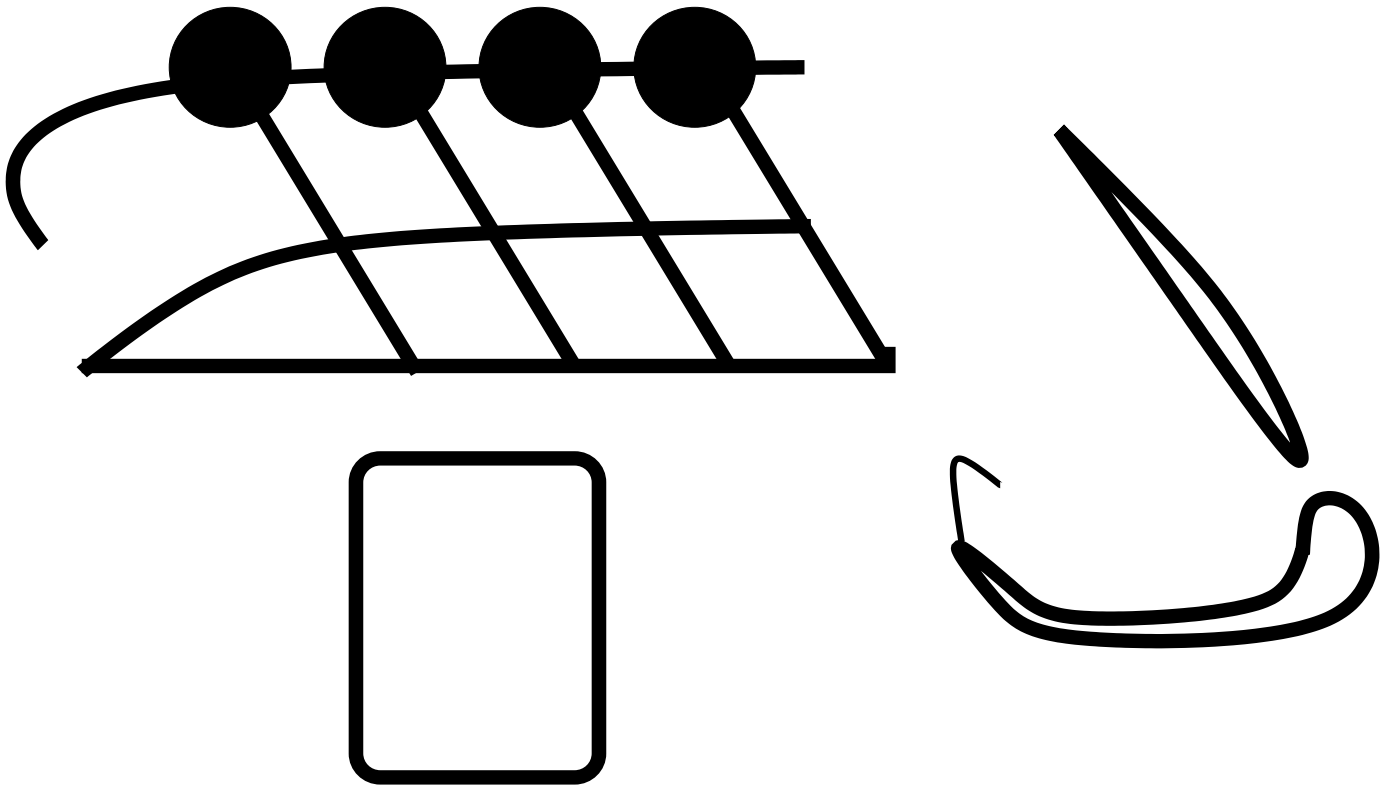

The Hieroglyphs

(Third Draft) The Hieroglyphs: Building Speech Applications Using CMU Sphinx and Related Resources

Arthur Chan Evandro Gouvêa Rita Singh
Mosur Ravishankar Ronald Rosenfeld Yitao Sun
David Huggins-Daines Mike Seltzer

March 11, 2007

Contents

List of Figures

List of Tables

Preface at Draft III

The open source project “CMU Sphinx” started by providing only the speech recognizers or devices for transforming speech to text (STT). By definition, they are devices which are capable of transforming acoustic waveforms into words.

Later on, the developers realized that building speech applications involved not only the recognizers (or technical the decoders) but also several interrelated resources. Therefore, the CMU Sphinx project now includes the acoustic model trainer, language model trainer, as well as publicly available acoustic and language models.

The comprehensiveness of the tools and components which CMU Sphinx-provides did not preclude the fact that building a speech application requires special techniques and knowledge. In the discussion forum of Sphinx, we observed many users are stumbled on trivial issues of building a system. It became obvious that the most challenging problem which many developers faced was still the access to documentation.

The difficulty to find documented information made the use of Sphinx restricted to users with long year of experience in programming and speech recognition research. If one could find any information, they are distributed in several web pages of experienced users and developers. Using Sphinx seemed like playing jigsaw puzzle, the user always need to gather a lot of information before building a real system.

Some of The Sphinx Developers , including me, were soon convinced that the best way to change this situation was to introduce a document that recorded most, if not all, knowledge about using Sphinx and its related resources to build speech applications.

When I told this idea to mentors and friends of mine in CMU, many of them shook their heads. It was a great task but also a tremendous one. Nevertheless, Prof. Alex Rudnicky, my mentor at that time, told me that it was a good idea. Half jokingly, he appointed me the “Editor” of this document. The true meaning of the title really means I cut and pasted a

lot of documents in this tiny notes.

That's why I started to work on this about 2 years ago. It starts with two chapters and the progress was very slow.

There are a lot of material to cover, so later on I decided to expand it to a thirteen chapters book. Dr. Evandro Gouvêa and David Huggins Daines have given me a lot help, At the end of 2005 summer. About 10 chapters of the book had the first draft.

That was the point I decided to give this document a name. Following the CMU Sphinx Group's tradition of naming projects after mythological or ancient items, we named the current effort "The Hieroglyphs".

The name "The Hieroglyphs" symbolizes an important aspect of learning any speech recognition toolkit: it requires extensive knowledge. It is not an exaggeration that experts in any speech recognition system are usually experts in several seemingly unrelated fields, such as phonetics, linguistics, signal processing, pattern recognition, computer algorithms and artificial intelligence. The knowledge required might be as deep as the knowledge required to understand hieroglyphs.

It was then a long year of work in funded project(s) for me. I was very exhausted towards the end of them and seeking for a new change. That's why I decided to leave CMU and join a small company named Scanscout. Project "The Hieroglyphs" has been abandoned for 8 months.

Professionalism made me restarted the writing again. Mainly because "The Hieroglyphs" is probably the only CMU project which is not wrapped up well. Sphinx 3.X now has X equals to 6. SphinxTrain with the lead of Dave and Evandro have became more robust. Even CMU-Cambridge LM Toolkit was re-licenced in version 3. These softwares are probably my favorite things, I was glad to see their progress.

I also witnessed the emergence of Sphinx 4 mainly by the Sun Microsystem Team and PocketSphinx mainly by David Huggins-Daines. Many believe that they are future of the sphinx projects. For this I absolutely agreed.

Progress doesn't seem to happen in "The Hieroglyphs". My life-time problem in writing is that I would like to share a lot of things and sometimes too much. I used to hope that "The Hieroglyphs" will also grow from being a mere manual for Sphinx into an introduction to speech recognition systems, a book that allows both beginners and experts to learn the essential facts of using Sphinx and its related tools. But that could be too much, in a way it makes me feel two exhausted to finish it.

However, I finally realized that we could stil share parts of the knowledge to the user first. Therefore, I decided to trim some part of the docu-

ment and put it as the third draft. There are several parts including,

- The history section.
- The introduction of speech recognition.
- Some portion in the recipe section
- All of the sections in acoustic modeling
- development and research using Sphinx
- FAQ.

They will be rewritten sometime in the future drafts.

Compare to the first draft, the only draft I have released in the public, the third draft looks more like a book. The reader will also find that it is more informative. Some chapters could be essentialy in working on speech recognition systems. For example, the chapter about the front-end is very useful, I usually used it as a reference myself.

I hope that the information we collect in last few years could be shared with the users, even if they are not sufficiently polished in a scholarly manner. I would like to say my knowledge in speech recognition, or even Sphinx, are very limited. Therefore, I hope the reader of this book could understand the drafty nature of the book and provide feedback to improve it in future.

My thanks go to other authors of this book, especially Dave and Evandro. I would also like to thank Prof. Alex Rudnicky, Prof. Richard Stern and Prof Alan Black for the encouragement.

Note in Second Draft: The writing process and current status

The Writing Process

There are vast amount of knowledge of using Sphinx, the fact that multiple versions of Sphinx(en)exist also steepen the users learning curve. Therefore, we chose to use one single recognizer as a starting point of the material. The goal of the first edition is therefore to create a single document that a developer could use

Sphinx 3 + SphinxTrain + CMU-Cambridge LM Toolkit

to create an application. We chose to use Sphinx 3 mainly because it was the active branch which CMU was maintaining at the time. We also found that the combination is more coherent because all three tools are written in same programming language.

Even with the more limited scope, the editor still found that there are 15 chapters of material (2 of them are appendices) are required to be completed. It is fortunate that the original of all previous documents of each part of Sphinx are willing to give permission to either reprint or re-use their material. It makes the writing process much easier.

As at Jun, 2005, the first drafts of 9 chapters have already fully completed.

As at the end of Aug, 2005, the first draft of Chapter 6, 7, 8 are also completed.

Here are descriptions of the completed chapters, material they references and their current status.

In Chapter 1, the license and version history of Sphinx 2 , Sphinx 3 , Sphinx 4 , SphinxTrain and CMU-Cambridge LM Toolkit will be described. This chapter is completed.

Chapter 2 describes the history of development of Sphinx 2 , Sphinx 3 , Sphinx 4 , SphinxTrain and CMU-Cambridge LM Toolkit . The difference between different versions of Sphinx and a brief guideline on how to choose which recognizer to use. The part of history of Sphinx largely referenced Dr Rita Singh's "History of Sphinx" document. This chapter is completed.

Chapter 3 describes how to install software of Sphinx. This part is completed.

Chapter 5 provides information of how Sphinx carries out feature extraction and dynamic coefficient computation. This part is largely referenced from Dr. Mike Seltzer's document of "Sphinx 3 front end Specification". This content of this part is almost completed but required proofreading. We also feel that it is necessary to add details such as CMN, variance normalization and AGC into this chapter.

Chapter 6 provides a general description of the software package of our training process. It is almost completed. However, some details such as description of Sphinx 3 file format and how the headers look like are still missing.

Chapter 7 provides a detail look on how acoustic model could be training for both continuous HMM and semi-continous HMM. It is largely adapted from Dr. Rita Singh's web page for "instruction of training". This

part is largely completed. However, it still lacks of example that could help users to solve a problem when they have a real-life situation. In the next revision, we will focus on solving these problems.

Chapter 8 provides a manual for using CMU-Cambridge language model toolkit. The material is largely adapted from Dr. Philip Clarkson's web page on the instruction.

Chapter 9 provides detail description on how the search of Sphinx 3 works. This part reference Dr. Mosur Ravishankar's tutorial on "Sphinx 3 " and my own presentations on "From Sphinx 3.3 to Sphinx 3 .4" and "From Sphinx 3 .4 to Sphinx 3 .5". This part is largely completed. However, important detail description on how lexical tree, n-gram search and context-dependent triphones work on **decode** and **decode_anytopo** is still missing at the current stage.

Chapter 10 discusses how to use the speaker adaptation facilities began to be provided by SphinxTrain and Sphinx 3 from 2004. This part is mostly completed. However, we might still want to add detail on how better alignment on the transcription could provide larger gain in speaker adaptation.

Appendix A provides all command line information for all tools in Sphinx 3 , SphinxTrain and CMU-Cambridge LM Toolkit . This part relies on automatic generation of the tex file from the SphinxTrain and Sphinx 3 's command line. The SphinxTrain tool part is completed but need to be updated. The command-line information SphinxTrain script, Sphinx 3 and CMU-Cambridge LM Toolkit are not yet completed.

Two chapters are ended up removed in the second draft

This includes "Development using Sphinx" which discuss how to make use of live-mode APIs of Sphinx 3 to develop speech application. The backbone of this chapter is completed. It was largely reference Yitao Sun's code documentation that was automatically generated by doc++. However, in 2005, the Sphinx developer decide to switch from doc++ to doxygen. Therefore, we will expect to replace the current documentation very soon.

Also "FAQ of using Sphinx" which lists all frequently asked questions that were collected by Dr. Rita Singh. This part largely copied from the page from cmusphinx.org which is first collected by Dr. Rita Singh and is no maintained by Dr. Evandro Gouvêa. This part is completed. However, updates will be necessary when new questions come up in the developers mind.

Focus of writing of second draft

The editor is currently focused on working chapter 4 which provides detail description on how to use the Sphinx System from scratch. After completion of it, the document will be released in the editor's web page again.

Furture Plan of this Document

The coexistence of Sphinx 2 , Sphinx 3 and Sphinx 4 and their extensive usage in industry are perhaps a sign of the creativity of workers of speech recognition. We hope that this trend could be continued. Each Sphinx represent a spectrum of speech application that developers or researchers want to build. Therefore, it is fair to explain their usage in details in this document as well.

As we mentioned, the writing of edition one of Hieroglyph will only cover Sphinx 3 which happens to be the active branch. After its completion, we will extend it to cover Sphinx 2 and Sphinx 4 's usage as well.

For the users

This manual was created to aide the users in dealing with difficulties of using Sphinx and its related tools. It is reasonable for us to include so much material in the first place. Serious knowledge of building an HMM-based speech recognizer could hardly be written using only 10 pages. Making a serious speech recognition is a very difficult task. Even with advanced tools like Sphinx, it could still require high-level of skill and perhaps more importantly tremendous amount of knowledege as well as patience.

We recommend the users without basic knowledge of HMM to at least go through the background material to find standard text book of the topic. At this point, some effort has been made to write and introduction of speech recognition. We hope that it could be completed in near future.

For users who have basic knowledge of HMM, we will recommend them to browse through Chapter 2 and then follow the instruction as given in Chapter 4. Chapter 2 has a nice interview what resource the CMU Sphinx Group to build a speech recognition system. Whereas Chapter 4 can serve as a reference to build the first system.

For users who already have knowledge in Sphinx, we will recommend

you to keep track of the new release of this document and the release and backward compatibility notes we augment in every minor version updates.

We are also well aware that the document itself could have mistakes and we are always welcome for users' feedback on how to improve it. Just send your opinions to the current maintainers, Arthur Chan, David Huggins-Daines, Evandro Gouvêa, Alan Black and Kevin Lenzo.

Arthur Chan

The Editor (who only cuts and pastes)

Part I

Before You Start

Chapter 1

License and use of CMU Sphinx, SphinxTrain and CMU-Cambridge LM Toolkit

Author: *Arthur Chan*, Editor: *Arthur Chan*

- CMU Sphinx 2 , CMU Sphinx 3 and SphinxTrain have been developed by researchers and developers at Carnegie Mell University (CMU).
- CMU Sphinx 4 have been co-developed by research and developers at Carnegie Mellon University (CMU), Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT)..
- CMU-Cambridge LM Toolkit was co-developed by researchers at CMU and Cambridge University.

If you are interested in any CMU Sphinx, your right regarding the code and binaries in Sphinx 2 , Sphinx 3 , Sphinx 4 , SphinxTrain and CMU-Cambridge LM Toolkit must be your major concern. Here is something you may want to know:

1. **Copying of Sphinx 2 , Sphinx 3 , Sphinx 4 and SphinxTrain** The code and binaries of all CMU Sphinx(en)are *free* for commercial/non-

commercial users with or without modification. For more details, please check the license terms below.

2. **Copying of CMU-Cambridge LM Toolkit** The code of the CMU-Cambridge LM Toolkit is made available for research purposes only. It may be redistributed freely for this purpose.

3. **Academic license** We are not using GPL-ed license. Hence, you are not obligated to distribute your source code.

4. **Warranty**

   Although the license of Sphinx 2 , Sphinx 3 , Sphinx 4 and Sphinx-Train source code are free and the CMU-Cambridge LM Toolkit 's source code is free for research purpose, we do not provide any warranty and support at all. However, the maintainers of the Sphinx Project are always willing to discuss and help developers to build applications using any Sphinx(en). As of March 11, 2007, the maintainers are Arthur Chan, David Huggins-Daines, Evandro Gouvêa, Alan Black and Kevin Lenzo, you are always welcome to talk to any one of them.

Sections 1.1, refssec:sphinxivlicense and 1.3 contain the full license terms for the software.

1.1 License Agreement of Sphinx 2 , Sphinx 3 and SphinxTrain Source Code

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

This work was supported in part by funding from the Defense Advanced Research Projects Agency and the National Science Foundation of the United States of America, and the CMU Sphinx Speech Consortium.

THIS SOFTWARE IS PROVIDED BY CARNEGIE MELLON UNIVERSITY "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY NOR ITS EMPLOYEES BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.2 License Agreement of CMU Sphinx 4

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Original authors' names are not deleted.

4. The authors' names are not used to endorse or promote products derived from this software without specific prior written permission.

This work was supported in part by funding from the Defense Advanced Research Projects Agency and the National Science Foundation of the United States of America, the CMU Sphinx Speech Consortium, and Sun Microsystems, Inc.

CARNEGIE MELLON UNIVERSITY, SUN MICROSYSTEMS, INC., MITSUBISHI ELECTRONIC RESEARCH LABORATORIES AND THE CONTRIB-

UTORS TO THIS WORK DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY, SUN MICROSYSTEMS, INC., MITSUBISHI ELECTRONIC RESEARCH LABORATORIES NOR THE CONTRIBUTORS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

1.3 License Agreement of the CMU-Cambridge LM Toolkit

1.4 About The Hieroglyphs

We hope that this document, The Hieroglyphs, itself can be distributed so that its quality can be improved. Hence, the following permissions are granted to the users:

- Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

- Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.
- Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Carnegie Mellon Unversity.

1.5 How to contribute?

The Sphinx Project is developed for all researchers and developers over the world. We chose to use the X11 style license because this will allow users to have higher flexibility in incorporating the code.

To truely achieve this goal, your participation is necessary. Due the limitation of resources, The Sphinx system's performance still has a certain gap with the state-of-the-art system. Your participation is necessary to refine and perfect this software. We would also like to collect models for different languages and in different conditions. This will allow users/developers in the world to develop speech recognition systems easily.

The software is still under heavy development to know which part of the software requires change. You are visit the CMU Sphinxweb page:

http://cmusphinx.org

to learn about the open projects within the Sphinx group.

Please contact Arthur Chan, David Huggins-Daines, Evandro Gouvêa, Alan Black and Kevin Lenzoif you are interested in contributing to the Sphinx speech recognition system.

1.6 Version History of Sphinx 2

1.6.1 Sphinx 2 .5 Release Notes

This release (Sphinx 2 -0.5) includes support for language model defined as a Finite State Grammar (FSG), as well as bug fixes.

The FSG implementation supports:

- an application can chose at run time which FSG to use, from a list provided by the user (also supported with the statistical LM).
- the FSG can be passed to the decoder as a data structure instead of as a file name.
- FSG automatically inserts multiple pronunciations and noise models, if provided.

Among the fixes, the executables and libraries created with MS Visual Studio are placed in a more sensible location.

It should work in several flavors of Unix (including Linux and Solaris), and Windows NT or later.

1.6.2 Sphinx 2 .4 Release Notes

CMU Sphinx is a speech recognition system.

This version includes a FreeBSD-style license (2 conditions), and numerous fixes. It should work for Linux, Solaris, various Unixes, and Windows under buth Visual Studio and cygwin

1.7 Version History of Sphinx 3

1.7.1 A note in current development of Sphinx 3.7

Sphinx 3.7 requires the use of a common library set called SphinxBase to compiled. Please consult the sphinx web site for instruction in compilation.

1.7.2 Sphinx 3 .6.3 Release Notes

Sphinx is a speaker-independent large vocabulary continuous speech recognizer released under a Berkeley-style license. It is also a collection of open source tools and resources that allows researchers and developers to build speech recognition systems.

You can download release 3.6.3 from here:

https://sourceforge.net/project/showfiles.php?group_id=1904

Sphinx 3.6.3

(Corresponds to Sourceforge SVN rev 5957.)

Sphinx 3.6.3 is a bug fix release.

(rev 5906) The hypseg generated by sphinx3_decode (mode 4) was ill-formed. This used to corrupt sphinx3_conf in confidence generation. This is now fixed.

(rev 5912) Frame length is now checked to make sure it is smaller than number of fft points. The number oof ffts is also checked to see if it is a power of 2.

For Developers (rev 5915) sphinx3_conf is now tested. PASS counts is now 98 in make check (rev 5955) wave2feat is not installed in sphinx3. Users and developers are recommended to use the SphinxTrain version instead.

1.7.3 Sphinx 3 .6.2 Release Notes

Sphinx is a speaker-independent large vocabulary continuous speech recognizer released under a Berkeley-style license. It is also a collection of open source tools and resources that allows researchers and developers to build speech recognition systems.

You can download release 3.6.2 from here:

https://sourceforge.net/project/showfiles.php?group_id=1904

Sphinx 3.6.2

(Corresponds to Sourceforge SVN rev 5842.)

Sphinx 3.6.2 fixed another issue related to 32bit LM; It now allows the use of LM with tg_seg size more than 0xffff. It also included a completed but un-tested version of full-covariance modeling.

Several bug fixes:

(rev 5814) When decode,decode_anytopo or lm_convert were used to read in LM with unigram less than 0xffff words, current code will assume the LM is in 16 bit mode. Now lm_convert will back-off to use 32 bit mode to read in. Users, except those who happened to read this message, will have no idea this happens.

(rev 5794) Fixed a compile warning which also happened to be a bug. A

16 bit LM word id was used in the 32 bit LM mode. This will cause problem when users put $< s >$ and $< /s >$ outside the range of 0xffff.

Developing Features:

(rev 5757:5761,5763,5771,5774:5776) Incorporated full covariance matrix as in acoustic modeling.

(rev 5766) Fixed front-end issues, now sphinx 3.6 could accept feature that was accepted in Sphinx 3.0 (or archive_s3)

For Developers:

(rev 5785, 5786, 5791 and 5792) sphinx3 is now indented using the scheme originally designed for PocketSphinx and sphinxbase.

(rev 5819) Several tests are added to support the fixes. Now PASSes count in make check is 97.

1.7.4 Sphinx 3 .6.1 Release Notes

Sphinx 3.6.1 (Corresponds to Sourceforge SVN rev 5753.)

Sphinx 3.6.1 (from rev 5660) enables 32-bit LM. It is tested under a real-life scenario and shown to provide better results. Also LTS code has bug so it was not truly enable in official sphinx 3.6

Several bug fixes:

- (rev 5621) Random crash with multiple LMs.
- (rev 5639) Don't allow ld_end_utt() to be called if ld_begin_utt() hasn't been called (this leads to crashing and badness).
- (rev 5663) Fixed FST output behavior.
- (rev 5711) Fixed underflow problem of ug if backoff weight is too small ($<<$ 0). LTS now truely enabled.
- (rev 5730) Fixed Visual Studio 8.0 compilation
- (rev 5732) Fixed astar memory problem, tested in a 64000 words scenario.

Developing features:

- (rev 5738) full covariance matrices
- (rev 5744) Enable fast GMM computation for align.

1.7.5 Sphinx 3 .6 Release Notes

Sphinx 3.6 official release included all changes one found in Sphinx 3.6 RC I.

From 3.6 RC I to 3.6 official:

New Features:

- Added support for sphinx 2-style semi-continuous HMM in Sphinx 3.
- Added sphinx3_continuous which performs on-line decoding in both windows and linux platforms.
- Synchronized the frontend with Sphinx2, adding implementation of VTLN. (i.e. -warp_type = inverse_linear, piewise_linear, affine)

Changes:

- Prefix "sphinx3_" has been added to programs align, allphone, astar, dag, decode, decode_anytopo, ep to avoid confusion in some unix systems.

Bug Fixes:

1459402 Serious memory relocation is fixed.

- In RCI, -dither was not properly implemented, this has been fixed.

Known Problem:

- When the model contains nan, there will be abnormal output of the result. At this point, this issue is resolved in SphinxTrain

Sphinx 3.6 Release Candidate I

The corresponding SphinxTrain's tag is SPHINX3_6_CMU_INTERNAL_RELEASE One can check out the matching SphinxTrain of the sphinx3.6 release by command,

svn co https://svn.sourceforge.net/svnroot/cmusphinx/tags/SPHINX3_6_CMU_I

A Summary of Sphinx 3.6 RC I

Sphinx 3.6 is a gently refactored version of Sphinx 3.5. Our programming is defensive and we only aim at further consolidation and unification our code-bases in Sphinx 3.

Despite our programming is defensive, there are still several interesting and new features could be found in this release. Their details could be found in the "New Feature" section below. Here is a brief summary:

1. Further speed-up of CIGMMS in the 4 level GMM Computation Schemes (4LGC)
2. Multiple regression classes an MAP adaptation in SphinxTrain
3. Better support in using LM in Sphinx 3.X.
4. FSG search is now supported. This is adapted from Sphinx 2.
5. Support of full triphone search in flat lexicon search.
6. Some support of different character sets other than of Sphinx 3.X. Models in multiple languages are now tested in Sphinx 3.X.

We hope you enjoy this release candidate. In future, we will continue to improve the quality of CMU Sphinx and CMU Sphinx's related software.

New Features

-Speaker Adaptation: Multiple regression class (phoneme-based) is now supported.

-GMM Computation

1. Improvements of CIGMMs is now incorporated.
   (a) One could specify the upper limit of the number of CD senones to be computed in each frame by specifying -maxcdsenpf.
   (b) The best Gaussian index (BGI) are not stored and could be used as a mechanism to speed up GMM computation
   (c) tightening-factor (-tighten_factor) is introduced to smooth between fix naive down-sampling technique and CI-GMMS.
2. Support of SCHMM and FCHMM
   (a) decode will fully support computation of SCHMM.

-Language Model

1. reading an LM in ARPA text format is now supported. Users now have an option to by-pass the use of lm3g2dmp.
2. live decoding API now supports switching of language models.
3. full support of class-based LM. See also the Bug fixes section
4. lm_convert is introduced. lm_convert supersede the functionalities of lm3g2dmp. Not only could lm_convert convert an LM from TXT format to DMP format. It could also do the reverse.

-Search

This part will detail the change we make in different search algorithms. In 3.6, collection of algorithms could all be used under a single executable decode. decode_anytopo is still reserved for backward compatibility purpose. Decode now support three modes of search.

1. Mode 2 (FSG): (Adapted from Sphinx 2) FSG search.
2. Mode 3 (FLAT): Flat-lexicon search. (The original search in decode_anytopo in 3.X (X $<$ 6))
3. Mode 4 (TREE): Tree-lexicion search. (The original search in decode in 3.X (x $<$6)

Some of these functionalities will only be applicable in one particular search. We will mark them with FSG, FLAT and TREE.

1. One could now turn off -bt_wsil to control whether silence should be used as the ending word. (FLAT, TREE)
2. In FLAT, full triphones could be used instead of multiplexed triphones.
3. FSG is a new added routine in 3.6 which is adapted from Sphinx 2.5

-Frontend

1. -dither is now supported in live_pretend and live_decode, the initial seed could always be set the command -seed. (Jerry Wolf will be very happy about this feature.)

-Miscellaneous

1. One could turn on built-in letter-to-sound rule in dict.c by using -lts_mismatch.
2. current Sphinx 3.6 is tested to work on setup of English, Chinese Mandarin, French and English.
3. changes in allphone: allphone can now generate a match and a match-seg just like decode* recognizers.

Bug fixes

1. Miscellaneous memory leak fixed in the tree search (mode 4)

2. Initialization class-based LM routine use to switch the order of word insertion penalty and language model weight. This is now fixed.

3. Assertion generated vithist.c is now turn into an error message. Instead of causing the whole program stopped. The decoding will just fail for that sentnece. We suspect that this is the problem which caused possible wipe out of memory in Sphinx 3.4 & 3.5

4. Number of CI phones could now be at most 32767 (instead of 127)

1236322 : str2words special character bugs.

Behavior Changes

1. Endpointer (ep) now used computation of s3 log.

2. Multi-stream GMM computation will not truncate the pdf to 8 bit anymore. This will avoid confusion of programmer. However

3. Except in allphone and align, When .cont. is used in -senmgau, the code will automatically turn to use fast GMM computation routine. To make sure the multiple-stream GMM computation will be in effect, one need to specify .s3cont.

4. executable dag hadn't accounted for the language weight. Now this issue is fixed.

5. (See Bug fixes also) decode will now return error message when vithist was fed in with history -1. Instead of asserting the problem. The recognizer will dump Warning message. Usually that means beam widths need to increase.

Functions still under test

1. Encoding conversion in lm_convert.

2. LIUM contribution: LM could now represented as AT&T fsm format.

Known bugs

1. Confidence estimation, the computations of forward and backward posterior probability have mismatch

2. In allphone, sometimes the scores generated in the matchseg file will have very low scores.

3. Regression test on second-stage search still have bugs.

Corresponding changes in SphinxTrain

Please note that SphinxTrain is distributed as a separate package and you can get it by: svn co https://svn.sourceforge.net/svnroot/cmusphinx/tags/SPHINX3_6

1. Support for generation of MAP, multiple-class MLLR.
2. Support for BBI tree generation

1.7.6 Sphinx 3 .5 Release Notes

New features of this release:

- Live-mode APIs are stable and are now officially released.
- Windows version of live-mode decoder based on the new APIS is released.
- Live-mode decoder simulator (livepretend) based on the new APIS is still under development. We expect it to be stable in Sphinx 3.6.
- Mean transformation-based adaptation is now supported.
- The feature extraction library of sphinx3 is now EXACTLY the same as SphinxTrain
- Four of the s3.0 (flat decoder) tools are now incorporated in Sphinx 3 .x.
  - align - a time aligner
  - astar - an N-best genertor
  - allphone - a phoneme recognizer
  - dag - a best-path finder for a lattice

SphinxTrain has been tagged to match the release 0.5 of Sphinx 3 . One can retrieve the matching SphinxTrain by the command:

```
 $ cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/cmusphinx/
co -r SPHINX3_5_CMU_INTERNAL_RELEASE SphinxTrain
```

New features of this release:

- The feature extraction libraries of SphinxTrain is now EXACTLY the same as sphinx3.

- New tools introduced
  - **mllr_solve**, given the EM posterior probabilities of the mean and regression class definition, this routine estimates regression matrices for each class.
  - **mllr_transform**, given a transformation matrix, this tools applies the transformation to the mean vector of the models.
- The command line interface for all SphinxTrain tools are unified.

1.7.7 Sphinx 3.4 Release Notes

1. Sphinx 3.4 has a re-written the Gaussian computation routine. The following techniques are implemented.
   - Naive and model-based down-sampling such that some of the frames are computed.
   - Context-independent phone-based Gaussian mixture model Selection.
   - Two Gaussian selection algorithms including Vector Quantizater-based Gaussian selection and Sub-vector quantizer-based Gaussian selection.
   - Sub-vector quantization.
2. Phoneme lookahead based on three different types of heuristics are implemented.
3. Sphinx 3.3 front-end problem is fixed. Current front-end supports the original s3_1x39 feature and also the standard 1s_c_d_dd features.
4. Live mode recognizer's core dump problem is fixed.
5. sphinx-test and sphinx-simple should work as it is.
6. Visual C++ .dsw file is moved to upper most level of the code, a project is also built for the program decode in windows.
7. README and several compilation document is now included in the distribution.

1.8 Version History of Sphinx 4

Sphinx 4 1.0 beta

In this release, we have provided the following new features and improvements over the 0.1 alpha release:

- Confidence scoring
- Dynamic grammar support
- JSGF limitations removed
- Improved performance for large, perplex JSGF grammars
- Filler support for JSGF Grammars
- Out-of-grammar utterance rejection
- Narrow bandwidth acoustic model
- WSJ5K Language model
- More demonstration programs
- Better control over microphone selection
- Lots of bug fixes

Sphinx 4 is a state-of-the-art, speaker-independent, continuous speech recognition system written entirely in the Java programming language. It was created via a joint collaboration between the Sphinx group at Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT).

The design of Sphinx 4 is based on patterns that have emerged from the design of past systems as well as new requirements based on areas that researchers currently want to explore. To exercise this framework, and to provide researchers with a "research-ready" system, Sphinx 4 also includes several implementations of both simple and state-of-the-art techniques. The framework and the implementations are all freely available via open source under a very generous BSD-style license.

With the 1.0 beta release, you get the complete Sphinx 4 source tree along with several acoustic and language models capable of handling a

variety of tasks ranging from simple digit recognition to large vocabulary n-Gram recognition.

Because it is written entirely in the Java programming language, Sphinx 4 can run on a variety of platforms without requiring any special compilation or changes. We've tested Sphinx 4 on the following platforms with success: the Solaris 9 Operating System on the SPARC platform, Mac OS X 10.3.5, RedHat 9.0, Fedora Core 1, Microsoft Windows XP, and Microsoft Windows 2000.

Please give Sphinx 4 1.0 beta a try and post your questions, comments, and feedback to one of the CMU Sphinx Forums:

https://sourceforge.net/forum/?group_id=1904

We can also be reached at cmusphinx-contacts@lists.sourceforge.net.

Sphinx 4 -alpha

Sphinx 4 is a state-of-the-art, speaker-independent, continuous speech recognition system written entirely in the Java programming language. It was created via a joint collaboration between the Sphinx group at Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT).

The design of Sphinx 4 is based on patterns that have emerged from the design of past systems as well as new requirements based on areas that researchers currently want to explore. To exercise this framework, and to provide researchers with a "research-ready" system, Sphinx 4 also includes several implementations of both simple and state-of-the-art techniques. The framework and the implementations are all freely available via open source under a very generous BSD-style license.

With the Alpha 0.1 release, you get the complete Sphinx 4 source tree along with several acoustic and language models capable of handling a variety of tasks ranging from simple digit recognition to large vocabulary n-Gram recognition.

Because it is written entirely in the Java programming language, Sphinx 4 can run on a variety of platforms without requiring any special compilation or changes. We've tested Sphinx 4 on the following platforms with success: the Solaris 9 operating system on the SPARC platform, Mac OS X 10.3.3, RedHat 9.0, Fedora Core 1, Microsoft Windows XP, and Microsoft Windows 2000.

Please give Sphinx 4 0.1 alpha a try and post your questions, comments, and feedback to one of the CMU SphinxForums:

https://sourceforge.net/forum/?group_id=1904

We can also be reached at cmusphinx-contacts@lists.sourceforge.net.

1.9 Version History of PocketSphinx

1.9.1 Pocket Sphinx 0.2

This is the first real release of PocketSphinx. PocketSphinx is a derivative of Sphinx-II designed for embedded and handheld systems.

See the release notes for more information.

This release corresponds to revision 5967 in Subversion.

1.9.2 Pocket Sphinx 0.2.1

This is a bug-fix release to make fixed-point computation work, which was inadvertently broken in the 0.2 release.

1.9.3 Pocket Sphinx 0.2.1

New versions of PocketSphinx and SphinxBase are now available.

This version of SphinxBase will be used as the base for the next version of Sphinx3. It includes many updates to the utility library, the feature extraction code, and the dynamic feature computation code.

This version of PocketSphinx is quite different from the last one. It is considerably smaller due to the elimination of large amounts of redundant and legacy code. It is also up to 20% faster in some cases. The command-line arguments have been synchronized with Sphinx3, and support for Sphinx2-format acoustic models has been removed.

Chapter 2

Introduction

Author: *Arthur Chan, Rita Singh*, Editor: *Arthur Chan*

In this chapter, we describe the history of Sphinx as a tool of speech recognition and give an overview of different software packages of Sphinx.

2.1 Why do I want to use CMU Sphinx?

If you are a developer, the major reason you want to use Sphinx is because Sphinx currently is the most complete archive of speech recognition tool. Sphinx is also released using a liberal license. You may aware that other very powerful toolkits, are also open-sourced. Unfortunately, some of their licenses disallows you to use their code in your project. It may also disallow you to use the code and distribute them later in your project.

Sphinx allows you to have full control of your source code as well Sphinx's code. You are free to use, modify and distribute your code with Sphinx's code. In another way, Sphinx is not using GPL, that means you don't have to make the code to be free and keep it close source. We believe this is the kind of flexibility developers really want.

Another aspects of the Sphinx Project is that based on Sphinx and its related resources, *and with enough knowledge* [1], you can build a lot of different types of applications. By speech applications, I mean things like dictation, automatic speech server. More importantly, the most common

[1] The knowledge of proper usage of a speech recognizer and produce high performance results will be as difficult as modifying the gcc compiler or the linux kernel source code.

denominator or all these applications, i.e the recognizer is open-sourced in CMU Sphinx. You will also get a lot of bundling resource such as dictionary, phone list from CMU's speech archive. This is something very important if you want to have full-control of you speech recognition system.

If you are researchers and your interest is in HMM-based speech recognition, there are several reasons why you may want to use Sphinx 3 and Sphinx 4 as your research tools.

First of all Sphinx(en)assumes that GMM computation and the search process are separate. The important consequence is that you could carry out two different types of research without conflicting each others. For example, you could try to device a new observation probability without touching the source code of the search routine. At the same time, you could also build a new search algorithms without thinking of the GMM computation. This gives you a better advantage in speech recognition reseach.

The second reason is in training. Most of the time when you were doing modeling research, what you would like to change is the algorithm of estimation. SphinxTrain 's Baum-Welch's algorithm solve this problem in two stages. It first dump the posterior probabitlites count in a separate file and can be easily readable by libraries of SphinxTrain . You can just play with these counts and there is no need to device you own Baum-Welch trainer. This advantage can greatly shorten your research time from weaks to dates.

The third reason is in the code-clairity, later recognizers such as Sphinx 3 and Sphinx 4 have put readability of the code as one of the most important design goal. Many researchers, not only they want to use the recognizer as a tool. They also want to change the code to suit there purpose. Sphinx 3 , for example, has only 4 layers of code from main() to the GMM computation. This become a good advantage for researchers who like to make changes in a speech recognizer.

2.2 The Hieroglyphs

Author: *Arthur Chan*, Editor: *Arthur Chan*

That is this manual.

As a continuous effort of improving every aspect of Sphinx, one of the goal of the team is to create high quality documentation for expert as well as general users. At 2004, this manual was created by Arthur Chan to

fulfill this goal.

The project name “Hieroglyph” symbolize two aspects of documentation of speech recognizer. First of all, the usage a speech recognizer and building sucessful speech recognition system required large amount of knowledege. The depth of the knowledge required by a researcher can be compared with how a type of character could endoded. The second aspect is that building a speech recognizer is sometimes very hard and that might required the developer or the researcher. It sometimes requires to pay effort that is similar to learn a type of Hieroglyphs.

The first draft of the Hieroglyph document aims at provide a comprehensive guide on how to use Sphinx 3 , SphinxTrain and CMU LM Toolkit to build a speech recognition system. Future plan will include the use of Sphinx 2 and Sphinx 4 .

As at 2005 Jun, 70% of the first draft of the hieroglyph document is completed and currently could be found in

http://www-2.cs.cmu.edu/∼archan/sphinxDoc.html

2.3 The Sphinx Developers

The Sphinx Developers are the people who maintain and continue to develop aspects of Sphinx Project . A large portion of them are from academic institutes such as CMU. There are also significant amount of them are from companies such as Sun, Compaq and Mitsubishi.

