Good-Turing Smoothing Without Tears

William A. Gale
AT&T Bell Laboratories
P. O. Box 636
Murray Hill, NJ, 07974-0636

gale@research.att.com

ABSTRACT

The performance of statistically based techniques for many tasks such as spelling correction, sense disambiguation, and translation is improved if one can estimate a probability for an object of interest which has not been seen before. Good-Turing methods are one means of estimating these probabilities for previously unseen objects. However, the use of Good-Turing methods requires a smoothing step which must smooth in regions of vastly different accuracy. Such smoothers are difficult to use, and may have hindered the use of Good-Turing methods in computational linguistics.

This paper presents a method which uses the simplest possible smooth, a straight line, together with a rule for switching from Turing estimates which are more accurate at low frequencies. We call this method the Simple Good-Turing (SGT) method. Two examples, one from prosody, the other from morphology, are used to illustrate the SGT.

While the goal of this research was to provide a simple estimator, the SGT turns out to be the most accurate of several methods applied in a set of Monte Carlo examples which satisfy the assumptions of the Good-Turing methods. The accuracy of the SGT is compared to two other methods for estimating the same probabilities, the Expected Likelihood Estimate (ELE) and two way cross validation. The SGT method is more complex than ELE but simpler than two way cross validation. On a set of twenty Monte Carlo examples spanning vocabulary sizes from 5,000 to 100,000 and Zipfian coefficients from -1.4 to -1.1, the SGT was far more accurate than ELE. It was more accurate than two way cross validation for frequencies of two or less, and of approximately the same accuracy for higher frequencies.

The importance of using the internal structure of objects to get differing estimates for the probabilities of unseen objects is discussed, and two examples show possible approaches.

Appendices present the complete data needed to apply Good-Turing methods to the prosody and Chinese plurals examples, and code that implements the SGT.

1. Why Good-Turing Smoothing is Useful in Linguistics

Many linguistic phenomena of great importance, such as words, word sequences, and syntactically related pairs of words, are essentially infinite in number. That is, given any finite amount of text or speech in the language, one expects to see new words, etc., in another equal sized sample of the language. Furthermore, the performance of statistically based techniques for tasks such as spelling correction, sense disambiguation, and machine translation is improved if one can estimate a probability for an object of interest which has not been seen before. It is typically vital that the probability of the unseen objects not be estimated as zero.

Good-Turing methods provide a simple estimate of the total probability of the objects not seen. How this probability is divided among the objects depends on the objects, especially on any structure (such as the component words of a bigram) the objects may have.

Good-Turing methods also estimate probabilities for observed objects that are consistent with the total

probability assigned to the unseen objects. These probabilities for the observed objects have not been easy to estimate, and may have deterred many from using Good-Turing methods. It is a major point of this paper that a simple method for treating the observed objects gives better accuracy than several other methods. In particular, the paper will compare the simple Good-Turing (SGT) method proposed to adding one half to each observation (ELE method), and to two way cross validation.

The paper aims to explain what the Good-Turing method is about in an intuitive fashion, and to show a simple method for treating the observed objects. Four examples will be used to illustrate the various points.

2. What Good-Turing Methods are About

Consider the following example taken from some ongoing prosody work with Joan Bachenko. An input speech signal is classified into a string of three symbols: C, R, and V, representing consonants, reduced vowels, and full vowels. The full vowels can be reliably recognized, and we consider as objects all sequences beginning and ending in a full vowel. While our interest centers on estimating the likelihood of such an object given the time between the full vowels, we also need prior probabilities for the objects. The following table shows some of the objects and their frequencies.

Table 1
Examples of Prosody Objects

object	frequency
VCV	7846
VCCV	6925
VCRCV	2395
VCCRCRCV	224
VCCCCRCV	23
VCCRCRCRV	2
VRCRCRV	1

The first line shows that in the TIMIT data base, from which the data were compiled, the sequence VCV occurred 7846 times (and was the most frequent object). A version of Table 1 with all objects shows that there were 120 objects which occurred exactly once, 40 objects that occurred exactly twice, etc., as shown in part by the following table, and completely in Appendix I.

Table 2
Partial Table of
Frequency of Frequencies for Prosody Example

frequency	frequency of frequency
r	N_r
1	120
2	40
3	24
4	13
5	15
6	5
7	11
8	2
9	2
10	1
11	0
12	3

Since there are 120 objects which occur just once, 120 is the frequency of frequency 1. Table 2 shows frequencies 1 through 12 and their frequencies. Notice that the frequency of 1 is the largest, with the frequency of frequencies declining from there, and becoming increasingly noisy. These are typical patterns for linguistic data.

It is usual to let r be a frequency, and N_r be the frequency of frequency r. Thus for the prosody example, $N_8 = 2$, while $N_{11} = 0$. Also, let $N = \Sigma r N_r$ be the total number of objects observed.

A useful part of Good-Turing methodology is the estimate that the total probability of all unseen objects is N_1/N. For the prosody example, N is 30902, so we estimate the total probability of unseen objects as 120/30902 = .0039.

Let p_r be the probability that we estimate for each of the objects seen r times. An important notation in discussions of Good-Turing methods is the definition of r^* to be such that $p_r \equiv r^*/N$. As an example, the ordinary maximum likelihood estimate (MLE) is $p_r = r/N$, so for the MLE, $r^* = r$. Since the MLE estimates $p_0 = 0$, and we are only interested in methods which can give estimates of p_0 which exceed zero, the MLE will not be considered further.

The remaining part of Good-Turing methods is a set of estimates for r^*, $r \geq 1$, such that the total probability for the all objects adds to 1. That is, we must reduce the total probability of the objects which *were* seen to be less than one in order to be consistent with estimating non-zero probabilities for unseen objects. A precise statement of the theorem underlying the Good-Turing methods [Church, Gale, and Kruskal, 1991] is that

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

where $E(x)$ represents expectation of the random variable x. The total probability of the unseen objects is precisely stated as $E(N_1)/N$. N_1 is typically the largest and best measured of the N_r. Thus it is not unreasonable to substitute N_1 for $E(N_1)$. When the substitution of N_r for $E(N_r)$ is made, we term the estimator the *Turing* estimator. However, the larger r is, the less reasonable this substitution is, as Table 2, and especially N_{11} show.

