A Language Modelling Tool for Statistical NLP

Daniel Bastos Pereira, Ivandré Paraboni

Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (EACH / USP)
Av.Arlindo Bettio, 1000 - 03828-000, São Paulo, Brazil.
{daniel.bastos,ivandre}@usp.br

Abstract. *In recent years the use of statistical language models (SLMs) has become widespread in most NLP fields. In this work we introduce jNina, a basic language modelling tool to aid the development of Machine Translation systems and many other text-generating applications. The tool allows for the quick comparison of multiple text outputs (e.g., alternative translations of a single source) based on a given SLM, and enables the user to build and evaluate her own SLMs from any corpora provided.*

1. Introduction

Consider a Portuguese native speaker who wants to say (in English) that she is thinking of buying a new car, but who is not sure about the verb choice. In that case, a reasonable (and helplessly wrong) guess would be to mimic the equivalent Portuguese structure "*Eu estou querendo comprar um carro novo*" to produce (a) instead of the correct form (b):

a. I am ~~wanting~~ to buy a new car.
b. I am thinking of buying a new car / I am considering buying a new car.

What actually makes (b) more appropriate than (a) is immaterial to the present discussion, but there is one thing that we can safely assume to be true: very frequently we hear "I am" being followed by verb forms such as "thinking" or "considering", but very seldom we hear "I am" being followed by "wanting", even though "wanting" is a proper verb form in its own right.

The simple idea that some word sequences are more *frequent* than others has lead to the concept of language modelling that is now part of mainstream NLP research. In particular, *Statistical Language Models* (SLMs) based on *n-grams* are now widely used in most NLP fields, not to mention the large amount of work on Statistical MT in which they play a central role (e.g., Brown et. al, 1990; 1993). Besides their power to represent some crucial aspects of language, SLMs are quick and easy to implement, and do not require the labour-intensive development of NLP resources such as dictionaries or grammars, making them ideal for research in relatively resource-poor languages such as Portuguese.

The uses of SLMs stretch well beyond the design of components of NLP applications. For example, Machine Translation (MT) systems and many other text-generating applications often make use of *overgeneration* techniques to produce a large number of versions of the output text, from which inappropriate (e.g., ill-formed or ungrammatical) alternatives are filtered out to arrive at the desired output. Thus, in order to decide how to translate e.g., the NP "Mao's red book" to Portuguese given a

standard bilingual dictionary, we may simply generate all possible permutations of the individual NP constituents and compare them to select the one that closest 'resembles' the target language, as in (f) below:

c. o de Mao vermelho livro
d. o vermelho de Mao livro
e. o livro de Mao vermelho
f. o livro vermelho de Mao

There are of course many ways in which we may decide that (f) is 'better'. One such way is by asking *human evaluators* to do so within a certain level of agreement. This may however be a costly and time-consuming task, and it may become unfeasible at all if the comparison needs to be performed frequently (e.g., as in the early stages of research), or if it involves specialised skills (e.g., native speakers of an unfamiliar language). Under these circumstances, comparing computer-generated text *without* human intervention, e.g., using instead some form of language representation, becomes a far more attractive strategy. And representing language in this way is precisely what SLMs are good for.

At the application development level, comparing different versions of a computer-generated text may also help to decide which, among several competing approaches, seems more promising. For example, assuming that (c)-(f) above were produced by different systems (or variations of a single system under investigation), early empirical data suggesting that (f) is best can help guiding the research efforts towards that particular approach, and that should ideally occur long before an entire system has been developed and evaluated. Once again, the use of SLMs may have a great impact on how quickly these decisions can be made.

Building a language model is a fairly straightforward task when using, for example, language modelling toolkits such as CMU-Cambridge (Clarkson & Rosenfeld, 1994) or SRILM (Stolcke, 2002). We are however interested in encapsulating basic SLM techniques in a high-level tool to aid the development and evaluation of NLP applications in general while keeping a special regard for MT systems, and we would like this to be presented in a more user-friendly fashion.

In this paper we describe the ongoing implementation of a basic language modelling tool of this kind called *jNina*. Using n-gram statistics as a basis, jNina is intended to automatically compare text generated by multiple sources, allowing us to quickly discard those alternatives that prove to be less valuable. Moreover, our tool enables us to build new SLMs from scratch from any given corpora and evaluate them at a glance, which is essential not only for our current interest in the development of Statistical MT systems, but possibly to the wider research community working on many other NLP fields.