The development of CMU Sphinxis not exclusive to Sphinx Developers . Users are welcomed to contribute patches of code to Sphinx and patches are continuously incorporate to enrich and strengthen the current code base. If you are interested in developing Sphinx, you could contact Arthur Chan, David Huggins-Daines, Evandro Gouvêa, Alan Black and Kevin Lenzo.

2.4 Which $CMUSphinx$ should I use?

It depends on what kind of application you are trying to build. Here are couple of consideration you need to think about.

- **FSG and natural language** In one word, if you expect to build a system with highly structured dialog. Possibly an FSG system is suitable

for you. Currently, your choice will be either Sphinx 2 or Sphinx 4 .

If you want to build a system with more natural input and less constraints. Then a speech recognizer + robust parser system will be something more suitable to you. What you need to do is to use a speech recognizer to generate output string . Then use a parser to parse this possilby errorful output. The later can be well handled by parser such as Phoenix.

- **Programming Languages** Sphinx 2 and III is in C which is quite portable in general. Sphinx 4 is in Java. You may also need to decide whether you and your team can handle either of these two languages. Many people are religious in only one programming language. The different versions could give you a lot of flexibility to choose.

- **Accuracy and Speed**

  We didn't carry out very extensive benchmarking of different Sphinx(en). The following can only be regarded as a rough estimate According to our measurement. The accuracy numbers can be rougly summarized as

  $$\text{Sphinx 4} >= \text{Sphinx 3} >> \text{Sphinx 2}$$

  and in terms of speed,

  $$\text{Sphinx 4} >= \text{Sphinx 3} >= \text{Sphinx 2}$$

  That means Sphinx 2 is still the fastest recognizer we had in our inventory. However its accuracy is also the lowest. Sphinx 3 contains our best C's recognizer which can be configured as both fast and pretty accurate. Sphinx 4 is possibly the most all round recognizer we have and contrary to people's belief, it can actually be pretty fast. Partially, Java's development has come to a stage where it can handle heavy loaded computation.

- **Interfaces**

  Sphinx have several versions. Depends on your goal, your choice of recognizer will be different.

  In general, Sphinx 4 provides the best interface among all. It can be used by specifying a config.xml and command-line. This advantage can make your web developers are much easier task in general.

  Usage of Sphinx 2 and Sphinx 3 requires much more skill in scripting and in general understanding of the program. At the current stage,

they are stil regarded as systems for expert users. Though the Sphinx team is seeking to continuously improve the interface and try to make it easier to use for more users.

- **Platforms**

  Sphinx 2 , Sphinx 3 and Sphinx 4 are all highly platform-independent. However, the use of Java language in Sphinx 4 could potentially allow higher degree of platform independency.

  It is also of common interest that whether any version of the decoders could be used in an embedded platform. David Huggins-Daines created an embedded version (based on Sharp Zaurus) for Sphinx 2 . In that case, Sphinx 2 will perhaps the choise of the recognizer for embedded platform.

- **Research**

  We would recommend you to use the three recognizer because all of them have clean design. They all supports continuous HMMs which is currently a de-facto standard of HMM

  In the case of doing individual research. We have the following recommendation.

  For acoustic modeling and fast GMM computation , Sphinx 3 is actually are pretty good research platforms for speech recognition. By itself, it support SCHMM, CHMM and Sub-vector quantized HMM. They represent there different kinds of modeling technique for speech recognitions.

  For research in search, Sphinx 4 is a good candidate, because you can avoid annoying memory management and focus on implementation of the search algorithm.

  For speaker adaptation or general acoustic model research, you will find that you need to directly change SphinxTrain . The stage's estimation process will be a very nice process for you to use.

  A lot of research requires a very fast implementation of speech recognizer. We believe Sphinx 2 coulbe the choice.

  It is very unlucky that currently Sphinx's architecture makes feature extraction research pretty difficult to carry out. In future, we will try to change situation.

- **Documentation, support and commnity**

  As at 2005, all three Sphinx(en)(II, III and IV) are actively maintained by current maintainers of CMU Sphinx. Most of the developers have years of experience in large scale code development and industrial

experience. They usually provide very fast turn around to most of the problems.

Documentation in both Sphinx 3 and Sphinx 4 are probably more comprehensive than that of Sphinx 2 . Both Sphinx 3 and Sphinx 4 provided Javadoc style of code documentation as well very detailed documentation for the software itself.

For the community, one could imagine Sphinx 4 attracts more young, energetic researchers. Whereas, many users of Sphinx 2 and Sphinx 3 turns out to be experts with experience. Though as each of the software turns out to be more balanced, more users found that all three recognizers are pretty good choice in general.

2.5 Other Open Source Speech Recognition Project in CMU: Hephaestus

Author: *Arthur Chan*, Editor: *Arthur Chan*

Project Hephaestus of CMU is a project that tries to collect open source toolkits in speech and language processing. CMU Sphinxis one of the key tool in this project. A lot of effort is also put in toolkit such as speech synthesis, dialogue system and general speech resource. In this section, you will learn some of these efforts. They are important because just having the speech recognizer cannot build interesting application such as dialog system. Running a recognizer and a trainer also requires many relevant resource such as dictionary and training data. Therefore, it is very important to have some basic understanding of each of these components.

2.5.1 Festival

Festival is a free open-sourced text-to-speech synthesizer developed in Edinburgh Unversity. It is now mainly maintained by Prof. Alan Black in CMU. In the simplest case, you can feed in a string and festival will create the waveform and output through your speaker. The funtionalities of festival allow users to modify speech-synthesis in multiple levels. It is also extremely flexible and has very good programming interface.

Festival has very good documentation on-line at

- Festival's main page **www.cstr.ed.ac.uk/projects/festival/**

- FestVox **www.festival.org**

2.5.2 CMU Communicator Toolkit

CMU Communicator is a project that allows users to get flight information using speech. It is led by Prof. Alex Rudnicky in CMU. The whole system is available as a tool kit so the public can used it freely. CMU Communicator contains several of open source components including CMU Sphinx(the recognizer), Phoenix (the robust parser), Helios (post-processor that generate confidence), Rosetta(the language generation routine) and Festival (speech synthesizer). Therefore, this is one very interesting demonstration on how speech recognition system can be built in general.

Another important aspect of it is that CMU Communicator data is free to be used and its collection process allows you to use is data as freely as you like. CMU will allow charge you the distribution charge for the whole acquisition process.

For more detail about Communicator, you can go to

http://www.speech.cs.cmu.edu/Communicator/

For the information of the corpus, you can go to

http://www.speech.cs.cmu.edu/Communicator/Corpus/

for details.

2.5.3 CMUdict, CMUlex

CMUdict is the dictionary collected and built in CMU from its 15 years of research. CMUdict is the short-form of Carnegie Mellon University Pronouncing Dictionary. You can download it at

http://www.speech.cs.cmu.edu/cgi-bin/cmudict

2.5.4 Speech Databases

CMU provides few databases for research purpose. For example, AN4 and Communicator databas. Please contact Arthur Chan, David Huggins-Daines, Evandro Gouvêa, Alan Black and Kevin Lenzofor further detail.

2.5.5 OpenVXML and OpenSALT browser

Voice XML (VXML) and Speech Application Language Tags (SALT) are two markup languages that integrates speech services into markup languages. It becomes a very important toolkit to builld speech application for people who used to web development. You can find their information at

http://www.speech.cs.cmu.edu/openvxi/index.html

and

http://hap.speech.cs.cmu.edu/salt/

2.5.6 Miscellaneous

CMU's speech group's web page contains a lot of useful information for using, building and researching speech recognition systems. You can find its web site at

http://www.speech.cs.cmu.edu/

2.6 How do I get more help?

CMU Sphinxis currently maintained by CMU, a lot of developers/researchers from research institutions such as Sun Microsystem, Hewlett-Packard, Mitsubishi and MIT. Feel free the ask any questions in the discussion forum at

http://sourceforge.net/forum/?group_id=1904

Part II

Tutorial

Chapter 3

Software Installation

Author: *Arthur Chan*, Editor: *Arthur Chan* This chapter describe general installation procedure for multiple software toolkits that you will need to build a speech recognition system. We will assume Unix is the environment we will use. If you were a window user, we will recommend you to use cygwin. (

There are couple of components you need to download before we start. Make sure you have the following standard software that can be found in Unix. These are tools if you installed linux or cygwin in a windows machine. They usually come standard. They are also not so difficult to get.

- **tar and gzip** Software to decompress a file. If you are Windows users, WinZip, WinRar and 7-Zip will work for you.
- **gcc** C compiler, Sphinx 3 supports both gcc 3.2.2 and 2.95 to know the version, type gcc -v
- **make** A utility that automatically determine which part of the software need to be compiled given a makefile.

3.1 Sphinx 3 .X

Thee first software we will get is Sphinx 3 .X and you can download it at

http://sourceforge.net/projects/cmusphinx/

Go to the section of "Latest File Releases" and click sphinx3. Then choose the top most labelled with sphinx3-0.5.tgz is the one. [1]

We will go through the standard compilation Linux toolkit compilation process here (i.e. configure $->$ make $->$ make install).

1. **Configure the makefile**

   ```
   $ configure
   ```

2. **Compile the program**

   ```
   $ make
   ```

3. **Install the program**

   ```
   $ make install
   ```

By default the program will be installed in /usr/local/bin/, if you want to change the path of installation, you can type

```
$ configure--prefix=<yourpath>
```

in the configuring step.

3.2 SphinxTrain

If you want to get a copy of SphinxTrain , you need to use cvs to get it. Again, you can use the above configure$->$make$->$make install process to compile the code.

1. **Check out the source code** You could get the latest source code from Sourceforge by typing

   ```
   $ cvs -d:pserver:anonymous@cvs.sf.net:/cvsroot/cmusphinx co
     SphinxTrain
   ```

   When you were ask for password, just type enter. Then check out process will start.

   You will see something like this on the screen

   ```
   cvs checkout:  Updating SphinxTrain
   ```

[1] For some reason that caused by early interaction of sphinx developers and Sourceforge, sphinx3-0.X.tgz end up means sphinx3.X . It becomes a legacy problem that later maintainer just follow the old rules.

```
U SphinxTrain/COPYING
U SphinxTrain/Makefile
U SphinxTrain/README.tracing
U SphinxTrain/README.txt
U SphinxTrain/ReleaseNotes
U SphinxTrain/SphinxTrain.dsw
U SphinxTrain/config.guess
U SphinxTrain/config.sub
U SphinxTrain/configure
U SphinxTrain/configure.in
U SphinxTrain/install-sh
U SphinxTrain/missing
U SphinxTrain/mkinstalldirs
cvs checkout:  Updating SphinxTrain/bin
```

2. **Configure the makefile**

   ```
   $ configure
   ```

3. **Compile the program**

   ```
   $ make
   ```

3.3 CMU LM Toolkit

You can get the Toolkit at

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

1. **Configure the makefile**

   > configure

2. **Compile the program**

   > make

3.4 Other tools related to the recognizer

There are two extra tools which are not packaged in the above distributions. One is called cepview which can allow one to inspect the mfcc file. One is called lm3g2dmp which can transform standard text-based LM to a specific DMP file format that could be accepted by Sphinx 2 and Sphinx 3 . Both of these could be found in the share directory in Sourceforge. You could get the latest source code from Sourceforge by typing

```
 $ cvs co -d:pserver:anonymous@cvs.sf.net:/cvsroot/cmusphinx
co share
```

3.5 FAQ of the installation process

3.5.1 What if I am a user of Microsoft Windows?

Our recommendation for you is that you should consider to use cygwin.

What if I know only how to use Visual C++? So in that case I won't be able to use Sphinx ? Don't worry. Sphinx's developers have create a setup for Microsoft Visual C++. From our testing, it works for both VC 6.0 and VC.Net at 2003. Thought, with the constant change of Microsoft's changes of programmers user interface. It is very unlikely we can follow all the changes they made. So if you have a new compiler, you may need to make the setup itself.

For sphinx 3.X, you just need to go to click program.dsw and do a Build. Then you are done. The files can be found in ./bin/{Debug,Release}.

For SphinxTrain , you also just need to go to click program.dsw and do a build. The files can be found in ./bin/{Debug,Release}.

Very unfortunately, there is no VC setup for CMU LM Toolkit. May be we will try to fix this problem later.

3.5.2 How about other Windows compilers?

The software itself doesn't impose any restrictions on the compilers. The problems where most people are facing usually caused by lack of knowledge of the compiler itself.

The basic principle of compilation is simple. First create libraries for libaudio/, libs3decoder/ and libutil/ . Then use the compile the files in program with the libaries. The VC setup are basically done by the above procedure.

If you work out a setup to compile Sphinx, please kindly contact Arthur Chan, David Huggins-Daines, Evandro Gouvêa, Alan Black and Kevin Lenzo.

3.5.3 How about other Unix platforms?

Sphinx 3 and SphinxTrain are written in ANSI C and they are fairly portable to other Unix platforms. Though there is not much testing has been done for platform other than windows , Linux and Mac OS X. Though, we continuously try to make sure the code is as portable as possible

3.5.4 What if I can't build Sphinx 3 .X or SphinxTrain ?

Please report to the Sphinx Developers .

3.5.5 What if I am not a fan of C?

In that case, try Sphinx 4 , it is a better, cleaner system written by Java. Documentation of Sphinx 4 is not the purpose of this manual. You can check out its installation procedure at

http://cmusphinx.sourceforge.net/sphinx4/

As we mentioned in the Preface, the current manual will only focus on the usage of Sphinx 3 . Of course, don't be disappointed. :-). There will always be a new addition.

3.5.6 How do I get help for installation?

Try the Sphinx Forum in Sourceforge, there are multiple experts there to help you. The URL is

http://sourceforge.net/forum/?group_id=1904

Chapter 4

Building speech recognition system

Author: *Arthur Chan*, Editor: *Arthur Chan*

Sphinx 3 and SphinxTrain are good candidates for application programmer to build a speech recognition systems. By itself, it only provide a bare bone demonstation system (or **livedecode**) for speech recognition. This is far from what most user need. [1] This chapter provides a step by step guide to build a speech recognition system.

There are two approaches of building a speech recognition system.

The first way is to create a system by using the decoder of Sphinx Project and some open source models of Sphinx Project , this mode of development is easiest.

The knowledge required in building a speech recognition system using the first approach can be divided into two parts. One is to understand the specialty of using speech as an input of a program. For basic knowledge of HMM-based speech recognition, please consult Chapter 3 for further detail. The other part is to interface the speech recognizer with other programs. Both of this knowledge is not trivial. The first 5 steps will guide you through the procedure.

The second way and a prefered way to create a system is to build a full system including the acoustic model yourself. There are several reasons for doing so. One of them is despite the fact that the model you could

[1] Unfortunately, manuy users have misunderstanding that **livedecode** is a full system.

found in open source model archive is fairly good. They may not be the best for *your* situation. For example, if one would like to build a simple digit recognition system, models trained with multiple hours of data may have Gaussian distribution which is too "flat" that doesn't match with the special need of digit recognition. Training your own model can allow you to control the quality of the acoustic model. [2].

Experts of the second mode of development usually attain advanced knowledge in pattern recognition, thorough understanding in speech recognition and also savvy skills of manipulate different tools of speech recognition tools. It is one of the most satisfactory process in the world. Step 6 to Step 11 will briefly go through this process. In Chapter 6, a detail description of every sub-component of the process will be described.

After you tried the last two modes of development, you might be interested in 1) tuning the current system to make faster and more accurate or 2) understand what's going on internal to Sphinx 3 . Step 12 and Step 13 will describe it further.

A final note of this tutorial, there are several high quality tutorials already exists in the web. If you found that this tutorial alone couldn't solve all of your problems. Also try the following Evandro Gouvêa's tutorial

http://www-2.cs.cmu.edu/ robust/Tutorial/opensource.html

It consists of a full run-through tutorial which contains a full recipe of training using the AN4 or RM1 database. The resulting system would be a very simple evaluation system for these databases. You are highly recommended to run-through either this tutorial or the on-line tutorial if you are new to speech recognition or Sphinx.

4.1 Step 1: Design of a Speech Recognition System

In this section, I will list some concerns you might need to consider before you build a speech recognition system.

[2]Sometimes it might give you a job as well.

4.1.1 Do you really need speech?

Not every type of information is suitable for speech input. For example, digit string, is very good candidate for input by speech and it is not very natural to be input by other modes of input (such as touch pad).

What kind of application is suitable for speech? For example, dictation is well-known to be one application that speech is doing very well. Other applications can be simply summarized as variants of command-and-control type of applications. For example, if you just need several types of "speech shortcut" when you are using Windows. You might just want to have a few commands in your system and when each command was recognized, corresponding action would be taken by your system.

4.1.2 Is Your Requirement Feasible?

Using a HMM-based speech recognition, there are several types of input are very hard to be recognized. Some of them are intrinsic to the problem itself.

Let's first talk about some linguistic confusion, in general, if there is no linguistic constraints (e.g. N-gram LM or FSM), the recognition rate will usually very low. This situation is a very common and that occurred in research of conversational speech recognition and there is no easy solution to solve it. So if your goal is to recognize *everything* that can be spoken by *anyone*, Sphinx (include all Sphinx(en)) probably can't help you so.

There are two major factors in desigining your systems, one is *linguistic confusibility* and one is *acoustic confusability*. They are usually the cause of how speech system is poor in practice.

Linguistic Confusibility

If you rank order the difficulties of building a system. You could probably get a list [3]

1. Recognize everything that could be said by anyone in the world

[3] A very important note should be put here. Despite the author put 3000 words to be doable, this is just reflecting an expert level that beginning users could do. It is very important to realize, Sphinx, with one capable speech recognition expert can create arbitrary type of speech application with large amount of vocabulary ($>$ 20k).

The case we just mentioned. This is very hard. There are also effects of multiple languages and it is generally not easy to do.

2. Recognize everything that could be said by everyone who speaks the same language

   This is slightly easier but still we have no constraints.

3. Recognize what two persons are saying in a conversation

   This might be easier if we could impose a constraint by the topic. For example, if we know that the two persons are programmers and if we know they are likely to talk about programming all the time, then we could probably assume terms such as “program”, “function” could appear more.

4. Recognize what a person want to write in a letter

   This is also known as *dictation*. This is still quite hard but it is quite constrained because given a topic of the letter and the type of letters in that topic. For example, if you know you recognizer “Dear” in a marketing letter, usually, it will always follow by “customer”. Usually, the amount of words a person could use could be confined to less 60k words. That explain the success of a lot of commercial speech recognizers.

   Sphinx, by itself, does not forbid developers to build a dictation system. However, building dictation system usually required a lot of application tuning and training. This might be too hard to be described in this manual. Just to collect data may take you months to complete. This might not be suitable for this tutorial. As you might know, there are already several sucessfull commercial recognizers in the world. There are also several efforts tried to build an open source speech recognizer using Sphinx. Unfortunately, most of them are following incorrect approaches and using incorrect method. We could only wish someday we would write another guide on how to use Sphinx in dictation.

   After this point, things will start to be more easily doable by you. (Of course if you read this manual.)

5. Build a 3000 word dialogue system

   This is already down and proved to be very sucessful. For example CMU Communicator system is one good example. In the Communicator system

   http://www.speech.cs.cmu.edu/Communicator/

One could ask information for the itinery of flight. The destination and arrival location can be obtained. This system is very sucessful because the speech of this type is very constrained. [4]

Flight information is just one type of information. What the user might want to query could be things like weather information, TV listing, product information, etc. Most of the time, this query is very constrained. This will make your life easier.

6. Build a 1000-person person phone directory system with English names

   This is also known *speech directory system* and this is definitely doable and useful. I give this example because it could actually be harder than the task of building a 3000 word dialog system because it reflect the intrinsic difficulty of this problem, sometimes people name sounds very similar. Even human being has problems with them, we will talk about this more when discuss acoustic confusability. If there are more names to be recognized, the chances of names being confused will be higher. This is the nature of the problem. However, this is a problem which is doable by Sphinx and could be done by people who follow this manual.

7. Build a 100-command command and control system in English

   This is obviously another easy task, Sphinx could definitely help you. For example, take a look of the "Robosapien Dancing Machine" project.

   http://www.robodance.com/

   Command and control system is very general term for many type of application. You might want to use it for controlling your windows applications or you might want to use it to control a character in a game.

There are two notes we need to mentioned at this point. The first note is that it is important for you to realize there are different difficulties of building speech recognition system. Some of them is very hard to be tackled by a single person. You might need to form a team of people and try to deal with them with expert guidance. This will set your expectation to a correct level.

The second reason is that we want to show you size of vocabulary is not the only reason why a task is hard. Whether the tasks are difficult partially

[4]Notice that CMU Communicator was only using Sphinx 2 , the Sphinx 3 system was found to be 30% more accurate in later years.

depends on the perplexity of the language model, a quantity one would describe more in Chapter 9. Another factor is whether words confused with each other acoustically. This is briefly mentioned in the task "Build a 1000-person person phone directory system with English names" Let us go on with several other tasks in the next paragraph.

After you read this part, I urge you to test yourself this. Is recognition of an unconstrained 16 digit string harder? Or is recognition of a credit card (VISA or Mastercard) harder? [5]

Acoustic Confusibility

Have you been to China? And use Chinese romanized name to call your friends? If you have done it once, you will know that it is very hard. Let us consider another task, "Build a 1000 romanized Chinese name directory system". This task could be extremely hard because romanized Chinese names can be easily confusible with other. Consider my romanized Chinese name:

Chan Yu Chung

and one my friend's name

Cheng Yue Chun.

Try to read it aloud on the phone, you will know it might be quite impossible even for human ear to recognize them.

The same situation happens in alphabet recognition. It is generally very difficult to differentiate between some of them. For example, A, J and K could hard to differentiate. B, C, D, E, G and V, Z could be hard to differentiate. M and N is yet another pair.

Yet another example of acoustic confusability is so called phoneme recognition. For example on the phone, many human misrecognized the word "Arthur" to be "Althur". This is a case where the system should consider to reprompt the user for spelling or other means of confirmation.

Another possibility is even more subtle. For example, it might happened that the two things you like to recognize have the same name. For example, there are two people who has the same name. This is a natural situation where name cannot be the unique identifier for them. Obviously, a speech recognizer could not do too much about that.

Acoustic confusibility is one important concern to build a system. If

[5]The second case is easier because credit card number are usually generated by some special algorithm and can check whether the 16 digits were valid or not.

you put in a lot of confusible words, it will end up to give you a very poor recognition rate.

4.1.3 System Concern

Running a speech recognizer could take a lot of computation of your machine. The speed of the system mainly depends on the size of the vocabulary of the task and the number of mixture component in the Gaussian distribution (A model parameter, detail can be found in Chapter 6)

We will recommend you to have a PC which has comparable performance with a Pentium 1GHz or more to use Sphinx 3 . The recommended system is a Pentium 2GHz or more. This is a range where a lot of the system ($<= 10$ K Vocabularies) can be run under 1 times real time. That is to say, a 1 second of speech will be recognized using less than 1 second of computation time. However, a task with only 10-100 words will have no problem in running under 1 times real time if a 500MHz machine is used.

4.1.4 Our recommendation

We don't recommend first-time user to build system more than 10k because at that limit expert choice is required to tune the system down to 1 times real time. This estimates is valid at Jun 2005. The member of the Sphinx Developers alway continues to improve the speed of the performance. May be when you read this manual later, the speed of the recognizer will be fast enough for your system requirement.

A vocabulary which is too large has another problems, most of the time such a system is not suitable for the user's requirement. This recommendation will hopefully make sure that the user will consider their requirement carefully.

4.1.5 Summary

Design your speech recognition system is very important. Not everything is suitable to be input by speech. There are also some requirement of vocabulary could be impossible to implement. Some of them might not be possible even when human is a listener.

Speed is another concern that one might need to consider and they are usually related to the size of vocabulary.

4.2 Step 2: Design the Vocabulary

4.2.1 The System for This Tutorial

In this tutorial we will just a recognition system that allow users to recognize 5 key phrases, “spread sheet”, “word processor”, “calculator”, “web browser” and “movie player”. The user would can say these 5 words and the system will control the system and start the corresponding applications. This is a very simple system but we could learn a lot of interesting aspect of speech recognition design. We will continue to develop this simple system first. Later we will then expand this simple system.

4.2.2 Pronunciations of the Words

Let’s look at the words we chose again.

spread sheet

word processor

calculator

web browser

movie player

There are several interesting aspects for the choice, the first aspect is that the words have at least 2 or more syllables. This is chosen because speech recognizer could have hard time to recognizer keywords.

Another aspect is that the word is not easy to confuse with each others. One simple common sense test is try to read pairs of word in pair. This is not automatic method but it is a basic principle of how people will figure out confusability of English words.

As you already learnt in Chapter 4, Sphinx is a phone-based speech recognizer. Simply speaking, in the run time, the recognizer will create a giant Hidden Markov Model (HMM) and the search will try to search the best path given this HMM. Now we have the word list we want to recognize. We obviously need the phone-based representation of these words. Some people will just call this representation as “pronouciations”.

How do we come up with these pronunciations? This is actually quite simple. What happened if need the pronunciation of a word? The obvious answer is to look up this pronunciation from the dictionary.

Create the Phonemes Manually

So the first way to get this pronunciation is to lookup a dictionary to decide this phone set. The caveat of doing this is that you need to make sure phoneme symbols you were using is the same as the phoneme set used in the open source acoustic model. [6]

Another concern is that the dictionary you were using might not match the dialect of English (same for other languages). For example, British English, American English and Australian English are actually quite different. If you used a British English dictionary for American English, the end result may not be the most ideal one.

In our case, let us simplify our discussion by using a simple choice of dictionary. If you are using open source acoustic model that can be found, the following method might be easier. You could use free dictionary such as cmudict and you can download it at

http://www.speech.cs.cmu.edu/cgi-bin/cmudict

CMUdict is the short term of Carnegie Mellon University Pronouncing Dictionary. It is a machine-readable pronunciation dictionary for North American English that contains over 125,000 words and their transcriptions. Though the pronouciation is mainly for American English, it is actually a reasonable start.

The current version is CMUdict 0.6 and if you look inside this file (you could use the command cat to do it).

```
,COMMA K AA M AH
-DASH D AE SH
-HYPHEN HH AY F AH N
...ELLIPSIS IH L IH P S IH S
.DECIMAL D EH S AH M AH L
.DOT D AA T
.FULL-STOP F UH L S T AA P
.PERIOD P IH R IY AH D
.POINT P OY N T
/SLASH S L AE SH
```

[6] If they were not matched, Sphinx 3 's decode and decode_anytopo will inform the user (you) that they couldn't find the model. So the recognizer will be stopped in the initialization.

```
MALEFACTORS M AE L AH F AE K T ER Z
:COLON K OW L AH N
;SEMI-COLON S EH M IY K OW L AH N
;SEMI-COLON(2) S EH M IH K OW L AH N
?QUESTION-MARK K W EH S CH AH N M AA R K
A AH
A'S EY Z
A(2) EY
A. EY
A.'S EY Z
A.S EY Z
A42128 EY F AO R T UW W AH N T UW EY T
AAA T R IH P AH L EY
AABERG AA B ER G
AACHEN AA K AH N
AAKER AA K ER
AALSETH AA L S EH TH
AAMODT AA M AH T
AANCOR AA N K AO R
AARDEMA AA R D EH M AH
AARDVARK AA R D V AA R K
AARON EH R AH N
AARON'S EH R AH N Z
```

The first column at every entries is the word, the rest is the phoneme representation of the the word.

What does this phoneme means? The following is a table that shows their usage.

We might talk more about the prounciation table. The phoneme example table is useful when you start to change the pronunciations yourself.

Usage of automatic tools

After having cmudict, you can write a simple perl script to look up the the prounciation for each word. This is actually pretty simple so we skip how to do it at here. I will recommend you to try to do it because it is fun.

Many people has gone through this process and some of them are nice enough to share these resources. For example, one of them, Prof Alex Rudnicky and Mr. Kevin Lenzo from CMU have written a web page.

http://www.speech.cs.cmu.edu/cgi-bin/cmudict

(The same web page we used in downloading cmudict v0.6) You could just look a pronounciations up by typing the word in the edit box "Look up words or a sentence".

You will soon get the prounounciation of each word

```
SPREADSHEET S P R EH D SH IY T

WORD_PROCESSOR W ER D P R AA S EH S ER

CALCULATOR K AE L K Y AH L EY T ER

WEB_BROWSER W EH B B R AW Z ER

MOVIE_PLAYER M UW V IY P L EY ER
```

We will call this file **simple.dict**.

Yet another way to do it is to use the QUICK LM tool by Prof. Alex Rudnicky.

http://www.speech.cs.cmu.edu/tools/lmtool.html

You could first create a file like this

```
SPREADHSEET

WORD_PROCESSOR

CALCULATOR

WEB_BROWSER

MOVIE_PLAYER
```

And let us called the file to be **simple.txt**.

You could submit the file just by clicking "Compile Knowledge Base" button. You would see the phone set will be slightly different. You need to map AXR to ER, replaced AX to AH, replaced all TS by T and S, replaced all PD by P, replaced all DX by either D or T. This could be easily done by perl or sed.

Whether you use automatic pronunciation tools or manual way to create the prounciation. Do remember that you could freely change the dictionary yourself. Be careful though, the best pronuunciations are those that best fit for your users but not the official pronunciation of a word. The general phenomenon is that human display a very high varieties of pronounciation. The so called “official” pronounciations may not fit into your cases.

Here comes a question that you will always come up with. If we fiddle the system paratemers or settings, how could we know whether these changes give a positive effect to they system? The key is to benchmark the system every time a change is made. Testing the system by actually using it is not a bad idea. However, as HMM-based speech recognition system is statistical. A failure for once, or a failure out of ten times doesn’t tell you too much about the performance of a system. They are just not statistically significant enough to tell you much. Therefore, it is important to do some quantitative benchmarking.

One metric people usually used is word error rate (WER). If your change improve the WER in a large test set usually means the performance of the system really improves. We will discuss technique of benchmarking and the definition of WER more later in Step 6 of this Chapter

4.3 Step 3: Design the Grammar for a Speech Recognition System

Grammar play an important part of a speech recognition system. It governs how the recognizer will actually recognize in the hypothesis. In speech recognition, there are two prominent way to represent grammar. One is n-gram. The other is finite state machine. In this tutorial, we will use the n-gram because currently Sphinx 3 only supports N-gram. [7]. If you would like to use finite state machine, you might consult the documentation for Sphinx 2 and Sphinx 4

4.3.1 Coverage of a Grammar

Generally speaking, if the grammar is too constrained, it won't cover the words which would be said by the users. In another words, a lot of *Out-of-Vocabulary(OOV)* words will be resulted. One trivial way to solve this problem is to add new words. However, blindly increase the coverage without moderation can easily introduce unnecessary confusion to speech recognition. Therefore, careful decision and benchmarking must be made before adding new words to the grammar.

4.3.2 Mismatch between the Dictionary and the Language Model

One common confusion is how the speech recognizer behave when the vocabulary of words and the grammar does not match. In Sphinx recognizers, both the unigram of the language model and the dictionary will decide whether a word should exist in the search graph of speech recognition. The recognizer will try to look up the pronounciation (or the phoneme representation) of a word from the dictionary. If the recognizer failed to find a the pronounciation of a word, the word will not be included in the search.

In other words, the search graph will include the set of words that is an intersection of the vocabularies of dictionary and the unigram. [8]

[7] This is very likely to change because the implementation existed in Sphinx 2 has already been incorporated in Sphinx 3

[8] From this point of view, it is actually ok to include a dictionary with large amount

Table 4.1: Description of Phone Set in CMU Lex

Phoneme	Example	Translation
AA	odd	AA D
AE	at	AE T
AH	hut	HH AH T
AO	ought	AO T
AW	cow	K AW
AY	hide	HH AY D
B	be	B IY
CH	cheese	CH IY Z
D	dee	D IY
DH	thee	DH IY
EH	Ed	EH D
ER	hurt	HH ER T
EY	ate	EY T
F	fee	F IY
G	green	G R IY N
HH	he	HH IY
IH	it	IH T
IY	eat	IY T
JH	gee	JH IY
K	key	K IY
L	lee	L IY
M	me	M IY
N	knee	N IY
NG	ping	P IH NG
OW	oat	OW T
Y	toy	T OY
P	pee	P IY
R	read	R IY D

4.3.3 Intepretation of ARPA N-gram format

The first thing if you used an n-gram based speech recognizer is the ARPA n-gram format. Hence it is quite important to learn how to interprete the format. The following is one example of it.

[Editor Notes: This example was not shown correctly.]

```
 ....  Comment Sections Skip here ....
data
ngram 1=2001
ngram 2=29445
ngram 3=75360
1-grams:
-2.0960 <UNK> -0.5198
-2.8716 'BOUT -0.5978
-4.6672 'CAUSE -0.3765
-5.1065 'FRISCO -0.2588
-4.4781 'HEAD -1.1010
-4.3863 'KAY -0.1962
-4.5624 'ROUND -0.5975
2-grams:
.
.
3-grams:
.
.
end
```

---

of words. Though, in the actual implementation, the dictionary is actually read into the memory. Hencse the size of the dictionary will be constrained by the system memory the user have.

The ̈data ̈section

This section show the number of each n-grams. In Sphinx, up to trigram is supported.

The Ẍ-gram: ̈section

It specify the log probabilities and the back-off weights for the N-gram.

The end section

This ends the file.

4.3.4 Grammar for this tutorial

For this tutorial, we just create a very simple language model for our purpose.

[Editor Notes: This file is not shown correctly]

```
data
ngram 1=107
ngram 2=1
1-grams:
-2.0253 </s> -99.0000
-99.0000 <s> 0.0000
-0.6990 SPREADSHEET 0.0000
-0.6990 WORD_PROCESSOR 0.0000
-0.6990 CALCULATOR 0.0000
-0.6990 WEB_BROWSER 0.0000
-0.6990 MOVIE_PLAYER 0.0000
2-grams:
0.0000 <s> </s>
0.0000 SPREADSHEET SPREADSHEET 0.0000
0.0000 SPREADSHEET WORD_PROCESSOR 0.0000
```

```
0.0000 SPREADSHEET CALCULATOR 0.0000
0.0000 SPREADSHEET WEB_BROWSER 0.0000
0.0000 SPREADSHEET MOVIE_PLAYER 0.0000
0.0000 WORD_PROCESSOR SPREADSHEET 0.0000
0.0000 WORD_PROCESSOR WORD_PROCESSOR 0.0000
0.0000 WORD_PROCESSOR CALCULATOR 0.0000
0.0000 WORD_PROCESSOR WEB_BROWSER 0.0000
0.0000 WORD_PROCESSOR MOVIE_PLAYER 0.0000
0.0000 CALCULATOR SPREADSHEET 0.0000
0.0000 CALCULATOR WORD_PROCESSOR 0.0000
0.0000 CALCULATOR CALCULATOR 0.0000
0.0000 CALCULATOR WEB_BROWSER 0.0000
0.0000 CALCULATOR MOVIE_PLAYER 0.0000
0.0000 WEB_BROWSER SPREADSHEET 0.0000
0.0000 WEB_BROWSER WORD_PROCESSOR 0.0000
0.0000 WEB_BROWSER CALCULATOR 0.0000
0.0000 WEB_BROWSER WEB_BROWSER 0.0000
0.0000 WEB_BROWSER MOVIE_PLAYER 0.0000
0.0000 MOVIE_PLAYER SPREADSHEET 0.0000
0.0000 MOVIE_PLAYER WORD_PROCESSOR 0.0000
0.0000 MOVIE_PLAYER CALCULATOR 0.0000
0.0000 MOVIE_PLAYER WEB_BROWSER 0.0000
0.0000 MOVIE_PLAYER MOVIE_PLAYER 0.0000
end
```

At here we use a trick to use n-gram to simulate a grammar where each of the keyword we chose will only be entered once. Because all the possible bigrams are zeroed. The only possible path will the hypothesis which enter a word only once. Let us call this file **simple.arpa** from now on.

4.3.5 Conversion of the language model to DMP file format

Sphinx 3 actually doesn't support the text ARPA file format. If you want to use your grammar in Sphinx 3 , you could use a tool called **lm3g2dmp** to convert the text file to .DMP file format. You could download it in the **share** directory in CVS from sourceforge using the software installation process discussed in Chapter 4.

Then conversion of the LM could be run as

```
lm3g2dmp simple.arpa
```

A file named **simple.arpa.DMP** will be created. You will need it in Step 5.

4.4 Step 4: Obtain an Open Source Acoustic Model

Enough for language model, it is time to gather resource such as acoustic model. There are a couple of notes before we start.

The first thing one need to know about open source acoustic model is that they are currently stills some rare and valuable resource in the world. One would perhaps appreciate this fact more when they know how acoustic model training is actually done. One would hope that the availability of open source speech recognizer would probably initiate more effort on building open source acoustic models. Do give two thumbs up to any one who build and open source the model. They actually do a good thing that is similar to St. Teresa for speech recognition.

The second thing about open source acoustic model (as well as language model) is that they might not be always suitable for different people need. In general, using large amount of speech to train a model will make an acoustic model to be generally very accurate. However, in specific domain, say in digit recognition, it might be true that just using utterance with digits will be better in some cases. This explains why a lot of developer found that open source models they could obtain usually don't satisfy them. If you don't think the open source models is the best, training is always an option for you. Sphinx also provides a trainer. So you have the freedom to fine-tune the model.

In this section, we will discuss various things one need to do get and use a model. Some basic knowledge on how to interprete the model will also be discussed.

4.4.1 How to get a model?

The first place to try is the open source page.

http://www.speech.cs.cmu.edu/sphinx/models/

There is one model that many people used which is so called the broadcast news model or hub-4 models.

This is one important model you will always come up with. So let us talk about this a little bit.

First of all, the model is trained by CMU and have been trained using 140 hours of 1996 and 1997 hub4 training data. This is probably the largest amount of data which have been put into an open source models.

Another thing that you might notice is that this model is trained for read-speech so this is not particularly good for conversational speech. In general, conversational speech could be indeed quite hard to handle.

The phoneset for which models have been provided is that of the dictionary cmudict_0.6d available through the CMU web pages. This matches with the procedure we were using in Step 2.

The dictionary has been used without stress markers, resulting in 40 phones, including the silence phone, SIL. Adding stress markers degrades perfromance by about 5within-word and cross-word triphone HMMs with no skips permitted between states.

There are two sets of models this package, comprised of 4000 and 6000 senones respectively. They are placed in directories named 4000 senones and 6000 senones respectively. The reasons why two sets of models are there partially because the trainer has trained two sets of them such that he/she could test the performance of the two settings. Also partially because we hope you could try which setting is more suitable for you.

There are also two settings you would see, one is 1s_c_d_dd, one is s3_1x39. These funny symbols means the type of dynamic coefficients one would compute. The detail of what they mean could be found in Chapter 5.

The two models could be used in both sphinx 3 fast (**decode** and sphinx 3 slow **decode_anytopo**. [9]

[9]At 20050523, this is slightly different from the notes one could found.

4.4.2 Are there other models?

Actually, that are more, For example, the TIDIGITS could be found in

????,

the RM models could be found in

????.

You could also train a model pretty easily for AN4. Though, you could observe those are much small models.

4.4.3 Why Acoustic Models are so Rare?

Acoustic model is actually pretty hard to come up. It takes weeks to prepare and take days to train. Most academic or commercial institutes just keep them because it means one valuable resource for them. Many people appreciate the fact that Sphinx is open sourced. I personally hope that more and more people will realize another thing, the HUB-4 models is open sourced because probably this is the first model one could use it to do large vocabulary speech recognition.