Therefore, Good (1953) suggested that replace the observed N_r with smoothed values, $S(N_r)$ in the above

equation for r^*. When this approach is taken we term the estimator a *Good-Turing* estimator. There can be many Good-Turing estimators depending on what smoothing process is used. Unfortunately, the uncertainties in the N_r vary greatly with r, and smoothing such data is difficult. Good's 1953 paper spends about a quarter of its space on various smoothing methods.

But before we discuss the details of smoothing, let us ask what the basis for the estimate of r^* given above is. Consider the prosody example, and suppose we had 120 objects each with probability 1/30902 and a binomial distribution from 30902 drawings. Then the expected distribution of total observations of the objects is shown in the following table:

Table 3
Example of Binomial Distribution

times seen	0	1	2	3	4	5	6	7
expected number	44.15	44.15	22.07	7.35	1.84	0.37	0.06	.01

The table shows that in our thought experiment, we would expect to find that 44 of the words had not appeared, and that only 44 of the words would be expected to be seen with frequency one again. So why did we get 120 observations of objects appearing once? The answer is that many objects with lower and higher probabilities than 1/30902 can be expected to appear just once in 30902 observations when *they* have binomial distributions. The essence of the theorems on the Good-Turing method is to count up the expected contribution from each object's true probability to each of the frequencies when each object has a binomial distribution. These totals then constrain the probability of objects seen r times, as stated in the result of the theorem. Consideration of this example will give insight into the nature of the Good-Turing methods, and may make the mathematics in [Church, Gale, and Kruskal, 1991] more understandable.

3. A Simple Method for Smoothing N_r

Consider another example of the use of Good-Turing methods. It comes from ongoing work with Richard Sproat and Chilin Shih on the segmentation of Chinese text into words. The example data were collected by Nancy Chang who joined us for a summer. Our dictionary does not list the plurals of words, but many Chinese words form plurals morphologically by the addition of the character "men2". However, we did not know whether this group of words was an open set or a (fairly large) closed set. If the class is closed, then we would just add the plurals to the dictionary, while if it is open, then we would add special code to allow for the possibility of the plural of a word.

The objects in this example are Chinese words with an attached "men2". The character "men2" occurs only at the end of a word, so these words could be detected reliably. The most frequently encountered word is the word for person, occurring in plural form in our corpus 1918 times. The following table, analogous to Table 2, shows some of the r and N_r for this example. The complete set of values is again in Appendix I.

Table 4
Frequency of Frequencies for Chinese Plurals Example

frequency r	frequency of frequency N_r
1	268
2	112
3	70
4	41
5	24
6	14
7	15
400	1
1918	1

The question of primary interest here is whether the class is open or closed. The question is best answered by considering the Turing estimator $1^* \simeq 2*N_2/N_1 = 2*112/268 = .84$, which is convincingly less than unity. The sign of a closed class is that $1^* > 1$. For, with a closed class, one will soon see most of the objects, and the frequency of objects seen just once will also become small. The size of the closed class matters also, of course. In this case we can conclude that the class of Chinese words forming a plural by the addition of "men2" is either open or considerably larger than the 683 types that we saw among the 6551 tokens. Having decided that the class is open, we are still interested in the probability of both observed and unseen plurals.

The range of both r and N_r in this example and in the prosody example is very large, and this is typical of linguistic applications. This suggests that when we make plots, we use the logarithms of both variables. We use base 10 logarithms for convenience of interpretation. The raw values for the Chinese plurals example are plotted in the left panel of the following figure. Since most of the N_r are zero for large r, we need to account for the N_r which are zero. Following the work in Church and Gale [1991], we average with each non-zero N_r the zero N_r's that surround it: order the non-zero N_r by r, and let q, r, and t be successive indices of non-zero values. We replace N_r by $Z_r = N_r/0.5(t-q)$. In other words we estimate the expected N_r by the density of N_r for large r. For small r, there is no difference, because the length of the intervals is unity. For large r, the change can make a difference of several orders of magnitude.

The following two figures show the effect of this averaging transformation for the Chinese plurals example and the Prosody example. The left panels show the plot of $log(N_r)$ versus $log(r)$. Note that the minimum value for N_r of 1 forces the data to curve off to the right. The right panels show the plot of $log(Z_r)$ versus $log(r)$. Note that the downward trends in the low and medium values of r are maintained for large r.

Figure 1
Chinese Plurals need the averaging transform

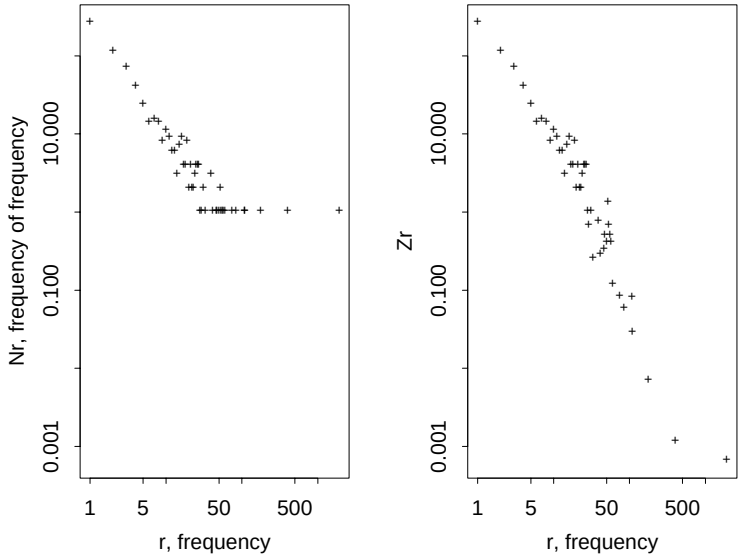

Figure 1. For both panels, the horizontal axis is frequency, r, plotted on a logarithmic scale. For the left panel, the vertical axis shows the raw N_r on a logarithmic scale. For the right panel, the vertical axis shows Z_r, which averages the many zero N_r with the few non-zero N_r. The data are from the Chinese plurals example. Note that the lower limit of 1 for N_r makes the left panel show a skew to the right. The averaging transform removes this lower limit and allows the downward trend evident at low and medium r to continue for large r.

