Statistical inference: n-gram model over sparse data

References:

1. Foundations of Statistical Natural Language Processing, chapter 6
2. Speech and Language Processing, chapter 6
3. A Bit of Progress in Language Modeling Extended Version, Joshua T. Goodman, 2001

Outline

- N-gram
- MLE
- Smoothing
- Evaluation

Introduction

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

- Statistical inference in general consists of taking some data and then making some inferences about the distribution.
  - Predict prepositional phrase attachment

- A running example of statistical estimation: language modeling.
  - Shannon game http://math.ucsd.edu/~crypto/java/ENTROPY/

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)
- Try to predict the target feature on the basis of various classificatory features.
- Using equivalence class: independence assumptions
  - Features are independent

Reliability vs. discrimination

- Dividing the data into many bins gives us greater discrimination.
- With small number of training instances, we can not do statistically reliable estimation.
- good compromise between two criteria?

Corpora

- Singular corpus
- Statistical processing of natural language is based on corpora, on-line collections of text and speech.
- We compute word probability by counting words.
- Text (sentence): punctuation-marks, ‘.’, ‘?’
- Speech (utterance): filled pauses, uh, um
  - See ‘Uh’ as words

n-gram models

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

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

- History: classification of the previous words

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

n-gram models

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

- Naming:
  - Gram is Greek root and so should be put together with prefixes
  - digram, tetragram

n-gram models

- Sue swallowed the large green ___.
  - green /frog/
  - large green /tree/car/mountain/
  - swallowed … /pill/

- 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.

- The last target value is automatically given by stochastic constraint that probability should sum to one.

n-gram models

- Five-gram model that we thought would be useful, may well not be practical, even if we a very large corpus.
- One way of reducing the number of parameters is to reduce the value of n.
- Or removing the inflectional ending from words
  - Stemming
- Or grouping words into semantic classes
- Or …(ref. ch12, ch14)

n-gram models

- Predicts the next word only simply by examining the previous two words seems almost preposterous?

- Indeed, it is difficult to beat a trigram model on the pure linear task of predicting the next word.

Building n-gram models

- Corpus: Jane Austen's novel
  - Free available and not too large

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

Building n-gram models

- Preprocessing
  - Remove punctuation leaving white-space.
  - Add SGML tag <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.

$$P(w_n \mid w_1 \ldots w_{n-1}) = \frac{P(w_1 \ldots w_n)}{P(w_1 \ldots w_{n-1})}$$

- 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: h_1a, h_2a, h_3a, h_4b,h_5b…reduce to a,b

Statistical Estimators

- Assume that the training text consists of N words. We append n-1 dummy start symbols to the beginning of the text.
  - N n-grams 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:

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

$$P_{\text{MLE}}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)}{N}$$

$$P_{\text{MLE}}(w_n \mid w_1 \ldots w_{n-1}) = \frac{C(w_1 \ldots w_n)}{C(w_1 \ldots w_{n-1})}$$

Maximum Likelihood Estimation

- Derivation of MLE:

$$P(w_1 \ldots w_n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \ldots P(w_k \mid w_1 \ldots w_{k-2}, w_{k-1})$$

$$\cong P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \ldots P(w_k \mid w_{k-n+1} \ldots w_{k-2}, w_{k-1})$$

$$= \prod_{w_i} P(w_i \mid h_i) = \prod_h \prod_{w_i} P(w_i \mid h)^{N_{hw_i}} = \prod_h \prod_{w_i} \lambda_{hw_i}^{N_{hw_i}} = \Phi$$

$$\overline{\Phi} = \Phi + \sum_h l_h (\sum_{w_j} \lambda_{hw_j} - 1)$$

$$\frac{\partial \log \overline{\Phi}}{\partial \lambda_{hw_i}} = \frac{\sum_h \sum_{wi} N_{hw_i} \log \lambda_{hw_i} + \sum_h l_h (\sum_{w_j} \lambda_{hw_j} - 1)}{\partial \lambda_{hw_i}}$$

$$\Rightarrow \frac{N_{hw_i}}{\lambda_{hw_i}} + l_h = 0 \Rightarrow \frac{N_{hw_i}}{\lambda_{hw_i}} = -l_h$$

