Modular Audio Recognition Framework v.0.3.0-devel-20050817 and its Applications

The MARF Research and Development Group

Montréal, Québec, Canada

Thu Aug 18 12:19:27 EDT 2005

Contents

1 Introduction **3**

1.1 What is MARF? 3

1.1.1 Purpose 3

1.1.2 Why Java? 3

1.1.3 Terminology and Notation 4

1.2 Authors, Contact, and Copyright Information 4

1.2.1 COPYRIGHT 4

1.2.2 Authors 4

1.3 A Short History of MARF 5

1.3.1 Developers Emeritus 5

1.3.2 Current Status and Past Releases 5

1.4 MARF Source Code 6

1.4.1 Project Source and Location 6

1.4.2 Formatting 6

1.4.3 Coding and Naming Conventions 7

2 MARF Architecture **8**

2.1 Application Point of View 8

2.2 Packages and Physical Layout 11

2.3 Current Limitations 16

3 Build System **17**

4 MARF Installation Instructions **18**

4.1 Short Version 18
4.2 Requirements 18
4.3 Getting The Source 19
4.4 If You Are Upgrading 19
4.5 Installation Procedure 19
4.6 Configuration 19
4.7 Build 19
4.8 Regression Tests 19
4.9 Installing The Files 20
4.10 Uninstall: 20
4.11 Cleaning: 20
4.12 Environment Variables – CLASSPATH 20
4.13 Getting Started 20
4.14 What Now? 20
4.15 Supported Platforms 20

5 Methodology **21**

5.1 Storage 21
5.1.1 Speaker Database 21
5.1.2 Storing Features, Training, and Classification Data 21
5.1.3 Assorted Training Notes 23
5.1.3.1 Clusters Defined 23
5.1.4 File Location 24
5.1.5 Sample and Feature Sizes 24
5.1.6 Parameter Passing 24
5.1.7 Result 24
5.1.8 Sample Format 25
5.1.9 Sample Loading Process 25
5.2 Assorted File Format Notes 28
5.3 Preprocessing 30
5.3.1 "Raw Meat" 30
5.3.1.1 Description 30

5.3.1.2 Implementation Summary 30

5.3.2 Normalization 30

5.3.3 FFT Filter 30

5.3.4 Low-Pass Filter 32

5.3.5 High-Pass Filter 33

5.3.6 Band-Pass Filter 33

5.3.7 High Frequency Boost 33

5.3.8 High-Pass High Frequency Boost Filter 36

5.3.9 Noise Removal 36

5.4 Feature Extraction 37

5.4.1 Hamming Window 37

5.4.1.1 Implementation 37

5.4.1.2 Theory 37

5.4.2 Fast Fourier Transform (FFT) 39

5.4.2.1 FFT Feature Extraction 39

5.4.3 Linear Predictive Coding (LPC) 40

5.4.3.1 Theory 40

5.4.3.2 Usage for Feature Extraction 41

5.4.4 F0: The Fundamental Frequency 41

5.4.5 Min/Max Amplitudes 42

5.4.5.1 Description 42

5.4.6 Random Feature Extraction 42

5.5 Classification 43

5.5.1 Chebyshev Distance 43

5.5.2 Euclidean Distance 43

5.5.3 Minkowski Distance 45

5.5.4 Mahalanobis Distance 45

5.5.4.1 Summary 45

5.5.4.2 Theory 45

5.5.5 Diff Distance 45

5.5.5.1 Summary 46

5.5.5.2 Theory 46

5.5.6 Artificial Neural Network 46

5.5.6.1 Theory 46

5.5.6.2 Training 47

5.5.6.3 Usage as a Classifier 47

5.5.7 Random Classification 48

6 Natural Language Processing (NLP) 49

7 GUI 50

7.1 Spectrogram 50

7.2 Wave Grapher 52

8 Sample Data and Experimentation 53

8.1 Sample Data 53

8.2 Comparison Setup 53

8.3 What Else Could/Should/Will Be Done 57

8.3.1 Combination of Feature Extraction Methods 57

8.3.2 Entire Recognition Path 57

8.3.3 More Methods 57

9 Experimentation Results 58

9.1 Notes 58

9.2 Configuration Explained 59

9.3 Consolidated Results 61

10 Applications 75

10.1 MARF Research Applications 75

10.1.1 SpeakerIdentApp - Text-Independent Speaker Identification Application 75

10.2 MARF Testing Applications 75

10.2.1 TestFilters 75

10.2.2 TestLPC 77

10.2.3 TestFFT 77

10.2.4 TestWaveLoader 78

10.2.5 TestLoaders 79

10.2.6 MathTestApp 79

10.3 External Applications 80

10.3.1 GIPSY . . . 80
10.3.2 ENCSAssetsDesktop . . . 80
10.3.3 ENCSAssets . . . 81
10.3.4 ENCSAssetsCron . . . 81

11 Conclusions **82**
11.1 Review of Results . . . 82
11.2 Acknowledgments . . . 82

Bibliography **83**

A Spectrogram Examples **84**

B MARF Source Code **85**

C The CVS Repository **86**
C.1 Getting The Source Via Anonymous CVS . . . 86

D `SpeakerIdentApp` and `SpeakersIdentDb` Source Code **88**
D.1 `SpeakerIdentApp.java` . . . 88
D.2 `SpeakersIdentDb.java` . . . 97

E TODO **108**

List of Figures

2.1 Overall Architecture . . . 9
2.2 The Core Pipeline Sequence Diagram . . . 10
2.3 The Core Pipeline Data Flow . . . 11
2.4 MARF Java Packages . . . 12

5.1 Storage . . . 22
5.2 Preprocessing . . . 31
5.3 Normalization of aihua5.wav from the testing set. . . . 32
5.4 FFT of normalized aihua5.wav from the testing set. . . . 33
5.5 Low-pass filter applied to aihua5.wav. . . . 34
5.6 High-pass filter applied to aihua5.wav. . . . 34
5.7 Band-pass filter applied to aihua5.wav. . . . 35
5.8 High frequency boost filter applied to aihua5.wav. . . . 35
5.9 Feature Extraction Class Diagram . . . 38
5.10 Classification . . . 44

7.1 GUI Package . . . 51

A.1 LPC spectrogram obtained for ian15.wav . . . 84
A.2 LPC spectrogram obtained for `graham13.wav` . . . 84

List of Tables

8.1 Speakers contributed their voice samples. 54

9.1 Consolidated results, Part 1. 62
9.2 Consolidated results, Part 2. 63
9.3 Consolidated results, Part 3. 64
9.4 Consolidated results, Part 4. 65
9.5 Consolidated results, Part 5. 66
9.6 Consolidated results, Part 6. 67
9.7 Consolidated results, Part 7. 68
9.8 Consolidated results, Part 8. 69
9.9 Consolidated results, Part 9. 70
9.10 Consolidated results, Part 10. 71
9.11 Consolidated results, Part 11. 72
9.12 Consolidated results, Part 12. 73
9.13 Consolidated results, Part 13. 74

Chapter 1

Introduction

Revision : 1.20

1.1 What is MARF?

MARF stands for **M**odular **A**udio **R**ecognition **F**ramework. It contains a collection of algorithms for Sound, Speech, and Natural Language Processing arranged into an uniform framework to facilitate addition of new algorithms for preprocessing, feature extraction, classification, parsing, etc. implemented in Java. MARF is also a research platform for various performance metrics of the implemented algorithms.

1.1.1 Purpose

Our main goal is to build a general open-source framework to allow developers in the audio-recognition industry (be it speech, voice, sound, etc.) to choose and apply various methods, contrast and compare them, and use them in their applications. As a proof of concept, a user frontend application for Text-Independent (TI) Speaker Identification has been created on top of the framework (the `SpeakerIdentApp` program). A variety of testing applications and applications that show how to use various aspects of MARF are also present. A new recent addition is the experimental NLP support, which is also included in MARF as of 0.3.0-devel-20050606. For more information on applications that employ MARF see Chapter 10.

1.1.2 Why Java?

We have chosen to implement our project using the Java programming language. This choice is justified by the binary portability of the Java applications as well as facilitating memory management tasks and other issues, so we can concentrate more on the algorithms instead. Java also provides us with built-in types and data-structures to manage collections (build, sort, store/retrieve) efficiently [Fla97].

1.1.3 Terminology and Notation

Revision : 1.4

The term "MARF" will be to refer to the software that accompanies this documentation. An **application programmer** could be anyone who is using, or wants to use, any part of the MARF system. A **MARF developer** is a core member of MARF who is hacking away the MARF system.

1.2 Authors, Contact, and Copyright Information

1.2.1 COPYRIGHT

Revision : 1.8

MARF is Copyright © 2002-2005 by the MARF Development Group and is distributed under the terms of the BSD-style license below.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL CONCORDIA UNIVERSITY OR THE AUTHORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF CONCORDIA UNIVERSITY OR THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CONCORDIA UNIVERSITY AND THE AUTHORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS-IS" BASIS, AND CONCORDIA UNIVERSITY AND THE AUTHORS HAVE NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

1.2.2 Authors

Authors Emeritus, in alphabetical order:

- Ian Clément, `i_clemen@cs.concordia.ca`
- Serguei Mokhov, `mokhov@cs.concordia.ca`, a.k.a Serge
- Dimitrios Nicolacopoulos, `d_nicola@cs.concordia.ca`, a.k.a Jimmy

- Stephen Sinclair, `step_sin@cs.concordia.ca`, a.k.a. Steve, radarsat1

Current maintainers:

- Serguei Mokhov, `mokhov@cs.concordia.ca`
- Shuxin Fan, `fshuxin@gmail.com`, a.k.a Susan

If you have some suggestions, contributions to make, or for bug reports, don't hesitate to contact us :-) For MARF-related issues please contact us at `marf-devel@lists.sf.net`. Please report bugs to `marf-bugs@lists.sf.net`.

1.3 A Short History of MARF

Revision : 1.9

The MARF project was initiated in September 26, 2002 by four students of Concordia University as their course project for Pattern Recognition under guidance of Dr. C.Y. Suen. This included Ian Clement, Stephen Sinclair, Jimmy Nicolacopoulos, and Serguei Mokhov.

1.3.1 Developers Emeritus

- Ian's primary contributions were the LPC and Neural Network algorithms support with the Spectrogram dump.
- Steve has done an extensive research and implementation of the FFT algorithm for feature extraction and filtering and Euclidean Distance with the `WaveGrapher` class.
- Jimmy was focused on implementation of the WAVE file format loader and other storage issues.
- Serguei designed the entire MARF framework and architecture, originally implemented general distance classifier and its Chebyshev, Minkowski, and Mahalanobis incarnations along with normalization of the sample data. Serguei designed the Exceptions Framework of MARF and was involved into the integration of all the modules and testing applications to use MARF.