The rest of this paper is structured as follows. Section 2 provides a brief introduction to the language modelling concepts that we have adopted. Section 3 describes our implemented work and Section 4 presents some preliminary evaluation results. Section 5 summarizes our efforts so far and hints at the developments to follow.

2. Background

An SLM can be viewed as a probability distribution $P(s)$ over a set of possible sentences s representing how often each individual sentence occurs in the modelled language. The most widely-used SLMs are based on *n-grams*, in which the probability of a given word w_i in a sentence is determined by the n-1 previous words (its so-called *recent history* (Charniak, 1993)). In its simplest form, an SLM may compute probabilities based on a *maximum likelihood* (ML) estimate with some allowance for unseen instances. For example, in a trigram model, the maximum likelihood conditional probability P_{ML} of a word w_i given its two predecessors w_{i-2} and w_{i-1} is defined as the counting (c) of occurrences of the trigram divided by the number of occurrences of its bigram constituent:

$$P_{ML}(w_i \mid w_{i-2}\, w_{i-1}) = \frac{c(w_{i-2}\, w_{i-1}\, w_i)}{c(w_{i-2}\, w_{i-1})}$$

This estimator is of course useless if a given instance has not been observed in the training data. In order to reserve some probability mass for unseen events (and also to improve the model accuracy), a wide variety of *smoothing techniques* have been proposed. Possibly the oldest of all, the *Additive smoothing* proposed by Lidstone in 1920 based on Laplace's Law (Charniak, 1993; Chen & Goodman, 1999) simply adds one unit to the observed counts, which effectively deals with the problem of data sparseness but generally presents a poor performance for allowing too much of the probability space to unseen events (Manning & Schültze, 2003).

To improve the model accuracy is necessary to take into account additional information about the data distribution. For example, the *Good-Turing estimator* (GT) (Gale & Sampson, 1995) assumes the probability mass of unseen events to be n_1 / N, where N is the total of instances observed in the training data, and n_1 is the number of instances observed only once. The frequencies of the least frequent n-grams are adjusted accordingly, usually up to a specified discount range limit beyond which the adjustment is no longer significant[1], and then renormalized to yield a probability distribution. In the GT estimate every n-gram that occurs r times in the training data will be assumed to occur r^* times, which is defined as follows:

$$r^* = (r + 1) * (n_r + 1) / n_r$$

Despite the simplicity of this technique, the resulting probability distribution successfully assigns non-zero estimates to unseen n-grams and is considered to be far more accurate than many other estimators, especially for large amounts of training data (Manning & Schültze, 2003).

A considerable shortcoming of a pure GT estimate approach is that all unseen n-grams will be assigned the same probability value. For example, in a trigram model all three-word permutations that do not happen to occur explicitly in the training data will be assigned the same fixed value. This means that a model built in this way cannot accurately compare unobserved n-grams, even though some may be clearly more likely then others. For example, assuming that none of the following trigrams have been

[1] For higher frequency n-grams the ML estimate tends to become gradually more accurate.

observed in the training data, the simple GT estimate cannot tell us that (i) is somewhat less likely than (g) and (h).

g. where she went
h. she went where
i. she where went

Because GT treats n-grams as atomic objects with no internal structure, this technique is not normally seen in isolation, but rather used as a basis for more robust techniques that take into account not only the n-grams themselves, but the combination of lower order n-grams that compose them. Thus, when e.g., a trigram has not been observed in the training data, at least we should be able to consider the frequencies of its bigram components to have a clue on its estimate. In the above example, this insight allows us to take into account that the bigram "she where" is itself highly unusual, which helps conclude that (i) should be given a lower estimate than (g) and (h).

Because n-grams of lower order are more frequent, this combined approach has the added advantage of estimating probabilities from more data, and hence with more accuracy. A well-known example of smoothing technique that combines multiple order models is the *Jelinek-Mercer* smoothing (Jelinek & Mercer, 1980), which makes use of held-out data to assign optimal weights to each n-gram component of the probability estimate. For a survey of this and related techniques, see Chen & Goodman (1999).

