STATISTICAL LANGUAGE MODELING USING THE CMU-CAMBRIDGE TOOLKIT

Philip Clarkson – prc14@eng.cam.ac.uk
Cambridge University Engineering Department,
Trumpington Street, Cambridge, CB2 1PZ, UK.

Ronald Rosenfeld – roni@cmu.edu
School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA.

ABSTRACT

The CMU Statistical Language Modeling toolkit was released in 1994 in order to facilitate the construction and testing of bigram and trigram language models. It is currently in use in over 40 academic, government and industrial laboratories in over 12 countries. This paper presents a new version of the toolkit. We outline the conventional language modeling technology, as implemented in the toolkit, and describe the extra efficiency and functionality that the new toolkit provides as compared to previous software for this task. Finally, we give an example of the use of the toolkit in constructing and testing a simple language model.

1. INTRODUCTION

While language modeling continues to be a fruitful research topic, the standard language modeling techniques which are used in most large vocabulary recognition systems have changed little over the past few years [7]. Even when improvements have been made over the traditional models, they have normally come about by combining a new model with a conventional trigram [2] model [1, 10, 11, 12, 3].

The CMU Statistical Language Modeling (CMU-SLM) toolkit [16] is a set of Unix software tools facilitating the construction and testing of conventional bigram and trigram language models. Version 1 was released in 1994, and is currently in use in over 40 academic, government and industrial laboratories in 12 countries.

Since the release of version 1, however, the ground has shifted a little in language modeling. Larger corpora are available than before, and more powerful computers are available to process them. Interest has grown in moving beyond trigram language models towards 4-gram and 5-gram models. Furthermore, some of version 1's inefficiencies which were tolerable when dealing with small corpora became a real problem when dealing with several hundreds of millions of words. Version 2 of the toolkit has been developed in order to address these shortcomings.

This paper will give a brief outline of conventional language modeling technology as implemented in version 1 of the toolkit. It will then describe some of the theoretical and implementational improvements which have been made in version 2, and provide an example of the use of the toolkit.

2. CONVENTIONAL LANGUAGE MODELING THEORY

2.1. Smoothing

Conventional language models are based on the bigram or trigram model. Because even the largest corpora available will contain only a fraction of the possible trigrams it is necessary to *smooth* the data in order to provide better estimates of the more infrequent or unseen events[1].

The maximum likelihood estimate for the probability of an event E which occurred r times out of a possible R is $P(E) = r/R$. In a sparse sample, however, the maximum likelihood estimate is biased high for observed events and biased low for unobserved ones. To correct this bias, we redistribute some probability mass from the observed events to the unseen ones, by *discounting* the counts by a *discount coefficient* d_r. That is, the modified count r^* is

$$r^* = rd_r \tag{1}$$

and the revised probability estimate is $P(E) = r^*/R$. The remaining probability mass is assigned to unseen events.

If a given N-gram $(w_1, w_2, \ldots, w_N)$ has not been observed in the training data, $P(w_N \mid w_1, w_2, \ldots, w_{N-1})$ can be estimated from the lower order model, namely from $P(w_N \mid w_2, \ldots, w_{N-1})$, in a process known as *backing-off* [8].

Frequently, backing-off and discounting are combined according to the scheme devised by Katz [8], which combines Good-Turing discounting [5] with backing-off. This is the approach to data smoothing which was implemented in version 1 of the toolkit.

2.2. Cutoffs

In order to reduce the size of a language model, infrequent N-grams are often removed from the model. The counts below which the N-grams are discarded are referred to as *cutoffs*. Table 1 shows how the bigram and trigram cutoffs affect the size and perplexity [2] of a trigram language model.

2.3. Context Cues

Language data is viewed by the toolkit as a stream of words interspersed with *context cues*. These are markers which indicate events such as sentence, paragraph and article boundaries. They provide useful information to the language model (and therefore may affect prediction), but they should not be predicted by the model itself.

2.4. Vocabulary Types

The toolkit supports three types of vocabulary, which each handle out-of-vocabulary (OOV) words in different ways.

A *closed vocabulary* model does not make any provision for OOVs. Any such words which appear in either the training or test data will cause an error. This type of model might be used in a command/control environment

[1] In the context of a trigram language model, for example, an "event" refers to an occurrance of a unigram, bigram or trigram.