A similar figure results for the Prosody example as shown in the following figure.

Figure 2
The Prosody data need the averaging transform

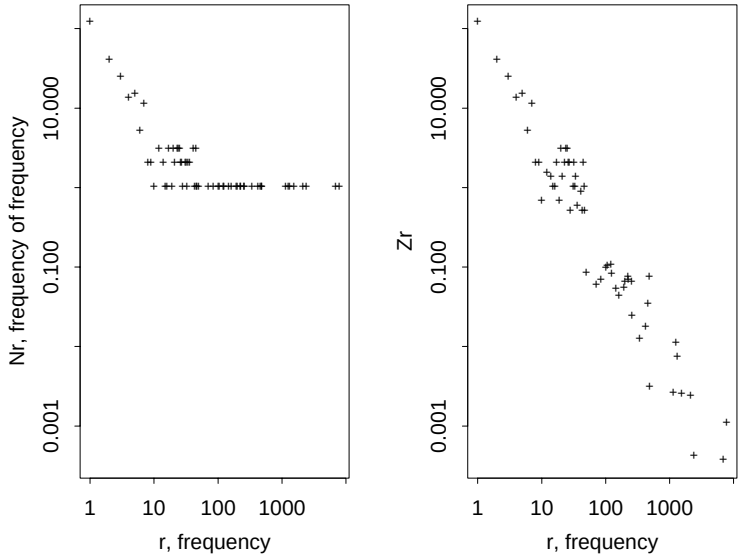

Figure 2. The scales in this figure are the same as in the previous figure, however the data are from the Prosody example. The averaging transform usually results in figure like these for linguistic data.

We are now ready to smooth the Z_r, and the need to do so is obvious.

Church and Gale [1991] used a smoother that would account for the well measured values at small r and also for the very poorly measured values at large r. However such smoothers are quite elaborate. Fortunately, we are not interested in the smoothed values of N_r, except to calculate values for r^*, so a simpler smoothing method may be useful. Recall that r^* is defined by $p_r = r^*/N$. We have two prior expectations about r^*, $r \geq 1$, that can guide us. First, we expect $r^* < r$, and second we expect r^*/r to increase to one as r increases. The first expectation comes because we are cutting the probability of the observed cases to make room for the probability we are assigning to the unseen cases. We will accomplish this by decreasing from the r, not increasing. Secondly, we know that the higher r is, the better it is measured, so we want to take less and less probability away as r increases. It is difficult to satisfy these expectations on r^* while making good smooths of the N_r. Thus the first proposal is to make an extremely simple smooth of the N_r which is certain to satisfy the priors on r^*. For the small values of r, simply using the Turing estimate may well be more accurate than the simple smooth. Thus, the second proposal is a rule for switching from the Turing estimates at low frequencies to the Good-Turing estimates for higher frequencies.

The simplest smooth is a line, and a downward sloping log-log line will satisfy the priors on r^* so long as the slope of the line b is less than -1. This is the proposed simple smooth, and we call the associated Good-Turing estimate the Linear Good Turing (LGT) estimate. To see that we need $b < -1$, note that if

$$log(N_r) = a + b\ log(r)$$

then

$$N_r = A\ r^b$$

and

$$\begin{aligned} r^* &= (r+1)\ \frac{N_{r+1}}{N_r} \\ &= (r+1)\frac{A(r+1)^b}{Ar^b} \\ &= r\ (1+\frac{1}{r})^{b+1} \end{aligned}$$

so $r^* < r$ if

$$b+1 < 0$$

or

$$b < -1$$

Thus if the estimate for the slope, *b*, is greater than -1, the method presented should be regarded as not applicable.

The line should be fit by simple linear regression of $\log(Z_r)$ on $\log(r)$. Simple linear regression is available in many statistical packages, and section 15.2 of (Press *et al.*, 1992) reviews its theory and present code. Many statistical textbooks have a longer review of the theory.

The rule for choosing between the Turing estimate and the LGT estimate is to use the Turing estimates so long as they are significantly different from the LGT estimates. Once we use an LGT estimate, then we continue to use them. Thus this is a rule that may have a few Turing estimates of r^* first, but then switches to the LGT estimates. Turing estimates are considered "significantly different" from LGT estimates if their difference exceeds 1.65 times the standard deviation (the square root of the variance) of the Turing estimate. The variance for the Turing estimate is approximately

$$Var(r_T^*) \approx (r+1)^2\ \frac{N_{r+1}}{N_r^2}\ (1\ +\ \frac{N_{r+1}}{N_r})$$

The approximations made to reach this are that N_r and N_{r+1} are independent, and that $Var(N_r) \approx N_r$. For once the independence assumption is reasonable, as might be gathered from how noisy N_r is. The variance approximation is good for binomial sampling of types with low probability, so it is consistent with Good-Turing methodology. Since we are combining two different estimates of probabilities, we do not expect them to add to one. In this condition, our estimates are called *unnormalized*. We make sure that the probability estimates add to one by dividing by the total of the unnormalized estimates. This is called *renormalization*.

$$p_r = (1-\frac{N_1}{N})\frac{p_r^{unnorm}}{\sum_{r=1} p_r^{unnorm}} \qquad r \geq 1$$

We will call this renormalized combination of Turing and LGT estimates the Simple Good-Turing (SGT) estimate. The following figure shows the SGT r^*/r for the prosody example and for the Chinese plurals examples. The Chinese plurals example needs more probability set aside for the unseen cases than does the prosody example (.04 versus .004), but it has twice as many types and five times the tokens to take this probability from than does the prosody example, so the r^* are not all that different. The plot of r^*/r versus r is a useful plot to examine when considering a Good-Turing estimate, however it is obtained.

