upper-bounded due to proper names, typos, etc.

Heaps' Law

If V is the size of the vocabulary and the n is the length of the corpus in words:

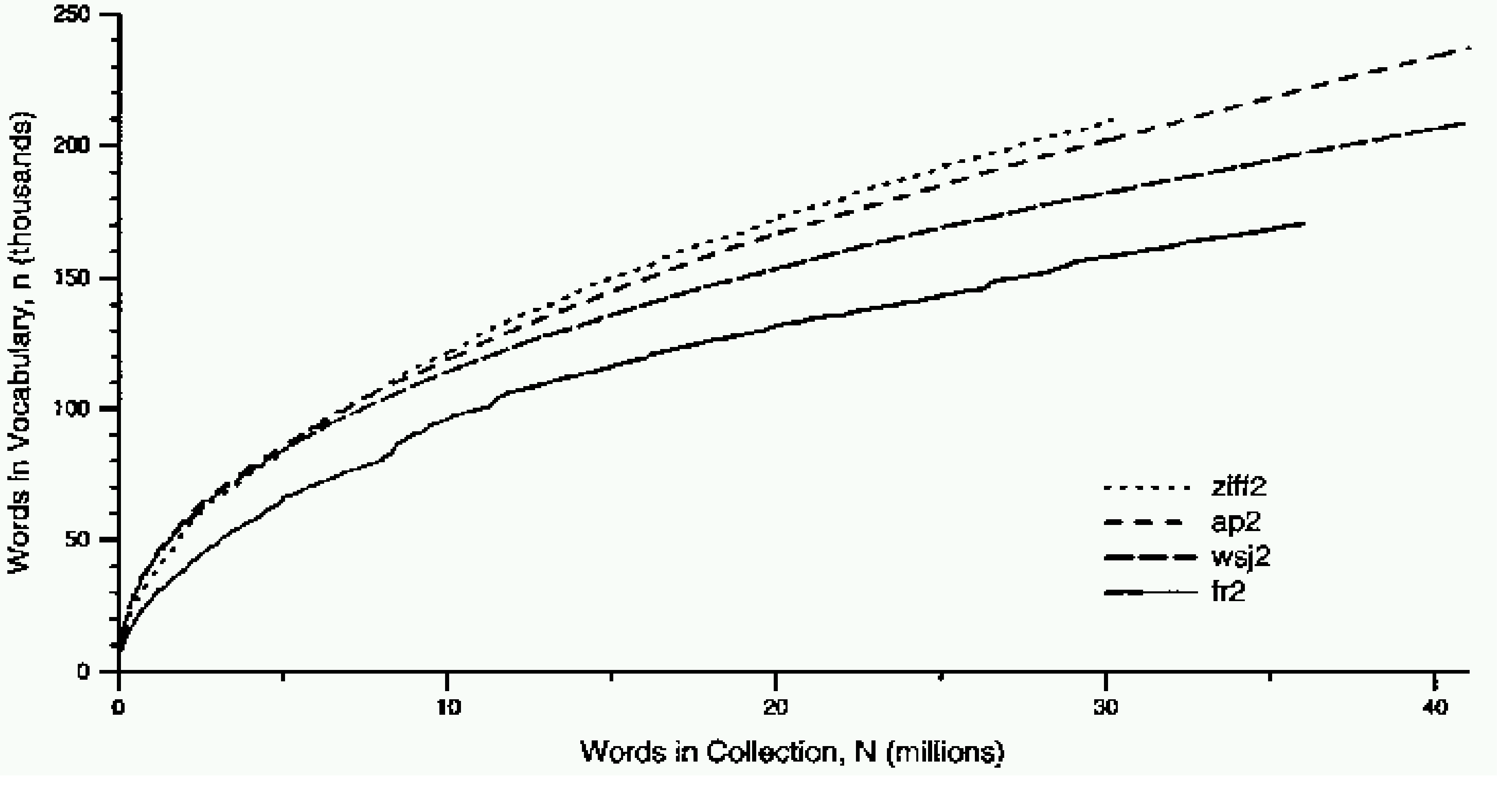
$$V = Kn^{\beta} \quad \text{with constants } K,\ 0 < \beta < 1$$

Typical constants:

$K \approx 10–100$

$\beta \approx 0.4–0.6$ (approx. square-root)

Heaps' Law Data



Letter-based models – do WE need them?… (a discovery)

Aoccdrnig to rscheearch at an Elingsh uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, olny taht the frist and lsat ltteres are at the rghit pcleas. The rset can be a toatl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae we do not raed ervey lteter by ilstef, but the wrod as a wlohe.