1.3.2 Current Status and Past Releases

Now it's a project on its own, being maintained and developed as we have some spare time for it. When the course was over, Serguei Mokhov is the primary maintainer of the project. He rewrote Storage support and polished all of MARF during two years and added various utility modules and NLP support and implementation of new algorithms and applications. Serguei maintains this manual, the web site and most of the sample database collection. He also made all the releases of the project as follows:

- 0.3.0-devel-20050817, Wednesday, August 17, 2005
- 0.3.0-devel-20050730, Saturday, July 30, 2005
- 0.3.0-devel-20050606, Monday, June 6, 2005
- 0.3.0-devel-20040614, Monday, June 14, 2004
- 0.2.1, Monday, February 17, 2003
- 0.2.0, Monday, February 10, 2003
- 0.1.2, December 17, 2002 - Final Project Deliverable
- 0.1.1, December 8, 2002 - Demo

The project is currently geared towards completion planned TODO items on MARF and its application along with recently joined Shuxin'Susan'Fan.

1.4 MARF Source Code

Revision : 1.8

1.4.1 Project Source and Location

Our project since the its inception has always been an open-source project. All releases including the most current one should most of the time be accessible via `<http://marf.sourceforge.net>` provided by `SourceForge.net`. We have a complete API documentation as well as this manual and all the sources available to download through this web page.

1.4.2 Formatting

Source code formatting uses a 4 column tab spacing, currently with tabs preserved (i.e. tabs are not expanded to spaces).

For Emacs, add the following (or something similar) to your ~/`.emacs` initialization file:

```
;; check for files with a path containing "marf"
(setq auto-mode-alist
  (cons '("\\(marf\\).*\\.java\\'" . marf-java-mode)
        auto-mode-alist))
(setq auto-mode-alist
  (cons '("\\(marf\\).*\\.java\\'" . marf-java-mode)
        auto-mode-alist))
```

```
(defun marf-java-mode ()
  ;; sets up formatting for MARF Java code
  (interactive)
  (java-mode)
  (setq-default tab-width 4)
  (java-set-style "bsd")      ; set java-basic-offset to 4, etc.
  (setq indent-tabs-mode t))  ; keep tabs for indentation
```

For `vim`, your ~/`.vimrc` or equivalent file should contain the following:

```
set tabstop=4
```

or equivalently from within vim, try

```
:set ts=4
```

The text browsing tools `more` and `less` can be invoked as

```
more -x4
less -x4
```

1.4.3 Coding and Naming Conventions

For now, please see `http://marf.sf.net/coding.html`.

TODO

Chapter 2

MARF Architecture

Revision : 1.26

Before we begin, you should understand the basic MARF system architecture. Understanding how the parts of MARF interact will make the follow up sections somewhat clearer. This document presents architecture of the MARF system, including the layout of the physical directory structure, and Java packages.

Let's take a look at the general MARF structure in Figure 2.1. The `MARF` class is the central "server" and configuration "placeholder", which contains the major methods – the core pipeline – a typical pattern recognition process. The figure presents basic abstract modules of the architecture. When a developer needs to add or use a module, they derive from the generic ones.

The core pipeline sequence diagram from an application up until the very end result is presented in Figure 2.2. It includes all major participants as well as basic operations. The participants are the modules responsible for a typical general pattern recognition pipeline. A conceptual data-flow diagram of the pipeline is in Figure 2.3. The grey areas indicate stub modules that are yet to be implemented.

Consequently, the framework has the mentioned basic modules, as well as some additional entities to manage storage and serialization of the input/output data.

2.1 Application Point of View

An application, using the framework, has to choose the concrete configuration and submodules for preprocessing, feature extraction, and classification stages. There is an API the application may use defined by each module or it can use them through the MARF.

There are two phases in MARF's usage by an application:

- Training, i.e. `train()`
- Recognition, i.e. `recognize()`

Training is performed on a virgin MARF installation to get some training data in. Recognition is an actual identification process of a sample against previously stored patterns during training.

