audioseg

Audio Segmentation Toolkit, release 1.2.
Last updated 14 January 2010.

Guillaume Gravier
Michaël Betser
Mathieu Ben

Table of Contents

1 Introduction

1.1 What is audioseg?

audioseg is a toolkit dedicated to audio segmentation and indexing, i.e. dedicated to segmenting a stream of audio descriptors, or features, into segments of the same nature. In particular, audioseg provides tools, or commands, for the following

- silence / audio activity detection,
- blind segmentation with the Bayesian information criterion (BIC),
- segment clustering,
- segment classification with Gaussian mixture models (GMM), and
- joint segmentation and classification using hidden Markov models (HMM).

The toolkit is not meant to be an "out-of-the-box" toolkit for audio segmentation and indexing but rather to be a set of commands to help prototyping and developing such applications.

As mentioned above, audioseg commands operate on streams of descriptors or feature vectors. However, no command for computing feature vectors from the audio signal is included in the toolkit which relies on a companion toolkit, SPro 4.0[1] (release 4.0 and later), for this purpose. In particular, audioseg makes use of the SPro library for audio feature I/Os. Current release of audioseg also rely on the GNU Scientific Library[2] for Gaussian modeling with full covariance matrices.

The toolkit comes with a library, written in C language, which provides the following functionalities

- segmentation I/O and manipulation, and
- GMM I/O, likelihood computation and parameter estimation.

As for now, some of the functionalities provided by the audioseg commands, such as silence detection or Viterbi decoding are not included in the library and cannot be embedded in C applications. To do so, one would have to copy the code from the corresponding command. However, we have tried to make the commands versatile enough so as to make prototyping an application with scripts easy.

1.2 How to read this manual?

The manual is divided into two main parts.

Chapter 3 [Segmenting audio streams], page 9, to Chapter 5 [HMM-based segmentation], page 25, correspond to the user manual and describe in details the techniques implemented and the use of the corresponding commands. In particular, Chapter 3 [Segmenting audio streams], page 9, details the audioseg segmentation file format and the commands for segmenting an input stream of descriptors into homogeneous segments. Chapter 4 [Gaussian mixture models], page 17, describes all the commands related to Gaussian mixture modeling

[1] http://gforge.inria.fr/projects/spro

[2] http://www.gnu.org/software/gsl

of the descriptors while Chapter 5 [HMM-based segmentation], page 25, is dedicated to joint segmentation and classification using hidden Markov models.

Chapter 6 [Quick reference guide], page 27, is intended as a reference manual and provides a summary of the `audioseg` commands syntax, synopsis and options.

1.3 Installing `audioseg`

Installation follows the standard GNU installation procedure. The following instructions were adapted from the generic '`INSTALL`' file distributed with GNU `autoconf`. The section below reproduces the '`INSTALL`' file in `audioseg` root directory.

1.3.1 Installation

1. `cd` to the the package root directory and type `./configure` to configure the package for your system. If you are using `csh` on an old version of System V, you might need to type `sh ./configure` instead to prevent `csh` from trying to execute `configure` itself. See below for `audioseg` specific options to `configure`.
2. Type `make` to compile the package.
3. Optionally, type `make check` to run any self-tests that come with the package.
4. Type `make install` to install the programs, the library archive and header and documentation.
5. You can remove the program binaries and object files from the source code directory by typing `make clean`. To also remove the files that `configure` created (so you can compile the package for a different kind of computer), type `make distclean`.

1.3.2 Compilers and Options

Some systems require unusual options for compilation or linking that the '`configure`' script does not know about. Run `./configure --help` for details on some of the pertinent environment variables.

You can give `configure` initial values for variables by setting them in the environment. You can do that on the command line like this

```
./configure CFLAGS="-Wall -O3"
```

Compiling `audioseg` with the '-O3' is recommended.

1.3.3 Installation Names

By default, `make install` will install the package files in '`/usr/local/bin`', '`/usr/local/man`', etc. You can specify an installation prefix other than '`/usr/local`' by giving `configure` the option '`--prefix=`*`path`*'.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give `configure` the option '`--exec-prefix=`*`path`*', the package will use *path* as the prefix for installing programs and libraries. Documentation and other data files will still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like '`--bindir=`*`path`*' to specify different values for particular kinds of files. Run `configure --help` for a list of the directories you can set and what kinds of files go in them.

1.3.4 Optional Features

`audioseg` relies on several external packages and libraries, namely SPro 4.0, SPHERE 2.6 and GNU Scientific Library (GSL) 2.1. The two last packages are optional but some functionalities of `audioseg` will be disabled if the packages are not found. `configure` will try to locate the necessary packages automatically in standard places if no specific location is specified using the corresponding '`--with-PACKAGE=`*path*' option. If *path* is '`no`', `configure` will not try to locate the packages and proceed as if the latter were not installed on your system.

Option '`--with-spro=`*path*' can be used to specified the location of the SPro library and header.

Option '`--with-sphere=`*path*' may be used to turn on support of the SPHERE library. This is necessary if the version of the SPro library you are linking with was initially compiled with support for this library. Otherwise, this is not strictly necessary except in `ssad` where the SPHERE library support enables direct I/O of waveforms in SPHERE format. If *path* is omitted, support of the SPHERE library is turned on and the library is searched for at standard places.

Option '`--with-gsl=`*path*' will force support for the GSL library, specifying the location of the GSL library. *path* may be omitted. By default, `audioseg` will search for the GSL library and use it if found. If the GSL library is not found, options for full covariance Gaussian modeling will not be activited.

1.4 Reporting bugs

Bugs should be reported to ggravier@irisa.fr. Feel free to submit a diagnostic or even a patch along with your bug report if you kindly bothered to do the trouble-shooting. This is always appreciated.

1.5 Contributors

The current release of `audioseg` benefited from the initial work of Grégoire Colbert who developed a Gaussian mixture modeling library from which the GMM related stuff in `audioseg` was adapted.

2 File formats

This section of the manual is dedicated to the description of the file formats manipulated by `audioseg`.

2.1 Descriptor file format

We briefly describe the SPro feature stream file format before discussing on-the-fly convertion.

2.1.1 File format

No commands for computing descriptors from a waveform are provided in `audioseg` which rather rely on the companion toolkit SPro for this. Writing your own descriptor extraction tool is also possible provided the output format is the SPro feature stream format. The best solution to write feature streams in this particular format is to write a code that uses the SPro library I/O functions. Otherwise, it's up to you to write your own I/O functions to convert your data in the appropriate format. Here under is a brief description of the SPro feature stream file format.

In its simplest form, the SPro feature stream file format for descriptors is a binary format which consists of a header followed by the feature vectors stored as float in time increasing order. The header is made of three fields: feature vector dimension (`unsigned short`), feature description flag (`long`) and frame rate in Hz (`float`). The feature description flag is actually a field of bits which describes the feature vectors content (energy, delta coefficients, mean normalization, etc). In most cases, setting this flag to zero should work.

See section "File formats" in *SPro 4.* Manual*, for more details.

2.1.2 Conversions

Every command that inputs SPro feature streams support the option '`--convert=`*`str`*', where *str* is a string containing SPro stream description flags among '`ZRDAN`'. Each letter correspond to a particular processing of the SPro features that is made right after reading the features on disk and before processing them. Each letter has the following meaning:

'`Z`' remove mean of static coefficients[1].

'`R`' normalize variance of static coefficients (requires '`Z`')

'`D`' add delta features

'`A`' add delta-delta features (requires '`D`')

'`N`' remove static log-energy (requires '`D`' and input features with energy)

As an example, the following command

```
sbic --convert=ZD foo.mfcc foo.seg
```

[1] Note that in SPro 4.0, energy is not affected by this convertion. In subsequent releases, energy is also normalized. The same applies for variance reduction.

would read MFCC features from file '`foo.mfcc`', before subtracting the mean ('`Z`') and adding the first order derivatives ('`D`') to the features, thus having `sbic` operate on mean normalized features with first order derivatives.

See section "Convertion flags" in *SPro 4.0 Manual*, for more details.

2.2 Segmentation file format

A segmentation is a collection of *segments*, generally time ordered, where a segment is a portion of the input stream. A segment is most often characterized by a *label* and the start and end times. Optionally, a score can be associated to a segment. The label is normally used to describe the content of a segment. Since multiple events can occur simultaneously, a label is made of names, separated by a plus sign ('`+`'). Labels can have one or several names but at least one must be given in order to identify the segment. Start and end times are given in seconds. More generally, the time unit in `audioseg` is the second and all times must be given in seconds, relative to the beginning of the stream. Finally, the score is a real valued number, typically the segment likelihood with a given model.

