Introduction to N-gram Models

COSI 114 – Computational Linguistics

James Pustejovsky

January 23, 2015

Brandeis University

Outline

- Human Word Presentation
- Language Model
- N-gram
- N-gram Language Model
- Markov chain
- Training and Testing
- Example
- Reduce Number of Parameters in N-gram
- Maximum Likelihood Estimation(MLE)
- Sparse Data Problem
- Smoothing and Backoff
- Zipf's Law
- Evaluation
- Application

Language model

- Predicting the next word
- Evaluating how 'English' a sequence of words is

Next Word Presentation

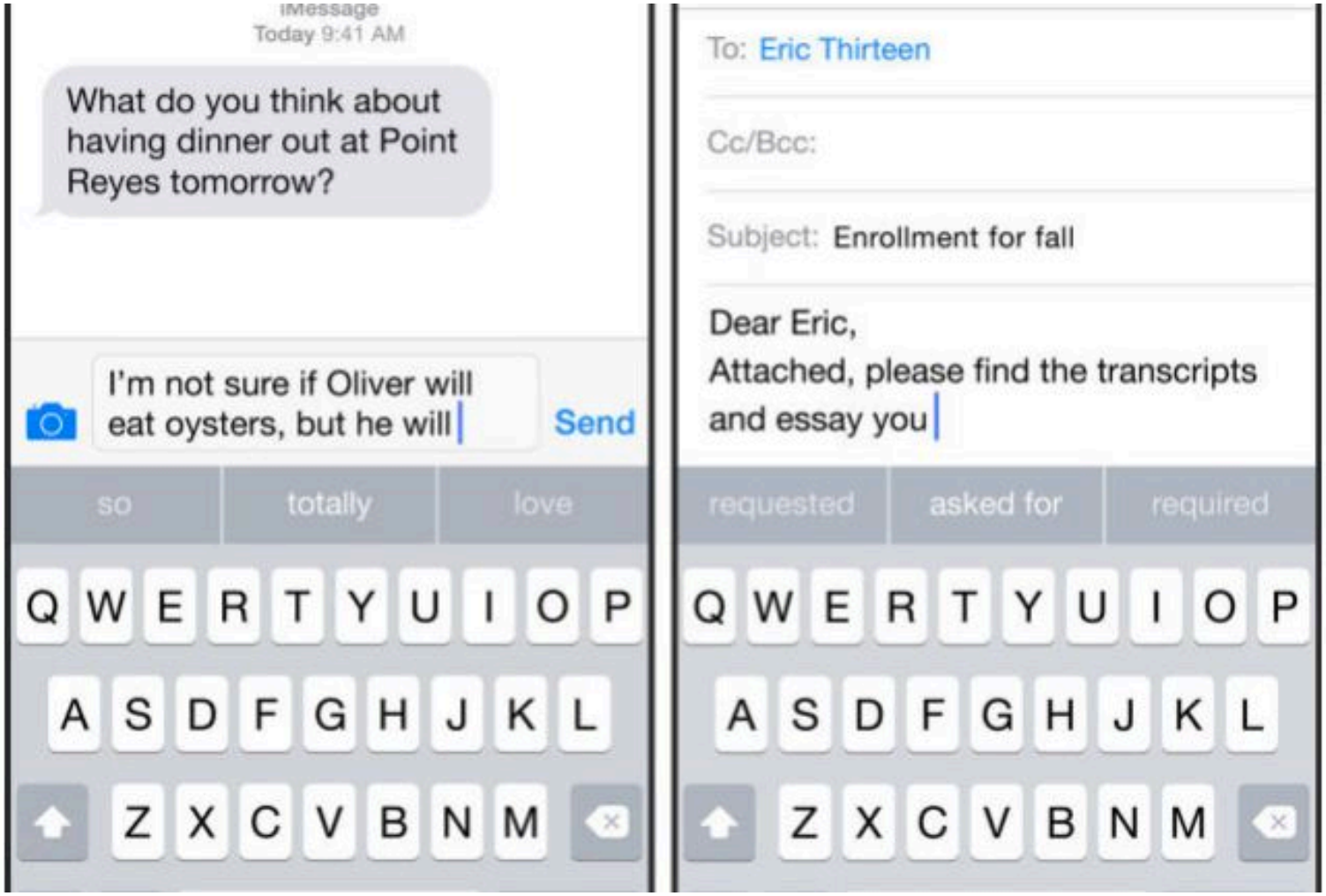
- From *The Boston Globe 1/19/15*
  - A mother …
  - A mother was ….
  - A mother was found ….
  - A mother was found dead …
  - A mother was found dead and her ….
  - A mother was found dead and her son was shot and …
  - A mother was found dead and her son was shot and killed by police early Monday.

Human Word Prediction

- We may have ability to predict future words in an utterance

- How?
  - Domain Knowledge
    - red blood
  - Syntactic knowledge
    - j'ai vu la …<feminine adj | feminine noun>
  - Lexical knowledge
    - baked <potato,beans,cod>

Predictive keyboard



Grammar checker

- The difference is that we don't eat beef.
  The difference are that we don't eat beef.
- The difference is that we don't eat beef.
  The difference is is that we don't eat beef.
- I wish I was there.
  I wish I were there.