Figure 3
The Simple Good-Turing Estimates for Two Examples

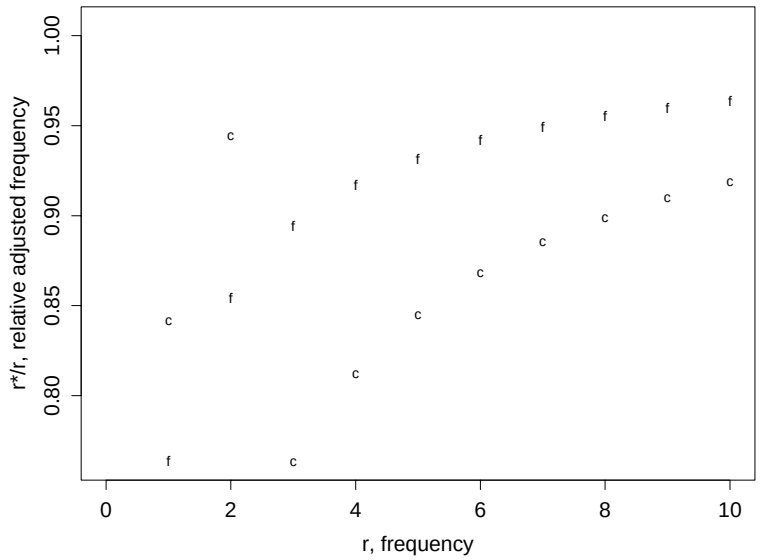

Figure 3. The horizontal axis shows the frequency, r. The vertical axis shows the results of the Simple Good-Turing estimator as r^*/r. The *c*'s mark values for the Chinese plurals examples, while the *f*'s mark values for the prosody example.

Figure 3 shows that the r^* estimated by the SGT method for the Chinese plurals example and for the prosody example satisfy our priors for an open class. The Chinese plurals example shows some hint that the class may be closed, because the Turing estimator was selected for the $r = 1$ and $r = 2$, and because these were greater than the corresponding LGT estimates. Still, since $1^* < 1$, we would probably do well to treat the class as open given only as much data as we have.

4. How Accurate is Simple Smoothing?

We do not know the answers (just what the average probability of types seen r times is) for either of the previous two examples, so we cannot say how accurate the estimates are. Here are two examples for which we do know the answers.

4.1 A Monte Carlo study

Monte Carlo studies are a well established technique for research in statistics, because they allow constructing examples for which the truth is known. A Monte Carlo study consists of defining a model

which will generate data of the sort that one wants to apply a method to, generating data by using the model and a (pseudo-)random number generator, applying the methods to be tested to the data, and comparing the results of the methods to the truth known from constructing the model. The utility of the results is related most closely to the validity of the model used to generate the artificial data.

For this study, we constructed twenty texts each having 100,000 tokens. The tokens drawn randomly from a vocabulary of size V types, with the probability of the i^{th} type having a probability proportional to i^z, for some z < -1. This is called a Zipfian distribution with exponent z. Specifically, the probability of the i^{th} type is

$$i^z / \sum_{i=1}^{i=V} i^z$$

Twenty examples were generated, using V = 5000, 10000, 25000, 50000, and 100000, and z = -1.1, -1.2, -1.3, -1.4. Note that while all the V are finite, since at most 15,000 types were observed, the examples cover both cases in which the finite vocabulary is important, and others in which the vocabulary could not be distinguished from an infinite vocabulary in a text of 100,000 tokens. The range of z is representative of values seen in linguistic data. The point of making a Monte Carlo simulation is that one knows the true answer. Here, the answer we are after is the average probability of the tokens which occurred exactly r times. Since we know the probability of each token by construction, it is easy to calculate these averages. The hardest cases are for small r, so accuracies and comparisons have been limited to $r \leq 10$. For these small r, the average probabilities have two to three significant figures.

Besides the SGT method described above, three other methods that one might consider using were applied to the same examples. The first method is the Expected Likelihood Estimate (ELE). Box and Tiao (1973, p. 34-36) an "uniformative prior" for a binomially distributed variable and show that after observing the variable, the *expected* probability can be found by adding 0.5 to its observation, and to the number of trials. The ELE is the expectation of the same likelihood which is maximized by the Maximum Likelihood Estimate. Thus there is some justification for adding 0.5 to each observation. By contrast to the MLE, the ELE at least provides a non zero estimate for unseen objects. Since we are adding 0.5 to each observation, the total number of observations needs to be augmented by $V/2$ in order to assure $\sum_{i=1}^{i=V} p_i = 1$. Since this method closely resembles the commonly used "add-one" method we do not consider add-one separately.

A variant of the ELE which is sometimes useful, although it lacks a statistical justification, is "add-tiny", which adds $1/V$ to each observation and adds one observation to the total to renormalize. Add-tiny is the second method we will compare the SGT with. The ELE and add-tiny are extremely simple to use, and may be useful for preliminary studies. However, we will see that they have poor accuracy on the Monte Carlo data sets.

A major alternative to Good-Turing methods is cross validation. The cross validation concept was applied to linguistic data, and termed "deleted estimation" by Jelinek and Mercer (1985), and explained further by Nádas (1985). Cross validation is a well established statistical technique. It makes a different assumption from the Good-Turing methods. It is a more difficult calculation than the SGT, but is usually no problem for modern computers. We consider the simplest two way cross validation.

The Good-Turing estimator is based on a theorem about the frequency one would expect in an additional sample, for objects which occur r times in an observed sample. An empirical realization of this idea is the *held-out* estimator defined by Jelinek and Mercer (1985). Let the available text be divided into two halves, called *retained* (0) and *held-out* (1). For each object, b, let $r_0(b)$ be its frequency in the retained (0) half of the text. The number of distinct objects with frequency r, (N_r), is $\sum_{b|r_0(b)=r} 1$. Then count all occurrences in the held-out (1) text of all the objects with frequency r in the retained (0) text, $C_r \equiv \sum_{b|r_0(b)=r} r_1(b)$, where the $r_1(b)$ is the observed frequency of the object, b, in the held-out (1) text. The adjusted frequency is

$r^* = C_r/N_r$. It is important to realize that in dividing the text in two parts, one estimates the probability $p_r(N/2)$ of an object seen r times in a sample of size $N/2$. In order to apply the estimate to a sample of size N, one makes the assumption that the probability of a token seen r times in a sample of size N is half the probability of a token seen r times in a sample of $N/2$, as would be true of Maximum Likelihood Estimates.

