Statistical Inference:
n-gram Models over Space Data

Chia-Hao Lee

Reference: 1.Foundation of Statistical Natural Language Processing
2.The Projection of Professor Berlin Chen
3.A Bit of Process in Language Modeling Extended Version, Joshua T. Goodman,2001

outline

- N-gram
- MLE
- Smoothing
  - Add-one
  - Lidstone’s Law
  - Witten-Bell
  - Good-Turing
- Back-off Smoothing
  - Simple linear interpolation
  - General linear interpolation
  - Katz
  - Kneser-Ney
- Evaluation

Introduction

- Statistical NLP aims to do statistical inference for the field of natural language.

- In general, statistical inference consists of taking some data and then making some inferences about this distribution.
  - Use to predict prepositional phrase attachment

- A running example of statistical estimation : language modeling

Reliability vs. Discrimination

- In order to do inference about one feature, we wish to find other features of the model that predict it.
  - Stationary model

- Based on various classificatory, we try to predict the target feature.

- We use the equivalence classing to help predict the value of the target feature.
  - Independence assumptions: Features are independent

Reliability vs. Discrimination

- The more classificatory features that we identify, the more finely conditions that we can predict the target feature.
- Diving the data into many bins gives us greater discrimination.
- Using a lot of bins, a particular bin may contain no or a very small number of training instances, and we can not do statistical estimation.
- Is there a good compress between two criteria??

N-gram models

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

$$P\left(w_n \middle| w_1, \cdots, w_{n-1}\right)$$

- History: classification of the previous words

- Markov assumption: only the last few words affect the next word

- The same $n-1$ words are placed in the same equivalence class:
  - ($n-1$) order Markov model or $n-gram$ model

N-gram models

- Naming:
  - *gram* is a Greek root and so should be put together with number Greek prefix
  - Shannon actually did use the term *digram*, but this usage has not survived now.
  - Now we always use *bigram* instead of *digram*.

N-gram models

- For example:

  She swallowed the large green ___ .

  - “swallowed” influence the next word more stronger than “the large green ___ “.

- However, there is the problem that if we divide the data into too many bins, then there are a lot of parameters to estimate.

N-gram models

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

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

N-gram models

- *Five-gram* model that we thought would be useful, may well not be practical, even if we have a very large corpus.
- One way of reducing the number of parameters is to reduce the value of n.
- Removing the inflectional ending from words
  - Stemming
- And grouping words into semantic classes
- Or …(ref. Ch12,Ch14)

Building n-gram models

- Corpus: Jane Austen’s novel
  - Freely available and not too large

- As our corpus for building models, reserving *Persuasion* for testing
  - *Emma, Mansfield Park, Northanger Abbey, Pride and Prejudice (傲慢與偏見), and Sense and Sensibility*

- Preprocessing
  - Remove punctuation leaving white-space
  - Add SGML tags <s> and </s>

- N=617,091 words , V=14,585 word types

Statistical Estimators

- Find out how to derive a good probability estimate for the target feature, using the following function:

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

- Can be reduced to having good solutions to simply estimating the unknown probability distribution of *n-grams* . (all in one bin, with no classificatory features)
  - bigram: $\underline{h_1 a}$, $\underline{h_2 a}$, $\underline{h_3 a}$, $\underline{h_4 b}$,$\underline{h_5 b}$…reduce to a and b

Statistical Estimators

- We assume that the training text consists of N words.

- We append $n-1$ dummy start symbols to the beginning of the text.
  - N n-gram with a uniform amount of conditioning available for the next word in all cases

N	Number of training instances
B	Number of values in the multinomial target feature distribution
V	Vocabulary size
w_{1n}	An n-gram $w_1 \cdots w_n$ in the training text
$C(w_1 \cdots w_n)$	Frequency of n-gram $w_1 \cdots w_n$ in training text
r	Frequency of an n-gram
$f(\cdot)$	Frequency estimate of a model
N_r	Number of target feature values seen r times in training instances
T_r	Total count of n-grams of frequency r in further data
h	'History' of preceding words

Table 6.2 Notation for the statistical estimation chapter.

