IRST Language Modeling Toolkit
USER MANUAL

M. Federico, N. Bertoldi, M. Cettolo
FBK-irst, Trento, Italy

June 26, 2015

Version 5.80.08

The official website of **IRSTLM Toolkit** is

http://hlt.fbk.eu/en/irstlm

It contains this manual, source code, examples and regression tests.

Users of **IRSTLM Toolkit** might cite in their publications:

> M. Federico, N. Bertoldi, M. Cettolo, *IRSTLM: an Open Source Toolkit for Handling Large Scale Language Models*, Proceedings of Interspeech, Brisbane, Australia, pp. 1618-1621, 2008.

[1] `http://www.gnu.org/licenses/gpl-3.0.html`

Contents

1 Introduction

This manual illustrates the functionalities of the IRST Language Modeling (LM) toolkit (**IRSTLM Toolkit**). It should put you quickly in the condition of:

- extracting the dictionary from a corpus
- extracting n-gram statistics from it
- estimating n-gram LMs using different smoothing criteria
- saving a LM into several textual and binary file
- adapting a LM on task-specific data
- estimating and handling gigantic LMs
- pruning a LM
- reducing LM size through quantization
- querying a LM through a command or script

IRSTLM Toolkit features very efficient algorithms and data structures suitable to estimate, store, and access very large LMs.
IRSTLM Toolkit provides adaptation methods to effectively adapt generic LM to specific task when only little task-related data are available.
IRSTLM Toolkit provides standalone programs for all its functionalities, as well as library for its exploitation in other softwares, like for instance speech recognizers, machine translation decoders, and POS taggers.
IRSTLM Toolkit has been integrated into a popular open source SMT decoder called `Moses`[2], and is compatible with LMs created with other tools, such as the SRILM Tooolkit[3].

Acknowledgments. Users of this toolkit might cite in their publications:

> M. Federico, N. Bertoldi, M. Cettolo, *IRSTLM: an Open Source Toolkit for Handling Large Scale Language Models*, Proceedings of Interspeech, Brisbane, Australia, pp. 1618-1621, 2008.

References to introductory material on n-gram LMs are given in Appendix A.

[2] http://www.statmt.org/moses/
[3] http://www.speech.sri.com/projects/srilm

2 Installation

> The installation procedure has been tested using the `bash` shell on the following operating systems: Mac OSx 10.6.8 (Snow Leopard), Ubuntu 14.04 LTS (trusty), Scientific Linux release 6.3 (carbon).

In order to install **IRSTLM Toolkit** on your machine, please perform the following steps.

2.1 Step 0: Preparation of the Configuration Scripts

Run the following command to prepare up-to-date configuration scripts.

```
$> ./regenerate-makefiles.sh [--force]
```

Warning: Run with the "–force" parameter if you want to recreate all links to the autotools.

2.2 Step 1: Configuration of the Compilation

Run the following command to prepare up-to-date compilation scripts, and to optionally set the installation directory (parameter "`-prefix`".

```
$> ./configure [--prefix=/path/to/install] [optional-parameters]
```

You can set other optional parameters to modify the standard compilation behavior.

```
--enable-doc|--disable-doc
  Enable or Disable (default) creation of documentation
--enable-trace|--disable-trace
  Enable (default) or Disable trace info at run-time
--enable-debugging|--disable-debugging
  Enable or Disable (default) debugging info ("-g -O2")
--enable-profiling|--disable-profiling
  Enable or Disable (default) profiling info
--enable-caching|--disable-caching
  Enable or Disable (default) internal caches
  to store probs and other info
--enable-interpolatedsearch|--disable-interpolatedsearch
  Enable or Disable (default) interpolated search for n-grams
--enable-optimization|--disable-optimization
  Enable or Disable (default) C++ optimization info ("-O3")
```

Run the following command to get more details on the compilation options.

```
$> configure --help
```

2.3 Step 2: Compilation

```
$> make clean
$> make
```

2.4 Step 3: Installation

```
$> make install
```

Libraries and commands are generated, respectively, under the directories
`/path/to/install/lib` and `/path/to/install/bin`.
If enabled and PdfLatex is installed, this user manual (in pdf) is generated under the directory
`/path/to/install/doc`.
Although caching is not enabled by default, it is highly recommended to activate through its compilation flag "`--enable-caching`".

2.5 Step 4: Environment Settings

Set the environment variable `IRSTLM` to `/path/to/install`.
Include the command directory `/path/to/install/bin` into your environment variable `PATH`. For instance, you can run the following commands

```
$> export IRSTLM=/path/to/install/
$> export PATH=${IRSTLM/bin:${PATH}
```

2.6 Step 5: Regression Tests

If the installation procedure succeeds, you can also run the regression tests to double-check the integrity of the software. Please go to Section 10 to learn hot to run the regression tests.
Regression tests should be run also in the case of any change made in the source code.

3 Getting started

After a successful installation, you are ready to use **IRSTLM Toolkit**.
In this Section, a basic 4-step procedure is given to estimate a LM and to compute its perplexity on a text. Many changes to this procedure can be done in order to optimize effectiveness and efficiency according to your needs.
Please refer to Section 6 to learn more about each IRSTLM commands, and to Section 7 to get hints about IRSTLM functionalities.

> All programs assume that the environment variable **IRSTLM** is correctly set to `/path/to/install/doc`, and that that environment variable **PATH** includes the command directory `/path/to/install/bin`. see above

Data used in the following usage examples can be found in an archive you can download from the official website of **IRSTLM Toolkit**. Most of them are very little, so the reported figures are not reliable.

3.1 Preparation of Training Data

In order to estimate a Language Model, you first need to prepare your training corpus. The corpus just consists of a text. We assume that the text is already preprocessed according to the user needs; this means that lowercasing, uppercasing, tokenization, and any other text transformation has to be performed beforehand with other tools.
You can only decide whether you are interested that **IRSTLM Toolkit** is aware of sentence boundaries, i.e. where a sentence starts and ends. Otherwise, it considers the corpus as a continuous stream of text, and does not identify sentence splits.
The following script adds start and end symbols (`<s>` and `</s>`, respectively) to all lines in your training corpus.

```
$> cat train.txt | add-start-end.sh > train.txt.se
```

IRSTLM Toolkit does not compute probabilities for cross-sentence n-grams, i.e. n-grams including the pair `</s> <s>`.

> **IRSTLM Toolkit** assumes that each line corresponds to a sentence, regardless the presence of punctuation inside or at the end of the line.

> Start and end symbols (`<s>` and `</s>`) should be considered reserved symbols, and used only as sentence boundaries.

3.2 Computation of n-gram statistics

You can now collect n-gram statistics for your training data (3-gram in this example) by running the command:

```
$> ngt -i=train.txt.se -n=3 -o=train.www -b=yes
```

The n-grams counts are saved in the binary file "train.www".