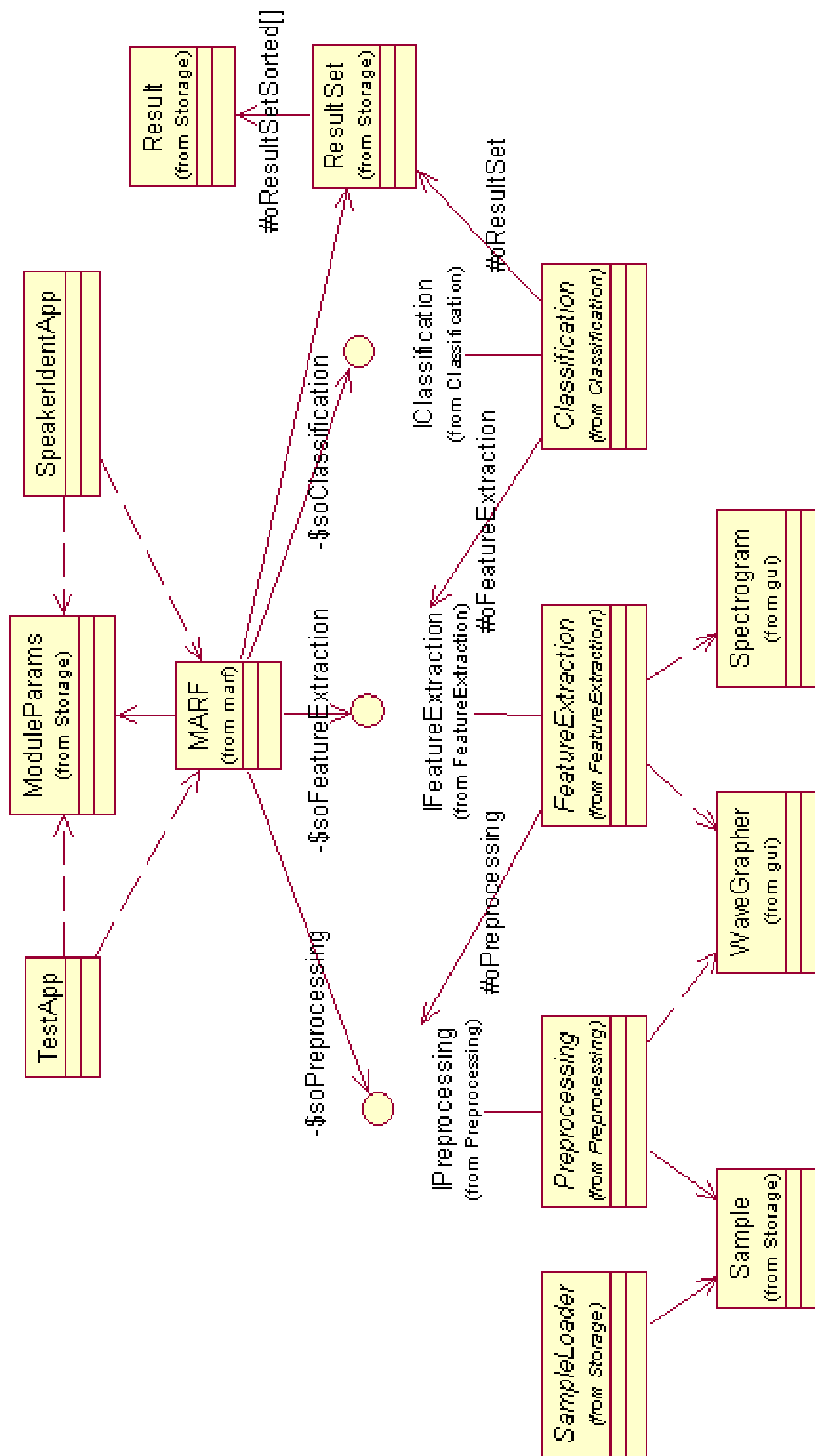

Figure 2.1: Overall Architecture

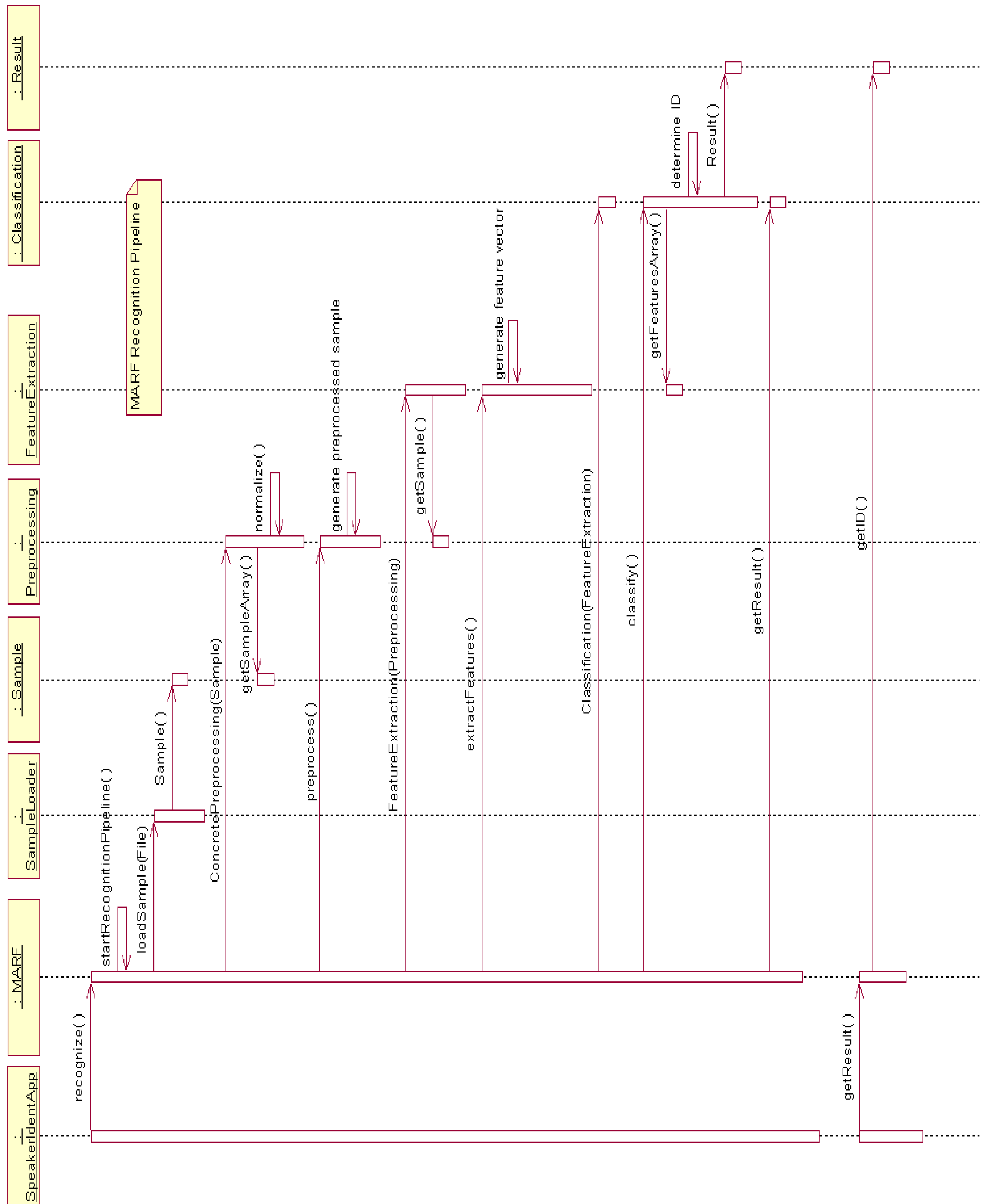

Figure 2.2: The Core Pipeline Sequence Diagram

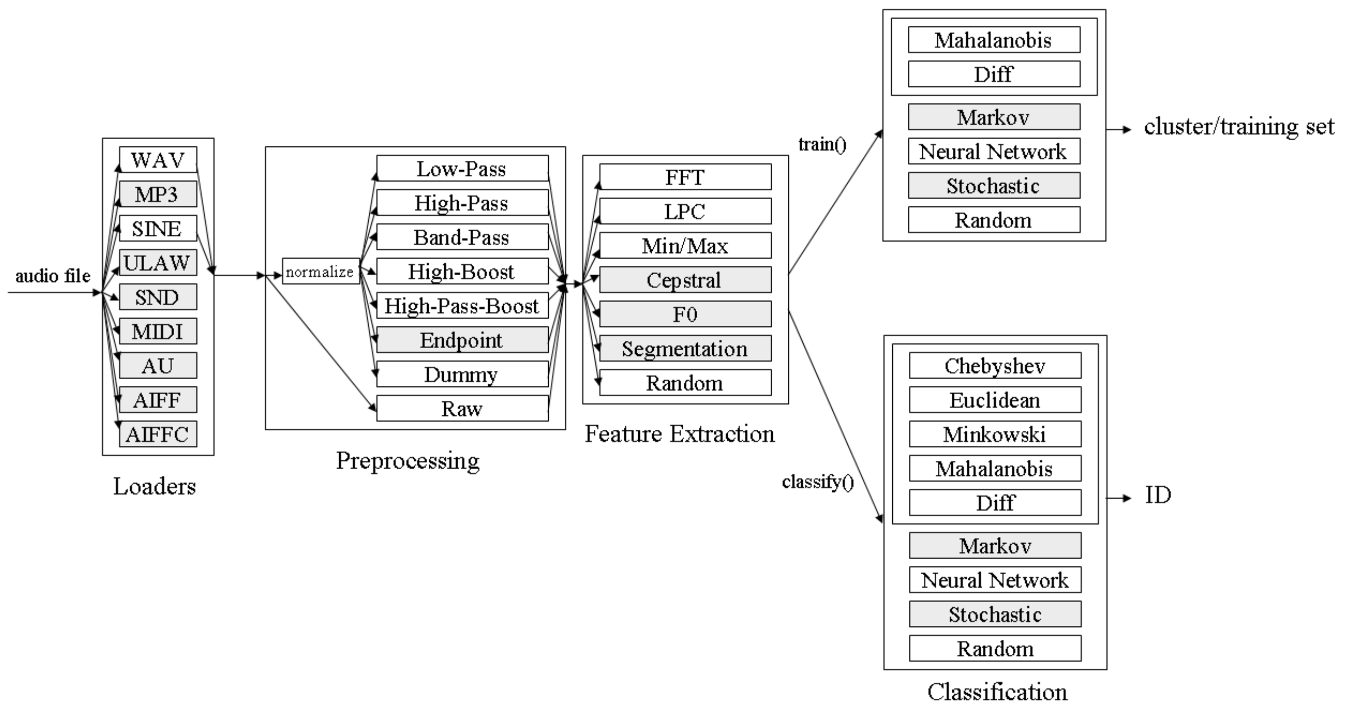

Figure 2.3: The Core Pipeline Data Flow

2.2 Packages and Physical Layout

The Java package structure is in Figure 2.4. The following is the basic structure of MARF:

```
marf.*

MARF.java - The MARF Server
            Supports Training and Recognition mode
            and keeps all the configuration settings.

marf.Preprocessing.* - The Preprocessing Package

/marf/Preprocessing/

   Preprocessing.java - Abstract Preprocessing Module, has to be subclassed
   PreprocessingExceotion.java

   /Endpoint/*.java   - Endpoint Filter as implementation of Preprocessing
   /Dummy/
       Dummy.java  - Normalization only
       Raw.java    - no preprocessing
```

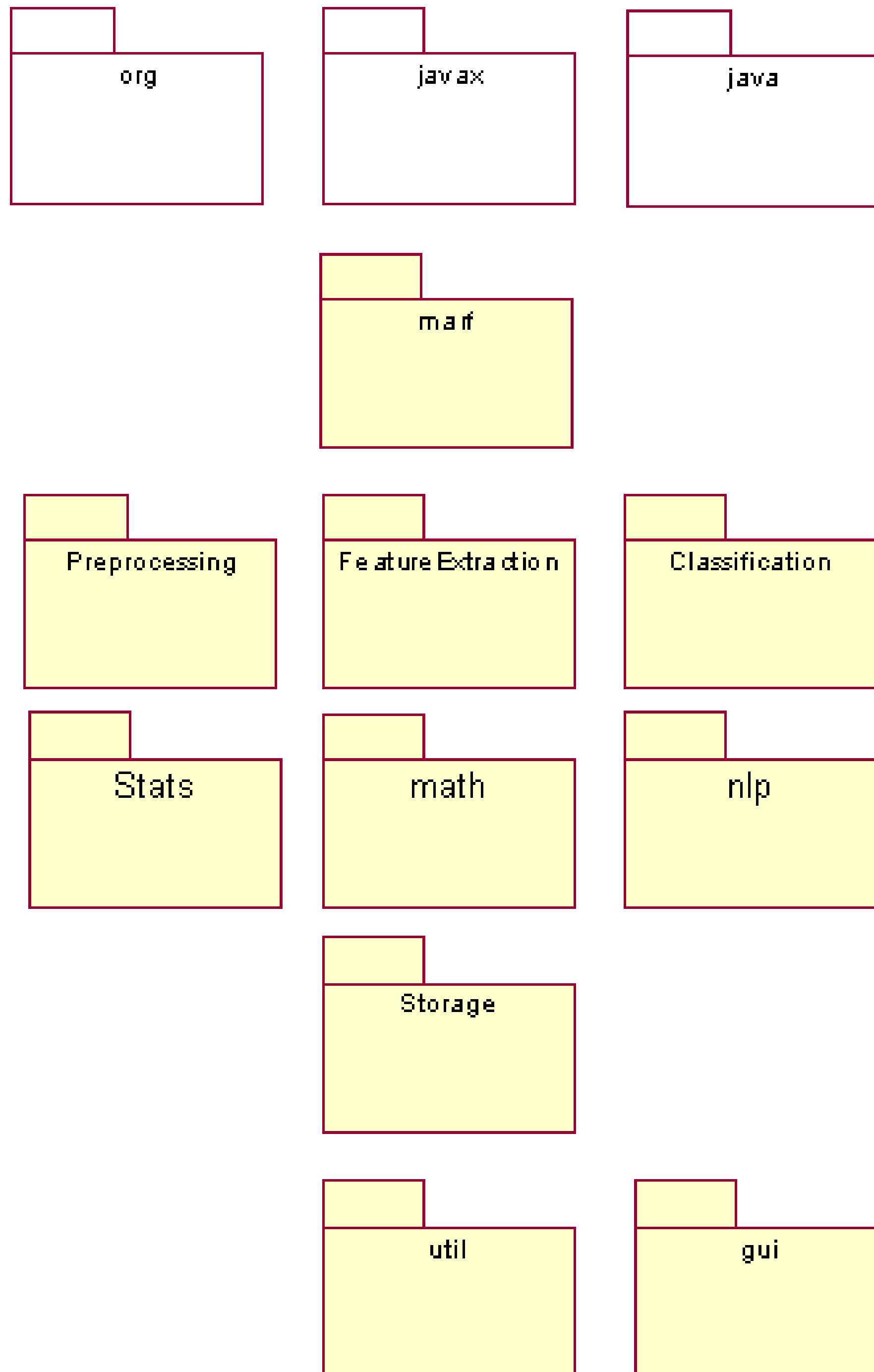

Figure 2.4: MARF Java Packages

```
   /FFTFilter/
       FFTFilter.java
       LowPassFilter.java
       HighPassFilter.java
       BandpassFilter.java - Band-pass Filter as implementation of Preprocessing
       HighFrequencyBoost.java

marf.FeatureExtraction.* - The Feature Extraction Package

/marf/FeatureExtraction/
    FeatureExtraction.java
    /FFT/FFT.java        - FFT implementation of Preprocessing
    /LPC/LPC.java        - LPC implementation of Preprocessing
    /MinMaxAmplitudes/MinMaxAmplitudes.java
    /Cepstral/*.java
    /Segmentation/*.java
    /F0/*.java

marf.Classification.* - The Classification Package

/marf/Classification/
    Classification.java
    ClassificationException.java
    /NeuralNetwork/
        NeuralNetwork.java
        Neuron.java
        Layer.java
    /Stochastic/*.java
    /Markov/*.java
    /Distance/
       Distance.java
       EuclideanDistance.java
       ChebyshevDistance.java
       MinkowskiDistance.java
       MahalonobisDistance.java
       DiffDistance.java

marf.Storage.* - The Physical Storage Management Interface

/marf/Storage/
```

```
    Sample.java
    ModuleParams.java
    TrainingSet.java
    FeatureSet.java
    Cluster.java
    Result.java
    ResultSet.java
    IStorageManager.java - Interface to be implemented by the above modules
    StorageManager.java  - The most common implementation of IStorageManager
    ISampleLoader.java   - All loaders implement this
    SampleLoader.java    - Should know how to load different sample format
    /Loaders/*.*         - WAV, MP3, ULAW, etc.
    IDatabase.java
    Database.java

marf.Stats.* - The Statistics Package meant to collect various types of stats.

/marf/Stats/
    StatsCollector.java - Time took, noise removed, patterns stored, modules available, etc.
    Ngram.java
    Observation.java
    ProbabilityTable.java
    StatisticalEstimators
    StatisticalObject.java
    WordStats.java
    /StatisticalEstimators/
        GLI.java
        KatzBackoff.java
        MLE.java
        SLI.java
        StatisticalEstimator.java
        /Smoothing/
            AddDelta.java
            AddOne.java
            GoodTuring.java
            Smoothing.java
            WittenBell.java

marf.gui.* - GUI to the graphs and configuration
```

```
/marf/gui/
    Spectrogram.java
    SpectrogramPanel.java
    WaveGrapher.java
    WaveGrapherPanel.java
    /util/
       BorderPanel.java
       ColoredStatusPanel.java
       SmartSizablePanel.java

marf.nlp.* - most of the NLP-related modules

/marf/nlp/
    Collocations/
    Parsing/
    Stemming/
    util/

marf.math.* - math-related algorithms are here

/marf/math/
    Algorithms.java
    MathException.java
    Matrix.java
    Vector.java

marf.util.* - important utility modules

/marf/util/
    Arrays.java
    BaseThread.java
    ByteUtils.java
    Debug.java
    ExpandedThreadGroup.java
    FreeVector.java
    InvalidSampleFormatException.java
    Logger.java
    MARFException.java
    Matrix.java
    NotImplementedException.java
```

```
OptionProcessor.java
SortComparator.java
/comparators/
    FrequencyComparator.java
    RankComparator.java
    ResultComparator.java
```

2.3 Current Limitations

Our current pipeline is maybe somewhat too rigid. That is, there's no way to specify more than one preprocessing or feature extraction module to process the same sample in one pass (as of 0.3.0.2 the preprocessing modules can be chained however. E.g. one filter may be used along with normalization together, or just normalization by itself).