Having obtained n-gram estimates over a sufficiently large amount of training data, the probability estimate of a whole sentence can be computed as the product of all conditional probabilities of its constituents given each of their n-1 previous words. For example, in a bigram (hence n = 2) model, we may estimate the probability of the sentence "the dog came yesterday" as

$$P(\textit{the dog came yesterday}) = P(\textit{dog} \mid \textit{the}) * P(\textit{came} \mid \textit{dog}) * P(\textit{yesterday} \mid \textit{came})$$ [2]

Intuitively, higher probabilities suggest that the given sentence is more likely to occur in the modelled language, and vice-versa. The consequences of this observation alone should make clear the importance of SLMs in the development of NLP applications in general.

Different models (e.g., based on different training data and / or using different smoothing techniques) may of course vary greatly in their ability to estimate probabilities. One way of evaluating an individual model is by computing the product of all probability estimates given to a particular test set. However, probability estimates do not tell us how different SLMs compare *among themselves*. For example, a probability estimate `p1` = 0.000003 in one model may actually be higher than `p2` = 0.000005 in a different one. For that reason, SLMs are commonly evaluated through measures such as *cross-entropy* and *perplexity*, both of which can be trivially derived from probability estimates as follows.

[2] For simplicity, in this example we omitted the special markers for beginning <*BOS*> and end of sentence <*EOS*>, both of which would usually play a role in the estimate, e.g., the first conditional probability of the example would actually be P(*the* | <*BOS*>).

Given the probability p(T) of a test set T containing w words, the cross-entropy Hp(T) of a particular model on data T is

$$H_p(T) = (-1 / w) * \log_2 p(T)$$

and the related notion of perplexity $PP_p(T)$ is defined as

$$PP_p(T) = 2^{Hp(T)}$$

Both cross-entropy and perplexity values attempt to describe the *uncertainty* of the model (and hence lower entropy and perplexity values are better). To a certain extent, it is still subject to debate which of the two measures best captures the behaviour of an SLM. For details, we report to Charniak (1993) and Manning & Schütze (2003). In the work described in the next section we follow Chen & Goodman (1999) and compute cross-entropy values only.

3. Implementation work

Due to our general interest in statistical MT systems, we decided to implement a general-purpose tool based on a number of customisable, reusable SLMs called *jNina*. The purpose of this tool is twofold: first, we would like to quickly compare multiple text sources (e.g., produced by various MT or other text-generating applications); second, we would like to build new SLMs from scratch based on any corpora provided by the user and have them readily evaluated.

Our work is not to be confused with evaluation metrics for MT such as BLEU (Papineni et. al, 2002), which is intended to measure the closeness between a gold standard translation and multiple alternatives. To put it simply, BLEU measures the quality of alternative translations, whereas we are only interested in measuring the quality of any text (be it a translation or not) with respect to a language model. Regarding the ability to build new SLMs, jNina is intended to work in a way not unlike existing SLM toolkits (Clarkson & Rosenfeld, 1994; Stolcke, 2002). In our work, however, we decided to provide a more user-friendly environment and, in a future development stage, a number of special features to aid the development of MT and other text-generating applications.

As a first step towards the implementation of more robust SLMs, the current release of jNina provides three basic n-gram models (namely, unigrams[3], bigrams and trigrams)[4] using simple Good-Turing estimate (Gale & Sampson, 1995) as the underlying smoothing technique, and a fourth model type that combines these using a customisable weight-based estimator. In the combined model, the weights assigned to trigrams, bigrams and unigrams are presently defined by the user by hand. In the future we intend to implement an algorithm for automatically learning their optimal values from held-out data as in, e.g., Jelinek & Mercer (1980).

[3] Despite the simplicity of these models, it should be noticed that even simple unigram statistics may play a key role in complex NLP tasks, e.g., the automatic evaluation of text summaries in (Lin & Hovy, 2003).

[4] Although the use of *fourgrams* has become popular in recent years, we found that presently our training corpus is not sufficiently large to support such models.

The program is divided into two basic modules: SLM generation and evaluation. A screenshot of the 'Model Generation' tab is shown in Figure 1.