Statistical Machine Translation

- Make the translation sound fluent
- Which one sounds the best?
  - he introduced reporters to the main contents of the statement
  - he briefed to reporters the main contents of the statement
  - he briefed reporters on the main contents of the statement

Applications

- LM usually does not stand alone.
  - spelling correction
    - Detect the error (low probability)
    - Correct the error by increasing probability
      - Theatre owners say popcorn/unicorn sales have doubled...
  - speech recognition
    - providing context
      - along with Starbucks lovers
      - a long list of ex loves

Language Model

- Language Model (LM)
  - A language model is a probability distribution over entire sentences or texts
    - N-gram: unigrams, bigrams, trigrams,…

- In a simple *n-gram language model*, the probability of a word, conditioned on some number of previous words

N-grams

- In other words, using the previous N-1 words in a sequence we want to predict the next word
  - Sue swallowed a large green ____.
    - A. frog
    - B. mountain
    - C. car
    - D. pill

What is an N-gram?

- An ***n*-gram** is a subsequence of *n* items from a given sequence.
  - Unigram: *n*-gram of size 1
  - Bigram: *n*-gram of size 2
  - Trigram: *n*-gram of size 3

- Item:
  - Phonemes
  - Syllables
  - Letters
  - Words
  - Anything else depending on the application.

Example

input=the dog smelled like a skunk

- *# = bos and eos*

- Bigram:
  - # the, the dog, dog smelled, smelled like, like a, a skunk, skunk#

- Trigram:
  - "# the dog", "the dog smelled", "dog smelled like", "smelled like a", "like a skunk" and "a skunk #".

How to predict the next word?

- Assume a language has T word types in its lexicon, how likely is word x to follow word y?

- Solutions:
  - Estimate likelihood of x occurring in new text, based on its general frequency of occurrence estimated from a corpus
    - popcorn is more likely to occur than unicorn
  - Condition the likelihood of x occurring in the context of previous words (bigrams, trigrams,…)
    - mythical unicorn is more likely than mythical popcorn

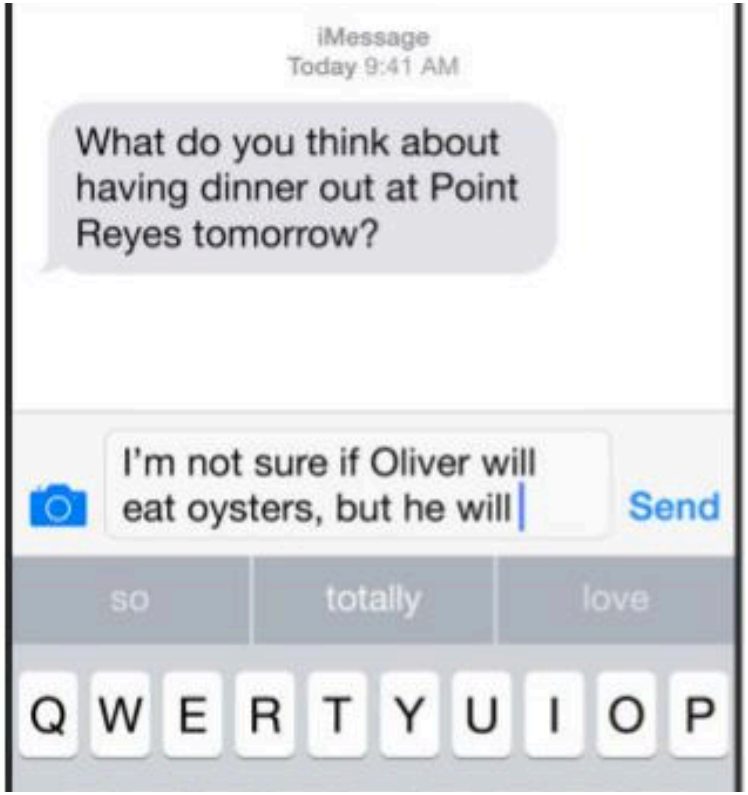
Estimate PDF

- With large enough dataset
  $P(X=x, S=s)$ can be estimated with
  $$\frac{C(X=x, S=s) \text{ in the dataset}}{\text{size of dataset}}$$
- $P(X=x|S=s)$ can be estimated with
  $$\frac{C(X=x, S=s) \text{ in the dataset}}{C(S=s) \text{ in the dataset}}$$
- More examples in lab next week

Predicting the next word from corpora

- P(love| but he will) =

  C(but he will love) / C (he will love)



Statistical View

- The task of predicting the next word can be stated as:
  - attempting to estimate the probability distribution function *P:*

$$P(w_n|w_1,\ldots,w_{n-1})$$

- In other words:
  - we use a classification of the previous history words (or context), *to predict the next word.*
  - On the basis of having looked at a lot of text, we know which words tend to follow other words.

How to assign probabilities to a sequence of words

- Statistical Machine Learning
  - We use language model to give probabilities to the translation outputs.
  - High probability = better translation
- P("he briefed reporters on the main contents of the statement") =

$$\frac{C(\text{"he briefed reporters on the main contents of the statement"})}{C(\text{all possible sequence of words})}$$

- To solve this, we use Chain Rule + Markov Assumption

N-Gram Models

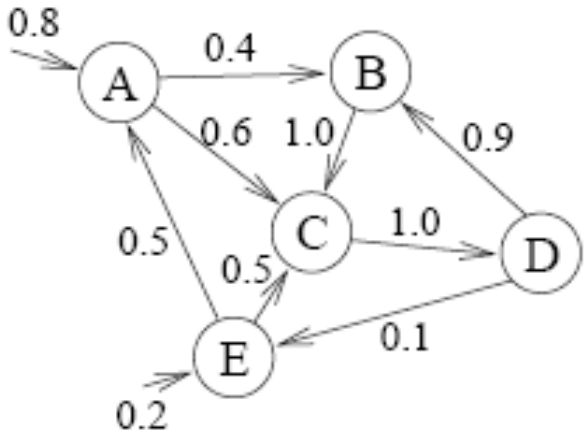
- Models Sequences using the *Statistical Properties* of N-Grams

- Idea: Shannon
  - given a sequence of letters, what is the *likelihood* of the next letter?
  - From training data, derive a *probability distribution* for the next letter given a *history* of size n.