Maximum Likelihood Estimation

- MLE estimates from relative frequencies.
  - Predict: *comes across* __?__
  - Using trigram model: 10 instances (*trigrams*)
  - Using relative frequency:

$$P(as) = 0.8, \quad P(more) = 0.1, \quad P(a) = 0.1, \quad P(x) = 0$$

$$P_{MLE}(w_1, \cdots, w_n) = \frac{C(w_1, \cdots, w_n)}{N}$$

$$P_{MLE}(w_n | w_1, \cdots, w_{n-1}) = \frac{C(w_1, \cdots, w_n)}{C(w_1, \cdots, w_{n-1})}$$

Smoothing

- Sparseness
  - Standard N-gram models is that they must be trained from some corpus.
  - Large number of cases of putative ‘zero probability’ n-gram that should really have some non-zero probability.

- Smoothing
  - Reevaluating some zero or low probability in $n\text{-}gram$.

Laplace's law

- To solute the failure of the MLE, the oldest solution is to employ Laplace's law (also called add-one) :

$$P_{Lap}(w_1,\cdots,w_n)=\frac{C(w_1,\cdots w_n)+1}{N+B}$$

- 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.

Add-One Smoothing

- Take counts before normalize

- Unigram MLE : (ordinary)

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

- The probability estimate for an $n\text{-}gram$ seen r times is

$P_r(w_i) = \frac{(r+1)}{(N+B)}$. (using add-one)

- So, the frequency estimate becomes $f_r(w_i) = N\frac{(r+1)}{(N+B)}$

Add-One Smoothing

- The alternative view : discounting
  - Lowing some non-zero counts that will be assigned to zero counts.

$$d_c = 1 - \frac{C^*}{C}$$

Add-One Smoothing

- Unigram example :

$$V = \{A, B, C, D, E\} \qquad |V| = 5$$

$$S = \{A, A, A, A, A, B, B, B, C, C\} \quad , \quad N = |S| = 10$$

$$5\ for\ 'A', \quad 3\ for\ 'B', \quad 2\ for\ 'C', \quad 0\ for\ 'D', 'E'$$

$$P(A) = \frac{(5+1)}{(10+5)} = 0.4 \qquad P(C) = \frac{(2+1)}{(10+5)} = 0.2$$

$$P(B) = \frac{(3+1)}{(10+5)} = 0.27 \qquad P(D) = P(E) = \frac{(0+1)}{(10+5)} = 0.067$$

Add-One Smoothing

- Bigram MLE :

$$P\left(w_n \middle| w_{n-1}\right) = \frac{C\left[w_{n-1} w_n\right]}{C\left[w_{n-1}\right]}$$

- Smoothed :

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

Add-One Smoothing : Example

For example : Bigram

	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

Figure 6.4 Bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ~10,000 sentences.

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

N (want)=1215

N (want,want)=0

N (want,to)=768

Add-One Smoothing :Example

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

Figure 6.5 Bigram probabilities for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ~10,000 sentences.

P (want|want)=0/1215=0

P (to|want)=786/1215=0.65

Add-One Smoothing :Example

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

Figure 6.7 Add-one smoothed bigram probabilities for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ~10,000 sentences.

P'(want|want)=(0+1)/(1215+1616)=0.00035

P'(to|want)=(786+1)/(1215+1616)=0.28

Add-One Smoothing

- P(want|want) changes from 0 to 0.00035
- P(to|want) changes from 0.65 to 0.28

- The sharp change occurs because too much probability mass is moved to all the zero.

- Gale and Church summarize add-one smoothing is worse at predicting the actual probability than unsmoothed MLE.