Segmentations are stored on disk as a text file where each line correspond to a segment and has the following syntax

```
label [start_time [end_time [score]]] [# comment]
```

Fields are separated by blanks, be they spaces or tabulations. Fields in square brackets are optional. Empty lines and comment lines starting with '`#`' are authorized. As mention previously, the label is mad of names separated by '`+`', where a name is an arbitrary string. Note that the name strings shall contain no space (' '), tabulation ('`\t`'), comment ('`#`') or plus ('`+`') characters as these are reserved respectively as field, comment and name separators.

The following is a simple example of a segmentation file with two classes ('`spk1`' and '`spk2`').

```
# This is a comment followed by an empty line
# (just to show it's possible)

spk1       0.00  1.04   -9.184351e+02
spk2       1.04  1.22   -5.754312e+00
spk1+spk2  1.22  1.57   -4.319016e+01
```

This is a typical example of a segmentation in speaker indexing tasks where '`spk1`' speaks for the 1.04 first seconds of the document with a likelihood equals to -918.4, followed by '`spk2`' and a small portion of overlapping speech from both speakers from 1.22 to 1.57 seconds.

2.3 Script file format

A *script* is a collection of descriptor stream files, possibly with an associated segmentation. Typically, scripts are used for GMM parameter estimation from several input files to avoid command line too long.

From the user's point of view, a script is a text file with one entry per line, where an entry is the filename of a stream of descriptors possibly followed with the filename of the corresponding segmentation, the two filenames being separated by spaces (either blanks ('

') or tabs ('`\t`')). Comments can be inserted using ('`#`') as in shell programs. The following is a simple example of a two file script.

```
# This is a comment followed by an empty line

/usr/toto/file1.prm /usr/toto/file1.seg
/usr/toto/file2.prm /usr/toto/file2.seg
```

Most of the `audioseg` commands support scripts via the option '`--file-list=`*fn*', where *fn* is the script filename.

2.4 I/Os and byte-order

To avoid problems with heterogeneous networks where big and little endian disks and machines can be found, all binary file formats in `audioseg` are independent on the machine endianness. Binary file formats are used for the SPro input feature files and for GMM and Gaussian models. See Chapter 4 [Gaussian mixture models], page 17, for details on GMM related tools.

All binary files are in little-endian byte order[2] (i.e. Intel-like processor byte order). SPro and model I/O functions automatically do the byte swapping if necessary, i.e. if running on a platform with a Motorola-like processor. But, if you want to generate SPro descriptor files by yourself, you must pay attention to the byte-order and, if running on a big-endian machine, perform byte swapping before writing the data to disk. If generating GMM by yourself, the best solution is to use the import feature of `sgmcopy`. See Section 4.5 [Import/Export], page 22.

All other files being in text format, no byte-order problems are expected. However, supposing you managed to recompile `audioseg` on a Windows-based OS, you may have to consider converting the text files, for example with `unix2dos` and `dos2unix`. Output will probably be in '`iso-8859-1`' encoding.

[2] The choice of little-endian byte order comes from the fact that the authors mainly work with Linux on Intel-like machines and do *not* want to bother with swap on their machines.

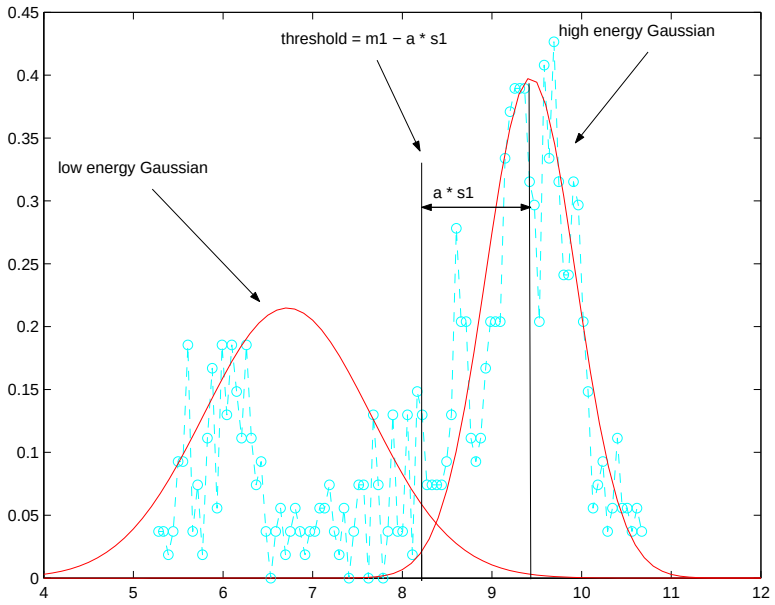
3 Segmenting audio streams

This section of the manual describes `audioseg` commands dedicated to the segmentation of a stream of audio descriptors. Such commands take as input a stream of descriptors to output a *segmentation*.

3.1 Silence detection

One of the first step of audio segmentation systems is to detect the portions of the input audio stream that exhibit some audio activity or, equivalently, the portions of silence.

A popular approach for audio activity detection is the bi-Gaussian model of the stream *energy profile*, where the energy profile is the frame energy or log-energy sequence. Based on this model, audio activity detection is a two step process. The first step consists in estimating a two Gaussian model from the profile's points. Obviously, the Gaussian with the smallest mean correspond to silence frames while the other Gaussian corresponds to frames of audio activity. In the second step, frames are tagged as silence or not based on the bi-Gaussian model, using either a maximum likelihood criterion or a threshold determined from the audio frame mean. The threshold is set as the mean of the "high" energy Gaussian shifted by a fraction of the standard deviation. The figure below illustrates the audio activity detection process.



Silence detection with `ssad`

The command `ssad` implements the bi-Gaussian silence and audio activity detection as explained above. Unlike other `audioseg` commands, the input is a waveform stream. The output is a segmentation indicating the silence segments or the speech segments or even both. Input and output to `stdin` and `stdout` respectively are possible using a dash ('`-`') for filename. The default setting implements a maximum likelihood classification based on the frame energies for silence detection. However, option '`--log`' can be used to operate

on the log-energy rather than the energy. Threshold-based selection is also possible via the '`--threshold=a`' option where *a* specifies the fraction of the standard deviation to be removed from the high energy Gaussian mean to set the threshold. Options '`--reverse`' and '`--all`' control the output, where the first one indicates audio activity detection rather than silence detection and the second one triggers the output of both types of segments. Audio activity and silence segments are given the arbitrary labels '`speech`' and '`sil`' respectively. Note that label '`speech`', whose name was chosen for historical reasons, does not necessarily correspond to actual speech but to any kind of audio activity.

One problem with bi-Gaussian classification is that the decision is made at the frame level. This may result in unrealistic segmentations with very short segments containing a few frames. To avoid this, `ssad` lets you specified a minimum silence segment length using the '`--min-length=d`' option. If this option is set, every silence segment whose duration is less than *d* seconds will be discarded. Note that minimum duration applies only to silence segments, not to audio activity segments which can be very short.

Supported input waveform file format are raw PCM mono, 16 bits/sample format, A-Law or Mu-Law mono, 8 bits/sample format, WAVE and possibly SPHERE if the package was compiled to support the latter. Use '`--format=str`' to specify the input file format which defaults to raw. Frame length and shift are specified with '`--length`' and '`--shift`' respectively while '`--window=str`' may be used to specify a frame weighting window.

Selecting high-energy frames

In some applications, such as speaker recognition, it might be usefull to select descriptors corresponding to high-energy frames. Furthermore, the selected features are often normalized to zero mean and unit variance. Frame normalization can be performed using `spfnorm`, possibly relying on the output of `ssad` for frame selection. Typically, the two commands

```
ssad --reverse foo.pcm16 foo.seg
spfnorm -S foo.seg foo.mfcc foo.mfcc_zr
```

will first write a segmentation corresponding to high energy frames to file '`foo.seg`' before normalizing those features in file '`foo.mfcc`' that corresponds to high-energy frames, writing the result to file '`foo.mfcc_zr`'.

3.2 Change detection

Detecting abrupt changes in the audio stream is another crucial segmentation step which results in the stream being segmented into "homogeneous" portions. Usually, homogeneous is to be understood in the sense of a uniform audio content though, in practice, this is not often the case. Indeed, as explained below, change detection algorithms search segments for which the feature vectors have the same probability distribution. Segments satisfying this criterion does not necessary have the same semantic or audio content.

3.2.1 Change detection algorithms

Abrupt change detection is usually carried out based on the Bayesian information criterion (BIC), or some other related criterion. The basic idea is to detect, within a sliding window of descriptors, whether a change is present or not at the window center, by comparing the models for the data using respectively two and one distribution. If the data is better modeled by two distributions, a change is detected. As a given set of data is always better

modeled by two distributions than by a single one, an information criterion difference is used rather than a generalized likelihood ratio (GLR) to hypothesize a change. Most information criteria can be seen as a log likelihood ratio penalized by the model complexity and the amount of training data for the model.