N-gram Model is a Markov Chain

- Give a set of *states, S = {s_1, s_2, … , s_r}.*
  - The process starts in a random state based on a probability distribution
  - Change states in sequence based on a probability distribution of the next state given the previous state

  - This model could generate the sequence {A,C,D,B,C} of length 5 with probability: $0.8 \cdot 0.6 \cdot 1.0 \cdot 0.9 \cdot 1.0 = 0.432$



Markov Assumption

- **Markov assumption**: the probability of the next word depends only on the previous k words.

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

(k+1)-gram or K^{th} order Markov approximation

- Common N-grams:

$$\text{Unigram: } P(w_1\ w_2 \ldots w_n) = P(w_1)\ P(w_2) \ldots P(w_n)$$
$$\text{Bigram: } P(w_1\ w_2 \ldots w_n) = P(w_1)\ P(w_2|w_1) \ldots P(w_n|w_{n-1})$$
$$\text{Trigram: } P(w_1\ w_2 \ldots w_n) = P(w_1)\ P(w_2|w_1) \ldots P(w_n|w_{n-2}\ w_{n-1})$$

Using N-Grams

- For N-gram models
  - $P(w_1,w_2, ...,w_n)$
  - By the Chain Rule we can decompose a joint probability, e.g. $P(w_1,w_2,w_3)$ as follows:

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

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

Example

- Compute the product of component conditional probabilities?
  - P(the mythical unicorn) = P(the) ∗ P(mythical | the) ∗ P(unicorn|the mythical)

- How to calculate these probability?

Training and Testing

- N-Gram probabilities come from a training corpus
  - 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

A Simple Bigram Example

- Estimate the likelihood of the sentence:
  I want to eat Chinese food.
  - 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) P(<end>|food)
- What do we need to calculate these likelihoods?
  - Bigram probabilities for each word pair sequence in the sentence
  - Calculated from a large corpus

Early Bigram Probabilities from BERP

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

The Berkeley Restaurant Project (BeRP) is a testbed for a speech recognition systems developed by the International Computer Science Institute in Berkeley, CA

<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

- 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

- N-gram models can be trained by counting and normalization

Calculating N-gram Probabilities

- Given a certain number of pieces of training data, the second goal is then finding out :
  - how to derive a good probability estimate for the target feature based on these data.

- For our running example of n-grams, we will be interested in $P(w_1 \cdots w_n)$ and the prediction task $P(w_n|w_1 \cdots w_{n-1})$, since:

$$P(w_n|w_1 \cdots w_{n-1}) = \frac{P(w_1 \cdots w_n)}{P(w_1 \cdots w_{n-1})}$$

Maximum Likelihood Estimation

- Given:
  - $C(w_1w_2\ldots w_n)$: the frequency of $w_1w_2\ldots w_n$ in the training text
  - N: the total number of training n-grams:

$$P_{MLE}(w_1w_2\ldots_{wn}) = C(w_1w_2\ldots w_n)/N$$

$$P_{MLE}(w_n|w_1w_2\ldots w_{n-1}) = C(w_1w_2\ldots w_n)/C(w_1w_2\ldots w_{n-1})$$

Early BERP 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

Early BERP Bigram Probabilities

- Normalization: divide each row's counts by appropriate unigram counts for w_{n-1}

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

- Computing the bigram probability of "I I"
  - C(I,I)/C(all I)
  - p (I|I) = 8 / 3437 = .0023

Calculating N-gram Probabilities

- Given a certain number of pieces of training data, the second goal is then finding out :
  - how to derive a good probability estimate for the target feature based on these data.

- For our running example of n-grams, we will be interested in $P(w_1 \cdots w_n)$ and the prediction task $P(w_n|w_1 \cdots w_{n-1})$, since:

$$P(w_n|w_1 \cdots w_{n-1}) = \frac{P(w_1 \cdots w_n)}{P(w_1 \cdots w_{n-1})}$$

Maximum Likelihood Estimation

- Given:
  - $C(w_1w_2\ldots w_n)$: the frequency of $w_1w_2\ldots w_n$ in the training text
  - N: the total number of training n-grams:

$$P_{MLE}(w_1w_2\ldots_{wn}) = C(w_1w_2\ldots w_n)/N$$

$$P_{MLE}(w_n|w_1w_2\ldots w_{n-1}) = C(w_1w_2\ldots w_n)/C(w_1w_2\ldots w_{n-1})$$

N-gram Parameters

- Is a high-order n-gram needed?

P(Most biologists and folklore specialists believe that in fact the mythical unicorn horns derived from the narwhal)

- Issues:
  - The *longer* the sequence, the *less likely* we are to find it in a training corpus
  - The larger the n, the more the number of parameters to estimate

Computing Number of Parameters

- Assume that a speaker is staying within a vocabulary of 20,000 words ,we want to see the estimates for numbers of parameters

Model	Parameters
1st order (bigram model):	$20,000 \times 19,999 = 400$ million
2nd order (trigram model):	$20,000^2 \times 19,999 = 8$ trillion
3th order (four-gram model):	$20,000^3 \times 19,999 = 1.6 \times 10^{17}$

Table 6.1 Growth in number of parameters for *n*-gram models.

- For this reason, *n-gram* systems currently use bigrams or trigrams (and often make do with a smaller vocabulary).