Lidstone's Law and Jeffreys-Perks Law

- Lidstone's Law :
  - Add some normally smaller positive value λ

$$P_{Lid}(w_1,\cdots,w_n)=\frac{C(w_1,\cdots,w_n)+\lambda}{N+B\lambda}$$

- Jeffreys-Perks Law:
  - Viewed as linear interpolation between MLE and a uniform prior
  - Also called ' *Expected Likelihood Estimation* '

$$P_{Lid}(w_1,\cdots,w_n)=\mu\frac{C(w_1,\cdots,w_n)}{N}+(1-\lambda)\frac{1}{B}=\frac{C(w_1,\cdots,w_n)+\lambda}{N+B\lambda}$$

$$\text{where : }\mu=\frac{N}{N+B\lambda}$$

Held out Estimation

- A cardinal sin in Statistical NLP is to test on training data. Why??
  - Overtraining
  - Models memorize the training text

- Test data is independent of the training data.

Held out Estimation

- When starting to work with some data, one should always separate it into a training portion and a testing portion.
  - Separate data immediately into training and test data(5~10%,reliable).
  - Divide training and test data into two again
  - Held out (validation) data(10%)
    - Independent of primary training and test data
    - Involve many fewer parameters
    - Sufficient data for estimating parameters

- Research:
  - Write an algorithm, train it and test it (X)
    - Subtly probing
  - Separate to Development test set, final test set (O)

Held out Estimation

- How to select test/held out data?
  - Randomly or aside large contiguous chunks

- Comparing average scores is not enough
  - Divide the test data into several parts
  - t-test

Held out Estimation : t-test

	System 1	System 2
Score	71,61,55,60,68,49, 42,72,76,55,64	42,55,75,45,54,51, 55,36,58,55,67
Total	673	593
n	11	11
mean $\overline{x_i}$	61.2	53.9
$s_i^2 = \sum (x_{ij} - \overline{x}_i)^2$	1081.6	1186.9
df	10	10

Pooled $s^2 = \frac{1{,}081.6 + 1{,}186.9}{10 + 10} \approx 113.4$ $\quad t = \frac{\overline{x_1} - \overline{x_2}}{\sqrt{\frac{2s^2}{n}}} = \frac{61.2 - 53.9}{\sqrt{\frac{2 \times 113.4}{11}}} \approx 1.60$

t=1.60<1.725, the data fail the significance test.

Held out Estimation

- The held out estimator : (for n-grams)

$$C_1(w_1 \cdots w_n) = \text{frequency of } \mathrm{w}_1 \cdots \mathrm{w}_\mathrm{n} \text{ in training data}$$

$$C_2(w_1 \cdots w_n) = \text{frequency of } \mathrm{w}_1 \cdots \mathrm{w}_\mathrm{n} \text{ in held out data}$$

$$T_r = \sum_{\{w_1 \cdots w_n : C_1(w_1 \cdots w_n) = r\}} C_2(w_1 \cdots w_n)$$

T_r :the total number of times that all $n\text{-}grams$ (that appeared r times in the training text) appeared in the held out data

- The probability of one of these n-grams :

$$P_{ho}(w_1 \ldots w_n) = \frac{T_r}{N_r N}$$

Cross-validation

- Dividing training data into two parts.
  - First, estimates are built by doing counts on one part.
  - Second, we use the other pool of held out data to refine those estimates.

- Two-way cross-validation
  - delete estimation

$$P_{del}(w_1, \cdots, w_n) = \frac{T_r^{01} + T_r^{10}}{N(N_r^0 + N_r^1)}$$

N_r^0 : the number of n-grams occurring r times in the 0^{th} part of the training data.
T_r^{01} : the total occurrences of those bigrams from part 0 in the 1^{th} part.

Cross-validation

- Leaving-one-out
  - Training corpus : $N-1$
  - Held out data : 1
  - Repeated N times

Witten-Bell Discounting

- A much better smoothing method that is only slightly more complex than add-one.
- Zero-frequency word or N-gram as one that just hasn't happened.
  - can be modeled by probability of seeing an $n\text{-}gram$ for the first time
- Key : **things seen once !**
- The count of 'first time' n-grams is just for the number of n-gram types w saw in data.