Two way cross validation makes a more efficient use of the data. We use the *deleted estimate* of Jelinek and Mercer (1985). Essentially, this is a way to combine held-out estimates made by interchanging the roles of held-out and retained halves of the text. Denoting the two halves of the data by 0 and 1, we have N_r^0 is the number of objects occurring r times in the half labeled 0, and C_r^{01} is the total number of occurrences in the half labeled 1 of those objects. Likewise, N_r^1 is the number of objects occurring r times in the half labeled 1, and C_r^{10} is the total number of occurrences in the half labeled 0 of those objects. The two held-out estimators would be C_r^{01}/N_r^0 and C_r^{10}/N_r^1. A simple way to combine these would be to take their mean. The deleted estimate is formed by combining the underlying measurements by

$$r^* = \frac{(C_r^{01} + C_r^{10})}{(N_r^0 + N_r^1)}$$

This estimator also assumes that the probability of a token seen r times in a sample of size N is half the probability of a token seen r times in a sample of $N/2$. This is a weaker assumption than that the objects have a binomial distribution, because it is implied by a binomial distribution.

To facilitate comparisons, each method is used with the knowledge of the vocabulary size, V, even though it would not ordinarily be known. This is a matter of some importance for ELE and add-tiny, because they would not have a means of estimating the total probability of unseen types without it. It is simply a matter of calculating the answers to which SGT and two-way cross validation are compared, however. Since the latter two turn out to be the most accurate, the extra knowledge plays no role in our conclusions.

A first crude look at the errors from the four methods is shown in the following table. The true probabilities and the estimates vary by several orders of magnitude, so an error is calculated as the logarithm of the ratio of the estimate to the true probability. The table shows the root mean square (RMS) log error over all frequencies $0 \le r \le 10$, vocabulary sizes, and Zipfian exponents for each of the four methods.

Table 5
Additive Methods have Bad Errors

method	ELE	add-tiny	SGT	two way cross validation
RMS log error	.47	2.62	0.062	0.18

The ELE method gets the order of magnitude correct on average, but the add-tiny method fails to achieve even that on the Monte Carlo data. The SGT gives the best overall RMS log error of the four methods considered.

Examining the errors by frequency shows where the various methods fail. The following figure shows that ELE and add-tiny methods methods fail worst in estimating the probability of unseen types ($r = 0$). The figure combines errors for a given r across all 20 data sets by the RMS.

Figure 4
Additive Methods Fail Badly for Small Frequencies

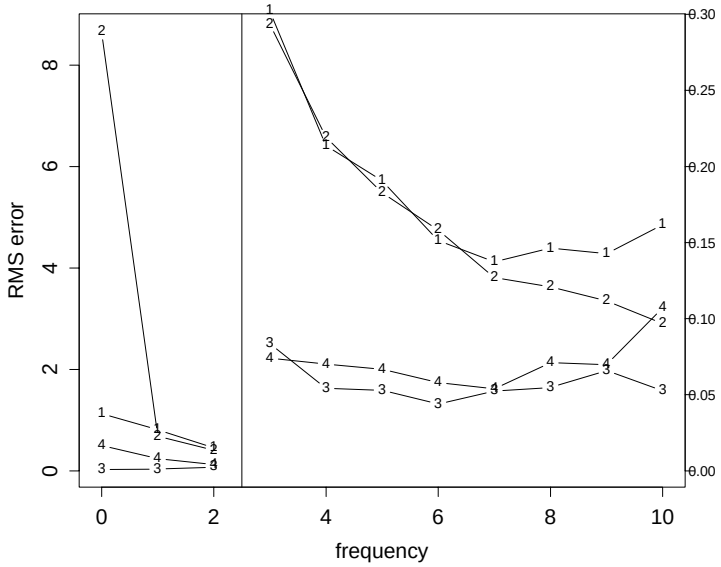

Figure 4. The horizontal scale shows frequencies. The RMS errors are plotted vertically. Note that the scale at the left applies for the low frequencies, the scale at the right for higher frequencies. The methods are indicated by the plotting symbols (1) ELE, (2) add-tiny, (3) Simple Good-Turing, and (4) two way cross validation. The add-tiny method is off by eight orders of magnitude for the unseen objects, while the ELE method is off by one order of magnitude for them.

Figure 4 shows that the additive methods are grossly wrong for unseen objects, and remain worse than SGT and two way cross validation over the range of frequencies shown.

The previous figure squashed the SGT estimate and the two way cross validation errors in order to encompass the errors from the other two methods. The following figure shows just the RMS errors from the SGT and two way cross validation.

Figure 5
Two Way Cross Validation is Poor for Small Frequencies

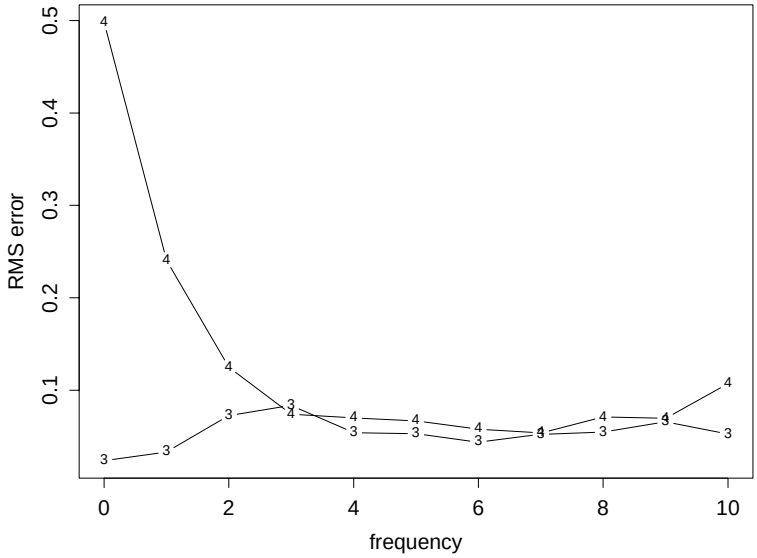

Figure 5. Frequency is plotted horizontally, RMS error vertically. The two lines shown are for (3) Simple Good Turing, and (4) two way cross validation. Two way cross validation is worse for frequencies of two or less, but thereafter the methods are comparable in accuracy.