4.4.4 Interpretation of Model Formats

Sphinx's HMM definition are spread in five different files. They are **mean**, **variance**, **mixture weight**, **transition matrices** and **model definition**. It is quite important to have some ideas on what it means before we go on further.

A model definition file

So, let's start from the model definition file. They are usually located in the model_architecture directory when SphinxTrain 's standard procedure is used to train the model.

Here is an example:

```
0.3
55 n_base
118004 n_tri
```

```
472236 n_state_map
2165 n_tied_state
165 n_tied_ci_state
55 n_tied_tmat
#
# Columns definitions
#base lft rt p attrib tmat ...  state id's ...
+BACKGROUND+ - - - filler 0 0 1 2 N
+BREATH+ - - - filler 1 3 4 5 N
+CLICKS+ - - - filler 2 6 7 8 N
+COUGH+ - - - filler 3 9 10 11 N
+FEED+ - - - filler 4 12 13 14 N
+LAUGH+ - - - filler 5 15 16 17 N
+NOISE+ - - - filler 6 18 19 20 N
+SMACK+ - - - filler 7 21 22 23 N
 +UH+ - - - filler 8 24 25 26 N
+UHUH+ - - - filler 9 27 28 29 N
 +UM+ - - - filler 10 30 31 32 N
 AA - - - n/a 11 33 34 35 N
 AE - - - n/a 12 36 37 38 N
 AH - - - n/a 13 39 40 41 N
 AO - - - n/a 14 42 43 44 N
 AW - - - n/a 15 45 46 47 N
 AX - - - n/a 16 48 49 50 N
 AXR - - - n/a 17 51 52 53 N
 AY - - - n/a 18 54 55 56 N
 B - - - n/a 19 57 58 59 N
 CH - - - n/a 20 60 61 62 N
 D - - - n/a 21 63 64 65 N
 DH - - - n/a 22 66 67 68 N
 DX - - - n/a 23 69 70 71 N
```

```
EH - - - n/a 24 72 73 74 N
ER - - - n/a 25 75 76 77 N
EY - - - n/a 26 78 79 80 N
F - - - n/a 27 81 82 83 N
G - - - n/a 28 84 85 86 N
HH - - - n/a 29 87 88 89 N
.     .     .
```

The first few line describes the number of parameters in the whole system

```
0.3
```

0.3 is the model format version.

```
55 n_base
```

n_base is the number of base phone including fillers in the system. In this case, the number is 55.

```
118004 n_tri
```

n_tri is the number of triphone in the system. In this case, the number is 118004.

```
472236 n_state_map
```

```
2165 n_tied_state
```

n_tied_state is the number of triphone in the system. In this case, the number is 118004.

```
165 n_tied_ci_state
```

n_tied_ci_state is the number tied CI state in the system. In this case, the number is 165

```
55 n_tied_tmat
```

n_tied_tmat is the number tied transition matrix in the system. In this case, the number is 55.

Now let us try to interpret the following line.

```
#base lft rt p attrib tmat ...  state id's ...
```

```
+BACKGROUND+ - - - filler 0 0 1 2 N
```

From left to right, it reads the base phone is +BACKGROUND+, no left context and no right context, so it is a CI phone. It is a filler. Its phone ID is 0, its first state has senone ID 0, second state 1, third state 2.

It is quite useful to know how the hmm state would map into a senone. (In HTK terminology, the tied-state.) In the above example, if you know that the senone ID for first state is 0. You just need to look up the file means and variances to get the value of them.

```
AA - - - n/a 11 33 34 35 N
```

Here is another entry in the model definition file. This time, we see a typical CI phone entry. If you used the standard training procedure of SphinxTrain (that you will also have a chance to play with in the second half of this chapter), then all the CI phone HMM's state will have their own senone which is unique for it.

As opposed to that, if you look at a triphone definition.

```
AA AA AA s n/a 11 177 190 216 N
AA AA AE s n/a 11 177 190 216 N
AA AA AH s n/a 11 177 190 216 N
AA AA AO s n/a 11 177 190 216 N
AA AA AW s n/a 11 177 190 216 N
AA AA AX s n/a 11 177 190 216 N
AA AA AXR s n/a 11 177 196 216 N
AA AA AY s n/a 11 177 190 216 N
AA AA B b n/a 11 178 189 219 N
AA AA B s n/a 11 177 189 217 N
AA AA CH s n/a 11 177 189 216 N
AA AA D b n/a 11 178 198 219 N
AA AA D s n/a 11 177 198 214 N
```

You will see multiple phones states are actually mapped to the same senone ID. Why does that happen? This is essentially the result of tying happend in training the context-dependent model. What happen is there are always not enough training data for triphone model. Therefore, it is necessary to cluster the hmm-state to make data could be more efficiently used. Most of the time, clustering are either done by decision tree-base clustering or bottom-up agglomerative clustering.

The Means and Variances, Mixture Weights and Transition Matrics

Given the above, you could probably make more sense for the rest of the four files. They are usually located in a directory called model_parameter.

All of them are binary files. To read them, you could use a tool called **printp**.

You could find the detail usage of **printp** in Appendix B. However, as it is the first tool in , let us make a more detail look on it. Every tool in SphinxTrain are designed in a consistent way such that user could use it easily.

If you just type

```
printp
```

The following help information would be shown (as at 20050811)

```
-help no Shows the usage of the tool
-example no Shows example of how to use the tool
-tmatfn The transition matrix parameter file name
-mixwfn The mixture weight parameter file name
-mixws Start id of mixing weight subinterval
-mixwe End id of mixing weight subinterval
-gaufn A Gaussian parameter file name (either for means or vars)
-gaucntfn A Gaussian parameter weighted vector file
-regmatcntfn MLLR regression matrix count file
-moddeffn The model definition file
-lambdafn The interpolation weight file
-lambdamin 0 Print int.  wt.  >= this
-lambdamax 1 Print int.  wt.  <= this
-norm yes Print normalized parameters
-sigfig 4 Number of significant digits in 'e' notation
```

If you type

```
printp -help yes,
```

help information would be displayed.

If you type

```
printp -example yes,
```

Several examples of the command would be shown.

Therefore, if you are familiar with these basic command usages, you could probably have no need to reference the manual too much. All the

symnopsis and examples are all replicated again at the end of this book. This will be a plus for people who like to have a book form of manual.

Let us get back to our example, one practical thing you may want to do from time to time is to inspect whether the mean values make sense. For example, acoustic model training could make some component of means and variance vector to be abnormal.

printp will allow you to inspect these values by display them on the screen. For means and variances, you could use

```
printp -gaufn mean,
```

or

```
printp -gaufn var,
```

to show the values. If you want to look at the mean for a particular senone, what you need to do is just to look up the model definition file to know the senone.

printp also allows you to load in a model definition so that you don't need to manually look up the model definition file all the time.

To read mixture weight and transition matices, you could use the -mixwfn and -tmatfn in the option list. The rest is pretty trivial, you could probably get it by just running the command once.

4.4.5 Summary

In this section, we retrieve a model and learnt how to interpret its parameters. The next section will really use all the resources from Step 1 to Step 4 to build a system.

4.5 Step 5: Develop a speech recognition system

Now you got the acoustic model, language model and dictionary. It is time to use it in the recognizer.

You could think of how people run a speech recognizer in two ways. One is called batch mode, the waveform is recorded first, then decoding is done for the whole waveform. One is called live mode, that is recognition is done on the fly when some speech samples was captured by the audio device.

The two modes are slightly different in practice because usually cepstral mean normalization (CMN) is run before decoding. What CMN does is that it would try to estimate the mean of all cepstrum and use it to normalize all cepstral vectors. (For detail, please read Chapter 5.) In batch mode, the estimation was usually carried out by computing the cepstral mean using *all* frames. Most of the time, this is usually a very good estimate. In live-mode though, since only part of the utterance is seen, one needs to approximate the CMN process with other means. These approximations usually introduce some degradations.

In this section, we will talk about how we could come up with a system that could that batch mode decoding, live mode simulation and actual live mode decoding. In terms of technical difficulty, you could say the three are in ascending order. So, going through one after one will be a good idea for novices.

4.5.1 About the Sphinx 3 's Package

Before we go on, let us take a look of what one could get from a Sphinx's package.

Assume you go to the install path as you have specified in Chapter 3, then the following software will be show up in the directory. The following executables are what you could find when you install Sphinx 3 .5.

```
align allphone dag livepretend
astar decode decode_anytopo
```

In windows, you would also another executable called **livedecode**.

For many, it is not easy to understand what these executables are supposed to do. So here is a simple description. **decode**, **decode_anytopo**,

livepretend and **livedecode** are decoders. However, each of them slightly different functions and should be used according to different situations.

- **decode** , also called "s3 fast", it is a batch mode decoder using tree lexicon, with optimization on both GMM computation and crossword triphone tranversal. On accuracy, it is poorer than **decode_anytopo**(as described below.). On speed, it is around 5 times faster. From Sphinx 3 .6,**decode** also allows decoding using finite state grammar and different implementations of search. Therefore it is a well-balanced recognizer and might be the first choice of batch mode recognition.

- **decode_anytopo**, also called "s3 slow", it is a batch mode decoder using flat lexicon, with optimization on trigram and cross-word triphones. Its performance is very close to a full trigram, full cross-word recognizer. Internal to CMU, it was used in several evaluation back in 1994 to 2000 and it provides the best performance among all the decoder.

  Both **decode** and **decode_anytopo** only works with cepstral file but not waveforms. Waveforms need to be transformed by cepstral vectors first using **wave2feat**.

- **livepretend**, is a live mode simulator using the engine of **decode**. What happens is livepretend will segment a waveform and feed the segments of speech to the search engine of decode. The segmentation is currently done in an ad-hoc manner by choosing 25 frames to be one segment. As we have explained before, segmentation of speech will make the cepstral mean normalization. So usually, **livepretend** works slightly poorer than **decode**.

- **livedecode**, is a live mode demonstration using the engine of **decode**. **livedecode** allows the user to try to recognize speech in push-button manner. It is by all means **NOT** a dictation machine. It is an example to show how the live mode decoding API (as detailed in Chapter **??**) could be used to build a system.

After the four major decoding tools, we should look at other tools such as **align, allphone, astar, dag**. They are search programs with specific purpose.

- **align** Given the transcription of an utterance, **align** could give the state, phone and word level of alignment to the user.

- **allphone** Carry out phoneme recognition for an utterance with full triphone expansions.

- **dag** Both **decode**, **decode_anytopo** could generate lattice for second-stage rescoring. **dag** as one machine which could do this rescoring could search for the best path within the lattice given a trigram language model.

- **astar astar** could generate an N-best list given a lattice. It is very useful for N-best rescoring.

When using all tools we mentioned, it is useful to bear in mind some common design properties of tools within the Sphinx 3 .x package. All the tools in Sphinx 3 .x has the following options.

`-logfn`.

This could allow one to record every messages of each program.

Applications of Sphinx 3 .x could have quite a lot of command line arguments. If you just type the command itself, a fairly detailed command-line usage description could be found.

If you don't like to specify command line manually, you have couple of options, the trivial one is store the command-line option and put in a Unix script file and it will look like this:

```
decode
-opt1 <val1>
-opt2 <val2>
```

Another way is that you could put the option in one file and retreat it as configuration file. So, you could create a file and look like this

```
-opt1 <val1>
-opt2 <val2>
```

and called it *config*

Then you just type

```
$ decode config
```

Then decode will automatically use the contents of config to initialize decode. That seem to save a lot of tyings for me. Besides, as some tools actually share the same command-line structure. This could give you a more convenience.

4.5.2 A batch mode system using decode and decode_anytopo: What could go wrong?

Let us start with building a batch mode system using **decode** and **decode_anytopo**. They are the first two decoders we will play with. To make your work easy, you need to have a big concept how a recognizer works. There are couple of resource a recognizer need to have before it could run.

- The dictionary In Sphinx 3 , that includes a lexical dictionary and a filler dictionary.
- The language model In Sphinx 3 , this could be an n-gram as we described or it could also be a finite state machine.
- The acoustic model As we mentioned, there are 5 components of it, the means, the variances, the mixture weight, the transition matrices and the model definition.

In batch mode, you also need to figure out a way to let the recognizer to read in the cepstral files.

Let us think about why the recognizer needs them. Essentially, an HMM-based speech recognizer just tries to compose a graph of states and search the best path within the graph.

Therefore, for the first level, a word graph is required. That could be either an N-gram and FSM.

For the second level, the recognizer needs to know how the word will map to a phone sequence. That's why a dictionary is necessary. As most large vocabulary recognizer, Sphinx also uses concept of fillers. Major reason is specifying fillers in language model is not feasible and most of the time, special treatment is necessary on fillers. That's why a separate dictionary is required.

For the final level, the recognizer will need to know how each phone could be represented as state sequences. That's how the hmm set was used.

The above concept is illustrated in Figure. 4.1.

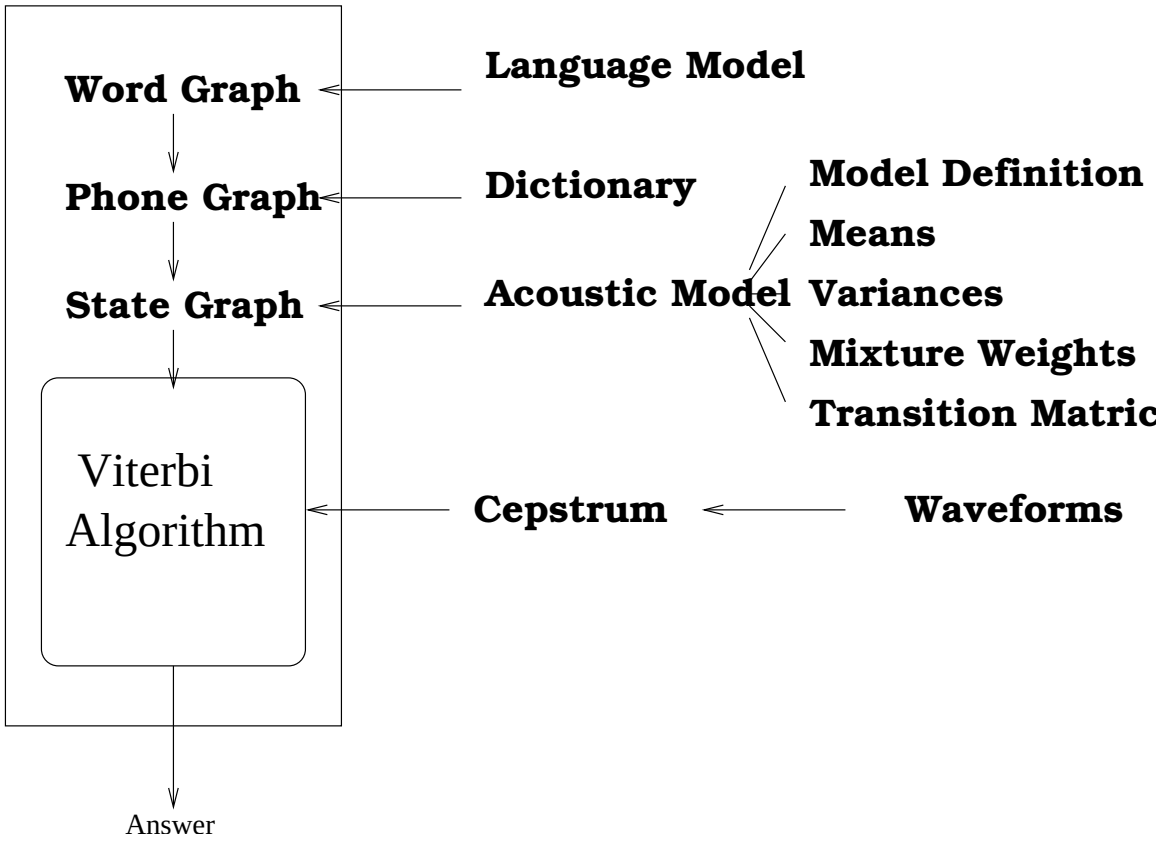

Figure 4.1: Resource used by the decoders

.

So, the four options are essentially what you need to have in the decoder. These are the four things you need to first identify. This sounds pretty simple.

"If it is that simple, why it takes me so much time to setup the decoder?" That happens to be many people's questions. Therefore, it is worthwhile to do some initial check of your resource. In general, you cannot put in arbitrary dictionary, language model, acoustic model and features. Most folks don't understand that, that's why most of the time the setup was wrong. So here is a check list you may want to go through first.

LM/Dictionary Mismatch

The first situation occurs between inconsistency between language model and dictionary. This is what happens, the language model will try to look up the dictionary and see whether a word exists. If it does, the word will be included in the word graph. If it doesn't, the word will not be included. This situation could be described as *Language mode-Dictionary mismatch*. All sphinx 3 decoders will warn the users about this. However, it is a design decision not to stop the recognizer being used even the situation exists because it is a very common situation in real life where the dictionary and language model are usually prepared by two different persons.

If you found that happend to you, make sure you add words to the dictionary, then things will be fine.

Dictionary/AM Mismatch

Then the next common thing that happens is mismatch between dictionary and the acoustic models. This is what hpappens. When the phone set used in the dictionary doesn't match with the phone set used in the acoustic model. The decoder will then stop. This happens a lot when the dictionary are obtained from two sources.

If you see that happen, make sure you do a check of all the phones of dictionary and compare it will what you have in the acoustic model. For models that trained from the recipe of SphinxTrain , this should not happen.

What if you have a model and a dictionary, you know that they have mismatch but you still don't want to use them together? That happens a lot because most of the time you may not be the one who train the model.

Now what you could do is to map the phones you found in acoustic models or the dictionary such that at the end the two sets are matched. Of course, make sure you know what you are doing. Also, please remind yourself that linguistic knowledege could be wrong in this particular situation. Validation of the model using data should be the final check.

AM/Feature mismatch

This problem might be the most subtle and could be the hardest to check. However, this is also the number one killer of most decoding setup occurred in Sphinx Forum. The idea is this unless you know what you are doing, difference between the front-end setting of acoustic model and front end could be detrimental to the recognition result. There are several parameters you will see in **wave2feat** and **livedecode** that could control the front-end. Bear in mind, parameters such as **-cmn**, **-agc** and **-varnorm** are also parameters that could control the frontend. If any of them are different, it is not easy to predict what's going on.

In general, when a model is released, to avoid model mismatch in future. It is important for the trainer to also release information about the feature parameters used in training.

Now, we could run it.

First convert the waveform to cepstral first, let us called, the input waveform to be **a.wav**.

[Editor Notes: We need to verify both the arguments of **wave2feat** and **decode at here**]

```
wave2feat
-alpha 0.97
-srate 16000.0
-frate -100
-wlen 0.0256
-nfft 512
-nfilt 40
-lowerf 133.33334
-upperf 6855.49756
-ncep 13
```

```
-dither yes
-mswav yes
-i a.wav
-o a.mfc
```

The setting follows exactly what one could find in the open source Hub-4 model at

http://www.speech.cs.cmu.edu/sphinx/models/hub4opensrc_jan2002/INFO_ABOUT_MODELS

The setting may not be totally fitted into your situation. First of all, we assume (**-mswav yes**), that is to say Microsoft wave format. **wave2feat** could assume input file to be either nist (**-nist yes**), raw (**-raw yes**). If you have a batch file, you could also use **-c** to specify a control file. Usage is trivial.

We also recommend you to specify **-dither yes**, the reason is that sometimes if the wavefile contains a frame that is all zero. **wave2feat** could add a small random number to the samples. Without doing so, the frontend could give errorenous result. [10].

After that, you just need to run decode to run the results.

```
decode
-lm simple.arpa.DMP
-dict simple.txt
-fdict filler.dict
-mdef hub4opensrc.6000.mdef
-mean means
-var variances
-mixw mixture_weights
-tmat transition_matrices
-ctl simple.control
-cepdir <your cepstral directory>
-agc none
-varnorm no
-cmn current
```

Inside **simple.control**, add **a** into the control file like this

[10]Mainly because the log operation.

a

Make sure that a exists in the directory ¡your cepstral directory¿. The result will then be displayed in the screen. You could also redirect the result to a file. If you think it is too chatty. You can also specify **-hyp** <**your hypothesis file**> which could give you just the answer. [11].

As we mentioned, you could also use **decode_anytopo** instead of **decode**. Starting from 3.6, **decode_anytopo** and **decode** have sychronized command-line interface. Therefore, we recommend you to use it instead of the older version. That is to say the usage will be the same for the two executables.

4.5.3 A live mode simulator using live_pretend

Now we make one further step, we now start to simulate a live mode recognition. **livepretend**'s usage is a very similar to the use of a combined **wave2feat** AND **decode**. So the usage is very similar to use the tools together. **livepretend** just like **livedecode** only usage of config file but disallow usage of command line arguments. They are the only two applications that have this properties. [12]

There are one interesting thing about using the Sphinx's live-mode recognizer which is it allows the user to backtrack the answer before the sentence is ended. In livepretend, you could turn on and off this functionalities by specifying **-phypdump 1/0** in your setting. By default, the code will always give you the partial hypothesis.

What is the use of the partial hypothesis? You may ask. One use of it could be, if you find that a particular keyword is already recognized, you could just stop the recognition without going through.

4.5.4 A Live-mode System

So, now we come to the final part of actually using a recognizer. For many of you, it might happen that using the batch mode recognizer already suffices for your purpose. Most people who use Sphinx are either researchers or developers. For researchers, just changing either part of the system and

[11] another trick you could use is that you could just grep the pattern FWDVIT from the recognition output

[12] This might be something we will want to change it in future. However, backward compatibility will also be maintained if that happened.

test the changes are usually the most important tasks in their lives. In that case, they could either use **decode** or **livepretend**. **decode** is essentially a batch mode decoder. Whereas **livepretend** could also simulate situation of live-mode decoder. The latter is slightly better for scenario where techniques could possibly make use of all the frames of an utterance.

For developers, using **decode** with **wave2feat**, by themselves, already form a very nice software collection to build some types of real-life application. For example, in case your application could afford it. Push-Button application could you to first record the waveform then recognize the waveform. In that case, one could first record the waveform, process it by **wave2feat** and then pass it by **decode**. The whole process could just be a replication of the script we mentioned in Section 4.5.2.

For developers though, using the batch mode decoder in an application may not be the best choice all the time. Major reason is delay. Consider this, using the record-then-recognize approach, the response time of the sytem will equal to the recording time plus the recognition time. If the user speak for 10s, then, let's say the recognizer also use 10s to recognize (or 1x RT scenario, explained later). Then, it might take 20s for the system to respond. Not many users will be happy about it.

Therefore, for application developers, a more favorable scenario will be while recording some frames in the users speech, start the recognizer to recognize the recorded speech even though it might not be finished. This is usually called *live-mode* recognition. Most commercial dictation engine works in that way. This is what we want to discus in this subsection.

So how to do it? We would not cover this in detail in this chapter. However, we will give you some ideas and point you to look at further detail in Chapter **??**

Let us just go through the process first. The first step is to do recording and always put it to some kind of buffer of your program. To record the program, you could try to use libs3audio routine, such as **cont_ad**. If you want to enjoy more flexibility, we would suggest you to use a library called **portaudio** [13]. The mentioned package could already give the desired result.

The samples should put to a feature extraction routine and the routine will convert the audio samples into cepstral coefficient. The coefficient can then transfer to the recognition routine and recognition will then be performed. Notice that this time only a segment of speech will be decoded one time. What you need to do is just to continue this process until there

[13]Not a Sphinx software but it has a pretty liberal licence, you could find the detail in **www.portaudio.org**

is no speech in the audio buffer.

The above description should give you some idea what you need to do. In Chapter **??**, you will also see detail program listing. We will also provide discussion on issues of creating a live-mode applications.

4.5.5 Could it be Faster?

Assume you have finished the above exercise of the building a recognition system, it is time to notice something wrong and try to improve it. The first thing you might have notice is that the code might be too slow to be useful. As the end of the whole section, we will provide you a look on how this problem could be tackled.

Search of speech recognition is a very time-consuming process. In Chapter **??**, we see that Viterbi algorithm is usually used to efficiently search for the best path in the HMM. In this chapter, we also learned that the actual HMM that will be composed by different level of resource could be huge.

This leaves us a difficult situation. Even with Viterbi algorithm, we need to search in a huge graph with many possible hypotheses. So, it is important know something fundamental on how to speed up the process. In this tutorial, we will explain the concept of beam pruning.

Consider a searching process where multiple paths are being scored in a time-sychronous manner. What one could do in this process is to "prune" or removed paths which have very low score. How could we do it? Here the idea of beam pruning comes to our excuse. We could use the score of the best path as a reference and remove all paths that have scores way lag behind the best path.

Usually, this process is controlled by a parameter called beam. If the beam is wider, more paths will be preserved and the search will take longer time. If the beam is narrower, the decoding will usually be faster.

In Sphinx 3 decoder set, including **decode**, **decode_anytopo** and **livepretend**. You could use the paramter **-beam** to control the number of paths you want to preserve. By default, it is set to be $1e-80$, If the number becomes larger, the beam width will become smaller. For example, you could consider using $1e-60$ to increase the speed.

Notice that it is a balance to control the size of beam and the accuracy rate you could get. Usually a tuning set is first defined and experiments are performed to determine the best beam-width for a task. Next section will give you a better idea on how this could be done.

Beam pruning is not the only way where the Sphinx 3 decoder set allows user to speed up the recognizer. There are several important features for fast recognition in Sphinx, that includes

1. Fast GMM computation including techniques such frame down-sampling, context-independent model-based GMM selection (CIGMMS) and Gaussian sele
2. Lexical tree implementation that compress the size of the search space by organizing it using a tree-based structure. There are several interesting issues could arise from this structure. Sphinx provides several interesting and practical solution for these issues.
3. Different levels of pruning the beam we mentioned in this chapter is only one type of beam we could use in practice. For example, in a lot of experiments, one find that usage of phone-level and word-level of beam could also be useful as well.

We will advise users to look at Chapter 9 for detail of these techniques.

4.6 Step 6: Evaluate your own system

It is not enough for just building a system. If you would like to know how to make it even better, evaluation of the system is necessary.

One may separate the methods of evaluation of a speech recognition system into two types, *on-line* and *off-line*.

In the on-line evaluation, a person will actually call the system and see whether it performs well. Usually, the evaluator will use his subjective judgment to decide whether the system is performing well enough. To make the evaluation more objective, it is reasonable to ask the evaluator to carry out multiple tasks to test different functionalities of the system.

In the off-line evaluation, a collection of recordings from the users will first be obtained. These recordings will then be used to test whether how they system performs. There are generally two accepted performance metrics for this type of evaluation. One is sentence error rate and one is word error rate.

It is generally true that using off-line evaluation with large test set will give you a more informative and objective results. However, we could not ignore the usefulness of on-line evaluation in general. This is because some system related issue could only be detected by actually using the system.

We will have some discussions here on which mode of evaluation should be used in your application at this section and generally how to.

4.6.1 On-line Evaluation

There are two distints issue in testing a real-life speech recognition system. One is to test the speech recognizer's performance and one is to test the application-specific's capability. In our opinion, for most of the time, on-line evaluation could be very useful in the latter case but it might be dangerous in the former case. This is the general reason why. Usually, on-line evaluation rely on subjective judgment of the evaluator on whether the recognizer is good or not. Chances are it could be the acoustic or language models *do* cover most of the speakers of the target population but the evaluator happens to be *an outlier*. The evalutor usually has no idea whether this is happening in the evaluation. Therefore, his opinion of the system could easily be biased to factors other than the strength of the system.

Another side of the danger of on-line evaluation is that the evaluator

could be the system programmer. He will then tend to evaluate the system using phrases which are easy to be recognized. This could result in over-optimistism about the system.

Despite these problems, on-line evaluation is indispensible to get a "feeling" of the system such as speed and system stability. Though, it is sufficed to say both issue could also be affected by other programs which are running in the background of the applications.

4.6.2 Off-line Evaluation

Off-line evaluation is perhaps a more objective way to carry out evaluation of speech systems. Usually, a set of testing data is first recorded, then the sentence error rate (SER) or word error rate (WER) are then found. Sentence error rate are defined as the percentage of the number of mis-recognized sentence out of all tested utterances. Whereas WER is usually defined as

$$A = \frac{H - S - I - D}{H} \times 100\%, \tag{4.1}$$

In order to take account of the insertion and deletion errors into computation of accuracy, the word accuracy is defined as,

where

H is the total number of words in the reference,

S is the number of substitution errors,

I is the number of insertions errors and

D is the number of deletions errors.

Usually, a dynamic string matching algorithm is used to decide the alignment between the transcription and the decoding output. One program we recommend is **sclite** developed by NIST

ftp://jaguar.ncsl.nist.gov/pub/sctk-2.1-20050603-1454.tgz

Despite the soundness of the method, it could be possible that a particular collected test set is only biased to certain type of system. Therefore, improvement of system should not be used to optimize on just one test set but on variety of conditions.

4.7 Interlude

Now you have build your first system, you sweat to collect all resources and incorporate them with the speech recognizer. You also evaluated it and get a number as a performance metric. This is usually not the most satisfactory result. As we explained, the model you used could be trained in a totally different conditions and might not be useful in your situation. For example, one common problem is that what one could get is only a 16K models but the actual scenario requires a 8k setup. What you could do is probably to recalculate the position filter banks. However, it doesn't always give you the best result you could have.

This is the time you may want to train a new model. However, it is important to realize that training could be harder than what you think. There are seveal processes that could take you a long time. These are unfortunately not fully understood by many novices of speech recognition system. [14]

In general, the actualy training procedure, compared to the preparation process such as data collection and transcription, are usually the least laborious. Therefore, instead of giving you any real technical advice first, we will recommend you to be *psychologically prepared* before the process. Because no matter how detail and careful our instruction are, the process of training still takes quite an amount of time.

In the rest of this chapter, we will go through the basic process of training. We will not cover everything and will point you to some later chapters for further details. We will consider alternatives of some laborous process and you could decide which way to go.

4.8 Step 7: Collection and Processing of Data

From now, we start to train a model from scratch from collecting data to create the final acoustic models. So, let us try to see the process from the beginning. That is how data is collected in the first place in speech recognition.

Essentially, what we need to do is just to collect some data and record it as waveforms. The tricky part is that

- It is better to collect as much data as possible.

[14] Worst of all, when they have no one to blame. They blamed the recognizer.

- The waveforms have to be recorded which are similar to the condition in the actualy decoding.

4.8.1 Determine the amount of data needed

How much recordings are needed? Most of the time, at least 1 hour of speech is required to a basic model. If a model is supposed to be used for speaker-independent speech recognition for a large population, perhaps more hours of speech is required.

The amount of data needs to be collected also relates to the parameter of the acoustic model.

Here is a real life example, if every Gaussian distribution requires 100 data points to train. With a system with 1000 senone, each senone is represented by 8 Gaussian, then loosely speaking [15] In that case 800000 frames of speech will be required. In that case, assuming that 1000 frames mean 10s of speech (This depends on the sampling rate and frame rate). You will need have 800 utterance which solely composed of speech. Now, let's say we assume that the speech density is 50%. Then 1600 utterances of waveforms will be needed. That if you do the calculation that will approximately $\frac{1600\times10}{60*60} \approx 6$ hours of speech.

It is also important to make sure that the recordings are phonetically balanced. What it means is it is better to make sure that the amount of training data that will train each phone should be balanced. For example, let's follow the above examples and you did collect 1600 utterances but all of them are "Hello", then in this case, only 4 phones will be trained. (The number of triphones are also pretty limited). Then, other phones in your phone list will be totally empty in your acoustic models. That usually will cause the training process behave abnormally one way or the other.

4.8.2 Record the data in appropiate condition: environmental noise

To explain what it means by appropiate, lets consider the following very simple example, model trained by data collected in an office room would likely perform badly in a condition where there are construction noise

[15] it's loose because Baum-Welch algorithm will give every states a fractional counts for training.

around. (And so does a human audio system.) Vice versa. In general, the environment usually change the speech characteristic.

In a signal processing term, there are usually two different types of noise. One is additive noise and the other is convolutive noise.

Additive noises are usually caused by external sound sources such as construction noise, babble noise, gun noise. The sound wave superimpose the speech and causing the receiver machine record some speech which is different from the original sound source.

Convolutive noise is also called channel noise. For example, when a sound wave pass through a media such as air. Or recorded through a microphone. Their characteristic will usually change simply because different frequency component of the sound wave response differently to the channel. Or in another words, the channel forms a filter to the signal.

From the above discussion, it is easy for us to understand why most dictation engine's instruction will usually instruct the users to record the speech in a quiet environment. [16] Some vendor will even provide users a prepared microphones. Likely, this is not really because the microphone is better than the other but because the type of microphone has been used consistently in collection process.

For developers, this implies that data collection could be the first challenge one needs to deal with. For a desktop application, a quiet room and consistently using same type of microphones would suffice. It is also important to instruct the users to speak to the microphone with constant distance especially headset microphone.

[16]The algorithm could actually account for additive noise

4.9 Step 8: Transcription of data

After recording all the data, the next stage is to create a text description of the waveforms.

So, how could we do it quickly? "Well, may be we could use a speech recognizer to recognize what the data said." You may say.

Of course, you might immediately see the chicken-and-egg property of the problem. For most of the time, we usually don't have a good recognizer in the first place. Hence transcription is usually done by a human being. In some sense, you can imagine this is a stage of our work where human need to be involved the most.

Human transcription is a topic of its own so it is very difficult for us to give it a complete exposition. Let us just start from several aspects of the process.

4.9.1 Word-level and Phoneme-level Transcription

Transcription is essentially just try to give the most accurate description of the waveforms. It is a relatively easy process in the word level, i.e. the final transcription is a word sequence. Then, what you need is to find a person who knows the language you are working on and create the transcription.

There are some situations where you need to create a phone-level of transcription. For example, to research on accented speech for a particular sets of populations. This is a situation you need an expert, for example, at least a student in the field of phonetics or in general linguistic, to help on your task.

4.9.2 Checking of Transcription

Unavoidably, transcription which is done by only one single transcriber could have a inconsistencies and problems. One common way to deal with this issue is to use another transcriber to double-check the transcription.

4.9.3 Software Tools for Transcription

Wavesurfer is great in analysing the waveforms.

[Editor Notes: We need a good recommendation of transcriber tools. I probably need to ask folks in sphinxmail]

4.9.4 Transcription for Acoustic Modeling and Language Modeling

After you have finished the transcription, it is important to format it and use it in different purposes. It is quite important to know that there are some important difference between transcriptions for acoustic model training and langauge model training. For example, it is not common to add filler words into langauge model training. Careful consideration should be made in every situation or transcript preparation.

4.9.5 SphinxTrain 's and align Transcription Formats

Generally, both the Sphinx 3 and SphinxTrain package accept transcription in a similar format. Usually, it looks like this

```
<s> HELLO </s> (hublog.19990723.001.002)
<s> HELLO </s> (hublog.19990723.001.003)
<s> I'D LIKE TO FLY TO SEATTLE TOMORROW AFTERNOON </s> (hublog.199907
<s> HOW 'BOUT SOMETHING LATER </s> (hublog.19990723.001.005)
```

Notice two things here

- <s> and </s> are by default used as sentence begine and end markers of the utterance. Internal to the code, they are also counted as silence.
- hublog.19990723.001.004.* is the utterance ID, this needs to match what you have in the control file of processing. The control file will generally look like this,

```
hublog.19990721.000.002
hublog.19990721.000.003
hublog.19990721.000.004
hublog.19990721.000.005
```

[Editor Notes: ARCHAN: How about the situation where the waveform is hugh?]

So you might need to write a simple script to convert the transcriptions format to this if you want to use SphinxTrain for training.

Because of different purpose of different tools, the best transcriptions for them are also different.

For all the tools in SphinxTrain, notably **bw** it is recommend to put the most acoustically correct transcription into the transcripts. Say something like

```
<s> I'D LIKE TO(3) FLY TO(3) SEATTLE TOMORROW AFTERNOON <s>
(hublog.19990723.001.004)
```

Major reason that the current **bw** required the best transcription is because it doesn't automatically generate multiple pronunciations and skip silences optionally. [17]. To generate this "most correct" transcription, the tool **align** is necessary. However, its use of transcription is different from **bw**.

And it will look like:

[Editor Notes: ARCHAN: Am I correct?] [Editor Notes: Not very sure about the technical contents at here.]

For **align** though, one just need to give the simple format of transcripts to it before the force alignment process. That is like

```
<s> I'D LIKE TO FLY TO SEATTLE TOMORROW AFTERNOON </s>
(hublog.19990723.001.004)
```

[17] This is likely to change in 2006

4.10 Step (7 & 8) Plan B : Buying Data

[If you don't want to sweat]

Likely, after you have read the last two sections, you might be scared by the amount of effort you need to put on data recording and transcription. Fortunately, if you have money, this problem could be solved in an easier way, which is to buy data.

The first place you could go to is Linguistic Data Consortium (LDC), their web site is

http://www.ldc.upenn.edu/

At there, you could consider to buy some of the corpora and use it for acoustic model training. There are usually two price tag on the corpora, one for academic use, one for commercial use. Rightly, the price for academic use is much lower than the one for commercial use.

From what we know, LDC actually welcomed a lot the public to buy and support the corpora. Most of the time, your site (assuming you are working in a school) might have already been the member of LDC and ordering corpora will be much cheaper in that way. Try to ask the local expert on this issue.

Other sites, for example, universities such as Carnegie Mellon Unversity (CMU), Oregon Graduate Institute (OGI) and Mississipi State University also open source data on their own. For detail, you could look them up in the individual web site.

There are couple of things you need to know about the bought data. One is on the legal side, for example, there might be restriction on the use of the data, *usually*[18], you could use the data obtained from LDC to train an acoustic model and distribute it for your purpose. However, it might also happen that it is not the case. So again, checking with the legal statement will give you the most.

The second issue is on the transcription, even though most of the data are transcribed. It doesn't mean the software you are using could handle the format used in the corpus. Hence, a procedure that transform this raw format transcription is usually necessary. This procedure is usually called text normalization.

The text normalization procedure is actually one crucial process to get a good acoustic model. For example, small issue in handling filler words such as silence could have huge difference in resulting error rate. Unfor-

[18]Please check the legal statement package in each CD and the web page for individual situation

tunately, the detail of these problems are usually specific to a task or a particular corpus. Therefore, we could not discuss too much. However, we still hope that the users are aware of these issues. [19]

4.11 Training of Acoustic Model. A note before we proceed

After data collection and transcription, it is probably the time to really train a model. The first concept of training is that training is never an easy task. Even though you might have put a lot of time to prepare, there are still some small problems that you need to solve. It is a statement which is independent of the tool you used. In the Sphinx system , there are indeed more concerns you might need to be careful.

There are two ways to proceed in training. Both of them have their fans. However, when used incorrectly, both of them could be difficult. We will talk about both of them and you would have a choice to choose which way to go. We will also discuss why people like or hate a particular approach.

4.11.1 SphinxTrain as a Set of Training Tools

The major tools you will use in training is a package called SphinxTrain . You could download it by using CVS

```
$ cvs -d:pserver:anonymous@cvs.sf.net:/cvsroot/cmusphinx
 co SphinxTrain
```

Then you just need to compile it through the standard dance.

```
$ configure
$ make
```

Like Sphinx 3 , SphinxTrain is not a single tool. As a matter of fact, it is actually a package that is composed of 35 C-based tools and 9 perl scripts. The 9 perl script is also called “standard recipe” of training because they call the low-level C-based tool in the training process. If use it properly, the

[19] From what we know, poor text normalization could be the possible reason why several open source dictation model builder fails in last couple of years.