3.3 Estimation of the LM

You can now estimate a n-gram LM (3-gram LM in this example) smoothed according to the Linear Witten Bell method by running the command:

```
$> tlm -tr=train.www -n=3 -lm=LinearWittenBell -obin=train.blm
```

The estimated LM is saved in the binary file "train.blm".

3.4 Computation of the Perplexity

With the estimated LM, you can now compute the perplexity of any text contained in "test.txt" by running the commands below.
To be compliant with the training data actually used to estimate the LM, start and end symbols are added to the text as well.

```
$> cat test.txt | add-start-end.sh > test.txt.se
$> compile-lm  train.blm  --eval=test.txt.se
```

which produces the output:

```
%% Nw=1009 PP=8547.90 PPwp=6870.51 Nbo=983 Noov=119 OOV=11.79%
```

The output shows the number of words (`Nw`), the LM perplexity (`PP`), the portion of PP due to the out-of-vocabulary words (`PPwp`), the amount of backoff calls(`Nbo`) required for computing PP, the amount of out-of-vocabulary words (`Noov`), and the out-of-vocabulary rate (`OOV`).

4 LM File Formats

IRSTLM Toolkit supports several types of input and output formats for handling LMs, n-gram counts, dictionaries.

4.1 File Formats for Dictionary

The dictionary is the data structure exploited by **IRSTLM Toolkit** to store a set of terms.

IRSTLM Toolkit saves a dictionary in textual file format consisting of:

- a header line specifying the most important information about the file itself: the keyword "dictionary", a fixed value 0, and the amount of terms the dictionary contains;
- a set of terms listed according to either their occurrence or their frequency in the data.

Here is an excerpt.

```
dictionary 0 7893
<s>
</s>
solemn
ceremony
marks
....
```

Optionally, the occurrence frequencies of each term can be stored as well; in this case the keyword is "DICTIONARY".
Here is an excerpt.

```
DICTIONARY 0 7893
<s> 5000
</s> 5001
solemn 7
ceremony 59
....
```

> The list order is used by **IRSTLM Toolkit** to define the internal codes of the terms. In the vast majority of cases, it is completely transparent and irrelevant to the user. Only in very few cases highlighted in this manual, this order is crucial.

4.2 File Formats for n-gram Table

The n-gram table is the data structure exploited by **IRSTLM Toolkit** to store a set of n-grams. **IRSTLM Toolkit** stores an n-gram table either in textual or binary formats.

4.2.a Textual format

The textual format consists of:

- a header line specifying the most important information about the file itself: the keyword "nGrAm", the order n of the n-grams, the amount of n-grams the n-gram table contains, and a second keyword representing the table type;
- a second line reporting the size of the dictionary associated to the n-grams;
- the terms of the dictionary (one term per line) with their frequency;
- the list of all n-grams with their counts.

Here is an excerpt.

```
NgRaM 3 76857 ngram
7893
<s> 5000
</s> 5001
solemn 7
ceremony 59
...
<s> <s> <s>     2
<s> <s> </s>    1
<s> </s> </s>   1
<s> solemn ceremony     1
<s> a solemn    1
```

4.2.b Binary format

The binary format is similar, but its main keyword is "`NgRaM`" (different caseing), and the list of n-grams is binarized; hence, the last portion of the binary n-gram table is not user-readable.

4.2.c Google n-gram format

IRSTLM Toolkit supports the Google n-gram format as well both for input and output. This format, always textual, simply consists of the list of all n-grams with their counts. Here is an excerpt.

```
<s> <s> <s>     2
<s> <s> </s>    1
<s> </s> </s>   1
<s> solemn ceremony     1
<s> a solemn    1
...
```

4.2.d Table types

The table type keyword represents the way the n-grams are collected and the way they are exploited for further computation:

- `ngram`: each entry is a standard n-gram, i.e. a contiguous sequence of n terms; they are usually used to estimate a standard n-gram LM;
- `co-occ`K: each entry is a xxxxxx, where K is XXXXXX;
- `hm`S: each entry is a xxxxxx, where K is XXXXXX;

4.3 File Formats for LM

IRSTLM Toolkit handles LM both in textual and binary formats. It provides facilities to save disk space storing probabilities as quantized values instead of floating point values, and to reduce access time saving n-grams in inverted order.

4.3.a Textual Format

The textual format is the well-known ARPA format introduced in DARPA ASR evaluations to exchange LMs. ARPA format is supported by most third party LM toolkit, like SRILM and KenLM.

The ARPA format consists of:

- one block reporting the amount n-grams stored for level m of the LM ($m < n$); this block starts with the keyword "`\data`";
- one block per level reporting the set of m-grams for that level together with their log-probability (first field), and the backoff log-probability (last field); each block starts with the keyword "`\`m`-grams`", with the correct value for m;
- the keyword "`\end\`" closes the file.

Here is an excerpt.

```
\data\
ngram  1=      7894
ngram  2=     46269
ngram  3=     12188
\1-grams:
-4.79871        <s>     -0.826378
....
-1.20244        <unk>
\2-grams:
-3.29024        <s> <s> -0.221849
...
-0.289359       restructuring of
\3-grams:
-0.397606       <s> <s> <s>
```

```
-1.67881        <s> a hong_kong
...
-0.420213       seymf council ,
\end\
```

Empty lines can occur before and after each block.
There is no limit to the order n of n-grams.

> Backoff log-probabilities are not reported if equal to 0; backoff log-probabilities do not exist for the largest order.

4.3.b Quantized Textual Format

This textual format extends the ARPA textual format including codebooks that quantize probabilities and back-off weights of each n-gram level.

The quantized ARPA format consists of:

- a header line specifying the most important information about the file itself: the keyword "qARPA", the order n of the LM, the size of the n codebooks
- one block reporting the amount n-grams stored for level m of the LM ($m < n$); this block starts with the keyword "\data";
- one block per level reporting first the codebooks for the level and then the set of m-grams for that level together with their quantized log probability (first field), and the backoff quantized log-probability (last field); each block starts with the keyword "\m-grams", with the correct value for m;
- the keyword "\end\" closes the file.

Here is an excerpt.

```
qARPA 3 256 256 256
\data\
ngram 1= 7894
ngram 2= 46269
ngram 3= 12188
\1-grams:
256
-4.79885 -99
-4.62261 -1.75587
....
0       <s>     53
186     </s>    0
....
\2-grams:
256
-3.79901 -99
```

```
-3.62278 -3.01953
...
7       <s>     <s>     255
65      <s>     </s>    255
....
\end\
```

4.3.c Intermediate Textual Format

This is an *intermediate* ARPA format used by **IRSTLM Toolkit** for optimizing computation of huge LM. It differs from the ARPA format in two aspects:

- the header line contains only the keyword `iARPA`;
- the first field of each n-gram entry is its smoothed frequency of instead of its log-probability.

