Statistical Language Modeling using SRILM Toolkit

Presented by:
Kamal Eldin Mahmoud

AGENDA

- **Introduction**
- **Basic SRILM Tools**
  - **ngram-count**
  - **ngram**
  - **ngram-merge**
- **Basic SRILM file format**
  - **ngram-format**
  - **nbest-format**

AGENDA

Basic SRILM Scripts

- **Training-scripts**
- **lm-scripts**
- **ppl-scripts**

Introduction

- SRILM is a collection of C++ libraries, executable programs, and helper scripts.

- The toolkit supports creation and evaluation of a variety of language model types based on N-gram statistics.

- The main purpose of SRILM is to support language model estimation and evaluation.

- 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.

Introduction

- 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 0

Basic SRILM Tools

ngram-count

ngram-count generates and manipulates N-gram counts, and estimates N-gram language models from them.

Syntax:

Ngram-count *[* ***-help*** *]* *option ...*

ngram-count options

Each filename argument can be an ASCII file, or a compressed file (name ending in .Z or .gz)

-help
Print option summary.
-version
Print version information.
-order n
Set the maximal order (length) of N-grams to count. This also determines the order of the estimated LM, if any. The default order is 3.
-memuse
Print memory usage statistics.

ngram-count options

-vocab *file*
Read a vocabulary from file.

-vocab-aliases *file*
Reads vocabulary alias definitions from file, consisting of lines of the form

alias word

This causes all tokens alias to be mapped to word.

-write-vocab *file*
-write-vocab-index *file*
Write the vocabulary built in the counting process to file.

ngram-count counting options

-tolower
Map all vocabulary to lowercase.

-text *textfile*
Generate N-gram counts from text file.

-no-sos
Disable the automatic insertion of start-of-sentence tokens in N-gram counting.
-no-eos
Disable the automatic insertion of end-of-sentence tokens in N-gram counting.

-read *countsfile*
Read N-gram counts from a file.

ngram-count counting options

-**read-google** *dir*
Read N-grams counts from an indexed directory structure rooted in dir, in a format developed by Google. The corresponding directory structure can be created using the script *make-google-ngrams* .

-write *file*
-**write-binary** *file*
-write-order n
-writen *file*
Write total counts to file.

-sort
Output counts in lexicographic order, as required for ngram-merge.

ngram-count lm options

-lm *lmfile*
-write-binary-lm
Estimate a backoff N-gram model from the total counts, and write it to *lmfile* .

-unk
Build an ``open vocabulary'' LM.

-map-unk *word*
Map out-of-vocabulary words to *word*.

ngram-count lm options

-cdiscount*n* *discount*
Use Ney's absolute discounting for N-grams of order *n*, using *discount* as the constant to subtract.

-wbdiscount*n*
Use Witten-Bell discounting for N-grams of order *n*.

-ndiscount*n*
Use Ristad's natural discounting law for N-grams of order *n*.

-addsmooth*n* *delta*
Smooth by adding *delta* to each N-gram count.

ngram-count lm options

-kndiscount*n*
Use Chen and Goodman's modified Kneser-Ney discounting for N-grams of order *n*.

-kn-counts-modified
Indicates that input counts have already been modified for Kneser-Ney smoothing.

-interpolate*n*
Causes the discounted N-gram probability estimates at the specified order *n* to be interpolated with lower-order estimates. Only Witten-Bell, absolute discounting, and (original or modified) Kneser-Ney smoothing currently support interpolation.

ngram

Ngram performs various operations with N-gram-based and related language models, including sentence scoring, and perplexity computation.

Syntax:

ngram *[* ***-help*** *] option …*

ngram options

-help
Print option summary.

-version
Print version information.

-order n
Set the maximal N-gram order to be used, by default 3.

-memuse
Print memory usage statistics for the LM.

ngram options

The following options determine the type of LM to be used.

-null

Use a `null' LM as the main model (one that gives probability 1 to all words).

-use-server *S*

Use a network LM server as the main model.

-lm *file*

Read the (main) N-gram model from *file*.

ngram options

-tagged
Interpret the LM as containing word/tag N-grams.

-skip
Interpret the LM as a ``skip'' N-gram model.

-classes *file*
Interpret the LM as an N-gram over word classes.

-factored
Use a factored N-gram model.

-unk
Indicates that the LM is an open-class LM.

ngram options

-ppl *textfile*
Compute sentence scores (log probabilities) and perplexities from the sentences in *textfile*.
The **-debug** option controls the level of detail printed.

-debug 0
Only summary statistics for the entire corpus are printed.