"standard recipe" should guide you through the whole process and create a new set of models.

So, it is really easy to see two paths to proceed in training.

- To use the recipe provided in SphinxTrain, (Using the perl script).
- To use C-application directly and build the whole process from scratch.

This book provides material on both paths. So it will be up to you to decide which way to go. The next two sections, what you need to do in both case.

4.11.2 Training follow a recipe

Training following the "recipe", in essense, is similar to training using a tool which is composed by both perl script and C-code. If you have run an initialization script of linux, then you will have some ideas of what's going on with the Sphinx recipe script.

There are several reasons why people like to use a training script. First of all, using the training script is indeed more convenient than building everything from scratch. You just need to change the configuration file a little bit to make sure things are correct. For many, this is usually very nice.

Second, if you are trying to learn how to train a model, the training script will teach you a lot of things about training.

What is the bad thing about using the training scripts then? Admittedly, having training scripts is indeed much better than having no training scripts. However, it doesn't mean the trainer (i.e. you) doesn't need to do anything to get the training process to complete. You still need to take care of a lot of strange situation that could be caused by your own mistakes or something that just merely gone wrong in the data or transcription.

A lot of people thought that the training script will solve *all* the problems for them. This is, of course, a very naive point of view. Even though, the script should be treated more as a reference on how things should be done. As in reality, the trainer needs to decide what kinds of things he needs to do. Most of the time, if they are not taken care correctly, the process will just die prematurely.

However, we couldn't blame the user too much because the existence of the training script has already given an illusion to the users. That is perhaps one major issue in using the training script. Or we should say to

use the training script *as is* without doing any changes. This will indeed make people *lazier*.

4.11.3 Training from scratch

The next possibility is to train everything from scratch, this is something we don't always recommend to novice users. The idea is to start training the acoustic model based on merely the C-commands provied by SphinxTrain.

Doing so usually require quite a level of skill in the platforms (say Unix and Windows) and far much less amount of people were able to do it. However, most people who succeed could either find a job in speech recognition or become very familar with the tools. Major reason is the SphinxTrain commands developed in these few years have started to be fairly robust and self-explanatory. Direct usage of them could allow the user to learn much more about the process than just using the script. The difficulty is obvious. However, the fact that building the own scripts usually means the user can also control parameters such as beams and threshood. This could be crucial in the training procedure. So most of the time, this is major factor of improvement over standard recipe.

There is one caveat of building one's own script which is seldom talked about but it is quite worthwhile to be mentioned. That is, one should treat the scripting process as a solid program. Most of the time, ad-hoc script will lack of robustness of well-tested script such as SphinxTrain 's standard recipe. Because of the fact a working training script will always be copied around from users to users, it happens that the initial training script could become a bottlenect of a site. This is something one need to be careful when starting to write a training script.

4.11.4 Our Recommendations

So which way to go, among Sphinx Developers , the general consenus is we will generally recommend the users to use the standard recipe first before they start to tweak the scripts. What we observed was that a lot of users still have some miscellaneous troubles. Therefore, what we provided on top of that is a the standard training script that can *run through* the whole process given a specific task, or push-here Dummy (PhD) script in general. In the next section, we will first describe one of the most common training script and describe how to use it.

Given the PhD script, it doesn't mean there is no need for the user to inspect the system from stage to stage for potential problems. Or even better, to customize some of the scripts appropiately to deal with individual situations. This will be the topic of Section **??**.

4.12 Step 9 and beyond

At this revision, we would ask the reader to take a look of the training tutorial page in cmusphinx.org first. In later draft of this document, we will add more detail in this part.

Part III

CMU SphinxSpeech Recognition System

Chapter 5

Sphinx's Front End

[Editor Notes: Need to update this chapter for latest work in Dave's work in sphinxbase.]

Author: *Evandro Gouvêa, Mike Seltzerand Arthur Chan*, Editor: *Arthur Chan*

5.1 Introduction

This chapter describes the signal processing front end of the Sphinx 3 Speech recognition system. The front end transforms a speech waveforms into a set of features to be used for recognition. The current implementation creates mel-frequency cepstral coefficient (MFCC). The chapter is largely adapted from Dr. Mike Seltzer's "Sphinx 3 Signal Processing Front End specification" which first occured in the Sphinx 3 's software package.

5.2 Block Diagram

Below is a block diagram of feature extraction operations performed by the Sphinx 3 front end.

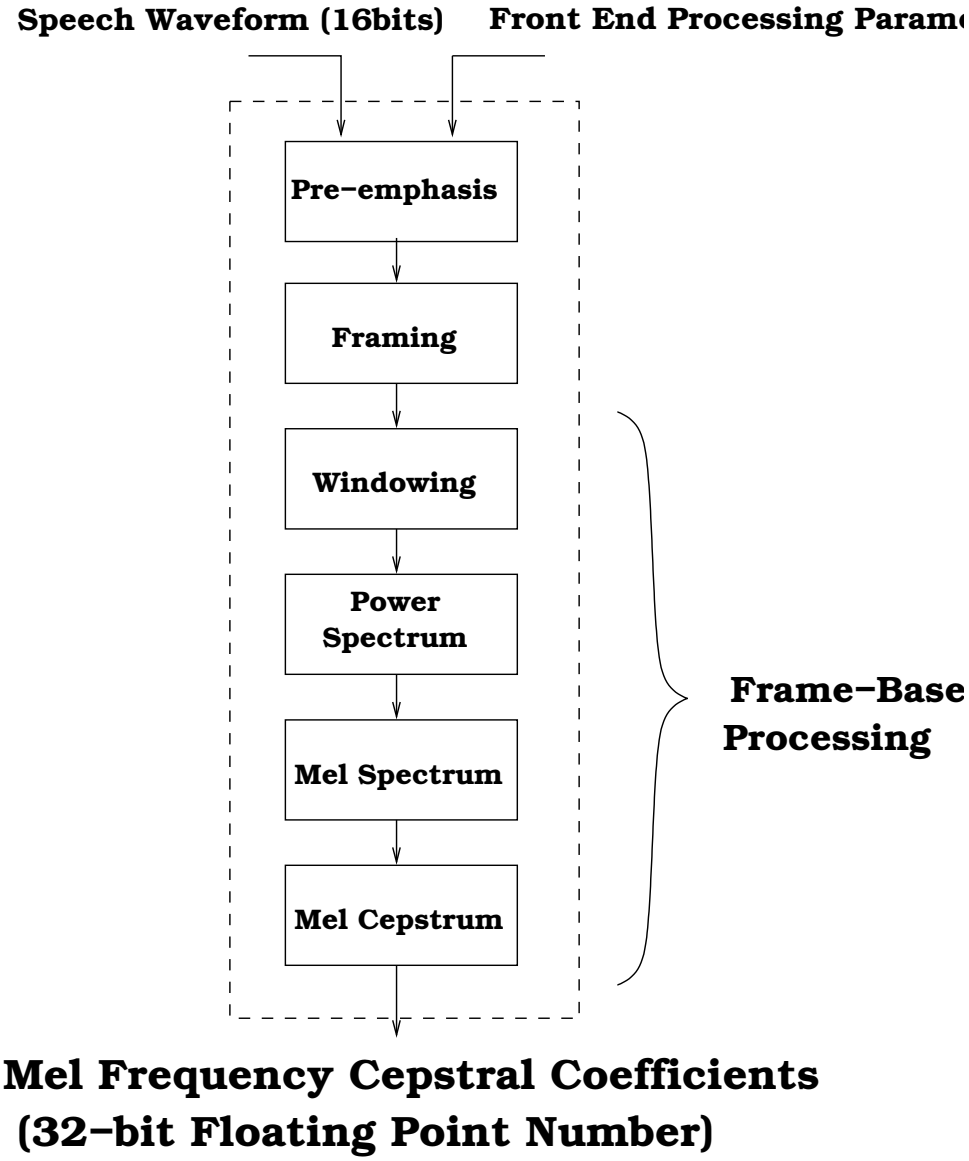

Figure 5.1: Block Diagram for the front-end processing

5.3 Front End Processing Parameters

There are several parameters which are controlling the front end. Here is a description of all of them. We also give the option names on how to use them in both **wave2feat** and the front-end of Sphinx 3 's decoders.

- **Sampling rate:** the sampling frequency of the input speech signal.
- **Frame rate:** the speed of how fast a window is moving. It is terms of number of samples.
- **Window length:** The size of moving window in terms of number of samples.
- **Number of Cepstra:** The number of coefficient *including* the energy coefficient in the feature vector.
- **Number of filters:** The number of filters that would be used in the filter banks.
- **The size of fft:** The number of points of FFT used.
- **The "lower" filter frequency:** It is actually the lower frequency of the lowest filter in the filter bank.
- **The "upper" filter frequency:** It is actually the upper frequency of the highest filter in the filter bank.
- **The pre-emphasis coefficient:** The value of α in the pre-emphasis filter. See Section refssec:preemp

There are two occasions where specification of the above parameters will be very important. One is when you need to use **wave2feat**. One is when you need to use the front-end of Sphinx 3 . Here is a table that shows the relationship. For further details, you could also consult the appendix of the book.

5.4 Detail of Front End processing

5.4.1 Pre-emphasis

The following finite impulse response pre-emphasis filter is applied to input waveforms:

Table 5.1: Command-line options for the front end parameters in **wave2feat** and Sphinx 3 's decoders

Parameters	Param. in **wave2feat**	Sphinx 3 's **livepretend** or **livedecode**
Sampling rate	-srate	-srate[1]
Frame rate	-frate	-frate
Window length	-wlen	-wlen
Number of Cepstra	-ncep	-ncep
Size of FFT	-nfft	-nfft
Number of filters	-nfilt	-nfilt
Lower filter frequency	-lowerf	-lowerf
Upper filter frequency	-upperf	-upperf
The pre-emphasis coefficient	-alpha	-alpha

$y[n] = x[n] - \alpha x[n-1]$

α is provided by the user or set to the default value. If $\alpha = 0$, then this step is skipped. In addition, the appropriate sample of the input is sorted as a history value for use during the next round of processing.

5.4.2 Framing

Framing will depend on the windows length (in terms of second) and the frame rate (in terms of frames/second).

[Editor Notes: Under construction]

5.4.3 Windowing

The frame is multiplied by the following Hamming window:

$w[n] = 0.54 - 0.46 \cos(\frac{2\pi n}{N-1})$

N is the length of the frame.

5.4.4 Power Spectrum

The power spectrum of the frame is computed by performing a discrete Fourier Transform of length specified by the user, and then computing its magnitude squared.

$S[k] = (Re(X[k])^2 + (Im(X[k])^2$

5.4.5 Mel Spectrum

The mel spectrum of the power spectrum is computed by multiplying the power spectrum by each of the triangular mel weighting filters and integrating the results:

$\tilde{S}[l] = \sum_{k=0}^{N/2} S[k] M_l[k]$ l=0,1,. . . ,L-1

where N is the length of the DFT and L is the total number of triangular mel weighting filters.

5.4.6 Mel Cepstrum

A DCT is applied to the natural logarithm of the mel spectrum to obtain the mel cepstrum:

$c[n] = \sum_{i=0}^{L-1} ln(\tilde{S}[i]) cos(\frac{\pi n}{2L}(2i+1))$ c=0,l,. . . ,C-1

C is the number of cepstral coefficients.

5.5 Mel Filter Banks

The mel scale filterbank is a series of L triangular bandpass filters that have been designed to simulate the bandpass filtering believed to occur in the auditory system. This corresponds to a series of bandpass filters with constant bandwidth and spacing on a mel frequency scale. On a linear frequency scale, this filter spacing is approximately linear up to 1kHz and logarithmic at higher frequencies. The following warping function transform linear frequencies to mel frequencies:

$mel(f) = 2595log(1 + \frac{f}{700})$

A plot of the warping function is shown below.

[Editor Notes: Should include Mike's diagram here]

A series of L triangular filters with 50% overlap are constructed such that they are equally spaced on the mel scale spanning $[mel(f_{min}), mel(f_{max})]$ where f_{min} and f_{max} can be set by the user. The triangles are all normalized so that they have unit area.

5.6 The Default Front End Parameters

These are the default values for the current Sphinx 3 front end:

Table 5.2: Default Parameters for the front end in Sphinx 3

parameter	default
Sampling Rate	16000.0Hz
Frame Rate	100 Frames/sec
Window Length	0.025625 sec
Filterbank	Mel Filterbank
Number of Cepstra	13
Number of Mel Fitlers	40
DFT Size	512
Lower Filer Frequency	133.33334 Hz
Upper Filer Frequency	6855.4976 Hz
Pre-Emphasis Coefficient(α)	0.97

5.7 Computation of Dynamic Coefficients

Author: *Arthur Chan*, Editor: *Arthur Chan*

There are three major types of mechanism for generation of dynamic coefficients. For space efficiency, the dynamic coefficients computation are usually carried out at the routine which directly uses the feature, for example, the decoders **decode** or **decode_anytopo**.

There are two sets of notation to represent the mechanism of dynamic coefficient generation, one for the decoder and one for the trainer. There is actually a one to one mapping between the two notations. The following table will be indispensable for one to understand the two notations.

The dynamic coefficient types s2_4x, s3_1x39, and 1s_c_d_dd are commonly used in Sphinx 3 . Currently other coefficient types, 1s_c, 1s_c_d, 1s_c_dd, are obsolete. They are currently safeguarded in various recognizers.

Table 5.3: Naming Convetion table for feature type s2_4x

Sphinx 2	s2_4x
SphinxTrain , simplified	4s_12c_24d_3p_12dd
SphinxTrain , generic	c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/
vector length	4 in parallel, dimensions 12, 24, 3, and 12
feature layout	12 ceps, 12 short term Δ ceps, 12 long term Δ ceps, pow, short termΔ pow, $\Delta\Delta$ pow 12 $\Delta\Delta$ ceps

Table 5.4: Naming Convetion table for feature type s3_1x39

Sphinx 3	s3_1x39
SphinxTrain , simplified	1s_12c_12d_3p_12dd
SphinxTrain , generic	c/1..L-1/d/1..L-1/c/0/d/0/dd/0/dd/1..L-1/
vector length	39
feature layout	12 ceps, 12 Δ ceps, pow, Δ pow, $\Delta\Delta$ pow 12 $\Delta\Delta$ ceps

Table 5.5: Naming Convetion table for feature type 1s_c_d_dd

Sphinx 3	1s_c_d_dd
SphinxTrain , simplified	1s_c_d_dd
SphinxTrain , generic	c/0..L-1/d/0..L-1/dd/0..L-1/
vector length	39
feature layout	13 ceps (*i.e.* power + 12 cepstra), 13 Δ ceps, 13 $\Delta\Delta$ ceps

5.7.1 Generic and simpilified notations in SphinxTrain

Because of code legacy, SphinxTrain actually maintained two sets of notations that are equivalent but written differntly. The following table will allow one to inter-convert them.

5.7.2 Formulae for Dynamic Coefficient

Let $x[i]$ be the incoming cepstra at time i, $c[i]$, the cepstral vector, $d[i]$, the delta cepstral vector, *i.e.* the first derivative, and $dd[i]$, the delta delta cepstral vector, *i.e.* the second derivative, all at time i. The dynamic coefficients for $s3\_1x39$ and $1s\_c\_d\_dd$ are computed as:

$$c[i] = x[i]$$

$$d[i] = x[i+2] - x[i-2]$$

$$dd[i] = d[i+1] - d[i-1] = (x[i+3] - x[i-1]) - (x[i+1] - x[i-3])$$

5.7.3 Handling of Initial and Final Position of an Utterance

The above formulae will require that, for every frame, the previous 3 frames and the future 3 frames exist. So, special treatment is required to pad 3 frames at the beginning of an utterance and 3 frames at the end of an

Table 5.6: Generic and Simplified notations fro parameters in SphinxTrain

Simplified	Generic	sphinx3
c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/	4s_12c_12d_3p_12dd	s2_4x
c/1..L-1/d/1..L-1/c/0/d/0/dd/0/dd/1..L-1/	1s_12c_12d_3p_12dd	s3_1x39
c/0..L-1/d/0..L-1/dd/0..L-1/	1s_c_d_dd	1s_c_d_dd
c/0..L-1/d/0..L-1/	1s_c_d	-
c/0..L-1/	1s_c	-
c/0..L-1/dd/0..L-1/	1s_c_d	-

utterance. Sphinx 3 .x takes care of it by replicating the first three frames of an utterance once. The sequence of features (*i.e.* $c[i]$, $d[i]$, $dd[i]$) will be delayed relative to the incoming cepstra. Moreover, the first couple of derivative coefficients are computed as if the sequence of frames had been converted from:

$x[0]x[1]x[2]x[3]x[4]x[5]\ldots$

to:

$x[1]x[2]x[3]x[0]x[1]x[2]x[3]x[4]x[5]\ldots$

[2]

Therefore, when actually viewing the frames (say dumping the code by printf), the first frame will have the delta and delta delta coefficients to be zero, whereas the second frame will have the delta coefficients to be zero.

5.8 Cautions in Using the Front End

Several common pitfalls of using the front end.

[2] In older version of version of Sphinx 3 , (Sphinx 3 .5 and before), we used to follow the convention $x[0]x[0]x[0]x[1]x[2]x[3]\ldots$ instead. We changed mainly for consistency and experimentally the new convention gave us slightly better results.

- The user is advised to use dither (**-dither**) for waveform decoding. In Sphinx 3 .X's wave2feat, this process is completely repeatable and it is controlled by the option **-seed**. The dither is useful because frames filled with zeros could cause the front end to run into a divide by zero condition.

- The user is strongly advised to match the front end parameters in both decoding and training. [3]

5.9 Compatibility of Front Ends in Different Versions

The cepstral generation source code is shared by Sphinx 2 , Sphinx 3 and SphinxTrain . However, the dynamic coefficients routine in Sphinx 3 only supports types s3_1x39 and 1s_c_d_dd. Moreover, Sphinx 2 is hardwired to use the type s2_4x. Therefore, if you are training models for Sphinx 2 , you will have to use this same type in SphinxTrain .

5.10 cepview

One useful tool for viewing the cepstral coefficient is **cepview**. It can be found in Sphinx 3 .X (X $>$ 5).

For example, the following command can be used to view the feature file **a.mfcc**.

```
$ cepview -d 13 -describe 1 -header 1 -f a.mfcc
```

You will see

```
$ 329 frames
 frame#:  c[ 0] c[ 1] c[ 2] c[ 3] c[ 4] c[ 5] c[ 6] c[ 7] ...
  0:  5.916 -0.248 0.128 0.083 -0.143 0.254 -0.012 -0.327 ...
  1:  5.668 -0.203 0.235 0.123 -0.184 0.145 0.031 -0.053 ...
```

[3] There are workarounds if the sampling frequency used in the training is higher than in the decoding. However, the upper and lower frequency in the filter bank need to scale for the decoding's frequency in that case. Avoid this unless you know what you are doing..

```
2:  5.371 -0.159 0.271 0.001 -0.055 0.242 -0.011 -0.071 ...
3:  5.499 -0.182 0.301 0.071 0.093 -0.036 -0.263 -0.164 ...
4:  5.447 -0.307 0.241 -0.067 -0.071 0.177 -0.050 -0.070 ...
5:  5.515 -0.070 0.062 -0.113 -0.008 0.180 -0.079 -0.073 ...
6:  5.431 -0.107 -0.007 -0.133 -0.037 0.137 -0.145 -0.089 ...
7:  5.502 -0.094 0.162 -0.142 -0.047 0.077 -0.073 -0.086 ...
```

Note: because of legacy reason, mainly to allow all the parameters to fit in a console which is 80 x60, the default number of parameters displayed are 10. Please beware that it doesn't mean that the cepstral vector only has 10 parameters.

Chapter 6

General Software Description of SphinxTrain

Author: *Arthur Chan*, Editor: *Arthur Chan*

The acoustic model training package, **SphinxTrain** , is a powerful set of routines that could be used to train several types of hidden Markov model for speech recognition.

As as Aug 2005, SphinxTrain consists of 35 C-based applications and 9 perl scripts. For many practitioners, its enormous size was the core difficulty to learn it. This chapter provides an overview of them and it is recommended for the users to read this chapter before reading Chapter 7: "Acoustic Model Training".

6.1 On-line Help of SphinxTrain

Tools of SphinxTrain have coherent design. They behave the same in the application level and there are common commands which would help the users to use them.

First of all, if the user just invoke the command without inputing any command-line arguments. Then a list of help for the command-line options will be displayed.

For example, for the tool printp, we would have

```
$ printp
```

```
[Switch] [Default] [Description]
-help no Shows the usage of the tool
-example no Shows example of how to use the tool
-tmatfn The transition matrix parameter file name
-mixwfn The mixture weight parameter file name
-mixws Start id of mixing weight subinterval
-mixwe End id of mixing weight subinterval
-gaufn A Gaussian parameter file name
-gaucntfn A Gaussian parameter weighted vector file
-regmatcntfn MLLR regression matrix count file
-moddeffn The model definition file
-lambdafn The interpolation weight file
-lambdamin 0 Print int.  wt.  >= this
-lambdamax 1 Print int.  wt.  <= this
-norm yes Print normalized parameters
-sigfig 4 Number of significant digits in 'e' notation
```

If the user needs further help, each routine provides two important mechanism for the user. One is the option **-help** and one is the option **-example**. The first one give a written description of each tool. For example,

```
printp -help yes
```

will produce the current description of the tool

```
./bin.i686-pc-linux-gnu/printp
 -help yes


Description:
Display numerical values of resources generated by Sphinx
Current we supoort the following formats
-tmatfn :  transition matrix
-mixwfn :  mixture weight file
-gaufn :  mean or variance
-gaucntn :  sufficient statistics for mean and diagonal covarian
-lambdafn :  interpolation weight
```

```
  Currently, some parameters can be specified as intervals such as mixtu
  You can also specified -sigfig the number of significant digits by you
  and normalize the parameters by -norm
```

whereas

```
$ printp -example yes
```

will produce

```
  Example:
  Print the mean of a Gaussian:
  printp -gaufn mean
  Print the variance of a Gaussian:
  printp -gaufn var
  Print the sufficient statistic:
  printp -gaucntfn gaucnt:
  Print the mixture weights:
  printp -mixw mixw
  Print the interpolation weight:
  printp -lambdafn lambda
```

The help and example message of SphinxTrain 's tools are also generated automatically and converted into instruction manual which one could find in the appendix A.2 of this manual.

6.2 Software Architecture of SphinxTrain

Author: *Arthur Chan*, Editor: *Arthur Chan*

Based on them, a 9-step procedure usually called "SphinxTrain perl script" was built. If one look at them, it looks like

6.2.1 General Description of the C-based Applications

As at Aug 2005, there are 35 tools in SphinxTrain , they are

```
agg_seg init_mixw mk_model_def mk_s3tmat
```

```
bldtree kmeans_init mk_s2cb mk_ts2cb
bw make_quests mk_s2hmm mllr_solve tiestate
cp_parm map_adapt mk_s2phone mllr_transform wave2feat
delint mixw_interp mk_s2phonemap norm
dict2tri mk_flat mk_s2sendump param_cnt
inc_comp mk_mdef_gen mk_s3gau printp
init_gau mk_mllr_class mk_s3mixw prunetree
```

[1]

It is perhaps very hard to describe them one by one without a proper categorization. Here is one categorization that was widely used.

1. Model conversion routines

   That includes **mk_s2cb**, **mk_s2hmm**, **mk_s2phone**, **mk_s2phonemap**, **mk_s2sendump**, they are for Sphinx 3 models format to Sphinx 2 model format.

   Also **mk_s3gau**, **mk_s3mixw**, **mk_s3tmat**, **mk_ts2cb**, they are for conversion of Sphinx 2 model format to Sphinx 3 model format.

2. Model manipulation routines

   That includes **printp**, which could display acoustic model parameters as well some important data structure.

   cp_parm which could copy model parameters.

   inc_comp which could increase the number of mixture components.

   mk_model_def, **mk_mdef_gen** which could create the model definition file.

3. Model adaptation routines

   mk_mllr_class which could create regression classes.

   mllr_transform which could transform a set of model based on a set of regression classes.

   map_adapt which could adapt a set of model based on the maximum a posterior (MAP) criterion.

   mllr_solve which could compute the regression matric based on sufficient statistics of the adaptation data.

4. Feature extraction routine **wave2feat** which could transform a wave file to cepstral coefficient.

[1]The tool **QUICK_COUNT** also exists, but it is largely replaced by dict2tri

5. Desicion tree mainipulation routine

   bldtree which could create a decision tree based on a set of questions.

   tiestate which could tie the state based on the decision trees.

   prunetree which could prune a decision tree to make it has a more reasonable size.

   make_quests which could automatically generate a set of questions.

6. Model Initialization routines **mk_flat** would could be used for flat initialization **init_gau, init_mixw** which could be used for ?? **agg_seg, kmeans_init** which could be used for mainipulation .

7. Model Estimation routines

   bw which could be used to carry out Baum-Welch estimation and generate the sufficient statistics..

   norm which could collect the sufficient statistics and generate the model.

   delint which could carry out deleted interpolation.

   mixw_interp which could carry out mixture weight interpolation

8. Miscellaneous routines **dict2tri** which could convert the transcription to triphone.

   param_cnt which could count the parameters out of several resources.

6.2.2 General Description of the Perl-based Tool

```
00.verify 06.prunetree
   01.vector_quantize 07.cd-schmm
   02.ci_schmm 08.deleted-interpolation
   03.makeuntiedmdef 09.make_s2_models
   04.cd_schmm_untied
   05.buildtrees
```

The current script in SphinxTrain could carry out both fully-continous HMM training and semi-continous training. The user is advised to first follow a recipe of training before they start from scratch. One well-maintained tutorial could be found at

http://www.cs.cmu.edu/ robust/Tutorial/opensource.html

6.3 Basic Acoustic Models Format in Sphinx-Train

In all Sphinx(en)(Sphinx 2 , Sphinx 3 and Sphinx 4), the basic acoustic model unit is the **HMM set**. However, for efficieny purpose, **the HMM set** is usually decomposed into several components as in representation. This is quite different from other software package such as HTK and ISIP recognizer.

For example, in Sphinx 3 , the models are represented by a set of means, variances, mixture weights, transition matrices and the model . definition. These method of decomposition allows researchers to just manipulate one type of quantity instead of all files.

You could find more information of the acoustic model format in the next Section

6.4 Sphinx data and model formats

6.4.1 Sphinx 2 data formats

Note: The data format described here is obsolete

Feature set: This is a binary file with all the elements in each of the vectors stored sequentially. The header is a 4 byte integer which tells us how many floating point numbers there are in the file. This is followed by the actual cepstral values (usually 13 cepstral values per frame, with 10ms skip between adjacent frames. Framesize is usually fixed and is usually 25ms).

```
<4_byte_integer header>
vec 1 element 1
vec 1 element 2
.
.
vec 1 element 13
vec 2 element 1
vec 2 element 2
```

```
.
.
vec 2 element 13
```

6.4.2 Sphinx 2 Model formats

Author: *Rita Singh and Arthur Chan*, Editor: *Arthur Chan*

Sphinx 2 semi-continuous HMM (SCHMM) formats:

The Sphinx 2 SCHMM format is rather complicated. It has the following main components (each of which has sub-components):

1. A set of codebooks
2. A "sendump" file that stores state (senone) distributions
3. A "phone" and a "map" file which map senones on to states of a triphone
4. A set of ".chmm" files that store transition matrices

Here is their description

Codebooks

There are 8 codebook files. The sphinx-2 uses a four stream feature set:

- cepstral feature: [c1-c12], (12 components)
- delta feature: [delta_c1-delta_c12,longterm_delta_c1-longterm_delta_c12],(24 components)
- power feature: [c0,delta_c0,doubledelta_c0], (3 components)
- doubledelta feature: [doubledelta_c-doubledelta_c12] (12 components)

The 8 codebooks files store the means and variances of all the gaussians for each of these 4 features. The 8 codebooks are,

- cep.256.vec [this is the file of means for the cepstral feature]
- cep.256.var [this is the file of variacnes for the cepstral feature]

- d2cep.256.vec [this is the file of means for the delta cepstral feature]
- d2cep.256.var [this is the file of variances for the delta cepstral feature]
- p3cep.256.vec [this is the file of means for the power feature]
- p3cep.256.var [this is the file of variances for the power feature]
- xcep.256.vec [this is the file of means for the double delta feature]
- xcep.256.var [this is the file of variances for the double delta feature]

All files are binary and have the following format: [4 byte int][4 byte float][4 byte float][4 byte float]..... The 4 byte integer header stores the number of floating point values to follow in the file. For the cep.256.var, cep.256.vec, xcep.256.var and xcep.256.vec this value should be 3328. For d2cep.* it should be 6400, and for p3cep.* it should be 768. The floating point numbers are the components of the mean vectors (or variance vectors) laid end to end. So cep.256.[vec,var] have 256 mean (or variance) vectors, each 13 dimensions long, d2cep.256.[vec,var] have 256 mean/var vectors, each 25 dimensions long, p3cep.256.[vec,var] have 256 vectors, each of dimension 3, xcep.256.[vec,var] have 256 vectors of length 13 each.

The 0th component of the cep,d2cep and xcep distributions are not used in likelihood computation and are part of the format for purely historical reasons.

The “sendump” file

The ”sendump” file stores the mixture weights of the states associated with each phone. (this file has a little ascii header, which might help you a little). Except for the header, this is a binary file. The mixture weights have all been transformed to 8 bit integer by the following operation intmixw = (-log(float mixw) ¿¿ shift) The log base is 1.0003. The ”shift” is the number of bits the smallest mixture weight has to be shifted right to fit in 8 bits. The sendump file stores,

```
for each feature (4 features in all)
 for each codeword (256 in all)
 for each ci-phone (including noise phones)
 for each tied state associated with ci phone,
 probability of codeword in tied state
```

```
    end
    for each CI state associated with ci phone, ( 5 states )
    probability of codeword in CI state
    end
    end
    end
   end
```

The sendump file has the following storage format (all data, except for the header string are binary):

Length of header as 4 byte int (including terminating '0')

```
    HEADER string (including terminating '0')
    0 (as 4 byte int, indicates end of header strings).
    256 (codebooksize, 4 byte int)
    Num senones (Total number of tied states, 4 byte int)
    [lut[0], (4 byte integer, lut[i] = -(i"<<"shift))
    prob_of_codeword[0]_of_feat[0]_1st_CD_sen_of_1st_ciphone (uchar)
    prob_of_codeword[0]_of_feat[0]_2nd_CD_sen_of_1st_ciphone (uchar)
    ..
    prob_of_codeword[0]_of_feat[0]_1st_CI_sen_of_1st_ciphone (uchar)
    prob_of_codeword[0]_of_feat[0]_2nd_CI_sen_of_1st_ciphone (uchar)
    ..
    prob_of_codeword[0]_of_feat[0]_1st_CD_sen_of_2nd_ciphone (uchar)
    prob_of_codeword[0]_of_feat[0]_2nd_CD_sen_of_2nd_ciphone (uchar)
    ..
    prob_of_codeword[0]_of_feat[0]_1st_CI_sen_of_2st_ciphone (uchar)
    prob_of_codeword[0]_of_feat[0]_2nd_CI_sen_of_2st_ciphone (uchar)
    ..
    ]
    [lut[1], (4 byte integer)
    prob_of_codeword[1]_of_feat[0]_1st_CD_sen_of_1st_ciphone (uchar)
    prob_of_codeword[1]_of_feat[0]_2nd_CD_sen_of_1st_ciphone (uchar)
```

```
..
prob_of_codeword[1]_of_feat[0]_1st_CD_sen_of_2nd_ciphone (uchar)
prob_of_codeword[1]_of_feat[0]_2nd_CD_sen_of_2nd_ciphone (uchar)
..
]
...  256 times ..
```

Above repeats for each of the 4 features

PHONE file

The phone file stores a list of phones and triphones used by the decoder. This is an ascii file It has 2 sections. The first section lists the CI phones in the models and consists of lines of the format For example:

```
AA 0 0 8 8
```

"AA" is the CI phone, the first "0" indicates that it is a CI phone, the first 8 is the index of the CI phone, and the last 8 is the line number in the file. The second 0 is there for historical reasons.

The second section lists TRIPHONES and consists of lines of the format

```
A(B,C)P -1 0 num num2
```

"A" stands for the central phone, "B" for the left context, and "C" for the right context phone. The "P" stands for the position of the triphone and can take 4 values "s","b","i", and "e", standing for single word, word beginning, word internal, and word ending triphone. The -1 indicates that it is a triphone and not a CI phone. num is the index of the CI phone "A", and num2 is the position of the triphone (or ciphone) in the list, essentially the number of the line in the file (beginning with 0).

The "map" file

The "map" file stores a mapping table to show which senones each state of each triphone are mapped to. This is also an ascii file with lines of the form

```
AA(AA,AA)s<0> 4
AA(AA,AA)s<1> 27
AA(AA,AA)s<2> 69
```

```
AA(AA,AA)s<3> 78
AA(AA,AA)s<4> 100
```

The first line indicates that the 0th state of the triphone "AA" in the context of "AA" and "AA" is modelled by th 4th senone associated with the CI phone AA. Note that the numbering is specific to the CI phone. So the 4th senone of "AX" would also be numbered 4 (but this should not cause confusion)

chmm FILES

There is one *.chmm file per ci phone. Each stores the transition matrix associated with that partiular ci phone in following binary format. (Note all triphones associated with a ci phone share its transition matrix) (all numbers are 4 byte integers):

- -10 (a header to indicate this is a tmat file)
- 256 (no of codewords)
- 5 (no of emitting states)
- 6 (total no. of states, including non-emitting state)
- 1 (no. of initial states. In fbs8 a state sequence can only begin with state[0]. So there is only 1 possible initial state)
- 0 (list of initial states. Here there is only one, namely state 0)
- 1 (no. of terminal states. There is only one non-emitting terminal state)
- 5 (id of terminal state. This is 5 for a 5 state HMM)
- 14 (total no. of non-zero transitions allowed by topology)

And also

```
[0 0 (int)log(tmat[0][0]) 0] (source, dest, transition prob, source i
[0 1 (int)log(tmat[0][1]) 0]
[1 1 (int)log(tmat[1][1]) 1]
[1 2 (int)log(tmat[1][2]) 1]
[2 2 (int)log(tmat[2][2]) 2]
[2 3 (int)log(tmat[2][3]) 2]
```

```
[3 3 (int)log(tmat[3][3]) 3]
[3 4 (int)log(tmat[3][4]) 3]
[4 4 (int)log(tmat[4][4]) 4]
[4 5 (int)log(tmat[4][5]) 4]
[0 2 (int)log(tmat[0][2]) 0]
[1 3 (int)log(tmat[1][3]) 1]
[2 4 (int)log(tmat[2][4]) 2]
[3 5 (int)log(tmat[3][5]) 3]
```

There are thus 65 integers in all, and so each *.chmm file should be 65*4 = 260 bytes in size.

6.4.3 Sphinx 3 model formats

Sphinx 3 models are composed of 5 parts

- Model Definition
- Means
- Variances
- Mixture Weights
- Transition Matrices

Headers of all binary formats

Model Definition

[Editor Notes: Plainly copied from the recipe chapter, need to rewrite] Here is an example:

```
0.3
55 n_base
118004 n_tri
472236 n_state_map
2165 n_tied_state
165 n_tied_ci_state
```

```
55 n_tied_tmat
#
# Columns definitions
#base lft rt p attrib tmat ...  state id's ...
+BACKGROUND+ - - - filler 0 0 1 2 N
+BREATH+ - - - filler 1 3 4 5 N
+CLICKS+ - - - filler 2 6 7 8 N
+COUGH+ - - - filler 3 9 10 11 N
+FEED+ - - - filler 4 12 13 14 N
+LAUGH+ - - - filler 5 15 16 17 N
+NOISE+ - - - filler 6 18 19 20 N
+SMACK+ - - - filler 7 21 22 23 N
 +UH+ - - - filler 8 24 25 26 N
+UHUH+ - - - filler 9 27 28 29 N
 +UM+ - - - filler 10 30 31 32 N
 AA - - - n/a 11 33 34 35 N
 AE - - - n/a 12 36 37 38 N
 AH - - - n/a 13 39 40 41 N
 AO - - - n/a 14 42 43 44 N
 AW - - - n/a 15 45 46 47 N
 AX - - - n/a 16 48 49 50 N
 AXR - - - n/a 17 51 52 53 N
 AY - - - n/a 18 54 55 56 N
 B - - - n/a 19 57 58 59 N
 CH - - - n/a 20 60 61 62 N
 D - - - n/a 21 63 64 65 N
 DH - - - n/a 22 66 67 68 N
 DX - - - n/a 23 69 70 71 N
 EH - - - n/a 24 72 73 74 N
 ER - - - n/a 25 75 76 77 N
 EY - - - n/a 26 78 79 80 N
```

```
F - - - n/a 27 81 82 83 N
G - - - n/a 28 84 85 86 N
HH - - - n/a 29 87 88 89 N
.     .     .
```

The first few line describes the number of parameters in the whole system

```
0.3
```

0.3 is the model format version.

```
55 n_base
```

n_base is the number of base phone including fillers in the system. In this case, the number is 55.

```
118004 n_tri
```

n_tri is the number of triphone in the system. In this case, the number is 118004.

```
472236 n_state_map
```

```
2165 n_tied_state
```

n_tied_state is the number of triphone in the system. In this case, the number is 118004.

```
165 n_tied_ci_state
```

n_tied_ci_state is the number tied CI state in the system. In this case, the number is 165

```
55 n_tied_tmat
```

n_tied_tmat is the number tied transition matrix in the system. In this case, the number is 55.

Now let us try to interpret the following line.

```
#base lft rt p attrib tmat ...  state id's ...
```

```
+BACKGROUND+ - - - filler 0 0 1 2 N
```

From left to right, it reads the base phone is +BACKGROUND+, no left context and no right context, so it is a CI phone. It is a filler. Its phone ID is 0, its first state has senone ID 0, second state 1, third state 2.

It is quite useful to know how the hmm state would map into a senone. (In HTK terminology, the tied-state.) In the above example, if you know that the senone ID for first state is 0. You just need to look up the file means and variances to get the value of them.

```
AA - - - n/a 11 33 34 35 N
```

Here is another entry in the model definition file. This time, we see a typical CI phone entry. If you used the standard training procedure of SphinxTrain (that you will also have a chance to play with in the second half of this chapter), then all the CI phone HMM's state will have their own senone which is unique for it.

As opposed to that, if you look at a triphone definition.

```
AA AA AA s n/a 11 177 190 216 N

AA AA AE s n/a 11 177 190 216 N

AA AA AH s n/a 11 177 190 216 N

AA AA AO s n/a 11 177 190 216 N

AA AA AW s n/a 11 177 190 216 N

AA AA AX s n/a 11 177 190 216 N

AA AA AXR s n/a 11 177 196 216 N

AA AA AY s n/a 11 177 190 216 N

AA AA B b n/a 11 178 189 219 N

AA AA B s n/a 11 177 189 217 N

AA AA CH s n/a 11 177 189 216 N

AA AA D b n/a 11 178 198 219 N

AA AA D s n/a 11 177 198 214 N
```

You will see multiple phones states are actually mapped to the same senone ID. Why does that happen? This is essentially the result of tying happend in training the context-dependent model. What happen is there are always not enough training data for triphone model. Therefore, it is necessary to cluster the hmm-state to make data could be more efficiently used. Most of the time, clustering are either done by decision tree-base clustering or bottom-up agglomerative clustering.