How to Reduce the Number of Parameters?

- Reducing the number of parameters:
  - Stemming
  - Semantic classes

- In practice, it is hard to beat a trigram model for language modeling.

Calculating N-gram Probabilities

- Given a certain number of pieces of training data, the second goal is then finding out :
  - how to derive a good probability estimate for the target feature based on these data.

- For our running example of n-grams, we will be interested in $P(w_1 \cdots w_n)$ and the prediction task $P(w_n|w_1 \cdots w_{n-1})$, since:

$$P(w_n|w_1 \cdots w_{n-1}) = \frac{P(w_1 \cdots w_n)}{P(w_1 \cdots w_{n-1})}$$

Maximum Likelihood Estimation

- Given:
  - $C(w_1w_2\ldots w_n)$: the frequency of $w_1w_2\ldots w_n$ in the training text
  - N: the total number of training n-grams:

$$P_{MLE}(w_1w_2\ldots_{wn}) = C(w_1w_2\ldots w_n)/N$$

$$P_{MLE}(w_n|w_1w_2\ldots w_{n-1}) = C(w_1w_2\ldots w_n)/C(w_1w_2\ldots w_{n-1})$$

Even trigram can be too rare

- What's the count of C("Robin kissed Hulk")?
- It is conceivable but probably shows up in the corpus.

Sparse Data Problem

- MLE is in general unsuitable for statistical inference in NLP because small parameters are hard to estimate.
- The problem is the sparseness of our data (even with the large corpus).
  - The vast majority of words are very uncommon
  - longer *n-grams involving them are thus much rarer*

- The MLE assigns a zero probability to unseen events
  - **Bad …**because the probability of the whole sequences will be zero
    - computed by multiplying the probabilities of subparts

Solution

- how do you handle unseen n-grams?
  - Smoothing
  - Backoff
  - Interpolation

Smoothing

- With MLE, a missing k-gram means zero probability and any longer n-gram that contains the k-gram will also have a zero probability.

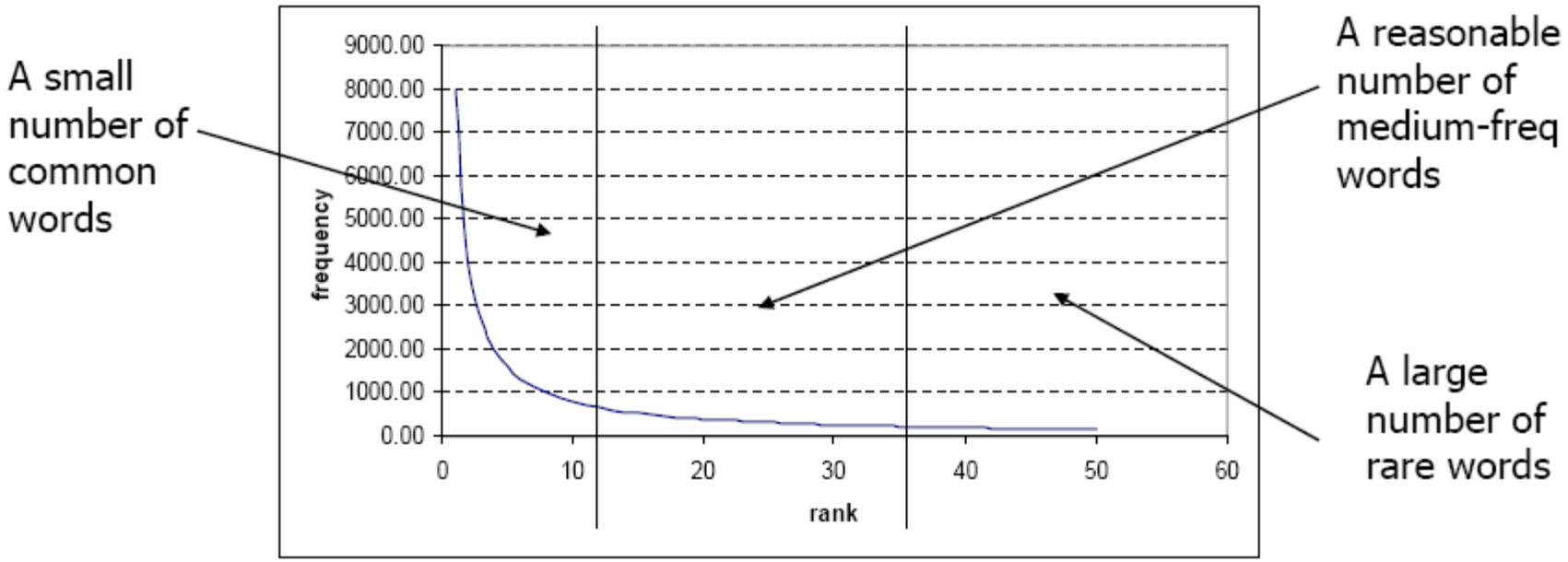
- Words follow a Zipfian distribution
  - no matter how big is a training set, there will always be a lot of rare events that may not be covered.

