SRILM — AN EXTENSIBLE LANGUAGE MODELING TOOLKIT

Andreas Stolcke

Speech Technology and Research Laboratory
SRI International, Menlo Park, CA, U.S.A.
http://www.speech.sri.com/

ABSTRACT

SRILM is a collection of C++ libraries, executable programs, and helper scripts designed to allow both production of and experimentation with statistical language models for speech recognition and other applications. SRILM is freely available for noncommercial purposes. The toolkit supports creation and evaluation of a variety of language model types based on N-gram statistics, as well as several related tasks, such as statistical tagging and manipulation of N-best lists and word lattices. This paper summarizes the functionality of the toolkit and discusses its design and implementation, highlighting ease of rapid prototyping, reusability, and combinability of tools.

1. INTRODUCTION

Statistical language modeling is the science (and often art) of building models that estimate the prior probabilities of word strings. Language modeling has many applications in natural language technology and other areas where sequences of discrete objects play a role, with prominent roles in speech recognition and natural language tagging (including specialized tasks such as part-of-speech tagging, word and sentence segmentation, and shallow parsing). As pointed out in [1], the main techniques for effective language modeling have been known for at least a decade, although one suspects that important advances are possible, and indeed needed, to bring about significant breakthroughs in the application areas cited above—such breakthroughs just have been very hard to come by [2, 3].

Various software packages for statistical language modeling have been in use for many years—the basic algorithms are simple enough that one can easily implement them with reasonable effort for research use. One such package, the CMU-Cambridge LM toolkit [1], has been in wide use in the research community and has greatly facilitated the construction of language models (LMs) for many practitioners.

This paper describes a fairly recent addition to the set of publicly available LM tools, the SRI Language Modeling Toolkit (SRILM). Compared to existing LM tools, SRILM offers a programming interface and an extensible set of LM classes, several non-standard LM types, and more a comprehensive functionality that goes beyond language modeling to include tagging, N-best rescoring, and other applications. This paper describes the design philosophy and key implementation choices in SRILM, summarizes its capabilities, and concludes by discussing deficiencies and plans for future development. For lack of space we must refer to other publications for an introduction to language modeling and its role in speech recognition and other areas [3, 4].

2. DESIGN GOALS AND HISTORY

SRILM grew out of a dissatisfaction with previously available LM tools at SRI, and a desire to design an LM toolkit from the ground up, with the following goals in mind:

- Efficient and careful implementation of state-of-the-art LM algorithms, to support development of competitive systems, mainly in speech recognition.
- Flexibility and extendibility, so as to facilitate research into new types of LMs, while being able to reuse existing components.
- A rational, clean software design, providing both an application programming interface (API) and a convenient toolbox of commands for LM building and testing.

The design was influenced by other related software implementations. The CMU-Cambridge toolkit [1], and discussions with its original author, Roni Rosenfeld, served as a general inspiration and reference point. The HTK Lattice Toolkit [5] (to which SRILM has an interface) provided many good ideas for a viable and efficient API for language models. The decision to explore object-oriented design was based on a prior project, an implementation of various types of statistical grammars in the Common Lisp Object System [6]. The software build system was borrowed from SRI's Decipher$^{\mathrm{TM}}$ speech recognition system [7].

A first implementation with minimal functionality for standard N-gram models was created prior to the 1995 Johns Hopkins Language Modeling Summer Workshop [8]. By the end of the workshop, support for dynamic LM interpolation and N-best rescoring had been added, and a small community of users outside SRI with an associated mailing list existed. Over the next four years a series of alpha versions were made available to this small group, while much of the current functionality (described below) was being added. In July 1999 a beta version was released for general distribution under an open source license, followed about a year later by version 1.0. As of this writing, the latest released version is 1.3, which added a word graph rescoring tool, a test suite, and support for Windows platforms (previous versions were Unix-only). Most ongoing government-funded LM research and development at SRI is based on SRILM; we therefore expect a steady stream of functionality enhancements (as well as bug fixes) to continue.