Also, it assumes that the whole sample is loaded before doing anything with it, instead of sending parts of the sample a bit at a time. Perhaps this simplifies things, but it won't allow us to deal with large samples at the moment. However, it's not a problem for our framework and the application since memory is cheap and samples are not too big. Additionally, we have streaming support already in the `WAVLoader` and some modules support it, but the final conversion to streaming did not happen in this release.

MARF provides only limited support for inter-module dependency. It is possible to pass module-specific arguments, but problems like number of parameters mismatch between feature extraction and classification, and so on are not tracked.

Chapter 3

Build System

Revision : 1.1

- Makefiles
- Eclipse
- NetBeans
- JBuilder

TODO

Chapter 4

MARF Installation Instructions

Revision : 1.5

Sync with the `INSTALL` file. Please see this for now for installation instructions and ignore the rest of this chapter until we automate sync between the two files.

TODO

4.1 Short Version

```
make install
```

The long version is the rest of this chapter.

4.2 Requirements

In general, any modern platform should be able to run MARF. The following software packages are required for building MARF:

- GNU `make` is required; other `make` programs will *not* work. GNU `make` is often installed under the name `gmake`; this document will always refer to it by that name. (On some systems GNU make is the default tool with the name `make`.) To test for GNU `make` enter
- You need a Java compiler. Recent versions of `javac` are recommendable.
- `NeuralNetwork` module requires the JAXP XML parser for JDK 1.3. You can get it at `http://java.sun.com/xml/downloads/javaxmlpack.html`. Click the "Download now" under the heading "Java XML Pack - Summer 02 Update Release". This should be the right one.

The following packages are required in the default configuration, as explained below.

4.3 Getting The Source

The MARF sources can be obtained from `http://marf.sf.net`.

```
tar xvfz marf-<version>.tar.gz
```

This will create a directory `marf-<version>` under the current directory with the MARF sources. Change into that directory for the rest of the installation procedure.

4.4 If You Are Upgrading

The internal data storage format changes with new releases of MARF. Therefore, .

If you are installing in the same place as the old version then it is also a good idea to move the old installation out of the way, in case you have trouble and need to revert to it. Use a command like this:

```
mv
```

4.5 Installation Procedure

4.6 Configuration

4.7 Build

To start the build, type

```
gmake
```

(Remember to use GNU `make`.) The last line displayed should be

```
(-: MARF build has been successful :-)
```

4.8 Regression Tests

If you want to test the newly built MARF before you install it, you can run the regression tests at this point. The regression tests are a test suite to verify that MARF runs on your machine in the way the developers expected it to. Type

```
gmake test
```

4.9 Installing The Files

To install MARF enter

```
gmake install
```

This will install the .jar file into the directory specified.

4.10 Uninstall:

remove the pertinent .jar file(s).

4.11 Cleaning:

After the installation you can make room by removing the built files from the source tree with the command `gmake clean`. This will preserve the files made by the configure program, so that you can rebuild everything with `gmake` later on.

4.12 Environment Variables – CLASSPATH

4.13 Getting Started

The following is a quick summary of how to get MARF up and running once installed.

4.14 What Now?

4.15 Supported Platforms

MARF has been verified by the developers to work on the platforms a JVM runs on listed below. A supported platform generally means that MARF builds and installs according to these instructions. Virtually any Java 1.4 or 1.5-enabled platform should be able to bear MARF.

Chapter 5

Methodology

Revision : 1.10

This section presents what methods and algorithms were implemented and used in this project. We overview storage issues first, then preprocessing methods followed by feature extraction, and ended by classification.

5.1 Storage

Revision : 1.19

Figure 5.1 presents basic `Storage` modules and their API.

5.1.1 Speaker Database

We store specific speakers in a comma-separated (CSV) file, `speakers.txt` within the application. It has the following format:

```
<id:int>,<name:string>,<training-samples:list>,<testing-samples:list>
```

Sample lists are defined as follows:

```
<*-sample-list> := filename1.wav|filename2.wav|...
```

5.1.2 Storing Features, Training, and Classification Data

We defined a standard `StorageManager` interface for the modules to use. That's part of the `StorageManager` interface which each module will override because each a module has to know how to serialize itself, but the applications using MARF should not care. Thus, this `StorageManager` is a base class with abstract methods `dump()` and `restore()`. These methods would generalize the model's serialization, in the sense that they are somehow "read" and "written".

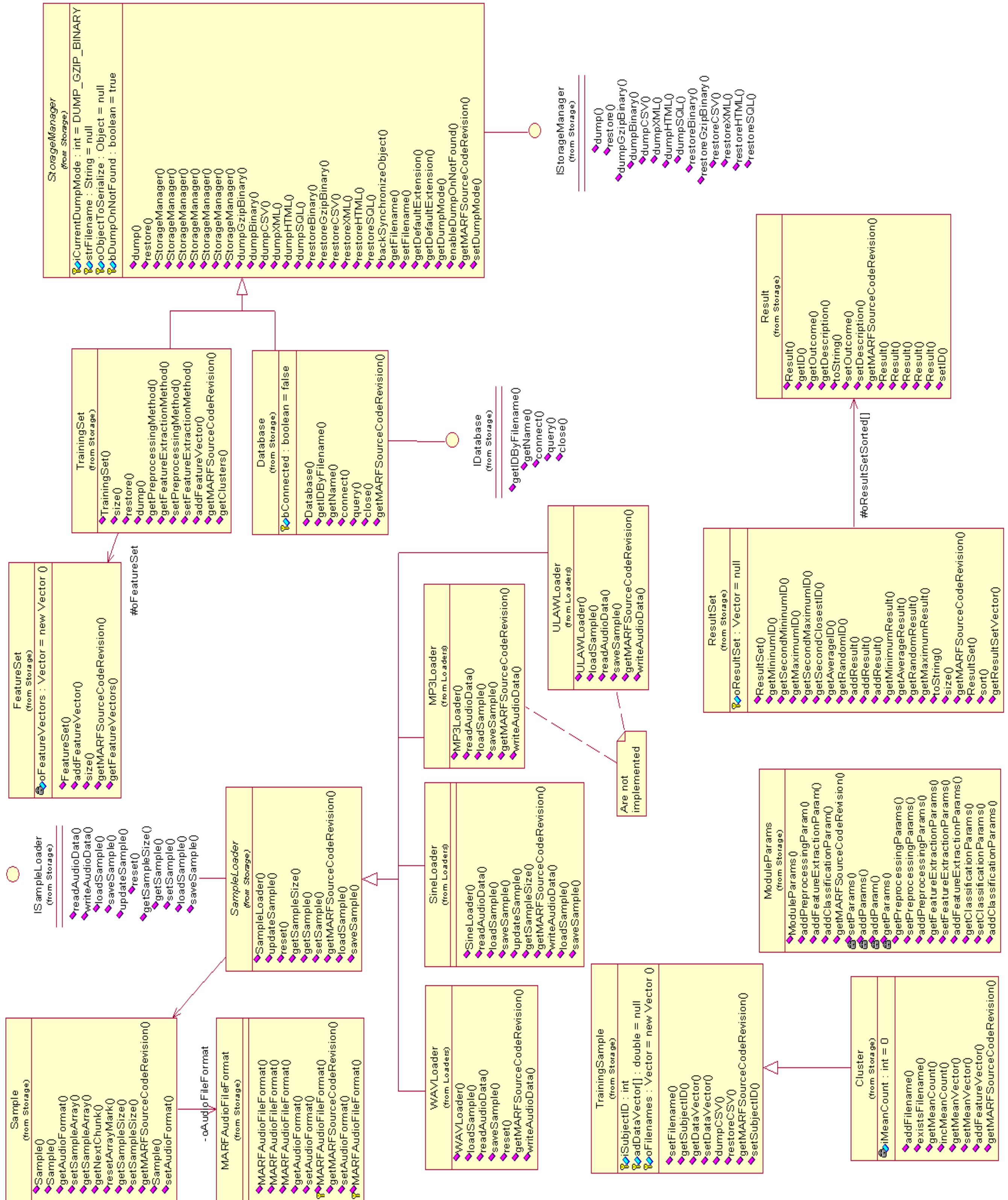

Figure 5.1: Storage

We have to store data we used for training for later use in the classification process. For this we pass FFT (Figure 5.4.2) and LPC (Figure 5.4.3) feature vectors through the `TrainingSet/TrainingSample` class pair, which, as a result, store mean vectors (clusters) for our training models.

In the Neural Network we use XML. The only reason XML and text files have been suggested is to allow us to easily modify values in a text editor and verify the data visually.

In the Neural Network classification, we are using one net for all the speakers. We had thought that one net for each speaker would be ideal, but then we'll lose too much discrimination power. But doing this means that the net will need complete re-training for each new training utterance (or group thereof).

We have a training/testing script that lists the location of all the wave files to be trained along with the identification of the speaker - `testing.sh`.

5.1.3 Assorted Training Notes

Revision : 1.5

For training we will be using sets of feature vectors created by FFT or LPC and passing them to a NNet or something like we discussed last night (cluster, or rather probabilistic "cluster"). But I think there are some issues:

1. Mapping. We will need a record of Speakers and ID for these speakers. This can be the same for NNet and Stochastic methods so long as the NNet will return the proper number and the "clusters" in the Stochastic module have proper labeling. We will also need a mechanism for adding speakers in marf (esp. if were to add a speaker during the demo!).

2. Feature vector generation. I think the best way to do this is for each application using marf to specify a feature file or directory which will contain lines/files with the following info:

   ```
   [a1, a2, ... , an] : <speaker id>
   ```

   Retraining for a new speaker would involve two phases 1) appending the features to the file/dir, then 2) re-training the models. The Classification modules will be aware of the scheme and re-train on all data required.

5.1.3.1 Clusters Defined

Given a set of feature vectors in n-dimensional space, if we want these to represent m "items" (in our case, speakers), we can make groupings of these vectors with a center point c_i (ie: m center points which will then represent the "items"). Suen discussed an iterative algorithm to find the optimal groupings (or clusters), but I forgot to fill you in :-(. Anyway, I don't believe that Suen's clustering stuff is at all useful, as we will know, through info from training, which speaker is associated with the feature vector and can create the "clusters" with that information.

So for NNet: No clusters, just regular NNet training. So for Stochastic: Clusters (kind of). I believe that we need to represent a Gaussian curve with a mean vector and a co-variance matrix. This will be

created from the set of feature vectors for the speaker. But again, we know who it is so the Optimal Clustering business is useless. If we do get stochastic done :-(.

5.1.4 File Location

We decided to keep all the data and intermediate files in the same directory or subdirectories of that of the application.

- `marf.Storage.TrainingSet.*` - represent training sets (global clusters) used in training with different preprocessing and feature extraction methods; they can either be gzipped binary (.bin) or CSV text (.csv).
- `speakers.txt.stats` - binary statistics file.
- `marf.Classification.NeuralNetwork.*.xml` - XML file representing a trained Neural Net for all the speakers in the database.
- `training-samples/` - directory with WAV files for training.
- `testing-samples/` - directory with WAV files for testing.

5.1.5 Sample and Feature Sizes

Wave files are read and outputted as an array of data points that represents the waveform of the signal.

Different methods will have different feature vector sizes. It depends on what kind of precision one desires. In the case of FFT, a 1024 FFT will result in 512 features, being an array of "doubles" corresponding to the frequency range.