- Discounting/smoothing techniques:
  - allocate some probability mass for the missing n-grams

Zipf's Law

- Given the frequency *f* of a word and its rank *r* in the list of words ordered:

$f \propto 1/r$ or $f \times r = k$ for a constant k



Laplace Smoothing

- Add 1 to all frequency counts. (not MLE anymore)
- Let V be the vocabulary size

$$P_{Lap}(w_i) = \frac{1+c(w_i)}{V+N}$$

- Bigram:

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

- n-gram:

$$P_{Lap}(w_n|w_1,...,w_{n-1}) = \frac{1+c(w_1,...,w_n)}{V+c(w_1,...,w_{n-1})}$$

Unigram Smoothing Example

- Tiny Corpus, V=4; N=20

$$P_{LP}(w_i) = \frac{c_i + 1}{N + V}$$

Word	True Ct	Unigram Prob	New Ct	Adjusted Prob
eat	10	.5	11	.46
British	4	.2	5	.21
food	6	.3	7	.29
happily	0	.0	1	.04
	20	1.0	~20	1.0

Problem with Laplace Smoothing

- Problem: give too much probability mass to unseen n-grams.

- For sparse sets of data over large vocabularies, such as n-grams, Laplace's law actually gives far too much of the probability space to unseen events.

- Can we smooth more usefully?

Lidstone's Law: Add-λ Smoothing

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

- Need to choose δ
- It works better than add-one, but still works horribly

Other estimators

- Given N-gram hierarchy
  - $P_3(w_3|w_1,w_2)$, $P_2(w_3|w_2)$, $P_1(w_3)$
- Combine these lower-order models
  - Interpolation
- Back off to a lower-order model
  - backoff

Backoff Estimator

- Backoff is an alternative to smoothing for e.g. trigrams
  - try to fill any gap in n-grams at level n by looking 'back' to level n-1
- Example:
  - If a particular trigram ”Robin kissed Hulk" has zero frequency
  - Maybe the bigram ”kissed Hulk" has a non-zero count
- At worst, we back off all the way to unigrams and we do smoothing, if we are still losing

Last time

- What's the purpose of language modeling?
- Why do we like 5-gram model?
- What's the solution to the sparsity problem?

This time

- Backoff
- Interpolation
- Kneser-Ney = Discounting + Interpolating

Stupid Backoff

- Given a trigram xyz
- Backoff and downweight

$$S(z|x,y) = \begin{cases} \frac{count(xyz)}{count(xy)} & \text{if } count(xyz) > 0 \\ \lambda S(z|y) & \text{otherwise} \end{cases}$$

- What's the problem with this approach?

Google NGrams

All Our N-gram are Belong to You

By Peter Norvig - 8/03/2006 11:26:00 AM

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training

```
File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens:     1,024,908,267,229
Number of sentences:     95,119,665,584
Number of unigrams:          13,588,391
Number of bigrams:          314,843,401
Number of trigrams:         977,069,902
Number of fourgrams:      1,313,818,354
Number of fivegrams:      1,176,470,663
```

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

Katz Backoff

- Unlike Stupid Backoff, Katz Backoff figures out the normalization factor (alpha).

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

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

Interpolation

- Consult all models at once but trust the higher order model more.

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

- Ensure valid PDF with the normalization factor $\lambda_1 + \lambda_2 + \lambda_3 = 1$

Kneser-Ney smoothing

- Still state-of-the-art n-gram LM
- It starts with Absolute Discounting

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{C(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

Intuition of KN smoothing

- Lower order model is important only when higher order model is sparse.
- λ should be optimized to perform in such situations

KN Smoothing example

I can't see without my reading _____ .

- *San Francisco* bigram occur a lot.
- *Francisco* ONLY follows *San*
- *C(San Francisco)= C(Francisco) > C(glasses)*
- but *glasses* are in many bigram types.

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{\text{CONTINUATION}}(w_i)$$

New unigram distribution

- How many bigram types end with *Francisco?*
- How many bigram types end with *glasses?*
- Ntypes(?, *Francisco*) < Ntypes(?, *glasses*) these are called continuation count.

$$P_{\text{CONTINUATION}}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}w_i) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}w_j) > 0\}|}$$

Example

- YZ bigram

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{\text{CONTINUATION}}(w_i)$$

- c(.) = # tokens ; cont(.) = # types
- first term = max(c(YZ) - d, 0) / c(Y?)
- second term = cont(?Z) / cont(??)

- lambda(Y?) = d / c(Y?) * cont(Y?)

Putting it all together

- Use recursion to derive trigram and 4-gram and so on

$$P_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}w_i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i|w_{i-n+2}^{i-n})$$

- c_{KN} = actual count for the highest order model
- c_{KN} = continuation count for lower order models
- 5-Gram (modified) KN LM is the state-of-the-art N-gram language model
  - use different d in each recursion step
  - tune for the best d. How do we tune?

Example of trigram continuation count

- XYZ trigram

$$P_{KN}(z|xy) = \frac{\max(c(xyz) - d, 0)}{c(xy)} + \lambda(xy?)P_{KN}(z|y)$$

$$P_{KN}(z|y) = \frac{\max(N(?yz) - d, 0)}{N(?y?)} + \lambda(y?)P_{KN}(z)$$

$$P_{KN}(z) = \frac{\max(N(?z) - d, 0)}{N(??)} + \lambda(?)P_{KN}(base)$$

$$P_{KN}(base) = \frac{1}{N(?)} = \frac{1}{V}$$

More practical issues

- Always use log-probability. Why?
- There are too many n-grams to keep in the memory.
  - prune the entries
  - store integer log probability instead
    - log(0.00005) ≈ -9.9034 ≈ -9
    - what do we lose? what do we gain?

Evaluation

- What is a good LM?
- How do you ***estimate*** how well your LM fits a corpus once you're done?
  - Extrinsic
  - Intrinsic
- Also important for tuning parameter e.g. the discounting factor d in KN.

Data split

- Training set is used for training the model.
  - Collecting n-gram counts
- Development set is used for tuning the model.
  - Trying out different discount factors
  - Trying out trigram vs 4-gram
  - Finding out optimal interpolation weights
- Test set
  - Don't peek. Evaluate on it at the very end.