4.3.d Binary Format

The binary format supported by **IRSTLM Toolkit** allows for save disk space and upload the LM quicker. The binary format consists of:

- a header line specifying the most important information about the file itself: the keyword "blmt", the order n of the LM, and the amount n-grams stored for level m of the LM ($m < n$);
- a second line reporting the size of the dictionary associated to the n-grams;
- the terms of the dictionary (one term per line) with their frequency, if available;
- a binary section containing n-grams and their probabilities; this portion is not user-readable.

```
blmt 3       7894      46269      12188
7894
<s> 1
</s> 5001
solemn 7
...
_binary_data_
```

4.3.e Quantized Binary Format

The quantized binary format stores the quantized version of a LM.
It consists of:

- a header line specifying the most important information about the file itself: the keyword "Qblmt", the order n of the LM, and the amount n-grams stored for level m of the LM ($m < n$);
- a second line specifying the most important information of the codebooks of each level: the keyword "NumCenters", and the size of the n codebooks;
- a third line reporting the size of the dictionary associated to the n-grams;

- the terms of the dictionary (one term per line) with their frequency, if available;
- a binary section containing the codebooks, the n-grams and their quantized probabilities; this portion is not user-readable.

```
Qblmt 3 7894 46269 12188
NumCenters 256 256 256
7894
<s> 1
</s> 5001
solemn 7
...
_binary_data_
```

4.3.f Inverted Binary Format

IRSTLM Toolkit can store the n-grams in inverted order to speed up access time. This applies to both standard and quantized binary formats, namely `blmt` or `Qblmt`. The keywords are `blmtI` or `QblmtI`, respectively.

5 LM Types

5.1 LM smoothing

5.2 Mixture LM

5.3 Interpolated LM

5.4 Class and Chunk LMs

IRSTLM Toolkit allows the use of class and chunk LMs, and a special handling of input tokens which are concatenation of $N \geq 1$ fields separated by the character #, e.g.

```
word#lemma#part-of-speech#word-class
```

The processing is guided by the format of the file passed to Moses or `compile-lm`: if it contains just the LM, either in textual or binary format, it is treated as usual; otherwise, it is supposed to have the following format:

```
LMMACRO <lmmacroSize> <selectedField> <collapse>
<lmfilename>
<mapfilename>
```

where:

```
LMMACRO is a reserved keyword
<lmmacroSize> is a positive integer
<selectedField> is an integer >=-1
<collapse> is a boolean value (true, false)
<lmfilename> is a file containing a LM (format compatible with {\IRSTLM})
<mapfilename> is an (optional) file with a (one|many)-to-one map
```

The various cases are discussed with examples in the following. Data used in those examples can be found in the directory `example/chunkLM/` which represents the relative path for all the parameters of the referred commands. Note that texts with different tokens (words, POS, word#POS pairs...) used either as input or for training LMs are all derived from the same multifield texts in order to allow direct comparison of results.

5.5 Field selection

The simplest case is that of the LM in `<lmfilename>` referring just to one specific field of the input tokens. In this case, it is possible to specify the field to be selected before querying the LM through the integer `<selectedField>` (0 for the first filed, 1 for the second...). With the value -1, no selection is applied and the LM is queried with n-grams of whole strings. The other parameters are set as:

```
<lmmacroSize> : set to the size of the LM in <lmfilename>
<collapse>    : false
```

The third line optionally reserved to `<mapfilename>` does not exist.

Examples:

5.5.a) selection of the second field:

```
$> compile-lm --eval test/test.w-micro cfgfile/cfg.2ndfield
%% Nw=126 PP=2.68 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
```

5.5.b) selection of the first field:

```
$> compile-lm --eval test/test.w-micro cfgfile/cfg.1stfield
%% Nw=126 PP=9.71 PPwp=0.00 Nbo=76 Noov=0 OOV=0.00%
```

The result of the latter case is identical to that obtained with the standard configuration involving just words:

5.5.c) usual case on words:

```
$> compile-lm --eval test/test.w lm/train.en.blm
%% Nw=126 PP=9.71 PPwp=0.00 Nbo=76 Noov=0 OOV=0.00%
```

5.6 Class LMs

Possibly, a many-to-one or one-to-one map can be passed through the <mapfilename> parameter which has the simple format:

```
w1 class(w1)
w2 class(w2)
 ...
wM class(wM)
```

The map is applied to each component of ngrams before the LM query. Examples:

5.6.a) map applied to the second field:

```
$> compile-lm --eval test/test.w-micro cfgfile/cfg.2ndfld-map
%% Nw=126 PP=16.40 PPwp=0.00 Nbo=33 Noov=0 OOV=0.00%
```

5.6.b) just to assess the correctness of the (16.2.a) result:

```
$> compile-lm --eval test/test.macro lm/train.macro.blm
%% Nw=126 PP=16.40 PPwp=0.00 Nbo=33 Noov=0 OOV=0.00%
```

5.7 Chunk LMs

A particular processing is performed whenever fields are supposed to correspond to microtags, i.e. the per-word projections of chunk labels. By means of the <collapse> parameter, it is possible to activate a processing aiming at collapsing the sequence of microtags defining a chunk. The chunk LM is then queried with ngrams of chunk labels, in an asynchronous manner with respect to the sequence of words, as in general chunks consist of more words.
The collapsing operation is automatically activated if the sequence of microtags is:

```
 TAG( TAG+ TAG+ ... TAG+ TAG)
```

Such a sequence is collapsed into a single chunk label (let us say CHNK) as long as TAG(, TAG+ and TAG) are all mapped into the same label CHNK. The map into different labels or a different use/position of characters (, + and) in the lexicon of tags prevent the collapsing operation even if <collapse> is set to true. Of course, if <collapse> is false, no collapse is attempted.

Warning: In this context, it assumes an important role the parameter <lmmacroSize>: it defines the size of the n-gram before the collapsing operation, that is the number of microtags of the actually processed sequence. <lmmacroSize> should be large enough to ensure that after the collapsing operation, the resulting n-gram of chunks is at least of the size of the LM to be queried (the <lmfilename>). As an example, assuming <lmmacroSize>=6, <selectedField>=1, <collapse>=true and 3 the size of the chunk LM, the following input

```
on#PP average#NP( 30#NP+ -#NP+ 40#NP+ cm#NP)
```

will yield to query the LM with just the bigram (PP,NP), instead of a more informative trigram; for this particular case, the value 6 for <lmmacroSize> is not enough. On the other side, for efficiency reasons, it cannot be set to an unlimited valued. A reasonable value could derive from the average number of microtags per chunk (2-3), which means setting <lmmacroSize> to two-three times the size of the LM in <lmfilename>. Examples:

5.7.a) second field, micro→macro map, collapse:

```
$> compile-lm --eval test/test.w-micro cfgfile/cfg.2ndfld-map-cllps
%% Nw=126 PP=1.84 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%

$> compile-lm --eval test/test.w-micro cfgfile/cfg.2ndfld-map-cllps -d=1
%% Nw=126 PP=1.83774013 ... OOV=0.00% logPr=-33.29979642
```

5.7.b) whole token, micro→macro map, collapse:

```
$> compile-lm --eval test/test.micro cfgfile/cfg.token-map-cllps
%% Nw=126 PP=1.84 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
```

5.7.c) whole token, micro→macro map, NO collapse:

```
$> compile-lm --eval test/test.micro cfgfile/cfg.token-map
%% Nw=126 PP=16.40 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
```

Note that the configuration (16.3.c) gives the same result of that in example (16.2.b), as they are equivalent.