3. FUNCTIONALITY

3.1. Basic LM operations

The main purpose of SRILM is to support language model estimation and evaluation. Estimation means the creation of a model from training data; evaluation means computing the probability of a test corpus, conventionally expressed as the test set perplexity. Since most LMs in SRILM are based on N-gram statistics, the tools to accomplish these two purposes are named `ngram-count` and `ngram`, respectively. A standard LM (trigram with Good-Turing discounting and Katz backoff for smoothing) would be created by

```
ngram-count -text TRAINDATA -lm LM
```

The resulting LM may then be evaluated on a test corpus using

```
ngram -lm LM -ppl TESTDATA -debug 2
```

The `ngram -debug` option controls the level of detail of diagnostic output. A value of 2 means that probabilities are to be reported at the word level, including the order of N-gram used, in addition to the standard log probabilities and perplexities. Some additional statistics that also help gauge LM quality are the number of out-of-vocabulary (OOV) words and the "hit rates" of various levels of N-grams (in LMs based on N-grams) [1]; these are either computed by `ngram` itself or (as in the case of hit rates) tallied by auxiliary scripts that analyze the `ngram` output.

SRILM by itself performs no text conditioning, and treats everything between whitespace as a word. Normalization and tokenization of text are highly corpus-dependent, and are typically accomplished with filters that preprocess the data.

3.2. Bells and whistles

The programs `ngram-count` and `ngram` have a rather large number of options to control the many parameters of LM estimation and testing. The most important parameters for LM training are

- the order of N-grams to use (e.g., unigram, bigram). There is no built-in limit on the length of N-grams.
- the type of discounting algorithm to use. Currently supported methods include Good-Turing, absolute, Witten-Bell, and modified Kneser-Ney [9]. Each of these discounting methods requires its own set of parameters, as well as a choice of whether higher- and lower-order estimates are to be combined via backoff or interpolation.
- an optional predefined vocabulary to limit or augment the set of words from the training data.
- whether unknown words are to be discarded or treated as a special "unknown word" token.
- whether to collapse case distinctions in the input text.

Beyond LM estimation, `ngram-count` performs useful N-gram count manipulations, such as generating counts from text, summing count files, and recomputing lower-order counts from higher-order counts. `ngram-count` handles integer or fractional counts, although only a subset of the smoothing algorithms supports the latter (generally speaking, those that rely on counts-of-counts statistics do not).

The main parameters controlling LM evaluation are the order of N-gram to use (which can be lower than what the LM includes, so that a 4-gram model may conveniently be used also as a bigram or trigram model), and the variant of N-gram model to use—for example, a word-based, class-based, or interpolated N-gram, as well as any additional parameters associated with that type of LM. The types of models supported are described further in Section 3.3.

Beyond LM evaluation, `ngram` is really a tool to manipulate LMs in a variety of ways. Besides computing test set log probabilities from text or counts, it can

- renormalize a model (recomputing backoff weights)
- approximate a class-based or interpolated N-gram with a standard word-based backoff LM
- prune N-gram parameters, using an entropy criterion [10]
- prepare LMs for conversion to finite-state graphs by removing N-grams that would be superseded by backoffs
- generate random sentences from the distribution embodied by the LM.

The ability to approximate class-based and interpolated N-gram LMs by a single word N-gram model deserves some discussion. Both of these operations are useful in situations where other software (e.g., a speech recognizer) supports only standard N-grams. Class N-grams are approximated by expanding class labels into their members (which can contain multiword strings) and then computing the marginal probabilities of word N-gram strings. This operation increases the number of N-grams combinatorially, and is therefore feasible only for relatively small models.

An interpolated backoff model is obtained by taking the union of N-grams of the input models, assigning each N-gram the weighted average of the probabilities from those models (in some of the models this probability might be computed by backoff), and then renormalizing the new model. We found that such interpolated backoff models consistently give slightly lower perplexities than the corresponding standard word-level interpolated models. The reason could be that the backoff distributions are themselves obtained by interpolation, unlike in standard interpolation, where each component model backs off individually.

3.3. Language model types

Besides the standard word-based N-gram backoff models, SRILM implements several other LM types, most of them based on N-grams as basic building blocks.

Class-based models — N-grams over word classes are an effective way to increase the robustness of LMs and to incorporate domain knowledge, e.g., by defining word classes reflecting the task semantics. SRILM allows class members to be multiword strings (e.g., "san francisco" can be a member of class "CITYNAME"). This, and the fact that words can belong to more than one class, requires the use of dynamic programming to evaluate a class N-gram. Word classes may be defined manually or by a separate program, `ngram-class`, which induces classes from bigram statistics using the Brown algorithm [11].

