Introduction to language modeling

Dr. Mohamed Waleed Fakhr

AAST

Language Engineering Conference

22 December 2009

Topics

- Why a language model?
- Probability in brief
- Word prediction task
- Language modeling (N-grams)
  - N-gram intro.
  - Model evaluation
  - Smoothing
- Other modeling approaches

Why a language model?

- Suppose a machine is required to translate: “The human Race”.
- The word “Race” has at least 2 meanings, which one to choose?
- Obviously, the choice depends on the “history” or the “context” preceding the word “Race”. E.g., “the human race” versus “the dogs race”.
- A statistical language model can solve this ambiguity by giving higher probability to the correct meaning.

Probability in brief

- Joint probability: P(A,B) is the probability that events A and B are simultaneously true (observed together).
- Conditional probability: P(A|B): is the probability that A is true given that B is true (observed).

Relation between joint and conditional probabilities

- **BAYES RULE:**

$P(A|B) = P(A,B)/P(B)$

$P(B|A) = P(A,B)/P(A)$

Or;

$P(A,B)= P(A).P(B|A) = P(B).P(A|B)$

Chain Rule

- The joint probability:
  P(A,B,C,D)=P(A).P(B|A).P(C|A,B).P(D|A,B,C)
- This will lend itself to the language modeling paradigm as we will be concerned by the joint probability of the occurrence of a word-sequence (W_1,W_2,W_3,….W_n):

  P(W_1,W_2,W_3,….W_n)

  which will be put in terms of conditional probability terms:
- P(W1).P(W2|W1).P(W3|W1,W2)………

  (More of this later)

Language Modeling?

In the narrow sense, statistical language modeling is concerned by estimating the joint probability of a word sequence . $P(W_1,W_2,W_3,\ldots.W_n)$

This is always converted into conditional probs: P(Next Word | History)

e.g., $P(W3|W1,W2)$

i.e., can we predict the next word given the previous words that have been observed?

In other words, if we have a History, find the Next-Word that gives the highest prob.

Word Prediction

- Guess the next word...

  ... It is too late I want to go ???

  ... I notice three guys standing on the ???

- There are many sources of knowledge that can be used to inform this task, including arbitrary world knowledge and deeper history (It is too late)
- But it turns out that we can do pretty well by simply looking at the preceding words and keeping track of some fairly simple counts.

Word Prediction

- We can formalize this task using what are called *N*-gram models.
- *N*-grams are token sequences of length *N*.
- Our 2nd example contains the following 2-grams (Bigrams)
  - (I notice), (notice three), (three guys), (guys standing), (standing on), (on the)
- Given knowledge of counts of N-grams such as these, we can guess likely next words in a sequence.

N-Gram Models

- More formally, we can use knowledge of the counts of *N*-grams to assess the conditional probability of candidate words as the next word in a sequence.
- In doing so, we actually use them to assess the joint probability of an entire sequence of words. (chain rule).

Applications

- It turns out that being able to predict the next word (or any linguistic unit) in a sequence is an extremely useful thing to be able to do.
- As we’ll see, it lies at the core of the following applications
  - Automatic speech recognition
  - Handwriting and character recognition
  - Spelling correction
  - Machine translation
  - Information retrieval
  - And many more.

ASR

$$\arg\max_{wordsequence} P(wordsequence \mid acoustics) =$$

$$\arg\max_{wordsequence} \frac{P(acoustics \mid wordsequence) \times P(wordsequence)}{P(acoustics)}$$

$$\arg\max_{wordsequence} P(acoustics \mid wordsequence) \times P(wordsequence)$$

Source Channel Model for Machine Translation

$$\arg\max_{wordsequence} P(wordsequence \mid acoustics) =$$

$$\arg\max_{wordsequence} \frac{P(acoustics \mid wordsequence) \times P(wordsequence)}{P(acoustics)}$$

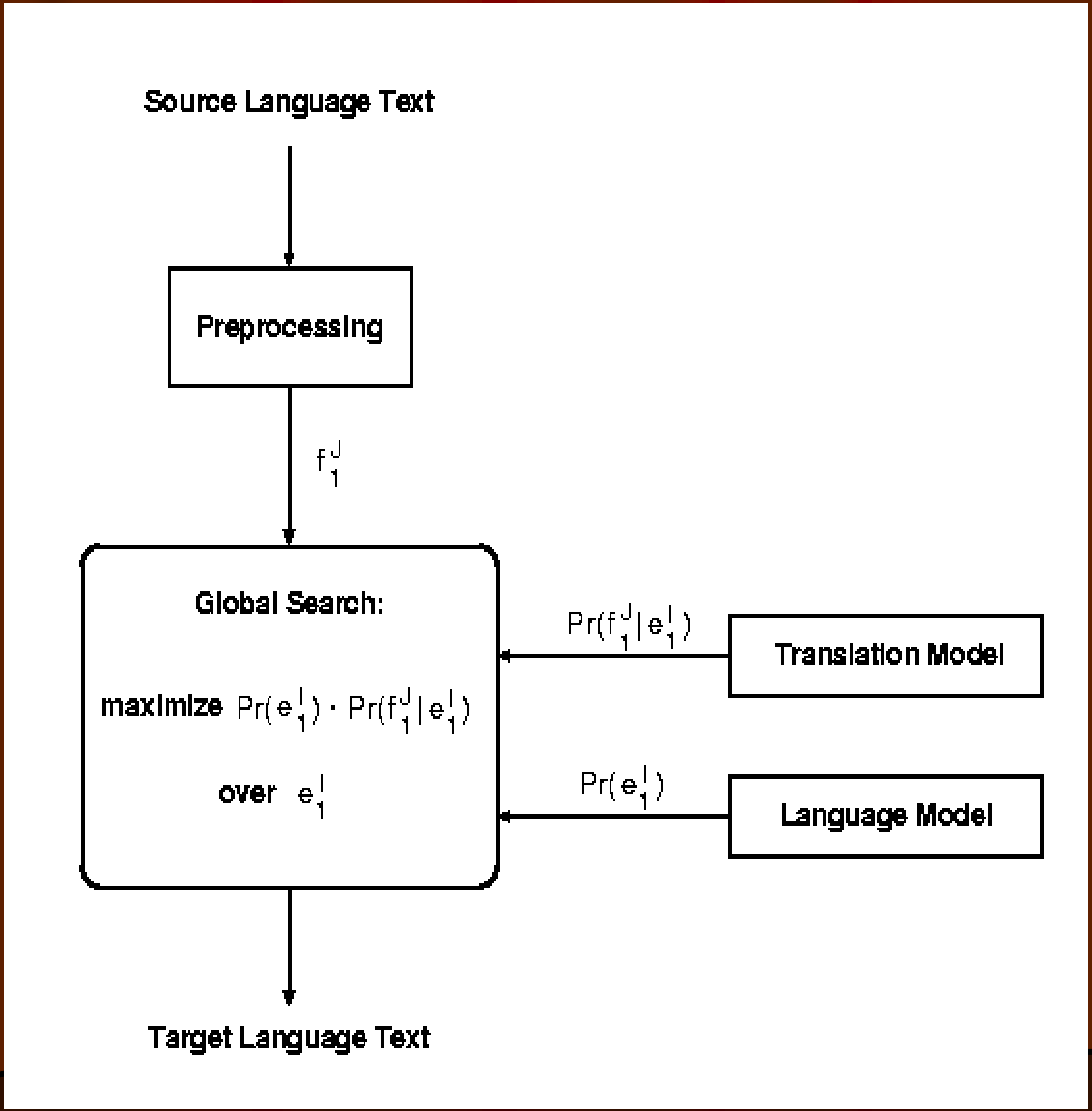
$$\arg\max_{wordsequence} P(acoustics \mid wordsequence) \times P(wordsequence)$$

$$\arg\max_{wordsequence} P(english \mid french) =$$

$$\arg\max_{wordsequence} \frac{P(french \mid english) \times P(english)}{P(french)}$$

$$\arg\max_{wordsequence} P(french \mid english) \times P(english)$$

SMT Architecture



Based on Bayes´ Decision Rule:

$$\hat{e} = \text{argmax}\{\, p(e \mid f) \,\}$$
$$= \text{argmax}\{\, p(e)\, p(f \mid e) \,\}$$

Counting

- Simple counting lies at the core of any probabilistic approach. So let’s first take a look at what we’re counting.
  - *He stepped out into the hall, was delighted to encounter a water brother.*
    - 13 tokens, 15 if we include “,” and “.” as separate tokens.
    - Assuming we include the comma and period, how many bigrams are there?