5.7.d) As an actual example related to the "warning" note reported above, the following configuration with usual LM:

```
$> compile-lm --eval test/test.chunk lm/train.macro.blm -d=1
Nw=73 PP=2.85754443 ... OOV=0.00000000% logPr=-33.28748842
```

not necessarily yields the same log-likelihood (logPr) nor the same perplexity (PP) of case (16.3.a). In fact, concerning PP, the length of the input sequence is definitely different (126 tokens before collapsing, 73 after that). Even the logPr is different (-33.29979642 vs. -33.28748842) because in (16.3.a) some 6-grams (<lmmacroSize> is set to 6) after collapsing reduce to n-grams of size less than 3 (the size of lm/train.macro.blm). By setting <lmmacroSize> to a larger value (e.g. 8), the same logPr will be computed.

6 IRSTLM commands

6.1 dict

`dict` is the command which copes with the dictionaries.

- It extracts the dictionary from a corpus or a dictionary;
- It computes and shows the dictionary growth curve;
- It computes and shows the out-of-vocabulary rate on a test corpus.

6.1.a Synopsis

USAGE

```
dict -i=<inputfile> [options]
```

OPTIONS

`Curve`	`c`	show dictionary growth curve; default is false
`CurveSize`	`cs`	default 10
`Freq`	`f`	output word frequencies; default is false
`Help`	`h`	print this help
`InputFile`	`i`	input file (Mandatory)
`IntSymb`	`is`	interruption symbol
`ListOOV`	`oov`	print OOV words to stderr; default is false
`LoadFactor`	`lf`	set the load factor for cache; it should be a positive real value; default is 0
`OutputFile`	`o`	output file
`PruneFreq`	`pf`	prune words with frequency below the specified value
`PruneRank`	`pr`	prune words with frequency rank above the specified value
`Size`	`s`	initial dictionary size; default is 10^6
`sort`		sort dictionary by frequency; default is false
`TestFile`	`t`	compute OOV rates on the specified test corpus

6.1.b Extraction of a dictionary

To extract the dictionary from a given a text and store it in a file, run the following command:

```
$> dict -i=train.txt.se -o=train.dict -f=true
```

The input text can be also generated on the fly by passing a command as value of the parameter`InputFile` ; in this case the single or double quotation marks are required.

```
$> dict -i="cat train.txt | add-start-end.sh" -o=train.dict -f=true
```

For some applications like speech recognition, it can be useful to limit the LM dictionary. You can obtain such a pruned list either by means of the parameter `PruneRank`, which only stores the top frequent, let us say, 10K words:

```
$> dict -i=train.txt.se -o=train.dict.pr10k -pr=10000
```

or by means of the parameter `PruneFreq`, which only store the terms occurring more than a given amount of times, let us say, 5:

```
$> dict -i=train.txt.se -o=train.dict.pf5 -pf=5
```

The two pruning strategies can be combined.

6.1.c Dictionary growth curve

`dict` can display the distribution of the terms according to their frequency in a text or in a pre-computed dictionary. This facility is enabled by the parameter `Curve`; the maximum frequency taken into account is specified by the parameter `CurveSize`.

```
dict -i=train.dict -c=yes -cs=50
```

The output looks as follows

```
Dict size: 7893
*************** DICTIONARY GROWTH CURVE ***************
Freq  Entries  Percent
>0    7893     100.00%
>1    4880      61.83%
>2    3721      47.14%
>3    2990      37.88%
...
>47   271        3.43%
>48   264        3.34%
>49   258        3.27%
*******************************************************
```

Each row of the table reports, given the value in the first column, the amount of terms (second column) having at least the given frequency (first column), and its percentage (third column) with respect to the total amount of entries.

6.1.d Out-of-vocabulary rate statistics

`dict` can display the distribution of the terms according to their frequency in a text or in a pre-computed dictionary; the maximum frequency taken into account is specified by the parameter `CurveSize`.

```
$> dict -i=train.dict -t=test.txt.se -cs=50
```

The output looks as follows

```
Dict size: 7893
Words of test: 1009
*************** OOV RATE STATISTICS ***************
Freq  OOV_Entries  OOV_Rate
<1    119          11.79%
<2    151          14.97%
```

```
<3    191          18.93%
...
<48   457          45.29%
<49   457          45.29%
<50   457          45.29%
*****************************************************
```

Each row of the table reports, given the value in the first column, the out-of-vocabulary rate on the test set, assuming to prune the dictionary at the given frequency. In other words, 191 (18.93%) of the running terms in the test set has a frequency smaller than 3 in the dictionary.

6.2 ngt

`ngt` is the command which copes with the n-gram counts.

- It extracts the n-gram counts and stores into a n-gram table.
- It prunes n-gram table.
- It merges n-gram tables.
- It transforms n-gram table formats.

A new n-gram table for the limited dictionary can be computed with `ngt` by specifying the sub-dictionary:

```
$> ngt -i=train.www -sd=top10k -n=3 -o=train.10k.www -b=yes
```

The command replaces all words outside top10K with the special out-of-vocabulary symbol `_unk_`.`dict` is the command which copes with the dictionaries.
Another useful feature of ngt is the merging of two n-gram tables. Assume that we have split our training corpus into files `text-a` and file `text-b` and have computed n-gram tables for both files, we can merge them with the option `-aug`:

```
$> ngt -i="gunzip -c text-a.gz" -n=3 -o=text-a.www -b=yes
$> ngt -i="gunzip -c text-b.gz" -n=3 -o=text-b.www -b=yes
$> ngt -i=text-a.www -aug=text-b.www -n=3 -o=text.www -b=yes
```

Warning: Note that if the concatenation of `text-a.gz` and `text-b.gz` is equal to `train.gz` the resulting n-gram tables `text.www` and `train.www` can slightly differ. This happens because during the construction of each single n-gram table few n-grams are automatically added to make it consistent for further computation.