Extrinsic Evaluation

- The LM is embedded in a wider application. Examples?
- Pros
  - directly estimate the performance gain
- Cons
  - slow
  - specific to the application

Intrinsic Evaluation

- The LM is evaluated directly using some measures:
  - Perplexity
  - Cross-entropy

Perplexity

- If our language model is good at predicting the next word, then we should be less perplexed when we see the next word.
- How much probability does a grammar or language model (LM) assign to the sentences of a corpus, compared to another LM?

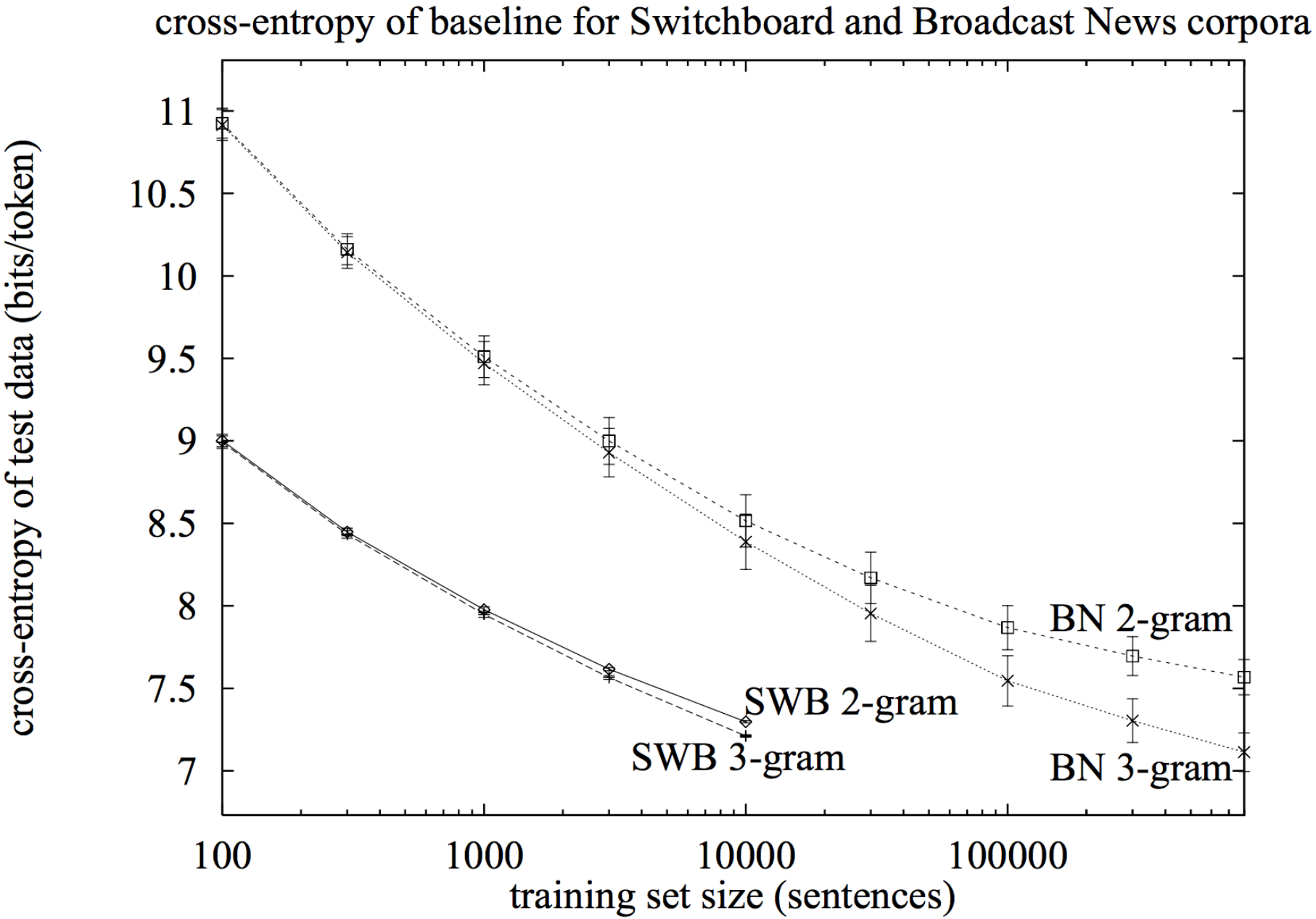
$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