Cutoffs		Size of	
Bigram	Trigram	Model	Perplexity
0	0	219 MB	114.0
1	1	80 MB	119.8
2	2	49 MB	124.0
5	5	26 MB	133.1
10	10	17 MB	143.4
20	20	11 MB	156.8
50	50	8 MB	180.7
100	100	6 MB	203.9

Table 1. The effect of cutoffs on the size and perplexity of a trigram language model. Results generated using the training and test sets from the 1996 H4 Broadcast News evaluation.

where the vocabulary is restricted to the number of commands that the system understands, and we can therefore guarantee that no OOVs will occur in the training or test data.

An *open vocabulary* model allows for OOVs to occur; out of vocabulary words are all mapped to the same symbol. Two types of open vocabulary model are implemented in the toolkit. The first type treats the OOV symbol the same way as any other word in the vocabulary. The second type of open vocabulary model is to cover situations where no OOVs occurred in the training data, but we wish to allow for the situation where they could occur in the test data. This situation could occur, for example, if we have a limited amount of training data, and are able to choose a vocabulary which provides 100% coverage of the training set. In this case, an arbitrary proportion of the probability mass is reserved for OOV words.

3. ENHANCEMENTS IN VERSION 2

3.1. Multiple Discounting Methods

There are many ways in which one can define the discount coefficient d_r (equation (1)).

3.1.1. Good-Turing discounting

If we define n_r as the number of events which occur r times, then the Good-Turing-based discounting scheme which was implemented in version 1 defines

$$d_r = \frac{\frac{(r+1)n_{r+1}}{rn_r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} \quad (2)$$

for $r < k$ (where typically $k \approx 7$) and $d_r = 1$ for higher counts [8], the belief being that events occurring more than 7 times are well estimated by maximum likelihood.

This approach does have some disadvantages, however. For example, we require $d_r > 0$ for all r, and this puts some constraints on the relative values of $n_1, n_2, \ldots, n_{k+1}$. In general, these constraints will be satisfied by naturally occurring data but may not be if one has doctored the data in some way (for example by boosting the counts of some subset of the N-grams).

Good-Turing discounting remains the default in version 2, but other discounting schemes have been implemented which do not suffer from this problem, and which sometimes produce superior results. These are presented below.

3.1.2. Linear discounting

In linear discounting [13] a quantity proportional to each count is subtracted from the count itself. That is, we set $d_r = 1 - \alpha$. In this case we select α such that linear discounting assigns the same probability to unseen events as Good-Turing discounting:

$$d_r = 1 - \frac{n_1}{R} \quad (3)$$

where R is the number of words in the training data.

3.1.3. Absolute discounting

Absolute discounting involves subtracting a constant b from each of the counts.

$$d_r = \frac{r-b}{r} \quad (4)$$

It can be shown [13] that setting $b = \frac{n_1}{n_1+2n_2}$ is approximately optimal in terms of maximising the log likelihood function using leaving-one-out.

3.1.4. Witten-Bell discounting

The discounting scheme which we refer to as Witten-Bell discounting is that which is referred to as type C in [17], and which was first applied to language modeling by [14]. The discounting ratio is not dependent on the event's count, but on t, the number of distinct events which followed the particular context. So, for the bigram `"A B"`, t is the number of distinct bigrams of the form `"A *"` in the model.

$$d_r(t) = \frac{R}{R+t} \quad (5)$$

3.2. N-grams of Arbitrary Size

The new version of the toolkit is no longer limited to the construction and testing of bigram and trigram models, and supports N-gram models for any value of N.

3.3. Flexible Handling of Context Cues

In version 1 the symbols `<s>`, `<p>` and `<art>` were fixed as context cues. Version 2 adopts the more flexible approach of allowing the user to specify any subset of the vocabulary to be context cues.

3.4. Efficient Memory Usage

The data structures used to store the N-grams in version 2 are more compact than those of version 1, with the result that language model construction is a less memory intensive task. For example, for a trigram language model, version 1 required 12 bytes per bigram and 4 bytes per trigram. Version 2 requires only 8 bytes per bigram and 4 bytes per trigram.

3.5. Interactive Language Model Evaluation

The tool used to evaluate the language models can now be run interactively. The language model is read in, and commands are read from the standard input. The language model can therefore be tested in a much more efficient fashion than in version 1, when the language model had to be read in each time a new test was to be performed on it.

3.6. Testing of ARPA Format Language Models

In version 1 it was only possible to perform tests on language models which had been written in the toolkit's own binary format. Version 2 allows the user to also evaluate the performance of language models that are in the standard ARPA format, that may have been supplied by a third-party, and need not have been generated by the toolkit.