In `audioseg`, the Bayesian information criterion (BIC) is used along with Gaussian modeling of the descriptors. The information criterion difference for two consecutive segments $a = \{y_1 \dots y_t\}$ and $b = \{y_{t+1} \dots y_T\}$ is given by

$$\Delta BIC(t) = -R(t) + \frac{\lambda}{2}\left(p + \frac{p(p+1)}{2}\right)\log(T)$$

where p is the descriptor vector dimension and $R(t)$ is the log likelihood ratio given by

$$R(t) = \frac{T}{2}\log|\Sigma_{ab}| - \frac{T_a}{2}\log|\Sigma_a| - \frac{T_b}{2}\log|\Sigma_b| \ .$$

In the above equation, T_a is the number of frames in a and Σ_a is the covariance matrix estimated from the descriptor vectors of a. Subscript ab denotes the union of the two segments a and b. Positive values of the criterion indicates possible changes.

Based on this principle, a three step segmentation algorithm can be defined[1]. The first step aims at detecting candidate changes in the stream. This is implemented using a window in which an hypothetical change point is varied. The difference $\Delta BIC(i)$ is computed for every position i of the hypothetical change point within the window. A candidate change is detected if $\max_i \Delta BIC(i) > 0$, the argument of this expression to the change occurrence time. If a change is detected, segmentation proceeds from the detected change. Otherwise, the window size is incremented and a new search is started.

The first step is usually carried out with a large window and a rather large increment between two consecutive hypothetical change points, thus resulting in rough estimates of the change occurrence times. The second pass aims at readjusting the latter, using the same principle as for the first pass with a window centered on the hypothesized change and a smaller increment. If a candidate change is detected within the window, its time will be used as the new change occurrence time. Otherwise, the candidate change of the first pass is discarded.

Finally, the last step validates the segmentation obtained after the second step. The principle is to compute the BIC difference between two consecutive segments. If the difference is negative, the change between those two segments is discarded. Otherwise, the change is validated.

The three step change detection algorithm can be rather slow if the input stream contains long homogeneous segments, in particular because of the first pass. A faster, single step, algorithm is also implemented. The single step algorithm computes the BIC difference at regular intervals for a fixed segment length, hypothesizing a change in the middle of the segment. An *emergence* criterion is then extracted from the BIC difference profile and a change is detected whenever the emergence criterion is above a threshold. The emergence criterion $I(t)$ aims at extracting the major significant peaks of the BIC difference profile.

[1] A. Tritschler. A segmentation-enabled speech recognition application using the BIC criterion. Ph. D. Thesis, Institut Eurecom, France, 1998.

$I(t)$ is obtained by subtracting to $\Delta BIC(t)$ the max of the left and right minima, where the minima are the minimum ΔBIC profile values respectively on the portions of the profile respectively to the right and to the left of t and for which $\Delta BIC(t-\tau)$ or $\Delta BIC(t+\tau)$ is less than $\Delta BIC(t)$.

3.2.2 Change detection with `sbic`

Change detection is performed with the command `sbic` which implements two algorithms derived from the BIC segmentation principle, as explained in the previous section.

The command reads a stream of descriptors and writes the corresponding segmentation with arbitrary segment labels '`unk`'. Input and output to `stdin` and `stdout` respectively are possible using a dash ('`-`') for filename. If `sbic` is also given an input segmentation, change point detection will be carried out independently in each segment of the input segmentation. Use '`--segmentation=`*`fn`*' to specify an input segmentation. Moreover, if a specific label is given via option '`--label=`*`str`*', only segments with the corresponding labels will be considered for change point detection. Segments with a different labels will be left untouched. The following is a particularly useful example of the use of an input segmentation and label

```
ssad --all foo.lin foo.seg
sbic --segmentation=foo.seg --label=speech foo.mfcc foo.seg
```

in which change point detection is carried out after silence detection and only for those segments that contains some audio signal, i.e. labeled as '`speech`'. In the above example, '`foo.mfcc`' is supposed to contain MEL cepstral coefficients corresponding to the waveform in file '`foo.lin`'. Also note that '`foo.seg`' is overwritten by `sbic`. If no input segmentation is given, option '`--time=`*`st`*`:`*`et`*' may be used to specify a portion of the input stream to segment, where *st* and *et* are the start and end time respectively. If *st* is omitted, start time defaults to 0. If *et* is omitted, end time defaults to the end of stream.

Options '`--size-X=`*`size`*' and '`--increment-X=`*`inc`*' specifies the window length and increment, in frames, for each step (`X` = 1,2). In the first step of the three step algorithm, the actual window length is incremented by *size*, as specified by '`--size-1`', if no change is detected. If *size* is set to zero for the second step (option '`--size-2`'), the one-step algorithm is used rather than the three step one and *inc* is the increment between two successive windows. Otherwise, *inc* specifies the increment between two hypothetical change points within a window in the three step algorithm. Option '`--lambda=`*`f`*' sets the weight of the BIC complexity penalty. By default, `sbic` assumes diagonal covariance Gaussian model. If `audioseg` was built with GSL support, option '`--full`' will trigger the use of full covariance matrices.

Note that the three-step algorithm will not work with input from a non-seek-able Unix stream such as `stdin` since this algorithm requires reading the stream several times.

3.3 Clustering

The last possible step of the segmentation process is clustering. The idea is to group together in a same cluster segments with similar audio content. A typical application of clustering is to find out all the segments from a same speaker, for example in order to increase the amount of adaptation data for that particular speaker in a speech recognition system.

3.3.1 Clustering algorithms

Clustering audio segments is typically done using a bottom-up clustering algorithm. In `audioseg`, the clustering algorithm, as the segmentation one, is based on the Bayesian information criterion. The main idea is, given an existing clustering of the input segments, to find out the best pair of clusters to merge and see if the corresponding merge is satisfies the information criterion, i.e. if the BIC difference is positive.

3.3.1.1 Gaussian cluster modeling

Finding out the best cluster pair is done according to a distance measure between two clusters, usually assuming a Gaussian model for the cluster's data. Though not distances in the strict sense, the generalized likelihood ratio or the information criterion difference are also good measures of the similarity between two clusters. As in the previous section, the GLR distance is defined as

$$d_r(C_i, C_j) = \frac{n_i + n_j}{2} \log|\Sigma_{ij}| - \frac{n_i}{2} \log|\Sigma_i| - \frac{n_j}{2} \log|\Sigma_j| \ ,$$

where n_i is the size of C_i and Σ_i the covariance matrix estimated from the cluster data. Similarly, the BIC distance is defined as

$$d_b(C_i, C_j) = d_r(C_i, C_j) - \frac{\lambda}{2} \left(d + \frac{d(d+1)}{2} \right) \log(n_{ij}) \ .$$

Note that the BIC distance defined here is the opposite of the BIC difference as defined in the previous section. Clearly, in every cases, the smaller the distance, the more similar the two clusters.

Given two candidate clusters, a new clustering is obtained by merging the two clusters if this merge results in an increase of the criterion, that is if the BIC difference between the current clustering and the one that would result from the merge is negative. In this case, the BIC difference is given by

$$\Delta BIC = d_r(C_i, C_j) - \frac{\lambda}{2} \left(p + \frac{p(p+1)}{2} \right) \log(n) \ ,$$

where n is the total number of descriptor vectors, i.e. $n = \sum_i n_i$. Thus, if the BIC difference is negative, the two clusters are merged into a single one. Otherwise, the algorithm is stopped. A possible variant is to search for the next best pair of candidate clusters for which the BIC difference is negative. In this case, the algorithm iterates until all pairs of clusters have a positive BIC difference.

3.3.1.2 Gaussian mixture cluster modeling

Alternately, one can model data in a cluster with Gaussian mixture models rather than models. In this case, the Kullback-Leibler divergence is no longer analytically tractable. Computing the generalized likelihood ratio requires a scan over the feature frame, thus implying that all feature vectors are stored in memory. Since release 4.1, `audioseg` experimentally enables GMM-based bottom-up clustering using approximations of the KL divergence and of the BIC criterion that can be computed directly from the parameters of two MAP adapted GMM[2]. This experimental feature is provided by the undocumented command `sgmcluster`. Run with option '`--help`' if you really want to use that.

[2] For details on the idea, see: M. Ben, M. Betser, F. Bimbot and G. Gravier, Speaker Diarization using bottom-up clustering based on a parameter-derived distance between adapted GMMs, In Proc. Intl. Conf. on Speech and Language Processing, 2004.

3.3.2 Clustering with `scluster`

Bottom-up clustering with the Bayesian information criterion is carried out using the command `scluster`.

