ngram-discount

NAME

ngram-discount - notes on the N-gram smoothing implementations in SRILM

NOTATION

a_z
: An N-gram where *a* is the first word, *z* is the last word, and "_" represents 0 or more words in between.

p(*a_z*)
: The estimated conditional probability of the *n*th word *z* given the first *n*-1 words (*a_*) of an N-gram.

*a_*
: The *n*-1 word prefix of the N-gram *a_z*.

_z
: The *n*-1 word suffix of the N-gram *a_z*.

c(*a_z*)
: The count of N-gram *a_z* in the training data.

n(*_*z*)
: The number of unique N-grams that match a given pattern. ``(*)" represents a wildcard matching a single word.

n1,*n*[1]
: The number of unique N-grams with count = 1.

DESCRIPTION

N-gram models try to estimate the probability of a word *z* in the context of the previous *n*-1 words (*a_*), i.e., *Pr*(*z*|*a_*). We will denote this conditional probability using*p*(*a_z*) for convenience. One way to estimate *p*(*a_z*) is to look at the number of times word *z* has followed the previous *n*-1 words (*a_*):

(1) $p(a\_z) = c(a\_z) / c(a\_)$

This is known as the maximum likelihood (ML) estimate. Unfortunately it does not work very well because it assigns zero probability to N-grams that have not been observed in the training data. To avoid the zero probabilities, we take some probability mass from the observed N-grams and distribute it to unobserved N-grams. Such redistribution is known as smoothing or discounting.

Most existing smoothing algorithms can be described by the following equation:

(2) $p(a\_z) = (c(a\_z) > 0) \ ? \ f(a\_z) \ : \ \mathrm{bow}(a\_) \ p(\_z)$

If the N-gram *a_z* has been observed in the training data, we use the distribution *f*(*a_z*). Typically *f*(*a_z*) is discounted to be less than the ML estimate so we have some leftover probability for the *z* words unseen in the context (*a_*). Different algorithms mainly differ on how they discount the ML estimate to get *f*(*a_z*).

If the N-gram *a_z* has not been observed in the training data, we use the lower order distribution *p*(*_z*). If the context has never been observed (*c*(*a_*) = 0), we can use the lower order distribution directly (bow(*a_*) = 1). Otherwise we need to compute a backoff weight (bow) to make sure probabilities are normalized: Sum_*z* *p*(*a_z*) = 1

Let *Z* be the set of all words in the vocabulary, *Z0* be the set of all words with *c*(*a_z*) = 0, and *Z1* be the set of all words with *c*(*a_z*) > 0. Given *f*(*a_z*), bow(*a_*) can be determined as follows:

(3) Sum_*Z* *p*(*a_z*) = 1
Sum_*Z1* *f*(*a_z*) + Sum_*Z0* bow(*a_*) *p*(*_z*) = 1
bow(*a_*) = (1 - Sum_*Z1* *f*(*a_z*)) / Sum_*Z0* *p*(*_z*)
= (1 - Sum_*Z1* *f*(*a_z*)) / (1 - Sum_*Z1* *p*(*_z*))
= (1 - Sum_*Z1* *f*(*a_z*)) / (1 - Sum_*Z1* *f*(*_z*))

Smoothing is generally done in one of two ways. The backoff models compute *p*(*a_z*) based on the N-gram counts *c*(*a_z*) when *c*(*a_z*) > 0, and only consider lower order counts *c*(*_z*)when *c*(*a_z*) = 0. Interpolated models take lower order counts into account when *c*(*a_z*) > 0 as well. A common way to express an interpolated model is:

(4) *p*(*a_z*) = *g*(*a_z*) + bow(*a_*) *p*(*_z*)

Where *g*(*a_z*) = 0 when *c*(*a_z*) = 0 and it is discounted to be less than the ML estimate when *c*(*a_z*) > 0 to reserve some probability mass for the unseen *z* words.
Given *g*(*a_z*),bow(*a_*) can be determined as follows:

(5) Sum_*Z* *p*(*a_z*) = 1
Sum_*Z1* *g*(*a_z*) + Sum_*Z* bow(*a_*) *p*(*_z*) = 1
bow(*a_*) = 1 - Sum_*Z1* *g*(*a_z*)

An interpolated model can also be expressed in the form of equation (2), which is the way it is represented in the ARPA format model files in SRILM:

(6) *f*(*a_z*) = *g*(*a_z*) + bow(*a_*) *p*(*_z*)
p(*a_z*) = (*c*(*a_z*) > 0) ? *f*(*a_z*) : bow(*a_*) *p*(*_z*)

Most algorithms in SRILM have both backoff and interpolated versions. Empirically, interpolated algorithms usually do better than the backoff ones, and Kneser-Ney does better than others.