6.3 tlm

Language models have to cope with out-of-vocabulary words, that is internally represented with the word class `_unk_`. In order to compare perplexity of LMs having different vocabulary size it is better to define a conventional dictionary size, or dictionary upper bound size, trough the parameter (`-dub`). In the following example, we compare the perplexity of the full vocabulary LM against the perplexity of the LM estimated over the more frequent 10K-words. In our comparison, we assume a dictionary upper bound of one million words.

```
$>tlm -tr=train.10k.www -n=3 -lm=wb -te=test -dub=1000000
  n=49984 LP=342160.8721 PP=939.5565162 OVVRate=0.07666453265

$>tlm -tr=train.www -n=3 -lm=wb -te=test -dub=1000000
  n=49984 LP=336276.7842 PP=835.2144716 OVVRate=0.05007602433
```

The large difference in perplexity between the two LMs is explained by the significantly higher OOV rate of the 10K-word LM.
N-gram LMs generally apply frequency smoothing techniques, and combine smoothed frequencies according to two main schemes: interpolation and back-off. The toolkit assumes interpolation as default. The back-off scheme is computationally more costly but often provides better performance. It can be activated with the option `-bo=yes`, e.g.:

```
$>tlm -tr=train.10k.www -n=3 -lm=wb -te=test -dub=1000000 -bo=yes
  n=49984 LP=337278.3227 PP=852.1186066 OVVRate=0.07666453265
```

This toolkit implements several frequency smoothing methods, which are specified by the parameter `-lm`. Three methods are particularly recommended:

a) **Modified shift-beta**, also known as "improved kneser-ney smoothing". This smoothing scheme gives top performance when training data is not very sparse but it is more time and memory consuming during the estimation phase:

```
$>tlm -tr=train.www -n=3 -lm=msb -te=test -dub=1000000 -bo=yes
  n=49984 LP=321877.3411 PP=626.1609806 OVVRate=0.05007602433
```

b) **Witten Bell smoothing**. This is an excellent smoothing method which works well in every data condition and is much less time and memory consuming:

```
$> tlm -tr=train.www -n=3 -lm=wb -te=test -dub=1000000  -bo=yes
  n=49984 LP=331577.2279 PP=760.2652095 OVVRate=0.05007602433
```

c) **Shift-beta smoothing**. This smoothing method is a simpler and cheaper version of the Modified shift-beta method and works sometimes better than Witten-Bell method:

```
$> tlm -tr=train.www -n=3 -lm=sb -te=test -dub=1000000  -bo=yes
  n=49984 LP=334724.5032 PP=809.6750442 OVVRate=0.05007602433
```

Moreover, the non linear smoothing parameter β can be specified with the option `-beta`:

```
$> tlm -tr=train.www -n=3 -lm=sb -beta=0.001 -te=test -dub=1000000
       -bo=yes
  n=49984 LP=449339.8282 PP=8019.836058 OVVRate=0.05007602433
```

This could be helpful in case we need to use language models with very limited frequency smoothing.

Limited Vocabulary

Using an n-gram table with a fixed or limited dictionary will cause some performance degradation, as LM smoothing statistics result slightly distorted. A valid alternative is to estimate the LM on the full dictionary of the training corpus and to use a limited dictionary just when saving the LM on a file. This can be achieved with the option `-d` (or `-dictionary`):

```
$> tlm -tr=train.www -n=3 -lm=msb -bo=y -te=test -o=train.lm -d=top10k
```

6.4 compile-lm

6.5 interpolate-lm

6.6 prune-lm

6.7 quantize-lm

7 IRSTLM functions

7.1 LM Adaptation

Language model adaptation can be applied when little training data is given for the task at hand, but much more data from other less related sources is available. `tlm` supports two adaptation methods.

7.2 Minimum Discriminative Information Adaptation

MDI adaptation is used when domain related data is very little but enough to estimate a unigram LM. Basically, the n-gram probs of a general purpose (background) LM are scaled so that they match the target unigram distribution.
Relevant parameters:

- `-ar=value`: the adaptation `rate`, a real number ranging from 0 (=no adaptation) to 1 (=strong adaptation).
- `-ad=file`: the adaptation file, either a text or a unigram table.
- `-ao=y`: open vocabulary mode, which must be set if the adaptation file might contain new words to be added to the basic dictionary.

As an example, we apply MDI adaptation on the "adapt" file:

```
$> tlm -tr=train.www -lm=wb -n=3 -te=test -dub=1000000 -ad=adapt -ar=0.8 -ao=yes
   n=49984 LP=326327.8053 PP=684.470312 OVVRate=0.04193341869
```

Warning: modified shift-beta smoothing cannot be applied in open vocabulary mode (`-ao=yes`). If this is the case, you should either change smoothing method or simply add the adaptation text to the background LM (use `-aug` parameter of `ngt`). In general, this solution should provide better performance.

```
$> ngt -i=train.www -aug=adapt -o=train-adapt.www -n=3 -b=yes
$> tlm -tr=train-adapt.www -lm=msb -n=3 -te=test -dub=1000000 -ad=adapt -ar=0.8
  n=49984 LP=312276.1746 PP=516.7311396 OVVRate=0.04193341869
```

7.3 Mixture Adaptation

Mixture adaptation is useful when you have enough training data to estimate a bigram or trigram LM and you also have data collections from other domains.
Relevant parameters:

- `-lm=mix` : specifies mixture smoothing method
- `-slmi=<filename>`: specifies filename with information about LMs to combine.

In the example directory, the file `sublmi` contains the following lines:

```
2
-slm=msb -str=adapt -sp=0
-slm=msb -str=train.www -sp=0
```

This means that we use train a mixture model on the `adapt` data set and combine it with the train data. For each data set the desired smoothing method is specified (disregard the parameter `-sp`). The file used for adaptation is the one in FIRST position.

```
$> tlm -tr=train.www -lm=mix -slmi=sublm -n=3 -te=test -dub=1000000
  n=49984 LP=307199.3273 PP=466.8244383 OVVRate=0.04193341869
```

Warning: for computational reasons it is expected that the n-gram table specified by `-tr` contains AT LEAST the n-grams of the last table specified in the slmi file, i.e. `train.www` in the example. Faster computations are achieved by putting the largest dataset as the last sub-model in the list and the union of all data sets as training file.
It is also IMPORTANT that a large `-dub` value is specified so that probabilities of sub-LMs can be correctly computed in case of out-of-vocabulary words.