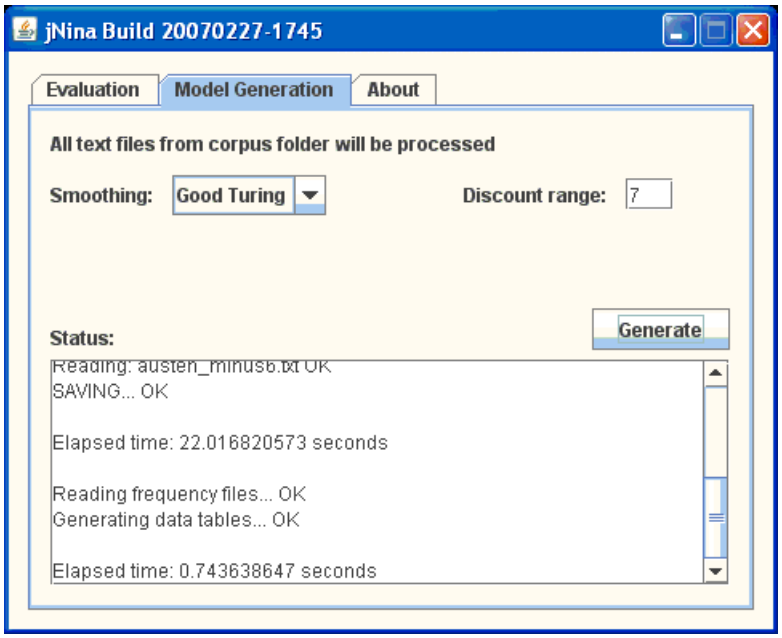

Figure 1. Model Generation

The model generation works as follows. A user (e.g., a researcher interested in comparing multiple MT approaches) may select an existing SLM to be evaluated against a number of sentences (or text files), or, alternatively, may decide to create new models from scratch by providing their relevant parameters (e.g., the discount range parameter for Good-Turing estimates etc). In the latter case (i.e., when building SLMs from scratch) the program will search all text documents found in its "*\corpus*" folder and use them as training data to build new unigram, bigram and trigram models according to the specification. Once generated, the models will be saved to individual files and may be applied to any text specified by the user.

A screenshot of the 'Model Evaluation' tab is shown in Figure 2. In this module the user must provide some test data (either as a sentence typed directly in the GUI, or as a filename of a document containing the input) and selects an existing model to be evaluated against it. The tool is currently able to read only raw text files without markup. Sentence boundaries are detected by means of simple heuristics based on punctuation marks and a customisable lists of special characters.

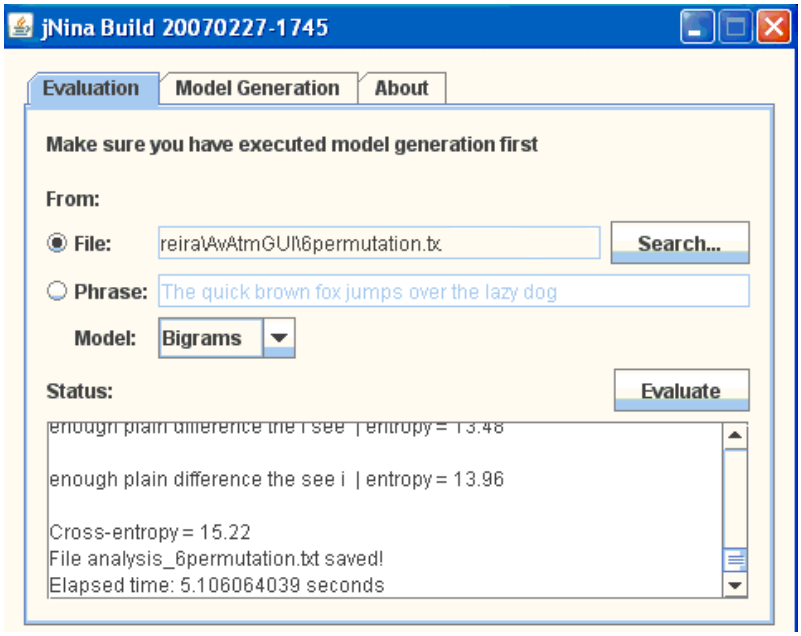

Figure 2. Model Evaluation

Once the user hits the 'Evaluate' button, jNina computes the cross-entropy estimates for each input sentence and displays the results. At the end of the evaluation, the measure of cross-entropy for the entire set of sentences is also displayed, and the evaluation output is saved to a text file. This evaluation can be useful in two ways: by keeping a fixed SLM under evaluation whilst varying the input text, it is possible to compare alternative text sources as required, for example, in our current research on MT systems. Conversely, by choosing a different SLM whilst keeping a fixed input text, it is possible to evaluate multiple models and determine which one works best. This is generally useful, for example, to the researcher interested in building her own SLM for a particular NLP application or language.