The figure shows that for r greater than about two, SGT has errors comparable to two way cross validation. It also shows that two way cross validation is poor for $r \leq 2$. This does not show that cross validation is a poor method for small r, merely that *two way* cross validation is poor. Our chief concern here, of course, is with the SGT errors. The following figure shows just the RMS errors for the SGT.

Figure 6
Simple Good Turing Errors are Least for Unseen Objects

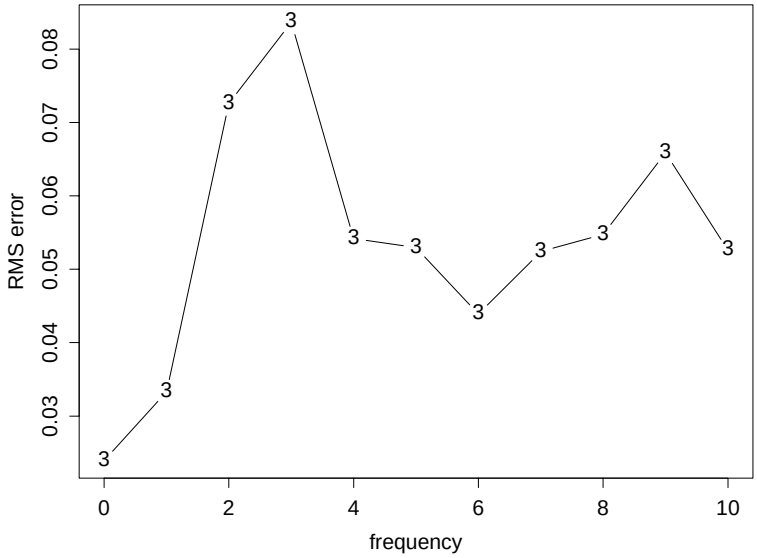

Figure 6. Frequency is plotted horizontally, RMS error for the Simple Good-Turing method vertically. The SGT method does best for unseen objects, because it relies on the Turing estimate for these.

This figure shows that the SGT does best for small *r* and settles down to about a 5% error for large *r*. There is an intermediate zone of a few frequencies in which it does less well. The reason for this behavior is that the SGT uses the Turing estimates for small *r*, and then switches to the LGT estimates. In the switching region both estimates have problems.

The design of the Monte Carlo data sets lets us examine the dependence of errors on vocabulary size and on Zipfian exponent. The following figure plots RMS errors for the SGT estimates by vocabulary size.

Figure 7
Simple Good-Turing Accuracy is Independent of Vocabulary Size

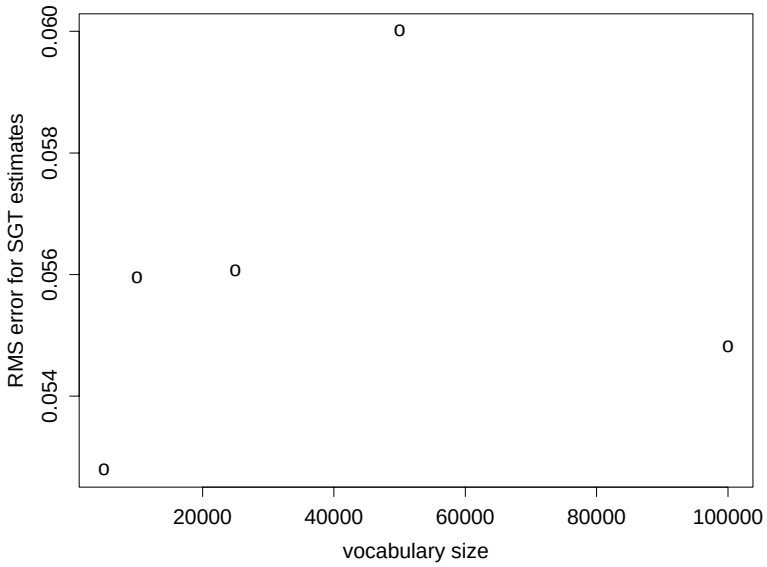

Figure 7. Vocabulary size is plotted horizontally, while the vertical axis shows RMS errors for the Simple Good-Turing estimates. There is no systematic dependence on vocabulary size, and the range of variation is small (note that the range of the vertical axis is a small part of the range in Figure 6.)

The figure shows little dependence on vocabulary size over the range from 5000 types to 100000 types for 100000 token texts. Furthermore, the range of RMS errors shown here is much less than that shown for dependence on frequency in the previous figure. The following figure plots RMS errors for the SGT estimates by Zipfian exponent.