Witten-Bell Discounting

- Probability of total unseen (zero) N-grams :

$$\sum_{i:C_i=0} p_i^* = \frac{T}{N+T}$$

  - T is the type we have already seen
  - T differs from V (V is total types we might see)

- Divide up to among all the zero N-grams
  - Divided equally

$$p_i^* = \frac{T}{Z(N+T)}$$

where : $Z = \sum_{i\ C_i=0} 1$ (number of n-gram types with zero-counts)

Witten-Bell Discounting

- Discounted probability of the seen *n−grams*

$$p_i^* = \frac{c_i}{N+T} \quad if \quad c_i > 0$$

(c_i : the count of a seen *n−gram* i)

- Another formulation (in term of frequency count)

$$c_i^* = \begin{cases} \frac{T}{Z}\frac{N}{N+T} & ,if \quad c_i = 0 \\ c_i \frac{N}{N+T} & ,if \quad c_i > 0 \end{cases}$$

Witten-Bell Discounting

- For example (of unigam modeling)

$$V = \{A, B, C, D, E\} \qquad |V| = 5$$

$$S = \{A, A, A, A, A, B, B, B, C, C\} \ , \quad N = |S| = 10$$

$$5\ for\ 'A', \quad 3\ for\ 'B', \quad 2\ for\ 'C', \quad 0\ for\ 'D', 'E', \quad T = |\{A, B, C\}| = 3, \quad Z = 2$$

$$P(A) = \frac{5}{(10+3)} = 0.385 \qquad P(C) = \frac{2}{(10+3)} = 0.154$$

$$P(B) = \frac{3}{(10+3)} = 0.23 \qquad P(D) = P(E) = \frac{3}{(10+3)} * \frac{1}{2} = 0.116$$

Witten-Bell Discounting

- Bigram
  - Consider bigrams with the history word w_x .
    - for zero-count bigram (with w_x as the history)

$$\sum_{i:C(w_x,w_i)=0} p^*(w_i|w_x) = \frac{T(w_x)}{C(w_x)+T(w_x)}$$

$$p^*(w_i|w_x) = \frac{T(w_x)}{Z(w_x)(C(w_{i-1})+T(w_{i-1}))}$$

  - $C(w_x)$: frequency count of word w_x in the corpus
  - $T(w_x)$: types of nonzero-count bigrams (with w_x as the history)
  - $Z(w_x)$: types of zero-count bigrams (with w_x as the history)

$$Z(w_x) = \sum_{i:C(w_x w_i)=0} 1$$

Witten-Bell Discounting

- Bigram
  - For nonzero-count bigram

$$\sum_{i:C(w_x w_i)>0} p^*(w_i|w_x) = \frac{C(w_x w_i)}{C(w_x)+T(w_x)}$$

Good-Turing estimation

- A method is slightly more complex than Witten-Bell.
  - To re-estimate zero or low counts by higher counts

- Good-Turing estimation :
  - For any n-gram, that occurs r times, we pretend it occurs r^* times:

$$r^* = (r+1)\frac{n_{r+1}}{n_r}\ ,$$ A new frequency count

n_r : the number of n-grams that occurs exactly r times in the training data

  - The probability estimate for a n-gram

$$P_{GT}(x) = \frac{r^*}{N}$$

N : the size of the training data

Good-Turing estimation

- The size (word counts) of the training data remains the same
  - Let $\sum_{r=1}^{\infty} r \cdot n_r = N$

$$\tilde{N} = \sum_{r=0}^{\infty} r^* \cdot n_r = \sum_{r=0}^{\infty} (r+1) \cdot n_{r+1} = \sum_{r'=1}^{\infty} r' \cdot n_{r'} = N \qquad (set \quad r' = r+1)$$

- Unseen: N_1/N, why?