OPTIONS

This section describes the formulation of each discounting option in ngram-count(1). After giving the motivation for each discounting method, we will give expressions for $f(a\_z)$ and bow(a_) of Equation 2 in terms of the counts. Note that some counts may not be included in the model file because of the **-gtmin** options; see Warning 4 in the next section.

Backoff versions are the default but interpolated versions of most models are available using the **-interpolate** option. In this case we will express $g(a\_z)$ and bow(a_) of Equation 4 in terms of the counts as well. Note that the ARPA format model files store the interpolated models and the backoff models the same way using $f(a\_z)$ and bow(a_);see Warning 3 in the next section. The conversion between backoff and interpolated formulations is given in Equation 6.

The discounting options may be followed by a digit (1-9) to indicate that only specific N-gram orders be affected. See ngram-count(1) for more details.

-cdiscount *D*

Ney's absolute discounting using *D* as the constant to subtract. *D* should be between 0 and 1. If *Z1* is the set of all words *z* with $c(a\_z) > 0$:

$$f(a\_z) = (c(a\_z) - D) \;/\; c(a\_)$$
$$p(a\_z) = (c(a\_z) > 0)\ ?\ f(a\_z)\ :\ \text{bow}(a\_)\ p(\_z) \quad ;\ \text{Eqn. 2}$$
$$\text{bow}(a\_) = (1 - \text{Sum}\_Z1\ f(a\_z)) \;/\; (1 - \text{Sum}\_Z1\ f(\_z)) \quad ;\ \text{Eqn. 3}$$

With the **-interpolate** option we have:

$$g(a\_z) = \max(0,\ c(a\_z) - D) \;/\; c(a\_)$$
$$p(a\_z) = g(a\_z) + \text{bow}(a\_)\ p(\_z) \quad ;\ \text{Eqn. 4}$$
$$\text{bow}(a\_) = 1 - \text{Sum}\_Z1\ g(a\_z) \quad ;\ \text{Eqn. 5}$$
$$= D\ n(a\_*) \;/\; c(a\_)$$

The suggested discount factor is:

$$D = n1 \;/\; (n1 + 2{*}n2)$$

where *n1* and *n2* are the total number of N-grams with exactly one and two counts, respectively. Different discounting constants can be specified for different N-gram orders using options **-cdiscount1**, **-cdiscount2**, etc.

-kndiscount and **-ukndiscount**

Kneser-Ney discounting. This is similar to absolute discounting in that the discounted probability is computed by subtracting a constant *D* from the N-gram count. The options **-kndiscount** and **-ukndiscount** differ as to how this constant is computed. The main idea of Kneser-Ney is to use a modified probability estimate for lower order N-grams used for backoff. Specifically, the modified probability for a lower order N-gram is taken to be proportional to the number of unique words that precede it in the training data. With discounting and normalization we get:

$$f(a\_z) = (c(a\_z) - D0) \;/\; c(a\_) \quad ;;\ \text{for highest order N-grams}$$
$$f(\_z) = (n(*\_z) - D1) \;/\; n(*\_*) \quad ;;\ \text{for lower order N-grams}$$

where the $n(*\_z)$ notation represents the number of unique N-grams that match a given pattern with (*) used as a wildcard for a single word. *D0* and *D1* represent two different discounting constants, as each N-gram order uses a different discounting constant. The resulting conditional probability and the backoff weight is calculated as given in equations (2) and (3):

$p(a\_z) = (c(a\_z) > 0) \ ? \ f(a\_z) \ : \ \text{bow}(a\_) \ p(\_z)$; Eqn. 2

$\text{bow}(a\_) = (1 - \text{Sum\_}Z1 \ f(a\_z)) \ / \ (1 - \text{Sum\_}Z1 \ f(\_z))$; Eqn. 3

The option **-interpolate** is used to create the interpolated versions of **-kndiscount** and **-ukndiscount**. In this case we have:

$p(a\_z) = g(a\_z) + \text{bow}(a\_) \ p(\_z)$; Eqn. 4

Let *Z1* be the set $\{z: c(a\_z) > 0\}$. For highest order N-grams we have:

$g(a\_z) = \max(0, \ c(a\_z) - D) \ / \ c(a\_)$

$\text{bow}(a\_) = 1 - \text{Sum\_}Z1 \ g(a\_z)$

$= 1 - \text{Sum\_}Z1 \ c(a\_z) \ / \ c(a\_) + \text{Sum\_}Z1 \ D \ / \ c(a\_)$

$= D \ n(a\_*) \ / \ c(a\_)$

Let *Z2* be the set $\{z: n(*\_z) > 0\}$. For lower order N-grams we have:

$g(\_z) = \max(0, \ n(*\_z) - D) \ / \ n(*\_*)$

$\text{bow}(\_) = 1 - \text{Sum\_}Z2 \ g(\_z)$

$= 1 - \text{Sum\_}Z2 \ n(*\_z) \ / \ n(*\_*) + \text{Sum\_}Z2 \ D \ / \ n(*\_*)$

$= D \ n(\_*) \ / \ n(*\_*)$

The original Kneser-Ney discounting (**-ukndiscount**) uses one discounting constant for each N-gram order. These constants are estimated as

$D = n1 \ / \ (n1 + 2*n2)$

where *n1* and *n2* are the total number of N-grams with exactly one and two counts, respectively.

