Probability Estimation

D. De Cao   R. Basili

Corso di *Web Mining* e *Retrieval*
a.a. 2008-9

May 19, 2009

Outline

- Laplace Estimator
- Good-Turing
- Backoff

The Sparse Data Problem

There is a major problem with the maximum likelihood estimation (MLE) process for training the parameters of an N-gram model.
But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.

The Sparse Data Problem

There is a major problem with the maximum likelihood estimation (MLE) process for training the parameters of an N-gram model.
But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.

Under Markov assumption

$$P(W) = P(w_1) \cdot P(w_2, w_1) \cdot \ldots \cdot P(w_{i+1}, w_i)$$

But what if we have never before seen $w_i w_{i+1}$ in string W?
The MLE estimate $P(w_{i+1}|w_i)$ is:

$$\frac{C(w_i, w_{i+1})}{C(w_i)} = \frac{0}{C(w_i)} = 0 \quad \text{So } P(W) = 0$$

The Sparse Data Problem

There is a major problem with the maximum likelihood estimation (MLE) process for training the parameters of an N-gram model.
But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.

Under Markov assumption

$$P(W) = P(w_1) \cdot P(w_2, w_1) \cdot \ldots \cdot P(w_{i+1}, w_i)$$

But what if we have never before seen $w_i w_{i+1}$ in string W?
The MLE estimate $P(w_{i+1}|w_i)$ is:

$$\frac{C(w_i, w_{i+1})}{C(w_i)} = \frac{0}{C(w_i)} = 0 \quad \text{So } P(W) = 0$$

Solution

Develop a model which decreases probability of seen events and allows the occurrence of previously unseen n-grams (a.k.a. *Discounting methods*)

Add-One Smooting (Laplace Estimator)

Estimate probabilities P assuming that each unseen word type actually occurred once.

Add-One Smooting (Laplace Estimator)

Estimate probabilities P assuming that each unseen word type actually occurred once. Then if we have N events and V possible words instead of

$$P(w) = \frac{occ(w)}{N}$$

Add-One Smooting (Laplace Estimator)

Estimate probabilities P assuming that each unseen word type actually occurred once. Then if we have N events and V possible words instead of

$$P(w) = \frac{occ(w)}{N}$$

we estimate:

$$P_{addone}(w) = \frac{occ(w)+1}{N+V}$$

Add-One Smooting (Laplace Estimator)

What about bigram?

Add-One Smooting (Laplace Estimator)

What about bigram?

MLE:

$$P(w_{i+1}|w_i) = \frac{C(w_i, w_{i+1})}{C(w_i)}$$

Add-One Smooting (Laplace Estimator)

What about bigram?

MLE:

$$P(w_{i+1}|w_i) = \frac{C(w_i, w_{i+1})}{C(w_i)}$$

Laplace Smooting:

$$P^*(w_{i+1}|w_i) = \frac{C(w_i, w_{i+1}) + 1}{C(w_i) + V}$$

Example of bigram count

	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

Example of bigram count

	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

Total word occurrence:

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

Example of bigram probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	.002	.33	0	.0036	0	0	0	.00079
want	.0022	0	.66	.0011	.0065	.0065	.0054	.0011
to	.00083	0	.0017	.28	.00083	0	.0025	.087
eat	0	0	.0027	0	.021	.0027	.056	0
chinese	.0063	0	0	0	0	.52	.0063	0
food	.014	0	.014	0	.00092	.0037	0	0
lunch	.0059	0	0	0	0	.0029	0	0
spend	.0036	0	.0036	0	0	0	0	0

Example of bigram count - Laplace smooting

	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

Example of bigram probabilities - Laplace smooting

	i	want	to	eat	chinese	food	lunch	spend
i	.0015	.21	.00025	.0025	.00025	.00025	.00025	.00075
want	.0013	.00042	.26	.00084	.0029	.0029	.0025	.00084
to	.00078	.00026	.0013	.18	.00078	.00026	.0018	.055
eat	.00046	.00046	.0014	.00046	.0078	.0014	.02	.00046
chinese	.0012	.00062	.00062	.00062	.00062	.052	.0012	.00062
food	.0063	.00039	.0063	.00039	.00079	.002	.00039	.00039
lunch	.0017	.00056	.00056	.00056	.00056	.0011	.00056	.00056
spend	.0012	.00058	.0012	.00058	.00058	.00058	.00058	.00058

Consideration

- Pro:
    - Very simple technique

Consideration

- Pro:
  - Very simple technique
- Cons:
  - Probability of frequent n-grams is underestimated

Consideration

- Pro:
  - Very simple technique
- Cons:
  - Probability of frequent n-grams is underestimated
  - Probability of rare (or unseen) n-grams is overestimated

Consideration

- Pro:
  - Very simple technique
- Cons:
  - Probability of frequent n-grams is underestimated
  - Probability of rare (or unseen) n-grams is overestimated
  - Therefore, too much probability mass is shifted towards unseen n-grams