Cache models — This well-known LM technique assigns nonzero probability to recent words, thus modeling the tendency of words to reoccur over short spans [12]. They are usually interpolated with a standard model to obtain an adaptive LM.

Disfluency and hidden event language models — Hidden event LMs incorporate special words that appear in the model's N-grams, but are not overt in the observed word stream. Instead, they correspond to the states of a hidden Markov model, and can be used to model linguistic events such as unmarked sentence boundaries. Optionally, these events can be associated with nonlexical likelihoods to condition the LM on other knowledge sources (e.g., prosody) [13]. A special type of hidden event LM can model speech disfluencies by allowing the hidden events to modify the word history; for example, a word deletion event would erase one or more words to model a false start [14].

Skip language models — In this LM, words in the history are probabilistically skipped, allowing more distant words to take their places. The skipping probabilities associated with each word are estimated using expectation maximization.

HMMs of N-grams — This LM consists of a hidden Markov model (HMM) where each state is associated with its own N-gram distribution. The model generates from a certain state until the local N-gram issues an end-of-sentence, at which point it transitions probabilistically to a neighboring state. HMMs of N-grams provide a general framework that can encode a variety of LM types proposed in the literature, such as sentence-level mixtures [15] and pivot LMs [16].

Dynamically interpolated LMs — Two or more LMs can be interpolated linearly at the word level such that the interpolation weights reflect the likelihoods of the models given the recent N-gram history [8]. With a null history, we obtain the usual static LM interpolation approach that is often used to combine different sources of training material into a single model.

3.4. Other applications of language models

Over the years SRILM has evolved to include tools that go beyond simple LM construction and evaluation, covering mainly LM applications arising in speech recognition.

`disambig` — an HMM-based tagger that uses N-gram LMs of arbitrary order to model the prior on tag sequences.

`hidden-ngram` — a word boundary tagger, based on hidden event N-gram models.

`segment-nbest` — a rescoring tool that applies a language model over a sequence of adjoining N-best lists, thereby overcoming sentence segmentation mismatch [17].

`lattice-tool` — a tool to rescore and expand word lattices.

`nbest-lattice` — a tool to perform word error minimization on N-best lists [18] or construct confusion networks ("sausages") [19]. Together with a helper script, this tool also implements a word posterior-based N-best generalization of the ROVER algorithm [20, 21].

`nbest-scripts` — a collection of wrapper scripts that manipulate and rescore N-best lists.

`pfsg-scripts` — for converting LMs to word graphs.

`nbest-optimize` — optimizes log linear score combination for word posterior-based ("sausage") decoding.

4. DESIGN AND IMPLEMENTATION

SRILM is designed and implemented in three layers.

1. At the core are libraries comprising about 50 C++ classes for language models and miscellaneous objects (such as vocabulary symbol tables, N-best lists, word graphs, DP trellises), which in turn make use of a library of efficient container classes (e.g., arrays, hash tables).
2. The 14 main executable tools such as `ngram-count`, `ngram`, and taggers, are written in C++ on top of the API provided by the libraries.
3. Numerous helper and wrapper scripts perform miscellaneous tasks that are more conveniently implemented in the gawk and Bourne shell scripting languages.

We summarize the characteristics of each implementation layer.

4.1. Class libraries

C++ class libraries implement the API of SRILM. Object-oriented programming turns out to be an excellent match for LM implementation, for several reasons. A class hierarchy naturally reflects the specialization relation between different LM types (e.g., a class N-gram is a special case of an N-gram, which is a special case of an LM). Inheritance allows new LM variants to be derived from existing ones with minimal effort. A new LM class minimally needs to define only a `wordProb` function, the method used to compute conditional probabilities given a word and its history.[1] Most LM functions are defined generically, and need not be reimplemented for a new derived LM class. For example, `sentenceProb` is defined in terms of `wordProb` and typically inherited from the generic LM class; however, a given LM can define its own version of `sentenceProb`, for efficiency or to change the standard behavior.

Hash tables, arrays, tries, and other basic data structures have been implemented from scratch, for speed and compactness under the types of uses associated with LM data structures.[2] Templatized data structures and functions are very useful beyond the low-level containers; N-gram statistics and estimation functions, for example, are templatized to support both integer and fractional counts.