Chen and Goodman's modified Kneser-Ney discounting (**-kndiscount**) uses three discounting constants for each N-gram order, one for one-count N-grams, one for two-count N-grams, and one for three-plus-count N-grams:

$Y = n1/(n1+2*n2)$

$D1 = 1 - 2Y(n2/n1)$

$D2 = 2 - 3Y(n3/n2)$

$D3+ = 3 - 4Y(n4/n3)$

Warning:

SRILM implements Kneser-Ney discounting by actually modifying the counts of the lower order N-grams. Thus, when the **-write** option is used to write the counts with **-kndiscount** or **-ukndiscount**, only the highest order N-grams and N-grams that start with <s> will have their regular counts $c(a\_z)$, all others will have the modified counts $n(*\_z)$ instead. See Warning 2 in the next section.

-wbdiscount

Witten-Bell discounting. The intuition is that the weight given to the lower order model should be proportional to the probability of observing an unseen word in the current context ($a\_$). Witten-Bell computes this weight as:

$\text{bow}(a\_) = n(a\_*) \ / \ (n(a\_*) + c(a\_))$

Here $n(a\_*)$ represents the number of unique words following the context ($a\_$) in the training data. Witten-Bell is originally an interpolated discounting method. So with the **-interpolate** option we get:

$g(a\_z) = c(a\_z) \ / \ (n(a\_*) + c(a\_))$

$p(a\_z) = g(a\_z) + \text{bow}(a\_)\ p(\_z)$; Eqn. 4

Without the **-interpolate** option we have the backoff version which is implemented by taking *f*(*a_z*) to be the same as the interpolated *g*(*a_z*).

$$f(a\_z) = c(a\_z) \ / \ (n(a\_*) + c(a\_))$$
$$p(a\_z) = (c(a\_z) > 0)\ ?\ f(a\_z)\ :\ \text{bow}(a\_)\ p(\_z) \quad ;\ \text{Eqn. 2}$$
$$\text{bow}(a\_) = (1 - \text{Sum\_}Z1\ f(a\_z)) \ / \ (1 - \text{Sum\_}Z1\ f(\_z)) \quad ;\ \text{Eqn. 3}$$

-ndiscount

Ristad's natural discounting law. See Ristad's technical report "A natural law of succession" for a justification of the discounting factor. The **-interpolate** option has no effect, only a backoff version has been implemented.

$$f(a\_z) = \frac{c(a\_z)}{c(a\_)} \ \frac{c(a\_)\ (c(a\_) + 1) + n(a\_*)\ (1 - n(a\_*))}{c(a\_)^2 + c(a\_) + 2\ n(a\_*)}$$

$$p(a\_z) = (c(a\_z) > 0)\ ?\ f(a\_z)\ :\ \text{bow}(a\_)\ p(\_z) \quad ;\ \text{Eqn. 2}$$
$$\text{bow}(a\_) = (1 - \text{Sum\_}Z1\ \text{f}(a\_z)) \ / \ (1 - \text{Sum\_}Z1\ f(\_z)) \quad ;\ \text{Eqn. 3}$$

-count-lm

Estimate a count-based interpolated LM using Jelinek-Mercer smoothing (Chen & Goodman, 1998), also known as "deleted interpolation." Note that this does not produce a backoff model; instead of count-LM parameter file in the format described in ngram(1) needs to be specified using **-init-lm**, and a reestimated file in the same format is produced. In the process, the mixture weights that interpolate the ML estimates at all levels of N-grams are estimated using an expectation-maximization (EM) algorithm. The options **-em-iters** and **-em-delta** control termination of the EM algorithm. Note that the N-gram counts used to estimate the maximum-likelihood estimates are specified in the **-init-lm** model file. The counts specified with **-read** or **-text** are used only to estimate the interpolation weights. \" ???What does this all mean in terms of the math???