$$\Rightarrow \frac{N_{hw_1}}{\lambda_{hw_1}} = \frac{N_{hw_2}}{\lambda_{hw_2}} = \frac{N_{hw_3}}{\lambda_{hw_3}} = \ldots = -l_h$$

$$\frac{\sum_{w_s} N_{hw_s}}{\sum_{w_j} \lambda_{hw_j}} = -l_h \Rightarrow l_h = -\sum_{w_s} N_{hw_s} = -N_h$$

$$\lambda_{hw_i} = \frac{N_{hw_i}}{N_h}$$

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-gram.

Cross-validation

- Dividing training data into two parts.
- Two-way cross-validation
  - Delete estimation

$$P_{\mathrm{del}}(w_1 \ldots w_n) = \frac{T_r^{01} + T_r^{10}}{T(N_r^0 + N_r^1)}$$

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

Add-One Smoothing

- Take counts before normalize
- Unigram MLE:

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

- Add one to count and multiply by a normalization factor

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

Add-One Smoothing

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

$$d_c = \frac{c^*}{c}$$

- Smoothed probability:

$$p_i^* = \frac{c_i + 1}{N + V}$$

- Also called Laplace's law

Add-One Smoothing

- Unigram Example:
- V={A,B,C,D,E}, |V|=5
- S={A,A,A,A,A,B,B,B,C,C,}, N=|S|=10
- 5 for ‘A’, 3 for ‘B’, 2 for ‘C’, 0 for ‘D’,’E’
- P(A)=(5+1)/(10+5)=0.4
- P(B)=(3+1)/(10+5)=0.27
- P(C)=(2+1)/(10+5)=0.2
- P(D)=P(E)=(0+1)/(10+5)=0.067

Add-One Smoothing

Bigram

- MLE:

$$P(w_n \mid w_{n-1}) = \frac{c(w_{n-1}w_n)}{c(w_{n-1})}$$

- Smoothed:

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

Unigram/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

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.

- N(want)=1215
- N(want want)=0
- N(want to)=786

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

Bigram probabilities

	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: bigram

	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() changes from 0 to 0.0035 ok
- P() changes from 0.65 to 0.28 bad

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

- Gale and Church (1994) 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

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

- Jeffreys-Perks law
  - Viewed as linear interpolation between MLE and a uniform prior
  - Also called Expected Likelihood Estimation

$$P_{\text{Lid}}(w_1 \ldots w_n) = \mu \frac{C(w_1 \ldots w_n)}{N} + (1 - \mu)\frac{1}{B}$$

$$\mu = N/(N + B\lambda)$$

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-gram for the first time.

- Key: **things seen once !**

Witten-Bell Discounting

- The count of ‘first time’ n-grams is just for the number of n-gram **types** we saw in data.
- 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)

Witten-Bell Discounting

- Divide up to among all the zero N-grams
  - Divided Equally:
  - Z: total number of zero count n-grams

$$Z = \sum_{i:Ci=0} 1 \qquad p_i^* = \frac{T}{Z(N+T)}$$

- Probability of all the seen n-grams:

$$p_i^* = \frac{c_i}{N+T} \text{if}(ci > 0)$$

Witten-Bell Discounting

- Represent as:

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

Witten-Bell Discounting

- Unigram Example:
- V={A,B,C,D,E}, |V|=5
- S={A,A,A,A,A,B,B,B,C,C,}, N=|S|=10
- 5 for ‘A’, 3 for ‘B’, 2 for ‘C’, 0 for ‘D’,’E’, T=|{A,B,C}|=3, Z=2
- P(A)=5/(10+3)=0.385
- P(B)=3/(10+3)=0.23
- P(C)=2/(10+3)=0.154
- P(D)=P(E)=3/(10+3)*(1/2)=0.116

Witten-Bell Discounting

- Bigram
  - Using the probability of seeing a bigram $w_{n-1}w_{n-2}$ starting with w_{n-1}

$$\sum_{i:c(w_x w_i)=0} p^*(w_i \mid w_x) = \frac{T(w_x)}{N(w_x)+T(w_x)}$$

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

$$p^*(w_i \mid w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N(w_{i-1})+T(w_{i-1}))} \quad \text{if}(c_{w_{i-1}w_i} = 0)$$