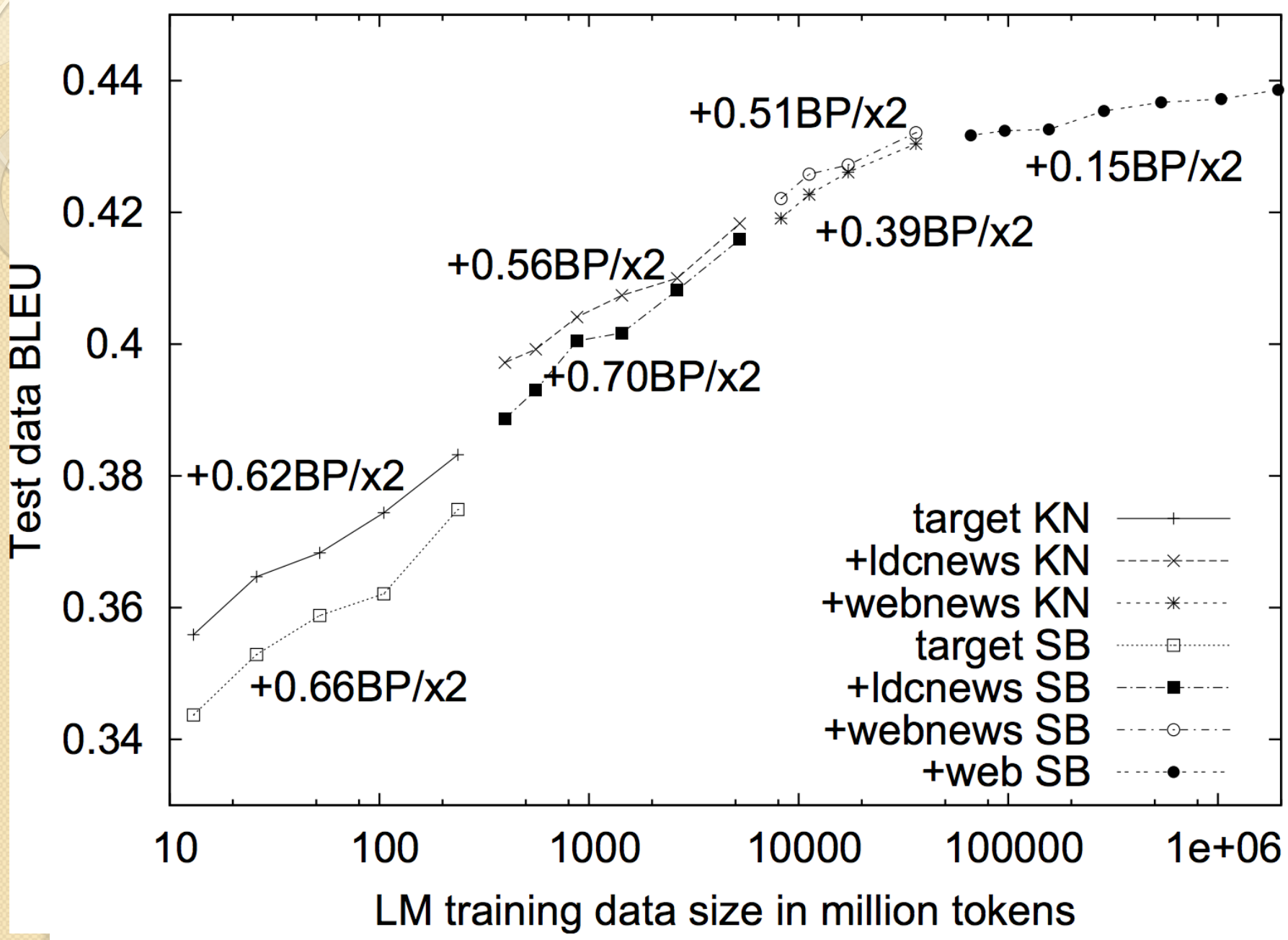
Evaluating using perplexity

- Minimizing perplexity is the same as maximizing probability
  - **Higher probability means lower Perplexity = better LM**
  - **The better the prediction, the lower perplexity**
  - The lower the perplexity, the closer we are to the true underlying language model.

How to train a good LM?

- Higher order is usually the better, but you must have enough data to train it.
- Sophisticated model vs More data
- Backoff vs Interpolation


cross-entropy of baseline for Switchboard and Broadcast News corpora
cross-entropy of test data (bits/token)
11
10.5
10
9.5
9
8.5
8
7.5
7
100
1000
10000
100000
training set size (sentences)
BN 2-gram
BN 3-gram
SWB 2-gram
SWB 3-gram