Counting

- Not always that simple
  - *I do uh main- mainly business data processing*
- Spoken language poses various challenges.
  - Should we count “uh” and other fillers as tokens?
  - What about the repetition of “mainly”? Should such do-overs count twice or just once?
  - The answers depend on the application.
    - If we’re focusing on something like ASR to support indexing for search, then “uh” isn’t helpful (it’s not likely to occur as a query).
    - But filled pauses are very useful in dialog management, so we might want them there.

Counting: Types and Tokens

- How about
  - *They picnicked by the pool, then lay back on the grass and looked at the stars.*
    - 18 tokens (again counting punctuation)
- But we might also note that "*the*" is used 3 times, so there are only 16 unique types (as opposed to tokens).
- In going forward, we'll have occasion to focus on counting both types and tokens of both words and *N*-grams.

Counting: Wordforms

- Should “cats” and “cat” count as the same when we’re counting?
- How about “geese” and “goose”?
- Some terminology:
  - Lemma: a set of lexical forms having the same stem, major part of speech, and rough word sense: (car, cars, automobile)
  - Wordform: fully inflected surface form
- Again, we’ll have occasion to count both lemmas, morphemes, and wordforms

Counting: Corpora

- So what happens when we look at large bodies of text instead of single utterances?
- Brown et al (1992) large corpus of English text
  - 583 million wordform tokens
  - 293,181 wordform types
- Google
  - Crawl of 1,024,908,267,229 English tokens
  - 13,588,391 wordform types
    - That seems like a lot of types... After all, even large dictionaries of English have only around 500 here?

•Numbers
•Misspellings
•Names
•Acronyms
•etc

Language Modeling

- Back to word prediction
- We can model the word prediction task as the ability to assess the conditional probability of a word given the previous words in the sequence
  - $P(w_n|w_1,w_2\ldots w_{n-1})$
- We'll call a statistical model that can assess this a *Language Model*

Language Modeling

- How might we go about calculating such a conditional probability?
  - One way is to use the definition of conditional probabilities and look for counts. So to get
  - P(*the* | *its water is so transparent that*)
- By definition that's

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

We can get each of those counts in a large corpus.

Very Easy Estimate

- According to Google those counts are 5/9.
  - Unfortunately... 2 of those were to these slides... So maybe it’s really   3/7
  - In any case, that’s not terribly convincing due to the small numbers involved.

Language Modeling

- Unfortunately, for most sequences and for most text collections we won’t get good estimates from this method.
  - What we’re likely to get is 0. Or worse 0/0.
- Clearly, we’ll have to be a little more clever.
  - Let’s use the chain rule of probability
  - And a particularly useful independence assumption.

The Chain Rule

- Recall the definition of conditional probabilities

$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

- Rewriting:

$$P(A,B) = P(B).P(A \mid B)$$

- For sequences...
  - $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$
- In general
  - $P(x_1,x_2,x_3,\ldots x_n) =$ $P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1\ldots x_{n-1})$

The Chain Rule

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned}$$

P(its water was so transparent)=

P(its)*

P(water|its)*

P(was|its water)*

P(so|its water was)*

P(transparent|its water was so)

Unfortunately

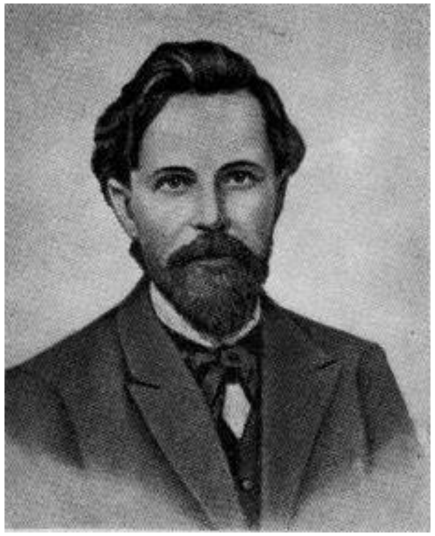
- There are still a lot of possible sentences
- In general, we’ll never be able to get enough data to compute the statistics for those longer prefixes
  - Same problem we had for the strings themselves

Independence Assumption

- Make the simplifying assumption
  - P(lizard|the,other,day,I,was,walking,along,and ,saw,a) = P(lizard|a)
- Or maybe
  - P(lizard|the,other,day,I,was,walking,along,and ,saw,a) = P(lizard|saw,a)
- That is, the probability in question is independent of its earlier history.

Independence Assumption

- This particular kind of independence assumption is called a *Markov assumption* after the Russian mathematician Andrei Markov.



Markov Assumption

So for each component in the product replace with the approximation (assuming a prefix of N)

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

Bigram version

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

Estimating Bigram Probabilities

- Th
  Es

The picture can't be displayed.

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

Normalization

- For N-gram models to be probabilistically correct they have to obey prob. Normalization constraints:

$$\sum_{over-all-j} P(W_j \mid Context_i) = 1$$

- The sum over all words for the same context (history) must be 1.
- The context may be one word (bigram) or two words (trigram) or more.

An Example: bigrams

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

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

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

estimates depend on the corpus

- The maximum likelihood estimate of some parameter of a model M from a training set T
  - Is the estimate that maximizes the likelihood of the training set T given the model M
- Suppose the word Chinese occurs 400 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text from the same distribution will be “Chinese”
- MLE estimate is 400/1000000 = .004
  - This may be a bad estimate for some other corpus

Berkeley Restaurant Project Sentences examples

- *can you tell me about any good cantonese restaurants close by*
- *mid priced thai food is what i'm looking for*
- *tell me about chez panisse*
- *can you give me a listing of the kinds of food that are available*
- *i'm looking for a good place to eat breakfast*
- *when is caffe venezia open during the day*

Bigram Counts

- Out of 9222 sentences
  - e.g. “I want” occurred 827 times

	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

Bigram Probabilities

- Divide bigram counts by prefix unigram counts to get probabilities.

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

	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

examples

- P(Want | I) = C(I Want) / C(I)

 = 827/2533 = 0.33

P(Food | Chinese) = C(Chinese Food) / C(Chinese)

= 82/158 = 0.52

Bigram Estimates of Sentence Probabilities

- P(<s> I want english food </s>) =

  P(i|<s>)*

  P(want|I)*

  P(english|want)*

  P(food|english)*

  P(</s>|food)*

  =.000031

Evaluation

- How do we know if our models are any good?
  - And in particular, how do we know if one model is better than another?