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

Witten-Bell Discounting

- T(w), V=1616

I	95
want	76
to	130
eat	124
Chinese	20
food	82
lunch	45

$$Z(w) = V - T(w)$$

Here are those Z values:

I	1,521
want	1,540
to	1,486
eat	1,492
Chinese	1,596
food	1,534
lunch	1,571

Witten-Bell Discounting

	I	want	to	eat	Chinese	food	lunch
I	8	1060	.062	13	.062	.062	.062
want	3	.046	740	.046	6	8	6
to	3	.085	10	827	3	.085	12
eat	.075	.075	2	.075	17	2	46
Chinese	2	.012	.012	.012	.012	109	1
food	18	.059	16	.059	.059	.059	.059
lunch	4	.026	.026	.026	.026	1	.026

Figure 6.9 Witten-Bell smoothed bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ~10,000 sentences.

- C(want want)=(76/1540)*(1215/(1215+76)=0.046
- C(want to)=786*1215/(1215+76)=740

Good-Turing Discounting

- A slightly more complex form than Witten-Bell.
- To re-estimate zero or low counts by higher counts.
- Nc: the number of N-grams that occurs c times.
  - Frequency of frequency

$$N_c = \sum_{n:c(n)=c} 1$$

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

Good-Turing Discounting

- $0^* = N_1/N_0$ (N_1:singleton or hapax legomenon)

c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270
1	2,018,046	0.446
2	449,721	1.26
3	188,933	2.24
4	105,668	3.24
5	68,379	4.22
6	48,190	5.19
7	35,709	6.21
8	27,710	7.24
9	22,280	8.25

Figure 6.10 Bigram 'frequencies of frequencies' from 22 million AP bigrams, and Good-Turing re-estimations after Church and Gale (1991)

Assume $N_0 = V^2$

Good-Turing Discounting

- Probability estimate:
  - Unseen: n_1/N, why?

- Unigram Example:
  - A:10, B:3, C:2, D:1, E:1, F:1,N_1:3,N_2:1,N_3:1,N_{10}:1, N=18
  - 1*=(1+1)*(1/3)=2/3, 2*=(2+1)*(1/1)=3(?), 3*= ?
  - C*(D)=C*(E)=C*(F)=2/3
  - C*(C)=2
  - C*(B)=3
  - C*(A)=10
  - C*(X)=1

Good-Turing Discounting

- Let *N* represent the total size of the training set, this left-over probability will be equal to N_1/N

$(r+1)N_{r+1}=0$

$$r^* = (r+1)\frac{N_{r+1}}{N_r} \qquad (r-(r+1)\frac{N_{r+1}}{N_r})\cdot N_r \qquad r\cdot N_r - (r+1)N_{r+1}$$

$$(r-1)^* = r\frac{N_r}{N_{r-1}} \qquad ((r-1)-r\frac{N_r}{N_{r-1}})\cdot N_{r-1} \qquad (r-1)\cdot r_{n-1} - r\cdot N_r$$

$$\ldots \qquad \ldots \qquad \ldots$$

$$2^* = 3\cdot\frac{N_3}{N_2} \qquad (2-3\cdot\frac{N_3}{N_2})\cdot N_2 \qquad 2\cdot N_2 - 3\cdot N_3$$

$$1^* = 2\cdot\frac{N_2}{N_1} \qquad (1-2\cdot\frac{N_2}{N_1})\cdot N_1 \qquad 1\cdot N_1 - 2\cdot N_2$$

Sum=n_1

- Some problems:
  - $N_{c+1}=0$
  - $P(c^*)>P((c+1)^*)$

- Solution: parameter k, $N_{k+1} \neq 0$, experimentally $4 \leqq k \leqq 8$

Combining Estimators

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

- Combine multiple probability estimates from various different models.
  - Simple linear interpolation
  - Katz Back-off
  - General linear interpolation
  - Maximum Entropy. ref. ch.16

Simple linear interpolation

- Also called mixture model

$$P_{li}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_1 P_1(w_n) + \lambda_3 P_2(w_n \mid w_{n-1}) + \lambda_3 P_3(w_n \mid w_{n-1}, w_{n-2})$$
$$\sum_i \lambda_i = 1,\ 0 \le \lambda_i \le 1$$

- How to find weight:
  - Expectation Maximization algorithm (ref.9.2.1)
  - Powell's algorithm.