-debug 1
Statistics for individual sentences are printed.

ngram options

-debug 2
Probabilities for each word, plus LM-dependent details about backoff used etc., are printed.

-debug 3
Probabilities for all words are summed in each context, and the sum is printed.

ngram options

-nbest *file*
Read an N-best list in nbest-format and rerank the hypotheses using the specified LM. The reordered N-best list is written to stdout.

-nbest-files *filelist*
Process multiple N-best lists whose filenames are listed in *filelist*.

-write-nbest-dir *dir*
Deposit rescored N-best lists into directory *dir*, using filenames derived from the input ones.

ngram options

-decipher-nbest
Output rescored N-best lists in Decipher 1.0 format, rather than SRILM format.

-no-reorder
Output rescored N-best lists without sorting the hypotheses by their new combined scores.

-max-nbest *n*
Limits the number of hypotheses read from an N-best list.

ngram options

-no-sos
Disable the automatic insertion of start-of-sentence tokens for sentence probability computation.

-no-eos
Disable the automatic insertion of end-of-sentence tokens for sentence probability computation.

ngram-merge

ngram-merge reads two or more lexicographically sorted N-gram count files and outputs the merged, sorted counts.

Syntax:

ngram-merge [-help] [-write outfile] [-float-counts] \ [--] infile1 infile2 ...

Ngram-merge options

-write *outfile*
Write merged counts to *outfile*.

-float-counts
Process counts as floating point numbers.

--
Indicates the end of options, in case the first input filename begins with ``-".

Basic SRILM file format

ngram-format

ngram-format File format for ARPA backoff N-gram models

\data
ngram 1=*n1*
ngram 2=*n2*.
..
ngram *N*=*nN*
\1-grams:
p *w* [*bow*]
...****
2-grams:
p *w1 w2* [*bow*]
...
*****N***-grams:***
p *w1 ... wN*
...
\end

nbest-format

SRILM currently understands three different formats for lists of N-best hypotheses for rescoring or 1-best hypothesis extraction. The first two formats originated in the SRI Decipher(TM) recognition system, the third format is particular to SRILM.

The first format consists of the header

NBestList1.0

followed by one or more lines of the form

(*score*) *w1 w2 w3* ...

where *score* is a composite acoustic/language model score from the recognizer, on the bytelog scale.

nbest-format

The second Decipher(TM) format is an extension of the first format that encodes word-level scores and time alignments. It is marked by a header of the form

NBestList2.0

The hypotheses are in the format

(*score*) *w1* (st: *st1* et: *et1* g: *g1* a: *a1*) *w2* ...

where words are followed by start and end times, language model and acoustic scores (bytelog-scaled), respectively.

nbest-format

The third format understood by SRILM lists hypotheses in the format

ascore lscore nwords w1 w2 w3 ...

where the first three columns contain the acoustic model log probability, the language model log probability, and the number of words in the hypothesis string, respectively. All scores are logarithms base 10.

Basic SRILM Scripts

Training-scripts

These scripts perform convenience tasks associated with the training of language models.

get-gt-counts

Syntax

get-gt-counts max=*K* **out=***name* [*counts* ...] **>** *gtcounts*

Computes the counts-of-counts statistics needed in Good-Turing smoothing. The frequencies of counts up to *K* are computed (default is 10). The results are stored in a series of files with root *name*, ***name*.gt1counts**,..., ***name*.gt*N*counts**.

Training-scripts

make-gt-discounts

Santax:

make-gt-discounts min=*min* max=*max gtcounts*

Takes one of the output files of get-gt-counts and computes the corresponding Good-Turing discounting factors. The output can then be passed to **ngram-count** via the **-gt*n*** options to control the smoothing during model estimation.

Training-scripts

make-abs-discount

Syntax

make-abs-discount *gtcounts*

Computes the absolute discounting constant needed for the **ngram-count -cdiscount***n* options. Input is one of the files produced by **get-gt-counts**.

Training-scripts

make-kn-discount

Syntax

make-kn-discounts min=*min gtcounts*

Computes the discounting constants used by the modified Kneser-Ney smoothing method. Input is one of the files produced by **get-gt-counts**.

Training-scripts

make-batch-counts

Syntax

make-batch-counts *file-list* \ [*batch-size* [*filter* [*count-dir* [*options* ...]]]]

Performs the first stage in the construction of very large N-gram count files. *file-list* is a list of input text files. Lines starting with a `#' character are ignored. These files will be grouped into batches of size *batch-size* (default 10). The N-gram count files are left in directory *count-dir* (``counts'' by default), where they can be found by a subsequent run of **merge-batch-counts**.