  $0^*=(0+1)*N_1/N_0$ and number of zero frequency words: N_0

  So, the probability = $((N_1/N_0)*N_0)/N= N_1/N$ (MLE)

Good-Turing estimation : Example

- Imagine you are fishing. You have caught 10 Carp (鯉魚), 3 Cod (鱈魚), 2 tuna (鮪魚), 1 trout (鱒魚), 1 salmon (鮭魚), 1 eel (鰻魚)
- How likely is it that next species is now?
  - $P_0=n_1/N=3/18=1/6$
- How likely is eel ? 1*
  - $n_1=3$, $n_2=1$
  - $1^*=(1+1) \times 1=2/3$
  - $P(eel)=1^*/N=(2/3)/18=1/27$
- How likely is tuna? 2*
  - $n_2=1$, $n_3=1$
  - $2^*=(2+1) \times 1/1=3$
  - $P(tuna)=2^*/N=3/18=1/6$
- But how likely is Cod? 3*
  - Need a smoothing for n_4 in advance

Good-Turing estimation

- The problem of Good-Turing estimate is that when $n_{r+1}=0$ and $P(r^*)>P((r+1)^*)$
  - The choice of k may be overcome the second problem.
  - Experimentally $4 \leqq k \leqq 8$ (Katz), Parameter k,$N_{k+1} \neq 0$

$$\hat{P}_{GT}(a_k) < \hat{P}_{GT}(a_{k+1}) \;\Rightarrow\; (k+1) \cdot n_{k+1}^2 - n_k \cdot n_{k+2}(k+2) < 0$$

Combining Estimators

- Because of the same estimate for all $n\text{-}grams$ that never appeared, we hope to produce better estimates by looking at $(n\text{-}1)\text{-}grams$.

- Combine multiple probability estimates from various different models.
  - Simple linear interpolation
  - Katz Back-Off
  - General linear interpolation

Simple linear interpolation

- Also called mixture model

$$P_{li}\left(w_n \middle| w_{n-2}, w_{n-1}\right) = \lambda_1 P_1\left(w_n\right) + \lambda_2 P_2\left(w_n \middle| w_{n-1}\right) + \lambda_3 P_3\left(w_n \middle| w_{n-1}, w_{n-2}\right)$$

where : $0 \le \lambda_i \le 1, \ \sum_i \lambda_i = 1$

- How to get the weights :
  - Expectation Maximization (EM) algorithm
  - Powell's algorithm

- The method works quite well. Chen and Goodman use it as baseline model.

General linear interpolation

- The difference between GLI and SLI is that the weights (λ_i) of the GLI is a function of the history.

$$P_{il}\left(w \middle| h\right) = \sum_{i=1}^{k} \lambda_i\left(h\right) P_i\left(w \middle| h\right)$$

where : $\forall h,\ \ 0 \le \lambda_i\left(h\right) \le 1,\ \ and\ \ \sum_i \lambda_i\left(h\right) = 1$

- Can make bad use of component models
  - Ex: unigram estimate is always combined in with the same weight regardless of whether the trigram is good or bad.

Katz Back-off

- Extend the intuition of the GT estimate by adding the combination of higher-order language models with lower-order ones

- Larger counts are not discounted because they are taken to be reliable. ($r>k$)

- Lower counts are total discounted. ($r<=k$)

Katz Back-off

- We take the bigram (n-gram, n=2) counts for example :

$$C^*[w_{i-1}w_i] = \begin{cases} r & if \quad r > k \\ d_r r & if \quad k \geq r > 0 \\ \beta(w_{i-1}) P_{Katz}(w_i) & if \quad r = 0 \end{cases}$$

1. $r = C[w_{i-1}w_i]$

2. $d_r = \dfrac{\dfrac{r^*}{r} - \dfrac{(k+1)n_{k+1}}{n_1}}{1 - \dfrac{(k+1)n_{k+1}}{n_1}}$