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

General linear interpolation

- Also called deleted interpolation
  - Weight are a function of the history

$$P_{li}(w \mid h) = \sum_{i=1}^{k} \lambda_i(h) P_i(w \mid h)$$

- 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 Smoothing

- A special case of general linear interpolation model.
  - Extend the intuition of the GT estimate.
  - Large counts are not discounted. (c>k)
  - Lower counts are total discounted. (c<=k)

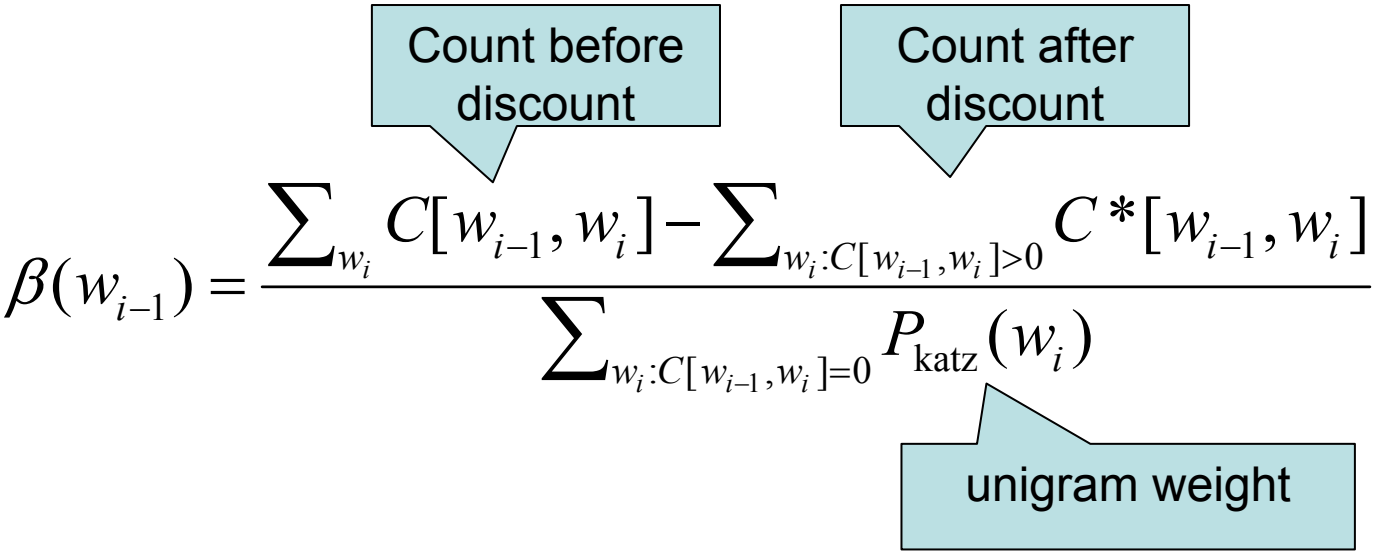
Katz Back-off Smoothing

$$c^*[w_{i-1}, w_i] = \begin{cases} r & \text{if r} > \text{k} \\ d_r r & \text{if k} \geq \text{r} > 0 \\ \beta(w_{i-1}) P_{katz}(wi) & \text{if r} = 0 \end{cases}$$

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

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

Katz Back-off Smoothing



Katz Back-off Smoothing

- Derivation of d_r:
- Known constraint:

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

$$d_r = \mu \frac{r^*}{r}$$

1.

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

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

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

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

Katz Back-off Smoothing

2.