Means, Variances, Mixture Weights and Transition Matrices

[Editor Notes: Need to fill in more information here.] Please read the chapter on recipe for further detail.

6.4.4 Sphinx 4 model formats

To create a Sphinx 4 model, a common way is generate a Sphinx 3 models through SphinxTrain and convert it to Sphinx 4 format. One could find detail information at

http://cmusphinx.sourceforge.net/sphinx4/doc/UsingSphinxTrainModels.

Chapter 7

Acoustic Model Training

[Editor Notes: At this draft, this part is not yet finished. We would like the user to read the training page in cmusphinx.org first.]

Chapter 8

Language Model Training

Author: *Roni Rosenfeld and Philip Clarkson*, Editor: *Arthur Chan*

[Editor Notes: Plain copy from Philip's web site. Pretty dangerous to just open source it.]

8.1 Installation of the Toolkit

For "big-endian" machines (eg those running HP-UX, IRIX, SunOS, Solaris) the installation procedure is simply to change into the src/ directory and type

```
$ make install
```

The executables will then be copied into the **bin/** directory, and the library file **SLM2.a** will be copied into the **lib/** directory. For "little-endian" machines (eg those running Ultrix, Linux) the variable BYTESWAP_FLAG will need to be set in the Makefile. This can be done by editing src/Makefile directly, so that the line

```
#BYTESWAP_FLAG = -DSLM_SWAP_BYTES
```

is changed to

```
BYTESWAP_FLAG = -DSLM_SWAP_BYTES
```

Then the program can be installed as before.

If you are unsure of the "endian-ness" of your machine, then the shell script endian.sh should be able to provide some assistance.

In case of problems, then more information can be found by examining src/Makefile.

Before building the executables, it might be worth adjusting the value of STD_MEM in the file src/toolkit.h. This value controls the default amount of memory (in MB) that the programs will attempt to assign for the large buffers used by some of the programs (this value can, of course, be overridden at the command line). The result is that the final process sizes will be a few MB bigger than this value. The more memory that can be grabbed, the faster the programs will run. The default value is 100, but if the machines which the tools will be run on contain less, or much more memory than this, then this value should be adjusted to reflect this.

8.2 Terminology and File Formats

Text stream	An ASCII file containing text. It may or may not have markers to indicate context cues, and white space can be used freely.	.text
Word frequency file	An ASCII file containing a list of words, and the number of times that they occurred. This list is not sorted; it will generally be used as the input to wfreq2vocab, which does not require sorted input.	.wfreq
Word n-gram file	ASCII file containing an alphabetically sorted list of n-tuples of words, along with the number of occurrences	.w3gram, .w4gram etc.
Vocabulary file	ASCII file containing a list of vocabulary words. Comments may also be included - any line beginning ## is considered a comment. The vocabulary is limited in size to 65535 words.	.vocab.20K, .vocab.60K etc., depending on the size of the vocabulary.
Context cues file	ASCII file containing the list of words which are to be considered "context cues". These are words which provide useful context information for the n-grams, but which are not to be predicted by the language model. Typical examples would be ¡s¿ and ¡p¿, the begin sentence, and begin paragraph tags.	.ccs
Id n-gram file	ASCII or binary (by default) file containing a numerically sorted list of n-tuples of numbers, corresponding to the mapping of the word n-grams relative to the vocabulary. Out of vocabulary (OOV) words are mapped to the number 0. .id3gram.bin, .id4gram.ascii etc. Binary language model file Binary file containing all the n-gram counts, together with discounting information and back-off weights. Can be read by evallm and used to generate word probabilities quickly.	.binlm
ARPA language model file	ASCII file containing the language model probabilities in ARPA-standard format.	.arpa
Probability stream	ASCII file containing a list of probabilities (one per line). The probabilities correspond the the probability for each word in a specific text stream, with context-cues and OOVs removed.	.fprobs
Forced back-off file	ASCII file containing a list of vocabulary words from which to enforce back-off, together with either an 'i' or an 'e' to indicate inclusive or exclusive forced back-off respectively.	.fblist

These files may all be written are read by all the tools in compressed or uncompressed mode. Specifically, if a filename is given a .Z extension, then it will be read from the specified file via a zcat pipe, or written via a compress pipe. If a filename is given a .gz, it will be read from the specified file via a gunzip pipe, or written via a gzip pipe. If either of these compression schemes are to be used, then the relevant tools (ie zcat, and compress or gzip) must be available on the system, and pointed to by the path.

If a filename argument is given as - then it is assumed to represent either the standard input, or standard output (according to context). Any file read from the standard input is assumed to be uncompressed, and therefore, all desired compression and decompression should take place in a pipe:

```
$ zcat < abc.Z | abc2xyz | compress > xyz.Z
```

8.3 Typical Usage

Given a large corpus of text in a file a.text, but no specified vocabulary.

- Compute the word unigram counts

  ```
  $ cat a.text | text2wfreq > a.wfreq
  ```

- Convert the word unigram counts into a vocabulary consisting of the 20,000 most common words

  ```
  $ cat a.wfreq | wfreq2vocab -top 20000 > a.vocab
  ```

- Generate a binary id 3-gram of the training text, based on this vocabulary

  ```
  $ cat a.text | text2idngram -vocab a.vocab > a.idngram
  ```

- Convert the idngram into a binary format language model

  ```
  $ idngram2lm -idngram a.idngram -vocab a.vocab -binary a.binlm
  ```

- Compute the perplexity of the language model, with respect to some test text b.text

  ```
  evallm -binary a.binlm
  Reading in language model from file a.binlm
  Done.
  evallm :  perplexity -text b.text
  ```

```
Computing perplexity of the language model with respect
to the text b.text
Perplexity = 128.15, Entropy = 7.00 bits
Computation based on 8842804 words.
Number of 3-grams hit = 6806674 (76.97%)
Number of 2-grams hit = 1766798 (19.98%)
Number of 1-grams hit = 269332 (3.05%)
1218322 OOVs (12.11%) and 576763 context cues were removed from the c
evallm :  quit
```

Alternatively, some of these processes can be piped together:

```
cat a.text | text2wfreq | wfreq2vocab -top 20000 > a.vocab
cat a.text | text2idngram -vocab a.vocab |
 idngram2lm -vocab a.vocab -idngram -
 -binary a.binlm -spec_num 5000000 15000000
echo "perplexity -text b.text" | evallm -binary a.binlm
```

8.4 Discounting Strategies

Discounting is the process of replacing the original counts with modified counts so as to redistribute the probability mass from the more commonly observed events to the less frequent and unseen events. If the actual number of occurrences of an event E (such as a bigram or trigram occurrence) is $c(E)$, then the modified count is $d(c(E))c(E)$, where $d(c(E))$ is known as the discount ratio.

8.4.1 Good Turing discounting

Good Turing discounting defines $d(r) = \frac{(r+1)n(r+1)}{rn(r)}$ where $n(r)$ is the number of events which occur r times.

The discounting is only applied to counts which occur fewer than K times, where typically K is chosen to be around 7. This is the "discounting range" which is specified using the **-disc_ranges** parameter of the idngram2lm program.

For further details see "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer", Slava M. Katz, in "IEEE Transactions on Acoustics, Speech and Signal Processing", volume ASSP-35, pages 400-401, March 1987.

8.4.2 Witten Bell discounting

The discounting scheme which we refer to here as "Witten Bell discounting" is that which is referred to as type C in "The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression", Ian H. Witten and Timothy C. Bell, in "IEEE Transactions on Information Theory, Vol 37, No. 4, July 1991".

The discounting ratio is not dependent on the event's count, but on t, the number of types which followed the particular context. It defines $d(r,t) = \frac{n}{(n+t)}$, where n is the size of the training set in words. This is equivalent to setting $P(w|h) = \frac{c}{(n+t)}$ (where w is a word, h is the history and c is the number of occurrences of w in the context h), for events that have been seen, and $P(w|h) = \frac{t}{(n+t)}$ for unseen events. Absolute discounting

Absolute discounting defines $d(r) = \frac{(r-b)}{r}$. Typically $b = \frac{n(1)}{(n(1)+2n(2))}$. The discounting is applied to all counts.

This is, of course, equivalent to simply subtracting the constant b from each count.

8.4.3 Linear discounting

Linear discounting defines $d(r) = 1 - \frac{(n(1)}{/}C$, where C is the total number of events. The discounting is applied to all counts.

For further details of both linear and absolute discounting, see "On structuring probabilistic dependencies in stochastic language modeling", H. Ney, U. Essen and R. Kneser in "Computer Speech and Language", volume 8(1), pages 1-28, 1994.

Chapter 9

Search structure and Speed-up of the speech recognizer

9.1 Introduction

Author: *Arthur Chan, Ravi Mosur*, Editor: *Arthur Chan*

[Editor Notes: This is largely adapted from Ravi Mosur's code-walk document file.]

In this chapter, we will describe the search structure of the recognizers in Sphinx 3 . There are several recognizers in the standard Sphinx 3 's distribution package. Each of them can be used for certain particular purpose. It is very important to understand their differences and apply each of them correctly. For example, s3.X decode_anytopo in 2004 was designed to be used as an offline batch mode recognizer. It is designed to be *accurate* and also optimization was not applied on the recognizer. These are all done by design. If you accidentally used this recognizer for real-time application, I am sure you will lose you client. :-)

9.2 Sphinx 3 .X's recognizer general architecture

Conceptually, all recognizers in Sphinx 3 .X share the the same recognition engine. One can always view Sphinx 3 's search as two separated but interelated components.

1. GMM Computation module
2. Graph Search module

At every frame, GMM computation will compute all the active GMM that marked by the graph search module. The graph search module will make use of the GMM scores to do the graph searching. Since there is pruning mechanim in the search. Only part of the GMM will be found to be active in the next frame. The information on whether the GMM is active will be used by GMM computation module.

In a first pass system, this separation of GMM computation and search modules are found to be very useful. The major reason is that techniques can be applied to individual modules and speed performance can be benchmarked separately. This usually provides valuable insights for further improvement of the system

The history of the speech recognizer's development shows that there are two major inter-related goals of speech recognizer design. One is to optimize the accuracy of speech recognizer. One is to allow an known imperfect speech recognizer to be used in a practical scenario. Usually, this practical scenario impose constaints to the design of the recognizer such that the recognizer has to run in a limited amount of computation resource (such as amount of random access memory and the computation power). For example, NIST evaluation, a kind of competition between research sites, requires each site to provide the best performing recognition system. This basically requires users to fullfill the first goal. Usually, these systems are run-in around 20x-100xRT.

Sphinx 3 , follow this guide, were implemented under similar historical trend, the Sphinx 3 .0 recognizer (or s3slow , s3 accurate) are designed to be an accurate but perhaps slow recognizer. In 2004, running S3 on the Wall Street Journal 5000 Words Task require 10-12xRT of computation. [1] This is the most accurate among the Sphinx 3 's family of recognizers. Whereas, Sphinx 3.X (or s3fast) can run the task in 0.94xRT with around 5% relative degradation.

[1] 5000 senones, trigram

It is therefore understable that why two implementations of the speech recognition engine were happened in the first place. s3slow is always meant for research purpose and allow the most accurate results. s3fast is alway meant to be designed as a fast and perhaps application specific recognizer. Hence, one found only small amount of speed optimization technique appears in s3slow. However, one can see s3fast accept employ different techniques in general and allow the user to choose them.

There exists customized modifications of each of the recognizers. For example, Arthur Chan has written a version of s3 which employ fast GMM computation. This was unfortunately not suitable to be incorporated into the main trunk of the s3slow. One can get this information from sphinx developer's web page

www.cs.cmu.edu/~archan/

The conceptual model described about is not fixed and from time to time, we will revise it and possibly incorporate more interesting features to the recognizers.

9.3 Importance of tuning

It is wide spread misunderstanding that a single default generic speech models can be used in all different situations. It is also incorrect to assume that the default parameters used in speech recognition is the best in general. First-class speech recognition system was tuned intensively for both speed and accuracy. In this chapter, apart from just describing the principle of search. We will also describe parameters that can affect the operation of the parameters. Internal to CMU, this parameter was tuned through exhaustively trial and error testing using a small test set.

So how about you? We recommend you to read the following description before you start to do tuning. Usually, tuning is done by tuning one parameter at a time. It is also possible to device a kind of multi-dimensional search is possible for this kind of problem.

9.4 Sphinx 3 .X's decode_anytopo (or s3slow)

In one sentence, **decode_anytopo** is a generic tri-gram speech recognizer which accept the use of multi-stream continuous HMM with tied mixture

and with any topologies of the HMM. At 2004 Sep, you can get this recognizer at

```
$ cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/cmusphinx
co archive_s3/
```

What it means is that one can use the standard fully continuous HMM (1-stream. tied state) or semi-continuous HMM (4-stream, tied mixture) as an input of the model. The user is allowed to specify the grammar as the form N-gram, It is very generic and can be used in most of the situation.

We will describe several aspects of decode_anytopo in this chapter. Note that decode_anytopo doesn't contain too much optimization in the code as we already mentioned its purpose is more on benchmarking.

9.4.1 GMM Computation

Gaussian Mixture Model (GMM or senone in CMU's terminology) was computed as it is, usually only active GMM is computed at every frame. One special aspect of GMM computation is that its output is a scaled integer. Back to 1995, this is the only way that log values can be efficiently computed. Internally, the code is also optimized using loop-unrolling [2]. Other than these two techniques, that is no approximation at all.

9.4.2 Search structure

The data structure for including different words in search is arranged as flat lexicons. This is the standard way to arrange the lexicon and will not have bad effect on accuracy as the tree lexicon (as described in Section ?). However, not able to tied the prefixes of different words together will double the time of traversing lattice. This is part of the reason why decode_anytopo is at least 2 times slower than decode.

The score accumulation in the search is based on integer. Again, the reason is that computing the log value of a number is best to be done using lookup tables when the recognizer is first implemented. Even, till now, computing log value using standard C library is still not as fast as integer lookup table.

[2] Loop unrolling is a technique in C-programming that makes try to take advantage of parrallel scheduling in modern day central processor. It is a common way to optimize code in a for loop

9.4.3 Treatment of language model

decode_anytopo can take care of trigram in a single-pass search. However, exhaustive search for full-trigram is extremely expansive because it always requires storing the two previous words at a particular time. It will actually caused the storage require in search to become n^2 instead of n. Instead of doing so, Sphinx 3 used so called it poor man trigram in decoding. The idea is that for a given word w_n, only store one single previous word w_{n-1}. For w_{n-2}, only used the most likely previous word that derived from the Viterbi search. In early experiment back to the date of evaluation of Wall Street Journal (WSJ) task (93-96). This assumption is found to be a very good approximation (less than 5% degradation of accuracy). It was therefore employed and used until nowadays.

9.4.4 Triphone representation

[Under construction]

9.4.5 Viteri Pruning

Although decode_anytopo is designed to be used for evaluation, it also implemented standard pruning feature in the search. Major reason is that people long realized that fully exhaustive Viterbi search will search a lot of very unlikely paths. Viterbi Pruning is the simplest and the most effective way to reduce the number of paths search.

There are two beams used in decodeanytopo,

- **-beam** Main pruning beam applied to triphones in forward search
- **-nwbeam** Pruning beam applied in forward search upon word exit

9.4.6 Search Tuning

There are several important parameters in the search that can affect the accuracy.

- **-langwt** Language weight: empirical exponent applied to LM probaby

- **-ugwt** LM unigram weight: unigram probs interpolated with unifodistribution with this weight
- **-inspen** Word insertion penalty
- **-silpen** Language model 'probability' of silence word
- **-noisepen** Language model 'probability' of each non-silence filler d
- **-fillpenfn** Filler word probabilities input file (used in place of -pen and -noisepen)

9.4.7 2-nd pass Search

One can used decode_anytopo to become a building block of what so called the multi-pass system. This is based on generating a word lattice of hypothesis instead of generating a single hypothesis. Then rescoring is done based on the lattice. The options that control this properties are,

- **-bestpath** Whether to run bestpath DAG search after forward Viterss
- **-dagfudge** (0..2); 1 or 2: add edge if endframe == startframe; 2:tart == end-1
- **-bestpathlw** Language weight for bestpath DAG search (default: samelangwt)
- **-inlatdir** Input word-lattice directory with per-utt files for reting words searched
- **-inlatwin** Input word-lattice words starting within +/- ¡this arg¿ of current frame considered during search
- **-outlatdir** Directory for writing word lattices (one file/utterancptional ,NODES suffix to write only the nodes
- **-latext** Word-lattice filename extension (.gz or .Z extension fmpression)
- **-bestscoredir** Directory for writing best score/frame (used to set beth; one file/utterance)

We will describe the behavior of 2-nd pass search in the Section ?.

9.4.8 Debugging

There are many situations you would like to get internal information of the Sphinx 3 .0's recognizer.

- **-hmmdumpsf** Starting frame for dumping all active HMMs (for debuggiagnosis/analysis)
- **-worddumpsf** Starting frame for dumping all active words (for debug-diagnosis/analysis)
- **-logfn** Log file (default stdout/stderr)
- **-backtrace** Whether detailed backtrace information (word segmentat-cores) shown in log

9.5 Sphinx 3 .X's decode (aka s3 fast)

The so called Sphinx 3 .X's decode or s3fast was implemented 4 years after implementation of decode_anytopo. The major motivation in those days was to allow a recognizer can be ran in less than 10xRT for the Broadcast News task. This target is fullfilled in those days and it is estimated that with proper tuning, broadcast news may be able to performed under less than 1xRT in within the next 5 years even using single-pass search. [3]

At 2004, it will be safe to assume that the 1-pass search can be tuned to speed less than 1xRT for system with less than 10000 words with LM perplexity around 80.

The major reason why it can be much faster than decode_anytopo is because many techniques have been applied to speed-up the search. This section will describe each of these techniques in detail.

9.6 Architecture of Search in decode

Author: *Ravi Mosur and Arthur Chan*, Editor: *Arthur Chan*

[Editor Notes: this part largely copied from Ravi's sphinx 3.3 codewalk]

[3] Writing at 2004 Sep, at that time, a PC equipped with 3G Pentium can produce 3-4xRT performance in s3.X

9.6.1 Initialization

The decoder is configured during the initialization step, and the configuration holds for the entire run. This means, for example, that the decoder does not dynamically reconfigure the acoustic models to adapt to the input. To choose another example, there is no mechanism in this decoder to switch language models from utterance to utterance, unlike in Sphinx 2 . The main initialization steps are outlined below.

Log-Base Initialization. Sphinx 3 performs all likelihood computations in the log-domain. Furthermore, for computational efficiency, the base of the logarithm is chosen such that the likelihoods can be maintained as 32-bit integer values. Thus, all the scores reported by the decoder are log-likelihood values in this peculiar log-base. The default base is typically 1.0003, and can be changed using the -logbase configuration argument. The main reason for modifying the log-base would be to control the length (duration) of an input utterance before the accumulated log-likelihood values overflow the 32-bit representation, causing the decoder to fail catastrophically. The log-base can be changed over a wide range without affecting the recognition.

Models Initialization: The lexical, acoustic, and language models specified via the configuration arguments are loaded during initialization. This set of models is used to decode all the utterances in the input. (The language model is actually only partly loaded, since s3.X uses a disk-based LM strategy. Though it is also possible to load the LM into memory by setting **-lminmemory** to 1)

Effective Vocabulary: After the models are loaded, the effective vocabulary is determined. It is the set of words that the decoder is capable of recognizing. Recall that the decoder is initialized with three sources of words: the main and filler lexicon files, and the language model. The effective vocabulary is determined from them as follows:

- Find the intersection of the words in the LM and the main pronunciation lexicon
- Include all the alternative pronunciations to the set derived above (using the main lexicon
- Include all the filler words from the filler lexicon, but excluding the distinguished beginning and end of sentence words: <s> and </s>.

The effective vocabulary remains in effect throughout the batch run. Currently, it is not possible to add to or remove from this vocabulary dynamically, unlike in the Sphinx 3 system.

9.6.2 Lexicon representation

A pronunciation lexicon (or dictionary) file specifies word pronunciations. In Sphinx 3 , pronunciations are specified as a linear sequence of phonemes. Each line in the file contains one pronunciation specification, except that any line that begins with a "#" character in the first column is treated as a comment and is ignored. Example dictionary for digits:

ZERO Z IH R OW

ONE W AH N

TWO T UW

THREE TH R IY

FOUR F AO R

FIVE F AY V

SIX S IH K S

SEVEN S EH V AX N

EIGHT EY TD

NINE N AY N

The lexicon is completely case-insensitive (unfortunately). For example, it's not possible to have two different entries Brown and brown in the dictionary. Multiple Pronunciations

A word may have more than one pronunciation, each one on a separate line. They are distinguished by a unique parenthesized suffix for the word string. For example:

ACTUALLY AE K CH AX W AX L IY

ACTUALLY(2nd) AE K SH AX L IY

ACTUALLY(3rd) AE K SH L IY

If a word has more than one pronunciation, its first appearance must be the unparenthesized form. For the rest, the parenthesized suffix may be any string, as long as it is unique for that word. There is no other significance to the order of the alternatives; each one is considered to be equally likely. Compound Words

In Sphinx 3 , the lexicon may also contain compound words. A compound word is usually a short phrase whose pronunciation happens to differ significantly from the mere concatenation of the pronunciations of its constituent words. Compound word tokens are formed by concatenating the component word strings with an underscore character; e.g.:

WANT_TO W AA N AX

(The s3.X decoder, however, treats a compound word as just another word in the language, and does not do anything special with it.)

Tree representation

The decoder constructs lexical trees from the effective vocabulary described above. Separate trees are constructed for words in the main and filler lexicons. Furthermore, several copies may be instantiated for the two, depending on the **-Nlextree** configuration argument.

The lexical tree representation is motivated by the fact that Most active HMMs are word-initial models, decaying rapidly subsequently. e.g. On 60K-word Hub-4 task, 55word-initial. But, no. of distinct word-initial model types are much fewer:

START S-T-AA-R-TD

STARTING S-T-AA-R-DX-IX-NG

STARTED S-T-AA-R-DX-IX-DD

STARTUP S-T-AA-R-T-AX-PD

START-UP S-T-AA-R-T-AX-PD

Therefore, it makes sense to use so called the prefix-tree to represent the lexicon and maximizing sharing among words

[Under construction] [Insert p.21 of Ravi's slides at here]

Triphone representation

[Under construction] [Insert p.22 and p.23 of Ravi's slides at here]

Integration with language model

[Under construction] [Insert p.24-32 of Ravi's slides at here]

LM lookahead

One of the key of fast search is to make sure high level information can be used as soon as possible. Here comes the idea of

Unigram LM probability is factored into the tree using technique [Need to create. described in Spoken Language Dialogue

9.6.3 Language model

The main language model (LM) used by the Sphinx decoder is a conventional bigram or trigram backoff language model. The CMU-Cambridge LM Toolkit is capable of generating such a model from LM training data. Its output is an ascii text file. But a large text LM file can be very slow to load into memory. To speed up this process, the LM must be compiled into a binary form. The code to convert from an ascii text file to the binary format is available at SourceForge in the CVS tree, in a module named share.

Unigrams, Bigrams, Trigrams, LM Vocabulary

A trigram LM primarily consists of the following:

- Unigrams: The entire set of words in this LM, and their individual probabilities of occurrence in the language. The unigrams must include the special beginning-of-sentence and end-of-sentence tokens: <s>, and </s> respectively.
- Bigrams: A bigram is mathematically P(word2 — word1). That is, the conditional probability that word2 immediately follows word1 in the language. An LM typically contains this information for some subset of the possible word pairs. That is, not all possible word1 word2 pairs need be covered by the bigrams.
- Trigrams: Similar to a bigram, a trigram is P(word3 — word1, word2), or the conditional probability that word3 immediately follows a word1 word2 sequence in the language. Not all possible 3-word combinations need be covered by the trigrams.

The vocabulary of the LM is the set of words covered by the unigrams.

The LM probability of an entire sentence is the product of the individual word probabilities. For example, the LM probability of the sentence "HOW ARE YOU" is:

P(HOW — <s>) *

P(ARE — <s>, HOW) *

P(YOU — HOW, ARE) *

P(</s> — ARE, YOU)

Pronunciation and Case Considerations

In Sphinx 3 , the LM cannot distinguish between different pronunciations of the same word. For example, even though the lexicon might contain two different pronunciation entries for the word READ (present and past tense forms), the language model cannot distinguish between the two. Both pronunciations would inherit the same probability from the language model.

Secondly, the LM is case-insensitive. For example, it cannot contain two different tokens READ and read.

The reasons for the above restrictions are historical. Precise pronunciation and case information has rarely been present in LM training data. It would certainly be desirable to do away with the restrictions at some time in the future.

Binary LM File

The binary LM file (also referred to as the LM dump file) is more or less a disk image of the LM data structure constructed in memory. This data structure was originally designed during the Sphinx 2 days, when efficient memory usage was the focus. In Sphinx 3 , however, memory usage is no longer an issue since the binary file enables the decoder to use a disk-based LM strategy. That is, the LM binary file is no longer read entirely into memory. Rather, the portions required during decoding are read in on demand, and cached. For large vocabulary recognition, the memory resident portion is typically about 10-20the bigrams, and 5-10

Since the decoder uses a disk-based LM, it is necessary to have efficient access to the binary LM file. Thus, network access to an LM file at a remote location is not recommended. It is desirable to have the LM file be resident on the local machine.

The binary dump file can be created from the ascii form using the lm3g2dmp utility, which is part of the Sphinx 2 distribution, and also available as standalone code, as mentioned before. (The header of the dump file itself contains a brief description of the file format.)

Silence and Filler Words

Language models typically do not cover acoustically significant events such as silence, breath-noise, UM or UH sounds made by a person hunting for the right phrase, etc. These are known generally as filler words, and are excluded from the LM vocabulary. The reason is that a language model training corpus, which is simply a lot of text, usually does not include such information.

Since the main trigram LM ignores silence and filler words, their "language model probability" has to be specified in a separate file, called the filler penalty file. The format of this file is very straightforward; each line contains one word and its probability, as in the following example:

++UH++ 0.10792

++UM++ 0.00866

++BREATH++ 0.00147

The filler penalty file is not required. If it is present, it does not have to contain entries for every filler word. The decoder allows a default value to be specified for filler word probabilities (through the **-fillprob** configuration argument), and a default silence word probability (through the **-silprob** argument).

Like the main trigram LM, filler and silence word probabilities are obtained from appropriate training data. However, training them is considerably easier since they are merely unigram probabilities.

Filler words are invisible or transparent to the trigram language model. For example, the LM probability of the sentence "HAVE CAR <sil> WILL TRAVEL" is:

P(HAVE — <s>) *

P(CAR — <s>, HAVE) *

P(<sil>) *

P(WILL — HAVE, CAR) *

P(TRAVEL — CAR, WILL) *

P(</s> — WILL, TRAVEL)

Language Weight and Word Insertion Penalty

During recognition the decoder combines both acoustic likelihoods and language model probabilities into a single score in order to compare vari-

ous hypotheses. This combination of the two is not just a straightforward product. In order to obtain optimal recognition accuracy, it is usually necessary to exponentiate the language model probability using a language weight before combining the result with the acoustic likelihood. (Since likelihood computations are actually carried out in the log-domain in the Sphinx decoder, the LM weight becomes a multiplicative factor applied to LM log-probabilities.)

The language weight parameter is typically obtained through trial and error. In the case of Sphinx, the optimum value for this parameter has usually ranged between 6 and 13, depending on the task at hand.

Similarly, though with lesser impact, it has also been found useful to include a word insertion penalty parameter which is a fixed penalty for each new word hypothesized by the decoder. It is effectively another multiplicative factor in the language model probability computation (before the application of the language weight). This parameter has usually ranged between 0.2 and 0.7, depending on the task.

9.6.4 Pruning

Each entry in the control file, or utterance, is processed using the given input models, and using the Viterbi search algorithm. In order to constrain the active search space to computationally manageable limits, pruning is employed, which means that the less promising hypotheses are continually discarded during the recognition process. There are two kinds of pruning in s3.X, beam pruning and absolute pruning.

Viterbi Pruning or Beam Pruning

Each utterance is processed in a time-synchronous manner, one frame at a time. At each frame the decoder has a number of currently active HMMs to match with the next frame of input speech. But it first discards or deactivates those whose state likelihoods are below some threshold, relative to the best HMM state likelihood at that time. The threshold value is obtained by multiplying the best state likelihood by a fixed beamwidth. The beamwidth is a value between 0 and 1, the former permitting all HMMs to survive, and the latter permitting only the best scoring HMMs to survive.

Similar beam pruning is also used in a number of other situations in the decoder, e.g., to determine the candidate words recognized at any time, or to determine the component densities in a mixture Gaussian that are

closest to a given speech feature vector. The various beamwidths have to be determined empirically and are set using configuration arguments.

Histogram Pruning or Absolute Pruning

Absolute Pruning. Even with beam pruning, the number of active entities can sometimes become computationally overwhelming. If there are a large number of HMMs that fall within the pruning threshold, the decoder will keep all of them active. However, when the number of active HMMs grows beyond certain limits, the chances of detecting the correct word among the many candidates are considerably reduced. Such situations can occur, for example, if the input speech is noisy or quite mismatched to the acoustic models. In such cases, there is no point in allowing the active search space to grow to arbitrary extents. It can be contained using pruning parameters that limit the absolute number of active entities at any instant. These parameters are also determined empirically, and set using configuration arguments.

- **-beam**: Determines which HMMs remain active at any given point (frame) during recognition. (Based on the best state score within each HMM.)
- **-pbeam**: Determines which active HMM can transition to its successor in the lexical tree at any point. (Based on the exit state score of the source HMM.)
- **-wbeam**: Determines which words are recognized at any frame during decoding. (Based on the exit state scores of leaf HMMs in the lexical trees.)
- **-maxhmmpf**: Determines the number of HMMs (approx.) that can remain active at any frame.
- **-maxwpf**: Controls the number of distinct words recognized at any given frame.
- **-maxhistpf**: Controls the number of distinct word histories recorded in the backpointer table at any given frame.
- **-subvqbeam**: For each senone and its underlying acoustic model, determines its active mixture components at any frame.

In order to determine the pruning parameter values empirically, it is first necessary to obtain a test set, i.e., a collection of test sentences not

used in any training data. The test set should be sufficiently large to ensure statistically reliable results. For example, a large-vocabulary task might require a test set that includes a half-hour of speech, or more.

It is difficult to tune a handful of parameters simultaneously, especially when the input models are completely new. The following steps may be followed to deal with this complex problem.

1. To begin with, set the absolute pruning parameters to large values, making them essentially ineffective. Set both **-beam** and **-pbeam** to 1e-60, and **-wbeam** to 1e-30. Set **-subvqbeam** to a small value (e.g., the same as **-beam**). Run the decoder on the chosen test set and obtain accuracy results. (Use default values for the LM related parameters when tuning the pruning parameters for the first time.)

2. Repeat the decoder runs, varying **-beam** up and down, until the setting for best accuracy is identified. (Keep **-pbeam** the same as **-beam** every time.)

3. Now vary **-wbeam** up and down and identify its best possible setting (keeping **-beam** and **-pbeam** fixed at their most recently obtained value).

4. Repeat the above two steps, alternately optimizing **-beam** and **-wbeam**, until convergence. Note that during these iterations **-pbeam** should always be the same as **-beam**. (This step can be omitted if the accuracy attained after the first iteration is acceptable.)

5. Gradually increase **-subvqbeam** (i.e., towards 1.0 for a narrower setting), stopping when recognition accuracy begins to drop noticeably. Values near the default are reasonable. (This step is needed only if a sub-vector quantized model is available for speeding up acoustic model evaluation.)

6. Now gradually increase **-pbeam** (i.e., towards 1.0), stopping when recognition accuracy begins to drop noticeably. (This step is optional; it mainly optimizes the computational effort a little more.)

7. Reduce **-maxhmmpf** gradually until accuracy begins to be affected. Repeat the process with **-maxwpf**, and then with **-maxhistpf**. (However, in some situations, especially when the vocabulary size is small, it may not be necessary to tune these absolute pruning parameters.)

In practice, it may not always be possible to follow the above steps strictly. For example, considerations of computational cost might dictate that the absolute pruning parameters or the parameters of the GMM

computation first. Lets assume if we use Sub-vector quantization as the GMM computation technique. The following is a practical way to tune the paramters. [4]

9.6.5 Phoneme look-ahead

[Add Jahanzeb's idea of phoneme lookahead at here.]

9.7 Architecture of GMM Computation in decode

The technique described in this section can be found in "Four-Level Categorization Scheme of Fast GMM Computation Techniques in Large Vocabulary Continuous Speech Recognition Systems" by Arthur Chan. One can treat the computation of GMM as a 4-level process.

- **Frame-level**
- **GMM-level**
- **Gaussian-level**
- **Component-level**

Approximation can be done at every level such that accuracy will not be sacrificed. Several techniques were implemented in Sphinx

9.7.1 Frame-level optimization

Algorithms that decide whether a frame's GMM scores should be computed or skipped are categorized as frame-layer algorithms. In our discussion, we will assume the score of a skipped frame will be copied from the most recently computed frame.[5] The simplest example is to compute frame scores

[4] 20040927(Arthur): This section is copied from Ravi's codewalk. it will later be expanded such that **-ci_pbeam** , **-ds** will also be employed and used for tuning.

[5] This implementation can preserve transition information.

only every other frame, which we call this simple down-sampling (SDS). One can apply this technique in Sphinx 3 .X, by specifying **-ds N**.

Other than SDS, we also evaluate another scheme in which a VQ codebook is trained from all means of GMMs of a set of trained acoustic models. Then, in decoding, every frame's feature vector is quantized using that codebook. A frame is skipped only if its feature vector was quantized to a codeword which is the same as that of the previous frame. We call this method VQ-based Down-Sampling (VQDS). This can be used by **-cond_ds**, if this flag is specified, flag **-gs** is necessary to be applied.

Comparatively speaking, SDS provide more gain in speed but cause some degradation and it is very ideal for user who seek for speed.

9.7.2 GMM-level optimization

Algorithms that ignore some GMMs in the computation in each computed frame are assigned to the GMM-layer. One representative technique is context-independent (CI) GMM-based GMM selection (CIGMMS) experimented by Lee et al. in Julius. Briefly the algorithm can be implemented as follows.At every frame, CI GMMs' scores are first computed and a beam is applied to these scores. For all context-dependent (CD) GMMs, if the corresponding CI GMM's score is within the beam, compute the detail CD GMM's score. If not, the CD GMM's score is backed-off by the corresponding CI GMM's score.

This scheme is highly effective in reducing GMM computation. However, because some scores are backed-off, they become the same when they are fed into the search module. As a consequence, beam pruning becomes less effective.

This technique can be used by specified -ci_pbeam. The range can be specified is between 10^{-5} to 10^{-80}.

9.7.3 Gaussian-level optimization : VQ-based Gaussian Selection and SVQ-based Gaussian Selection

Usually only a few Gaussians will dominate the likelihood of a GMM and different techniques are used to decide which Gaussian dominates the likelihood computation. We categorize these techniques as part of the Gaussian layer. Generally a rough model, either vector-quantizer (VQ)

sub-vector-quantizer(SVQ) or kd-tree is first used to decide which Gaussian should be computed in the GMM. Hence, these techniques are also called Gaussian selection.

The major issue in using any Gaussian selection techniques is the need to trade-off between rough model computation (e.g. the VQ codebook) and accuracy degradation. Usually, a more detailed model (e.g., a higher-ordered VQ codebook) gives better accuracy, but results in more computation.

Usually, the neigborhood for a particular code word is computed by thresholding the distance of Gaussian mean from the codeword. (cite Bochierri's paper). The consequence of using this scheme is that there will be cases which a neigborhood contains no Gaussian at all. Hence, in work by Gales et. al. , it is suggested that one could fix this problem by setting a minimum number of Gaussian in a certain neigborhood. Note that in work by Douglas (cite in Douglas's work) suggested that instead of using the code word as a reference point of deciding the neighborhood, one could use the closest Gaussian's mean as the reference point. In this case, one can always use the closest point as one of the Gaussian in the neighborhood.

One can use -gsfn to specify the gaussian selector map. The Gaussian selector map can be generated by gselect.

In Sphinx 3 .X, we focused on two techniques which made use of simple VQ as a Gaussian selector (VQGS) and SVQ as a Gaussian selector (SVQGS). We ignored tree-based techniques because of the practical difficulty in then applying adaptation techniques such as Maximum Likelihood Linear Regression (MLLR). One can use -svqfn to specify sub-VQ map. The sub-vector quantization can be generated by **gausubvq**.

9.7.4 Gaussian/Component-level optimization : Sub-vector quantization

As in the full-feature space, the distribution of features in a projection of the full-space (or subspace) can be approximated as a GMM. The full-space likelihood can thus be obtained by summing individual sub-spaces likelihood. Similar to situation in the full-space, only few Gaussians will dominate the subspace likelihood in a particular subspace. Therefore, techniques can be used to choose the best Gaussians in individual sub-spaces and combined them. We categorize algorithms which make use of this fact to be component-layer algorithm. We will focus one representative technique in this layer, sub-vector quantization (SVQ) in which VQ

was used as a Gaussian selector in subspaces.

The sub-vector quantization can be generated by **gausubvq**,

```
$ gausubvq
  -mean ./hub4_cd_continuous_8gau_1s_c_d_dd/means
  -var ./hub4_cd_continuous_8gau_1s_c_d_dd/variances
  -mixw ./hub4_cd_continuous_8gau_1s_c_d_dd/mixture_weights
  -svspec 0-38
  -iter 20
  -svqrows 16
  -subvq svq.out
```

in a file called svq.out. This can be used in **decode**

9.7.5 Interaction between different level of optimization

Our categorization scheme results in a layered architecture for GMM computation and many fast GMM techniques can be categorized into one of the layers [6]. The advantage of this scheme is that one can follow a conceptual model when implementing different algorithms. It also simplifies the studies of interaction of different schemes. For example, once we understand that two techniques are in the same-layer (such as VQGS and SVQGS), we will probably not want to implement them in the same system, as that can only result in higher overhead.

9.7.6 Related tools, gs_select, gs_view, gausubvq

[To be completed]

9.7.7 Interaction between GMM computation and search routines in Sphinx 3 .X

The performance of fast GMM computation techniques is usually less effective when a tighter beam is used in search. From the perspective of our

[6] We also note that some techniques has the characteristics of multiple layers.

conceptual model, pruning can be regarded as a GMM-layer algorithm, and as such, only affects the layers above, namely, the GMM and frame layers.

We summarize the effect below.

Relationship with Frame-Layer: We assume skipped frames' scores are copied from previous frames' scores. However, the search module will decide whether a clustered-GMM is active or not based on pruning. There will be problematic situations where some GMMs are active in the current frame but deactivated in previous frame. Hence, recomputation of active states is necessary. When the beam is tight, recomputation can be time-consuming and can cancel out the computation gain obtained from down-sampling.[7]

Relationship with GMM-Layer: As the GMM's scores will feed into the search module, one observation is that if CIGMMS applied, the search time increases. If the CI scores' beam (mentioned in Section [Editor Notes: Need to find the section this refers to] is tight, CIGMMS will cause many CD-GMMs' scores to be the same and also narrows the range of scores. Hence, the search module will be more computationally intensive. In practice, this problem can be solved by tightening the Viterbi beam.