3. $\beta(w_{i-1}) = \dfrac{\sum_{w_i} C[w_{i-1}, w_i] - \sum_{w_i : C[w_{i-1}, w_i] > 0} C^*[w_{i-1}, w_i]}{\sum_{w_i : C[w_{i-1}, w_i] = 0} P_{katz}(w_i)}$

Katz Back-off

count before discount

count after discount

$$\beta(w_{i-1}) = \frac{\sum_{w_i} C[w_{i-1}, w_i] - \sum_{w_i : C[w_{i-1}, w_i] > 0} C^*[w_{i-1}, w_i]}{\sum_{w_i : C[w_{i-1}, w_i] = 0} P_{katz}(w_i)}$$

Unigram weight

Katz Back-off

- Derivation of d_r:
  - Before of the derivation, the d_r have to satisfy two equation:

$$\sum_{r=1}^{k} n_r (1-r)r = n_1 \quad \text{and} \quad d_r = \mu \frac{r^*}{r}$$

1.

$$\sum_{r=1}^{k} r n_r - \sum_{r=1}^{k} r^* n_r = n_1 - (k+1) n_{k+1}$$

$$\Rightarrow \sum_{r=1}^{k} \left(r n_r - r^* n_r \right) = n_1 - (k+1) n_{r+1}$$

$$\Rightarrow \frac{\sum_{r=1}^{k} \left(r n_r - r^* n_r \right)}{n_1 - (k+1) n_{r+1}} = 1$$

$$\Rightarrow \sum_{r=1}^{k} \frac{n_r \left(r - r^* \right) n_1}{n_1 - (k+1) n_{r+1}} = n_1 \qquad (1)$$

2.

$$\sum_{r=1}^{k} n_r (1 - d_r) r = n_1$$

$$\Rightarrow \sum_{r=1}^{k} n_r \left(1 - \mu \frac{r^*}{r} \right) r = n_1$$

$$\Rightarrow \sum_{r=1}^{k} n_r \left(r - \mu r^* \right) = n_1 \qquad (2)$$

Katz Back-off

$\because$ (1)=(2);

$$\therefore \quad \frac{n_r\left(r-r^*\right)n_1}{n_1-(k+1)n_{r+1}} = n_r\left(r-\mu r^*\right)$$

$$\Rightarrow \frac{\left(r-r^*\right)n_1}{r\left[n_1-(k+1)n_{k+1}\right]} = 1-\mu\frac{r^*}{r} = 1-d_r$$

$$\Rightarrow d_r = 1-\frac{\left(r-r^*\right)n_1}{r\left[n_1-(k+1)n_{k+1}\right]}$$

$$= \frac{r\left[n_1-(k+1)n_{k+1}\right]-\left(r-r^*\right)n_1}{r\left[n_1-(k+1)n_{k+1}\right]}$$

$$= \frac{r^*n_1-r(k+1)n_{k+1}}{r\left[n_1-(k+1)n_{k+1}\right]}$$

divide r*n₁

$$\therefore d_r = \frac{\frac{r^*}{r}-\frac{(k+1)n_{k+1}}{n_1}}{1-\frac{(k+1)n_{k+1}}{n_1}}$$

Katz Back-off

- Take the conditional probabilities of bigrams (n-gram, n=2)

  For example :

$$P_{Katz}(w_i|w_{i-1}) = \begin{cases} \dfrac{C[w_{i-1}, w_i]}{C[w_{i-1}]} & if \quad r > k \\ d_r \dfrac{C[w_{i-1}, w_i]}{C[w_{i-1}]} & if \quad k \ge r > 0 \\ \alpha(w_{i-1}) P_{Katz}(w_i) & if \quad r = 0 \end{cases}$$

1. $$d_r = \frac{\dfrac{r^*}{r} - \dfrac{(k+1)n_{k+1}}{n_1}}{1 - \dfrac{(k+1)n_{k+1}}{n_1}}$$

2. $$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i : C[w_{i-1} w_i] > 0} P_{Katz}(w_i|w_{i-1})}{\sum_{w_i : C[w_{i-1} w_i] = 0} P_{Katz}(w_i)}$$

Katz Back-off : Example

- A small vocabulary consists of only five words, i.e., $V = \{w_1, w_2, \cdots, w_5\}$.