Evaluation

- Standard method
  - Train parameters of our model on a **training set**.
  - Look at the models performance on some new data
    - This is exactly what happens in the real world; we want to know how our model performs on data we haven’t seen
  - So use a **test set**. A dataset which is different than our training set, but is drawn from the same source
  - Then we need an **evaluation metric** to tell us how well our model is doing on the test set.
    - One such metric is **perplexity**

Unknown Words

- But once we start looking at test data, we'll run into words that we haven't seen before (pretty much regardless of how much training data you have) (zero unigrams)
- With an ***Open Vocabulary* task**
  - Create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L, of size V
      - From a dictionary or
      - A subset of terms from the training set
    - At text normalization phase, any training word not in L changed to **<UNK>**
    - Now we count that like a normal word
  - At test time
    - Use <**UNK>** counts for any word not in training

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

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

- Chain rule:

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

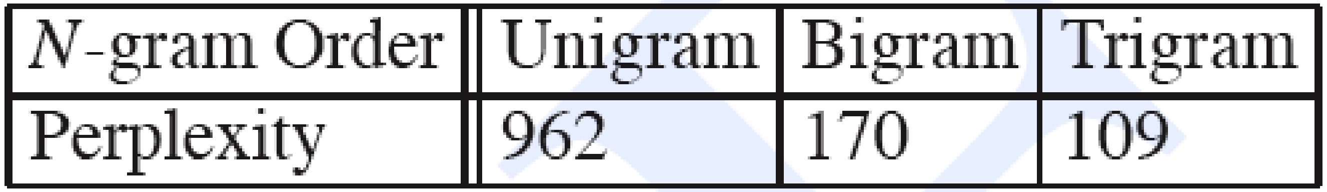
- For bigrams:

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

- Minimizing perplexity is the same as maximizing probability
  - **The best language model is one that best predicts an unseen test set**

Lower perplexity means a better model

- Training 38 million words, test 1.5 million words, WSJ (Wall-Street Journal)



N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Evaluating *N*-Gram Models

- Best evaluation for a language model
  - Put model *A* into an application
    - For example, a speech recognizer
  - Evaluate the performance of the application with model *A*
  - Put model *B* into the application and evaluate
  - Compare performance of the application with the two models
  - ***Extrinsic evaluation***

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

- Extrinsic evaluation
  - This is really time-consuming
  - Can take days to run an experiment
- So
  - To evaluate N-grams we often use an **intrinsic** evaluation, an approximation called **perplexity**
  - But perplexity is a poor approximation unless the test data looks **similar to** the training data
  - So is **generally only useful in pilot experiments**
  - **But still, there is nothing like the real experiment!**

N-gram Zero Counts

- For the English language,
  - V^2= 844 million possible bigrams...
  - So, for a medium size training data, e.g., Shakespeare novels, 300,000 bigrams were found Thus, 99.96% of the possible bigrams were never seen (have zero entries in the table)
  - Does that mean that any ***test*** sentence that contains one of those bigrams should have a probability of 0?

N-gram Zero Counts

- Some of those zeros are really zeros...
  - Things that really can’t or shouldn’t happen.
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count (probably a count of 1).
- Zipf’s Law (long tail phenomenon):
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are sparse ! We have no counts at all for the vast bulk of things we want to estimate!
- Answer:
  - ***Estimate*** the likelihood of unseen (zero count) N-grams!
  - **N-gram Smoothing techniques**

Laplace Smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- This adds extra *V* observations (*V* is vocab. Size)

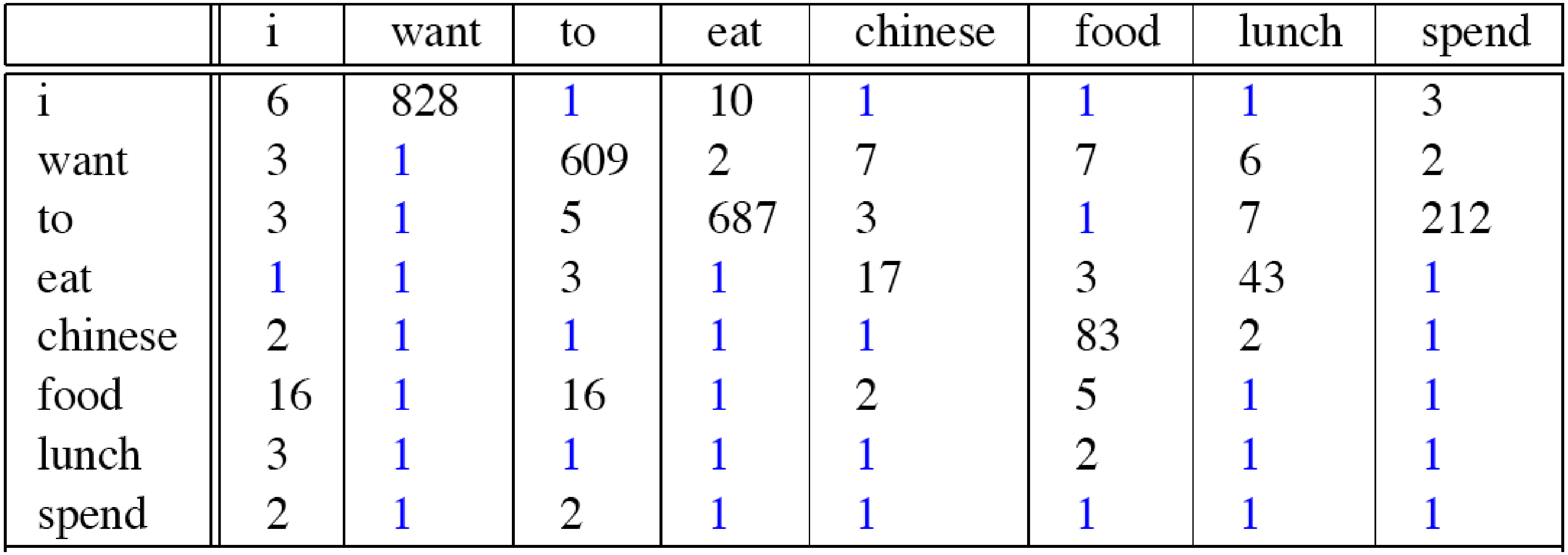
- MLE estimate: $$P(w_i) = \frac{c_i}{N}$$

- Laplace estimate: $$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$ $$\boldsymbol{P_{Laplace}} = \frac{1}{N} \frac{(ci+1).N}{(N+V)}$$

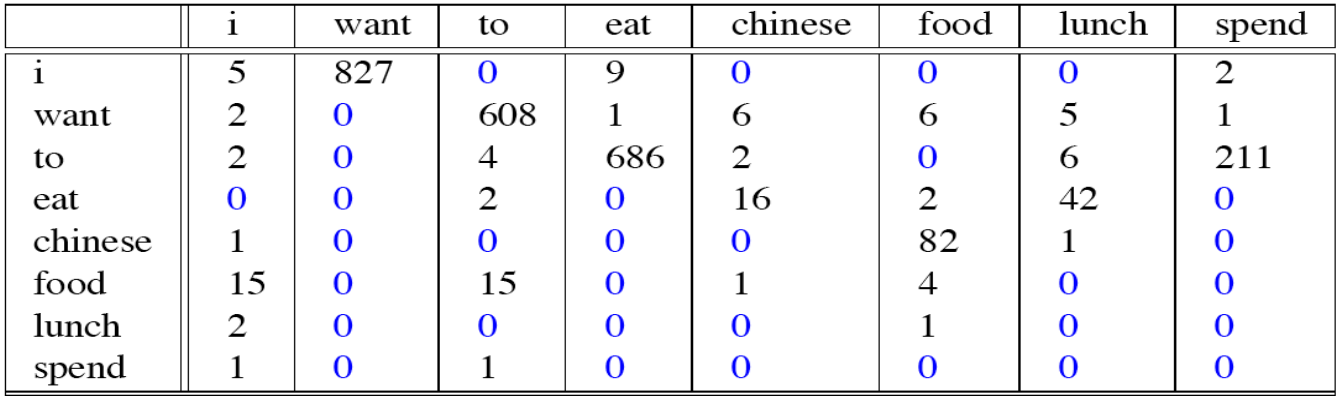
- Reconstructed counts:

(making the volume N again) $$c_i^* = (c_i + 1)\frac{N}{N + V}$$

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



	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

Laplace-Smoothed Bigram Probabilities

The picture can't be displayed.

	i	ch	spend
i	0.00	0025	0.00075
want	0.00	025	0.00084
to	0.00	018	0.055
eat	0.00	2	0.00046
chinese	0.00	012	0.00062
food	0.00	0039	0.00039
lunch	0.00	0056	0.00056
spend	0.00	0058	0.00058

Reconstructed Counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

	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

$$P(w1 \mid w2) = \frac{C(w2w1)+1}{C(w2)+V} = \frac{C(w2)}{C(w2)}\,\frac{C(w2w1)+1}{C(w2)+V} = \frac{1}{C(w2)}\,\frac{C(w2).[C(w2w1)+1]}{[C(w2)+V]}$$

Big Change to the Counts!