[O'S00] said about using 3 ms for phoneme analysis and something like one second for general voice analysis. At 8 kHz, 1024 samples represents 128 ms, this might be a good compromise.

5.1.6 Parameter Passing

A generic `ModuleParams` container class has been created to for an application to be able to pass module-specific parameters when specifying model files, training data, amount of LPC coefficients, FFT window size, logging/stats files, etc.

5.1.7 Result

When classification is over, its result should be stored somehow for further retrieval by the application. We have defined the `Result` object to carry out this task. It contains ID of the subject identified as well as some additional statistics (such as second closest speaker and distances from other speakers, etc.)

5.1.8 Sample Format

Revision : 1.20

The sample format used for our samples was the following:

- Audio Format: PCM signed (WAV)
- Sample Rate: 8000 Hz
- Audio Sample Size: 16 bit
- Channels: 1 (mono)
- Duration: from about 7 to 20 seconds

All training and testing samples were recorded through an external sound recording program (MS Sound Recorder) using a standard microphone. Each sample was saved as a WAV file with the above properties and stored in the appropriate folders where they would be loaded from within the main application. The PCM audio format (which stands for Pulse Code Modulation) refers to the digital encoding of the audio sample contained in the file and is the format used for WAV files. In a PCM encoding, an analog signal is represented as a sequence of amplitude values. The range of the amplitude value is given by the audio sample size which represents the number of bits that a PCM value consists of. In our case, the audio sample size is 16-bit which means that that a PCM value can range from 0 to 65536. Since we are using PCM-signed format, this gives an amplitude range between -32768 and 32768. That is, the amplitude values of each recorded sample can vary within this range. Also, this sample size gives a greater range and thus provides better accuracy in representing an audio signal then using a sample size of 8-bit which limited to a range of $(-128, 128)$. Therefore, a 16-bit audio sample size was used for our experiments in order to provide the best possible results. The sampling rate refers to the number of amplitude values taken per second during audio digitization. According to the Nyquist theorem, this rate must be at least twice the maximum rate (frequency) of the analog signal that we wish to digitize ([IJ02]). Otherwise, the signal cannot be properly regenerated in digitized form. Since we are using an 8 kHz sampling rate, this means that actual analog frequency of each sample is limited to 4 kHz. However, this limitation does not pose a hindrance since the difference in sound quality is negligible ([O'S00]). The number of channels refers to the output of the sound (1 for mono and 2 for stereo – left and right speakers). For our experiment, a single channel format was used to avoid complexity during the sample loading process.

5.1.9 Sample Loading Process

To read audio information from a saved voice sample, a special sample-loading component had to be implemented in order to load a sample into an internal data structure for further processing. For this, certain sound libraries (`javax.sound.sampled`) were provided from the Java programming language which enabled us to stream the audio data from the sample file. However once the data was captured, it had to be converted into readable amplitude values since the library routines only provide PCM values of

the sample. This required the need to implement special routines to convert raw PCM values to actual amplitude values (see `SampleLoader` class in the `Storage` package).

The following pseudo-code represents the algorithm used to convert the PCM values into real amplitude values ([Mic05]):

```
function readAmplitudeValues(Double Array : audioData)
{
    Integer: MSB, LSB,
             index = 0;

    Byte Array: AudioBuffer[audioData.length * 2];

    read audio data from Audio stream into AudioBuffer;

    while(not reached the end of stream OR index not equal to audioData.length)
    {
        if(Audio data representation is BigEndian)
        {
            // First byte is MSB (high order)
            MSB = audioBuffer[2 * index];
            // Second byte is LSB (low order)
            LSB = audioBuffer[2 * index + 1];
        }
        else
        {
            // Vice-versa...
            LSB = audioBuffer[2 * index];
            MSB = audioBuffer[2 * index + 1];
        }

        // Merge high-order and low-order byte to form a 16-bit double value.
        // Values are divided by maximum range
        audioData[index] = (merge of MSB and LSB) / 32768;
    }
}
```

This function reads PCM values from a sample stream into a byte array that has twice the length of `audioData`; the array which will hold the converted amplitude values (since sample size is 16-bit). Once the PCM values are read into `audioBuffer`, the high and low order bytes that make up the amplitude value are extracted according to the type of representation defined in the sample's audio format. If the data representation is *big endian*, the high order byte of each PCM value is located at every even-numbered position in `audioBuffer`. That is, the high order byte of the first PCM value is found at position 0, 2 for

the second value, 4 for the third and so forth. Similarly, the low order byte of each PCM value is located at every odd-numbered position (1, 3, 5, etc.). In other words, if the data representation is *big endian*, the bytes of each PCM code are read from left to right in the `audioBuffer`. If the data representation is not *big endian*, then high and low order bytes are inversed. That is, the high order byte for the first PCM value in the array will be at position 1 and the low order byte will be at position 0 (read right to left). Once the high and low order bytes are properly extracted, the two bytes can be merged to form a 16-bit double value. This value is then scaled down (divide by 32768) to represent an amplitude within a unit range $(-1, 1)$. The resulting value is stored into the `audioData` array, which will be passed to the calling routine once all the available audio data is entered into the array. An additional routine was also required to write audio data from an array into wave file. This routine involved the inverse of reading audio data from a sample file stream. More specifically, the amplitude values inside an array are converted back to PCM codes and are stored inside an array of bytes (used to create new audio stream). The following illustrates how this works:

```
public void writePCMValues(Double Array: audioData)
{
    Integer: word  = 0,
             index = 0;

    Byte Array: audioBytes[(number of ampl. values in audioData) * 2];

    while(index not equal to (number of ampl. values in audioData * 2))
    {
        word = (audioData[index] * 32768);
        extract high order byte and place it in appropriate position in audioBytes;
        extract low order byte and place it in appropriate position in audioBytes;
    }

    create new audio stream from audioBytes;
}
```

5.2 Assorted File Format Notes

Revision : 1.4

We decided to stick to Mono-8000Hz-16bit WAV files. 8-bit might be okay too, but we could retain more precision with 16-bit files. 8000Hz is supposed to be all you need to contain all frequencies of the vocal spectrum (according to Nyquist anyways...). If we use 44.1 kHz we'll just be wasting space and computation time.

There are also MP3 and ULAW and other file format loaders stubs which are unimplemented as of this version of MARF.

Also: I was just thinking I think I may have made a bit of a mistake downsampling to 8000Hz... I was saying that the voice ranges to about 8000Hz so that's all we should need for the samples, but then I realized that if you have an 8000Hz sample, it actually only represents 4000Hz, which would account for the difference I noticed.. but maybe we should be using 16KHz samples. On the other hand, even at 4KHz the voice is still perfectly distinguishable...

I tried the WaveLoader with one of the samples provided by Stephen (jimmy1.wav naturally!) and got some nice results! I graphed the PCM obtained from the getaudioData() function and noticed quite a difference from the PCM graph obtained with my "test.wav". With "test.wav", I was getting unexpected results as the graph ("rawpcm.xls") didn't resemble any wave form. This lead me to believe that I needed to convert the data on order to represent it in wave form (done in the "getWaveform()" function). But after having tested the routine with "jimmy1.wav", I got a beautiful wave-like graph with just the PCM data which makes more sense since PCM represents amplitude values! The reason for this is that my "test.wav" sample was actually 8-bit mono (less info.) rather than 16-bit mono as with Stephen's samples. So basically, we don't need to do any "conversion" if we use 16-bit mono samples and we can scrap the "getWaveform()" function. I will come up with a "Wave" class sometime this week which will take care of loading wave files and windowing audio data. Also, we should only read audio data that has actual sound, meaning that any silence (say -10 ¡ db ¡ 10) should be discarded from the sample when extracting audio data. Just thinking out loud!

I agree here. I was thinking perhaps the threshold could be determined from the "silence" sample.

5.3 Preprocessing

Revision : 1.16

This section outlines the preprocessing mechanisms considered and implemented in MARF. We present you with the API and structure in Figure 5.2, along with the description of the methods.

5.3.1 "Raw Meat"

Revision : 1.3

5.3.1.1 Description

This is a basic "pass-everything-through" method that doesn't do actually any preprocessing. Originally developed within the framework, it was meant to be a base line method, but it gives three best results out of four in three configurations.

5.3.1.2 Implementation Summary

- Implementation: `marf.Preprocessing.Dummy.Raw`
- Depends on: `marf.Preprocessing.Dummy.Dummy`
- Used by: `test`, `marf.MARF`, `SpeakerIdentApp`

5.3.2 Normalization

Since not all voices will be recorded at exactly the same level, it is important to normalize the amplitude of each sample in order to ensure that features will be comparable. Audio normalization is analogous to image normalization. Since all samples are to be loaded as floating point values in the range $[-1.0, 1.0]$, it should be ensured that every sample actually does cover this entire range.

The procedure is relatively simple: find the maximum amplitude in the sample, and then scale the sample by dividing each point by this maximum. Figure 5.3 illustrates normalized input wave signal.

5.3.3 FFT Filter

The FFT filter is used to modify the frequency domain of the input sample in order to better measure the distinct frequencies we are interested in. Two filters are useful to speech analysis: high frequency boost, and low-pass filter (yet we provided more of them, to toy around).

Speech tends to fall off at a rate of 6 dB per octave, and therefore the high frequencies can be boosted to introduce more precision in their analysis. Speech, after all, is still characteristic of the speaker at

