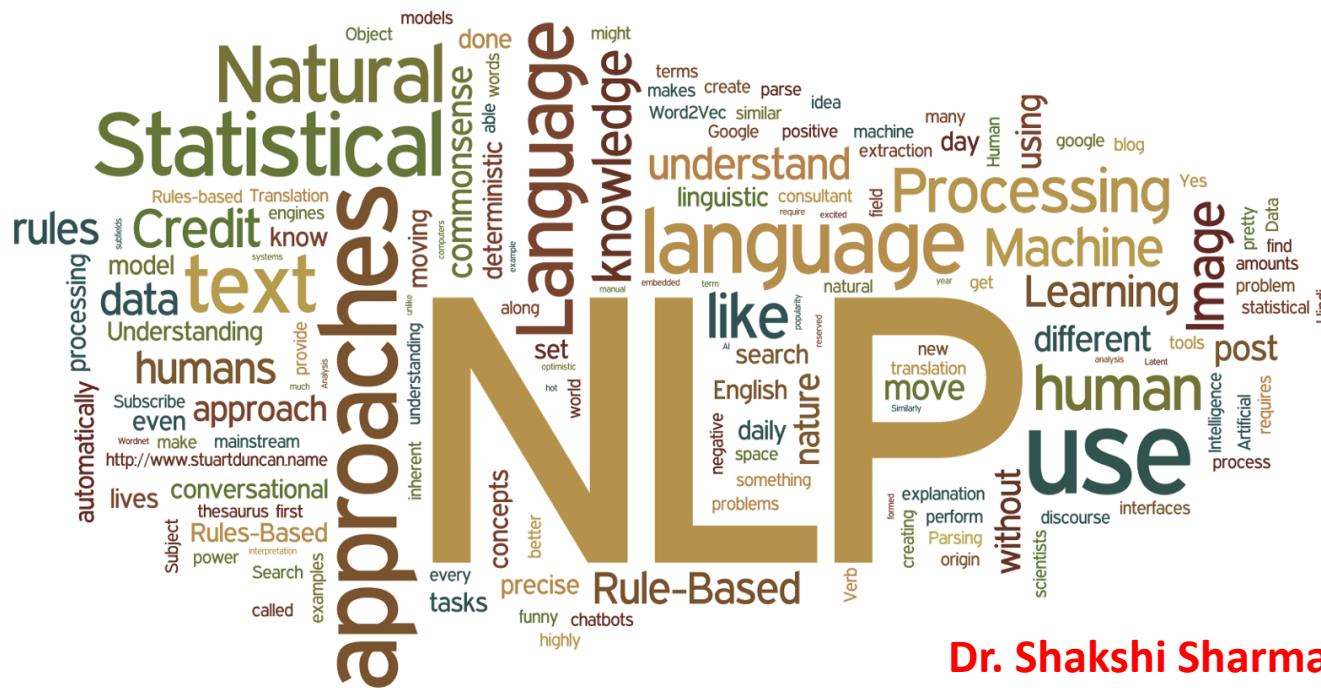


Natural Language Processing



Dr. Shakshi Sharma

Assistant Professor

School of Artificial Intelligence (SoAI), Bennett University

PhD | Junior Researcher | Estonia (Europe)

Research Intern | University of Sheffield, United Kingdom (UK)

Website: <https://sites.google.com/view/shakshi-sharma/home>

N-grams

An N-gram means a sequence of N words.

It is about predicting the n-th word from `n-1' words

So for example, “Medium blog” is a 2-gram (a **bigram**)

- “A Medium blog post” is a **4-gram**
- “Write on Medium” is a 3-gram (**trigram**).

We still have to look at the probability used with n-grams, which is quite interesting.

Why N-gram though?

Before we move on to the probability stuff, let's answer this question first.

- Why is it that **we need to learn n-gram and the related probability?**
- In Natural Language Processing, n-grams are used for a variety of things.

Some **examples** include

- **Auto completion** of sentences (such as the one we see in Gmail these days)
- **Auto spell check** (yes, we can do that as well),
- We can check for **grammar** in a given sentence.

We'll see some examples of this later when we talk about assigning probabilities to n-grams.

N-gram Probabilities

Let's take the example of **a sentence completion** system.

- This system suggests words which could be used next in a given sentence.
- Suppose I give the system the sentence “**Thank you so much for your**” and expect the system to **predict** what the next word will be.
- Now you and me both know that the next word is “**help**” with a very high probability.

But how will the system know that?