9.8 Overall search structure of decode

In the last two sections, we described the search and the GMM computation. The following algorithm provides a unified point of view of what's going on.

Initialize start state of <s> with path-score = 1;

[To be completed]

find final </s> BP table entry and back-trace through table to retrieve result;

[7] One can also deactivate the state. However, in our preliminary experiment, we found that deactivation can result in no valid paths at the final frame.

9.9 Multi-pass systems using Sphinx 3 .X

9.9.1 Word Lattice

During recognition the decoder maintains not just the single best hypothesis, but also a number of alternatives or candidates. For example, REED is a perfectly reasonable alternative to READ. The alternatives are useful in many ways: for instance, in N-best list generation. To facilitate such post-processing, the decoder can optionally produce a word lattice output for each input utterance. This output records all the candidate words recognized by the decoder at any point in time, and their main attributes such as time segmentation and acoustic likelihood scores.

The term "lattice" is used somewhat loosely. The word-lattice is really a directed acyclic graph or DAG. Each node of the DAG denotes a word instance that begins at a particular frame within the utterance. That is, it is a unique ¡word,start-time¿ pair. (However, there could be a number of end-times for this word instance. One of the features of a time-synchronous Viterbi search using beam pruning is that word candidates hypothesized by the decoder have a well-defined start-time, but a fuzzy range of end-times. This is because the start-time is primarily determined by Viterbi pruning, while the possible end-times are determined by beam pruning.)

There is a directed edge between two nodes in the DAG if the start-time of the destination node immediately follows one of the end times of the source node. That is, the two nodes can be adjacent in time. Thus, the edge determines one possible segmentation for the source node: beginning at the source's start-time and ending one frame before the destination's start-time. The edge also contains an acoustic likelihood for this particular segmentation of the source node.

Note: The beginning and end of sentence tokens, <s> and </s>, are not decoded as part of an utterance by the s3.X decoder. However, they have to be included in the word lattice file, for compatibility with the older Sphinx 3 decoder software. They are assigned 1-frame segmentations, with log-likelihood scores of 0. To accommodate them, the segmentations of adjacent nodes have to be "fudged" by 1 frame.

Word Lattice File Format

A word lattice file essentially contains the above information regarding the nodes and edges in the DAG. It is structured in several sections, as follows:

1. A comment section, listing important configuration arguments as comments
2. Frames section, specifying the number of frames in utterance
3. Nodes section, listing the nodes in the DAG
4. Initial and Final nodes (for <s> and </s>, respectively)
5. BestSegAscr section, a historical remnant now essentially empty
6. Edges section, listing the edges in the DAG

The file is formatted as follows. Note that any line in the file that begins with the # character in the first column is considered to be a comment.

getcwd: <current-working-directory>

-logbase <logbase-in-effect>

-dict <main lexicon>

-fdict <filler lexicon>

... (other arguments, written out as comment lines)

#

Frames <number-of-frames-in-utterance>

#

Nodes <number-of-nodes-in-DAG> (NODEID WORD STARTFRAME FIRST-ENDFRAME LAST-ENDFRAME)

<Node-ID> <Word-String> <Start-Time> <Earliest-End-time> <Latest-End-Time>

<Node-ID> <Word-String> <Start-Time> <Earliest-End-time> <Latest-End-Time>

<Node-ID> <Word-String> <Start-Time> <Earliest-End-time> <Latest-End-Time>

... (for all nodes in DAG)

#

Initial <Initial-Node-ID>

Final <Final-Node-ID>

#

BestSegAscr 0 (NODEID ENDFRAME ASCORE)

#

Edges (FROM-NODEID TO-NODEID ASCORE)

<Source-Node-ID> <Destination-Node-ID> <Acoustic Score>

<Source-Node-ID> <Destination-Node-ID> <Acoustic Score>

<Source-Node-ID> <Destination-Node-ID> <Acoustic Score>

... (for all edges in DAG) End

Note that the node-ID values for DAG nodes are assigned sequentially, starting from 0. Furthermore, they are sorted in descending order of their earliest-end-time attribute.

9.9.2 astar

Given a lattice, the program astar is able to generate the N-best results.

In the following example, a file name abc.lat is generated in a directory ./an4/ . The following command will rescore generate the N-Best list for each of these lattice.

```
$ src/programs/astar
  -mdeffn ./hub4_cd_continuous_8gau_1s_c_d_dd/hub4opensrc.6000.mdef
  -fdict ./filler.dict
  -dict ./an4.dict
  -lm ./an4.ug.lm.DMP
  -langwt 13.0
  -inspen 0.2
  -ctlfn ./an4.ctl
  -inlatdir ./an4/
  -logbase 1.0003
  -backtrace 1
```

9.9.3 dag

Given a lattice, the program dag is able to get the best path through the lattice.

```
$ src/programs/dag
  -mdef hub4_cd_continuous_8gau_1s_c_d_dd/hub4opensrc.6000.mdef
```

```
-fdict filler.dict
-dict an4.dict
-lm an4.ug.lm.DMP
-lw 13.0
-wip 0.2
-ctl an4.ctl
-inlatdir an4/
-logbase 1.0003
-backtrace
```

The usage of **dag** is very similar to astar. Please consult the manual for **astar**.

9.10 Other tools inside S3.X packages

9.10.1 align

This tool can create phone-level alignment given the transcription. The tool will guess the timing information for each word in the transcription.

For example:

```
$ align
 -logbase 1.0003
 -mdef ./hub4opensrc.6000.mdef
 -mean ./means
 -var ./variances
 -mixw ./mixture_weights
 -tmat ./transition_matrices
 -feat 1s_c_d_dd
 -topn 1000
 -beam 1e-80
 -senmgaufn .cont.
 -fdict .//model/lm/an4/filler.dict
```

```
-dict .//model/lm/an4/an4.dict
-ctl .//model/lm/an4/an4.ctl
-cepdir .//model/lm/an4/
-insent .//model/lm/an4/align.correct
-outsent @.out
-wdsegdir ./
-phsegdir ./
```

9.10.2 allphone

This tool can do generate the best phoneme sequence that matches a wave forms.

Example.

```
$ allphone
  -logbase 1.0003
  -mdeffn ./hub4opensrc.6000.mdef
  -meanfn ./means
  -varfn ./variances
  -mixwfn ./mixture_weights
  -tmatfn ./transition_matrices
  -feat 1s_c_d_dd
  -topn 1000
  -beam 1e-80
  -senmgaufn .cont.
  -ctlfn ./an4.ctl
  -cepdir ./an4/
  -phsegdir ./
```

9.11 Using the Sphinx 3 decoder with semi-continuous and continuous models

There are two flags which are specific to the type of model being used, the rest of the flags are independent of model type. The flags you need to change to switch from continuous models to semi-continuous ones are:

- the **-senmgaufn** flag would change from ".cont." to ".semi."
- the -feat flag would change from the feature you are using with continuous models to the feature you are using with the semicontinuous models (usually it is s3_1x39/1s_c_d_dd for continuous models and s2_4x for semi-continuous models)

Some of the other decoder flags and their usual settings are as follows:

```
-logbase 1.0001
-bestpath 0
-mdef $mdef
-senmgau .cont.
-mean ACMODDIR/means
-var ACMODDIR/variances
-mixw ACMODDIR/mixture_weights
-tmat ACMODDIR/transition_matrices
-langwt 10.5
-feat s3_1x39
-topn 32
-beam 1e-80
-nwbeam 1e-40
-dict dict
-fdict fdict
-fillpen fillpen
-lm lmfile
-inspen 0.2
-ctl ctl
```

```
-ctloffset ctloffset
-ctlcount ctlcount
-cepdir cepdir
-bptblsize 400000
-matchseg matchfile
-outlatdir outlatdir
-agc none
-varnorm yes
```

Chapter 10

Speaker Adaptation using Sphinx 3

[Editor Notes: Need more math.]

10.1 Speaker Adaptation

Author: *David Huggins-Daines, Arthur Chan*, Editor: *Arthur Chan*

In the previous chapters, various tools for creating acoustic model and language model for a speech application were introduced. One problem of the maximum likelihood framework used in SphinxTrain is that if there are mismatch between the training and testing conditions, the recognition rate will be significantly affected.

There are several sources of mismatch:

1. **Environmental Mismatches** The influence of additive and convolutive noise can significantly change the characteristics of the acoustic models.

2. **Speaker Variabilities** Inter-speaker variabilities of speech, e.g. Speaker A and Speaker B will utter the same word with very different acoustic characteristics.

3. **Language Mismatches** The difference of usage in the training and testing corpora can reduce the power of the language model in decoding.

In this chapter, we will discuss techniques of speaker adaptation provided by SphinxTrain and Sphinx 3 . This chapter currently discusses only acoustic model adaptation, which can alleviate the first two problems mentioned above, but not the third.

10.2 Different Principles of Speaker Adaptation

10.2.1 In terms of the mode of collecting adaptation data

Generally, speaker adaptation assumes that an acoustic model already exists and a new set of adaptation data is available. This new set of adaptation can be collected by asking the users to speak following a predefined transcription. This is called **supervised adaptation**. The process of collecting data for adaptation is usually called "enrollment".

The process of enrollment can be tedious and time-consuming and many users dislike it. In some applications, it is also impossible to allow such a design. For example, for telephony applications, users may be travelling while they make the calls. Asking the users to provide enrollment data can be very annoying.

However, supervised adaptation has its advantages. The performance of the adapted system is usually better because the transcription is known. Therefore, a lot of systems such as dictation usually ask for users to give 5 minutes to 30 minutes of enrollment data (sometimes more). This usually gives much better results.

As opposed to supervised adaptation, **unsupervised adaptation** attempts to determine the transcription by using an automatic speech recognizer. This avoids the need for enrollment. However, the performance of the resulting adaptation will usually be poorer because it is inevitable that the speech recognizer will make some mistakes in determining the transcription.

So which way to go? The choise of using supervised and unsupervised adaptation depends on many human factors. Major one is whether the user is patient enough to give enrollment data. In terms of performance, it can be very hard to forsee how much gain one can get from speaker adaptation. Usually, the number is relatively 5-15unsupervised adaptation, the

accuracy can even end up worse than the original system. Therefore, the use of each of these modes of adaptation should be a subject of serious consideration.

10.2.2 In terms of technique of parameter estimation

There are two major ways to do speaker adaptations. One is the so called "maximum a-posterior" (MAP) re-estimation of the parameters. One is maximum likelihood linear regression (MLLR).

We will mainly discuss speaker adaptation using MLLR. The use of regression analysis to find the best linear (or non-linear) relation between two sets of data is well-known. Linear regression consists of trying to find the terms A and b in the following set of equations:

$$y = Ax + b$$

The standard way of solving this problem is using the least minimum square estimate (LMSE) method. Maximum likelihood linear regression basically put linear regression into the Baum-Welch estimation framework. The consequence of this technique is that the transform that is estimated is optimal according to the maximum likelihood criterion. [1]

The use of MLLR allows speaker adaptation of all phonemes even if there is only few utterances of adaptation data. In such a case, all the data will be used to estimate a single tranformation matrix. This also adds some robustness to incorrect transcriptions, making MLLR suitable for unsupervised adaptation even with fairly noisy data.

The problem of single-transformation MLLR is that each phone usually has different acoustic characteristic and the transformations are usually different. Therefore, in order to make more efficient use of the adaptation data, it is necessary to allow different phones to have their own speaker adaptation matrix, or to tie data from different phones together and create the same transformation matrix for them. Usually, this technique is called multiple regression classes.

In practice, how many regression classes should be used is usually determined by validation using testing data.

[1] The term "regression" means "going back to the linear relation" when it first used by Fisher. (Is that true?)

MAP adaptation is really a variant of the acoustic model training procedure. The standard Baum-Welch algorithm results in a maximum likelihood estimate of the HMM model parameters given the training data and an initial model, which serves only to fill in the "hidden" data in the re-estimation formula (i.e. the state sequence and observation counts). The MAP technique extends this to produce a maximum a-posteriori estimate, where re-estimation takes into account the prior distribution of the model parameters, which in the case of speaker adaptation is (or can be directly calculated from) the parameters of the baseline, speaker-independent model.

In the case where only the means are adapted (which is sufficient for continuous density HMMs), this can be simplified to an interpolation between the baseline acoustic mean and the ML estimate of the adaptation data, weighted by the observation count of the adaptation data and the variance of the baseline and adaptation data models. Intuitively, we move the parameters for each senone observed in the adaptation data such that we maximize the likelihood of the adaptation data, while at the same time minimizing the variance.

10.3 MLLR with SphinxTrain

This section describes the use of the procedure of using speaker adaptation facilities of Sphinx. Sphinx provides facilities for supervised adaptation.

1. Given a previous acoustic model and new set of adaptation data. bw (refer to Chapter ?) is first used to collect the sufficient statistics count. For MLLR, what we need is just to have the mean vector. For MAP, we need the mean, variance, and optionally also the mixture weights and transition matrices.

2. Estimate the transformation matrix using the program mllr_solve.

3. Apply the transformation to the mean vector

10.3.1 Using bw for adaptation

As it is described in Chapter, bw is the the program which carries out forward-backward algorithm and collects the sufficient statistics [2] For a particular set of sentences. MLLR can be thought as a regression matrix estimator operated on these sufficient statistics. The actual command is:

```
$ bw
 -moddeffn mdef
 -mixwfn mixw
 -meanfn mean
 -varfn var
 -tmatfn tmat
 -dictfn dict
 -fdictfn filler_dict
 -cepdir feat_dir
 -cepext .mfc
 -lsnfn transcripts_fn
 -meanreest yes
 -varreest yes
 -2passvar yes
 -feat 1s_c_d_dd
 -ceplen 13
 -ctlfn adapt.ctl
 -accumdir accumdir
 -agc none
 -cmn current
```

The command will use the adaptation data found in *adapt.ctl* to gather the mean and variance statistics in the directory *accumdir*.

[2] you will fined many people called it posterior probablities. They are correct. But to learn the reason, I need to explain the mathematics so I tried to avoid it . :-)

10.3.2 mllr_solve

mllr_solve is the program that actually computes the transformation matrix.

```
$ mllr_solve
 -outmllrfn adapt.matrix
 -accumdir accumdir
 -meanfn mean
 -varfn vars
 -moddefn mdef
 -cdonly yes
```

The command will make use of the mean accumulators in director *accumdir* and estimate the vector. You also see the mean and variances are loaded in. The MLLR algorithm is basically an iterative algorithm so theoretically if you do the above process for more turns, the matrix you get will give better performance. In practice, 1-2 iterations are usually enough.

Notice that we only apply the transformation on the CD-senones as we specified -cdonly at here. Notice that when you specify option -cdonly, you also need to specify a model definition because it provides the mapping to determine which model is CI or CD.

In some cases, e.g. when you are using CI-based GMM selection, you may consider to adapt also CI-senones. Notice that, though, the effect of transformation-based adaptation (like MLLR) on the fast search is sometimes less studied. We have found that in general, it does not hurt to adapt the CI senones as well, so it is reasonable to omit the `-cdonly` option.

10.3.3 Using mllr_transform to do offline mean transformation

There are two ways to use the resulting matrix, one is to use it to transform the mean vectors *offline*, which is shown in this subsection. One is to use it to transform mean vectors on-line which we will show in next subsection.

```
$ mllr_transform
 -inmeanfn mean
 -outmeanfn out.mean
```

```
-mllrmat adapt.matrix
-cdonly yes
```

In such as case a new set of means, *out.mean* will be formed and you can just use them in speech recognition.

10.3.4 On-line adaptation using Sphinx 3 .0 and Sphinx 3 .X decoder

In Sphinx 3 .0 and Sphinx 3 .X decoders (decode_anytopo and decode), one can apply the adaptation matrix on-line by specifying a MLLR control file. For example, if you want the n-sentence to use regression matrix *abc* to adapt the mean on-line. Then you can create a file call *ctl.mllr* that contains

```
$  abc
```

as the n-th line of the file. You can then apply this transformation by specifying -ctl_mllr in the command line of `decode`. Then when the recognizer is decoding the n-th sentence, transformation matrix *abc* will be used to transform the mean.

In addition, Sphinx 3 .6 will allow you to specify a default adaptation matrix, using the -mllr option to specify the transformation matrix file.

10.4 MAP with SphinxTrain

The process of doing MAP adaptation is similar to that of MLLR. First, you must collect statistics from the adaptation data using the `bw` tool, as described above. It is important that you specify the `-2passvar no` option to `bw`, because it is necessary to collect some statistics that are required for adaptation of variances (if, that is, you wish to adapt your variances). If you wish to do MAP adaptation of the mixture weights and transition matrices, you should also pass `-mixwreest` and `-tmatreest` to `bw`.

Next, the `map_adapt` tool is used, in much the same way as `norm` is during training, to create a MAP re-estimate of the model paramters:

```
$ map_adapt
  -accumdir accumdir
  -meanfn means
```

```
-varfn variances
-mixwfn mixture_weights
-tmatfn transition_matrices
-mapmeanfn map_means
-mapvarfn map_variances
-mapmixwfn map_mixture_weights
-maptmatfn map_transition_matrices
```

As alluded to above, you may not wish to actually use all the re-estimated paramters. In particular, it is possible for re-estimation of variances and mixture weights to degrade the accuracy of the models. Re-estimation of transition matrices generally has no effect on accuracy. Unfortunately the optimal set of parameters must be determined experimentally. It is not necessary to specify all of the input and output files on the command line for `map_adapt`, only the ones which you wish to adapt, although in all cases the means and variances are required.

There are two drawbacks and one great advantage to using MAP. First, the bad news: MAP is not suitable for unsupervised adaptation. It is very important that the transcriptions of the adaptation data be correct; you may wish to consider force-aligning them, even. A somewhat lesser concern is that since MAP requires re-estimation of the means, there is no "on-line" adaptation and thus it is necessary to store a separate copy of the mean parameter file for each speaker.

The good news is that MAP can make much more effective use of large amounts of adaptation data. If you have more than 50 sentences of adaptation data, it is a good idea to use MAP in addition to MLLR. To do this, the recommended procedure is to perform MLLR adaptation as described above, using `mllr_transform` to generate a new means file, then *re-run* `bw` on the adaptation data using the transformed means, and finally do MAP adaptation as described here.

There are a few extra arguments to `map_adapt` that allow you to tune the adaptation process. MAP adaptation relies on a prior distribution which is estimated from the speaker-independent parameters in conjunction with an extra parameter τ, which controls the "speed" of adaptation, i.e. the weight given to the adaptation data vs. the prior models. By default this parameter is not used for means, and is estimated automatically for all other parameters. To enable the use of τ for means re-estimation, you can specify `-bayesmean no` on the command line. If you wish to manually set a uniform τ across all models, you should specify `-fixedtau yes` and use the `-tau` parameter to specify a value. The smaller the value of τ, the less weight is given to the prior models, and the faster the adap-

tation will be. In general, using a fixed $\tau = 10.0$ works well, particularly for semi-continuous models, where the estimation of τ is not particularly robust.

Appendix A

Command Line Information

A.1 Sphinx 3 Decoders

A.1.1 decode

Tool Description

Example

Command-line Argument Description

Usage: [options]

- **-agc** (Default Value : max)
  Description: tic gain control for c0 ('max' or 'none'); (max: c0 -= max-over-current-sentence(c0))

- **-beam** (Default Value : 1.0e-55)
  Description: eam selecting active HMMs (relative to best) in each frame [0(widest)..1(narrowest)]

- **-bghist** (Default Value : 0)
  Description: m-mode: If TRUE only one BP entry/frame; else one per LM state

- **-bptbldir** (Default Value : Direc)
  Description: tory in which to dump word Viterbi back pointer table (for debugging)

- **-cepdir** (Default Value : Input)
  Description: cepstrum files directory (prefixed to filespecs in control file)

- **-ci_pbeam** (Default Value : 1e-80)
  Description: CI phone beam for CI-based GMM Selection. [0(widest) .. 1(narrowest)]

- **-cmn** (Default Value : current)
  Description: pstral mean normalization scheme (default: Cep -= mean-over-current-sentence(Cep))

- **-cond_ds** (Default Value : 0)
  Description: itional Down-sampling, override normal down sampling.

- **-ctl** (Default Value : Control)
  Description: ile listing utterances to be processed

- **-ctlcount** (Default Value : 1000000)
  Description: 000 No. of utterances to be processed (after skipping -ctloffset entries)

- **-ctloffset** (Default Value : 0)
  Description: of utterances at the beginning of -ctl file to be skipped

- **-ctl_lm** (Default Value : Contr)
  Description: ol file that list the corresponding LM for an utterance

- **-ctl_mllr** (Default Value : Cont)
  Description: rol file that list the corresponding MLLR matrix for an utterance

- **-dict** (Default Value : Pronunci)
  Description: ation dictionary input file

- **-ds** (Default Value : 1)
  Description: Down-sampling the frame computation.

- **-epl** (Default Value : 3)
  Description: Per Lextree; #successive entries into one lextree before lextree-entries shifted to the next

- **-fdict** (Default Value : Filler)
  Description: word pronunciation dictionary input file

- **-feat** (Default Value : 1s_c_d_)
  Description: dd Feature type: Must be s3_1x39 / 1s_c_d_dd/ s2_4x .

- **-fillpen** (Default Value : Filler)
  Description: word probabilities input file

- **-fillprob** (Default Value : 0.1)
  Description: fault non-silence filler word probability

- **-gs** (Default Value : Gaussian)
  Description: election Mapping.

- **-gs4gs** (Default Value : 1)
  Description: that specified whether the input GS map will be used for Gaussian Selection. If it is disabled, the map will only provide information to other modules.

- **-hmmdump** (Default Value : 0)
  Description: er to dump active HMM details to stderr (for debugging)

- **-hmmhistbinsize** (Default Value : 5)
  Description: 000 Performance histogram: #frames vs #HMMs active; #HMMs/bin in this histogram

- **-hyp** (Default Value : Recogniti)
  Description: on result file, with only words

- **-hypseg** (Default Value : Recogn)
  Description: ition result file, with word segmentations and scores

- **-latext** (Default Value : lat.gz)
  Description: Filename extension for lattice files (gzip compressed, by default)

- **-lextreedump** (Default Value : 0)
  Description: hether to dump the lextree structure to stderr (for debugging)

- **-lm** (Default Value : Word)
  Description: am language model input file

- **-lmctlfn** (Default Value : Contro)
  Description: l file for language model

- (Default Value : NO DEFAULT VALUE)
  Description:

- **-lmdumpdir** (Default Value : The)
  Description: directory for dumping the DMP file.

- **-lminmemory** (Default Value : 0)
  Description: ad language model into memory (default: use disk cache for lm

- **-log3table** (Default Value : 1)
  Description: ermines whether to use the log3 table or to compute the values at run time.

- **-logbase** (Default Value : 1.0003)
  Description: Base in which all log-likelihoods calculated

- **-lw** (Default Value : 8.5)
  Description: e weight

- **-maxhistpf** (Default Value : 100)
  Description: ax no. of histories to maintain at each frame

- **-maxhmmpf** (Default Value : 20000)
  Description: Max no. of active HMMs to maintain at each frame; approx.

- **-maxwpf** (Default Value : 20)
  Description: no. of distinct word exits to maintain at each frame

- **-mdef** (Default Value : Model)
  Description: finition input file

- **-mean** (Default Value : Mixture)
  Description: gaussian means input file

- **-mixw** (Default Value : Senone)
  Description: ixture weights input file

- **-mixwfloor** (Default Value : 0.0000)
  Description: 001 Senone mixture weights floor (applied to data from -mixw file)

- **-Nlextree** (Default Value : 3)
  Description: of lextrees to be instantiated; entries into them staggered in time

- **-outlatdir** (Default Value : Dire)
  Description: ctory in which to dump word lattices

- **-outlatoldfmt** (Default Value : 1)
  Description: Whether to dump lattices in old format

- **-pbeam** (Default Value : 1.0e-50)
  Description: Beam selecting HMMs transitioning to successors in each frame [0(widest)..1(narrowest)]

- **-pheurtype** (Default Value : 0)
  Description: bypass, 1= sum of max, 2 = sum of avg, 3 = sum of 1st senones only

- **-pl_beam** (Default Value : 1.0e-80)
  Description: Beam for phoneme look-ahead. [1 (narrowest)..10000000(very wide)]

- **-pl_window** (Default Value : 1)
  Description: ndow size (actually window size-1) of phoneme look-ahead.

- **-ptranskip** (Default Value : 0)
  Description: wbeam for phone transitions every so many frames (if >= 1)

- **-senmgau** (Default Value : .cont.)
  Description: Senone to mixture-gaussian mapping file (or .semi. or .cont.)

- **-silprob** (Default Value : 0.1)
  Description: ault silence word probability

- **-subvq** (Default Value : Sub-vec)
  Description: tor quantized form of acoustic model

- **-subvqbeam** (Default Value : 3.0e-3)
  Description: Beam selecting best components within each mixture Gaussian [0(widest)..1(narrowest)]

- **-svq4svq** (Default Value : 0)
  Description: g that specified whether the input SVQ will be used as approximate scores of the Gaussians

- **-tmat** (Default Value : HMM)
  Description: e transition matrix input file

- **-tmatfloor** (Default Value : 0.0001)
  Description: HMM state transition probability floor (applied to -tmat file)

- **-treeugprob** (Default Value : 1)
  Description: TRUE (non-0), Use unigram probs in lextree

- **-utt** (Default Value : Utterance)
  Description: file to be processed (-ctlcount argument times)

- **-uw** (Default Value : 0.7)
  Description: weight

- **-var** (Default Value : Mixture)
  Description: aussian variances input file

- **-varfloor** (Default Value : 0.0001)
  Description: Mixture gaussian variance floor (applied to data from -var file)

- **-varnorm** (Default Value : no)
  Description: ance normalize each utterance (yes/no; only applicable if CMN is also performed)

- **-vqeval** (Default Value : 3)
  Description: ue added which used only part of the cepstral vector to do the estimation

- **-wbeam** (Default Value : 1.0e-35)
  Description: Beam selecting word-final HMMs exiting in each frame [0(widest)..1(narrowest)]

- **-wend_beam** (Default Value : 1.0e-)
  Description: 80 Beam selecting word-final HMMs exiting in each frame [0(widest) .. 1(narrowest)]

- **-wip** (Default Value : 0.7)
  Description: nsertion penalty

- (Default Value : NO DEFAULT VALUE)
  Description:

A.1.2 livedecode

Tool Description

Example

Command-line Argument Description

Usage: [options]

- **-agc** (Default Value : max)
  Description: tic gain control for c0 ('max' or 'none'); (max: c0 -= max-over-current-sentence(c0))

- **-alpha** (Default Value : 0.97)
  Description: ha for pre-emphasis window

- **-beam** (Default Value : 1.0e-55)
  Description: eam selecting active HMMs (relative to best) in each frame [0(widest)..1(narrowest)]

- **-bghist** (Default Value : 0)
  Description: m-mode: If TRUE only one BP entry/frame; else one per LM state

- **-bptbldir** (Default Value : Direc)
  Description: tory in which to dump word Viterbi back pointer table (for debugging)

- **-cepdir** (Default Value : Input)
  Description: cepstrum files directory (prefixed to filespecs in control file)

- **-ci_pbeam** (Default Value : 1e-80)
  Description: CI phone beam for CI-based GMM Selection. Good number should be [0(widest) .. 1(narrowest)]

- **-cmn** (Default Value : current)
  Description: pstral mean normalization scheme (default: Cep -= mean-over-current-sentence(Cep))

- **-cond_ds** (Default Value : 0)
  Description: itional Down-sampling, override normal down sampling.

- **-ctl** (Default Value : Control)
  Description: ile listing utterances to be processed

- **-ctlcount** (Default Value : 1000000)
  Description: 000 No. of utterances to be processed (after skipping -ctloffset entries)

- **-ctloffset** (Default Value : 0)
  Description: of utterances at the beginning of -ctl file to be skipped

- **-ctl_lm** (Default Value : Contr)
  Description: ol file that list the corresponding LMs

- **-dict** (Default Value : Pronunci)
  Description: ation dictionary input file

- **-doublebw** (Default Value : 0)
  Description: her mel filter triangle will have double the bandwidth, 0 is false

- **-ds** (Default Value : 1)
  Description: Down-sampling the frame computation.

- **-epl** (Default Value : 3)
  Description: Per Lextree; #successive entries into one lextree before lextree-entries shifted to the next

- **-fdict** (Default Value : Filler)
  Description: word pronunciation dictionary input file

- **-feat** (Default Value : 1s_c_d_)
  Description: dd Feature type: Must be 1s_c_d_dd / s3_1x39 / s2_4x / cep_dcep[,

- **-fillpen** (Default Value : Filler)
  Description: word probabilities input file

- **-fillprob** (Default Value : 0.1)
  Description: fault non-silence filler word probability

- **-frate** (Default Value : 100)
  Description: e rate

- **-gs** (Default Value : Gaussian)
  Description: election Mapping.

- **-gs4gs** (Default Value : 1)
  Description: that specified whether the input GS map will be used for Gaussian Selection. If it is disabled, the map will only provide information to other modules.

- **-hmmdump** (Default Value : 0)
  Description: er to dump active HMM details to stderr (for debugging)

- **-hmmhistbinsize** (Default Value : 5)
  Description: 000 Performance histogram: #frames vs #HMMs active; #HMMs/bin in this histogram

- **-hyp** (Default Value : Recogniti)
  Description: on result file, with only words

- **-hypseg** (Default Value : Recogn)
  Description: ition result file, with word segmentations and scores

- **-input_endian** (Default Value : 0)
  Description: the input data byte order, 0 is little, 1 is big endian

- **-latext** (Default Value : lat.gz)
  Description: Filename extension for lattice files (gzip compressed, by default)

- **-lextreedump** (Default Value : 0)
  Description: hether to dump the lextree structure to stderr (for debugging)

- **-lm** (Default Value : Word)
  Description: am language model input file

- **-lmctlfn** (Default Value : Contro)
  Description: l file for language model

- (Default Value : NO DEFAULT VALUE)
  Description:

- **-lmdumpdir** (Default Value : The)
  Description: directory for dumping the DMP file.

- **-lminmemory** (Default Value : 0)
  Description: ad language model into memory (default: use disk cache for lm

- **-log3table** (Default Value : 1)
  Description: ermines whether to use the log3 table or to compute the values at run time.

- **-logbase** (Default Value : 1.0003)
  Description: Base in which all log-likelihoods calculated

- **-lowerf** (Default Value : 200)
  Description: er edge of filters

- **-lw** (Default Value : 8.5)
  Description: e weight

- **-machine_endian** (Default Value : NO DEFAULT VALUE)
  Description: 0 the machine's endian, 0 is little, 1 is big endian

- **-maxcepvecs** (Default Value : 256)
  Description: Maximum number of cepstral vectors that can be obtained from a single sample buffer

- **-maxhistpf** (Default Value : 100)
  Description: ax no. of histories to maintain at each frame

- **-maxhmmpf** (Default Value : 20000)
  Description: Max no. of active HMMs to maintain at each frame; approx.

- **-maxhyplen** (Default Value : 1000)
  Description: Maximum number of words in a partial hypothesis (for block decoding)

- **-maxwpf** (Default Value : 20)
  Description: no. of distinct word exits to maintain at each frame

- **-mdef** (Default Value : Model)
  Description: finition input file

- **-mean** (Default Value : Mixture)
  Description: gaussian means input file

- **-mixw** (Default Value : Senone)
  Description: ixture weights input file

- **-mixwfloor** (Default Value : 0.0000)
  Description: 001 Senone mixture weights floor (applied to data from -mixw file)

- **-nfft** (Default Value : 256)
  Description: ts for FFT

- **-nfilt** (Default Value : 31)
  Description: r of mel filters

- **-Nlextree** (Default Value : 3)
  Description: of lextrees to be instantiated; entries into them staggered in time

- **-outlatdir** (Default Value : Dire)
  Description: ctory in which to dump word lattices

- **-outlatoldfmt** (Default Value : 1)
  Description: Whether to dump lattices in old format

- **-pbeam** (Default Value : 1.0e-50)
  Description: Beam selecting HMMs transitioning to successors in each frame [0(widest)..1(narrowest)]

- **-pheurtype** (Default Value : 0)
  Description: bypass, 1= sum of max, 2 = sum of avg, 3 = sum of 1st senones only

- **-pl_beam** (Default Value : 1.0e-80)
  Description: Beam for phoneme look-ahead. [0(widest) .. 1(narrowest)]

- **-pl_window** (Default Value : 1)
  Description: ndow size (actually window size-1) of phoneme look-ahead.

- **-ptranskip** (Default Value : 0)
  Description: wbeam for phone transitions every so many frames (if >= 1)

- **-samprate** (Default Value : 8000)
  Description: ampling rate (only 8K and 16K currently supported)

- **-senmgau** (Default Value : .cont.)
  Description: Senone to mixture-gaussian mapping file (or .semi. or .cont.)

- **-silprob** (Default Value : 0.1)
  Description: ault silence word probability

- **-subvq** (Default Value : Sub-vec)
  Description: tor quantized form of acoustic model

- **-subvqbeam** (Default Value : 3.0e-3)
  Description: Beam selecting best components within each mixture Gaussian [0(widest)..1(narrowest)]

- **-svq4svq** (Default Value : 0)
  Description: g that specified whether the input SVQ will be used as approximate scores of the Gaussians

- **-tmat** (Default Value : HMM)
  Description: e transition matrix input file

- **-tmatfloor** (Default Value : 0.0001)
  Description: HMM state transition probability floor (applied to -tmat file)

- **-treeugprob** (Default Value : 1)
  Description: TRUE (non-0), Use unigram probs in lextree

- **-upperf** (Default Value : 3500)
  Description: per edge of filters

- **-utt** (Default Value : Utterance)
  Description: file to be processed (-ctlcount argument times)

- **-uw** (Default Value : 0.7)
  Description: weight

- **-var** (Default Value : Mixture)
  Description: aussian variances input file

- **-varfloor** (Default Value : 0.0001)
  Description: Mixture gaussian variance floor (applied to data from -var file)

- **-varnorm** (Default Value : no)
  Description: ance normalize each utterance (yes/no; only applicable if CMN is also performed)

- **-vqeval** (Default Value : 3)
  Description: any vectors should be analyzed by VQ when building the shortlist. It speeds up the decoder, but at a cost.

- **-wbeam** (Default Value : 1.0e-35)
  Description: Beam selecting word-final HMMs exiting in each frame [0(widest)..1(narrowest)]

- **-wend_beam** (Default Value : 1.0e-)
  Description: 80 Beam selecting word-final HMMs exiting in each frame [0(widest) .. 1(narrowest)]

- **-wip** (Default Value : 0.7)
  Description: nsertion penalty

- **-wlen** (Default Value : 0.0256)
  Description: ndow length

A.1.3 livepretend

Tool Description

Example

Command-line Argument Description

Usage: [options]

- **-agc** (Default Value : max)
  Description: tic gain control for c0 ('max' or 'none'); (max: c0 -= max-over-current-sentence(c0))

- **-alpha** (Default Value : 0.97)
  Description: ha for pre-emphasis window

- **-beam** (Default Value : 1.0e-55)
  Description: eam selecting active HMMs (relative to best) in each frame [0(widest)..1(narrowest)]

- **-bghist** (Default Value : 0)
  Description: m-mode: If TRUE only one BP entry/frame; else one per LM state

- **-bptbldir** (Default Value : Direc)
  Description: tory in which to dump word Viterbi back pointer table (for debugging)

- **-cepdir** (Default Value : Input)
  Description: cepstrum files directory (prefixed to filespecs in control file)

- **-ci_pbeam** (Default Value : 1e-80)
  Description: CI phone beam for CI-based GMM Selection. Good number should be [0(widest) .. 1(narrowest)]

- **-cmn** (Default Value : current)
  Description: pstral mean normalization scheme (default: Cep -= mean-over-current-sentence(Cep))

- **-cond_ds** (Default Value : 0)
  Description: itional Down-sampling, override normal down sampling.

- **-ctl** (Default Value : Control)
  Description: ile listing utterances to be processed

- **-ctlcount** (Default Value : 1000000)
  Description: 000 No. of utterances to be processed (after skipping -ctloffset entries)

- **-ctloffset** (Default Value : 0)
  Description: of utterances at the beginning of -ctl file to be skipped

- **-ctl_lm** (Default Value : Contr)
  Description: ol file that list the corresponding LMs

- **-dict** (Default Value : Pronunci)
  Description: ation dictionary input file

- **-doublebw** (Default Value : 0)
  Description: her mel filter triangle will have double the bandwidth, 0 is false

- **-ds** (Default Value : 1)
  Description: Down-sampling the frame computation.

- **-epl** (Default Value : 3)
  Description: Per Lextree; #successive entries into one lextree before lextree-entries shifted to the next

- **-fdict** (Default Value : Filler)
  Description: word pronunciation dictionary input file

- **-feat** (Default Value : 1s_c_d_)
  Description: dd Feature type: Must be 1s_c_d_dd / s3_1x39 / s2_4x / cep_dcep[,

- **-fillpen** (Default Value : Filler)
  Description: word probabilities input file

- **-fillprob** (Default Value : 0.1)
  Description: fault non-silence filler word probability

- **-frate** (Default Value : 100)
  Description: e rate

- **-gs** (Default Value : Gaussian)
  Description: election Mapping.

- **-gs4gs** (Default Value : 1)
  Description: that specified whether the input GS map will be used for Gaussian Selection. If it is disabled, the map will only provide information to other modules.

- **-hmmdump** (Default Value : 0)
  Description: er to dump active HMM details to stderr (for debugging)

- **-hmmhistbinsize** (Default Value : 5)
  Description: 000 Performance histogram: #frames vs #HMMs active; #HMMs/bin in this histogram

- **-hyp** (Default Value : Recogniti)
  Description: on result file, with only words

- **-hypseg** (Default Value : Recogn)
  Description: ition result file, with word segmentations and scores

- **-input_endian** (Default Value : 0)
  Description: the input data byte order, 0 is little, 1 is big endian

- **-latext** (Default Value : lat.gz)
  Description: Filename extension for lattice files (gzip compressed, by default)

- **-lextreedump** (Default Value : 0)
  Description: hether to dump the lextree structure to stderr (for debugging)

- **-lm** (Default Value : Word)
  Description: am language model input file

- **-lmctlfn** (Default Value : Contro)
  Description: l file for language model

- (Default Value : NO DEFAULT VALUE)
  Description:

- **-lmdumpdir** (Default Value : The)
  Description: directory for dumping the DMP file.

- **-lminmemory** (Default Value : 0)
  Description: ad language model into memory (default: use disk cache for lm

- **-log3table** (Default Value : 1)
  Description: ermines whether to use the log3 table or to compute the values at run time.

- **-logbase** (Default Value : 1.0003)
  Description: Base in which all log-likelihoods calculated

- **-lowerf** (Default Value : 200)
  Description: er edge of filters

- **-lw** (Default Value : 8.5)
  Description: e weight

- **-machine_endian** (Default Value : NO DEFAULT VALUE)
  Description: 0 the machine's endian, 0 is little, 1 is big endian

- **-maxcepvecs** (Default Value : 256)
  Description: Maximum number of cepstral vectors that can be obtained from a single sample buffer

- **-maxhistpf** (Default Value : 100)
  Description: ax no. of histories to maintain at each frame

- **-maxhmmpf** (Default Value : 20000)
  Description: Max no. of active HMMs to maintain at each frame; approx.