Training-scripts

merge-batch-counts

Syntax

merge-batch-counts *count-dir* [*file-list*|*start-iter*]

Completes the construction of large count files. Optionally, a *file-list* of count files to combine can be specified. A number as second argument restarts the merging process at iteration *start-iter*.

Training-scripts

make-google-ngrams

Syntax

make-google-ngrams [**dir=***DIR*] [**per_file=***N*] [**gzip=0**] \ [**yahoo=1**] [*counts-file* ...]

Takes a sorted count file as input and creates an indexed directory structure, in a format developed by Google to store very large N-gram collections. Optional arguments specify the output directory *dir* and the size *N* of individual N-gram files (default is 10 million N-grams per file). The **gzip=0** option writes plain. The **yahoo=1** option may be used to read N-gram count files in Yahoo-GALE format.

Training-scripts

tolower-ngram-counts

Syntax

tolower-ngram-counts [*counts-file* ...]

Maps an N-gram counts file to all-lowercase. No merging of N-grams that become identical in the process is done.

Training-scripts

reverse-ngram-counts

Syntax

reverse-ngram-counts [*counts-file* ...]

Reverses the word order of N-grams in a counts file or stream.

reverse-text

Syntax

reverse-text [*textfile* ...]

Reverses the word order in text files, line-by-line.

Training-scripts

compute-oov-rate

Syntax

compute-oov-rate *vocab* [*counts* ...]

Determines the out-of-vocabulary rate of a corpus from its unigram *counts* and a target vocabulary list in *vocab*.

lm-scripts

add-dummy-bows

Syntax

add-dummy-bows [*lm-file*] **>** *new-lm-file*

Adds dummy backoff weights to N-grams, even where they are not required, to satisfy some broken software that expects backoff weights on all N-grams (except those of highest order).

lm-scripts

change-lm-vocab

Syntax

change-lm-vocab -vocab *vocab* **-lm** *lm-file* **-write-lm** *new-lm-file* \ [**-tolower**] [**-subset**] [*ngram-options* ...]

Modifies the vocabulary of an LM to be that in *vocab*. Any N-grams containing OOV words are removed, new words receive a unigram probability, and the model is renormalized. The **-tolower** option causes case distinctions to be ignored. **-subset** only removes words from the LM vocabulary, without adding any.

lm-scripts

make-lm-subset

Syntax

make-lm-subset *count-file*|**-** [lm-file |**-**] **>** *new-lm-file*

Forms a new LM containing only the N-grams found in the *count-file*. The result still needs to be renormalized with **ngram -renorm** .

lm-scripts

get-unigram-probs

Syntax

get-unigram-probs [**linear=1**] [*lm-file*]

Extracts the unigram probabilities in a simple table format from a backoff language model. The **linear=1** option causes probabilities to be output on a linear (instead of log) scale.

ppl-scripts

These scripts process the output of the ngram option **-ppl** to extract various useful information.

add-ppls

Syntax

add-ppls [*ppl-file* ...]

Takes several ppl output files and computes an aggregate perplexity and corpus statistics.

ppl-scripts

subtract-ppls

Syntax

subtract-ppls *ppl-file1* [*ppl-file2* ...]

Similarly computes an aggregate perplexity by removing the statistics of zero or more *ppl-file2* from those in *ppl-file1*.

ppl-scripts

compare-ppls

Syntax

compare-ppls [**mindelta=***D*] *ppl-file1 ppl-file2*

Tallies the number of words for which two language models produce the same, higher, or lower probabilities. The input files should be **ngram -debug 2 -ppl** output for the two models on the same test set. The parameter *D* is the minimum absolute difference for two log probabilities to be considered different.

ppl-scripts

compute-best-mix

Syntax

compute-best-mix [**lambda='***l1 l2* ...**'**]
[**precision=***P*] \ *ppl-file1* [*ppl-file2* ...]

Takes the output of several **ngram -debug 2 –ppl** runs on the same test set and computes the optimal interpolation weights for the corresponding models. Initial weights may be specified as *l1 l2 ...*. The computation is iterative and stops when the interpolation weights change by less than *P* (default 0.001).

ppl-scripts

compute-best-sentence-mix

Syntax

compute-best-sentence-mix [**lambda='***l1 l2* ...**'**] [**precision=***P*] \ *ppl-file1* [*ppl-file2* ...]

similarly optimizes the weights for sentence-level interpolation of LMs. It requires input files generated by **ngram -debug 1 -ppl**.

Thank You ☺