Lecture 4
N-grams

CS 6320

Outline

- N-Grams
- Smoothing
- Good-Touring Discounting
- Backoff
- Evaluating Language Models
- Perplexity's relation to Entropy

N-Grams 1/11

- Up until now we've concentrated on words in isolation.
- Now we are going to look at a word in context.
- The task: predict the next word in the sequence.
- Useful for several NLP applications; main feature of a language model.

N-Grams 2/11

- Context—sensitive spelling error correction:

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

The study was conducted mainly *be* John Black

The design *an* construction of the system will take more than a year.

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

Cant hey *lave* him my messages?

I need to *notified* the bank of [this problem.]

He is trying to *fine* out.

Some attested real-word spelling errors from Kukich (1992)

Solution: Compute the likeliness of a sequence

N-Grams 3/11

- Example of text

  Guessing the next..??

  Guessing the next word turns out to..??

  Guessing the next word turns out to be closely related to another..??

  Guessing the next word turns out to be closely related to another problem: computing..?

  Guessing the next word turns out to be closely related to another problem: computing the probability of a sequence of words.

N-Grams 4/11

- Ngram model uses the previous N-1 words to predict the next one.
- It seems that we need:
  - Domain knowledge
  - Syntactic knowledge
  - Lexical knowledge
- Actually done with simple statistical techniques.

N-Grams 5/11

- Definitions:
  - wordform: words as appear in the corpus.
  - lemma: a set of lexical forms having the same stem, the same part of speech, and the same word sense. *Example: cat, cats.*
  - types: number of distinct words in a corpus, i.e. the size of the vocabulary.
  - tokens: the total number of running words.

N-Grams 6/11

- Probability of "The big dog" is P(*The^ big^dog*).

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

$$P(A \wedge B) = P(B \mid A)P(A)$$

$$P(The \wedge dog) = P(dog \mid The)P(The)$$

Unigrams: P(dog)

Bigrams: P(dog|big)

Trigrams: P(dog|the big)

Quadrigrams: P(dog|the big fat)

N-Grams 7/11

- Without any context, the probability for next word is:

$$\frac{1}{number\ of\ \ word\ types}$$

- A better approximation is to consider the conditional probability of a word given the previous words.

$$w_1 \ldots w_n \ as \ w_1^n$$

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1})$$

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

- How to compute $P(w_k \mid w_1^{k-1})$?

N-Grams 8/11

- Bigrams model - approximates the probability of a word given all the previous words by the conditional probability of the preceding word.

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

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

- Ngram model - approximates the probability of a word given all the previous words by considering only the conditional probability of the previous N-1 words.

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

N-Grams 9/11

	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

Figure 4.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

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

Unigram Counts

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

N-Grams 10/11

- Berkley Restaurant Project

  It is captured:

$$P(want \mid I) = 0.33$$
$$P(to \mid want) = 0.66$$
$$P(eat \mid to) = 0.28$$
$$P(food \mid Chinese) = 0.52$$
$$P(lunch \mid eat) = 0.056$$

  Observations:

$P(I \mid I)$	I I I want
$P(I \mid want)$	I want I want
$P(I \mid food)$	the kind of food I want

N-Grams 11/11

- Shakespeare produced:

  884 647 tokens

  29,000 wordform types

  300,000 bigram types

- Out of 844,000,000 possible bigrams 99.96% of possible bigrams were never used.

Smoothing 1/2

- The bigram matrix is sparse.
- Ngrams tend to underestimate the probability of strings due to the limited size of corpus.
- Smoothing is the task of reevaluating some of the zero-probability and lowprobability.
- Ngrams and assigning them nonzero values.

Smoothing 2/2

- For unigrams

$$P(w_x) = \frac{C(w_x)}{N}$$

C – count (number) of words in a type

N – no. of word tokens

$$\sum_{i=1}^{v} C_i = N$$

$$\sum_{i=1}^{v} C_i^* = N$$

Add-One Smoothing 1/4

C_i - nr of counts for type i

$$C_i^* = (C_i + 1) \frac{N}{N+V}$$

Why?

$$C_i^* N + C_i^* V = C_i N + N$$

take $\sum_{i=1}^{V}$

$$\sum_{i=1}^{V} C_i^* = (\sum_{i=1}^{V} C_i + \sum_{i=1}^{V} 1) \frac{N}{N+V}$$

$$N = (N + V) \frac{N}{N+V}$$

Add-One Smoothing 2/4

Add-One Smoothing

$$C_i^* = (C_i + 1)\frac{N}{N+V}$$

$$P_i^* = \frac{(C_i + 1)}{N+V}$$

For bigrams:

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

Add-One Smoothing 3/4

	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

Figure 4.5 Add-one smoothed bigram counts for eight of the words (out of V = 1446) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

V=1446

I	2533 + 1446 =3979
want	927 + 1446 = 2373
to	2417 + 1446 = 3863
eat	746 + 1446 = 2192
Chinese	158 + 1446 = 1604
food	1093 + 1446 = 2539
lunch	341 + 1446 = 1787
spend	278 + 1446 = 1724

Add-One Smoothing 4/4

	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

Figure 4.6 Add-one smoothed bigram probabilities for eight of the words (out of V = 1446) in the BeRP corpus of 9332 sentences. Previously-zero probabilities are in gray.