4.2. Executable tools

The executable tools are designed to be both self-contained and combinable in useful ways. Thus, as shown earlier, a standard LM can be built from a text file in a single invocation. More complex manipulations are possible by chaining together tools, using the Unix standard input/output and "pipe" mechanism. Thus, a class-based N-gram model can be trained, pruned, expanded into a word trigram model, and interpolated with another model using the following pipeline (some options have been omitted to save space):

```
replace-words-with-classes TRAINDATA | \
ngram-count -text - -lm - | \
ngram -lm - -prune 1e-5 -write-lm - | \
ngram -lm - -expand-classes 3 -write-lm - | \
ngram -lm - -mix-lm LM2 -write-lm MIXLM
```

4.3. Helpers and wrappers

Miscellaneous other tools are implemented in gawk and shell scripts, either because they involve simple text manipulations that are more conveniently done this way (such as `replace-words-with-classes` in the example above), or because they only require a wrapper that combines the basic tools. An example of the latter is `change-lm-vocab`, a script that modifies the vocabulary of an existing N-gram LM. The script eliminates N-grams that have become OOV (using the textual LM format) and then lets `ngram` fill in new unigrams and renormalize the backoff weights. Other scripts parse the diagnostic output of the tools, such as `compute-best-mix`, which computes optimal interpolation weights from `ngram -ppl` output.

[1] Often a `read` function is also needed, but can be borrowed from an existing class if the same external representation is used, as is frequently the case with N-gram based models.

[2] We considered switching to the Standard Template Library (STL) for containers, but found that this would incur a significant loss of both speed and compactness.

4.4. File formats

SRILM uses standard file formats where possible—in particular, the ARPA format for N-gram backoff LMs. Word graphs use SRI's probabilistic finite-state grammar (PFSG) format, which can be converted to and from that used by AT&T's finite state machine toolkit [22]. Where new file formats were needed we chose easy-to-parse textual representations. All the main tools can read and write compressed files, as large amounts of data and models are often associated with LM work. We avoided binary file formats because of their lack of portability and flexibility, and prefer to use compressed textual formats instead.

5. SHORTCOMINGS AND FUTURE DEVELOPMENTS

Many well-established LM techniques are not implemented in SRILM, such as deleted interpolation or maximum entropy modeling, mainly because these have not proven essential in our work so far. One candidate for future addition is a more flexible class-based model, since refinements of class-based LMs seem to provide an effective and efficient way to incorporate grammatical information into the LM [23]. The low-level implementation of data structures is currently biased toward speed and convenience rather than memory usage; it might be worthwhile to reevaluate this choice to accommodate ever-larger training corpora. SRILM currently assumes single-byte character encoding and uses only whitespace for tokenization; it would be desirable to include support for multi-byte character sets and SGML-tagged formats at some point. Ultimately, however, development of the toolkit will continue to be driven by research needs, and is therefore hard to predict.

Availability. SRILM is freely available for noncommercial users under an Open Source Community License, designed to ensure that enhancements by others find their way back into the user community. Licensing for commercial purposes is also available. Documentation and software are online at http://www.speech.sri.com/projects/srilm/.

6. ACKNOWLEDGMENTS

Fuliang Weng wrote the initial version of the lattice rescoring tool in SRILM; Dimitra Vergyri developed the score combination optimizer based on simplex search; Anand Venkataraman contributed N-best decoding and other enhancements to the statistical tagging tools. Development of SRILM has benefited greatly from its use and constructive criticism by many colleagues at SRI, the Johns Hopkins summer workshops, and the larger research community. Barbara Peskin helped improve this paper with valuable suggestions. The work described here was in part supported by DARPA under contract N66001-97-C-8544 and by NSF-STIMULATE grant IRI-9619921. The views herein are those of the author and do not reflect the policies of the funding agencies.

7. REFERENCES

[1] P. Clarkson and R. Rosenfeld, "Statistical language modeling using the CMU-Cambridge toolkit", in G. Kokkinakis, N. Fakotakis, and E. Dermatas, editors, *Proc. EUROSPEECH*, vol. 1, pp. 2707–2710, Rhodes, Greece, Sep. 1997.

[2] F. Jelinek, "Up from trigrams! The struggle for improved language models", *in Proc. EUROSPEECH*, pp. 1037–1040, Genova, Italy, Sep. 1991.