- C(want to) went from 608 to 238!
- P(to|want) from .66 to .26!
- Discount d= c*/c
  - d for “Chinese food” = 0.1 !!! A 10x reduction
  - So in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)
- But Laplace smoothing not used for N-grams, as we have much better methods
- Despite its flaws, Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn’t so huge.

Better Smoothing

- Intuition used by many smoothing algorithms, for example;
  - Good-Turing
  - Kneyser-Ney
  - Witten-Bell
- Is to use the count of things we've seen ***<u>once</u>*** to help estimate the count of things we've never seen

Good-Turing
Josh Goodman Intuition

- Imagine you are fishing
  - There are 8 species in this waters: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?
  - 3/18 (3 is number of events that seen once)
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18 because we just stole 3/18 of our probability mass to use on unseen events

Good-Turing

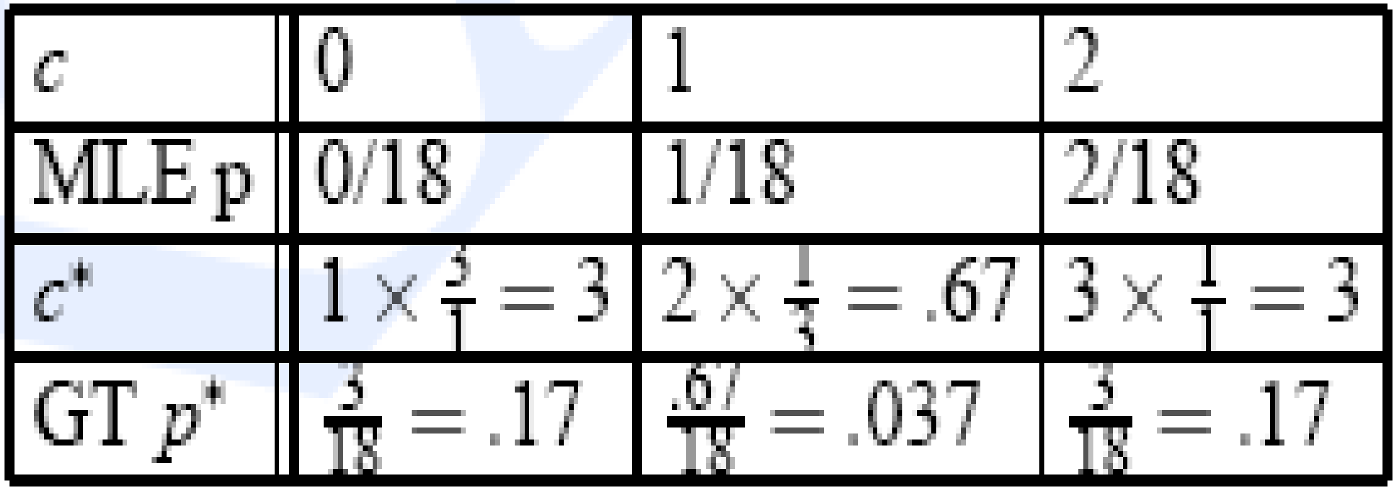
Notation: Nx is the frequency-of-frequency-x

- So N**10**=1
  - Number of fish species seen 10 times is 1 (carp)
- N**1**=3
  - Number of fish species seen 1 time is 3 (trout, salmon, eel)

  - To estimate total number of unseen species (seen 0 times)
- Use number of species (bigrams) we’ve seen once (i.e. 3)
- So, the estimated count c* for <unseen> is 3.
- All other estimates are adjusted (down) to account for the stolen mass given for the unseen events, using the formula:

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

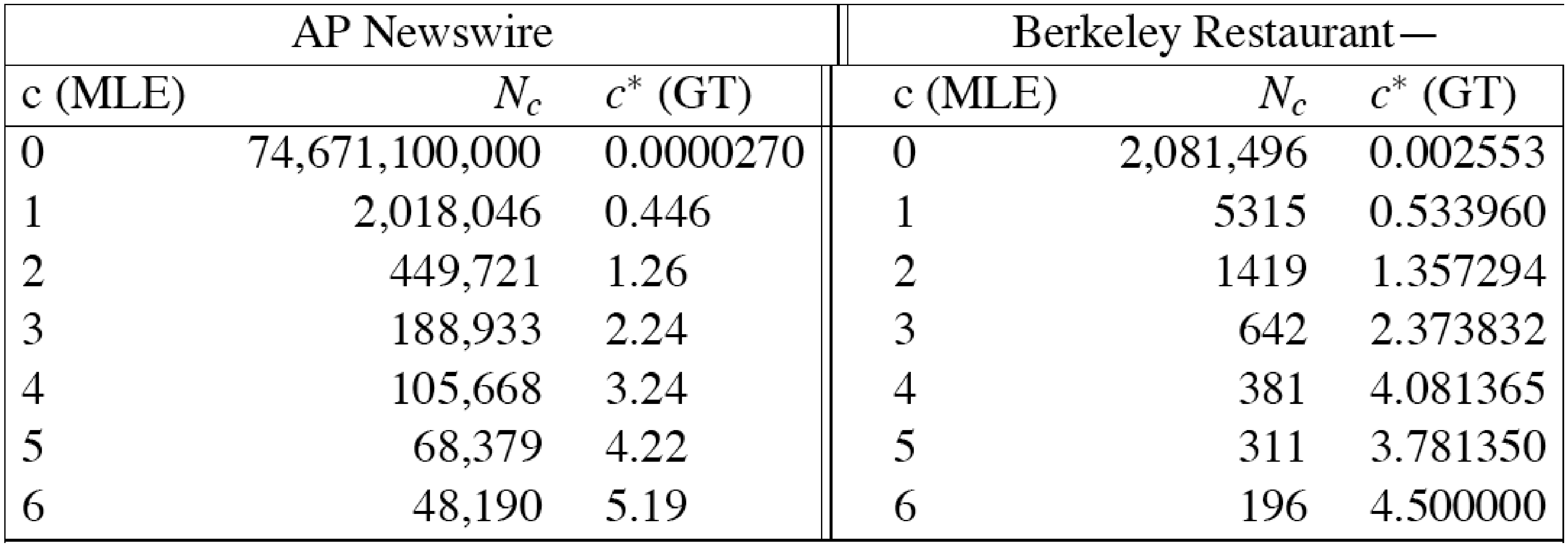
GT Fish Example



c	0	1	2
MLE p	0/18	1/18	2/18
c^*	$1 \times \frac{3}{1} = 3$	$2 \times \frac{1}{3} = .67$	$3 \times \frac{1}{1} = 3$
GT p^*	$\frac{3}{18} = .17$	$\frac{.67}{18} = .037$	$\frac{3}{18} = .17$

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

Bigram Frequencies of Frequencies and GT Re-estimates



AP Newswire			Berkeley Restaurant—		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

AP Newswire: 22million words,   Berkeley: 9332 sentences

Backoff and Interpolation

- Another really useful source of knowledge
- If we are estimating:
  - trigram p(z|x,y)
  - but count(xyz) is zero
- Use info from:
  - Bigram p(z|y)
- Or even:
  - Unigram p(z)
- How to combine this trigram, bigram, unigram info in a valid fashion?