7.4 Estimating Gigantic LMs

LM estimation starts with the collection of n-grams and their frequency counters. Then, smoothing parameters are estimated for each n-gram level; infrequent n-grams are possibly pruned and, finally, a LM file is created containing n-grams with probabilities and back-off weights. This procedure can be very demanding in terms of memory and time if it applied on huge corpora. We provide here a way to split LM training into smaller and independent steps, that can be easily distributed among independent processes. The procedure relies on a training scripts that makes little use of computer RAM and implements the Witten-Bell smoothing method in an exact way.
Before starting, let us create a working directory under `examples`, as many files will be created:

```
$> mkdir stat
```

The script to generate the LM is:

```
$> build-lm.sh -i "gunzip -c train.gz" -n 3  -o train.ilm.gz -k 5
```

where the available options are:

```
-i    Input training file e.g. 'gunzip -c train.gz'
-o    Output gzipped LM, e.g. lm.gz
-k    Number of splits (default 5)
-n    Order of language model (default 3)
-t    Directory for temporary files (default ./stat)
-p    Prune singleton n-grams (default false)
-s    Smoothing: witten-bell (default), kneser-ney, improved-kneser-ney
-b    Include sentence boundary n-grams (optional)
-d    Define subdictionary for n-grams (optional)
-v    Verbose
```

The script splits the estimation procedure into 5 distinct jobs, that are explained in the following section. There are other options that can be used. We recommend for instance to use pruning of singletons to get smaller LM files. Notice that `build-lm.sh` produces a LM file `train.ilm.gz` that is NOT in the final ARPA format, but in an intermediate format called `iARPA`, that is recognized by the `compile-lm` command and by the Moses SMT decoder running with **IRSTLM Toolkit**. To convert the file into the standard ARPA format you can use the command:

```
$> compile-lm train.ilm.gz --text yes train.lm
```

this will create the proper ARPA file `lm-final`. To create a gzipped file you might also use:

```
$> compile-lm train.ilm.gz --text yes /dev/stdout | gzip -c > train.lm.gz
```

In the following sections, we will discuss on LM file formats, on compiling LMs into a more compact and efficient binary format, and on querying LMs.

7.5 Estimating a LM with a Partial Dictionary

A sub-dictionary can be defined by just taking words occurring more than 5 times (`-pf=5`) and at most the top frequent 5000 words (`-pr=5000`):

```
$>dict -i="gunzip -c train.gz" -o=sdict -pr=5000 -pf=5
```

The LM can be restricted to the defined sub-dictionary with the command `build-lm.sh` by using the option `-d`:

```
$> build-lm.sh -i "gunzip -c train.gz" -n 3  -o  train.ilm.gz -k 5 -p -d sdict
```

Notice that all words outside the sub-dictionary will be mapped into the `<unk>` class, the probability of which will be directly estimated from the corpus statistics. A preferable alternative to this approach is to estimate a large LM and then to filter it according to a list of words (see Filtering a LM).

7.6 LM Pruning

Large LMs files can be pruned in a smart way by means of the command `prune-lm` that removes n-grams for which resorting to the back-off results in a small loss. **IRSTLM Toolkit** implements a method similar to the Weighted Difference Method described in the paper *Scalable Backoff Language Models* by Seymore and Rosenfeld.
The syntax is as follows:

```
$> prune-lm --threshold=1e-6,1e-6  train.lm.gz  train.plm
```

Thresholds for each n-gram level, up from 2-grams, are based on empirical evidence. Threshold zero results in no pruning. If less thresholds are specified, the right most is applied to the higher levels. Hence, in the above example we could have just specified one threshold, namely `--threshold=1e-6`. The effect of pruning is shown in the following messages of `prune-lm`:

```
1-grams: reading 15059 entries
2-grams: reading 142684 entries
3-grams: reading 293685 entries
done
OOV code is 15058
OOV code is 15058
pruning LM with thresholds:
 1e-06 1e-06
savetxt: train.plm
save: 15059 1-grams
save: 138252 2-grams
save: 194194 3-grams
```

The saved LM table `train.plm` contains about 3% less bigrams, and 34% less trigrams. Notice that the output of prune-lm is an ARPA LM file, while the input can be either an ARPA or binary LM. In order to measure the loss in accuracy introduced by pruning, perplexity of the resulting LM can be computed (see below).

Warning: the possible quantization should be performed after pruning.

Warning: **IRSTLM Toolkit** does not provide a reliable probability for the special 1-gram composed by the "sentence start symbol" (<s>) , because none should ever ask for it. However, this pruning method requires the computation of the probability of this 1-gram. Hence, (only) in this case the probability of this special 1-gram is arbitrarily set to 1.

7.7 LM Quantization

A language model file in ARPA format, created with the IRST LM toolkit or with other tools, can be quantized and stored in a compact data structure, called language model table. Quantization can be performed by the command:

```
$> quantize-lm  train.lm train.qlm
```

which generates the quantized version `train.qlm` that encodes all probabilities and back-off weights in 8 bits. The output is a modified ARPA format, called qARPA. Notice that quantized LMs reduce memory consumptions at the cost of some loss in performance. Moreover, probabilities of quantized LMs are not supposed to be properly normalized.

7.8 LM Compilation

LMs in ARPA, iARPA, and qARPA format can be stored in a compact binary table through the command:

```
$> compile-lm train.lm train.blm
```

which generates the binary file `train.blm` that can be quickly loaded in memory. If the LM is really very large, `compile-lm` can avoid to create the binary LM directly in memory through the option `-memmap 1`, which exploits the *Memory Mapping* mechanism in order to work as much as possible on disk rather than in RAM.

```
$> compile-lm --memmap 1 train.lm train.blm
```

This option clearly pays a fee in terms of speed, but is often the only way to proceed. It is also recommended that the hard disk for the LM storage belongs to the computer on which the compilation is performed.
Notice that most of the functionalities of `compile-lm` (see below) apply to binary and quantized models. By default, the command uses the directory "/tmp" for storing intermediate results. For huge LMs, the temporary files can grow dramatically causing a "disk full" system error. It is possible to explicitly set the directory used for temporary computation through the parameter "–tmpdir".

```
$> compile-lm --tmpdir=<mytmpdir> train.lm train.blm
```

7.9 Inverted order of ngrams

For a faster access, the ngrams can be stored in inverted order with the following two commands:

```
$> sort-lm.pl -inv -ilm train.lm -olm train.inv.lm
$> compile-lm train.inv.lm train.inv.blm --invert yes
```

Warning: The following pipeline is no more allowed!!

```
$> cat train.lm | sort-lm.pl -inv | \
   compile-lm /dev/stdin train.inv.blm --invert yes
```

7.10 LM Interpolation

We provide a convenient tool to estimate mixtures of LMs that have been already created in one of the available formats. The tool permits to estimate interpolation weights through the EM algorithm, to compute the perplexity, and to query the interpolated LM.
Data used in those examples can be found in the directory `example/interpolateLM/`, which represents the relative path for all the parameters of the referred commands.
Interpolated LMs are defined by a configuration file in the following format:

```
3
0.3 lm-file1
0.3 lm-file2
0.4 lm-file3
```

The first number indicates the number of LMs to be interpolated, then each LM is specified by its weight and its file (either in ARPA or binary format). Notice that you can interpolate LMs with different orders

Given an initial configuration file `lmlist.init` (with arbitrary weights), new weights can be estimated through Expectation-Maximization on some text sample `test` by running the command:

```
$> interpolate-lm lmlist.init --learn test
```

New weights will be written in the updated configuration file, called by default `lmlist.init.out`. You can also specify the name of the updated configuration file as follows:

```
$> interpolate-lm lmlist.init --learn test lmlist.final
```