- **-maxhyplen** (Default Value : 1000)
  Description: Maximum number of words in a partial hypothesis (for block decoding)

- **-maxwpf** (Default Value : 20)
  Description: no. of distinct word exits to maintain at each frame

- **-mdef** (Default Value : Model)
  Description: finition input file

- **-mean** (Default Value : Mixture)
  Description: gaussian means input file

- **-mixw** (Default Value : Senone)
  Description: ixture weights input file

- **-mixwfloor** (Default Value : 0.0000)
  Description: 001 Senone mixture weights floor (applied to data from -mixw file)

- **-nfft** (Default Value : 256)
  Description: ts for FFT

- **-nfilt** (Default Value : 31)
  Description: r of mel filters

- **-Nlextree** (Default Value : 3)
  Description: of lextrees to be instantiated; entries into them staggered in time

- **-outlatdir** (Default Value : Dire)
  Description: ctory in which to dump word lattices

- **-outlatoldfmt** (Default Value : 1)
  Description: Whether to dump lattices in old format

- **-pbeam** (Default Value : 1.0e-50)
  Description: Beam selecting HMMs transitioning to successors in each frame [0(widest)..1(narrowest)]

- **-pheurtype** (Default Value : 0)
  Description: bypass, 1= sum of max, 2 = sum of avg, 3 = sum of 1st senones only

- **-pl_beam** (Default Value : 1.0e-80)
  Description: Beam for phoneme look-ahead. [0(widest) .. 1(narrowest)]

- **-pl_window** (Default Value : 1)
  Description: ndow size (actually window size-1) of phoneme look-ahead.

- **-ptranskip** (Default Value : 0)
  Description: wbeam for phone transitions every so many frames (if >= 1)

- **-samprate** (Default Value : 8000)
  Description: ampling rate (only 8K and 16K currently supported)

- **-senmgau** (Default Value : .cont.)
  Description: Senone to mixture-gaussian mapping file (or .semi. or .cont.)

- **-silprob** (Default Value : 0.1)
  Description: ault silence word probability

- **-subvq** (Default Value : Sub-vec)
  Description: tor quantized form of acoustic model

- **-subvqbeam** (Default Value : 3.0e-3)
  Description: Beam selecting best components within each mixture Gaussian [0(widest)..1(narrowest)]

- **-svq4svq** (Default Value : 0)
  Description: g that specified whether the input SVQ will be used as approximate scores of the Gaussians

- **-tmat** (Default Value : HMM)
  Description: e transition matrix input file

- **-tmatfloor** (Default Value : 0.0001)
  Description: HMM state transition probability floor (applied to -tmat file)

- **-treeugprob** (Default Value : 1)
  Description: TRUE (non-0), Use unigram probs in lextree

- **-upperf** (Default Value : 3500)
  Description: per edge of filters

- **-utt** (Default Value : Utterance)
  Description: file to be processed (-ctlcount argument times)

- **-uw** (Default Value : 0.7)
  Description: weight

- **-var** (Default Value : Mixture)
  Description: aussian variances input file

- **-varfloor** (Default Value : 0.0001)
  Description: Mixture gaussian variance floor (applied to data from -var file)

- **-varnorm** (Default Value : no)
  Description: ance normalize each utterance (yes/no; only applicable if CMN is also performed)

- **-vqeval** (Default Value : 3)
  Description: any vectors should be analyzed by VQ when building the shortlist. It speeds up the decoder, but at a cost.

- **-wbeam** (Default Value : 1.0e-35)
  Description: Beam selecting word-final HMMs exiting in each frame [0(widest)..1(narrowest)]

- **-wend_beam** (Default Value : 1.0e-)
  Description: 80 Beam selecting word-final HMMs exiting in each frame [0(widest) .. 1(narrowest)]

- **-wip** (Default Value : 0.7)
  Description: nsertion penalty

- **-wlen** (Default Value : 0.0256)
  Description: ndow length

- (Default Value : NO DEFAULT VALUE)
  Description:

- (Default Value : NO DEFAULT VALUE)
  Description:

A.1.4 gausubvq

Tool Description

Example

Command-line Argument Description

Usage: [options]

- **-eps** (Default Value : 0.0001)
  Description: pping criterion: stop iterations if relative decrease in sq(error) < eps

- **-iter** (Default Value : 100)
  Description: o. of k-means iterations for clustering

- **-log3table** (Default Value : 1.0003)
  Description: Determines whether to use the log3 table or to compute the values at run time.

- **-mean** (Default Value : Means)
  Description: le

- **-mixw** (Default Value : Mixture)
  Description: weights file (needed, even though it's not part of the computation)

- **-mixwfloor** (Default Value : 0.0000)
  Description: 001 Floor for non-zero mixture weight values in input model

- **-stdev** (Default Value : 0)
  Description: d.dev. (rather than var) in computing vector distances during clustering

- **-subvq** (Default Value : Output)
  Description: subvq file (stdout if not specified)

- **-svqrows** (Default Value : 4096)
  Description: . of codewords in output subvector codebooks

- **-svspec** (Default Value : Subvec)
  Description: tors specification (e.g., 24,0-11/25,12-23/26-38 or 0-12/13-25/26-38)

- **-var** (Default Value : Variances)
  Description: file

- **-varfloor** (Default Value : 0.0001)
  Description: Floor for non-zero variance values in input model

A.1.5 align

A.1.6 allphone

A.1.7 dag

A.1.8 astar

A.1.9 cepview

Tool Description

cepview could view the cepstral coefficient create by **wave2feat**.

Command-line Argument Description

A.2 SphinxTrain : Executables

A.2.1 agg_seg

Tool Description

Description:

agg_seg sample accumulate feature vectors and used it for quantizing the vector space. This functionality is very useful in S2 training initialization. Not all features vectors are used. They are sampled using option -stride.

There are many other options of this command is currently obsolete. Please type -example yes to get a working argument list.

Example

Example:

agg_seg -segdmpdirs segdmpdir -segdmpfn dumpfile -segtype all -ctlfn ctl -cepdir cepdir -cepext .mfc -ceplen 13 -stride 10

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-segdmpdirs Segment dump directories (Default Value : NONE)

-segdmpfn Segment dump file (Default Value : NONE)

-segidxfn Segment index into the dump file. (Default Value : NONE)

-segtype Type of segments to dump. all,st,phn (Default Value : st)

-cntfn Per id count file (Default Value : NONE)

-ddcodeext Extension of the VQ 2nd difference cepstrum files (Default Value : xcode)

-lsnfn Lexical transcript file (contains all utts in ctl file) (Default Value : NONE)

-sentdir Root directory of sent (lexical transcript) files (Default Value : NONE)

-sentext Extension of sent (lexical transcript) files (Default Value : NONE)

-ctlfn The control file name (enumerates utts in corpus) (Default Value : NONE)

-mllrctlfn Lists the MLLR transforms for each utterance (Default Value : NONE)

-mllrdir Directory for MLLR matrices (Default Value : NONE)

-nskip The number of utterances to skip in the control file (Default Value : 0)

-runlen The number of utterances to process after skipping (Default Value : -1)

-moddeffn Model definition file containing all the triphones in the corpus. State/transition matrix definitions are ignored. (Default Value : NONE)

-ts2cbfn Tied state to codebook mapping file (may be '.semi.' or '.cont.') (Default Value : NONE)

-cb2mllrfn codebook to MLLR class mapping file (may be '.1cls.') (Default Value : NONE)

-dictfn Lexicon containing all the words in the lexical transcripts. (Default Value : NONE)

-fdictfn Lexicon containing all the filler words in the lexical transcripts. (Default Value : NONE)

-segdir Root directory of the state segmentation files (Default Value : NONE)

-segext Extension of the state segmentation files (Default Value : v8_seg)

-ccodedir Root directory of the VQ cepstrum files (Default Value : NONE)

-ccodeext Extension of the VQ cepstrum files (Default Value : ccode)

-dcodedir Root directory of the VQ difference cepstrum files (Default Value : NONE)

-dcodeext Extension of the VQ cepstrum files (Default Value : d2code)

-pcodedir Root directory of the VQ power files (Default Value : NONE)

-pcodeext Extension of the VQ power files (Default Value : p3code)

-ddcodedir Root directory of the VQ 2nd difference cepstrum files (Default Value : NONE)

-ddcodeext Extension of the VQ 2nd difference cepstrum files (Default Value : xcode)

-cepdir Root directory of the cepstrum files (Default Value : NONE)

-cepext Extension of the cepstrum files (Default Value : mfc)

-ceplen # of coefficients per cepstrum frame (Default Value : 13)

-agc The type of automatic gain control to do max—emax (Default Value : max)

-cmn The do cepstral mean normalization based on current—prior utterance(s) (Default Value : current)

-varnorm Normalize utterance by its variance (Default Value : no)

-silcomp Do silence compression based on current—prior utterance (Default Value : none)

-feat Feature set to compute (Default Value : NONE)

d (Default Value : 4s_12c_24d_3p_12d)

d (Default Value : 1s_12c_12d_3p_12d)

-cachesz Feature cache size in Mb (Default Value : 200)

-stride Take every -stride'th frame when producing dmp (Default Value : 1)

A.2.2 bldtree

Tool Description

Description:

Given a set of questions. Build decision tree for a set of feature of a particular phone. By default, decision tree are not built for filler phones and the phone tagged with SIL. One very confusing parameters of this tool is -stwt, if you are training a n-state HMM, you need to specify n values after this flag.

Example

bld_tree -treefn tree -moddeffn mdef -mixwfn mixw -meanfn mean -varfn var -psetfn questions -stwt 1.0 0.05 0.01 -state 0 -ssplitmin 1 -ssplitmax 7 -ssplitthr 1e-10 -csplitmin 1 -csplitmax 2000 -csplitthr 1e-10 -cntthresh 10

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-treefn Name of output tree file to produce (Default Value : NONE)

-moddeffn Model definition file of the discrete models (Default Value : NONE)

-ts2cbfn The type of models to build trees on (Default Value : .semi.)

-meanfn means file for tree building using continuous HMMs (Default Value : NONE)

-varfn variances file for tree building using continuous HMMs (Default Value : NONE)

-varfloor The minimum variance (Default Value : 0.00001)

-cntthresh Ignore all states with counts less than this (Default Value : 0.00001)

-mixwfn PDF's for tree building using semicontinuous HMMs (Default Value : NONE)

-psetfn phone set definitions for phonetic questions (Default Value : NONE)

-phone Build trees over n-phones having this base phone (Default Value : NONE)

-state Build tree for this state position. E.g. For a three state HMM, this value can be 0,1 or 2. For a 5 state HMM, this value can be 0,1,2,3 or 4, and so on (Default Value : NONE)

-mwfloor Mixing weight floor for tree building (Default Value : 1e-4)

-stwt Weights on neighboring states, This flag needs a string of numbers equal to the number of HMM-states (Default Value : NONE)

-ssplitthr Simple node splitting threshold (Default Value : 8e-4)

-ssplitmin Minimum of simple tree splits to do. (Default Value : 1)

-ssplitmax The maximum number of bifurcations in the simple tree before it is used to build complex questions. (Default Value : 5)

-csplitthr Compound node splitting threshold (Default Value : 8e-4)

-csplitmin Minimum # of compound tree splits to do (Default Value : 1)

-csplitmax Minimum # of compound tree splits to do (Default Value : 100)

A.2.3 bw

Tool Description

Description: Strictly speaking, bw only implements the first-part of the Baum-Welch algorithm. That is it go through forward and backward algortihm and collect the necessary statistics for parameter estimation.

The advantage of this architecture is that researcher can easily write programs to do parameter estimation and they have no need to tweak the huge and usually difficult Baum-Welch algorithm.

In terms of functionality, one important thing you need to know is option -part and -npart. They can allow you to split the training into N equal parts Say, if there are M utterances in your control file, then this will enable you to run the training separately on each (M/N)th part. This flag may be set to specify which of these parts you want to currently train on. As an example, if your total number of parts (-npart) is 3, -part can take one of the values 1,2 or 3.

To control the speed of the training, -abeam (control alpha search) and -bbeam (control beta search) can be used to control the searching time. Notice that if the beams are too small, the path may not reach the end of the search and results in estimation error Too many lost path may also cause training set likelihood not unable to increase

Several options allow the user to control the behaviour of bw such that silence or pauses can be taken care. For example. One could use -sildelfn to specify periods of time which was assume to be silence. One could also use -sildel and -siltag to specify a silence and allow them to be optionall deleted.

Finally, one can use the viterbi training mode of the code. Notice though, the code is not always tested by CMU's researcher

I also included the following paragraph from Rita's web page. I largely adapted from here and I think it is a pretty good wrap-up of the convergence issues "bw is an iterative re-estimation process, so you have to run many passes of the Baum-Welch re-estimation over your training data. Each of these passes, or iterations, results in a slightly better set of models for the CI phones. However, since the objective function maximized in each of theses passes is the likelihood, too many iterations would ultimately result in models which fit very closely to the training data. you might not want this to happen for many reasons. Typically, 5-8 iterations of Baum-Welch are sufficient for getting good estimates of the CI models. You can automatically determine the number of iterations that you need by looking at the total likelihood of the training data at the end of the

first iteration and deciding on a "convergence ratio" of likelihoods. This is simply the ratio of the total likelihood in the current iteration to that of the previous iteration. As the models get more and more fitted to the training data in each iteration, the training data likelihoods typically increase monotonically. The convergence ratio is therefore a small positive number. The convergence ratio becomes smaller and smaller as the iterations progress, since each time the current models are a little less different from the previous ones. Convergence ratios are data and task specific, but typical values at which you may stop the Baum-Welch iterations for your CI training may range from 0.1-0.001. When the models are variance-normalized, the convergence ratios are much smaller."

Example

Example: Command used to train continuous HMM (Beware, this only illustrates how to use this command, for detail on how to tune it, please consult the manual.) bw -moddeffn mdef -ts2cbfn .cont. -mixwfn mixw -tmatfn tmatn -meanfn mean -varfn var -dictfn dict -fdictfn fillerdict -ctlfn control_files -part 1 -npart 1 -cepdir feature_dir -cepext mfc -lsnfn transcription -accumdir accumdir -abeam 1e-200 -bbeam 1e-200 -meanreest yes -varreest yes -tmatreest yes -feat 1s_12c_12d_3p_12dd -ceplen 13

If yo want to do parallel training for N machines. Run N trainers with -part 1 -npart N -part 2 -npart N . . -part N -npart N

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn The model definition file for the model inventory to train (Default Value : NONE)

-tmatfn The transition matrix parameter file name (Default Value : NONE)

-mixwfn The mixture weight parameter file name (Default Value : NONE)

-meanfn The mean parameter file name (Default Value : NONE)

-varfn The var parameter file name (Default Value : NONE)

-mwfloor Mixing weight smoothing floor (Default Value : 0.00001)

-tpfloor Transition probability smoothing floor (Default Value : 0.0001)

-varfloor The minimum variance (Default Value : 0.00001)

-topn Compute output probabilities based this number of top scoring densities. (Default Value : 4)

-dictfn The content word dictionary (Default Value : NONE)

-fdictfn The filler word dictionary (e.g. SIL, SILb, ++COUGH++) (Default Value : NONE)

-ctlfn The training corpus control file (Default Value : NONE)

-nskip The number of utterances to skip at the beginning of a control file (Default Value : NONE)

-runlen The number of utterances to process in the (skipped) control file (Default Value : -1)

-part Identifies the corpus part number (range 1..NPART) (Default Value : NONE)

-npart Partition the corpus into this many equal sized subsets (Default Value : NONE)

-cepext The cepstrum file extension (Default Value : mfc)

-cepdir The cepstrum data root directory (Default Value : NONE)

-segext State segmentation file extension (Default Value : v8_seg)

-segdir State segmentation file root directory (Default Value : NONE)

-sentdir The sentence transcript file directory (Default Value : NONE)

-sentext The sentence transcript file extension (Default Value : sent)

-lsnfn The corpus word transcript file (Default Value : NONE)

-accumdir A path where accumulated counts are to be written. (Default Value : NONE)

-ceplen The length of the input feature (e.g. MFCC) vectors (Default Value : 13)

-agc The type of automatic gain control to do max—emax (Default Value : max)

-cmn The do cepstral mean normalization based on current—prior utterance(s) (Default Value : current)

-varnorm Variance Normalize? (Default Value : no)

-silcomp Do silence compression based on current—prior utterance (Default Value : none)

-sildel Allow optional silence deletion in the Baum-Welch algorithm or the Viterbi algorithm. (Default Value : no)

-siltag Specify the tag of silence, by default it is <sil>. (Default Value : SIL)

-abeam Evaluate alpha values subject to this beam (Default Value : 1e-100)

-bbeam Evaluate beta values (update reestimation sums) subject to this beam (Default Value : 1e-100)

-varreest Reestimate variances (Default Value : yes)

-meanreest Reestimate means (Default Value : yes)

-mixwreest Reestimate mixing weights (Default Value : yes)

-tmatreest Reestimate transition probability matrices (Default Value : yes)

-spkrxfrm A speaker transform to use for SAT modelling (Default Value : NONE)

-mllrmult Reestimate multiplicative term of MLLR adatpation of means (Default Value : no)

-mllradd Reestimate shift term of MLLR adaptation of means (Default Value : no)

-ts2cbfn Tied-state-to-codebook mapping file name (Default Value : NONE)

-feat This argument selects the derived feature computation to use. (Default Value : NONE)

-timing Controls whether profiling information is displayed (Default Value : yes)

-viterbi Controls whether Viterbi training is done (Default Value : no)

-2passvar Reestimate variances based on prior means (Default Value : no)

-sildelfn File which specifies frames of background 'silence' to delete (Default Value : NONE)

-cb2mllrfn Codebook-to-MLLR-class mapping file name (Default Value : NONE)

-spthresh State posterior probability floor for reestimation. States below this are not counted (Default Value : 0.0)

-maxuttlen Maximum # of frames for an utt (0 => no fixed limit) (Default Value : 0)

-ckptintv Checkpoint the reestimation sums every -chkptintv utts (Default Value : NONE)

A.2.4 cp_parm

Tool Description

Description: Copy parameters such as means and variances from one model to another model. You need to specify a "copy operation file" which each operation looks like this dest_idx1 src_idx1 dest_idx2 src_idx2 dest_idx3 src_idx3 . . . For example, the first line will instruct cp_param copy src model index 1 to destionation model index 1. This tool is still under heavy development at 20040807

Example

Example: This example copy mean from a single file from the source file to 5 mean vector in th destination file. First you need to prepare a file like this 0 0 1 0 2 0 3 0 4 0

Lets call it cp_op cp_parm -cpopsfn cp_op -igaufn globalmean -ncbout 5 -ogaufn out.means -feat [Your feature type]

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-cpopsfn Copy operation file name (Default Value : NONE)

-imixwfn Input mixing weight file (Default Value : NONE)

-omixwfn Output mixing weight file (Default Value : NONE)

-nmixwout # of mixing weight arrays in the output file (Default Value : NONE)

-itmatfn Input transition matrix file (Default Value : NONE)

-otmatfn Output transition matrix file (Default Value : NONE)

-ntmatout # of transition matrices in the output file (Default Value : NONE)

-igaufn Input Gaussian density parameter file (Default Value : NONE)

-ogaufn Output Gaussian density parameter file (Default Value : NONE)

-ncbout # of codebooks in the output file (Default Value : NONE)

-feat Feature set to use (Default Value : c[1..L-1]d[1..L-1]c[0]d[0]dd[0]dd[1..L-1])

A.2.5 delint

Tool Description

Description: (copied from Rita's web page.)

Deleted interpolation is the final step in creating semi-continuous models. The output of deleted interpolation are semi-continuous models in sphinx-3 format. These have to be further converted to sphinx-2 format, if you want to use the SPHINX-II decoder.

Deleted interpolation is an iterative process to interpolate between CD and CI mixture-weights to reduce the effects of overfitting. The data are divided into two sets, and the data from one set are used to estimate the optimal interpolation factor between CI and CD models trained from the other set. Then the two data sets are switched and this procedure is repeated using the last estimated interpolation factor as an initialization for the current step. The switching is continued until the interpolation factor converges.

To do this, we need *two* balanced data sets. Instead of the actual data, however, we use the Bauim-Welch buffers, since the related math is convenient. we therefore need an *even* number of buffers that can be grouped into two sets. DI cannot be performed if you train using only one buffer. At least in the final iteration of the training, you must perform the training in (at least) two parts. You could also do this serially as one final iteration of training AFTER BW has converegd, on a non-lsf setup.

Note here that the norm executable used at the end of every Baum-Welch iteration also computes models from the buffers, but it does not require an even number of buffers. BW returns numerator terms and denominator terms for the final estimation, and norm performs the actual division. The number of buffers is not important, but you would need to run norm at the end of EVERY iteration of BW, even if you did the training in only one part. When you have multiple parts norm sums up the numerator terms from the various buffers, and the denominator terms, and then does the division.

Example

delint -accumdirs accumdir -moddeffn mdef -mixwfn mixw -cilambda 0.9 -feat c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/ -ceplen 13 -maxiter 4000

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn The model definition file name (Default Value : NONE)

-mixwfn The mixture weight parameter file name (Default Value : NONE)

-accumdirs A path where accumulated counts are to be read. (Default Value : NONE)

-cilambda Weight of CI distributions with respect to uniform distribution (Default Value : 0.9)

-maxiter max # of iterations if no lambda convergence (Default Value : 100)

-feat 2dd feature stream definition (Default Value : 4s_12c_24d_3p_1)

-ceplen Input feature vector length (e.g. MFCC) (Default Value : 13)

A.2.6 dict2tri

Tool Description

Description: find the triphone list from the dictionary

Example

Example : dict2tri -dictfn dict -basephnfn phonelist -btwtri yes

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-dictfn Dictionary file name (Default Value : NONE)

-basephnfn Base phone list file name (Default Value : NONE)

-btwtri Compute between-word triphone set (Default Value : yes)

A.2.7 inc_comp

Tool Description

Description:

Increase the number of mixture of a continuous HMM. Notice that option -ninc actually means the finally number of mixture one wants to obtain. Usually, it is the power of two. You are also recommended to split the number of mixture from 1 -> 2 -> 4 -> 8 -> and so on.

Example

Example :

inc_comp -ninc 16 -dcountfn mixture_weights -inmixwfn mixture_weights -outmixwfn out_mixture_weights -inmeanfn means -outmeanfn out_means -invarfn variance -outvarfn out_variance -ceplen 13

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-ninc The # of densities to split (Default Value : 1)

-inmixwfn The weight file for all N den/mix (Default Value : NONE)

-outmixwfn The output mixing weight file name w/ N+NINC density weights/mixtur
(Default Value : NONE)

-inmeanfn The source mean file w/ N means (Default Value : NONE)

-outmeanfn The new mean file w/ N+NINC means (Default Value : NONE)

-invarfn The source variance file w/ N means (Default Value : NONE)

-outvarfn The new variance file w/ N+NINC means (Default Value : NONE)

-dcountfn The density counts for the N source den/mix (Default Value :
NONE)

-ceplen The length of the input feature (e.g. MFCC) vectors (Default Value
: 13)

-feat Defines the acoustic feature set. (Default Value : NONE)

A.2.8 init_gau

Tool Description

Description: (Copy from Rita's web manual) To initialize the means and variances, global values of these parameters are first estimated and then copied into appropriate positions in the parameter files. The global mean is computed using all the vectors you have in your feature files. This is usually a very large number, so the job is divided into many parts. At this stage you tell the Sphinx how many parts you want it to divide this operation into (depending on the computing facilities you have) and the Sphinx "accumulates" or gathers up the vectors for each part separately and writes it into an intermediate buffer on your machine. The executable init_gau is used for this purpose.

Example

Example:

init_gau -accumdir accumdir -ctlfn controlfn -part 1 -npart 1 -cepdir cepdir -feat 1s_12c_12d_3p_12dd -ceplen 13

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn Model definition file for the single density HMM's to initialize (Default Value : NONE)

-ts2cbfn Tied-state-to-codebook mapping file (Default Value : NONE)

-accumdir Where to write mean/var counts (Default Value : NONE)

-meanfn Mean file for variance initialization (Default Value : NONE)

-ctlfn Control file of the training corpus (Default Value : NONE)

-nskip # of lines to skip in the control file (Default Value : NONE)

-runlen # of lines to process in the control file (after any skip) (Default Value : NONE)

-part Identifies the corpus part number (range 1..NPART) (Default Value : NONE)

-npart Partition the corpus into this many equal sized subsets (Default Value : NONE)

-lsnfn All word transcripts for the training corpus (consistent order w/ -ctlfn!) (Default Value : NONE)

-dictfn Dictionary for the content words (Default Value : NONE)

-fdictfn Dictionary for the filler words (Default Value : NONE)

-segdir Root directory of the training corpus state segmentation files. (Default Value : NONE)

-segext Extension of the training corpus state segmentation files. (Default Value : v8_seg)

-scaleseg Scale existing segmentation to fit new parameter stream length. (Default Value : no)

-cepdir Root directory of the training corpus cepstrum files. (Default Value : NONE)

-cepext Extension of the training corpus cepstrum files. (Default Value : mfc)

-silcomp Controls silence compression. (Default Value : none)

-cmn Controls cepstral mean normalization. (Default Value : current)

-varnorm Controls variance normalization. (Default Value : no)

-agc Controls automatic gain control. (Default Value : max)

-feat Controls which feature extraction algorithm is used. (Default Value : NONE)

-ceplen # of components in cepstrum vector (Default Value : 13)

A.2.9 init_mixw

Tool Description

Description: Initialization of mixture weight

Example

Example:

init_mixw -src_moddeffn src_moddeffn -src_ts2cbfn .semi. -src_mixwfn src_mixwfn -src_meanfn src_meanfn -src_varfn src_varfn -src_tmatfn src_tmatfn -dest_moddeffn dest_moddeffn -dest_ts2cbfn .semi. -dest_mixwfn dest_mixwfn -dest_me anfn dest_meanfn -dest_varfn dest_varfn -dest_tmatfn dest_tmatfn -feat c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/ -ceplen 13

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-src_moddeffn The source model definition file name (Default Value : NONE)

-src_ts2cbfn The source state definition file name (Default Value : NONE)

-src_mixwfn The source mixing weight file name (Default Value : NONE)

-src_meanfn The source mean file name (Default Value : NONE)

-src_varfn The source variance file name (Default Value : NONE)

-src_tmatfn The source transition matrix file name (Default Value : NONE)

-dest_moddeffn The destination model definition file name (Default Value : NONE)

-dest_ts2cbfn The destination state definition file name (Default Value : NONE)

-dest_mixwfn The destination mixing weight file name (Default Value : NONE)

-dest_meanfn The destination mean file name (Default Value : NONE)

-dest_varfn The destination variance file name (Default Value : NONE)

-dest_tmatfn The destination transition matrix file name (Default Value : NONE)

-feat 2dd Derived feature computation to use (Default Value : 1s_12c_12d_3p_1)

-ceplen Size of the input feature vector length (Default Value : 13)

A.2.10 kmeans_init

Tool Description

Description:

Using the segment dump file generated by external software such as agg_seg to initilaize the model. It performs k-mean clustering to create the initial means and variances for s2 hmms. This is an important process of initialization of s2 training.

Example

Example :

kmeans_init -gthobj single -stride 1 -ntrial 1 -minratio 0.001 -ndensity 256 -meanfn $outhmm/means-varfn$outhmm/variances -reest no -segdmpdirs $segdmpdir - segdmpfn$dumpfile -ceplen 13

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-segdir Directory containing the state segmentations (Default Value : NONE)

-segext Extention of state segmentation files (Default Value : v8_seg)

-omoddeffn Model definition of output models (Default Value : NONE)

-dmoddeffn Model definition used for observation dump (Default Value : NONE)

-ts2cbfn Tied-state-to-codebook mapping file (Default Value : NONE)

-lsnfn LSN file name (word transcripts) (Default Value : NONE)

-dictfn Dictionary file name (Default Value : NONE)

-fdictfn Filler word dictionary file name (Default Value : NONE)

-cbcntfn File containing # of times a codebook ID appears in the corpus (Default Value : NONE)

-maxcbobs Cluster at most this many observations per codebook (Default Value : NONE)

-maxtotobs Cluster at most approximately this many observations over all codebooks (Default Value : NONE)

-featsel The feature stream (0, 1, ...) to select (Default Value : NONE)

-ctlfn The training corpus control file (Default Value : NONE)

-cepext The cepstrum file extension (Default Value : mfc)

-cepdir The cepstrum data root directory (Default Value : NONE)

-ceplen The length of the input feature (e.g. MFCC) vectors (Default Value : 13)

-agc The type of automatic gain control to do max—emax (Default Value : max)

-cmn The do cepstral mean normalization based on current—prior utterance(s) (Default Value : current)

-varnorm Normalize utterance by its variance (Default Value : no)

-silcomp Do silence compression based on current—prior utterance (Default Value : none)

-feat 2dd This argument selects the derived feature computation to use. (Default Value : 1s_12c_12d_3p_1)

-segdmpdirs segment dmp directories (Default Value : NONE)

-segdmpfn segment dmp file (Default Value : NONE)

-segidxfn segment dmp index file (Default Value : NONE)

-fpcachesz # of file descriptors to cache for observation dmp files (Default Value : 3000)

-obscachesz # of Mbytes cache to use for observations (Default Value : 92)

-ndensity # of densities to initialize per tied state per feature (Default Value : NONE)

-ntrial random initialized K-means: # of trials of k-means w/ random initialization from within corpus (Default Value : 5)

-minratio K-means: minimum convergence ratio, (p_squerr - squerr) / p_squerr (Default Value : 0.01)

-maxiter K-means: maximum # of iterations of updating to apply (Default Value : 100)

-mixwfn Output file for mixing weights (Default Value : NONE)

-meanfn Output file for means (Default Value : NONE)

-varfn Output file for variances (Default Value : NONE)

-method Initialization method. Options: rkm — fnkm (Default Value : rkm)

-reest Reestimate states according to usual definitions assuming vit seg. (Default Value : yes)

-niter # of iterations of reestimation to perform per state (Default Value : 20)

-gthobj Gather what kind of obj state, phn, frame (Default Value : state)

-stride Gather every -stride'th frame (Default Value : 32)

-runlen # of utts to process from ctl file (Default Value : NONE)

-tsoff Begin tied state reestimation with this tied state (Default Value : 0)

-tscnt Reestimate this many tied states (if available) (Default Value : NONE)

-tsrngfn The range of tied states reestimated expressed as offset,count (Default Value : NONE)

-vartiethr Tie variances if # of observations for state exceed this number (Default Value : 0)

A.2.11 map_adapt

Tool Description

Description:

Given a speaker-independent (or other baseline) model and a maximum-likelihood estimate of model parameters from adaptation data, map_adapt will update (interpolate) the mean vectors to maximize the a posteriori probability of the adaptation data.

The ML estimate is generated by running a single iteration of the forward-backward algorithm on the adaptation data, using the baseline models as the initial estimate. This can be accomplished using the programs 'bw' and 'norm'. The -mlmeanfn and -mlvarfn arguments are the output parameter files created by 'norm'. The -mlcountfn argument is the file containing observation counts, which is generated with the -dcountfn argument to 'norm'.

Example

Example: map_adapt -mapmeanfn map_model/means -mlmeanfn ml_model/means -mlvarfn ml_model/variances -simeanfn baseline/means -sivarfn baseline/variances -mlcntfn bwaccumdir/gauden_counts

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-mapmeanfn The output MAP mean file (Default Value : NONE)

-simeanfn The input speaker-independent (baseline) mean file (Default Value : NONE)

-sivarfn The input speaker-independent (baseline) var file (Default Value : NONE)

-mlmeanfn The input ML mean file (output of 'norm' on adaptation data) (Default Value : NONE)

-mlvarfn The in ML var file (output of 'norm' on adaptation data) (Default Value : NONE)

-mlcntfn The ML total observation count (output of 'norm -dcountfn') file (Default Value : NONE)

-dnom_weight The prior adaptation weight (Default Value : 1.0)

A.2.12 make_quests

Tool Description

Description:

This is an implementation of Dr. Rita Singh's automatic question generation. (Copied from Rita's comment) The current algorithm clusters CI distributions using a hybrid bottom-up top-down clustering algorithm to build linguistic questions for decision trees. (From Arthur : I need to do some tracing before understand it what's the internal of the code)

Example

Example: make_quest -moddeffn mdef -meanfn mean -varfn var -mixwfn mixwfn -npermute 8 -niter 1 -qstperstt 20 -tempfn temp -questfn questions

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn Model definition file of the ci models (Default Value : NONE)

-meanfn means file for tree building using continuous HMMs (Default Value : NONE)

-varfn variances file for tree building using continuous HMMs (Default Value : NONE)

-varfloor The minimum variance (Default Value : 1.0e-08)

-mixwfn PDF's for tree building using semicontinuous HMMs (Default Value : NONE)

-npermute The minimum variance (Default Value : 6)

-niter Number of iterations (Default Value : 0)

-qstperstt something per state (Default Value : 8)

-tempfn File to write temprorary results to (important) (Default Value : /tmp/TEMP.QUESTS)

-questfn File to write questions to (Default Value : NONE)

-type HMM type (Default Value : NONE)

A.2.13 mixw_interp

Tool Description

Description:

A routine that provides and ad-hoc way of speaker adaptation by mixture weight interpolation. SD and SI model's mixture weight are first determined and they act as an inputs of this program. The output is the interpolated mixture weight.

The interpolation is controlled by the value lambda (-sillambda)

Example

Example:

mixw_interp -SImixwfn si_mixw -SDmixwfn sd_mxiw -outmixwfn final_mixw -SIlambad 0.7

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-SImixwfn The SI mixture weight parameter file name (Default Value : NONE)

-SDmixwfn The SD mixture weight parameter file name (Default Value : NONE)

-outmixwfn The output interpolated mixture weight parameter file name (Default Value : NONE)

-SIlambda Weight given to SI mixing weights (Default Value : 0.5)

A.2.14 mk_flat

Tool Description

Description: (Copied from Rita's web page) In flat-initialization, all mixture weights are set to be equal for all states, and all state transition probabilities are set to be equal. Unlike in continuous models, the means and variances of the codebook Gaussians are not given global values, since they are already estimated from the data in the vector quantization step. To flat-initialize the mixture weights, each component of each mixture-weight distribution of each feature stream is set to be a number equal to 1/N, where N is the codebook size. The mixture_weights and transition_matrices are initialized using the executable mk_flat

Example

Example:

mk_flat -moddeffn CFS3.ci.mdef -topo CFS3.topology -mixwfn mixture_weights -tmatfn transition_matrices -nstream 1 -ndensity 1

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn A SPHINX-III model definition file name (Default Value : NONE)

-mixwfn An output SPHINX-III mixing weight file name (Default Value : NONE)

-topo A template model topology matrix file (Default Value : NONE)

-tmatfn An output SPHINX-III transition matrix file name (Default Value : NONE)

-nstream Number of independent observation streams (Default Value : 4)

-ndensity Number of densities per mixture density (Default Value : 256)

A.2.15 mk_mdef_gen

Tool Description

Description:

(Copied from Rita's comment and I think it is a pretty good description.) Multi-function routine to generate mdef for context-independent training, untied training, and all-triphones mdef for state tying. Flow: if (triphonelist) make CI phone list and CD phone list if alltriphones mdef needed, make mdef if (rawphonelist) Make ci phone list, if cimdef needed, make mdef Generate alltriphones list from dictionary if alltriphones mdef needed, make mdef if neither triphonelist or rawphonelist quit Count triphones and triphone types in transcript Adjust threshold according to min-occ and maxtriphones Prune triphone list Make untied mdef

Example

Example: Create CI model definition file mk_mdef_gen -phnlstfn phonefile -ocimdef ci_mdeffile -n_state_pm 3

Create untied CD model definition file mk_mdef_gen -phnlstfn rawphonefile -dictfn dict -fdictfn filler_dict -lsnfn transcription -ountiedmdef untie_mdef -n_state_pm 3 -maxtriphones 10000 Create tied CD model definition file mk_mdef_gen -phnlstfn rawphone -oalltphnmdef untie_mdef -dictfn dict -fdictfn filler_dict -n_state_pm 3.

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-phnlstfn List of phones (Default Value : NONE)

-inCImdef Input CI model definition file. (Default Value : NONE)

nored (Default Value : If)

(Default Value : NONE)

-triphnlstfn A SPHINX-III triphone file name. (Default Value : NONE)

incimdef ignored (Default Value : If)

(Default Value : NONE)

-inCDmdef Input CD model definition file. (Default Value : NONE)

phnlstfn, -incimdef ignored (Default Value : If)

(Default Value : NONE)

-dictfn Dictionary (Default Value : NONE)

-fdictfn Filler dictionary (Default Value : NONE)

-lsnfn Transcripts file (Default Value : NONE)

-n_state_pm No. of states per HMM (Default Value : 3)

-ocountfn Output phone and triphone counts file (Default Value : NONE)

-ocimdef Output CI model definition file (Default Value : NONE)

-oalltphnmdef Output all triphone model definition file (Default Value : NONE)

-ountiedmdef Output untied model definition file (Default Value : NONE)

-minocc Min occurances of a triphone must occur for inclusion in mdef file (Default Value : 1)

-maxtriphones Max. number of triphones desired in mdef file (Default Value : 100000)

A.2.16 mk_mllr_class

Tool Description

Description: Create the senone to mllr class mapping.

Example

Example: mk_mllr_class -nmap mapfile -nclass 4 -cb2mllrfn out.cb2mllr

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-nmap # of codebook -> MLLR class mappings (Default Value : NONE)

-nclass # of MLLR classes to map into (Default Value : NONE)

-cb2mllrfn Codebook-to-MLLR mapping file to create (Default Value : NONE)

A.2.17 mk_model_def

Tool Description

Description:

(Copied from Eric Thayers' comment) Make SPHINX-III model definition files from a variety input sources. One input source is a SPHINX-II senone mapping file and triphone file. Another is a list of phones (in which case the transition matrices are tied within base phone class and the states are untied)

Example

Example: [Under construction]

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn A SPHINX-III model definition file name (Default Value : NONE)

-triphonefn A SPHINX-II triphone file name (Default Value : NONE)

-phonelstfn List of phones (first 4 columns of phone def'ns) (Default Value : NONE)

-mapfn A SPHINX-II senone mapping file name (Default Value : NONE)

-n_cdstate total # of CD senones/tied_states (even though mapfn may reference fewer) (Default Value : NONE)

-n_cistate # of CI senones/tied_states (Default Value : NONE)

-n_tmat # of tied transition matrices (Default Value : NONE)

-n_state_pm # of states/model (Default Value : 5)

A.2.18 mk_s2cb

Tool Description

Description:

Convert s3 means and variances to s2 codebook format. cepstrum, delta, delta-delta and power cepstrum basename could be changed by uers.

Example

Example:

mk_s2cb -meanfn s3mean -varfn s3var -cbdir s2dir -varfloor 0.00001

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-meanfn A SPHINX-III mean density parameter file name (Default Value : NONE)

-varfn A SPHINX-III variance density parameter file name (Default Value : NONE)

-cbdir A directory containing SPHINX-II 1PD codebooks (Default Value : NONE)

-varfloor Minimum variance value (Default Value : NONE)

-cepcb Basename of the cepstrum codebook (Default Value : cep.256)

-dcepcb Basename of the difference cepstrum codebook (Default Value : d2cep.256)

-powcb Basename of the power codebook (Default Value : p3cep.256)

-2dcepcb Basename of the 2nd order difference cepstrum codebook (Default Value : xcep.256)

-meanext Mean codebook extension (Default Value : vec)

-varext Variance codebook extension (Default Value : var)

A.2.19 mk_s2hmm

Tool Description

Description:

Convert s3 model definition file, mixture weight and transition matrices to s2 hmm format.