N-gram Probabilities

- One important thing to note here is that, as for any other artificial intelligence or machine learning model, we need **to train the model with a huge corpus of data**.
- Once we do that, the system, or the NLP model will have a pretty good idea of the **“probability” of the occurrence of a word after a certain word**.
- So, assuming that we have **trained** our model with a huge corpus of data, we’ll assume that the model gave us the correct answer.
- When we’re building an NLP model for predicting words in a sentence, the probability of the occurrence of a word in a sequence of words is what matters. And **how do we measure that?**

N-gram Probabilities

- Let's say we're working with a **bigram** model here, and we have the following sentences as the training **corpus**:

- Thank you so much for your help.
- I really appreciate your help.
- Excuse me, do you know what time it is?
- I'm really sorry for not inviting you.
- I really like your watch.

N-gram Probabilities

- Let's suppose that after training our model with this data, I want to write the sentence "**I really like this garden.**"
- Now this is a bigram model, the model will learn the occurrence of **every two words**, to determine the probability of a word occurring after a certain word.

For example, from the 2nd, 4th, and the 5th sentence in the example above, we know that after the word "really" we can see either the **word "appreciate", "sorry", or the word "like"** occurs.

So, the **model will calculate the probability** of each of these sequences.



Probabilistic Language Models

- Today's goal: assign a probability to a sentence
 - Machine Translation:
 - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
 - Spell Correction
 - The office is about fifteen **minuets** from my house
 - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
 - Speech Recognition
 - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
 - + Summarization, question-answering, etc., etc.!!

Why?



Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

- A model that computes either of these:

$P(W)$ or $P(w_n | w_1, w_2 \dots w_{n-1})$ is called a **language model**.

- Better: **the grammar** But **language model** or **LM** is standard

Dan Jurafsky



How to compute $P(W)$

- How to compute this joint probability:
 - $P(\text{its, water, is, so, transparent, that})$
- Intuition: let's rely on the Chain Rule of Probability



Reminder: The Chain Rule

- Recall the definition of conditional probabilities

Rewriting:

- More variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

- The Chain Rule in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\dots P(x_n|x_1, \dots, x_{n-1})$$



The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1 w_2 \dots w_n) = \prod_i P(w_i | w_1 w_2 \dots w_{i-1})$$

$P(\text{"its water is so transparent"}) =$

$P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water})$

$\times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so})$



How to estimate these probabilities

- Could we just count and divide?

$P(\text{the} \mid \text{its water is so transparent that}) =$

$$\frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Dan Jurafsky



Markov Assumption

- Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$



Andrei Markov



Markov Assumption

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$$

- In other words, we approximate each component in the product

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-k} \dots w_{i-1})$$



Simplest case: Unigram model

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the



Bigram model

- Condition on the previous word:

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

texaco, rose, one, in, this, issue, is, pursuing, growth, in,
a, boiler, house, said, mr., gurria, mexico, 's, motion,
control, proposal, without, permission, from, five, hundred,
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november



N-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
 - because language has **long-distance dependencies**:
- “The computer which I had just put into the machine room on the fifth floor crashed.”
- But we can often get away with N-gram models



Estimating bigram probabilities

- The Maximum Likelihood Estimate

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$



An example

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

< s > I am Sam < /s >
< s > Sam I am < /s >
< s > I do not like green eggs and ham < /s >

$$P(I | < s >) = \frac{2}{3} = .67$$

$$P(< /s > | Sam) = \frac{1}{2} = 0.5$$

$$P(Sam | < s >) = \frac{1}{3} = .33$$

$$P(Sam | am) = \frac{1}{2} = .5$$

$$P(am | I) = \frac{2}{3} = .67$$

$$P(do | I) = \frac{1}{3} = .33$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

Here are a few other useful probabilities:

$$P(\text{i} | \langle s \rangle) = 0.25$$

$$P(\text{food} | \text{english}) = 0.5$$

$$P(\text{english} | \text{want}) = 0.0011$$

$$P(\langle /s \rangle | \text{food}) = 0.68$$

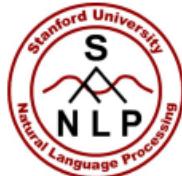
Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

$$\begin{aligned} P(<\text{s}> \text{ i want english food } </\text{s}>) \\ = P(\text{i } | <\text{s}>)P(\text{want } | \text{i})P(\text{english } | \text{want}) \\ \quad P(\text{food } | \text{english})P(</\text{s}> | \text{food}) \\ = 0.25 \times 0.33 \times 0.0011 \times 0.5 \times 0.68 \\ = 0.000031 \end{aligned}$$

- Compute the probability of i want chinese food?

Check:

$P(\text{compute the probability of i want chinese food})$ **is greater than or less than** $P(\text{compute the probability of i want English food})$



Raw bigram counts

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0



Raw bigram probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram estimates of sentence probability

$$P(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

$P(< s > i \text{ want chinese food } </ s >) =$

$P(i | < s >)$

$\times P(\text{want} | i)$

$\times P(\text{chinese} | \text{want})$

$\times P(\text{food} | \text{chinese})$

$\times P(</ s > | \text{food})$

$= \dots$

$$P(w_1, w_2, \dots, w_n) \approx \prod_i P(w_i | w_{i-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0



Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
 - Assign higher probability to “real” or “frequently observed” sentences
 - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
 - A **test set** is an unseen dataset that is different from our training set, totally unused.
 - An **evaluation metric** tells us how well our model does on the test set.



Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
 - Time-consuming; can take days or weeks
- So
 - Sometimes use **intrinsic** evaluation: **perplexity**
 - Bad approximation
 - unless the test data looks **just** like the training data
 - So **generally only useful in pilot experiments**
 - But is helpful to think about.



Intuition of Perplexity

- The Shannon Game:

- How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

- Unigrams are terrible at this game. (Why?)
- A better model of a text
 - is one which assigns a higher probability to the word that actually occurs

{ mushrooms 0.1
pepperoni 0.1
anchovies 0.01
....
fried rice 0.0001
....
and 1e-100



Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

For bigrams:

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

The Shannon Game intuition for perplexity

- How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9' at random
 - Perplexity 10
- How hard is recognizing (30,000) names at random
 - Perplexity = 30,000
- If a system has to recognize
 - Operator (1 in 4)
 - Sales (1 in 4)
 - Technical Support (1 in 4)
 - 30,000 names (1 in 120,000 each)
 - Perplexity is 53
- Perplexity is weighted equivalent branching factor

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Perplexity as branching factor

- Language with higher perplexity means the number of words branching from a previous word is larger on average.
- The difference between the perplexity of a language model and the true perplexity of the language is an indication of the quality of the model.

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109



Approximating Shakespeare

1
gram

–To him swallowed confess hear both. Which. Of save on trail for are ay device and
rote life have
–Hill he late speaks; or! a more to leg less first you enter

2
gram

–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live
king. Follow.
–What means, sir. I confess she? then all sorts, he is trim, captain.

3
gram

–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say,
'tis done.
–This shall forbid it should be branded, if renown made it empty.

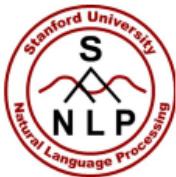
4
gram

–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A
great banquet serv'd in;
–It cannot be but so.



Shakespeare as corpus

- $N=884,647$ tokens, $V=29,066$
- Shakespeare produced 300,000 bigram types out of $V^2= 844$ million possible bigrams.
 - So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it *is* Shakespeare



The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
 - In real life, it often doesn't
 - We need to train robust models that generalize!
 - One kind of generalization: Zeros!
 - Things that don't ever occur in the training set
 - But occur in the test set

Zeros

- Training set:
 - ... denied the allegations
 - ... denied the reports
 - ... denied the claims
 - ... denied the request
- Test set
 - ... denied the offer
 - ... denied the loan

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

Dan Jurafsky



Zero probability bigrams

- Bigrams with zero probability
 - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!



Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- MLE estimate:

$$P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

- Add-1 estimate:



The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w | \text{denied the})$

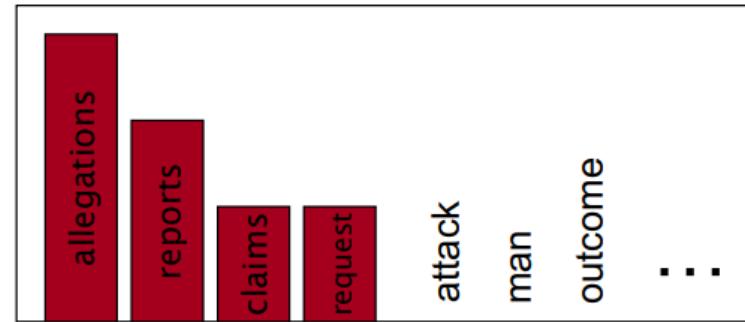
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

$P(w | \text{denied the})$

2.5 allegations

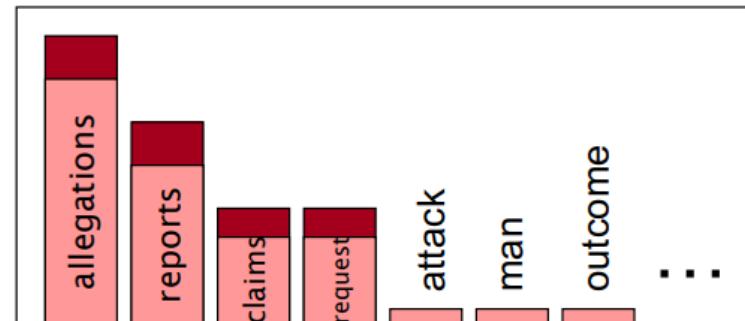
1.5 reports

0.5 claims

0.5 request

2 other

7 total





Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1



Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058



Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Other Methods for Smoothing

Interpolation, Backoff,
and Web-Scale LMs