The command takes as input a stream of descriptor and the corresponding segmentation and output the resulting segmentation after clustering. Output label names are set to '`CX`' where '`X`' correspond to an arbitrary cluster number. Input can be made from `stdout` using a dash ('`-`') either for the input data file or for the input segmentation. If an output name is specified, the resulting segmentation will be written to that file, or to `stdout` if the name is replaced by '`-`'. If no output name is given, the output filename will be derived from the input segmentation one depending on '`--suffix=`*`str`*' which the suffix to add to the input filename for output. If no suffix is specified, the input segmentation file is overwritten. Alternately, clustering can be carried out over a collection of files as given in a script. Since `scluster` operates on segmentations, there must be a segmentation filename specified in each entry in the script. When using scripts, `scluster` outputs a segmentation for each entry. The segmentation filename for each entry is derived from the input segmentation filename as described above. As for `sbic`, a specific label name can be specified using '`--label=`*`name`*'. In this case, only segments with label *name* are considered for clustering and the other segments are left untouched.

By default, two adjacent segments with the same label after clustering, are kept as two distinct segments. Option '`--merge`' changes this behavior and forces adjacent segments with same labels to be merged into a single segment.

The distance between clusters is specified by option '`--distance=`*`str`*', where *str* can be one of '`BIC`' (the default), '`LLR`' or '`KL2`'. To use Gaussian model with full covariance matrices, use option '`--full`'.

By default, clustering stops when the BIC difference for the best pair of clusters is positive. However, option '`--iterate`' may be used to specify the iterative variant described above in which the candidate cluster pair is the best pair with a positive BIC difference. In addition to the BIC difference positiveness, option '`--num-clusters=`*`n`*' may be used to specify a minimum number of clusters. If n is greater than one, clustering will also stop upon reaching a clustering with n clusters. Option '`--max-distance=`*`f`*' may also be used to set a threshold on the distance between two clusters. The idea is that if the distance between two candidate clusters is above the threshold, then the two clusters should not be merged, regardless of the BIC decrease. In this sense, the threshold f corresponds to a maximum distance between two clusters of the same class. When several stop criteria are specified, clustering ends as soon as at least one of the criteria is met.

With option '`--log=`*`fn`*', `scluster` can generate a log file which contains all the necessary information to redo the clustering process. This is particularly usefull to work on stopping criteria as it can avoid the (heavy) cost of recomputing the inter-cluster distances. A typical log file will look like

```
segment bic.1 male 0.000 2.400 C[0] 240 2.9956
segment bic.1 female 2.400 5.000
segment bic.1 female 5.000 6.200
segment bic.1 female 6.200 8.800
segment bic.1 male 8.800 14.800 C[1] 600 4.2087
segment bic.1 male 14.800 19.600 C[2] 480 3.8317
segment bic.1 male 19.600 20.200 C[3] 60 0.9494
segment bic.1 female 20.200 22.800
segment bic.1 female 22.800 23.950
```

```
init 1380 4 5140.31407 1301.37098 3188.25760
merge C[1] C[2] 216.64228 4923.67179 976.02824 3459.62944 -271.37184
merge C[0] C[3] 314.79587 4608.87592 650.68549 3632.84769 -173.21825
merge C[0] C[1] 666.35578 3942.52014 325.34275 3454.50602 178.34166
```

The first lines are segment definitions where segments considered for clustering are used to initialize a cluster. For instance, the first line define a segment in file '`bic.1`' from instant `0.000` to `2.400` with label `male`, from which cluster `C[0]` is initialized with `240` frames and a generalized log-likelihood of `2.9956`. The second line shows a segment not considered for clustering (because of the `female` label). The `init` line provide information regarding the initial partition where each segment is in a separate cluster: the total number of frames (`1380`), the total number of clusters (`4`) followed by the generalized log-likelihood (`5140.31407`), the BIC penalty term (`1301.37098`) and the BIC value obtained by combining the two previous terms (`3188.25760` = (`5140.31407`) - 1.5 (`1301.37098`) in this example). Each `merge` line following the `init` line describes the merge operated at each step of the clustering algorithm. In the line

```
merge C[1] C[2] 216.64228 4923.67179 976.02824 3459.62944 -271.37184
```

clusters `C[1]` and `C[2]`, at a distance of `216.64228`, are merged, the resulting cluster being identified as `C[1]`. After merging, the generalized log-likelihood is `4923.67179`, the BIC penalty `976.0282` and the BIC criterion `3459.62944`. The last value reports the BIC variation with previous step.

The log option is particularly usefull with option '`--disable-bic`' which turns off the BIC stopping criterion. Hence, the command

```
scluster --disable-bic --log=LOG foo.spf foo.seg /dev/null
```

will perform clustering until all segments are grouped, writing each step of the process to `LOG`. Finding out what is the best way to stop the clustering algorithm can then be studied directly from the log file thus saving a lot of useless computation.

4 Gaussian mixture models

Gaussian mixture modeling is probably the most commonly used technique in audio segmentation, indexing and content analysis, in particular because of the ability of GMM to approach any distribution. Gaussian mixture models are widely used for audio classification purposes such as audio event detection, speaker recognition, gender recognition, etc. In conjonction with hidden Markov models, Gaussian mixtures can be used to simultaneously segment and classify the input audio stream. See Chapter 5 [HMM-based segmentation], page 25, for details.

4.1 Parameter estimation

The first step in designing a GMM based application is to estimate the parameters of the required model from a set of training data.

4.1.1 Estimatinon criteria

Two criteria are widely used for parameter estimation, namely maximum likelihood (ML) and maximum a posteriori (MAP). Given a set of training sample $y = \{y_1 \dots y_T\}$, the maximum likelihood criterion aims at finding out the best parameters according to

$$\widehat{\Theta} = arg\max_{\Theta} \sum_{t=1}^{T} \ln p(y_t; \Theta) \ ,$$

where $p(y_t; \Theta)$ is the likelihood of y_t given by

$$p(y_t; \Theta) = \sum_{i=1}^{K} \frac{w_i}{\sqrt{2\pi|\Sigma_i|}} \exp\left(-\frac{1}{2} \left(y_t - \mu_i\right)' \Sigma_i^{-1} \left(y_t - \mu_i\right) \right) \ ,$$

with K the number of components in the mixture model and w_i, μ_i and Σ_i the model parameters Θ. The covariance matrices Σ_i are considered diagonal.

The maximum a posteriori criterion is given by

$$\widehat{\Theta} = arg\max_{\Theta} \ \ln p(\Theta) + \sum_{t=1}^{T} \ln p(y_t; \Theta) \ ,$$

where $p(\Theta)$ is the prior distribution of the parameters.

4.1.2 Initialization

For both criteria, the estimation is carried out using an iterative Expectation-Maximization (EM) algorithm which requires a good initial solution. Initializing a model can be done in one of two ways.

The most simple solution is to initialize the mean vectors according to the extrema, independently in each dimension. Variance vectors can then be set either to an arbitrary (non null) value or to the global variance of the training samples. When using the global variance, a variance floor can be set to avoid overtraining problems on small training data sets. Finally, weights are set to a $1/K$. However, this approach often does not provide a

good enough approximation of the ML or MAP estimate, in particular when large training data sets are used.

A much better, but much slower solution consists in quantizing the training samples into K clusters. The GMM mean vectors are then initialized with the clusters mean vectors. Variance vectors and component weights are initialized as before. Though initializing the variances and weights from the clusters is possible, this technique is not very robust in practice. Clustering is done using a split VQ algorithm. The split VQ algorithm consists in increasing progressively the codebook size by dividing each cluster into two and by optimizing the new codebook with a k-means algorithm. Splitting a cluster is done by perturbing the cluster mean vector by a fraction of the variance in two opposite directions.

Model initialization is performed with `sgminit` which takes as input a collection of training data files, and outputs a model. The input data files can be specified on the command line or via a script. In the last case, a segmentation may also be specified for each data file in the script thus enabling to estimate parameters solely using segments with a particular name specified using '`--label=str`'. If no label is specified, all segments are considered for training. If no segmentation is specified, the entire data file is considered. The number of components in the output model is specified using '`--num-comp=n`'.

By default, means are initialized from the data extrema and variance are set to the floor specified using option '`--variance-floor=f`'. Alternately, variance can be initialized from the global variance of the data using option '`--global`'. In this case, global variance are floored according to the variance floor *f*. Note that the default variance floor of 0.01 is meant for global variance flooring and should be set to a higher value, typically one, to initialize variances to the floor value. When using global variance initialization, the global variance vector can be saved to file *fn* in text format with '`--variance=fn`'.