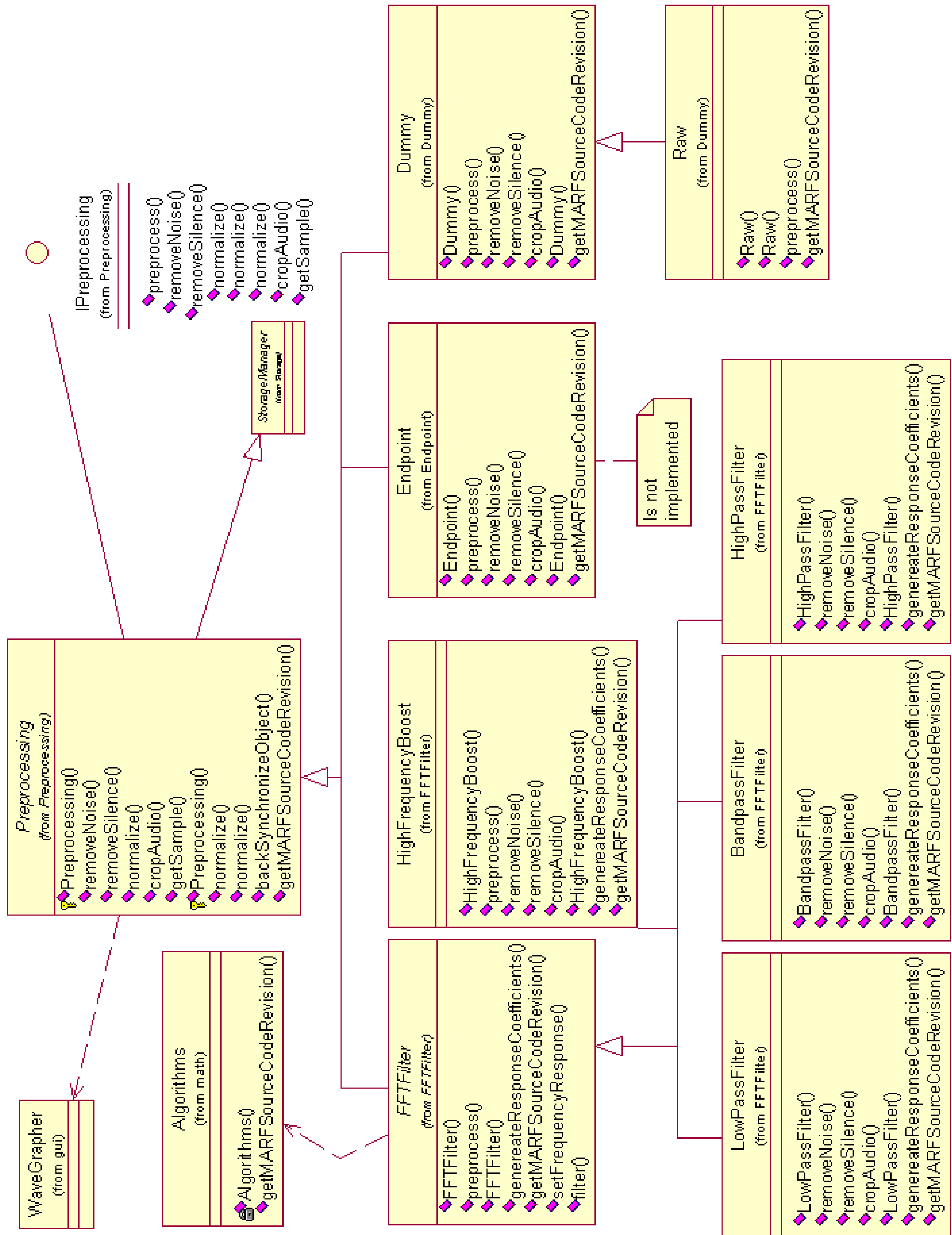

Figure 5.2: Preprocessing

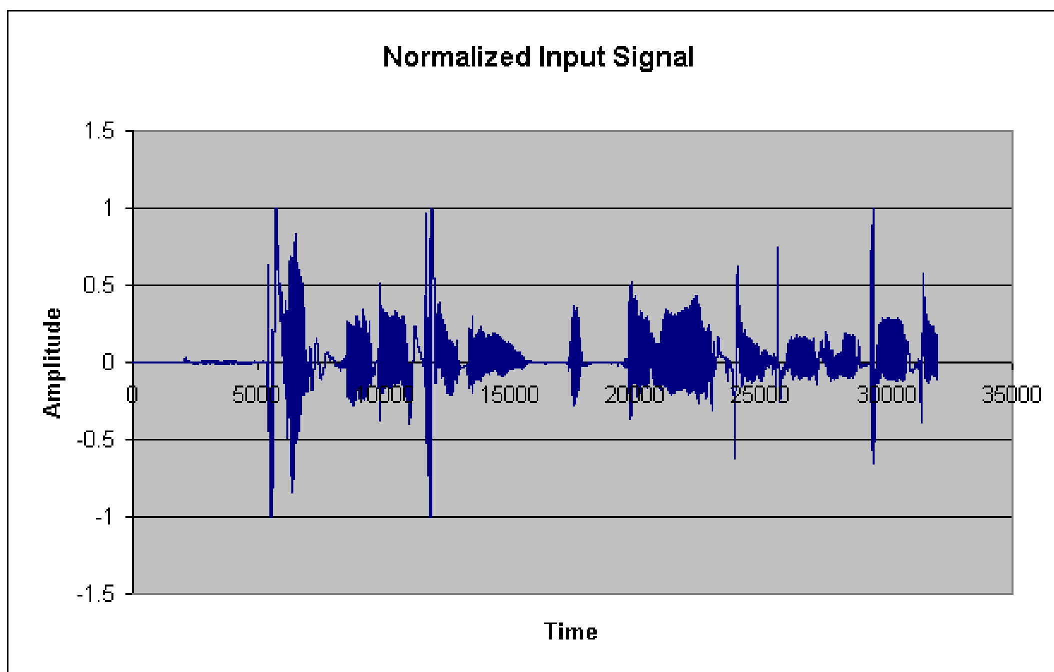

Figure 5.3: Normalization of aihua5.wav from the testing set.

high frequencies, even though they have a lower amplitude. Ideally this boost should be performed via compression, which automatically boosts the quieter sounds while maintaining the amplitude of the louder sounds. However, we have simply done this using a positive value for the filter's frequency response. The low-pass filter (Section 5.3.4) is used as a simplified noise reducer, simply cutting off all frequencies above a certain point. The human voice does not generate sounds all the way up to 4000 Hz, which is the maximum frequency of our test samples, and therefore since this range will only be filled with noise, it may be better just to cut it out.

Essentially the FFT filter is an implementation of the Overlap-Add method of FIR filter design [Ber05]. The process is a simple way to perform fast convolution, by converting the input to the frequency domain, manipulating the frequencies according to the desired frequency response, and then using an Inverse-FFT to convert back to the time domain. Figure 5.4 demonstrates the normalized incoming wave form translated into the frequency domain.

The code applies the square root of the hamming window to the input windows (which are overlapped by half-windows), applies the FFT, multiplies the results by the desired frequency response, applies the Inverse-FFT, and applies the square root of the hamming window again, to produce an undistorted output.

Another similar filter could be used for noise reduction, subtracting the noise characteristics from the frequency response instead of multiplying, thereby remove the room noise from the input sample.

5.3.4 Low-Pass Filter

The low-pass filter has been realized on top of the FFT Filter, by setting up frequency response to zero for frequencies past certain threshold chosen heuristically based on the window size where to cut off. We

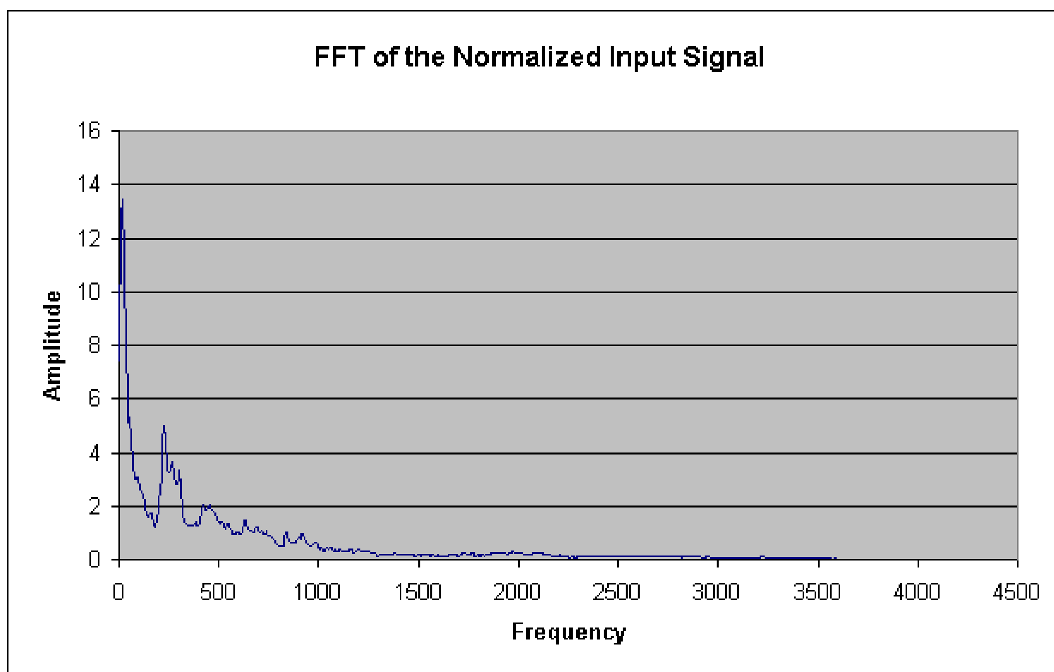

Figure 5.4: FFT of normalized aihua5.wav from the testing set.

filtered out all the frequencies past 2853 Hz.

Figure 5.5 presents an FFT graph of a low-pass filtered signal.

5.3.5 High-Pass Filter

Revision : 1.7

As the low-pass filter, the high-pass filter (e.g. is in Figure 5.6) has been realized on top of the FFT Filter, in fact, it is the opposite to low-pass filter, and filters out frequencies before 2853 Hz. The implementation of the high-pass filter can be found in `marf.Preprocessing.FFTFilter.HighPassFilter`.

5.3.6 Band-Pass Filter

Band-pass filter in MARF is yet another instance of an FFT Filter (Section 5.3.3), with the default settings of the band of frequencies of $[1000, 2853]$ Hz. See Figure 5.7.

5.3.7 High Frequency Boost

Revision : 1.11

This filter was also implemented on top of the FFT filter to boost the high-end frequencies. The frequencies boosted after approx. 1000 Hz by a factor of 5π, heuristically determined, and then re-normalized. See Figure 5.8. The implementation of the high-frequency boost preprocessor can be found in `marf.Preprocessing.FFTFilter.HighFrequencyBoost`.

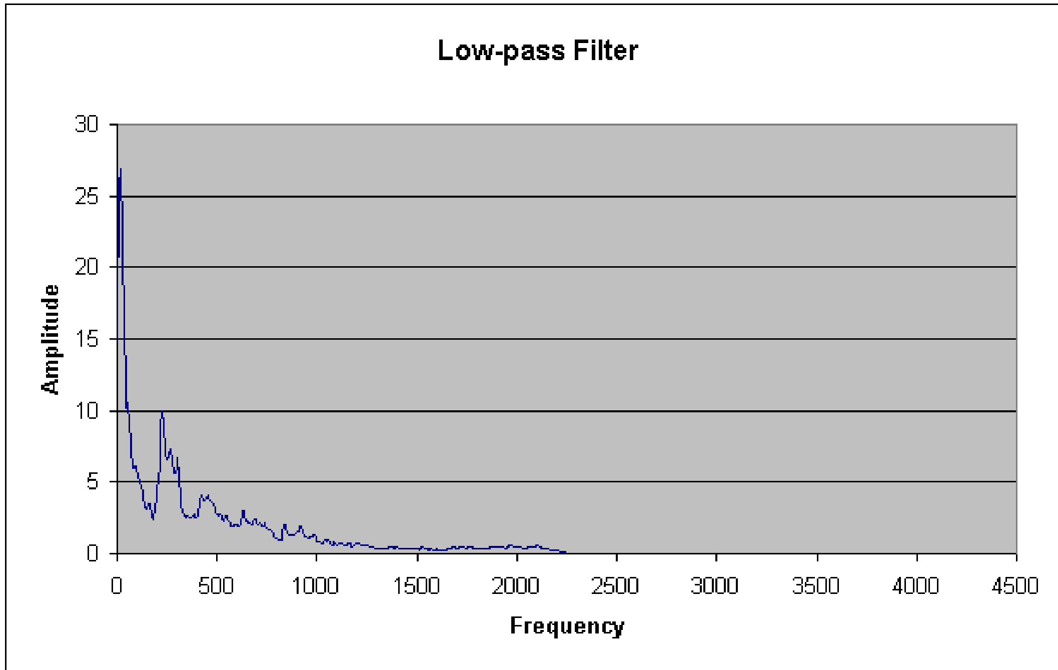

Figure 5.5: Low-pass filter applied to aihua5.wav.