Backoff Vs. Interpolation

1. **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram
2. **Interpolation**: mix all three by weights

Interpolation

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) \;=\; & \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ & +\lambda_2 P(w_n|w_{n-1}) \\ & +\lambda_3 P(w_n)\end{aligned}$$

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) \;=\; & \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ & +\lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ & +\lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned}$$

How to Set the Lambdas?

- Use a **held-out, or development** corpus
- Choose lambdas which maximize the probability of some held-out data
  - I.e. fix the *N*-gram probabilities
  - Then search for lambda values that when plugged into previous equation give largest probability for held-out set
  - Can use EM to do this search
  - Can use direct search methods (Genetic, Swarm, etc…)

Katz Backoff (very popular)

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|x,y) = \begin{cases} P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\ P^*(z), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|y) = \begin{cases} P^*(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P^*(z), & \text{otherwise.} \end{cases}$$

Why discounts P* and alpha?

- MLE probabilities sum to 1

$$\sum_i P(w_i|w_j w_k) = 1$$

- So if we used MLE probabilities but backed off to lower order model when MLE prob is zero we would be adding extra probability mass (it is like in smoothing), and total probability would be greater than 1. So, we have to do discounting.

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Other Approaches

Class-based LMs
Morpheme-based LMs
Skip LMs

Class-based Language Models

- Standard word-based language models

$$p(w_1, w_2, ..., w_T) = \prod_{t=1}^{T} p(w_t \mid w_1, ..., w_{t-1})$$

$$\approx \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

- How to get robust n-gram estimates ($p(w_t \mid w_{t-1}, w_{t-2})$)?
  - Smoothing
    - E.g. Kneyser-Ney, Good-Turing
  - Class-based language models

$$p(w_t \mid w_{t-1}) \approx p(w_t \mid C(w_t)) p(C(w_t) \mid C(w_{t-1}))$$

Limitation of Word-based Language Models

- **Words are inseparable whole units**.
  - E.g. “book” and “books” are distinct vocabulary units
- Especially problematic in **morphologically-rich languages**:
  - E.g. Arabic, Finnish, Russian, Turkish
  - Many unseen word contexts
  - High out-of-vocabulary rate
  - High perplexity

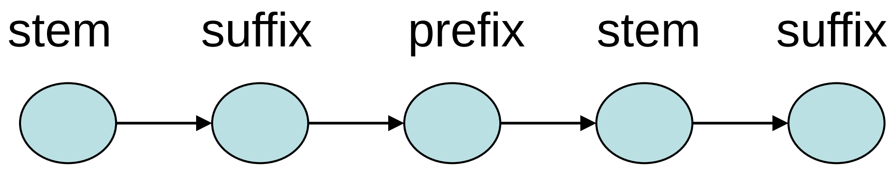
Arabic k-t-b	
Kitaab	A book
Kitaab-iy	My book
Kitaabu-hum	Their book
Kutub	Books

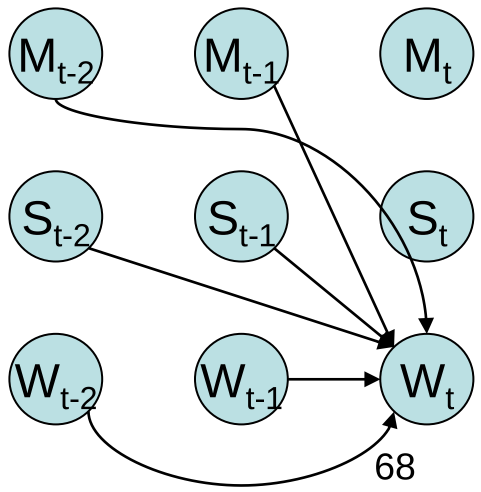
Solution: Word as Factors

- Decompose words into “factors” (e.g. stems)
- Build language model over factors: P(w|factors)
- Two approaches for decomposition
  - Linear
    - [e.g. Geutner, 1995]
  - Parallel

[Kirchhoff et. al., JHU Workshop 2002]

[Bilmes & Kirchhoff, NAACL/HLT 2003]





Different Kinds of Language Models

- cache language models (constantly adapting to a floating text)
- trigger language models (can handle long distance effects)
- POS-based language models, LM over POS tags
- class-based language models based on semantic classes
- multilevel *n*-gram language models (mix many LM together)
- interleaved language models (different LM for different parts of text)
- morpheme-based language models (separate words into core and modifyers)
- context free grammar language models (use simple and efficient LM-definition)
- decision tree language models (handle long distance effects, use rules)
- HMM language models (stochastic decision for combination of independent LMs)

HTK Tool Kit

HTK Tool Kit

What is HTK tool kit

The HTK language modeling tools are a group of programs designed for constructing and testing statistical *n-gram* language models

HTK Tool Kit

What to prepare

Training & Test Text

Dictionary

HTK Tool Kit

Training & Test Text

- Plain text sentences
- One sentence per line
- Sentence starts with <s>
- Sentence ends with </s>

HTK Tool Kit

Training Text Sample

<s> IT WAS ON A BITTERLY COLD NIGHT AND FROSTY MORNING TOWARDS THE END OF THE WINTER OF NINETY SEVEN THAT I WAS AWAKENED BY A TUGGING AT MY SHOULDER </s>

<s> IT WAS HOLMES </s>

HTK Tool Kit

Dictionary

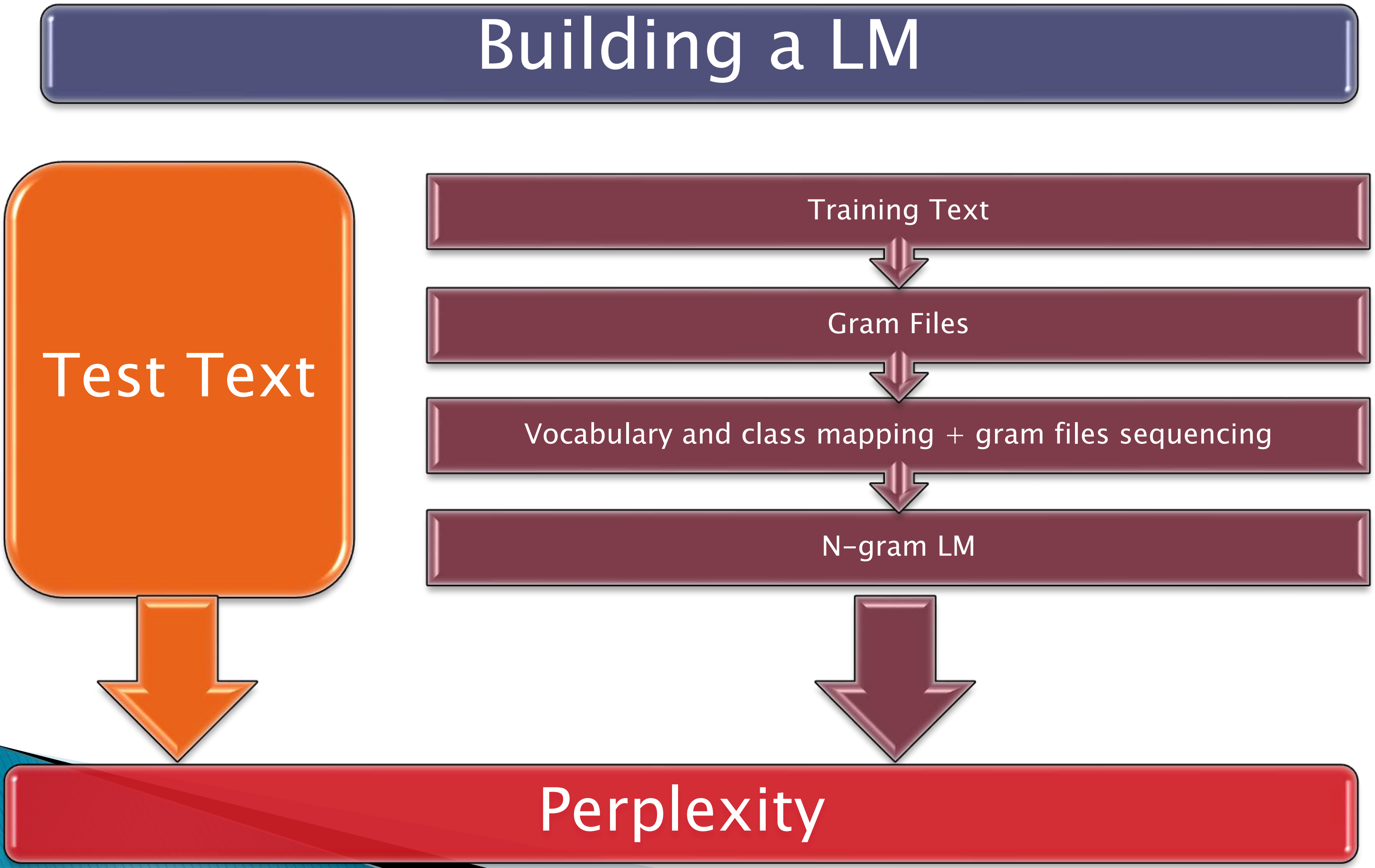
- Plain text wordlist
- One word per line
- Alphabetically ordered