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

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

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

$1 = 2$

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

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

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

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

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

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

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

Katz Back-off Smoothing

- Katz smoothing is based on the Good-Turing formula
- Let n_r represent the number of n-grams that occur r times
- *discount* :

$$disc(r) = (r+1)\frac{n_{r+1}}{n_r} = r^*$$

$$P_{\text{Katz}}(w_i \mid w_{i-n+1}...w_{i-1})$$

$$= \begin{cases} \dfrac{disc(C(w_{i-n+1}...w_i))}{C(w_{i-n+1}...w_{i-1})} & \text{if } C(w_{i-n+1}...w_i) > 0 \\ \alpha(w_{i-n+1}...w_{i-1}) \times P_{\text{Katz}}(w_i \mid w_{i-n+2}...w_{i-1}) & \text{otherwise} \end{cases}$$

$$\alpha(w_{i-n+1}...w_{i-1}) = \frac{1 - \sum_{w_i : c(w_{i-n+1}^i) > 0} P_{\text{Katz}}(w_i \mid w_{i-n+1}...w_{i-1})}{1 - \sum_{w_i : c(w_{i-n+1}^i) > 0} P_{\text{Katz}}(w_i \mid w_{i-n+2}...w_{i-1})}$$

Kneser-Ney smoothing

- Absolute discounting
- Weight Zero probability by Branch

- Absolute discounting

$$P_{\text{abs}}(w_1 \ldots w_n) = \frac{\max\{C[w_{i-1}, w_i] - D, 0\}}{C[w_{i-1}]}$$

- Branch:
  - SABCAABBCS
  - C[• A]=3, C[• B]=2, C[• C]=1, C[• S]=1

Kneser-Ney smoothing

- Bigram

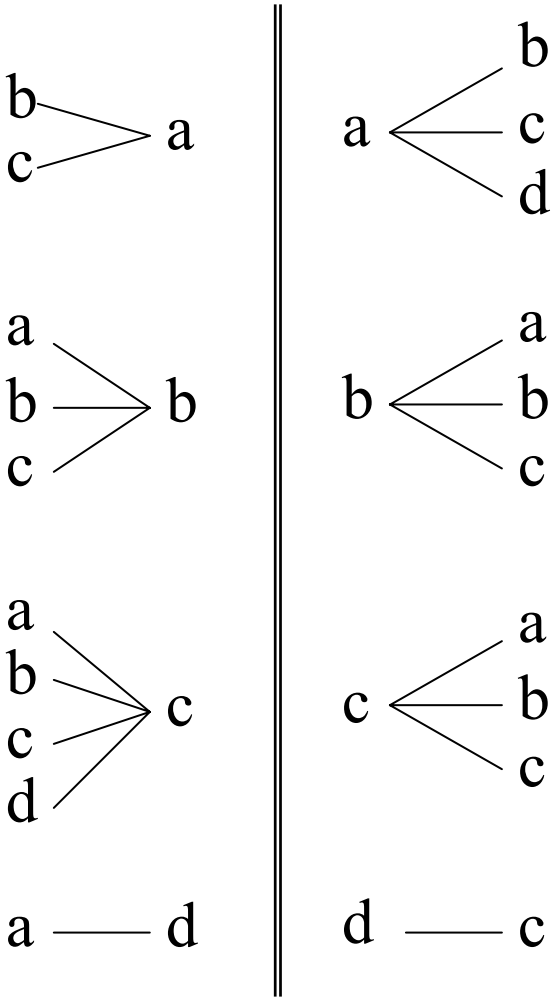
$$P_{\mathrm{KN}}(w_i \mid w_{i-1}) = \begin{cases} \dfrac{\max\{C[w_{i-1}, w_i] - D, 0\}}{C[w_{i-1}]} & \text{if } \mathrm{c}[\mathrm{w_{i-1}}, \mathrm{w_i}] > 0 \\ \\ \alpha(w_{i-1}) P_{KN}(w_i) & \text{otherwise} \end{cases}$$

$$P_{KN}(w_i) = C[\bullet w_i] / \sum_{w_j} C[\bullet w_j]$$

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i : C(w_{i-1} w_i) > 0} \dfrac{\max\{C(w_{i-1} w_i) - D, 0\}}{C(w_{i-1})}}{\sum_{w_i : C(w_{i-1} w_i) = 0} P_{KN}(w_i)}$$