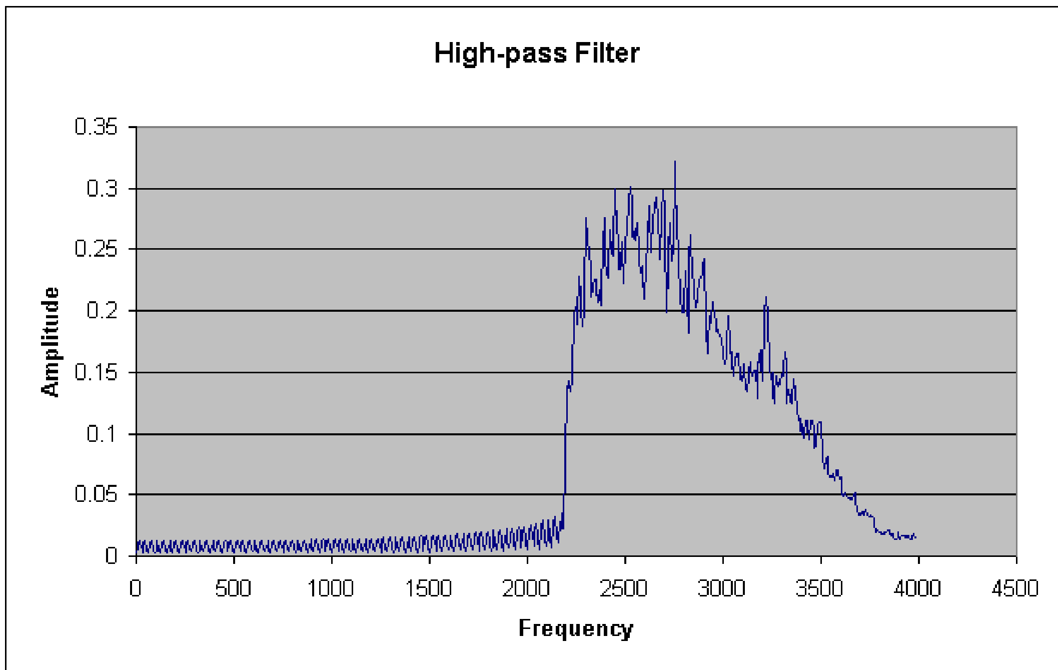

Figure 5.6: High-pass filter applied to aihua5.wav.

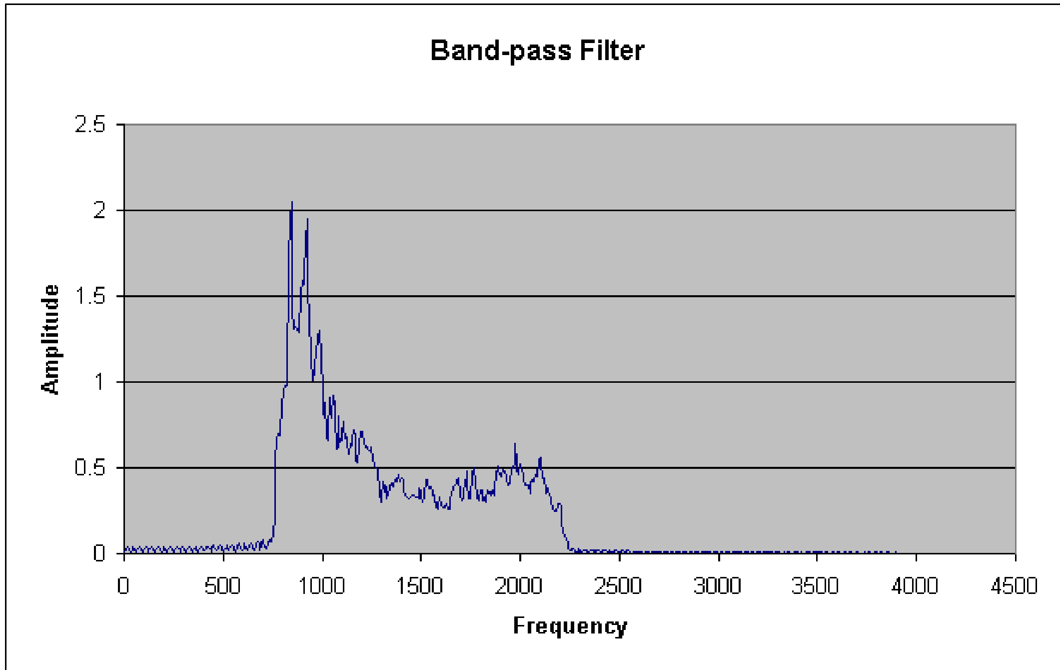

Figure 5.7: Band-pass filter applied to aihua5.wav.

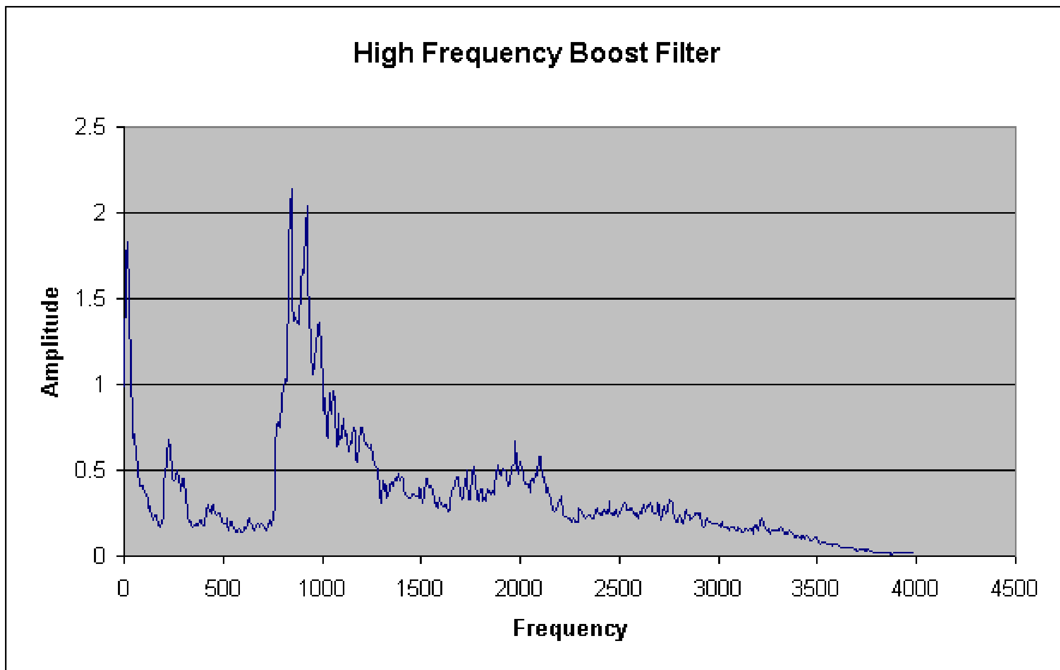

Figure 5.8: High frequency boost filter applied to aihua5.wav.

5.3.8 High-Pass High Frequency Boost Filter

Revision : 1.2

For experimentation we said what would be very useful to do is to test a high-pass filter along with high-frequency boost. While there is no immediate class that does this, `MARF` now chains the former and the latter via the new addition to the preprocessing framework (in 0.3.0-devel-20050606) where a constructor of one preprocessing module takes up another allowing a preprocessing pipeline on its own. The results of this experiment can be found in the Consolidated Results section. While they did not yield a better recognition performance, it was a good try to see. More tweaking and trying is required to make a final decision on this approach as there is an issue with the re-normalization of the entire input instead of just the boosted part.

5.3.9 Noise Removal

Any vocal sample taken in a less-than-perfect (which is always the case) environment will experience a certain amount of room noise. Since background noise exhibits a certain frequency characteristic, if the noise is loud enough it may inhibit good recognition of a voice when the voice is later tested in a different environment. Therefore, it is necessary to remove as much environmental interference as possible.

To remove room noise, it is first necessary to get a sample of the room noise by itself. This sample, usually at least 30 seconds long, should provide the general frequency characteristics of the noise when subjected to FFT analysis. Using a technique similar to overlap-add FFT filtering, room noise can then be removed from the vocal sample by simply subtracting the noise's frequency characteristics from the vocal sample in question.

That is, if $S(x)$ is the sample, $N(x)$ is the noise, and $V(x)$ is the voice, all in the frequency domain, then

$$S(x) = N(x) + V(x)$$

Therefore, it should be possible to isolate the voice:

$$V(x) = S(x) - N(x)$$

Unfortunately, time has not permitted us to implement this in practice yet.

5.4 Feature Extraction

Revision : 1.12

This section outlines feature extraction methods of the MARF project. First we present you with the API and structure, followed by the description of the methods. The class diaram of this module set is in Figure 5.9.

5.4.1 Hamming Window

Revision : 1.13

5.4.1.1 Implementation

The Hamming Window implementation in MARF is in the `marf.math.Algortithms.Hamming` class as of version 0.3.0-devel-20050606 (a.k.a 0.3.0.2).

5.4.1.2 Theory

In many DSP techniques, it is necessary to consider a smaller portion of the entire speech sample rather than attempting to process the entire sample at once. The technique of cutting a sample into smaller pieces to be considered individually is called "windowing". The simplest kind of window to use is the "rectangle", which is simply an unmodified cut from the larger sample.

$$r(t) = \begin{cases} 1 & \text{for } (0 \leq t \leq N-1) \\ 0 & \text{otherwise} \end{cases}$$

Unfortunately, rectangular windows can introduce errors, because near the edges of the window there will potentially be a sudden drop from a high amplitude to nothing, which can produce false "pops" and "clicks" in the analysis.

A better way to window the sample is to slowly fade out toward the edges, by multiplying the points in the window by a "window function". If we take successive windows side by side, with the edges faded out, we will distort our analysis because the sample has been modified by the window function. To avoid this, it is necessary to overlap the windows so that all points in the sample will be considered equally. Ideally, to avoid all distortion, the overlapped window functions should add up to a constant. This is exactly what the Hamming window does. It is defined as:

$$x = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{l-1}\right)$$

where x is the new sample amplitude, n is the index into the window, and l is the total length of the window.