Similarly to `compile-lm`, interpolated LMs can be queried through the option `--score`

```
$> interpolate-lm lmlist.final --score yes < test
```

and can return the perplexity of a given input text ("`--eval text-file`"), optionally at sentence level by enabling the option "`--sentence yes`",

```
$> interpolate-lm lmlist.final --eval test
$> interpolate-lm lmlist.final --eval test --sentence yes
```

If there are binary LMs in the list, `interpolate-lm` can avoid to load them in memory through the memory mapping option `-memmap 1`.
The full list of options is:

```
--learn text-file   learn optimal interpolation for text-file
--order n           order of n-grams used in --learn (optional)
--eval text-file    compute perplexity on text-file
--dub dict-size     dictionary upper bound (default 10^7)
--score [yes|no]    compute log-probs of n-grams from stdin
--debug [1-3]       verbose output for --eval option (see compile-lm)
--sentence [yes|no] (compute perplexity at sentence level (identified
                    through the end symbol)
--memmap 1          use memory map to read a binary LM
```

7.11 Filtering a LM

A large LM can be filtered according to a word list through the command:

```
$> compile-lm  train.lm --filter list filtered.lm
```

The resulting LM will only contain n-grams inside the provided list of words, with the exception of the 1-gram level, which by default is preserved identical to the original LM. This behavior can be changed by setting the option `--keepunigrams no`. LM filtering can be useful once very large LMs can be specialized in advance to work on a particular portion of language. If the original LM is in binary format and is very large, `compile-lm` can avoid to load it in memory, through the memory mapping option `-memmap 1`.

8 Parallel Computation

This package provides facilities to build a gigantic LM in parallel in order to reduce computation time. The script implementing this feature is based on the `SUN Grid Engine` software[4].
To apply the parallel computation run the following script (instead of `build-lm.sh`):

```
$> build-lm-qsub.sh -i "gunzip -c train.gz" -n 3  -o train.ilm.gz -k 5
```

Besides the options of `build-lm.sh`, parameters for the SGE manager can be provided through the following one:

```
-q      parameters for qsub, e.g. "-q <queue>", "-l <resources>"
```

The script performs the same *split-and-merge* policy described in Section 7.4, but some computation is performed in parallel (instead of sequentially) distributing the tasks on several machines.

[4] http://www.sun.com/software/gridware

9 IRSTLM Interface

LMs are useful when they can be queried through another application in order to compute perplexity scores or n-gram probabilities. **IRSTLM Toolkit** provides two possible interfaces:

- at the command level, through `compile-lm`
- at the c++ library level, mainly through methods of the class `lmtable`

In the following, we will only focus on the command level interface. Details about the c++ library interface will be provided in a future version of this manual.

9.1 Perplexity Computation

Assume we have estimated and saved the following LM:

```
$> tlm -tr=train.www -n=3 -lm=wb -te=test -o=train.lm -ps=no
 n=49984 LP=308057.0419 PP=474.9041687 OVVRate=0.05007602433
```

To compute the perplexity directly from the LM on disk, we can use the command:

```
$> compile-lm train.lm  --eval test
 %% Nw=49984 PP=1064.40 PPwp=589.50 Nbo=38071 Noov=2503 OOV=5.01%
```

Notice that `PPwp` reports the contribution of OOV words to the perplexity. Each OOV word is indeed penalized by dividing the LM probability of the `unk` word by the quantity

`DictionaryUpperBound - SizeOfDictionary`

The OOV penalty can be modify by changing the `DictionaryUpperBound` with the parameter `--dub` (whose default value is set to 10^7).

The perplexity of the pruned LM can be computed with the command:

```
$> compile-lm train.plm --eval test --dub 10000000
%% Nw=49984 PP=1019.69 PPwp=564.73 Nbo=39907 Noov=2503 OOV=5.01%
```

Interestingly, a slightly better value is obtained which could be explained by the fact that pruning has removed many unfrequent trigrams and has redistributed their probabilities over more frequent bigrams.
Notice that `PPwp` reports the perplexity with a fixed dictionary upper-bound of 10 million words. Indeed:

```
$> tlm -tr=train.www -n=3 -lm=wb -te=test -o=train.lm -ps=no -dub=10000000
n=49984 LP=348396.8632 PP=1064.401254 OVVRate=0.05007602433
```

Again, if the LM is in binary format and is very large, `compile-lm` can avoid to load it in memory, through the memory mapping option `-memmap 1`.

By enabling the option "`--sentence yes`", `compile-lm` computes perplexity and related figures (OOV rate, number of backoffs, etc.) for each input sentence. The end of a sentence is identified by a given symbol (`</s>` by default).

```
$> compile-lm train.plm --eval test --dub 10000000 --sentence yes

%% sent_Nw=1 sent_PP=23.22 sent_PPwp=0.00 sent_Nbo=0 sent_Noov=0 sent_OOV=0.00%
%% sent_Nw=8 sent_PP=7489.50 sent_PPwp=7356.27 sent_Nbo=7 sent_Noov=2 sent_OOV=25.00%
%% sent_Nw=9 sent_PP=1231.44 sent_PPwp=0.00 sent_Nbo=14 sent_Noov=0 sent_OOV=0.00%
%% sent_Nw=6 sent_PP=27759.10 sent_PPwp=25867.42 sent_Nbo=19 sent_Noov=1 sent_OOV=16.67%
.....
%% sent_Nw=5 sent_PP=378.38 sent_PPwp=0.00 sent_Nbo=39893 sent_Noov=0 sent_OOV=0.00%
%% sent_Nw=15 sent_PP=4300.44 sent_PPwp=2831.89 sent_Nbo=39907 sent_Noov=1 sent_OOV=6.67%
%% Nw=49984 PP=1019.69 PPwp=564.73 Nbo=39907 Noov=2503 OOV=5.01%
```

Finally, tracing information with the `--eval` option are shown by setting debug levels from 1 to 4 (`--debug`):

1. reports the back-off level for each word
2. adds the log-prob
3. adds the back-off weight
4. check if probabilities sum up to 1.

9.2 Probability Computations

Word-by-word log-probabilities can be computed as well from standard input with the command:

```
$> compile-lm train.lm --score yes < test

> </s>  1 p= NULL
> <s> <unk>     1 p= NULL
> <s> <unk> of  1 p= -3.530047e+00 bo= 2
> <unk> of the  1 p= -1.250668e+00 bo= 1
> of the senate 1 p= -1.170901e+01 bo= 1
> the senate (  1 p= -5.457265e+00 bo= 2
> senate ( <unk>        1 p= -2.166440e+01 bo= 2
....
....
```

the command reports the currently observed n-gram, including `_unk_` words, a dummy constant frequency 1, the log-probability of the n-gram, and the number of back-offs performed by the LM.

Warning: All cross-sentence n-grams are skipped. The 1-grams with the sentence start symbol are also skipped. In a n-grams all words before the sentence start symbol are removed. For n-grams, whose size is smaller than the LM order, probability is not computed, but a `NULL` value is returned.