3.7. Forced Back-off

The tool used for evaluating language models allows the user to specify a set of *forced back-off* parameters. There may be items in the vocabulary (especially context cues and the "unknown" symbol) from which we may want to back-off all the time. For example, if we see the word string `"A <s> B"` (where `<s>` is a context cue indicating a sentence boundary), then instead of predicting the probability of `B` based on the full context (P(`B` | `A <s>`)), we may wish to disregard the information before the sentence boundary. Therefore we might want to back-off to the bigram distribution P(`B` | `<s>`) (*inclusive* forced back-off) or even to the unigram distribution P(`B`) (*exclusive* forced back-off). Version 2 supports both types of forced back-off from arbitrary vocabulary items.

3.8. Linear Interpolation of Models

In order to facilitate the combination of language models, the toolkit also includes a program [15, Appendix B] for calculating maximum likelihood weights for a set of models by use of the expectation maximisation (EM) algorithm [4]. The models are described by their output on a common set of items, and it is these probability streams which the program receives as input. These may have been generated via models created by the toolkit, or by other software. For example, one may have generated cache-based probabilities [10], and these could be interpolated with probabilities from a trigram model created by the toolkit.

3.9. More Efficient Pre-processing Tools

Many of the tools in version 1 were simple filters which would convert a data stream into a slightly different format. The tools often relied heavily on Unix tools such as `sort` and `awk`, with the result that a great deal of disk I/O was incurred when very large files were processed.

Version 2 sacrifices a certain amount of the simplicity and modularity of its predecessor in favour of a more efficient approach. The tools now work much better if they can grab a large amount of RAM, and do the sorting there, reading from and writing to disk as little as possible.

4. EXAMPLE OF USAGE

This section will provide an example of the use of the toolkit in defining a language model's vocabulary, generating a language model from a large corpora of training text, and finally evaluating this language model's performance based on some held-out test text.

4.1. Creating a Vocabulary

A preliminary stage in constructing a language model is to define the model's vocabulary.

The tool `text2wfreq` outputs the number of occurrences of each word in the input text, and `wfreq2vocab` turns this list into a vocabulary file, in this case containing the most common 20,000 words.

```
cat training.text | text2wfreq | \
  wfreq2vocab -top 20000 > training.vocab
```

A context cues file should also be generated:

```
echo "<s>" > training.ccs
```

4.2. Constructing a Language Model

The first step is to turn the training text into a list of id N-grams (N-grams with each word mapped to an integer id, which will be zero for OOVs).

```
cat training.text | text2idngram -n 4 \
  -vocab training.vocab -buffer 100 \
  -temp /usr/tmp/ > training.id4gram
```

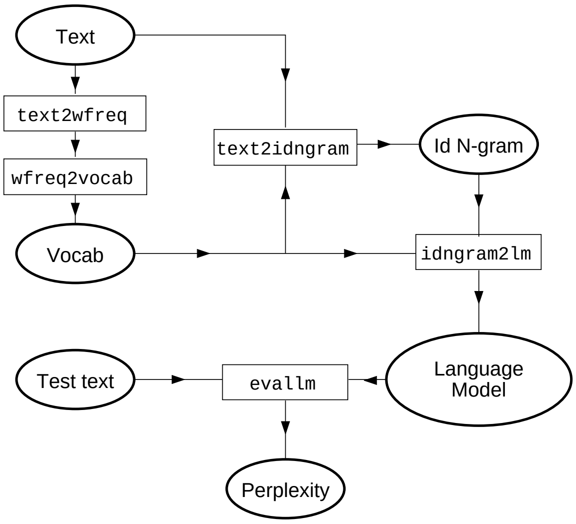

Figure 1. Use of the toolkit

The `-n 4` option indicates that we are building a 4-gram model, the `-buffer` option tells the program how much RAM to grab (in MB), and the `-temp` option allows us to specify where to store the temporary files.

The second step is to convert the id N-gram stream into a binary language model file.

```
idngram2lm -idngram training.id4gram \
  -vocab training.vocab -n 4 \
  -binary training.4gram.binlm \
  -cutoffs 1 1 5 -witten_bell \
  -context training.ccs
```

Here we have specified Witten-Bell discounting, and cutoffs of 1 for bigrams and trigrams, and 5 for 4-grams. At this stage we can also specify the vocabulary type, and various other parameters. We could also write out the language model in the standard ARPA format.