-addsmooth *D*

Smooth by adding *D* to each N-gram count. This is usually a poor smoothing method, included mainly for instructional purposes.

$$p(a\_z) = (c(a\_z) + D) \ / \ (c(a\_) + D\ n(*))$$

default

If the user does not specify any discounting options, **ngram-count** uses Good-Turing discounting (aka Katz smoothing) by default. The Good-Turing estimate states that for any N-gram that occurs *r* times, we should pretend that it occurs *r*' times where

$$r' = (r+1)\ n[r+1]/n[r]$$

Here *n*[*r*] is the number of N-grams that occur exactly *r* times in the training data. Large counts are taken to be reliable, thus they are not subject to any discounting. By default unigram counts larger than 1 and other N-gram counts larger than 7 are taken to be reliable and maximum likelihood estimates are used. These limits can be modified using the **-gt***n***max** options.

$$f(a\_z) = (c(a\_z) \ / \ c(a\_)) \quad \text{if } c(a\_z) > gtmax$$

The lower counts are discounted proportional to the Good-Turing estimate with a small correction *A* to account for the high-count N-grams not being discounted. If 1 <=*c*(*a_z*) <= *gtmax*:

$$A = (gtmax + 1)\ \frac{n[gtmax + 1]}{n[1]}$$

$$c'(a\_z) = (c(a\_z) + 1)\ \frac{n[c(a\_z) + 1]}{n[c(a\_z)]}$$

$$f(a\_z) = \frac{c(a\_z)}{c(a\_)}\ \frac{(c'(a\_z) / c(a\_z) - A)}{(1 - A)}$$

The **-interpolate** option has no effect in this case, only a backoff version has been implemented, thus:

$p(a\_z) = (c(a\_z) > 0)\ ?\ f(a\_z)\ :\ \mathrm{bow}(a\_)\ p(\_z)$; Eqn.2

$\mathrm{bow}(a\_) = (1 - \mathrm{Sum}\_Z1\ f(a\_z))\ /\ (1 - \mathrm{Sum}\_Z1\ f(\_z))$; Eqn.3

FILE FORMATS

SRILM can generate simple N-gram counts from plain text files with the following command:

ngram-count -order *N* -text *file.txt* -write *file.cnt*

The **-order** option determines the maximum length of the N-grams. The file *file.txt* should contain one sentence per line with tokens separated by whitespace. The output *file.cnt*contains the N-gram tokens followed by a tab and a count on each line:

a_z <tab> *c*(*a_z*)

A couple of warnings:

Warning 1

SRILM implicitly assumes an <s> token in the beginning of each line and an </s> token at the end of each line and counts N-grams that start with <s> and end with </s>. You do not need to include these tags in *file.txt*.

Warning 2

When **-kndiscount** or **-ukndiscount** options are used, the count file contains modified counts. Specifically, all N-grams of the maximum order, and all N-grams that start with <s> have their regular counts *c*(*a_z*), but shorter N-grams that do not start with <s> have the number of unique words preceding them *n*(**a_z*) instead. See the description of **-kndiscount** and **-ukndiscount** for details.

For most smoothing methods (except **-count-lm**) SRILM generates and uses N-gram model files in the ARPA format. A typical command to generate a model file would be:

ngram-count -order *N* -text *file.txt* -lm *file.lm*

The ARPA format output *file.lm* will contain the following information about an N-gram on each line:

log10(*f*(*a_z*)) <tab> *a_z* <tab> log10(bow(*a_z*))

Based on Equation 2, the first entry represents the base 10 logarithm of the conditional probability (logprob) for the N-gram *a_z*. This is followed by the actual words in the N-gram separated by spaces. The last and optional entry is the base-10 logarithm of the backoff weight for (*n*+1)-grams starting with *a_z*.

Warning 3

Both backoff and interpolated models are represented in the same format. This means interpolation is done during model building and represented in the ARPA format with logprob and backoff weight using equation (6).

Warning 4

Not all N-grams in the count file necessarily end up in the model file. The options **-gtmin**, **-gt1min**, ..., **-gt9min** specify the minimum counts for N-grams to be included in the LM (not only for Good-Turing discounting but for the other methods as well). By default all unigrams and bigrams are included, but for higher order N-grams only those with count >= 2 are included. Some exceptions arise, because if one N-gram is included in the model file, all its prefix N-grams have to be included as well. This causes some higher order 1-count N-grams to be included when using KN discounting, which uses modified counts as described in Warning 2.

Warning 5

Not all N-grams in the model file have backoff weights. The highest order N-grams do not need a backoff weight. For lower order N-grams backoff weights are only recorded for those that appear as the prefix of a longer N-gram included in the model. For other lower order N-grams the backoff weight is implicitly 1 (or 0, in log representation).

SEE ALSO

ngram(1), ngram-count(1), ngram-format(5),
S. F. Chen and J. Goodman, ``An Empirical Study of Smoothing Techniques for Language Modeling," TR-10-98, Computer Science Group, Harvard Univ., 1998.

BUGS

Work in progress.

AUTHOR

Deniz Yuret <dyuret@ku.edu.tr>
Andreas Stolcke <stolcke@speech.sri.com>