Figure 8
Simple Good-Turing Accuracy is Independent of Zipfian Coefficient

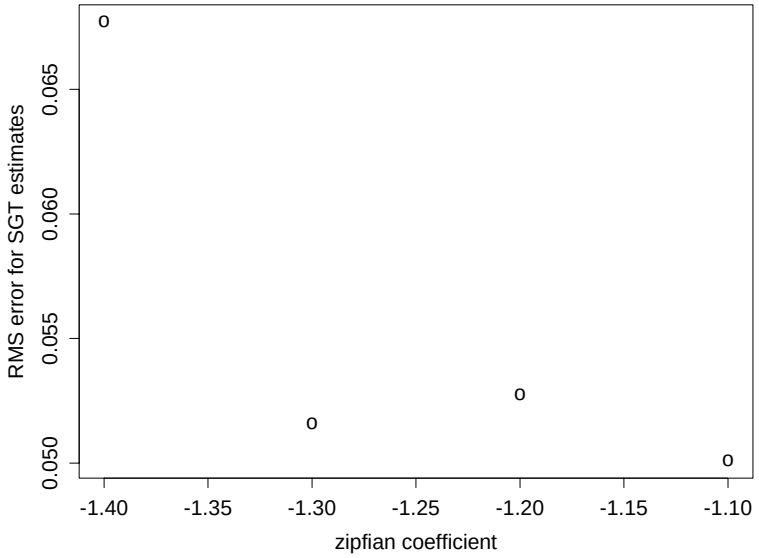

Figure 8. The Zipfian coefficient is plotted horizontally, while the vertical axis shows RMS errors for the Simple Good-Turing estimates. There is no systematic dependence on Zipfian coefficient, and the range of variation is small.

This figure does not show a systematic dependence of error on Zipfian coefficient, although there could be somewhat larger errors for larger magnitude coefficients. However, the range of errors is small, and the dependence on frequency remains the predominant effect.

The errors shown here are likely to be lower than errors for actual linguistic data because tokens do not have binomial distributions in actual data. Mosteller and Wallace (1964) showed that negative binomial distributions described their observations better than binomial distributions. Nevertheless, they made their study using both distributions, and drew the same conclusions either way. The difficulties of using the negative binomial distribution seem to have kept it from further study or use in computational linguistics.

4.2 Accuracy for bigrams

The second example is the bigram example. The data are taken from Tables 1 and 2 of Church and Gale [1991]. The bigrams counted were all the bigrams in 22 million words of AP newswire text. A second sample of equal size allowed us to use the held-out estimator as a "gold standard" against which to compare other methods. The following table shows r, N_r, the Turing estimators r^*_T, the SGT estimators r^*_{SGT}, and the held-out estimators r^*_{HO}.

Table 6
SGT Estimation is Accurate for Bigram Example

r	N_r	r^*_T	r^*_{SGT}	r^*_{HO}
1	2 018 046	0.446	0.446	0.448
2	449 721	1.26	1.26	1.25
3	188 933	2.24	2.24	2.24
4	105 668	3.24	3.24	3.23
5	68 379	4.22	4.22	4.21
6	48 190	5.19	5.19	5.23
7	35 709	6.21	6.21	6.21
8	37 710	7.24	7.24	7.21
9	22 280	8.25	8.25	8.26

Note that with the huge values for N_r in this example, the SGT has selected the Turing estimator for all of the values shown in the table, and that no error exceeds 1%. This may be an unrepresentative example, but the result of simply applying the SGT method is at least reassuring. (The LGT has a maximum error of 6%, for $r = 0$.)

5. On Estimating the Probabilities of Unseen Objects

Different estimates of probabilities for unseen objects are far preferable to the same probability for each object. To make such estimates requires relying on the structure of the objects, and thus depends on the application.

Church and Gale (1991) show one example. A bigram is a pair of words, and even if the pair has not been seen, a unigram model provides a probability for each word if one assumes independence. Then when a bigram $w_1 w_2$ has not been seen, $p_u(w_1) p_u(w_2)$ provides an estimate of the bigram's probability assuming the words are independent (without this assumption, the unigram probabilities could not be just multiplied). This estimate must be calibrated before being used, and not used directly, or the individual estimates will not add to the Turing estimate for the total probability of the unseen objects.

Another example can be described from the project with Richard Sproat and Chilin Shih on the segmentation of Chinese text into words. The recognition of Chinese names is quite difficult, because almost any character can be used in a given name. A large list of names provided counts for single character given names (and similar analyses were done separately for the first and second character of two character given names). These characters would have clear semantic connotations to literate Chinese. Since it would be unusual for parents to name a child by a character with a connotation of death, one might suppose that characters with meanings of death would be less likely as a name than characters with meanings of plants, which include the flowers. The internal structure (radicals) of characters allows a crude division of this sort into a few hundred classes. Assuming the probability of unseen objects *within each class* to be the same, they were estimated as

$$p_0^{cls} \propto \frac{S(N_1^{cls})}{N \; N_0^{cls}}$$

where N_0^{cls} is the number of unobserved members of the class *cls*, and N_1^{cls} is the number of members observed once. (All members of each class are known *a priori*.) $S(N_1^{cls})$ is a smoothing of N_1^{cls}, where the smoothing was done with respect to the number of objects in the class. This can be recognized as a Good-Turing estimator of p_0 by class. The renormalization required to make the total probability equal the Turing estimate from all the data was about 10%.

The categories with lowest probabilities for unseen characters were named as follows (upper case names have semantic connotations, lower case names do not).

TURTLE
sprout
neg
NOSE
SKIN
BIG_TURTLE
DRUM
FACE
LEEK
march
STICK
HOG
TRIPOD
RAT
DEATH

Chinese speakers judged that categories low on the list were unlikely to be used for names.

6. Summary

This paper has presented a Good-Turing method for estimating the probabilities of seen and unseen objects in linguistic applications. The method uses the simplest possible smooth of the frequency of frequencies, N_r, a straight line, together with a rule for switching from Turing estimates which are more accurate at low frequencies. It has been called the Simple Good-Turing (SGT) method.

The SGT method is more complex than the Expected Likelihood Estimate (add 0.5 to each observation), but simpler than two way cross validation. On a set of twenty Monte Carlo examples spanning vocabulary sizes from 5,000 to 100,000 and Zipfian coefficients from -1.4 to -1.1, the SGT was far more accurate than ELE. It was more accurate than two way cross validation for frequencies of two or less, and of approximately the same accuracy for higher frequencies.

The importance of using the internal structure of objects to get differing estimates for the probabilities of unseen objects was discussed, and two examples showed possible approaches.

The major assumption of Good-Turing methods is that the objects of interest have binomial distributions. The binomial distribution also implies the validity of the assumption required for cross-validation methods. The accuracy tests made here are on artificial data for which the objects do have binomial distributions. The usefulness of these various methods when the objects do not have a binomial distribution needs to be examined.