HTK Tool Kit

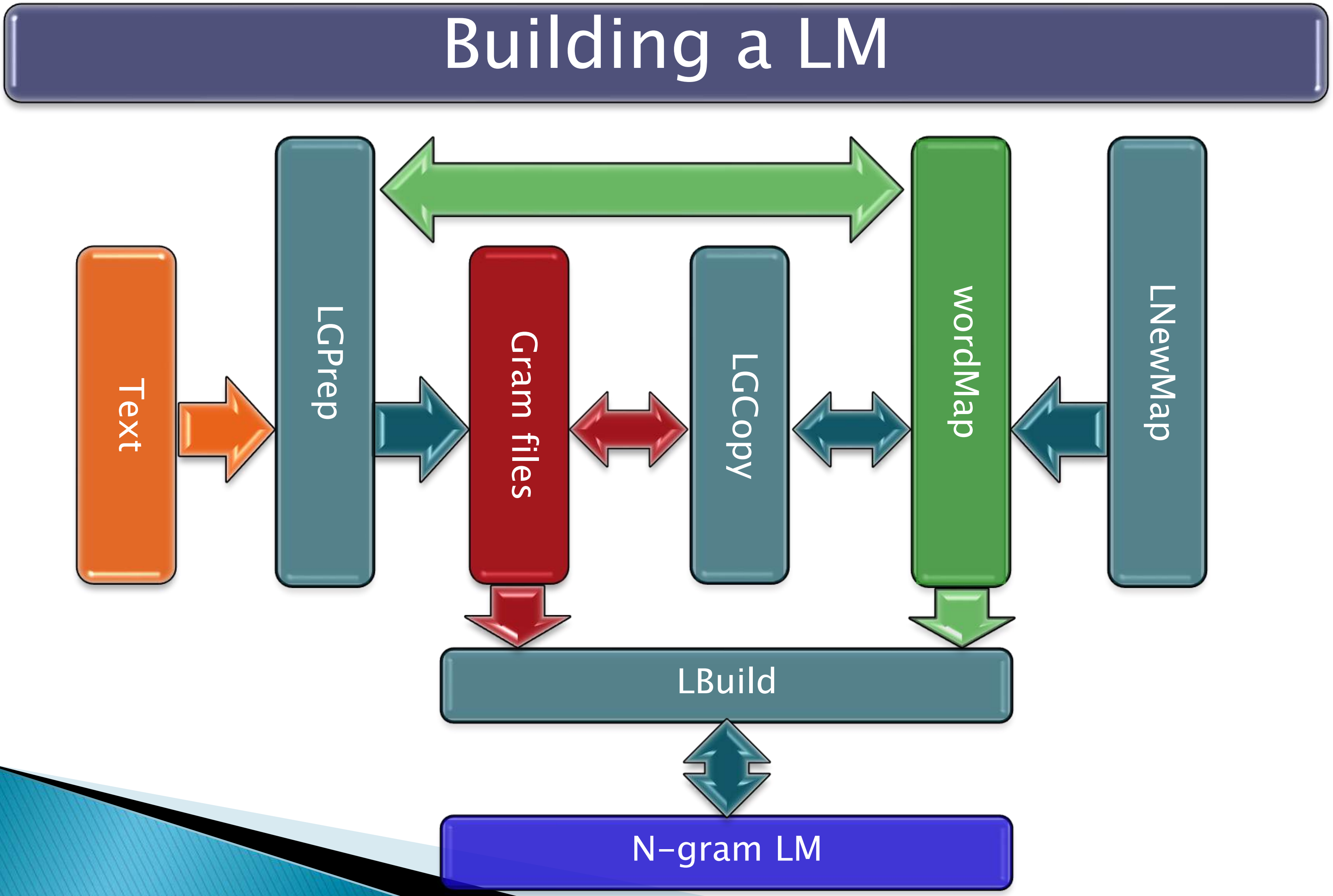
Dictionary Sample

```
</s>
<s>
A
A.
ABANDON
ABANDONED
ABBEY
ABDULLAH
ABE
```

HTK Tool Kit


Building a LM
Test Text
Training Text
Gram Files
Vocabulary and class mapping + gram files sequencing
N-gram LM
Perplexity

HTK Tool Kit



Building a LM
Text
LGPrep
Gram files
LGCopy
wordMap
LNewMap
LBuild
N-gram LM

HTK Tool Kit

LNewMap

LNewMap [options] name mapfn

-e esc Change the contents of the EscMode header to esc. Default is RAW.

-f fld Add the field fld to the Fields header.

HTK Tool Kit

LNewMap

Example:

```
LNewMap –f WFC Holmes empty.wmap
```

```
Name = Holmes
SeqNo = 0
Entries = 0
EscMode = RAW
Fields = ID,WFC
\Words\
```

HTK Tool Kit

LGPrep

LGPrep [options] wordmap [textfile ...]

-a n Allow upto n new words in input texts (default 100000).

-b n Set the internal gram buffer size to n (default 2000000). LGPrep stores incoming n-grams in this buffer. When the buffer is full, the contents are sorted and written to an output gram file. Thus, the buffer size determines the amount of process memory that LGPrep will use and the size of the individual output gram files.

HTK Tool Kit

LGPrep cont'd

LGPrep [options] wordmap [textfile ...]

-d Directory in which to store the output gram files (default current directory).

-i n Set the index of the first gram file output to be n (default 0).

-n n Set the output n-gram size to n (default 3).

-r s Set the root name of the output gram files to s (default "gram").

HTK Tool Kit

LGPrep cont'd

LGPrep [options] wordmap [textfile ...]

-s s Write the string s into the source field of the output gram files. This string should be a comment describing the text source.

-z Suppress gram file output. This option allows LGPrep to be used just to compute a word frequency map. It is also normally applied when applying edit rules to the input.

HTK Tool Kit

LGPrep cont’d

Example:

```
LGPrep –T 1 –a 100000 –b 2000000 –d holmes.0 –n 4
–s "Sherlock Holmes" empty.wmap
D:\train\abbey_grange.txt, D:\train\beryl_coronet.txt,...
```

HTK Tool Kit

LGPrep cont'd

WMAP file

```
Name = Holmes
SeqNo = 1
Entries = 18080
EscMode = RAW
Fields = ID,WFC
\Words\
<s>     65536   33669
IT      65537   8106
WAS     65538   7595
...
```

HTK Tool Kit

LGCopy

LGCopy [options] wordmap [mult] gramfiles

-b n Set the internal gram buffer size to n (default 2000000). LGPrep stores incoming n-grams in this buffer. When the buffer is full, the contents are sorted and written to an output gram file. Thus, the buffer size determines the amount of process memory that LGPrep will use and the size of the individual output gram files.