Example

Example:

mk_s2hmm -meanfn s3mdef -varfn s3mixw -tmat s3tmat -hmmdir s2dir

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn Model definition file for the S3 models (Default Value : NONE)

-tmatfn S3 transition matrix parameter file name (Default Value : NONE)

-tpfloor Transition probability smoothing floor (Default Value : 0.0001)

-mixwfn S3 mixing weight parameter file name (Default Value : NONE)

-hmmdir S2 model output directory (Default Value : NONE)

-hmmext Extension of a SPHINX-II model file. (Default Value : chmm)

-cepsenoext Extension of cepstrum senone weight file (Default Value : ccode)

-dcepsenoext Extension of difference cepstrum senone weight file (Default Value : d2code)

-powsenoext Extension of power senone weight file (Default Value : p3code)

-2dcepsenoext Extension of 2nd order difference cepstrum senone weight file (Default Value : xcode)

-mtype Model type sdm,dhmm (Default Value : sdm)

A.2.20 mk_s2phone

Tool Description

Description:

(Copied from Eric Thayer's comment) * Make a SPHINX-II phone file given a list of phones (in SPHINX-III * format). Ok, this sounds like a bizarre fn, but I did need it * once.

Example

Example:

mk_s2phone -s2phonefn s2.phone -phonelstfn s3.phone

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-s2phonefn A SPHINX-II phone file name (Default Value : NONE)

-phonelstfn List of phones (Default Value : NONE)

(Default Value : NONE)

A.2.21 mk_s2phonemap

Tool Description

Description:

Convert s3 model definition file to s2 phone and map files.

Example

Example:

mk_s2phonemap -moddeffn s3mean -phonefn s2/phone -mapfn s2/map

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn The model definition file for the model to be converted (Default Value : NONE)

-phonefn Output phone file name (Default Value : NONE)

-mapfn Output map file name (Default Value : NONE)

A.2.22 mk_s2sendump

Tool Description

Description: Convert s3 model definition file and s3 mixture weight file to a s2 senddump file.

Example

Example:

mk_s2sendump -moddeffn s3mdef -mixwfn s3mixw -tpfloor 0.0000001 -feattype s2_4x -sendumpfn s2dir/sendump

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn The model definition file for the model inventory to train (Default Value : NONE)

-mixwfn The mixture weight parameter file name (Default Value : NONE)

-sendumpfn Output sendump file name (Default Value : NONE)

-feattype Feature type e.g. s2_4x (Default Value : NONE)

-tpfloor Transition probability smoothing floor (Default Value : 0.0001)

A.2.23 mk_s3gau

Tool Description

Description: Conversion from sphinx 2 codebook to sphinx3 means and variances

Example

Example: mk_s3gau -meanfn s3mean -varfn s3var -cbdir s2dir -feat 4s_12c_24d_3p_1

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-meanfn A SPHINX-III mean density parameter file name (Default Value : NONE)

-varfn A SPHINX-III variance density parameter file name (Default Value : NONE)

-cbdir A directory containing SPHINX-II 1PD codebooks (Default Value : NONE)

-varfloor Minimum variance value (Default Value : NONE)

-cepcb Basename of the cepstrum codebook (Default Value : cep.256)

-dcepcb Basename of the difference cepstrum codebook (Default Value : d2cep.256)

-powcb Basename of the power codebook (Default Value : p3cep.256)

-2dcepcb Basename of the 2nd order difference cepstrum codebook (Default Value : xcep.256)

-meanext Mean codebook extension (Default Value : vec)

-varext Variance codebook extension (Default Value : var)

-fixpowvar Fix the power variance to the SPHINX-II standards (Default Value : false)

-feat 2dd Defines the feature set to use (Default Value : 4s_12c_24d_3p_1)

-ceplen Defines the input feature vector (e.g. MFCC) len (Default Value : 13)

A.2.24 mk_s3mix

Tool Description

Description: Conversion from sphinx 2 hmms to sphinx3 mixture weights

Example

Example: (By Arthur: Not sure, may be obsolete) mk_s3mixw -mixwfn s3mixw -moddeffn s3mdef -hmmdir s2hmmdir

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-mixwfn A SPHINX-III mixture weight parameter file name (Default Value : NONE)

-moddeffn A SPHINX-III model definition file name (Default Value : NONE)

-floor Floor weight value to apply before renormalization (Default Value : NONE)

-ci2cd CD weights initialized to CI weights (-hmmdir is to a set of CI models) (Default Value : false)

-hmmdir A directory containing SPHINX-III models consistent with -moddeffn (Default Value : NONE)

-cepsenoext Extension of cepstrum senone weight file (Default Value : ccode)

-dcepsenoext Extension of difference cepstrum senone weight file (Default Value : d2code)

-powsenoext Extension of power senone weight file (Default Value : p3code)

-2dcepsenoext Extension of 2nd order difference cepstrum senone weight file (Default Value : xcode)

A.2.25 mk_s3tmat

Tool Description

Example

Command line Argument Decription

A.2.26 mk_ts2cb

Tool Description

Description: (copied from Eric's comments) * Create a tied-state-to-codebook mapping file for semi-continuous, * phone dependent or fully continuous Gaussian density tying.

Example

Example: (By Arthur: Not sure, may be obsolete) mk_ts2cb -moddeffn semi -ts2cbfn ts2cb

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-ts2cbfn A SPHINX-III tied-state-to-cb file name (Default Value : NONE)

-moddeffn A SPHINX-III model definition file name (Default Value : NONE)

-tyingtype Output a state parameter def file for fully continuous models (Default Value : semi)

-pclassfn A SPHINX-II phone class file name (Default Value : NONE)

A.2.27 mllr_solve

Tool Description

Description:

Given a set of mean accumulator, mllr_solve can compute the transform matrix based on the maximum likelihood criteria.

The mean vand variance are required to be input in arguements -meanfn and -varfn For linear regression equation y=Ax+b, If you specific only -mllrmult, then only A will be estimated. If you specific only -mllradd, then only b will be estimated.

Example

Example: The simplest case: mllr_solve -outmllrfn output.matrix -accumdir accumdir -meanfn mean -varfn var

Adapt based on only CD-senones mllr_solve -outmllrfn output.matrix -accumdir accumdir -meanfn mean -varfn var -cdonly yes -moddeffn mdef.

Only adapt on A : mllr_solve -outmllrfn output.matrix -accumdir accumdir -meanfn mean -varfn var -mllrmult yes -mllradd no

help and example: mllr_solve -help yes -example yes

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-outmllrfn Output MLLR regression matrices file (Default Value : NONE)

-accumdir Paths containing reestimation sums from bw (Default Value : NONE)

-meanfn Baseline Gaussian density mean file (Default Value : NONE)

-varfn variance (baseline-var, or error-var) file (Default Value : NONE)

-cb2mllrfn Codebook to mllr class mapping index file (If it is given, ignore -cdonly) (Default Value : .1cls.)

-cdonly Use only CD senones for MLLR (If yes, -moddeffn should be given.) (Default Value : no)

-moddeffn Model Definition file (to get CD starting point for MLLR) (Default Value : NONE)

-mllrmult Reestimate full multiplicative term of MLLR adatpation of means (yes/no) (Default Value : yes)

-mllradd Reestimate shift term of MLLR adaptation of means (yes/no) (Default Value : yes)

-varfloor var floor value (Default Value : 1e-3)

A.2.28 mllr_transform

Tool Description

Description:

Given a set of MLLR transform, mllr_transform can transform the mean according to formulat y=Ax+b.

The output and input files are specified by -outmeanfn and -inmeanfn respectively. You may also transform the context- dependent model using the option -cdonly. In that case you need to specify a model definition using -moddeffn.

Example

Example: The simplest case: mllr_transform -inmeanfn inMeans -outmeanfn outMeans -mllrmat matrix

Adapt only on CD phones: mllr_transform -inmeanfn inMeans -outmeanfn outMeans -mllrmat matrix -cdonly yes -moddeffn mdef

Help and example: nmllr_transform -help yes -example yes

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-inmeanfn Input Gaussian mean file name (Default Value : NONE)

-outmeanfn Output Gaussian mean file name (Default Value : NONE)

-mllrmat The MLLR matrix file (Default Value : NONE)

-cb2mllrfn The codebook-to-MLLR class file. Override option -cdonly (Default Value : .1cls.)

-cdonly Use CD senones only. -moddeffn must be given. (Default Value : no)

-varfloor Value of the variance floor. Mainly for smoothing the mean. (Default Value : 1e-2)

-moddeffn Model Definition file. (Default Value : NONE)

-varfn Gaussian variance file name. For smoothing. (Default Value : NONE)

(Default Value : NONE)

A.2.29 norm

Tool Description

Description: compute the HMM's parameter generated by bw.

Example

Example: norm -accumdir accumdir -mixwfn mixw -tmatfn tmat -meanfn mean -varfn variances

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-accumdir Paths containing reestimation sums from bw (Default Value : NONE)

-oaccumdir Path to contain the overall reestimation sums (Default Value : NONE)

-tmatfn Transition prob. matrix file to produce (if any) (Default Value : NONE)

-mixwfn Mixing weight file to produce (if any) (Default Value : NONE)

-meanfn Gaussian density mean file to produce (if any) (Default Value : NONE)

-varfn Gaussian density variance file to produce (if any) (Default Value : NONE)

-regmatfn MLLR regression matrices file to produce (if any) (Default Value : NONE)

-dcountfn Gaussian density count file to produce (Default Value : NONE)

-inmixwfn Use mixing weights from this file if never observed (Default Value : NONE)

-inmeanfn Use mean from this file if never observed (Default Value : NONE)

-invarfn Use var from this file if never observed (Default Value : NONE)

-feat The feature set to use. (Default Value : NONE)

-ceplen The vector length of the source features (e.g. MFCC) (Default Value : NONE)

A.2.30 param_cnt

Tool Description

Description:

Find the number of times each of the triphones listed

in a given model definition file (by -moddeffhn) occured in a set of transcription, specified by -lsnfn

Example

Example : param_cnt -moddeffn mdef -ts2cbfn .cont. -ctlfn controlfile -lsnfn transcripts -dictfn dict -fdictfn fillerdict -paramtype phone

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn Model definition file for the single density HMM's to initialize (Default Value : NONE)

-ts2cbfn Tied-state-to-codebook mapping file (Default Value : NONE)

-ctlfn Control file of the training corpus (Default Value : NONE)

-part Identifies the corpus part number (range 1..NPART) (Default Value : NONE)

-npart Partition the corpus into this many equal sized subsets (Default Value : NONE)

-nskip # of lines to skip in the control file (Default Value : NONE)

-runlen # of lines to process in the control file (after any skip) (Default Value : NONE)

-lsnfn All word transcripts for the training corpus (consistent order w/ -ctlfn!) (Default Value : NONE)

-dictfn Dictionary for the content words (Default Value : NONE)

-fdictfn Dictionary for the filler words (Default Value : NONE)

-segdir Root directory of the training corpus state segmentation files. (Default Value : NONE)

-segext Extension of the training corpus state segmentation files. (Default Value : v8_seg)

-paramtype Parameter type to count 'state', 'cb', 'phone' (Default Value : state)

A.2.31 printp

Tool Description

Description:

Display numerical values of resources generated by Sphinx Current we supoort the following formats

-tmatfn : transition matrix

-mixwfn : mixture weight file

-gaufn : mean or variance

-gaucntn : sufficient statistics for mean and diagonal covariance

-lambdafn : interpolation weight

Currently, some parameters can be specified as intervals such as mixture weight.

You can also specified -sigfig the number of significant digits by you would like to see.

and normalize the parameters by -norm

Example

Example:

Print the mean of a Gaussian: printp -gaufn mean

Print the variance of a Gaussian: printp -gaufn var

Print the sufficient statistic: printp -gaucntfn gaucnt:

Print the mixture weights: printp -mixw mixw

Print the interpolation weight: printp -lambdafn lambda

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-tmatfn The transition matrix parameter file name (Default Value : NONE)

-mixwfn The mixture weight parameter file name (Default Value : NONE)

-mixws Start id of mixing weight subinterval (Default Value : NONE)

-mixwe End id of mixing weight subinterval (Default Value : NONE)

-gaufn A Gaussian parameter file name (either for means or vars) (Default Value : NONE)

-gaucntfn A Gaussian parameter weighted vector file (Default Value : NONE)

-regmatcntfn MLLR regression matrix count file (Default Value : NONE)

-moddeffn The model definition file (Default Value : NONE)

-lambdafn The interpolation weight file (Default Value : NONE)

-lambdamin Print int. wt. >= this (Default Value : 0)

-lambdamax Print int. wt. <= this (Default Value : 1)

-norm Print normalized parameters (Default Value : yes)

-sigfig Number of significant digits in 'e' notation (Default Value : 4)

A.2.32 prunetree

Tool Description

Description: Using prunetree, the bifurcations in the decision trees which resulted in the minimum increase in likelihood are progressively removed and replaced by the parent node. The selection of the branches to be pruned out is done across the entire collection of decision trees globally.

Example

Example:

prunetree -itreedir input_tree_dir -nseno 5000 -otreedir output_tree_dir -moddefn mdef -psetfn questions -minocc 100

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn CI model definition file (Default Value : NONE)

-psetfn Phone set definition file (Default Value : NONE)

-itreedir Input tree directory (Default Value : NONE)

-otreedir Output tree directory (Default Value : NONE)

-nseno # of senones defined by the output trees (Default Value : NONE)

-minocc Prune nodes w/ fewer than this # of observations (Default Value : 0.0)

A.2.33 tiestate

Tool Description

Description : Create a model definition file with tied state from model definition file without tied states.

Example

Example: tiestate -imoddeffn imdef -omoddeffn omdef -treedir trees -psetfn questions

This is an example of the input and output format, I copied from Rita's web page,

This is an hypothetical input of tiestate # triphone: (null) # seno map: (null) # 0.3 5 n_base 34 n_tri 156 n_state_map 117 n_tied_state 15 n_tied_ci_state 5 n_tied_tmat # # Columns definitions #base lft rt p attrib tmat ... state id's ... SIL - - - filler 0 0 1 2 N AE - - - n/a 1 3 4 5 N AX - - - n/a 2 6 7 8 N B - - - n/a 3 9 10 11 N T - - - n/a 4 12 13 14 N AE B T i n/a 1 15 16 17 N AE T B i n/a 1 18 19 20 N AX AX AX s n/a 2 21 22 23 N AX AX B s n/a 2 24 25 26 N AX AX SIL s n/a 2 27 28 29 N AX AX T s n/a 2 30 31 32 N AX B AX s n/a 2 33 34 35 N AX B B s n/a 2 36 37 38 N AX B SIL s n/a 2 39 40 41 N AX B T s n/a 2 42 43 44 N AX SIL AX s n/a 2 45 46 47 N AX SIL B s n/a 2 48 49 50 N AX SIL SIL s n/a 2 51 52 53 N AX SIL T s n/a 2 54 55 56 N AX T AX s n/a 2 57 58 59 N AX T B s n/a 2 60 61 62 N AX T SIL s n/a 2 63 64 65 N AX T T s n/a 2 66 67 68 N B AE AX e n/a 3 69 70 71 N B AE B e n/a 3 72 73 74 N B AE SIL e n/a 3 75 76 77 N B AE T e n/a 3 78 79 80 N B AX AE b n/a 3 81 82 83 N B B AE b n/a 3 84 85 86 N B SIL AE b n/a 3 87 88 89 N B T AE b n/a 3 90 91 92 N T AE AX e n/a 4 93 94 95 N T AE B e n/a 4 96 97 98 N T AE SIL e n/a 4 99 100 101 N T AE T e n/a 4 102 103 104 N T AX AE b n/a 4 105 106 107 N T B AE b n/a 4 108 109 110 N T SIL AE b n/a 4 111 112 113 N T T AE b n/a 4 114 115 116 N

is used as the base to give the following CD-tied model definition file with 39 tied states (senones):

triphone: (null) # seno map: (null) # 0.3 5 n_base 34 n_tri 156 n_state_map 54 n_tied_state 15 n_tied_ci_state 5 n_tied_tmat # # Columns definitions #base lft rt p attrib tmat ... state id's ... SIL - - - filler 0 0 1 2 N AE - - - n/a 1 3 4 5 N AX - - - n/a 2 6 7 8 N B - - - n/a 3 9 10 11 N T - - - n/a 4 12 13 14 N AE B T i n/a 1 15 16 17 N AE T B i n/a 1 18 16 19 N AX AX AX s n/a 2 20 21 22 N AX AX B s n/a 2 23 21 22 N AX AX SIL s n/a 2 24 21 22 N AX AX T s n/a 2 25 21 22 N AX B AX s n/a 2 26 21 27 N AX B

B s n/a 2 23 21 27 N AX B SIL s n/a 2 24 21 27 N AX B T s n/a 2 25 21 27 N AX SIL AX s n/a 2 26 21 28 N AX SIL B s n/a 2 23 21 28 N AX SIL SIL s n/a 2 24 21 28 N AX SIL T s n/a 2 25 21 28 N AX T AX s n/a 2 26 21 29 N AX T B s n/a 2 23 21 29 N AX T SIL s n/a 2 24 21 29 N AX T T s n/a 2 25 21 29 N B AE AX e n/a 3 30 31 32 N B AE B e n/a 3 33 31 32 N B AE SIL e n/a 3 34 31 32 N B AE T e n/a 3 35 31 32 N B AX AE b n/a 3 36 37 38 N B B AE b n/a 3 36 37 39 N B SIL AE b n/a 3 36 37 40 N B T AE b n/a 3 36 37 41 N T AE AX e n/a 4 42 43 44 N T AE B e n/a 4 45 43 44 N T AE SIL e n/a 4 46 43 44 N T AE T e n/a 4 47 43 44 N T AX AE b n/a 4 48 49 50 N T B AE b n/a 4 48 49 51 N T SIL AE b n/a 4 48 49 52 N T T AE b n/a 4 48 49 53 N

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-imoddeffn Untied-state model definition file (Default Value : NONE)

-omoddeffn Tied-state model definition file (Default Value : NONE)

-treedir SPHINX-III tree directory containing pruned trees (Default Value : NONE)

-psetfn Phone set definiton file (Default Value : NONE)

A.2.34 wave2feat

Tool Description

Description:

Create cepstra from audio file.

The main parameters that affect the final output are srate, lowerf, upperf, nfilt, and nfft. Typical values for nfft are 256 and 512. The input format can be raw audio (pcm, two byte signed integer), NIST Sphere (.sph) or MS Wav (.wav).

Typical values are shown on table A.2.34.

Table A.1: Typical paramater values for common sampling rates

srate	8000	11025	16000
lowerf	200	130	130
upperf	3700	5200	16000
nfilt	31	37	40

Example

Example:

This example creates a cepstral file named ”output.mfc” from an input audio file named ”input.raw”, which is a raw audio file (no header information), which was originally sampled at 16kHz.

```
wave2feat
        -i              input.raw
        -o              output.mfc
        -raw            no
        -input_endian   little
        -samprate       16000
        -lowerf         130
```

```
-upperf       6800
-nfilt        40
-nfft         512
```

Command-line Argument Description

Usage: [options]

- **-help** (Default Value : no)
  Description: Shows the usage of the tool

- **-example** (Default Value : no)
  Description: Shows example of how to use the tool

- **-i** (Default Value : NO DEFAULT VALUE)
  Description: Single audio input file

- **-o** (Default Value : NO DEFAULT VALUE)
  Description: Single cepstral output file

- **-c** (Default Value : NO DEFAULT VALUE)
  Description: Control file for batch processing

- **-di** (Default Value : NO DEFAULT VALUE)
  Description: Input directory, input file names are relative to this, if defined

- **-ei** (Default Value : NO DEFAULT VALUE)
  Description: Input extension to be applied to all input files

- **-do** (Default Value : NO DEFAULT VALUE)
  Description: Output directory, output files are relative to this

- **-eo** (Default Value : NO DEFAULT VALUE)
  Description: Output extension to be applied to all output files

- **-nist** (Default Value : no)
  Description: Defines input format as NIST sphere

- **-raw** (Default Value : no)
  Description: Defines input format as raw binary data

- **-mswav** (Default Value : no)
  Description: Defines input format as Microsoft Wav (RIFF)

- **-input_endian** (Default Value : little)
  Description: Endianness of input data, big or little, ignored if NIST or MS Wav

- **-nchans** (Default Value : 1)
  Description: Number of channels of data (interlaced samples assumed)

- **-whichchan** (Default Value : 1)
  Description: Channel to process

- **-logspec** (Default Value : no)
  Description: Write out logspectral files instead of cepstra

- **-feat** (Default Value : sphinx)
  Description: SPHINX format - big endian

- **-mach_endian** (Default Value : little)
  Description: Endianness of machine, big or little

- **-alpha** (Default Value : 0.97)
  Description: Preemphasis parameter

- **-srate** (Default Value : 16000.0)
  Description: Sampling rate

- **-frate** (Default Value : 100)
  Description: Frame rate

- **-wlen** (Default Value : 0.025625)
  Description: Hamming window length

- **-nfft** (Default Value : 512)
  Description: Size of FFT

- **-nfilt** (Default Value : 40)
  Description: Number of filter banks

- **-lowerf** (Default Value : 133.33334)
  Description: Lower edge of filters

- **-upperf** (Default Value : 6855.4976)
  Description: Upper edge of filters

- **-ncep** (Default Value : 13)
  Description: Number of cep coefficients

- **-doublebw** (Default Value : no)
  Description: Use double bandwidth filters (same center freq)

- **-blocksize** (Default Value : 200000)
  Description: Block size, used to limit the number of samples used at a time when reading very large audio files

- **-dither** (Default Value : no)
  Description: Add 1/2-bit noise

- **-verbose** (Default Value : no)
  Description: Show input filenames

A.2.35 QUICK_COUNT

Tool Description

Description:

Generate a set of context-dependent phones list (usually triphones) given a dictionary.

A phone list is first created in the following format:

phone1 0 0 0 0 phone2 0 0 0 0 phone3 0 0 0 0

The phone list of CI model training must be used to generate this.

Next a temporary dictionary is generated, which has all words except the filler word (word enclosed in ++()++). The entry. SIL SIL must be added to the temporary dictionary and the dictionary must be sort in alphabetic order.

-q: a mandatory flag that tell QUICK_COUNT to consider all word pairs while constructing the phone list -p: the formatted phone list -b: a temporary dictionary file -o: output triphone list

Example

Example :

QUICK_COUNT -q yes -p phonelist -b dictionary -o outputlist

Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-v Verbose (Default Value : no)

-q Flag to consider all word pairs (Default Value : yes)

-b Base file or the dictionary file (Default Value : NONE)

-p Phone list file (Default Value : NONE)

-P Another argument for phone list file. Equivalent to -p. (keep it for backward compatibility purpose.) (Default Value : NONE)

-o Output triphone list (Default Value : NONE)

-i Input control file (Default Value : NONE)

-I Another argument for input control file. Equivalent to -i. (Keep it for backward compatibility purpose) (Default Value : NONE)

-f Find examples (?) (Default Value : no)

-S Use single path to make triphones (Default Value : no)

-s Directory of sentences (Default Value : NONE)

A.3 SphinxTrain: Perl Training Scripts

A.3.1 00.verify/verify_all.pl

A.3.2 01.vector_quantize/slave.VQ.pl

A.3.3 02.ci_schmm/slave_convg.pl

A.3.4 03.makeuntiedmdef/make_untied_mdef.pl

A.3.5 04.cd_schmm_untied/slave_convg.pl

A.3.6 05.buildtrees/slave.treebuilder.pl

A.3.7 06.prunetree/prunetree.pl

A.3.8 07.cd-schmm/slave_convg.pl

A.3.9 08.deleted-interpolation/deleted_interpolation.pl

A.3.10 Summary of Configuration Parameter in script level

A.3.11 Notes on parallelization

A.4 CMU-Cambridge LM Toolkit

A.4.1 binlm2arpa

Description

binlm2arpa : Convert a binary format language model to ARPA format.

Input : A binary format language model, as generated by idngram2lm.

Output : An ARPA format language model.

Command Line Syntax:

binlm2arpa -binary .binlm -arpa .arpa [-verbosity 2]

A.4.2 evallm

Input : A binary or ARPA format language model, as generated by idngram2lm. In addition, one may also specify a text stream to be used to compute the perplexity of the language model. The ARPA format language model does not contain information as to which words are context cues, so if an ARPA format lanaguage model is used, then a context cues file may be specified as well.

Output : The program can run in one of two modes.

- **compute-PP** Output is the perplexity of the language model with respect to the input text stream.
- **validate** Output is confirmation or denial that the sum of the probabilities of each of the words in the context supplied by the user sums to one.

Command Line Syntax:

evallm [-binary .binlm — -arpa .arpa [-context .ccs]]

Notes: evallm can receive and process commands interactively. When it is run, it loads the language model specified at the command line, and waits for instructions from the user. The user may specify one of the following commands:

- **perplexity**

  Computes the perplexity of a given text. May optionally specify words from which to force back-off.

  Syntax:

  perplexity -text .text [-probs .fprobs] [-oovs .oov_file] [-annotate .annotation_file] [-backoff_from_unk_inc — -backoff_from_unk_exc] [-backoff_from_ccs_inc — -backoff_from_ccs_exc] [-backoff_from_list .fblist] [-include_unks]

  If the -probs parameter is specified, then each individual word probability will be written out to the specified probability stream file.

  If the -oovs parameter is specified, then any out-of-vocabulary (OOV) words which are encountered in the test set will be written out to the specified file.

  If the -annotate parameter is used, then an annotation file will be created, containing information on the probability of each word in the test set according to the language model, as well as the back-off class

for each event. The back-off classes can be interpreted as follows: Assume we have a trigram language model, and are trying to predict P(C — A B). Then back-off class "3" means that the trigram "A B C" is contained in the model, and the probability was predicted based on that trigram. "3-2" and "3x2" mean that the model backed-off and predicted the probability based on the bigram "B C"; "3-2" means that the context "A B" was found (so a back-off weight was applied), "3x2" means that the context "A B" was not found.

To force back-off from all unknown words, use the -backoff_from_unk_inc or -backoff_from_unk_exc flag (the difference being the difference between inclusive or exclusive forced back-off). To force back-off from all context-cues, use the -backoff_from_ccs_inc or -backoff_from_ccs_inc flag. One can also specify a list of words from which to back-off, by storing this list in a forced back-off list file and using the -backoff_from_list switch.

-include_unks results in a perplexity calculation in which the probability estimates for the unkown word are included.

- **validate**

  Calculate the sum of the probabilities of all the words in the vocabulary given the context specified by the user.

  Syntax:

  validate [-backoff_from_unk_inc — -backoff_from_unk_exc] [-backoff_from_ccs_inc — -backoff_from_ccs_exc] [-backoff_from_list .fblist] word1 word2 ... word_(n-1)

  Where n is the n in n-gram.

- **help** Displays a help message.

  Syntax: help

- **quit** Exits the program.

  Syntax:

  quit

Since the commands are read from standard input, a command file can be piped into it directly, thus removing the need for the program to run interactively:

```
$ echo "perplexity -text b.text" | evallm -binary a.binlm
```

A.4.3 idngram2lm

Input : An id n-gram file (in either binary (by default) or ASCII (if specified) mode).

Output : A list of the frequency-of-frequencies for each of the 2-grams, ... , n-grams, which can enable the user to choose appropriate cut-offs, and to specify appropriate memory requirements with the **-spec_num** option in idngram2lm.

Command Line Syntax:

idngram2lm -idngram .idngram -vocab .vocab -arpa .arpa — -binary .binlm [-context .ccs] [-calc_mem — -buffer 100 — -spec_num y ... z] [-vocab_type 1] [-oov_fraction 0.5] [-two_byte_bo_weights [-min_bo_weight nnnnn] [-max_bo_weight nnnnn] [-out_of_range_bo _weights]] [-four_byte_counts] [-linear — -absolute — -good_turing — -witten_bell] [-disc_ranges 1 7 7] [-cutoffs 0 ... 0] [-min_unicount 0] [-zeroton_fraction] [-ascii_input — -bin_input] [-n 3] [-verbosity 2]

Usage:

The -context parameter allows the user to specify a file containing a list of words within the vocabulary which will serve as context cues (for example, markers which indicate the beginnings of sentences and paragraphs).

-calc_mem, -buffer and -spec_num x y ... z are options to dictate how it is decided how much memory should be allocated for the n-gram counts data structure. -calc_mem demands that the id n-gram file should be read twice, so that we can accurately calculate the amount of memory required. -buffer allows the user to specify an amount of memory to grab, and divides this memory equally between the 2,3, ..., n-gram tables. -spec_num allows the user to specify exactly how many 2-grams, 3-grams, ... , and n-grams will need to be stored. The default is -buffer STD_MEM.

The toolkit provides for three types of vocabulary, which each handle out-of-vocabulary (OOV) words in different ways, and which are specified using the -vocab_type flag.

A closed vocabulary (-vocab_type 0) model does not make any provision for OOVs. Any such words which appear in either the training or test data will cause an error. This type of model might be used in a command/control environment where the vocabulary is restricted to the number of commands that the system understands, and we can therefore guarantee that no OOVs will occur in the training or test data.

An open vocabulary model allows for OOVs to occur; out of vocabulary words are all mapped to the same symbol. Two types of open vocabulary model are implemented in the toolkit. The first type (-vocab_type 1) treats

this symbol the same way as any other word in the vocabulary. The second type (-vocab_type 2) of open vocabulary model is to cover situations where no OOVs occurred in the training data, but we wish to allow for the situation where they could occur in the test data. This situation could occur, for example, if we have a limited amount of training data, and we choose a vocabulary which provides 100proportion of the discount probability mass (specified by the -oov_fraction option) is reserved for OOV words.

The discounting strategy and its parameters are specified by the -linear, -absolute, -good_turing and -witten_bell options. With Good Turing discounting, one can also specify the range over which discounting occurs, using the -disc_ranges option.

The user can specify the cutoffs for the 2-grams, 3-grams, ..., n-grams by using the -cutoffs parameter. A cutoff of K means that ¿ n-grams occurring K or fewer times are discarded. If the parameter is omitted, then all the cutoffs are set to zero. The -zeroton_fraction option specifies that P(zeroton) (the unigram probability assigned to a vocabulary word that did not occurred at all in the training data) will be at least that fraction of P(singleton) (the probability assigned to a vocabulary word that occurred exactly once in the training data).

By default, the n-gram counts are stored in two bytes by use of a count table (this allows the counts to exceed 65535, while keeping the data structures used to store the model compact). However, if more than 65535 distinct counts need to be stored (very unlikely, unless constructing 4-gram or higher language models using Good-Turing discounting), the -four_byte_counts option will need to be used.

The floating point values of the back-off weights may be stored as two-byte integers, by using the -two_byte_alphas switch. This will introduce slight rounding errors, and so should only be used if memory is short. The -min_alpha, -max_alpha and -out_of_range_alphas are parameters used by the functions for using two-byte alphas. Their values should only be altered if the program instructs it. For further details, see the comments in the source file src/two_byte_alphas

A.4.4 idngram2stats

indngram2stats : Report statistics for an id n-gram file.

Input : An id n-gram file (in either binary (by default) or ASCII (if specified) mode).

Output : A list of the frequency-of-frequencies for each of the 2-grams, ... , n-grams, which can enable the user to choose appropriate cut-offs, and to specify appropriate memory requirements with the **-spec_num** option in idngram2lm.

Command Line Syntax:

idngram2stats [-n 3] [-fof_size 50] [-verbosity 2] [-ascii_input] < .idngram > .stats

A.4.5 interpolate

Input : Files containing probability streams, as generated by the -probs option of the perplexity command of evallm. Alternatively these probabilities could be generated from a seperate piece of code, which assigns word probabilities according to some other language model, for example a cache-based LM. This probability stream can then be linearly interpolated with one from a standard n-gram model using this tool.

Output : An optimal set of interpolation weights for these probability streams, and (optionally) a probability stream corresponding to the linear combination of all the input streams, according to the optimal weights. The optimal weights are calculated using the expectation maximisation (EM) algorithm.

Command Line Syntax :

interpolate +[-] model1.fprobs +[-] model2.fprobs ... [-test_all — -test_first n — -test_last n — -cv] [-tag .tags] [-captions .captions] [-in_lambdas .lambdas] [-out_lambdas .lambdas] [-stop_ratio 0.999] [-probs .fprobs] [-max_probs 6000000]

The probability stream filenames are prefaced with a + (or a +- to indicate that the weighting of that model should be fixed).

There are a range of options to determine which part of the data is used to calculate the weights, and which is used to test them. One can test the perplexity of the interpolated model based on all the data, using the -test_all option, in which case a set of lambdas must also be specified with the -lambda option (ie the lambdas are pre-specified, and not calculated by the program). One can specify that the first or last n items are the test set by use of the -test_first n or -test_last n options. Or one can perform two-way cross validation using the -cv option. If none of these are specified then the whole of the data is used for weight estimation.

By default, the initial interpolation weights are fixed as 1/number_of_models, but alternative values can be stored in a file and used via the -in_lambdas option.

The -probs switch allows the user to specify a filename in which to store the combined probability stream. The optimal lambdas can also be stored in a file by use of the -out_lambdas command.

The program stops when the ratio of the test-set perplexity between two successive iterations is above the value specified in the -stop_ratio option.

The data can be partitioned into different classes (with optimisation being performed seperately on each class) using the -tags parameter. The tags file will contain an integer for each item in the probability streams

corresponding to the class that the item belongs to. A file specified using the -captions option will allow the user to attach names to each of the classes. There should be one line in the captions file for each tag, with each line corresponding to the name of the tag.

The amount of memory allocated to store the probability streams is dictated by the -max_probs option, which indicates the maximum number of probabilities allowed in one stream.

A.4.6 mergeidngram

Input : A set of id n-gram files (in either binary (by default) or ASCII (if specified) format - note that they should all be in the same format, however).

Output : One id n-gram file (in either binary (by default) or ASCII (if specified) format), containing the merged id n-grams from the input files.

Notes : This utility can also be used to convert id n-gram files between ascii and binary formats.

Command Line Syntax:

mergeidngram [-n 3] [-ascii_input] [-ascii_output] .idngram_1 .idngram_2 ... .idngram_N ¿ .idngram

A.4.7 ngram2mgram

Input : Either a word n-gram file, or an id n-gram file.

Output : Either a word m-gram file, or an id m-gram file, where m ¡ n.

Command Line Syntax:

ngram2mgram -n N -m M [-binary — -ascii — -words] < .ngram > .mgram

The **-binary**, **-ascii**, **-words** correspond to the format of the input and output (Note that the output file will be in the same format as the input file). **-ascii** and **-binary** denote id n-gram files, in ASCII and binary formats respectively, and -words denotes a word n-gram file.

A.4.8 text2idngram

Input : Text stream, plus a vocabulary file.

Output : List of every id n-gram which occurred in the text, along with its number of occurrences.

Notes : Maps each word in the text stream to a short integer as soon as it has been read, thus enabling more n-grams to be stored and sorted in memory.

Command Line Syntax:

text2idngram -vocab .vocab [-buffer 100] [-temp /usr/tmp/] [-files 20] [-gzip — -compress] [-n 3] [-write_ascii] [-fof_size 10] [-verbosity 2] < .text > .idngram

By default, the id n-gram file is written out as binary file, unless the **-write_ascii** switch is used.

The size of the buffer which is used to store the n-grams can be specified using the -buffer parameter. This value is in megabytes, and the default value can be changed from 100 by changing the value of STD_MEM in the file src/toolkit.h before compiling the toolkit.

The program will also report the frequency of frequency of n-grams, and the corresponding recommended value for the -spec_num parameters of idngram2lm. The -fof_size parameter allows the user to specify the length of this list. A value of 0 will result in no list being displayed.

The -temp option allows the user to specify where the program should store its temporary files.

In the case of really huge quantities of data, it may be the case that more temporary files are generated than can be opened at one time by the filing system. In this case, the temporary files will be merged in chunks, and the -files parameter can be used to specify how many files are allowed to be open at one time.

A.4.9 text2wfreq

Input : Text stream

Output : List of every word which occurred in the text, along with its number of occurrences.

Notes : Uses a hash-table to provide an efficient method of counting word occurrences. Output list is not sorted (due to "randomness" of the hash-table), but can be easily sorted into the user's desired order by the UNIX sort command. In any case, the output does not need to be sorted in order to serve as input for wfreq2vocab.

Usage: text2wfreq [-hash 1000000] [-verbosity 2] < .text > .wfreq

Higher values for the -hash parameter require more memory, but can reduce computation time.

A.4.10 text2wngram

Input : Text stream

Output : List of every word n-gram which occurred in the text, along with its number of occurrences.

The maximum numbers of charactors and words that can be stored in the buffer are given by the -chars and -words options. The default number of characters and words are chosen so that the memory requirement of the program is approximately that of STD_MEM, and the number of charactors is seven times greater than the number of words.

The -temp option allows the user to specify where the program should store its temporary files.

Usage : text2wngram [-n 3] [-temp /usr/tmp/] [-chars 63636363] [-words 9090909] [-gzip — -compress] [-verbosity 2] < .text > .wngram

A.4.11 wfreq2vocab

Example

wfreq2vocab : Generate a vocabulary file from a word frequency file.

Input : A word unigram file, as produced by text2wfreq

Output : A vocabulary file.

The **-top** parameter allows the user to specify the size of the vocabulary; if the program is called with the command -top 20000, then the vocabulary will consist of the most common 20,000 words.

The **-gt** parameter allows the user to specify the number of times that a word must occur to be included in the vocabulary; if the program is called with the command -gt 10, then the vocabulary will consist of all the words which occurred more than 10 times.

If neither the -gt, nor the -top parameters are specified, then the program runs with the default setting of taking the top 20,000 words.

The **-records** parameter allows the user to specify how many of the word and count records to allocate memory for. If the number of words in the input exceeds this number, then the program will fail, but a high number will obviously result in a higher memory requirement.

Usage : wfreq2vocab [-top 20000 — -gt 10] [-records 3000000] [-verbosity 2] < .wfreq > .vocab

A.4.12 wngram2idngram

Input : Word n-gram file, plus a vocabulary file.

Output : List of every id n-gram which occurred in the text, along with its number of occurrences, in either ASCII or binary format.

Note : For this program to be successful, it is important that the vocabulary file is in alphabetical order. If you are using vocabularies generated by the wfreq2vocab tool then this should not be an issue, as they will already be alphabetically sorted.

Command Line Syntax:

wngram2idngram -vocab .vocab [-buffer 100] [-hash 200000] [-temp /usr/tmp/] [-files 20] [-gzip — -compress] [-verbosity 2] [-n 3] [-write_ascii] < .wngram > .idngram

The size of the buffer which is used to store the n-grams can be specified using the **-buffer** parameter. This value is in megabytes, and the default value can be changed from 100 by changing the value of STD_MEM in the file src/toolkit.h before compiling the toolkit.

The program will also report the frequency of frequency of n-grams, and the corresponding recommended value for the **-spec_num** parameters of idngram2lm. The **-fof_size** parameter allows the user to specify the length of this list. A value of 0 will result in no list being displayed.

Higher values for the **-hash** parameter require more memory, but can reduce computation time.

The **-temp** option allows the user to specify where the program should store its temporary files.

The **-files** parameter is used to specify the number of files which can be open at one time.

Bibliography

[1] R. Bellman, *Dynamic programming*, 1957.

Index