Consideration

- Pro:
  - Very simple technique
- Cons:
  - Probability of frequent n-grams is underestimated
  - Probability of rare (or unseen) n-grams is overestimated
  - Therefore, too much probability mass is shifted towards unseen n-grams
  - All unseen n-grams are smoothed in the same way

Consideration

- Pro:
  - Very simple technique
- Cons:
  - Probability of frequent n-grams is underestimated
  - Probability of rare (or unseen) n-grams is overestimated
  - Therefore, too much probability mass is shifted towards unseen n-grams
  - All unseen n-grams are smoothed in the same way
- Using a smaller added-count improves things but only some

Good-Turing smoothing

The Good-Turing formula provides another way to smooth probabilities.

Good-Turing smoothing

The Good-Turing formula provides another way to smooth probabilities.

Basic idea:

use the count of things you've seen *once* to help estimate the count of things you've *never seen*.

Good-Turing smoothing

The Good-Turing formula provides another way to smooth probabilities.

Basic idea:

use the count of things you've seen *once* to help estimate the count of things you've *never seen*. Word or N -gram (or any event) that occurs once is called a **singleton**.

Good-Turing smoothing

The Good-Turing formula provides another way to smooth probabilities.

Basic idea:

use the count of things you've seen *once* to help estimate the count of things you've *never seen*. Word or N -gram (or any event) that occurs once is called a **singleton**. In order to compute the frequency of **singletons**, we'll need to compute N_c , the number of event that occur c times. (Assumes that all item are binomially distributed.)

- Let N_r the number of items that occur r times.

Good-Turing smoothing

The Good-Turing formula provides another way to smooth probabilities.

Basic idea:

use the count of things you've seen *once* to help estimate the count of things you've *never seen*. Word or N -gram (or any event) that occurs once is called a **singleton**. In order to compute the frequency of **singletons**, we'll need to compute N_c , the number of event that occur c times. (Assumes that all item are binomially distributed.)

- Let N_r the number of items that occur r times.
- N_r can be used to provide a better estimate of r, given the binomial distribution.

Good-Turing smoothing

The Good-Turing formula provides another way to smooth probabilities.

Basic idea:

use the count of things you've seen *once* to help estimate the count of things you've *never seen*. Word or N -gram (or any event) that occurs once is called a **singleton**. In order to compute the frequency of **singletons**, we'll need to compute N_c , the number of event that occur c times. (Assumes that all item are binomially distributed.)

- Let N_r the number of items that occur r times.
- N_r can be used to provide a better estimate of r, given the binomial distribution.
- the adjusted frequency r^* is than:

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

Good-Turing smoothing

bigram

In case of bigram events Good-Turing assumes we know N_0, the number of bigrams we haven't seen.

Good-Turing smoothing

bigram

In case of bigram events Good-Turing assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

Good-Turing smoothing

bigram

In case of bigram events Good-Turing assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

revisited Good-Turing

In practice, the general discounted estimate c^* is not used for all counts c.

Good-Turing smoothing

bigram

In case of bigram events Good-Turing assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

revisited Good-Turing

In practice, the general discounted estimate c^* is not used for all counts c. First, large counts (where $c > k$ for some threshold k) are assumed to be reliable.

Good-Turing smoothing

bigram

In case of bigram events Good-Turing assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

revisited Good-Turing

In practice, the general discounted estimate c^* is not used for all counts c. First, large counts (where $c > k$ for some threshold k) are assumed to be reliable. Katz (1987) suggests setting k at 5.

Good-Turing smoothing

bigram

In case of bigram events Good-Turing assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

revisited Good-Turing

In practice, the general discounted estimate c^* is not used for all counts c. First, large counts (where $c > k$ for some threshold k) are assumed to be reliable. Katz (1987) suggests setting k at 5.
Thus we define:

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

Good-Turing smoothing

bigram

In case of bigram events Good-Turing assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

revisited Good-Turing

In practice, the general discounted estimate c^* is not used for all counts c. First, large counts (where $c > k$ for some threshold k) are assumed to be reliable. Katz (1987) suggests setting k at 5.
Thus we define:

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

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}$$

Good-Turing smoothing - Example

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

Bigram *frequencies* and *Good-Turing* re-estimations from the 22 million AP bigrams from Church and Gale (1991), and from the Berkeley Restaurant corpus of 9332 sentences

Backoff - Key idea

- Why are we treating all novel events as the same?