-d Directory in which to store the output gram files (default current directory).

HTK Tool Kit

LGCopy cont’d

LGCopy [options] wordmap [mult] gramfiles

-o n Output class mappings only. Normally all input *n*–grams are copied to the output, however, if a class map is specified, this options forces the tool to output only *n*–grams containing at least one class symbol.

HTK Tool Kit

LGCopy cont’d

Example:

```
LGCopy –T 1 –b 2000000 –d D:\holmes.1
D:\ holmes.0\wmap  D:\ holmes.0\gram.1 D:\
holmes.0\gram.2.....
```

HTK Tool Kit

LBuild

```
LBuild  [options]  wordmap  outfile  [mult] gramfile ..
```

-c n c Set cutoff for n-gram to c.

-n n Set final model order to n.

HTK Tool Kit

LBuild cont’d

Example:

```
LBuild –T 1 –c 2 1 –c 3 1 –n 3 D:\lm_5k\5k.wmap
D:\lm_5k\tg2-1_1 D:\holmes.1\data.1
D:\holmes.1\data.2…  D:\lm_5k\data.1 D:\lm_5k\data.12
```

HTK Tool Kit

LPlex

LPlex [options] langmodel labelFiles

-n n Perform a perplexity test using the n-gram component of the model. Multiple tests can be specified. By default the tool will use the maximum value of n available.

-t Text stream mode. If this option is set, the specified test files will be assumed to contain plain text.

HTK Tool Kit

LPlex cont'd

Example:

```
Lplex -n 3 -t D:\lm_5k\tg1_1 D:\test\red-
headed_league.txt
```

Statistical Language Modeling using SRILM Toolkit

Presented by:
Kamal Eldin Mahmoud

AGENDA

- **Introduction**
- **Basic SRILM Tools**
  - **ngram-count**
  - **ngram**
  - **ngram-merge**
- **Basic SRILM file format**
  - **ngram-format**
  - **nbest-format**

AGENDA

Basic SRILM Scripts

- **Training-scripts**
- **lm-scripts**
- **ppl-scripts**

Introduction

- SRILM is a collection of C++ libraries, executable programs, and helper scripts.

- The toolkit supports creation and evaluation of a variety of language model types based on N-gram statistics.

- The main purpose of SRILM is to support language model estimation and evaluation.

- Since most LMs in SRILM are based on N-gram statistics, the tools to accomplish these two purposes are named ngram-count and ngram, respectively.

Introduction

- A standard LM (trigram with Good-Turing discounting and Katz backoff for smoothing) would be created by

ngram-count -text TRAINDATA -lm LM

- The resulting LM may then be evaluated on a test corpus using

ngram -lm LM -ppl TESTDATA -debug 0

Basic SRILM Tools

ngram-count

ngram-count generates and manipulates N-gram counts, and estimates N-gram language models from them.

Syntax:

Ngram-count *[* ***-help*** *]* *option ...*

ngram-count options

Each filename argument can be an ASCII file, or a compressed file (name ending in .Z or .gz)

-help
Print option summary.
-version
Print version information.
-order n
Set the maximal order (length) of N-grams to count. This also determines the order of the estimated LM, if any. The default order is 3.
-memuse
Print memory usage statistics.

ngram-count options

-vocab *file*
Read a vocabulary from file.

-vocab-aliases *file*
Reads vocabulary alias definitions from file, consisting of lines of the form

alias word

This causes all tokens alias to be mapped to word.

-**write-vocab** *file*
-write-vocab-index *file*
Write the vocabulary built in the counting process to file.

ngram-count counting options

-tolower
Map all vocabulary to lowercase.

-text *textfile*
Generate N-gram counts from text file.

-no-sos
Disable the automatic insertion of start-of-sentence tokens in N-gram counting.
-no-eos
Disable the automatic insertion of end-of-sentence tokens in N-gram counting.

-read *countsfile*
Read N-gram counts from a file.

ngram-count counting options

-**read-google** *dir*
Read N-grams counts from an indexed directory structure rooted in dir, in a format developed by Google. The corresponding directory structure can be created using the script *make-google-ngrams* .

-**write** *file*
-**write-binary** *file*
-write-order n
-writen *file*
Write total counts to file.

-sort
Output counts in lexicographic order, as required for ngram-merge.

ngram-count lm options

-lm *lmfile*
-write-binary-lm
Estimate a backoff N-gram model from the total counts, and write it to *lmfile* .

-unk
Build an ``open vocabulary'' LM.

-map-unk *word*
Map out-of-vocabulary words to *word*.

ngram-count lm options

-cdiscount*n* *discount*
Use Ney's absolute discounting for N-grams of order *n*, using *discount* as the constant to subtract.

-wbdiscount*n*
Use Witten-Bell discounting for N-grams of order *n*.

-ndiscount*n*
Use Ristad's natural discounting law for N-grams of order *n*.

-addsmooth*n* *delta*
Smooth by adding *delta* to each N-gram count.

ngram-count lm options

-kndiscount*n*
Use Chen and Goodman's modified Kneser-Ney discounting for N-grams of order *n*.

-kn-counts-modified
Indicates that input counts have already been modified for Kneser-Ney smoothing.

-interpolate*n*
Causes the discounted N-gram probability estimates at the specified order *n* to be interpolated with lower-order estimates. Only Witten-Bell, absolute discounting, and (original or modified) Kneser-Ney smoothing currently support interpolation.

ngram

Ngram performs various operations with N-gram-based and related language models, including sentence scoring, and perplexity computation.

Syntax:

ngram *[* ***-help*** *] option ...*

ngram options

-help
Print option summary.

-version
Print version information.

-order n
Set the maximal N-gram order to be used, by default 3.

-memuse
Print memory usage statistics for the LM.

ngram options

The following options determine the type of LM to be used.

-null

Use a `null' LM as the main model (one that gives probability 1 to all words).

-use-server *S*

Use a network LM server as the main model.

-lm *file*

Read the (main) N-gram model from *file*.

ngram options

-tagged
Interpret the LM as containing word/tag N-grams.

-skip
Interpret the LM as a ``skip'' N-gram model.

-classes *file*
Interpret the LM as an N-gram over word classes.

-factored
Use a factored N-gram model.

-unk
Indicates that the LM is an open-class LM.

ngram options

-ppl *textfile*
Compute sentence scores (log probabilities) and perplexities from the sentences in *textfile*.
The **-debug** option controls the level of detail printed.

-debug 0
Only summary statistics for the entire corpus are printed.

-debug 1
Statistics for individual sentences are printed.

ngram options

-debug 2
Probabilities for each word, plus LM-dependent details about backoff used etc., are printed.

-debug 3
Probabilities for all words are summed in each context, and the sum is printed.

ngram options

-nbest *file*
Read an N-best list in nbest-format and rerank the hypotheses using the specified LM. The reordered N-best list is written to stdout.

-nbest-files *filelist*
Process multiple N-best lists whose filenames are listed in *filelist*.

-write-nbest-dir *dir*
Deposit rescored N-best lists into directory *dir*, using filenames derived from the input ones.

ngram options

-decipher-nbest
Output rescored N-best lists in Decipher 1.0 format, rather than SRILM format.

-no-reorder
Output rescored N-best lists without sorting the hypotheses by their new combined scores.

-max-nbest *n*
Limits the number of hypotheses read from an N-best list.

ngram options

-no-sos
Disable the automatic insertion of start-of-sentence tokens for sentence probability computation.

-no-eos
Disable the automatic insertion of end-of-sentence tokens for sentence probability computation.

ngram-merge

ngram-merge reads two or more lexicographically sorted N-gram count files and outputs the merged, sorted counts.