4.3. Evaluating the Language Model

The `evallm` tool is used for interactive evaluation of the language model.

```
evallm -binary training.4gram.binlm
evallm : perplexity -text test.text
Computing perplexity of the language model
  with respect to the text test.text
Perplexity = 176.99, Entropy = 7.47 bits
Computation based on 9596233 words.
Number of 4-grams hit = 2413343
Number of 3-grams hit = 3866964
Number of 2-grams hit = 2674322
Number of 1-grams hit = 641604
464893 OOVs (4.62%) and 576763 context cues
  were removed from the calculation.
```

5. FUTURE DEVELOPMENT

Future development of the toolkit may include the following:

1. Support for class-based N-grams [6]. Currently classes can be supported implicitly by mapping the training set into a given set of classes $\{C_i\}$, and building a standard model in the class vocabulary. This results in a model of the form $P(C_N \mid C_1, \ldots, C_{N-1})$, which can be combined with some estimate of $P(w_i \mid C_i)$. But other variants of class

N-grams, such as $P(w_N \mid C_1, \ldots, C_{N-1})$, are not currently supported. The most general model will allow each word position to have its own distinct class vocabulary.

2. Porting to run under Microsoft Windows 95, to make the toolkit accessible to a larger community of users.
3. In [9] a smoothing strategy is described in which backing-off distributions which have been optimized for the back-off model are used to give an improvement over conventional methods. This technique will be implemented in a future version of the toolkit.
4. More flexible backing-off. The idea behind backing-off is merely that if one language model is unable to provide a reliable estimate for an event's probability, then we should use a more general language model. This need not take the form of merely backing-off from an N-gram to an $(N-1)$-gram. One might instead want to back-off from a specific task-dependent language model to a language model trained on more general text, or from a word-based N-gram model to a class-based N-gram model.

6. CONCLUSION

This paper has presented a new toolkit for the construction and testing of statistical language models. The toolkit represents a significant improvement over existing publicly available software in terms of functionality and computational efficiency. The paper has also described many techniques which are standard in current language modeling theory, and which are implemented within the toolkit.

More details about the toolkit, including details of how to download the latest version can be found at

`http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html`

ACKNOWLEDGEMENTS

Philip Clarkson is supported by an EPSRC advanced studentship.

REFERENCES

[1] L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer. A Tree-Based Statistical Language Model for Natural Language Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7), 1989.

[2] L.R. Bahl, F. Jelinek, and R.L. Mercer. A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2), 1983.

[3] P.R. Clarkson and A.J. Robinson. Language Model Adaptation using Mixtures and and Exponentially Decaying Cache. In *Proceedings IEEE ICASSP*, 1997.

[4] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum Likelihood from Incomplete Data Using the EM Algorithm. *Journal of the Royal Society of Statistics*, 39(1):1–38, 1977.

[5] I.J. Good. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, 40, 1953.

[6] F. Jelinek. Self-Organized Language Models for Speech Recognition. In A. Waibel and K.-F Lee, editors, *Readings in Speech Recognition*, pages 450–506. Morgan Kaufman Publishers, 1990.

[7] F. Jelinek. Up From Trigrams! The Struggle for Improved Language Models. In *Proceedings Eurospeech*, 1991.

[8] S.M. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, 1987.

[9] R. Kneser and H. Ney. Improved Backing-Off for M-gram Language Modeling. In *Proceedings IEEE ICASSP*, 1995.

[10] R. Kuhn and R. De Mori. A Cache-Based Natural Language Model for Speech Reproduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

[11] R. Kuhn and R. De Mori. Corrections to 'A Cache-Based Natural Language Model for Speech Reproduction'. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:691–692, 1992.

[12] R. Lau, R. Rosenfeld, and S. Roukos. Trigger-Based Language Models: A Maximum Entropy Approach. In *Proceedings IEEE ICASSP*, 1993.

[13] H. Ney, U. Essen, and R. Kneser. On Structuring Probabilistic Dependencies in Stochastic Language Modelling. *Computer Speech and Language*, 8(1):1–38, 1994.

[14] P. Placeway, R. Schwartz, P. Fung, and L. Nguyen. The Estimation of Powerful Language Models from Small and Large Corpora. In *Proceedings IEEE ICASSP*, 1993.

[15] R. Rosenfeld. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. PhD thesis, School of Computer Science, Carnegie Mellon University, April 1994. Published as *Technical Report CMU-CS-94-138*.

[16] R. Rosenfeld. The CMU Statistical Language Modeling Toolkit, and its use in the 1994 ARPA CSR Evaluation. In *ARPA Spoken Language Technology Workshop, Austin, TX*, January 1995.

[17] I.T. Witten and T.C. Bell. The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, July 1991.