Backoff - Key idea

- Why are we treating all novel events as the same?
- p(zygote | see the) **vs.** p(baby | see the)
  - Suppose both trigrams have zero count

Backoff - Key idea

- Why are we treating all novel events as the same?
- p(zygote | see the) **vs.** p(baby | see the)
  - Suppose both trigrams have zero count
- *baby* beats *zygote* as a unigram

Backoff - Key idea

- Why are we treating all novel events as the same?
- p(zygote | see the) **vs.** p(baby | see the)
  - Suppose both trigrams have zero count
- *baby* beats *zygote* as a unigram
- *the baby* beats *the zygote* as a bigram

Backoff - Key idea

- Why are we treating all novel events as the same?
- $p(\text{zygote} \mid \text{see the})$ **vs.** $p(\text{baby} \mid \text{see the})$
  - Suppose both trigrams have zero count
- *baby* beats *zygote* as a unigram
- *the baby* beats *the zygote* as a bigram
- Shouldn't *see the baby* beat *see the zygote*?

Backoff smoothing

Key idea

If a n-gram $w_{i-n}, \ldots w_i$ is not in the training data, combine different order N-gram by linearly interpolating all the models.

Backoff smoothing

Key idea

If a n-gram $w_{i-n},\ldots w_i$ is not in the training data, combine different order N-gram by linearly interpolating all the models.

In trigram

Estimate the trigram probability as $P(w_i|w_{i-1}w_{i-2})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ:

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

such that the λs sum to 1:

$$\sum_i \lambda_i = 1$$

Backoff smoothing

Key idea

If a n-gram $w_{i-n}, \ldots w_i$ is not in the training data, combine different order N-gram by linearly interpolating all the models.

In trigram

Estimate the trigram probability as $P(w_i|w_{i-1}w_{i-2})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ:

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

such that the λs sum to 1:

$$\sum_i \lambda_i = 1$$

λ is the *confidence* weight for the longer n-gram.

Backoff smoothing

How estimate λ ?

Backoff smoothing

How estimate λ?

- In general λs are learned from a **held-out** corpus.

Backoff smoothing

How estimate λ?

- In general λs are learned from a **held-out** corpus.
- We can do this choosing the λ values which maximize the *likelihood* of the **held-out** corpus.

Backoff smoothing

How estimate λ?

- In general λs are learned from a **held-out** corpus.
- We can do this choosing the λ values which maximize the *likelihood* of the **held-out** corpus.
- One way is to use the *Expectation Maximization* (EM) algorithm.

Backoff smoothing - Katz backoff

Katz backoff variant

It is a version of backoff algorithm that uses Good-Turing discounting as well.

Backoff smoothing - Katz backoff

Katz backoff variant

It is a version of backoff algorithm that uses Good-Turing discounting as well.
In this model, if the N-gram we need has zero counts, we approximate it by baking off to the $(N-1)$-gram.

Backoff smoothing - Katz backoff

Katz backoff variant

It is a version of backoff algorithm that uses Good-Turing discounting as well.
In this model, if the N-gram we need has zero counts, we approximate it by baking off to the $(N-1)$-gram. We continue baking off until we reach a history that has some counts:

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

Backoff smoothing - Katz backoff

Katz backoff variant

It is a version of backoff algorithm that uses Good-Turing discounting as well.
In this model, if the N-gram we need has zero counts, we approximate it by baking off to the $(N-1)$-gram. We continue baking off until we reach a history that has some counts:

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

trigram version of Katz backoff

$$P_{\text{katz}}(w_i|w_{i-2}w_{i-1}) = \begin{cases} P^*(w_i|w_{i-2}w_{i-1}) & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{i-1}w_i)P^*(w_i|w_{i-1}) & \text{else if } C(w_{i-1}w_i) > 0 \\ \alpha(w_i)P^*(w_i) & \text{otherwise} \end{cases}$$

Katz backoff

Consideration

- Katz backoff gives us a better way to distribute the probability mass among unseen trigram events, by relying on information from unigram and bigram

Katz backoff

Consideration

- Katz backoff gives us a better way to distribute the probability mass among unseen trigram events, by relying on information from unigram and bigram
- We use discounting to tell us how much total probability mass to set aside for all the events we haven't seen, and backoff to tell us how to distribute this probability.

Katz backoff

Consideration

- Katz backoff gives us a better way to distribute the probability mass among unseen trigram events, by relying on information from unigram and bigram
- We use discounting to tell us how much total probability mass to set aside for all the events we haven't seen, and backoff to tell us how to distribute this probability.
- Why do we need α values?

Katz backoff

Consideration

- Katz backoff gives us a better way to distribute the probability mass among unseen trigram events, by relying on information from unigram and bigram
- We use discounting to tell us how much total probability mass to set aside for all the events we haven't seen, and backoff to tell us how to distribute this probability.
- Why do we need α values? Because without α weights, the result of equation would not be a true probability!

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

References

- SPEECH and LANGUAGE PROCESSING,
  Jurafsky & Martin, Chapter 4 - N-Grams
- Katz, S. M. (1987). *Estimation of probabilities from sparse data for the language model component of a speech recogniser.*
  IEEE Transactions on Acoustics, Speech, and Signal Processing