Mean initialization via clustering is turned on using option '`--vq`', in which case the specified number of components must be a power of 2. Options '`--max-iter=n`' and '`--epsilon=f`' are used to control the convergence of the split VQ algorithm by specifying, respectively, the maximum number iterations and the minimum relative decrease in the total distance in the k-means optimization. By default, the k-mean algorithm uses a euclidian distance but the Mahalanobis distance can be used instead ('`--mahalanobis`'). The tree resulting from the hierarchical VQ can be saved to file with option '`--save-tree=fn`'. The format is the following

```
nnodes nleaves dim
node_id parent_id left_id right_id npoints
m[1] m[2]  ........................  m[dim]
v[1] v[2]  ........................  v[dim]
...
```

Left and right child IDs are set to '`-1`' for leaves. So is the parent ID of the root node. Field '`npoints`' is the number of points in the cluster while '`m`' and '`v`' are the cluster's mean and variance respectively.

4.1.3 Estimation

Maximum likelihood parameter estimation is carried out using the EM algorithm. Given an estimate of the model parameters w_i, m_i and v_i, and a set of training samples $y = \{y_1 \dots y_T\}$,

new estimates are obtained according to

$$\widehat{w}_i = \frac{\sum_t \gamma_i(t)}{T}$$

for the weights, to

$$\widehat{m}_i = \frac{\sum_t \gamma_i(t)\ y_t}{\sum_t \gamma_i(t)}$$

for the mean vectors and to

$$\widehat{v}_i = \frac{\sum_t \gamma_i(t)\ y_t^2}{\sum_t \gamma_i(t)} - \widehat{m}_i^2$$

for the variance vectors[1]. In the above equations, $\gamma_i(t)$ denotes the probability that sample y_t originates from component i, known as the occupation probabilities and given by

$$\gamma_i(t) = \frac{w_i\ N(y_t; m_i, v_i)}{\sum_j w_j\ N(y_t; m_j, v_j)}$$

with $N(y_t; \mu, \sigma^2)$ the Gaussin likelihood of y_t assuming mean vector μ and variance vector σ^2. The reestimation process carries on with the new estimates until convergence is reached. It can be shown that each iteration increases the likelihood of the training data. In practice, variance may be floored at each reestimation step to avoid overtraining.

Maximum a posteriori estimation also follow an iterative EM algorithm. Assuming prior parameter values $\Theta^{(p)} = \{w_i^{(p)}, m_i^{(p)}, v_i^{(p)}\}$, the reestimation formulae are given by

$$\widehat{w}_i = (1-\alpha_i)\ \frac{\sum_t \gamma_i(t)}{T} + \alpha_i w_i^{(p)}\ ,$$

$$\widehat{m}_i = (1-\alpha_i)\ \frac{\sum_t \gamma_i(t)\ y_t}{\sum_t \gamma_i(t)} + \alpha_i m_i^{(p)}$$

and

$$\widehat{v}_i = (1-\alpha_i)\ \frac{\sum_t \gamma_i(t)\ y_t^2}{\sum_t \gamma_i(t)} + \alpha_i \left(v_i^{(p)} + {m_i^{(p)}}^2 \right) - \widehat{m}_i^2\ .$$

In the above equations, α_i controls the relative weight of the prior and posterior estimates and is defined as

$$\alpha_i = 1 - \frac{\sum_t \gamma_i(t)}{r + \sum_t \gamma_i(t)}$$

where r is a user defined constant known as the prior weight. Clearly, if $r = 0$, the weight of the prior distribution is null and the MAP estimate simplifies to the ML one. Note that the actual relative weight between the prior and posterior estimates depends on the total component occupation count.

The command `sgmestim` can be used to compute both ML and MAP estimates. The command reads an existing model (e.g. initialized with `sgminit`) and a collection of training data files, either as command line arguments or via a script. As for `sgminit`, segmentation files can be specified for each script entry and segments with a particular label can be

[1] Note the notation shortcut where y_t^2 denotes vector y_t with all the elements squared. The same holds for $\widehat{m}_i^2$

selected via the option '`--label=str`'. The reestimated model is overwrites the input file unless '`--output=fn`' is specified, in which case the output model is written to file *fn*.

MAP estimation is available via '`--map=f`' where *f* is the prior weight. By default, *f* is null and ML estimation is performed. Options '`--max-iter=n`' and '`--epsilon=f`' are used to control the convergence of the EM algorithm.

It is often practicle not to reestimate all the model parameters, specially when doing model adaptation from small amount of training data. In this case, not reestimating the variance is a common way to avoid overfitting the training data. The parameters to update can be specified by the option '`--update=str`' where *str* is a string made of one or several letters among '`wmv`' (respectively for weights, means and variances) specifying which parameters are to be updated.

Floors can be set for weights and variance. Weight floors ('`--weight-floor`') are used to avoid components from dying. Indeed, a component whose weight falls below 1e-5 is considered as dead and will not be used for likelihood computation. Use option '`--weight-floor=f`' to set the weight floor to *f*. Variance flooring can either be done according to a constant floor set using option '`--variance-floor=f`' or according to the global variance. If a variance file is specified with '`--variance=fn`' in addition to '`--variance-floor=f`', the variance floor will be set to a fraction *f* of the global variance for each dimension. The variance file is a text file containing the variance for each dimension as output by the '`--variance=fn`' option of `sgminit`.

If training data are spread accross many files and segments, sequentially reading all segments at each iteration may seriously slow down training. To avoid this problem, the command `spfcat` can be used to concatenate all training frames into a single file that will be given as argument to `sgmestim`.

4.2 Likelihood computation

The command `sgmlike` aims at computing the log-likelihood of segments or entire data files given one or several models. In order to be as versatile as possible, the command supports several possible invocation forms.

In its mots simple form, the command inputs a model and one or several data files, in order to compute the log-likelihood of each input data file with the specified model. The output of `sgmlike` is a text file containing the likelihoods for each input data files. Output is made to `stdout` unless otherwise specified by option '`--output=fn`'. Input data files can be specified as command line arguments or using a script. Alternately, several models can be specified to compute at once the likelihood of each data file according to each input model. Multiple models are specified using option '`--list=fn`', where *fn* is a file containing a list of path to GMM files.

The generic output likelihood file format follows the syntax

```
filename start_time end_time nframes label nmodels llk llk ...
```

where '`filename`' is the path of the input filename, '`start_time`' and '`end_time`' are the start and end times of the scored segment, '`nframes`' is the number of frames, '`label`' the scored segment label or '`-`' if no label and '`nmodels`' is the number of models the segment was scored against. The remaining fields are the segment log-likelihoods with each model in the same order as in the model list. Fields are separated by one or several spaces. For

example, the command `sgmlike -f models.lst foo.lfcc`, where '`models.lst`' is a file containing the path of two models would produce the following output

```
foo.lfcc 0.00 -1.00 2395 - 2 -7945.8212     -9826.8346
```

In this example, '`end_time`' is set to -1 to indicate that no end time was specified.

A second invocation form consists in specifying a segmentation for each input data file. In this case, each segment in the specified segmentations is scored. Segmentation-based scoring is specified with option '`--segmentation`' or '`--segmentation=`*fn*'. If input files are read from a script, using '`--segmentation`' will cause `sgmlike` to score the input files according to the corresponding segmentation as specified in the script. Note that in this case, the script must contain segmentation files. Alternately, when a single data file is specified via the command line, the corresponding segmentation can be specified by the *fn* argument. It is not possible to use segmentations with several input files on the command line and a script must be used in this case. In addition, if a label is specified using '`--label=`*str*', only segments with label *str* are scored. To illustrate this, using options `--segmentation=foo.seg` and `--label=spk1` in the above example, would produce an output similar to

```
foo.lfcc 0.26 2.41 216 spk1 2 -1306.7011 -2764.1254
foo.lfcc 9.20 12.81 360 spk1 2 -2153.7889 -3487.7223
foo.lfcc 16.55 19.64 308 spk1 2 -1551.6085 -2984.2018
foo.lfcc 20.03 22.62 260 spk1 2 -1531.5742 -2951.9374
```

where each line in the output corresponds to a segment in '`foo.seg`' with label '`spk1`'.

The third way to use `sgmlike` is with fixed length segments. This mode is intended for model-based segmentation or event detection and tracking without prior segmentation. In this case, scores are computed for the specified models on fixed-length segments. Using option '`--length=`*n*' will result in scoring segments containing *n* frames. The shift between two consecutive segments is set using '`--shift=`*m*' thus enabling overlapping segments. However, if not specified, non overlapping segments are used and *m* defaults to the specified segment length *n*. Note that the last segment may contain less than *n* frames.

Usually, a segment score correspond to the sum of the frame log-likelihoods accross the segment. However, when comparing scores from segments with different lengths, it is best to normalize scores by the corresponding segment length. Though this can easily be done from the output of `sgmlike` when comparing the scores, option '`--normalize`' is provided for this. When using several models, log-likelihoods can also be replaced by posterior probabilities using '`--posterior`'. The posterior probability for a given model is defined as the likelihood for that model divided by the sum of the likelihoods for all the models.

