A Continuous Good-Turing Estimate

Adam Hesterberg

January 15, 2009

1 Introduction

If we run an experiment ten times with mutually exclusive outcomes X, Y, and Z whose probabilities are known, the binomial theorem gives the probability that there will be, say, 7 Xs, 3 Ys, and 0 Zs. Going the other way is more interesting: given that there were 7 Xs, 3 Ys, and 0 Zs, can we infer the best probabilities of X, Y, and Z? For instance, this is what we do when we train an n-gram language model.[1] Of course, this depends on what we mean by "best." In many contexts, the maximum likelihood estimate (here $P(X) = .7$, $P(Y) = .3$, $P(Z) = 0$) is ideal. If, however, we have some prior knowledge about the distribution of probabilities, or want not the maximum likelihood but minimum entropy, or have few observations relative to the number of possible outcomes, then the maximum likelihood estimate is not necessarily ideal.

All three of the above apply to language models: we know a priori that no

[1]My work on this started in a computational linguistics REU under the mentorship of Brian Roark (roark@cslu.ogi.edu, http://www.cslu.ogi.edu/people/roark/), but most of what appears here is new.

sequence of n words in human language, or "n-gram," should have probability 0; we want minimum entropy to minimize the language's information content; and, since all language models are trained on a finite amount of data, the number of words in a training corpus is usually some small number like 10^8 or 10^{12}, and on data sets of such sizes, there will certainly be types of language use that don't show up, but should be given a nonzero probability. For instance, if one trains a trigram language model by reading in trigrams from a corpus of 10^8 words and decreeing the probability of each trigram in the language to be its frequency there, then no trigram will be assigned a probability between 0 and 10^{-8}. However, this is exactly the range where *most* trigrams must fall: if there are, say, 10^5 words in the lexicon, then there are 10^{15} trigrams, at most 10^8 of which can have frequency at least 10^{-8} in the corpus; this leaves the overwhelming majority between 0 and 10^{-8}. Algorithms to correct this are called "smoothing" algorithms, after that we'd like to "smooth" some of the probability from more frequently observed events to those never observed.

2 Background

The greatest advancement in the history of smoothing was the idea to simply add a constant to the number of observations of each n-gram, as if one had simply appended the dictionary to the training data. This was the greatest advancement not because of any of its own merits (it's easy to compute and explain, but not good for much else), but because, by avoiding zero counts at all, it reduced entropy from infinity (before, when an n-gram with count 0 in the training data was observed, a probability of 0 would be assigned, giving infinite entropy) to something finite; since then we've just been chipping away at fractions of bits.

There are many popular smoothing techniques, and most of them rely on the "Good-Turing Estimate," which uses the distribution of observation counts—that an event was observed 0 times gives relatively little information, but the number of such events is useful. For a statement, derivation, and applications of this, including several of the most popular smoothing methods based on it, see Chen and Goodman's summary of smoothing techniques[2]

There are three problems with the Good-Turing estimate, however. The first is aesthetic: when Chen and Goodman wrote their seminal paper, they thought the original derivation so ugly that they created their own, a mess of binomial coefficients and sums that was nevertheless a significant improvement. The second is with high counts, which tend to have low counts of counts—when the counts of counts approach 0, the reliability of the Good-Turing estimate decreases, just when we should have the most information, since there's a high count. The third is generalizability: the estimate and its two derivations require the restriction of counts to integers. If, however, one wished to train a language model not from a reliable corpus but from, say, speech recognizer output consisting of words labeled with probabilities, then the above objections to the ML estimate would still apply. However, we cannot apply Good-Turing directly here, because it requires that the number of events with a given count be large—there may be many n-grams observed 3 times, but probably 0 observed exactly π times.

[2]Stanley Chen and Joshua Goodman. *An Empirical Study of Smoothing Techniques for Language Modeling.* 1998. http://research.microsoft.com/j̃oshuago/tr-10-98.pdf.

3 Methods

We first mathematically derive a generalization of the Good-Turing estimate; later we'll describe testing it. Specifically, we can calculate in two ways the expected value of the probability that an observation w is of an event seen exactly $r+1$ times in $N+1$ samples. First, some notation:

1. $E_N(X)$ is the expected value of X when we make N observations.
2. $c(X)$ is the number of observations of the event X
3. $|\{X\}|$ is the number of elements of X.
4. $p(X|Y)$ is the probability of X given Y.