10 Regression Tests

A Reference Material

The following books contain basic introductions to statistical language modeling:

- *Spoken Dialogues with Computers*, by Renato DeMori, chapter 7.
- *Speech and Language Processing*, by Dan Jurafsky and Jim Martin, chapter 6.
- *Foundations of Statistical Natural Language Processing*, by C. Manning and H. Schuetze.
- *Statistical Methods for Speech Recognition*, by Frederick Jelinek.
- *Spoken Language Processing*, by Huang, Acero and Hon.

The following papers describe the IRST LM toolkit:

- Efficient data structures to handle huge language models:

  Marcello Federico and Mauro Cettolo, *Efficient Handling of N-gram Language Models for Statistical Machine Translation*, In Proc. of the Second Workshop on Statistical Machine Translation, pp. 88–95, ACL, Prague, Czech Republic, 2007.

- Language Model quantization:

  Marcello Federico and Nicola Bertoldi, *How Many Bits Are Needed To Store Probabilities for Phrase-Based Translation?*, In Proc. of the Workshop on Statistical Machine Translation. pp. 94-101, NAACL, New York City, NY, 2006.

- Language Model adaptation with mixtures:

  Marcello Federico and Nicola Bertoldi, *Broadcast news LM adaptation over time*, Computer Speech and Language. 18(4): pp. 417-435, October, 2004.

- Language Model adaptation with MDI:

  Marcello Federico, *Efficient LM Adaptation through MDI Estimation*. In Proc. of Eurospeech, Budapest, Hungary, 1999.

B Release Notes

If present, the index in parentheses refers to the revision number in IRSTLM repository (until 5.60.02) or SourceForge repository (from 5.60.03).

B.1 Version 3.2

- Quantization of probabilities
- Efficient run-time data structure for LM querying
- Dismissal of MT output format

B.2 Version 4.2

- Distinction between open source and internal Irstlm tools
- More memory efficient versions of binarization and quantization commands
- Memory mapping of run-time LM
- Scripts and data structures for the estimation and handling of gigantic LMs
- Integration of **IRSTLM Toolkit** into Moses Decoder

B.3 Version 5.00

- Fixed bug in the documentation
- General script `build-lm.sh` for the estimation of large LMs.
- Management of iARPA file format.
- Bug fixes
- Estimation of LM over a partial dictionary.

B.4 Version 5.04

- Extended documentation with ShiftBeta smoothing.
- Smoothing parameter of ShiftBeta can be set manually.
- Robust handling for smoothing parameters of ModifiedShiftBeta.
- Fixed probability checks in TLM.
- Parallel estimation of gigantic LM through SGE
- Better management of sub dictionary with build-lm.sh
- Minor bug fixes

B.5 Version 5.05

- (Optional) computation of OOV penalty in terms of single OOV word instead of OOV class
- Extended use of OOV penalty to the standard input LM scores of compile-lm.
- Minor bug fixes

B.6 Version 5.10

- Extended ngt to compute statistics for approximated Kneser-Ney smoothing
- New implementation of approximated Kneser-Ney smoothing method
- Minor bug fixes
- More to be added here ....

B.7 Version 5.20

- Improved tracing of back-offs
- Added command prune-lm (thanks to Fabio Brugnara)
- Extended lprob function to supply back-off weight/level information
- Improved back-off handling of OOV words with quantized LM
- Added more debug modalities to compile-lm
- Fixed minor bugs in regression tests
- Updated documentation

B.8 Version 5.21

- Addition of interpolate-lm
- Added LM filtering to compile-lm
- Improved regression tests
- Integration of interpolated LMs in Moses
- Extended tests on compilers and platforms
- Improved documentation with website

B.9 Version 5.22

- Use of AutoConf/AutoMake toolkit compilation and installation

B.10 Version 5.30

- Support for a safe management of LMs with a total amount of n-grams larger than 250 million
- Use of a new parameter to specify a directory for temporary computation because the default (”/tmp”) could be too small
- Improved a safer method of concatenation of gzipped sub lms
- Improved management of log files

B.11 Version 5.40

- Merging of internal-only tlm code into the public version
- Updated documentation into the public version
- Included documentation into the public version

B.12 Version 5.50

- **5.50.01**
  - Binary saving directly with tlm
  - Speed improvement through
  - Caching of probability and states of n-grams in the LM interface
  - Storing of n-grams in inverted order
- **5.50.02**
  - Optional creation of documentation
  - Improved documentation
  - Optional computation of the perplexity at sentence-level

B.13 Version 5.60

- **5.60.01**
  - Handling of class/chunk LMs with both compile-lm and interpolate-lm
  - Improved pruning strategy to handle with sentence-start symbols
  - Improved documentation and examples
- **5.60.02**
  - Code cleanup
- **5.60.03 (r404)**
  - Xcode project
  - import from IRSTLM repository (revision 4263)

B.14 Version 5.70

- **5.70.01 (r454)**
  - Class-based LM
  - Added improved-kneser-ney smoothing for lm-build-qsub.sh
  - Enabled different singleton pruning policy for each submodel of mixture LM
  - Enabled the possibility to load an existing LM up to a specific level smaller than the actual LM order
  - Code tracing
  - Handling of error codes
  - Handling of long filenames and parameters
  - Improved parallel code compilation
  - Improved documentation and examples
- **5.70.02 (r469)**
  - Code optimization
  - Common interface for all LM types

B.15 Version 5.80

- **5.80.01 (r501)**
  - Facility to *beautify* source code
  - Re-activation of filtering on a sub-dictionary
  - Code optimization related to LM dumping
  - Transformation of scripts into Bourne shell scripts
- **5.80.03 (r579)**
  - Data selection tool
  - Handling of precision upper- and lower-bounds by means of constants.
  - Improved of Xcode project
  - Improved code compilation
  - Improved documentation
  - Improved handling of help
  - Facility to check whether IRSTLM is compile with or without caching
- **5.80.05 (r642)**
  - Introduction of *namespace irstlm*
  - Code optimization
  - Code compliant with OsX Maverick

  - Support for Redis output format to ngt
  - Support CRC16 algorithm
  - Improved plsa command and regression test
  - Improved handling of tracing
  - Improved handling of help
  - Improved handling of error messages

- **5.80.06 (r647)**
  - Improved code compilation

- **5.80.07**
  - Changes to LM smoothing types; removed Good-Turing, added Approximated Modified Shift-Beta, renaming
  - Added an additional pruning method, based on level-dependent pruning frequency
  - Improved code compilation
  - Improved documentation
  - Code cleanup
  - Added support for long names of parameters
  - Improved output format

- **5.80.08**
  - Added functionality to score n-grams in isolation
  - Added level-based caches for storing prob, state, and statesize
  - Improved management of tracing assert
  - Improved management of tracing/assert/caching AutoConf compilation flags
  - Improved output format
  - Improved code compilation
  - Code cleanup