[3] R. Rosenfeld, "Two decades of statistical language modeling: Where do we go from here?", *Proceedings of the IEEE*, vol. 88, 2000.

[4] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice-Hall, Upper Saddle River, NJ, 2000.

[5] J. J. Odell, *Lattice and Language Model Toolkit Reference Manual*, Entropic Cambridge Research Laboratories, Inc., 1995.

[6] A. Stolcke, *Bayesian Learning of Probabilistic Language Models*, PhD thesis, University of California, Berkeley, CA, July 1994.

[7] H. Murveit, J. Butzberger, V. Digalakis, and M. Weintraub, "Large-vocabulary dictation using SRI's DECIPHER speech recognition system: Progressive search techniques", *in Proc. ICASSP*, vol. II, pp. 319–322, Minneapolis, Apr. 1993.

[8] M. Weintraub, Y. Aksu, S. Dharanipragada, S. Khudanpur, H. Ney, J. Prange, A. Stolcke, F. Jelinek, and E. Shriberg, "LM95 Project Report: Fast training and portability", Research Note 1, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, Feb. 1996.

[9] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling", Technical Report TR-10-98, Computer Science Group, Harvard University, Aug. 1998.

[10] A. Stolcke, "Entropy-based pruning of backoff language models", *in Proceedings DARPA Broadcast News Transcription and Understanding Workshop*, pp. 270–274, Lansdowne, VA, Feb. 1998. Morgan Kaufmann.

[11] P. F. Brown, V. J. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language", *Computational Linguistics*, vol. 18, pp. 467–479, 1992.

[12] R. Kuhn and R. de Mori, "A cache-base natural language model for speech recognition", *IEEE PAMI*, vol. 12, pp. 570–583, June 1990.

[13] A. Stolcke, E. Shriberg, D. Hakkani-Tür, and G. Tür, "Modeling the prosody of hidden events for improved word recognition", *in Proc. EUROSPEECH*, vol. 1, pp. 307–310, Budapest, Sep. 1999.

[14] A. Stolcke and E. Shriberg, "Statistical language modeling for speech disfluencies", *in Proc. ICASSP*, vol. 1, pp. 405–408, Atlanta, May 1996.

[15] R. Iyer, M. Ostendorf, and J. R. Rohlicek, "Language modeling with sentence-level mixtures", *in Proc. ARPA HLT Workshop*, pp. 82–86, Plainsboro, NJ, Mar. 1994.

[16] K. W. Ma, G. Zavaliagkos, and M. Meteer, "Sub-sentence discourse models for conversational speech recognition", *in Proc. ICASSP*, vol. II, pp. 693–696, Seattle, WA, May 1998.

[17] A. Stolcke, "Modeling linguistic segment and turn boundaries for N-best rescoring of spontaneous speech", in G. Kokkinakis, N. Fakotakis, and E. Dermatas, editors, *Proc. EUROSPEECH*, vol. 5, pp. 2779–2782, Rhodes, Greece, Sep. 1997.

[18] A. Stolcke, Y. Konig, and M. Weintraub, "Explicit word error minimization in N-best list rescoring", in G. Kokkinakis, N. Fakotakis, and E. Dermatas, editors, *Proc. EUROSPEECH*, vol. 1, pp. 163–166, Rhodes, Greece, Sep. 1997.

[19] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus in speech recognition: Word error minimization and other applications of confusion networks", *Computer Speech and Language*, vol. 14, pp. 373–400, Oct. 2000.

[20] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER)", *in Proceedings IEEE Automatic Speech Recognition and Understanding Workshop*, pp. 347–352, Santa Barbara, CA, 1997.

[21] A. Stolcke, H. Bratt, J. Butzberger, H. Franco, V. R. Rao Gadde, M. Plauché, C. Richey, E. Shriberg, K. Sönmez, F. Weng, and J. Zheng, "The SRI March 2000 Hub-5 conversational speech transcription system", *in Proceedings NIST Speech Transcription Workshop*, College Park, MD, May 2000.

[22] M. Mohri, F. Pereira, and M. Riley, "FSM Library—general-purpose finite-state machine software tools, version 3.6", http://www.research.att.com/sw/tools/fsm/, 1998.

[23] W. Wang, Y. Liu, and M. P. Harper, "Rescoring effectiveness of language models using different levels of knowledge and their integration", *in Proc. ICASSP*, Orlando, FL, May 2002.