4. Preliminary Evaluation

As a preliminary assessment of our work, we carried out a simple test to illustrate how well our SLMs were able to predict the likelihood of random strings made out of permutations of a given input sentence. For this purpose, we took a small collection of four Jane Austen's novels[5] containing 485,409 words in total. For ease of processing, all words were set to lower case and special characters such as the underline symbol or dots as in "Mrs." were removed.

One single sentence was randomly removed from the text, namely, "I see the difference plain enough", and set apart to build our test data. The reminder of the text was used as training data to build a simple bigram SLM using GT smoothing[6].

[5] "Emma", "Pride and Prejudice", "Sense and Sensibility" and "Persuasion", all taken from the Gutenberg project website (http://www.gutenberg.org).

[6] Our present data set was not sufficiently large for building a useful trigram model, or a combination of these and lower order n-grams.

By shuffling the word order of the selected sentence, we generated all its possible (6! = 720) permutations. These included the original string itself, a few (possibly acceptable) variations such as "I plain enough see the difference" and a majority of ungrammatical permutations.

Using the set of 720 unseen sentences as test data, we would like our bigram model to assign low cross-entropy values to the starting sentence and its well-formed permutations, and progressively higher values to the less acceptable ones. The following Table 1 summarizes our findings.

Table 1. Most likely sentences according to the Bigram model of Austen's novels

#	test sentence	H_P
1	*I see the difference plain enough*	6.29
2	*plain enough I see the difference*	6.80
3	*I see the plain enough difference*	7.16
4	*the difference plain enough I see*	7.43
5	*I see the difference enough plain*	8.11
6	*enough I see the difference plain*	8.14
7	*the plain enough I see difference*	8.19
8	*I see the plain difference enough*	8.36
9	*see the difference plain enough I*	8.48
10	*difference I see the plain enough*	8.56
11	*the difference I see plain enough*	8,61
…	…	…
720	*difference the i enough see plain*	*16.24*

Not surprisingly, the original sentence (#1) fares best of all, followed by three acceptable variations (#2-4). From sentence (#5) on there is a trend towards progressively more problematic instances[7], up to the sentence that scored worst of all (#720) "difference the I enough see plain".

In an informal inspection of the entire 720-sentences set we could not find any acceptable instances beyond sentence #14 (not shown). Such seemingly high concentration of well-formed sentences on the top positions may suggest that our small SLM is indeed able to rate sentences closest to well-formed English much higher than their ill-formed permutations. We are however are aware that more evidence is needed to substantiate this claim, and that the definition of what counts as a 'well-formed' sentence needs to be made clear, e.g., making use of human judges and to compare their estimates to ours. This is precisely what we intend to do once we have built a more powerful SLM using a combination of multiple order n-grams and, crucially, a much larger training data set.

[7] Given the small amount of training data, however, the distinction between problematic and unproblematic is not clear-cut. For example, we observe that sentence #11 is still rather acceptable.

5. Final Remarks and Future Work

This paper described our ongoing efforts to implement jNina, a general-purpose tool for statistical language modelling. The work is mainly motivated by the need to quickly compare the output of multiple MT systems but, in a broader sense, is expected to support the development of various NLP applications, ranging from text interpretation to generation.

In its current stage our tool is still very similar (and indeed inferior) to existing language model toolkits. However, a number of enhancements to aid the development of statistical MT systems are under way. These include, for example, the ability to compare multiple documents in a single task and the selection of the winning translation of each sentence.

To illustrate part of what remains to be done, the following Figure 3 presents a sketch (in Portuguese) of the planned implementation work. Each text column is intended to display the evaluation of a given source (e.g., the outputs of multiple MT systems or sentences provided by the user) and to highlight the winning sentence of each group[8]. At the bottom, the number of victories of each source and the corresponding percentage will also be presented, possibly followed by additional evaluation statistics (to be defined).