Finally, when using several models with a large number of Gaussians, scoring can be slow and, in certain particular cases, using only the best scoring Gaussians in the mixture can speed up significantly things. In particular, this is a common practice speaker verification tasks where a segment is tested against several speaker models, all adapted from the same generic model (a.k.a. background model in the speaker verification literature). In this case, the best mixture component are determined on the background model and only those components are used when scoring with the speaker models, thus saving a significant amount of computations. This is the purpose of option '`--n-best=`*n*' in `sgmlike`. When using this option, the *n* best components are selected from the first model in the model list and used in the subsequent score computations. Typical values for *n* are between 5 and 10. Note

that using '`--n-best=n`' with a single model may also save a little time on some machines[2] but the gain is much less spectacular that when using several models.

4.3 Classification with GMMs

Classification is probably the most common use of Gaussian mixture models. Though `audioseg` does not provide any specific command for classification, all the necessary elements are given to easily implement such tools. The main reason for not developping any specific classification tool is that, in most cases, it is more convenient to write such tools with scripts using the basic commands provided by `audioseg`. Indeed, though classification with a maximum a posteriori or a maximum likelihood criterion is a quite generic technique, there are many task specificities (e.g. score normalization, open/close set classification, etc) which makes the job of writing a generic classification program a hard one.

Assuming models have been estimated for each class that is to be detected, for example with `sgminit` and `sgmestim`, `sgmlike` can be used to compute the likelihoods of the input data with each class model. A classification script would then just scan the output of `sgmlike` to select the class with the best likelihood or with the best posterior probability assuming prior probabilities for each class have been defined.

4.4 Drawing samples

The command `sgmdraw` can be used to draw samples from a Gaussian mixture model. Samples are drawn from the specified model and written on disk in SPro format. Option '`--num-samples=n`' sets the number of data samples to generate to n. n defauls to 10000 samples. The sample drawing command was mainly written for debug purposes. However, it has since been used for score normalization based on the Kullback-Liebler distance between models, where the KL distance is approximated using a Monte-Carlo method, the latter requiring samples drawn from a model.

4.5 Import/Export

In `audioseg`, models are saved to disk in binary format. Viewing the model parameters for diagnostic purposes or importing models from an alien format is therefore not easy unless you know exactly what the binary format is. Fortunately, the command `sgmcopy` can import models from or export models to a much mor readable text format. The GMM text format is defined as follow

```
ncomp dim
id w
m[id][0]            ...     m[id][dim-1]
v[id][0]            ...     v[id][dim-1]
```

where components IDs '`id`' range from 0 to `ncomp`-1, '`w`' is the component weight while '`m`' and '`v`' are the mean and variance vectors respectively.

Using option '`--import`' in `sgmcopy` will cause the command to read models in text format. Similarly, '`--export`' will cause the output of scopy to be in text format rather

[2] With a single model, n-best scoring saves $K - n$ exponentiations but adds a sort of the K component scores.

than in binary format. By default, output goes to `stdout`. Option '`--output=`*fn*' will cause the output to go to file *fn*.

`sgmcopy` can also be used to combine several models into a single one. Combining models consists in pooling together all the components of the input models into a single model, the component weights being renormalized to sum to one. In other words, the resulting model is the sum of the input models with equal prior. Option '`--combine`' will cause `sgmcopy` to combine the input models specified on the command line.

4.6 Gaussian modeling

Yes, Gaussian models are a particular case of GMM with a single component! However, the Gaussian model is often used with full covariance matrices while diagonal covariance matrices are usually considered for mixture models. While implementing a generic mixture model, possibly with full covariance matrices, would indeed have been possible, it would have been a waste of time. Hence, the two commands `sgestim` and `sglike` are provided specifically for Gaussian modeling. Their syntax and behavior is the same as for `sgmestim` and `sgmlike` respectively.

The `sgestim` command will estimate the mean and covariance matrix/vector from a set of training samples using maximum likelihood estimation. Note that, if several segments are provided, the parameters are computed globally accross all segments. As can be expected, `sglike` will compute the likelihood of some segments given Gaussian models. But `sglike` also implements model comparison by first computing a Gaussian model for a segment before computing a distance between the segment's model and the reference model. Two distances are provided, namely (symetric) Kullback-Leibler and Arithmetic-Harmonic Sphericity[3]. The option '`--distance=`*str*' option will trigger the use of model distances, where *str* is either '`kl2`' or '`ash`'.

[3] See F. Bimbot, I. Magrin-Chagnolleau and L. Mathan. Second-Order Statistical Measures for Text-Independent Speaker Identification. Speech Communication, 17:51-54, 1995.

5 HMM-based segmentation

5.1 Viterbi segmentation

A highly popular model for audio analysis is the hidden Markov model (HMM) which enables to jointly segment and classify an input stream of descriptors. In its simplest form, a HMM can be seen as a weighted directed graph where each node, known as a state, correspond to one audio class. The distribution of feature vectors for a given audio class is classically modeled using a Gaussian mixture model known as the state conditional density. The model is therefore defined by the number of states, the GMM associated to each state and a transition matrix (or adjacency matrix in the graph terminology) defining the probability of transiting from one state to another (or, in other words, the weights along the vertices of the graph).

Given such a model and a sequence of acoustic feature vectors, the Viterbi algorithm can find out the most likely alignment between the feature vectors and the states of the model. The alignment equation is given by

$$\widehat{s}_a^b = arg\max_s \sum_{t=a}^{b} \ln p(y_t|s_t) + \ln p(s_{t-1}|s_t) \ ,$$

where $p(s_{t-1}|s_t)$ denotes the transition probability from s_{t-1} to s_t and $p(y_t|s_t)$ the likelihood of feature vector y_t given the state conditional density for state s_t. Hence, as a result, all frames are mapped to one state of the HMM thus providing a class label for each frame. The result is therefore a segmentation, obtained by grouping in a single segment all consecutive frames mapped to the same state, and a class label for each segment.

The main limitation of HMMs for audio class segmentation is their natural tendency to change state every frame. The reason for this behavior is that the score is dominated by state conditional likelihoods while transition probabilities accounts for next to nothing. A few tricks are therefore necessary. The most classical one is the use of a transition penalty added to every transition from one state to a different one. A less classical (yet equivalent) way to balance state conditional likelihoods and transition probabilities is to scale transition probabilities. Both techniques can be used at the same time. Therefore, in practice, the alignment equation writes as

$$\widehat{s}_a^b = arg\max_s \sum_{t=a}^{b} \ln p(y_t|s_t) + \alpha \ln p(s_{t-1}|s_t) - \pi(s_{t-1}, s_t) \ ,$$

where α is the scaling factor and $\pi(s_{t-1}, s_t) = p$, the transition penalty, if $s_{t-1} \neq s_t$ and 0 otherwise.

In theory, state conditional distributions are somewhat arbitrary and therefore there is no (theoretical) reason to have the same number of Gaussian components in each state. However, if you do so, states with a high number of components are more likely to appear in the result as, for such states, the state conditional likelihood is usually better than for states with less components (the more parameters, the better). In order to use different number of components per state, it is therefore better to use an information criterion rather than

the mere likelihood. In `audioseg`, it is possible to replace the log-likelihood $\ln p(s_{t-1}|s_t)$ by the Akaike information criterion defined as

$$\ln p(s_{t-1}|s_t) - 0.5\alpha(n(2d+1)-1)$$

where n is the number of components and d the feature vector dimension.

5.2 Using `sviterbi`

The `audioseg` toolkit provides very basic HMM decoding capabilities with the `sviterbi` command. In particular, no HMM training with the EM algorithm is provided as HMMs for broad audio segmentation and classification does not require precise training. In a typical scenario, one should first gather training segments for each of the audio classes to be detected. From those training segments, Gaussian mixture models, one for each class, can be estimated using `sgminit` and `sgmestim`. Such models are used as state conditional densities to define a HMM whose transition probabilities are estimated from empirical measures. Finally, `sviterbi` will provide the segmentation given the model and an input feature stream.

Hidden Markov models are defined as text files as in the following example which illustrate a typical HMM to segment speech from two different speakers.

```
2
spk1 dat/spk1.gmm
spk2 dat/spk2.gmm
0.5 0.5
0.2 0.8
0.8 0.2
0.5 0.5
```

where a model with `2` states is defined. The first (resp. second) state bears label `spk1` (resp. `spk2`) and the model in file '`dat/spk1.gmm`' (resp. '`dat/spk2.gmm`') defines the state conditional density for state 1 (resp. 2). The subsequent lines define the transition matrix where the first and last lines correspond to, respectively, the initial (input) and final (output) probabilities.

The command `sviterbi` takes as input a model and a feature stream and output the segmentation. Transition penalty and scaling are accessible using options '`--scale=s`' and '`--penatly=s`' respectively. Option '`--time`' can be used to specify that decoding is to be performed only on a fraction of the stream.