Now, we want to calculate in two ways

$$E_{N+1}(p(c(w) = r+1)),$$

that is, the expected value of the probability that an observed word is one with count $r+1$. On one hand, if an event is seen $r+1$ times in $N+1$ samples, it contributes $\frac{r+1}{N+1}$ to the probability, and we can simply multiply by the expected number of events whose count is $r+1$, that is,

$$\frac{r+1}{N+1} E_{N+1}(|\{w : c(w) = r+1\}|) = E_{N+1}(p(c(w) = r+1)).$$

On the other hand, our observation w will be of an event seen exactly $r+1$ times if and only if the same event occurs exactly r times in the other N observations. The expected number of such events is $E_N(|\{w : c(w) = r\}|)$, and the probability of our observation w given that it's of one of those events is $E_N(p(w)|c(w) = r)$, so

$$E_{N+1}(p(c(w) = r+1)) = E_N(|\{w : c(w) = r\}|)E_N(p(w)|c(w) = r).$$

Putting these together gives the estimate

$$E_N(p(w)|c(w) = r) = \frac{r+1}{N+1}\frac{E_{N+1}(|\{w : c(w) = r+1\})}{E_N(|\{w : c(w) = r\}|)}.$$

If we make the approximation that for large N, $E_N(|\{w : c(w) = r\}|)$ is approximately $|\{w : c(w) = r\}|$ in one sample, this gives the probability of a word with count r in terms of the distribution of counts; it's exactly the Good-Turing estimate.

Already, this fixes one of the three noted problems with the Good-Turing Estimate, the arbitrariness of its derivation. This nicer derivation also generalizes easily to intervals of counts, allowing us to apply Good-Turing to the continuous case and for high counts. The argument is the same; if I is an interval of counts, then:

$$\begin{aligned}\frac{1}{N+1}E_{N+1}(\sum_{i:c(w_i)\in I+1} c(w_i)) =& E_{N+1}(p(c(w) \in I+1)) \\ =& E_N(|\{w : c(w) \in I\}|)E_N(p(w)|c(w) \in I) \\ =& E_N(|\{w : c(w) \in I\}|)\frac{r^*}{N} \\ r^* \approx & \frac{\sum_{w:c(w)\in I+1} c(w)}{\sum_{w:c(w)\in I} 1}.\end{aligned}$$

The last line gives the corrected count r^* of an event seen r times in terms of events seen in some intervals I and $I + 1$ around r and $r + 1$; the size of these intervals can be varied according to the scarcity of data. In particular, in the testing we use three intervals: the smallest interval centered at r with at least 5 words, the smallest interval above r with at least 5 words (if it exists), and the smallest interval below r with at least 5 words, then average the results.

A full test of the efficacy of this method would be prohibitively difficult, because Good-Turing is only the foundation of most smoothing algorithms;

in actual applications one would build Good-Turing into a more complicated smoothing algorithm taking into account some of the structure of language and sub-grams of n-grams[3], so we instead test this new Good-Turing estimate against no smoothing at all, additive smoothing, and the old Good-Turing. The last of these is the most relevant baseline with which to compare the new Good-Turing, since it's what we'd replace; this comparison isn't perfect because we can't tell whether improvements in Good-Turing would actually trickle up to the full smoothing algorithms, but if the new Good-Turing does better than the old one on raw data, it would be reasonable to assume that it does better in actual use. Also, continuous data on which to test this are not readily available, so we'll content ourselves to test it on discrete data; for the continuous case (as in, say, language modeling for resource-poor languages), there currently is no competitor to the proposed variant Good-Turing. Hence all we test here is the discrete case.

Some details of the testing: we divide the Brown corpus into 15 parts. A language model was trained from each of them and smoothed with each of the four smoothing methods list above, then tested on each of the other 14 sections. All $15 \cdot 14 \cdot 4$ results are printed. Also, the dictionary is the union of all words that appear in the Brown corpus.

The attached tarball contains the code used for the testing described above in NLTK. To run it, just import everything nltk, then everything from it, then run print_final_results().

[3]This was, in fact, an entire thesis at Harvard by Chen & Goodman, above.

4 Materials

We test using NLTK's included Brown corpora, processed down to counts of n-grams.

5 Results

- Average entropy per word without smoothing: ∞.
- Average entropy per word with Klingon smoothing: 7.573
- Average entropy per word with old GT smoothing: 1.753
- Average entropy per word with my GT smoothing: 7.546

These results are disappointing—the new Good-Turing smoothing did indeed run, but barely performed better than the baseline Klingon smoothing algorithm. There are two possible reasons for this. First, there could be an uncaught bug in the code used to test the method, but the author spent 5 hours looking for such a bug, and thinks it unlikely. Second, some of the kludgy approximations used to make the smoothing calculations easier may have thrown off the results. For instance, the 10 most frequently occurring words took almost all of the time in early runs of the code—there is probably a data structure that avoids this, but in the attached code, the 10 most frequently occurring words were simply calculated differently, and since they're the most frequently occurring, errors in them are particularly significant. Manual inspection of some of their calculated values shows that they're of reasonable magnitude, but could be off. Also, the choice of intervals described above (for any count, one interval entirely below that count, one entirely above, and one in the middle) was arbitrary, made on the theory that they'd be easy to code and probably not significantly worse than a better way of choosing the intervals, but I have no proof of this; the proper choice of intervals isn't contained in the nice derivation above.

6 Conclusion

Ultimately, however, the new Good-Turing method isn't competing with the old one, but extending it to places where no good smoothing method exists. For more conservative choices of intervals, the new Good-Turing estimate *equals* the old one, so certainly for small intervals they're fairly close; so the new Good-Turing estimate is almost as good when used conservatively, and can still be extended to the continuous case. It won't revolutionize smoothing, but it's better than nothing when no other methods exist.