	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

Figure 4.7 Add-one reconstituted counts for eight of the words (out of V = 1446) in the BeRP corpus of 9332 sentences. Previously-zero probabilities are in gray.

Add-One Smoothing

- Problems with AddOne Smoothing
  - Too much probability mass is moved to all the zeroes.
  - The value "1" was picked arbitrarily.

- Idea: Add smaller values to the counts "addonehalf", "addone thousandth".

Good – Turing Discounting

- Idea:
  - Assign N-grams to buckets based on their frequency and order them (i.e. produce a histogram).
  - Re-estimate the amount of probability mass to assign to N-grams with zero or low counts by looking at the number of N-grams with higher counts.

- Nc nr. of N-grams that occur C times. (frequency of frequency of C).

Good – Turing Discounting

- Nc - the number of N-grams that occur c times.

$$N_c = \sum_{X:count\ (x)=c} 1$$

- Good – Turing smoothing estimates the probability of N-grams that occur c times by the probability of N-grams that occur c + 1 times in the corpus.

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

For N_0 –

$$P^*_{GT}(things\ with\ frequency\ zero\ in\ training) = \frac{N_1}{N}$$

Example

In a lake there are 8 species:

bass, carp, catfish, eel, perch, salmon, trout, whitefish

Assume a sample of fish shows

10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon 1 eel, 0 bass, 0 catfish

What is the probability the next catch is bass or catfish?

$$P^*_{GT} = \frac{N_1}{N} = \frac{3}{18}$$

Example

What is the probability next fish is another trout?

$$c^*\,(trout) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = 0.67$$

$$P^*(trout) = \frac{C^*}{N} = \frac{.67}{18} = 0.037$$

	Unseen (Bass or Catfish)	Trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT P^*_{GT}	$P^*_{GT}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$P^*_{GT}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Good-Turing Discounting

	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

Figure 4.8 Bigram "frequencies of frequencies" and Good-Turing re-estimations for the 22 million AP bigrams from Church and Gale (1991) and from the Berkeley Restaurant corpus of 9332 sentences.

Additional Complexities

1. Cases when $N_{c+1} = 0$

- Solution: Smooth N_c counts to replace any zeros in the sequence.

$$\log(N_c) = a + b \log(c)$$

2. When c is large – no need for smoothing c > k. Eg k=5

$$c^* = c \text{ for } c > k = 5$$

Katz introduced equation

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \text{for } 1 \leq c \leq k.$$

3. Treat N-grams with low rate counts as if the counts were 0. Apply Good-Turing to these, and then use smoothing.

Interpolation

Idea: combine different order N-grams by linearly interpolating all the models

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

Such that

$$\sum_i \lambda_i = 1$$

A more accurate approximation is

$$\widehat{P}(w_n|w_{n-2}\ w_{n-1}) = \lambda_1 (w_{n-2}^{n-1})\ P(w_n|w_{n-2}w_{n-1}) + \lambda_2 (w_{n-2}^{n-1})\ P(w_n|w_{n-1}) + \lambda_3 (w_{n-2}^{n-1})\ P(w_n)$$

Problem is how to set up λ values.

By training.

Backoff

- Idea: If we have non-zero trigram counts use only trigram counts. Only back off to a lower-order N-gram when counts for a higher-order N-gram are zero.
- Note that interpolation always mixes N-gram estimates.

  Katz back-off with Good Turning discounting
  - If N-gram has zero counts, approximate by backing off to the (N-1)-gram. Continue backing off until reach an order that has some counts.

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

Backoff

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

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

Combine Backoff with Discounting

- Use the discounting algorithm to tell how much total probability mass to set aside for all events we have not seen, and the backoff algorithm to tell how to distribute this probability in a clever way.

Evaluating Language Models

- Extrinsic evaluation
  - Use language model as part of an application and measure overall performance

- Intrinsic evaluation
  - Measure performance of a model independent of any application
  - Use a training corpus to train the model, then measure performance on a test set

Perplexity

- Perplexity is a probabilistic-based metric for language models
- Definition:

  Perplexity (PP) of a language model on a test set is the reverse probability of the test set normalized be the number of words

  For $W = w_1 w_2 \dots w_n$

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_{2\dots} w_N)}}$$

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

  Bigram perplexity

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

Perplexity

- Example

  Train set 38 million words from WSJ.

  Word vocabulary 19,979 words

  Test set 1.5 million words

	Unigram	Bigram	Trigram
Perplexity	962	170	109

Conclusion: The more information the n-gram model gives about the word sequence, the lower the perplexity.

Note: Perplexity is related to entropy.

Perplexity as a branching factor

- Branching factor of a language is the number of possible next words that can follow any word.
  - Example-

    Consider words with equal probability

    For example digits

    $W \in (zero, one, \ldots, nine)$

$$P(w) = \frac{1}{10}$$

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \left(\left(\frac{1}{10}\right)^N\right)^{-\frac{1}{N}} = \left(\frac{1}{10}\right)^{-1} = 10$$