In all cases, the output is a segmentation where the score associated to a segment correspond to the log-likelihood along the alignment path for the model. The use of option '`--normalize`' will cause `sviterbi` to normalize the log-likelihood by the segment length.

Finally, keep in mind that `sviterbi` is a very basic implementation of HMM where no pruning is performed. As a result, decoding with large number of states tends to be slow.

6 Quick reference guide

This chapter is meant as a reference guide for all the audioseg tools, summarizing the syntax, synopsis and options. This is actually a printed version of the online help message obtained with '`--help`'.

6.1 ssad

Usage

`ssad [options] ifile ofile`

Synopsis

Silence and/or audio activity detection. I/O via `stdin` and `stdout` using '`-`'.

Options

`-F, --format=str`
: Specify the input waveform file format. Available formats are '`PCM16`', '`ALAW`', '`ULAW`', '`wave`' or '`sphere`'. Default: '`PCM16`'.

`-f, --sample-rate=f`
: Set input waveform sample rate to *f* Hz for '`PCM16`', '`ALAW`' and '`ULAW`' waveform files. Default: 8 kHz.

`-x, --channel=n`
: Set the channel to consider for feature extraction. Default: 1.

`-B, --swap`
: Swap the input waveform samples.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-s, --log` Use log-energy rather than energy.

`-l --length=f`
: Set the analysis frame length to *f* ms. Default: 20.0 ms.

`-d, --shift=f`
: Set the interval between two consecutive frames to *f* ms. Default: 10.0 ms.

`-w, --window=str`
: Specify the waveform weighting window. Available windows are '`Hamming`', '`Hanning`', '`Blackman`' or '`none`'. Default: '`Hamming`'.

`-t, --threshold=f`
: Set signal (log-)energy threshold to m1 - *f* * s1, where m1 and s1 are the mean an variance of the high energy Gaussian. Default: maximum likelihood criterion.

`-m, --min-length=len`
: Set minimum silence segment duration in seconds. Default: 0.

`-r, --reverse`
: Output audio activity segments. Default: silence segments.

`-a, --all`
: Output both silence and audio activity segments. Default: silence segments.

`-v, --verbose[=n]`
: Set trace level to n. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

2 report segmentation progress

3 report information on bi-Gaussian mapping.

6.2 spfnorm

Usage

`spfnorm [options] ifile ofile`

Synopsis

Feature normalization and frame selection. I/O via `stdin` and `stdout` using '`-`'.

Options

`-n, --norm-buff=n`
: Set size of normalization buffer to *n* frames. Default: entire file

`-s, --segmentation=fn`
: Consider only those frames specified by the segmentation in *fn*.

`-x, --label=s`
: Consider only segments with label *s*. Default: all segments

`-t, --time=[sn][:en]`
: Extract and normalize from frame *sn* (defaults to 1) to frame *en* (defaults to end of file).

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-v, --verbose`
: Turn on verbose mode. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

6.3 sbic

Usage

`sbic [options] ifile ofile`

Synopsis

Abrupt change detection. I/O via `stdin` and `stdout` using '`-`'. Reading descriptor streams from `stdin` is not possible in the three-pass algorithm.

Options

`-b, --size-1=n`
: Set window size for the first pass to *n*. Default: 300.

`-a, --increment-1=n`
: Set whithin window shift for the first pass to *n*. Default: 60.

`-d, --size-2=n`
: Set window size for the second step. If *n* is null, use the single pass algorithm. Default: 200.

`-c, --increment-2=n`
: Set whithin window shift for the second pass. Default: 20.

`-l, --lambda=f`
: Set the complexity penalty weight in BIC or, the change detection threshold if the one-pass algorithm us being used ('`--size-2=0`'). Default: 1.5

`-s, --segmentation=fn`
: Apply change detection independently for each segment in file *fn*. Default: segment entire input stream.

`-x, --label=str`
: In conjonction with '`--segmentation=`*fn*', apply change detection independently for each segment in file *fn* whose label name is *str*. Other segments are left unchanged. Default: process all segments in *fn*.

`-t, --time=st:et`
: Specifiy start and end times in seconds between which change detection is to be carried out. If *st* is omitted, start time defaults to 0. If *et* is omitted, end time defaults to the end of stream. Default: process the entire input stream.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'. Default: no convertion.

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

2 report change detection

6.4 scluster

Usage

```
scluster [options] data seg [oseg]
   scluster [options] --file-list=fn
```

Synopsis

Bottom-up segment clustering. I/O via `stdin` and `stdout` using '`-`' possible only in the first form.

Options

`-t, --distance=s`
: Define distance between two clusters as *s*. Possible values are '`KL2`', '`GLR`' (default) and '`BIC`'.

`-l, --lambda=f`
: Set the complexity penalty weight in BIC. Default: 1.0

`-f, --full`
: Assume full covariance matrices.

`-d, --max-distance=f`
: Stop clustering if the distance between the two clusters to merge is more than *f*.

`-n, --num-clusters=n`
: Set minimum number of clusters to *n*. Default: 1

`-i, --iterate`
: Turn on iterative mode. Default: no.

`-z, --disable-bic`
: Do not check BIC stopping criterion. Default: use BIC stopping criterion

`-L, --file-list=fn`
: Set input script to *fn*.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'. Default: no convertion.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-x, --label=str`
: In conjonction with '`--segmentation=fn`', apply change detection independently for each segment in file *fn* whose label name is *str*. Other segments are left unchanged. Default: process all segments in *fn*.

`-b, --suffix=str`
: Set output segmentation filename suffix to *str* if no output filename is specified or if a script is used. Default is to overwrite the input segmentation file.

`-m, --merge`
: Merge adjacent segments in same cluster into a single segment: Default: keep such segments distincts.

`-g, --log=fn`
: log clustering operations to file *fn*. Default: no log

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

2 report convergence information

3 report merges

6.5 sgestim

Usage

`sgestim [options] model data ...`

`sgestim [options] --file-list=script model`

Synopsis

Gaussian model maximum likelihood parameter estimation. The input script file contains a list of all the training data files, optionally with the corresponding segmentation files (required if '`--label`' is specified).

Options

`-c, --full`
: Use full covariance matrix.

`-f, --variance-floor=f`
: Floor variance below *f*. Default: 1e-4

`-L, --file-list=fn`
: Use training data in script *fn* rather than using the command line arguments.

`-x, --label=str`
: Initialize parameters using only segments with label *str*. Option '`--file-list=fn`' must be set and the script in *fn* must contain segmentation files for each entry.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

6.6 sglike

Usage

```
sglike [options] model data ...
   sglike [options] --list=fn data ...
   sglike [options] --file-list=fn model
   sglike [options] --list=fn --file-list=fn
```

Synopsis

Score data files with the specified model(s). If a segmentation is given, sgmlike scores each segment in the specified segmentation file(s). The segmentation is taken from the file list if it is specified, otherwise the program will load the segmentation from the file specified as argument to '`--segmentation`'. Fixed length segmentation can be specified with the '`--length`' and '`--shift`' options.

Output a likelihood file where each line corresponds to a scored segment and has the following syntax:

```
filename start_time end_time nframes label nmodels llk llk ...
```

where '`filename`' is the path of the input filename, '`start_time`' and '`end_time`' are the start and end times of the scored segment, '`nframes`' is the number of frames, '`label`' the scored segment label or '`-`' if no label and '`nmodels`' is the number of models the segment was scored against. The remaining fields are the segment log-likelihoods with each model in the same order as in the model list. Fields are separated by one or several spaces.

Options

`-k, --use-distance[=type]`
: Compute distance between models rather than data log-likelihood, using specified distance. Valid distances are '`ASH`' and '`KL2`'.

`-s, --segmentation[=fn]`
: Score according to the segmentation. If a script is used, segmentation files are read from the script. Otherwise, the command line must have a single argument and *fn* specifies the corresponding segmentation file. Default: score entire file.

`-x, --label=str`
: Score only segments with label *str*. Requires '`--segmentation`'.

`-l, --length=n`
: Score using fixed-length segments of length *n* seconds. Fixed-length segmentation is overwritten by option '`--segmentation`'. Default: score entire file.

`-d, --shift=n`
: Shift, in seconds, between two consecutive segments when using fixed-length segments. Default shift is equal to the specified segment length.

`-t, --time=[st][:et]`
: Score input file(s) starting at *st* seconds and ending at *et*. If omitted, *st* defaults to 0 and *et* to the end of the input data file. If a segmentation is specified, the time boundaries specified by '`--time`' are ignored.

`-L, --file-list=fn`
: Score all files in script *fn*.

`-f, --list=fn`
: Score segments against all the models listed in *fn*.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-o, --output=fn`
: Write output to file *fn*. Default: `stdout`

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

6.7 sgminit

Usage

```
sgminit [options] model data ...
   sgminit [options] --file-list=script model
```

Synopsis

Gaussian mixture model parameter initialization. If specified, the input script contains a list of all the training data files, optionally with the corresponding segmentation files. Output file *model* is created.