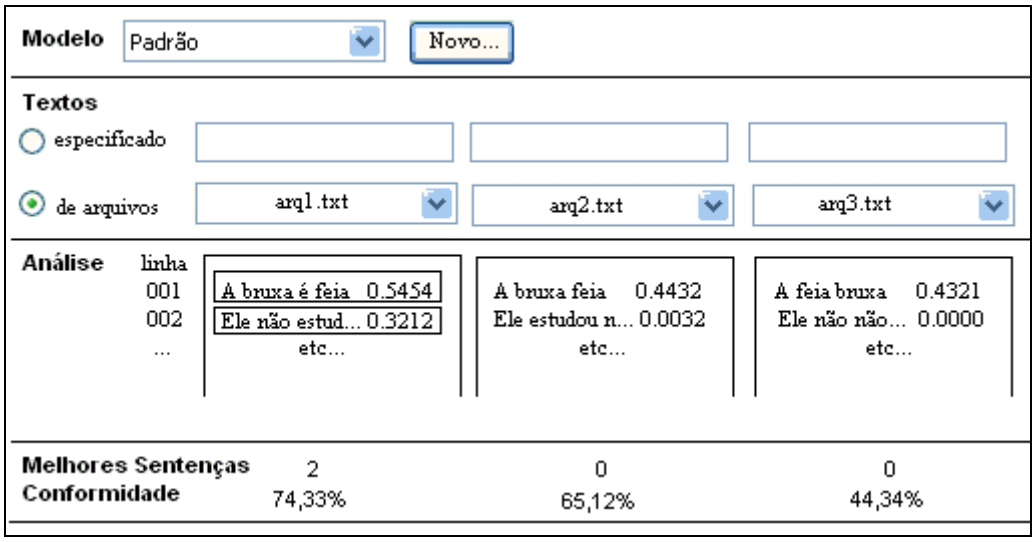

Figure 3. A sketch of the MT-evaluation module (to be implemented)

Besides the above implementation task, we are now in the process of collecting larger corpora to build higher-quality SLMs, and this will be followed by a comprehensive evaluation work. For this purpose, we are developing pre-processing tools required to handle the 40-million words NILC corpus of Portuguese[9]. Other developments to follow include the implementation of basic techniques for treatment of out-of-vocabulary words, and a wider choice of smoothing techniques.

[8] In this tentative example we chose to highlight the sentence of lowest probability estimate in each line.

[9] http://nilc.icmc.sc.usp.br/nilc/tools/corpora.htm

Acknowledgments

The first author acknowledges support from the "Ensinar com Pesquisa" grant of the University of São Paulo (USP / EACH). The second author is partially supported by FAPESP grant number 2006/03941-7.

References

Brown, P.F. et al. (1990) A statistical approach to machine translation. Computational Linguistics vol. 16, pp.79-85.

Brown, P.F. et al. (1993) The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics, Vol. 19, pp.263-311.

Charniak, E. (1993) Statistical Language Learning. Cambridge: MIT Press.

Chen, S.F. and J. Goodman (1999) An empirical study of smoothing techniques for language modeling. Computer Speech and Language 13, pp.359-394.

Clarkson, P. and R. Rosenfeld (1994) The CMU statistical language modeling toolkit as it is used in the 1994 ARPA CSR evaluation. Proc. of the Spoken Language Systems Technology Workshop.

Gale, W.A. and G. Sampson (1995) Good-Turing frequency estimation without tears. Journal of Quantitative Linguistics 2:217-237.

Jelinek, F. and R. L. Mercer (1980) Interpolated estimation of Markov source parameters from sparse data. Proc. of the Workshop 'Pattern Recognition in Practice'. Amsterdam, The Netherlands. North-Holland, pp.381-397.

Lin, C-Y. & Hovy, E. (2003) Automatic Evaluation of Summaries Using N-gram Co-Occurrence Statistics. Human Technology Conference HLT-NAACL-2003. Edmonton, Canada, June.

Manning, C. D. and Schütze, H. (2003) Foundations of Statistical Natural Language Processing. Cambridge: MIT Press.

Papineni, K., S. Roukos, T. Ward and W-J. Zhu (2002) BLEU: a Method for Automatic Evaluation of Machine Translation. 40th Annual Meeting of the Association for Computational Linguistics, pp. 311-318. Philadelphia, PA.

Stolcke, A. (2002) SRILM - An extensible language modeling toolkit. International Conference on Spoken Language Processing, vol. 2, (Denver, CO), pp. 901-904, September.