  The frequency counts for word pairs started with w_1 are:

  $$C[w_1, w_2] = 3, C[w_1, w_3] = 2, C[w_1, w_4] = 1, C[w_1, w_1] = C[w_1, w_5] = 0$$

  , and the word frequency counts are :

  $$C[w_1] = 6, C[w_2] = 8, C[w_3] = 10, C[w_4] = 6, C[w_5] = 4$$

  Katz back-off smoothing with Good-Turing estimate is used here for word pairs with frequency counts equal to or less than two. Show the conditional probabilities of word bigrams started with w_1 ,i.e.,

  $$P_{Katz}(w_1|w_1), P_{Katz}(w_2|w_1), \cdots, P_{Katz}(w_5|w_1)?$$

Katz Back-off : Example

$$r^* = (r+1)\frac{n_{r+1}}{n_r}, where \quad n_r \quad is \quad the \quad number \quad of \quad n-grams \quad that \quad occurs \quad exactly \quad r \quad times \quad in \quad the \quad training \quad data$$

$$\therefore P_{katz}(w_2|w_1) = P_{ML}(w_2|w_1) = \frac{3}{6} = \frac{1}{2}$$

$$1^* = (1+1)\cdot\frac{1}{1} = 2 \qquad 2^* = (2+1)\cdot\frac{1}{1} = 3$$

$$d_2 = \frac{\frac{3}{2} - \frac{(2+1)\cdot 1}{1}}{1 - \frac{(2+1)\cdot 1}{1}} = \frac{\frac{3}{2} - 3}{-2} = \frac{3}{4} \qquad d_1 = \frac{\frac{2}{1} - \frac{(2+1)\cdot 1}{1}}{1 - \frac{(2+1)\cdot 1}{1}} = \frac{2-3}{1-3} = \frac{1}{2}$$

$$For \quad r=2 \quad \Rightarrow \quad P_{Katz}(w_3|w_1) = d_2 * P_{ML}(w_3|w_1) = \frac{3}{4}\cdot\frac{2}{6} = \frac{1}{4}$$

$$For \quad r=1 \quad \Rightarrow \quad P_{Katz}(w_4|w_1) = d_1 * P_{ML}(w_4|w_1) = \frac{1}{2}\cdot\frac{1}{6} = \frac{1}{12}$$

$$\alpha(w_1) = \frac{1 - \frac{1}{2} - \frac{1}{4} - \frac{1}{12}}{\frac{6}{34} + \frac{4}{34}} = \frac{34}{10}\cdot\frac{2}{12}$$

$$For \quad r=0 \quad \Rightarrow \quad P_{Katz}(w_1|w_1) = \alpha(w_1) * P_{ML}(w_1) = \frac{34}{10}\cdot\frac{2}{12}\cdot\frac{6}{34} = \frac{1}{10}$$

$$P_{Katz}(w_5|w_1) = \alpha(w_1) * P_{ML}(w_5) = \frac{34}{10}\cdot\frac{2}{12}\cdot\frac{4}{34} = \frac{1}{15}$$

$$And \quad P_{Katz}(w_1|w_1) + P_{Katz}(w_2|w_1) + \cdots + P_{Katz}(w_5|w_1) = 1$$

Kneser-Ney Back-off smoothing

- Absolute discounting

- The lower $n\text{-}gram$ (back-off n-gram) is not proportional to the number of occurrences of a word but instead of the number of different words that it follows.