Options

`-n, --num-comp=n`
: Set the number of components. Default: 64

`-g, --global`
: Initialize variance vectors to the global variance on the training data. Default: constant variance.

`-f, --variance-floor=f`
: If '`--global`' is set, floor variances at *f*. If option '`--global`' was not specified, initialize all variances to *f*. Default: 1e-4

`-c, --variance=fn`
: Save global variance vector to file *fn*.

`-q, --quantize`
: Initialize mean vectors using split VQ. Default: flat initialization.

`-i, --max-iter=n`
: Set the maximum number of k-means iterations between two splits in the split VQ algorithm to *n*. Default: 10

`-e, --epsilon=f`
: Set the convergence factor in k-means clustering to *f*. Stop k-means iterations if the relative distance decrease between two iterations is less than *f*. Default: 0

`-m, --mahalanobis`
: Use the Mahalanobis distance in k-means clustering. Default: eucilidian distance.

`-o, --save-tree=fn`
: Save the binary quantization tree to file *fn*.

`-L, --file-list=fn`
: Use training data in script *fn* rather than using the command line arguments.

`-x, --label=str`
: Initialize parameters using only segments with label *str*. Option '`--file-list=fn`' must be set and the script in *fn* must contain segmentation files for each entry.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

2 report basic tree building

3 report quantization convergence

6.8 sgmestim

Usage

```
sgmestim [options] model data ...
   sgmestim [options] --file-list=script model
```

Synopsis

Estimate parameters of the model in *model* using the either the ML or MAP criterion. The input script file contains a list of all the training data files, optionally with the corresponding segmentation files

Options

`-m, --map=f`
: Use MAP restimation with a prior weight *f*. By default, *f*=0 and ML restimation is performed.

`-u, --update=str`
: Specify wich parameters to update. *str* is a string made of the letters '`wmv`' and specifying the parameters to update ('`w`' = weight, '`m`' = mean, '`v`' = variance). Default: update all parameters (`str=wmv`).

`-i, --max-iter=n`
: Set the maximum number EM iterations to *n*. Default: 10

`-e, --epsilon=f`
: Set the convergence factor to *f*. Stop iterations when the relative likelihood increase falls below *f*. Default: 0

`-w, --weight-floor=f`
: Floor component weights at *f*. If no floor is used, components with weight below 1e-5 will be considered as inactive. Default: 0

`-f, --variance-floor=f`
: Floor variances below *f* after each parameter restimation. If a global variance file is specified, the floor is a vector set to a the fraction *f* of the global variance vector. Default: 1e-4

`-c, --variance=fn`
: Specify a global variance vector in *fn* for variance flooring (see '`--variance-floor`').

`-L, --file-list=fn`
: Use training data in script *fn* rather than using the command line arguments.

`-x, --label=str`
: Initialize parameters using only segments with label *str*. Option '`--file-list=fn`' must be set and the script in *fn* must contain segmentation files for each entry.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

2 report basic EM iterations

3 report EM accumulation

6.9 sgmlike

Usage

```
sgmlike [options] model data ...
   sgmlike [options] --list=fn data ...
   sgmlike [options] --file-list=fn model
   sgmlike [options] --list=fn --file-list=fn
```

Synopsis

Score data files with the specified model(s). If a segmentation is given, sgmlike scores each segment in the specified segmentation file(s). The segmentation is taken from the file list if it is specified, otherwise the program will load the segmentation from the file specified as argument to '`--segmentation`'. Fixed length segmentation can be specified with the '`--length`' and '`--shift`' options.

Output a likelihood file where each line corresponds to a scored segment and has the following syntax:

```
filename start_time end_time nframes label nmodels llk llk ...
```

where '`filename`' is the path of the input filename, '`start_time`' and '`end_time`' are the start and end times of the scored segment, '`nframes`' is the number of frames, '`label`' the scored segment label or '`-`' if no label and '`nmodels`' is the number of models the segment was scored against. The remaining fields are the segment log-likelihoods with each model in the same order as in the model list. Fields are separated by one or several spaces.

Options

`-s, --segmentation[=fn]`
: Score according to the segmentation. If a script is used, segmentation files are read from the script. Otherwise, the command line must have a single argument and *fn* specifies the corresponding segmentation file. Default: score entire file.

`-x, --label=str`
: Score only segments with label *str*. Requires '`--segmentation`'.

`-l, --length=n`
: Score using fixed-length segments of length *n* seconds. Fixed-length segmentation is overwritten by option '`--segmentation`'. Default: score entire file.

`-d, --shift=n`
: Shift, in seconds, between two consecutive segments when using fixed-length segments. Default shift is equal to the specified segment length.

`-t, --time=[st][:et]`
: Score input file(s) starting at *st* seconds and ending at *et*. If omitted, *st* defaults to 0 and *et* to the end of the input data file. If a segmentation is specified, the time boundaries specified by '`--time`' are ignored.

`-a, --add`
: Add scores accross segments for each model(s), thus producing a single line output. This option is usefull in conjonction with '`--label`' to get the cumulated score accross segments with a specified name.

`-b, --n-best=n`
: Score using the *n* best components where the best components are determined solely on the first model of the input model list. The best components are then used for scoring with the subsequent model, thus saving some computation load. Note that using this only makes sense if all the models are derived from the first one, for example by adaptation.

`-n, --normalize`
: Normalize log-likelihoods by the segment length.

`-p, --posterior`
: Output posterior probabilities rather than log-likelihoods.

`-L, --file-list=fn`
: Score all files in script *fn*.

`-f, --list=fn`
: Score segments against all the models listed in *fn*.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'.

`-o, --output=fn`
: Write output to file *fn*. Default: `stdout`

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

6.10 sgmcopy

Usage

`sgmcopy [options] model ...`

Synopsis

Copy models with possible format convertion. Alternately, combine models with equal prior.

Options

`-i, --import`
: Import model(s) from text format. When using '`--import`', assume that all the input models are in text format.

`-x, --export`
: Save output model in text format.

`-c, --combine`
: Combine input models into a single one with equal priors. The resulting model is a GMM whose components are all the components of the input models. Component weights are renormalized accordingly.

`-o, --output=fn`
: Output model to file *fn*. If not specified, output is directed to `stdout` (even if binary).

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

6.11 `sgmdraw`

Usage

```
sgmdraw [options] model ofile
```

Synopsis

Draw samples according to *model*. The samples are stored in *ofile* in SPro format.

Options

`-n, --num-samples=n`
: Set number of samples to draw to *n*. Default is 10000.

`-S, --buffer-size=n`
: Set input buffer size in kbytes. Default size is 10 Mbytes.

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

6.12 spfcat

Usage

```
spfcat [options] ofile data ...
   spfcat [options] --file-list=script ofile
```

Synopsis

Concatenate frames all (selected) frames into a single file.

Options

`-L, --file-list=fn`
: Use training data in script *fn* rather than using the command line arguments.

`-x, --label=str`
: Initialize parameters using only segments with label *str*. Option '`--file-list=fn`' must be set and the script in *fn* must contain segmentation files for each entry.

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'.

`-S, --buffer-size=n`
: Set the input buffer size to *n* kbytes. Default: 10 Mb.

`-v, --verbose[=n]`
: Set trace level to *n*. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

6.13 `sviterbi`

Usage

`sviterbi [options] ifile model ofile`

Synopsis

HMM decoding of *ifile* using hidden Markov model in file *model*.

Options

`-a, --akaike`
: Use Akaike information criterion instead of state conditional likelihoods. This option is useful when different number of Gaussian components are used in each state.

`-p, --penalty=f`
: Set transition penalty to *f*. Default: 0

`-s, --scale=f`
: Scale transition probabilities by a factor *f*. Default: 1

`-n, --normalize`
: Normalize output scores by segment length.

`-t, --time=[et][:st]`
: Perform HMM segmentation between instants *st* and *et*. When not specified, *st* defaults to 1 and *et* to the end of the feature stream. Default: entire file

`-C, --convert=str`
: Set convertion flag for descriptors where *str* is a string containing SPro stream description flags among '`ZRDAN`'. Default: no convertion.

`-S, --buffer-size=n`
: Set input buffer size in kbytes. Default size is 10 Mbytes.

`-v, --verbose[=n]`
: Set trace level to n. Without arguments, trace level is set to $n = 1$. Default: no trace.

`-h, --help`
: Print help message and exit.

`-V, --version`
: Print version information and exit.

Trace levels

0 no trace

1 report file I/O

2 report alignment details

Appendix A GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgments" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant

Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this

License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being list their titles, with the
Front-Cover Texts being list, and with the Back-Cover Texts being list.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being *list*"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index