Kneser-Ney smoothing

V={a,b,c,d}



$$P_{KN}(\mathrm{a}) = \frac{2}{10}, P_{KN}(\mathrm{b}) = \frac{3}{10},$$

$$P_{KN}(\mathrm{c}) = \frac{4}{10}, P_{KN}(\mathrm{d}) = \frac{1}{10}$$

$$P_{BKN}(\mathrm{a} \mid \mathrm{d}) = \alpha(\mathrm{d}) \cdot P_{KN}(\mathrm{a})$$

$$= \frac{1 - \frac{\max\{C(\mathrm{dc}) - D, 0\}}{C(\mathrm{d})}}{1 - \frac{4}{10}} \cdot \frac{2}{10}$$

Kneser-Ney smoothing

- How to set discount D: empirically
- Interpolated Kneser-Ney smoothing

$$P_{\mathrm{K}N}(w_i \mid w_{i-1}) = \frac{\max\{C[w_{i-1}, w_i] - D, 0\}}{C[w_{i-1}]} + \lambda(w_{i-1}) P_{KN}(w_i)$$

- Modified Kneser-Ney smoothing
  - Different discount with different n-gram

$$P_{\mathrm{IKN}}(w_i \mid w_{i-2} w_{i-1}) = \frac{C(w_{i-2} w_{i-1} w_i) - D_3}{C(w_{i-2} w_{i-1})} + \lambda(w_{i-2} w_{i-1}) P_{\text{ikn-mod-bigram}}(w_i \mid w_{i-1})$$

$$P_{\text{ikn-mod-bigram}}(w_i \mid w_{i-1}) = \frac{|\{v \mid C(v w_{i-1} w_i) > 0\}| - D_2}{\sum_w |\{v \mid C(v w_{i-1} w) > 0\}|} + \lambda(w_{i-1}) P_{\text{ikn-mod-unigram}}(w_i)$$

$$P_{\text{ikn-mod-unigram}}(w_i) = \frac{|\{v \mid C(v w_i) > 0\}| - D_1}{\sum_w |\{v \mid C(v w) > 0\}|} + \lambda \frac{1}{|V|}$$

Held out estimation

- How do we know that giving of the probability to unseen is too much? Test empirically!

- The held out estimator:

$$C_1(w_1 \ldots w_n) = \text{frequency of } w_1..w_n \text{ in training data}$$

$$C_2(w_1 \ldots w_n) = \text{frequency of } w_1..w_n \text{ in held out data}$$

$$T_r = \sum_{\{w_1 \ldots w_n : C1(w_1 \ldots w_n) = r\}} C_2(w_1 \ldots w_n)$$

- T_r: the total number of times that all n-grams that appeared in the training data appeared in held out data.

Held out estimation

- The probability of one of these n-grams

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

- A cardinal sin in Statistical NLP is to test on training data. Why?
  - Overtraining
  - Models memorize the training text (MLE may be good enough)
  - Cheating!
- Test data is independent of the training data.

- Separate data immediately into training and test data(5~10%,reliable).

Held out estimation

- Held out (validation) data(10%)
  - Independent of primary training and test data
  - Involve many fewer parameters
- Training
  - Write an algorithm, train it and test it (X)
  - Separate to Development test set, final test set (O)
- Testing
  - How to select test/held out data? Randomly or aside large contiguous chunk
  - Comparing average scores is not enough
  - Divide the test data into several parts
  - T-test

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
$\text{mean } \bar{x}_i$	61.2	53.9
$s_i^2 = \sum (x_{ij} - \bar{x}_i)^2$	1081.6	1186.9
df	10	10

$$\text{Pooled } s^2 = \frac{1081.6 + 1186.9}{10 + 10} \approx 113.4 \quad t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{2s^2}{n}}} = \frac{61.2 - 53.9}{\sqrt{\frac{2*113.4}{11}}} \approx 1.60$$

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

Evaluation

cross entropy :

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

$$\text{perplexity} = 2^{\text{Entropy}}$$

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

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

$$\textit{So, perplexity} = 2^{\log_2 100} = 100$$

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:

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

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

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

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

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

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

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

Introduction
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 ?