The complexities of smoothing may have hindered the use of Good-Turing methods in computational linguistics. It is hoped that the SGT is simple enough and of sufficient accuracy to remedy this.

References

1. Becker, R., J. Chambers, and A. Wilks, (1988), *The New S Language*, Wadsworth & Brooks/Cole, Pacific Grove, California.

2. Box, G. and G. Tiao, (1973), *Bayesian Inference in Statistical Analysis*, Addison-Wesley, Reading, Mass.

3. Church, K. and W. Gale, (1991), ‘‘A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams,’’ *Computer Speech and Language*, v. 5, pp. 19-54.

4. Church, K., W. Gale, J. Kruskal, (1991), Appendix I of (Church and Gale, 1991).R

5. Good, I., (1953), ‘‘The population frequencies of species and the estimation of population parameters,’’ *Biometrika*, v. 40, pp. 237-264.

6. Jelinek, F. and R. Mercer, (1985), ‘‘Probability distribution estimation from sparse data,’’ *IBM Technical Disclosure Bulletin*, v. 28, pp. 2591-2594.

7. Nádas, A. (1985), ‘‘On Turing’s formula for word probabilities,’’ *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. ASSP-33, pp. 1414-1416.

8. Press, W., S. Teukolsky, W. Vetterling, and B. Flannery, 1992, *Numerical Recipes in C*, Cambridge University Press, Cambridge.

Appendix I

This appendix contains the complete set of (r,N_r) pairs for the prosody and Chinese plurals examples.

Prosody

r	N_r
1	120
2	40
3	24
4	13
5	15
6	5
7	11
8	2
9	2
10	1
12	3
14	2
15	1
16	1
17	3
19	1
20	3
21	2
23	3
24	3
25	3
26	2
27	2
28	1
31	2
32	2
33	1
34	2
36	2
41	3
43	1
45	3
46	1
47	1
50	1
71	1
84	1
101	1
105	1
121	1
124	1
146	1
162	1
193	1

199	1
224	1
226	1
254	1
257	1
339	1
421	1
456	1
481	1
483	1
1140	1
1256	1
1322	1
1530	1
2131	1
2395	1
6925	1
7846	1

Chinese Plurals

r	N_r
1	268
2	112
3	70
4	41
5	24
6	14
7	15
8	14
9	8
10	11
11	9
12	6
13	6
14	3
15	7
16	9
17	4
18	4
19	8
20	2
21	4
22	2
23	2
24	3
25	4
26	4
27	4
28	1
29	1

31	2
33	1
39	3
41	1
46	1
47	1
50	1
52	2
53	1
55	1
57	1
60	1
74	1
84	1
108	1
109	1
177	1
400	1
1918	1

Appendix II

This appendix gives code which implements the Simple Good-Turing analysis using S (Becker, Chambers, and Wilks, 1988) to perform the calculations. It consists of a short shell script which is labeled gt.s, a few S functions labeled gtfuncs.S, and an S script labeled gtanal.S.

gt.s

```
#Perform a Simple Good-Turing analysis, using S.
#
#Input is a file with two items per line, frequency, r, and
#non-zero frequencies of frequencies Nr.
#
#Output to stdout has a first line containing the total number of objects, N,
#and the total number of types seen.
#The second line contains a zero and the TOTAL probability of unseen objects.
#The following lines contain the frequency r, and the SGT r*.
#
#To get the probability for an object seen r>=1 times, divide r* by N.
#
#The script destroys files named "freqhist" and "gtanal"
#and overwrites a number of S data sets, leaving the analysis to
#be examined in S.
#
#Usage: gt.s inputfile
#
cp $1 freqhist
S <gtanal.S
cat gtanal
rm freqhist gtanal
```

gtfuncs.S

```
nrzest<-function(r, nr)
{
      d <- c(1, diff(r))
      dr <- c(0.5 * (d[-1] + d[ - length(d)]), d[length(d)])
      return(nr/dr)
}
rstest<-function(r, coef)
{
      return(r * (1 + 1/r)^(1 + coef[2]))
}
```

gtanal.S

```
#read in data
xm<-matrix(scan("freqhist",0),ncol=2,byrow=T)
xr<-xm[,1]
xnr<-xm[,2]
```

```
xN<-sum(xr*xnr)

#make averaging transform
xnrz<-nrzest(xr,xnr)

#get Linear Good-Turing estimate
xf<-lsfit(log(xr),log(xnrz))
xcoef<-xf$coef
xrst<-rstest(xr,xcoef)
xrstrel<-xrst/xr

#get Turing estimate
xrtry<-xr==c(xr[-1]-1,0)
xrstarel<-rep(0,length(xr))
xrstarel[xrtry]<-(xr[xrtry]+1)/xr[xrtry]*c(xnr[-1],0)[xrtry]/xnr[xrtry]

#make switch from Turing to LGT estimates
tursd<-rep(1,length(xr))
for(i in 1:length(xr))if(xrtry[i])
      tursd[i]<-(i+1)/xnr[i]*sqrt(xnr[i+1]*(1+xnr[i+1]/xnr[i]))
xrstcmbrel<-rep(0,length(xr))
useturing<-TRUE
for(r in 1:length(xr)){
      if(!useturing) xrstcmbrel[r]<-xrstrel[r]
      else if(abs(xrstrel-xrstarel)[r]*r/tursd[r] > 1.65)
            xrstcmbrel[r]<-xrstarel[r]
      else {useturing<-FALSE; xrstcmbrel[r]<-xrstrel[r]}
      }

#renormalize the probabilities for observed objects
sumpraw<-sum(xrstcmbrel*xr*xnr/xN)
xrstcmbrel<-xrstcmbrel*(1-xnr[1]/xN)/sumpraw

#output
cat(xN,sum(xnr),"0,file="gtanal")
cat(0,xnr[1]/xN,"0,file="gtanal",append=TRUE)
for(i in 1:length(xr))cat(xr[i],xr[i]*xrstcmbrel[i],"0,
  file="gtanal",append=TRUE)
```