Syntax:

ngram-merge [-help] [-write outfile] [-float-counts] \ [--] infile1 infile2 ...

Ngram-merge options

-write *outfile*
Write merged counts to *outfile*.

-float-counts
Process counts as floating point numbers.

--
Indicates the end of options, in case the first input filename begins with ``-''.

Basic SRILM file format

ngram-format

ngram-format File format for ARPA backoff N-gram models

```
\data\
ngram 1=n1
ngram 2=n2.
..
ngram N=nN
\1-grams:
p          w                   [bow]
...\
2-grams:
p          w1 w2               [bow]
...
\N-grams:
p          w1 ... wN
...
\end\
```

nbest-format

SRILM currently understands three different formats for lists of N-best hypotheses for rescoring or 1-best hypothesis extraction. The first two formats originated in the SRI Decipher(TM) recognition system, the third format is particular to SRILM.

The first format consists of the header

```
NBestList1.0
```

followed by one or more lines of the form

```
(score) w1 w2 w3 ...
```

where *score* is a composite acoustic/language model score from the recognizer, on the bytelog scale.

nbest-format

The second Decipher(TM) format is an extension of the first format that encodes word-level scores and time alignments. It is marked by a header of the form

NBestList2.0

The hypotheses are in the format

(*score*) *w1* (st: *st1* et: *et1* g: *g1* a: *a1*) *w2* ...

where words are followed by start and end times, language model and acoustic scores (bytelog-scaled), respectively.

nbest-format

The third format understood by SRILM lists hypotheses in the format

ascore lscore nwords w1 w2 w3 ...

where the first three columns contain the acoustic model log probability, the language model log probability, and the number of words in the hypothesis string, respectively. All scores are logarithms base 10.

Basic SRILM Scripts

Training-scripts

These scripts perform convenience tasks associated with the training of language models.

get-gt-counts

Syntax

get-gt-counts max=*K* **out=***name* [*counts* ...] **>** *gtcounts*

Computes the counts-of-counts statistics needed in Good-Turing smoothing. The frequencies of counts up to *K* are computed (default is 10). The results are stored in a series of files with root *name*, ***name*.gt1counts**,..., ***name*.gt*N*counts**.

Training-scripts

make-gt-discounts

Santax:

make-gt-discounts min=*min* max=*max gtcounts*

Takes one of the output files of get-gt-counts and computes the corresponding Good-Turing discounting factors. The output can then be passed to **ngram-count** via the **-gt*n*** options to control the smoothing during model estimation.

Training-scripts

make-abs-discount

Syntax

make-abs-discount *gtcounts*

Computes the absolute discounting constant needed for the **ngram-count -cdiscount***n* options. Input is one of the files produced by **get-gt-counts**.

Training-scripts

make-kn-discount

Syntax

make-kn-discounts min=*min gtcounts*

Computes the discounting constants used by the modified Kneser-Ney smoothing method. Input is one of the files produced by **get-gt-counts**.

Training-scripts

make-batch-counts

Syntax

make-batch-counts *file-list* \ [*batch-size* [*filter* [*count-dir* [*options* ...]]]]

Performs the first stage in the construction of very large N-gram count files. *file-list* is a list of input text files. Lines starting with a `#' character are ignored. These files will be grouped into batches of size *batch-size* (default 10). The N-gram count files are left in directory *count-dir* (``counts'' by default), where they can be found by a subsequent run of **merge-batch-counts**.

Training-scripts

merge-batch-counts

Syntax

merge-batch-counts *count-dir* [*file-list*|*start-iter*]

Completes the construction of large count files. Optionally, a *file-list* of count files to combine can be specified. A number as second argument restarts the merging process at iteration *start-iter*.

Training-scripts

make-google-ngrams

Syntax

make-google-ngrams [**dir=***DIR*] [**per_file=***N*] [**gzip=0**] \ [**yahoo=1**] [*counts-file* ...]

Takes a sorted count file as input and creates an indexed directory structure, in a format developed by Google to store very large N-gram collections. Optional arguments specify the output directory *dir* and the size *N* of individual N-gram files (default is 10 million N-grams per file). The **gzip=0** option writes plain. The **yahoo=1** option may be used to read N-gram count files in Yahoo-GALE format.

Training-scripts

tolower-ngram-counts

Syntax

tolower-ngram-counts [*counts-file* ...]

Maps an N-gram counts file to all-lowercase. No merging of N-grams that become identical in the process is done.

Training-scripts

reverse-ngram-counts

Syntax

reverse-ngram-counts [*counts-file* ...]

Reverses the word order of N-grams in a counts file or stream.

reverse-text

Syntax

reverse-text [*textfile* ...]

Reverses the word order in text files, line-by-line.

Training-scripts

compute-oov-rate

Syntax

compute-oov-rate *vocab* [*counts* ...]

Determines the out-of-vocabulary rate of a corpus from its unigram *counts* and a target vocabulary list in *vocab*.

lm-scripts

add-dummy-bows

Syntax

add-dummy-bows [*lm-file*] **>** *new-lm-file*

Adds dummy backoff weights to N-grams, even where they are not required, to satisfy some broken software that expects backoff weights on all N-grams (except those of highest order).

lm-scripts

change-lm-vocab

Syntax

change-lm-vocab -vocab *vocab* **-lm** *lm-file* **-write-lm** *new-lm-file* \ [**-tolower**] [**-subset**] [*ngram-options* ...]

Modifies the vocabulary of an LM to be that in *vocab*. Any N-grams containing OOV words are removed, new words receive a unigram probability, and the model is renormalized. The **-tolower** option causes case distinctions to be ignored. **-subset** only removes words from the LM vocabulary, without adding any.

lm-scripts

make-lm-subset

Syntax

make-lm-subset *count-file*|**-** [lm-file |**-**] **>** *new-lm-file*

Forms a new LM containing only the N-grams found in the *count-file*. The result still needs to be renormalized with **ngram -renorm** .

lm-scripts

get-unigram-probs

Syntax

get-unigram-probs [**linear=1**] [*lm-file*]

Extracts the unigram probabilities in a simple table format from a backoff language model. The **linear=1** option causes probabilities to be output on a linear (instead of log) scale.

ppl-scripts

These scripts process the output of the ngram option **-ppl** to extract various useful information.

add-ppls

Syntax

add-ppls [*ppl-file* ...]

Takes several ppl output files and computes an aggregate perplexity and corpus statistics.

ppl-scripts

subtract-ppls

Syntax

subtract-ppls *ppl-file1* [*ppl-file2* ...]

Similarly computes an aggregate perplexity by removing the statistics of zero or more *ppl-file2* from those in *ppl-file1*.

ppl-scripts

compare-ppls

Syntax

compare-ppls [**mindelta=***D*] *ppl-file1 ppl-file2*

Tallies the number of words for which two language models produce the same, higher, or lower probabilities. The input files should be **ngram -debug 2 -ppl** output for the two models on the same test set. The parameter *D* is the minimum absolute difference for two log probabilities to be considered different.

ppl-scripts

compute-best-mix

Syntax

compute-best-mix [**lambda='***l1 l2* ...**'**] **[precision=***P*] \ *ppl-file1* [*ppl-file2* ...]

Takes the output of several **ngram -debug 2 –ppl** runs on the same test set and computes the optimal interpolation weights for the corresponding models. Initial weights may be specified as *l1 l2 ...*. The computation is iterative and stops when the interpolation weights change by less than *P* (default 0.001).

ppl-scripts

compute-best-sentence-mix

Syntax

compute-best-sentence-mix [**lambda='***l1 l2* ...**'**] [**precision=***P*] \ *ppl-file1* [*ppl-file2* ...]

similarly optimizes the weights for sentence-level interpolation of LMs. It requires input files generated by **ngram -debug 1 -ppl**.

Thank You ☺