Kneser-Ney Back-off smoothing

- Take the conditional probabilities of bigrams for example :

$$P_{KN}\left(w_i \middle| w_{i-1}\right)=\begin{cases}\dfrac{\max\left\{C\left[w_{i-1},w_i\right]-D\ \ ,\ \ 0\right\}}{C\left[w_{i-1}\right]} & if\quad C\left[w_{i-1},w_i\right]>0\\ \alpha\left(w_{i-1}\right)P_{KN}\left(w_i\right) & otherwise\end{cases}$$

1. $P_{KN}\left(w_i\right)=\dfrac{C\left[\bullet\, w_i\right]}{\sum_{w_j} C\left[\bullet\, w_j\right]}$

2. $\alpha\left(w_{i-1}\right)=\dfrac{1-\sum_{w_i:C\left[w_{i-1}w_i\right]>0}\dfrac{\max\left\{C\left[w_{i-1}w_i\right]-D\ \ ,\ \ 0\right\}}{C\left[w_{i-1}\right]}}{\sum_{w_i:C\left[w_{i-1}w_i\right]=0} P_{KN}\left(w_i\right)}$

Kneser-Ney Back-off smoothing : Example

- Given a text sequence as the following :

  SABCAABBCS (*S* is the sequence's start/end marks)

  Show the corresponding unigram conditional probabilities:

$$C[\bullet A] = 3 \qquad C[\bullet B] = 2$$

$$C[\bullet C] = 1 \qquad C[\bullet S] = 1$$

$$\Rightarrow P_{KN}(A) = \frac{3}{7} \qquad P_{KN}(B) = \frac{2}{7}$$

$$P_{KN}(C) = \frac{1}{7} \qquad P_{KN}(S) = \frac{1}{7}$$

Evaluation

- Cross entropy :

$$\mathrm{H(X,q) = H(X) + D(p \,||\, q) = -\sum_{x} p(x) \log q(x)}$$

- Perplexity = $2^{Entropy}$

- A LM that assigned probability to 100 words would have perplexity 100

$$Entropy = -\sum_{i=1}^{100} \frac{1}{100} \log_2 \frac{1}{100} = \sum_{i=1}^{100} \frac{1}{100} \log_2 100 = \log_2 100$$

$$perplexity = 2^{\log_2 100} = 100$$

Evaluation

- In general, the perplexity of a LM is equal to the geometric average of the inverse probability of the words measured on test data:

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

Evaluation

$$perplexity = 2^{Entropy} = 2^{-\sum_{i=1}^{N} \frac{1}{N} \cdot \log_2 P(w_i | w_1 \ldots w_{i-1})}$$

$$= \frac{1}{\prod_{i=1}^{N} 2^{\frac{1}{N} \cdot \log_2 P(w_i | w_1 \ldots w_{i-1})}}$$

$$= \frac{1}{\prod_{i=1}^{N} P(w_i \mid w_1 \ldots w_{i-1})^{\frac{1}{N}}}$$

$$= \prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \ldots w_{i-1})^{\frac{1}{N}}}$$

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

$P(w_{1n})=p(w_1)p(w_2)\ldots p(w_n)$

$\log P(w_{1n})=\sum p(w_i)$

Evaluation

- "true" model for any data source will have the lowest possible perplexity
- The lower the perplexity of our model, the closer it is, in some sense, to the true model
- Entropy, which is simply $\log_2$ of perplexity
- Entropy is the average number of bits per word that would be necessary to encode the test data using an optimal coder

Evaluation

entropy	.01	.1	.16	.2	.3	.4	.5	.75	1
perplexity	0.69%	6.7%	10%	13%	19%	24%	29%	41%	50%

- entropy : 5→4
  perplexity : 32→16 50%
- entropy : 5→4.5
  perplexity : 32→$16\sqrt{2}$ 29.3%

Conclusions

- A number of smoothing method are available which often offer similar and good performance.

- More powerful combining methods ?