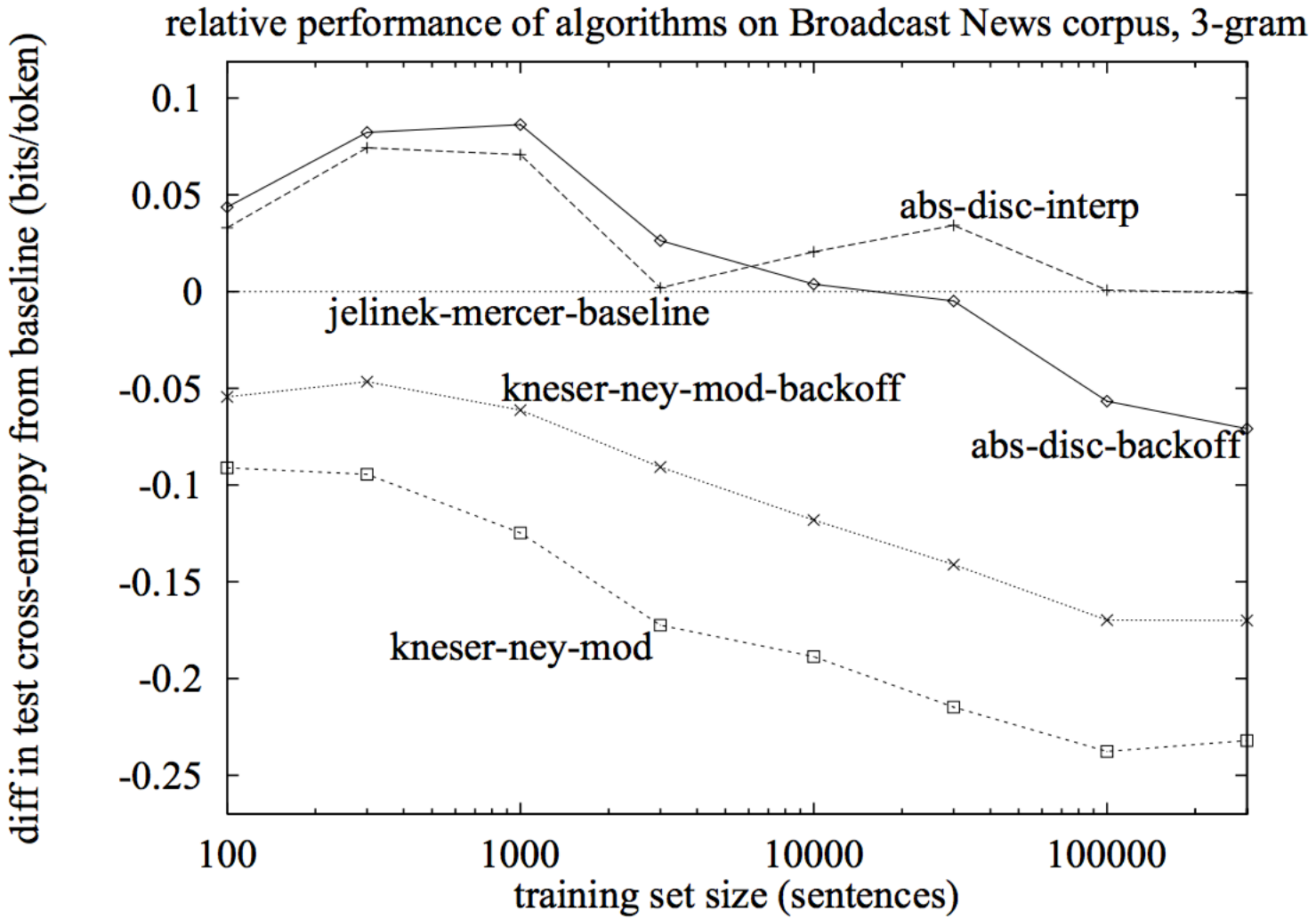

Test data BLEU
0.44
0.42
0.4
0.38
0.36
0.34
10
100
1000
10000
100000
1e+06
LM training data size in million tokens
+0.51BP/x2
+0.15BP/x2
+0.39BP/x2
+0.56BP/x2
+0.70BP/x2
+0.62BP/x2
+0.66BP/x2
target KN
+ldcnews KN
+webnews KN
target SB
+ldcnews SB
+webnews SB
+web SB

Modeling vs Data

- If we have tons and tons of data, we can get away with a stupid model.
- But can't we just always dump in more data?

	target	*webnews*	*web*
# tokens	237M	31G	1.8T
vocab size	200k	5M	16M
# n-grams	257M	21G	300G
LM size (SB)	2G	89G	1.8T
time (SB)	20 min	8 hours	1 day
time (KN)	2.5 hours	2 days	–
# machines	100	400	1500

Table 2: Sizes and approximate training times for 3 language models with Stupid Backoff (SB) and Kneser-Ney Smoothing (KN).


relative performance of algorithms on Broadcast News corpus, 3-gram
diff in test cross-entropy from baseline (bits/token)
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
100
1000
10000
100000
training set size (sentences)
abs-disc-interp
jelinek-mercer-baseline
kneser-ney-mod-backoff
abs-disc-backoff
kneser-ney-mod

Take-away messages

- LMs assign probabilities to a sequence of words.
- N-gram language model considers limited size of context (up to n-1 previous words)
- Data sparsity makes things hard.
  - smoothing, backoff, and interpolation
  - dumping in more data

Example

- Example in terms of word-level perplexity:
  - The perplexity of a text with respect to the word "ice“:
    - Represents the number of words which could follow it.
    - There are some given number of words which **may follow** the word "ice" (cream, age, and cube),
    - similarly, some words which would most likely **not follow** "ice" (nuclear, peripheral, and happily).
    - **Since the number of possible words which could follow "ice" is relatively high, its perplexity is high**. Why?

- Automating Authorship attribution
  - Problem:
    - identifying the author of an anonymous text, or text whose authorship is in doubt
    - Plagiarism detection
  - Common approaches:
    - Determine authorship using stylistic analysis
    - Based on lexical measures representing:
      - the richness of the author's vocabulary
      - the frequency of common word use
    - Two steps:
      - style markers
      - classification procedure

- Disadvantage:
  - Techniques used for style marker extraction are almost always language dependent
  - feature selection is not a trivial process,
  - involves setting thresholds to eliminate uninformative features
  - perform their analysis at the word level
- New Approach:
  - a byte-level n-gram author profile of an author's writing.
    - choosing the optimal set of n-grams to be included in the profile,
    - calculating the similarity between two profiles.

References

- Manning, C.D., and Schutze, H. (1999). Foundations of Statistical Natural Language Processing, MIT Press, Cambridge, MA.
- Julia Hirschberg, n-grams and corpus linguistics, Power point presentations, *www.cs.columbia.edu/~julia/courses/CS4705/**ngrams**.ppt (Accessed November 14, 2010)*
- Julia Hockenmaier, Probability theory, N-gram and Perplexity, Powerpoint slides, http://www.cs.uiuc.edu/class/fa08/cs498jh/Slides/Lecture3.pdf *(Accessed November 14, 2010)*
- Probability and Language Model, *www.cis.upenn.edu/~cis530/**slides**-2010/**Lecture2**.ppt, (Accessed November 14, 2010)*
- Vlado Keselj, Fuchun Peng, Nick Cercone, and Calvin Thomas. (2003). N-gram-based Author Profiles for Authorship Attribution, In *Proceedings of the Conference Pacific Association for Computational Linguistics, PACLING'03*, Dalhousie University, Halifax, Nova Scotia, Canada, pp. 255-264, August 2003.