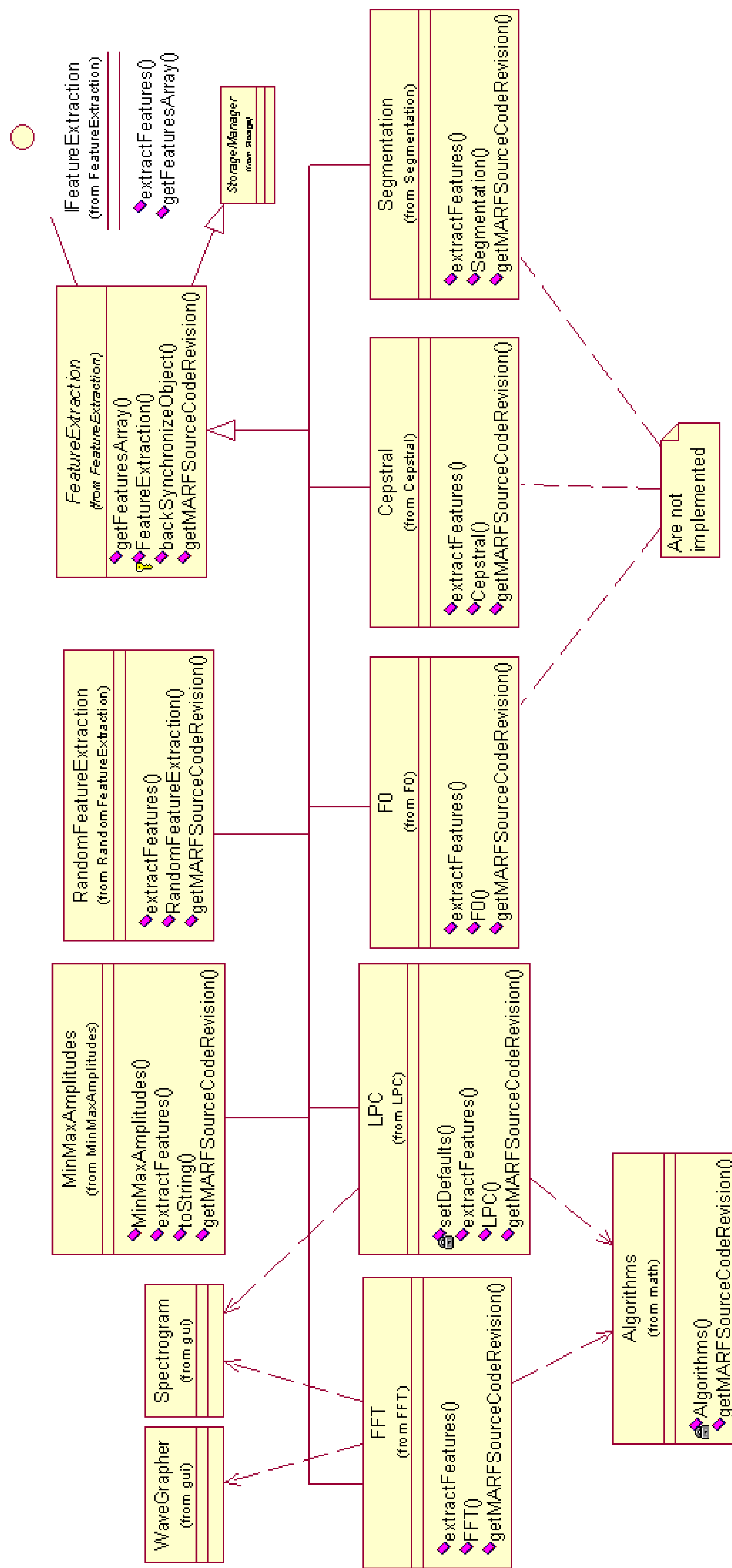

Figure 5.9: Feature Extraction Class Diagram

5.4.2 Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) algorithm is used both for feature extraction and as the basis for the filter algorithm used in preprocessing. Although a complete discussion of the FFT algorithm is beyond the scope of this document, a short description of the implementation will be provided here.

Essentially the FFT is an optimized version of the Discrete Fourier Transform. It takes a window of size 2^k and returns a complex array of coefficients for the corresponding frequency curve. For feature extraction, only the magnitudes of the complex values are used, while the FFT filter operates directly on the complex results.

The implementation involves two steps: First, shuffling the input positions by a binary reversion process, and then combining the results via a "butterfly" decimation in time to produce the final frequency coefficients. The first step corresponds to breaking down the time-domain sample of size n into n frequency-domain samples of size 1. The second step re-combines the n samples of size 1 into 1 n-sized frequency-domain sample.

The code used in MARF has been translated from the C code provided in the book, "Numeric Recipes in C", [Pre93].

5.4.2.1 FFT Feature Extraction

The frequency-domain view of a window of a time-domain sample gives us the frequency characteristics of that window. In feature identification, the frequency characteristics of a voice can be considered as a list of "features" for that voice. If we combine all windows of a vocal sample by taking the average between them, we can get the average frequency characteristics of the sample. Subsequently, if we average the frequency characteristics for samples from the same speaker, we are essentially finding the center of the cluster for the speaker's samples. Once all speakers have their cluster centers recorded in the training set, the speaker of an input sample should be identifiable by comparing its frequency analysis with each cluster center by some classification method.

Since we are dealing with speech, greater accuracy should be attainable by comparing corresponding phonemes with each other. That is, "th" in "the" should bear greater similarity to "th" in "this" than will "the" and "this" when compared as a whole.

The only characteristic of the FFT to worry about is the window used as input. Using a normal rectangular window can result in glitches in the frequency analysis because a sudden cutoff of a high frequency may distort the results. Therefore it is necessary to apply a Hamming window to the input sample, and to overlap the windows by half. Since the Hamming window adds up to a constant when overlapped, no distortion is introduced.

When comparing phonemes, a window size of about 2 or 3 ms is appropriate, but when comparing whole words, a window size of about 20 ms is more likely to be useful. A larger window size produces a higher resolution in the frequency analysis.

5.4.3 Linear Predictive Coding (LPC)

This section presents implementation of the LPC Classification module.

One method of feature extraction used in the MARF project was Linear Predictive Coding (LPC) analysis. It evaluates windowed sections of input speech waveforms and determines a set of coefficients approximating the amplitude vs. frequency function. This approximation aims to replicate the results of the Fast Fourier Transform yet only store a limited amount of information: that which is most valuable to the analysis of speech.

5.4.3.1 Theory

The LPC method is based on the formation of a spectral shaping filter, $H(z)$, that, when applied to a input excitation source, $U(z)$, yields a speech sample similar to the initial signal. The excitation source, $U(z)$, is assumed to be a flat spectrum leaving all the useful information in $H(z)$. The model of shaping filter used in most LPC implementation is called an "all-pole" model, and is as follows:

$$H(z) = \frac{G}{\left(1 - \sum_{k=1}^{p}(a_k z^{-k})\right)}$$

Where p is the number of poles used. A pole is a root of the denominator in the Laplace transform of the input-to-output representation of the speech signal.

The coefficients a_k are the final representation if the speech waveform. To obtain these coefficients, the least-square autocorrelation method was used. This method requires the use of the autocorrelation of a signal defined as:

$$R(k) = \sum_{m=k}^{n-1}(x(n) \cdot x(n-k))$$

where $x(n)$ is the windowed input signal.

In the LPC analysis, the error in the approximation is used to derive the algorithm. The error at time n can be expressed in the following manner: $e(n) = s(n) - \sum_{k=1}^{p}(a_k \cdot s(n-k))$. Thusly, the complete squared error of the spectral shaping filter $H(z)$ is:

$$E = \sum_{n=-\infty}^{\infty}\left(x(n) - \sum_{k=1}^{p}(a_k \cdot x(n-k))\right)$$

To minimize the error, the partial derivative $\frac{\delta E}{\delta a_k}$ is taken for each $k = 1..p$, which yields p linear equations of the form:

$$\sum_{n=-\infty}^{\infty}(x(n-i) \cdot x(n)) = \sum_{k=1}^{p}(a_k \cdot \sum_{n=-\infty}^{\infty}(x(n-i) \cdot x(n-k))$$

For $i = 1..p$. Which, using the autocorrelation function, is:

$$\sum_{k=1}^{p}(a_k \cdot R(i-k)) = R(i)$$

Solving these as a set of linear equations and observing that the matrix of autocorrelation values is a Toeplitz matrix yields the following recursive algorithm for determining the LPC coefficients:

$$k_m = \frac{\left(R(m) - \sum_{k=1}^{m-1}(a_{m-1}(k)R(m-k))\right)}{E_{m-1}}$$

$$a_m(m) = k_m$$

$$a_m(k) = a_{m-1}(k) - k_m \cdot a_m(m-k) \text{ for } 1 \leq k \leq m-1,$$

$$E_m = (1 - k_m^2) \cdot E_{m-1}$$

.

This is the algorithm implemented in the MARF LPC module.

5.4.3.2 Usage for Feature Extraction

The LPC coefficients were evaluated at each windowed iteration, yielding a vector of coefficient of size p. These coefficients were averaged across the whole signal to give a mean coefficient vector representing the utterance. Thus a p sized vector was used for training and testing. The value of p chosen was based on tests given speed vs. accuracy. A p value of around 20 was observed to be accurate and computationally feasible.

5.4.4 F0: The Fundamental Frequency

Revision : 1.6

[WORK ON THIS SECTION IS IN PROGRESS AS WE PROCEED WITH F0 IMPLEMENTATION IN MARF]

F0, the fundamental frequency, or "pitch".

Ian: "The text ([O'S00]) doesn't go into too much detail but gives a few techniques. Most seem to involve another preprocessing to remove high frequencies and then some estimation and postprocessing correction. Another, more detailed source may be needed."

Serguei: "One of the prerequisites we already have: the low-pass filter that does remove the high frequencies."

5.4.5 Min/Max Amplitudes

Revision : 1.4

5.4.5.1 Description

The Min/Max Amplitudes extraction simply involves picking up X maximums and N minimums out of the sample as features. If the length of the sample is less than $X + N$, the difference is filled in with the middle element of the sample.

TODO: This feature extraction does not perform very well yet in any configuration because of the simplistic implementation: the sample amplitudes are sorted and N minimums and X maximums are picked up from both ends of the array. As the samples are usually large, the values in each group are really close if not identical making it hard for any of the classifiers to properly discriminate the subjects. The future improvements here will include attempts to pick up values in N and X distinct enough to be features and for the samples smaller than the $X + N$ sum, use increments of the difference of smallest maximum and largest minimum divided among missing elements in the middle instead one the same value filling that space in.

5.4.6 Random Feature Extraction

By default given a window of size 256 samples, it picks at random a number from a Gaussian distribution, and multiplies by the incoming sample frequencies. This all adds up and we have a feature vector at the end. This should be the bottom line performance of all feature extraction methods. It can also be used as a relatively fast testing module.

5.5 Classification

Revision : 1.13

This section outlines classification methods of the MARF project. First, we present you with the API and overall structure, followed by the description of the methods. Overall structure of the modules is in Figure 5.10.

5.5.1 Chebyshev Distance

Chebyshev distance is used along with other distance classifiers for comparison. Chebyshev distance is also known as a city-block or Manhattan distance. Here's its mathematical representation:

$$d(x,y) = \sum_{k=1}^{n}(|x_k - y_k|)$$

where x and y are feature vectors of the same length n.

5.5.2 Euclidean Distance

The Euclidean Distance classifier uses an Euclidean distance equation to find the distance between two feature vectors.

If $A = (x_1, x_2)$ and $B = (y_1, y_2)$ are two 2-dimensional vectors, then the distance between A and B can be defined as the square root of the sum of the squares of their differences:

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

This equation can be generalized to n-dimensional vectors by simply adding terms under the square root.

$$d(x,y) = \sqrt{(x_n - y_n)^2 + (x_{n-1} - y_{n-1})^2 + ... + (x_1 - y_1)^2}$$

or

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$$

or

$$d(x,y) = \sqrt{(x - y)^T (x - y)}$$

A cluster is chosen based on smallest distance to the feature vector in question.

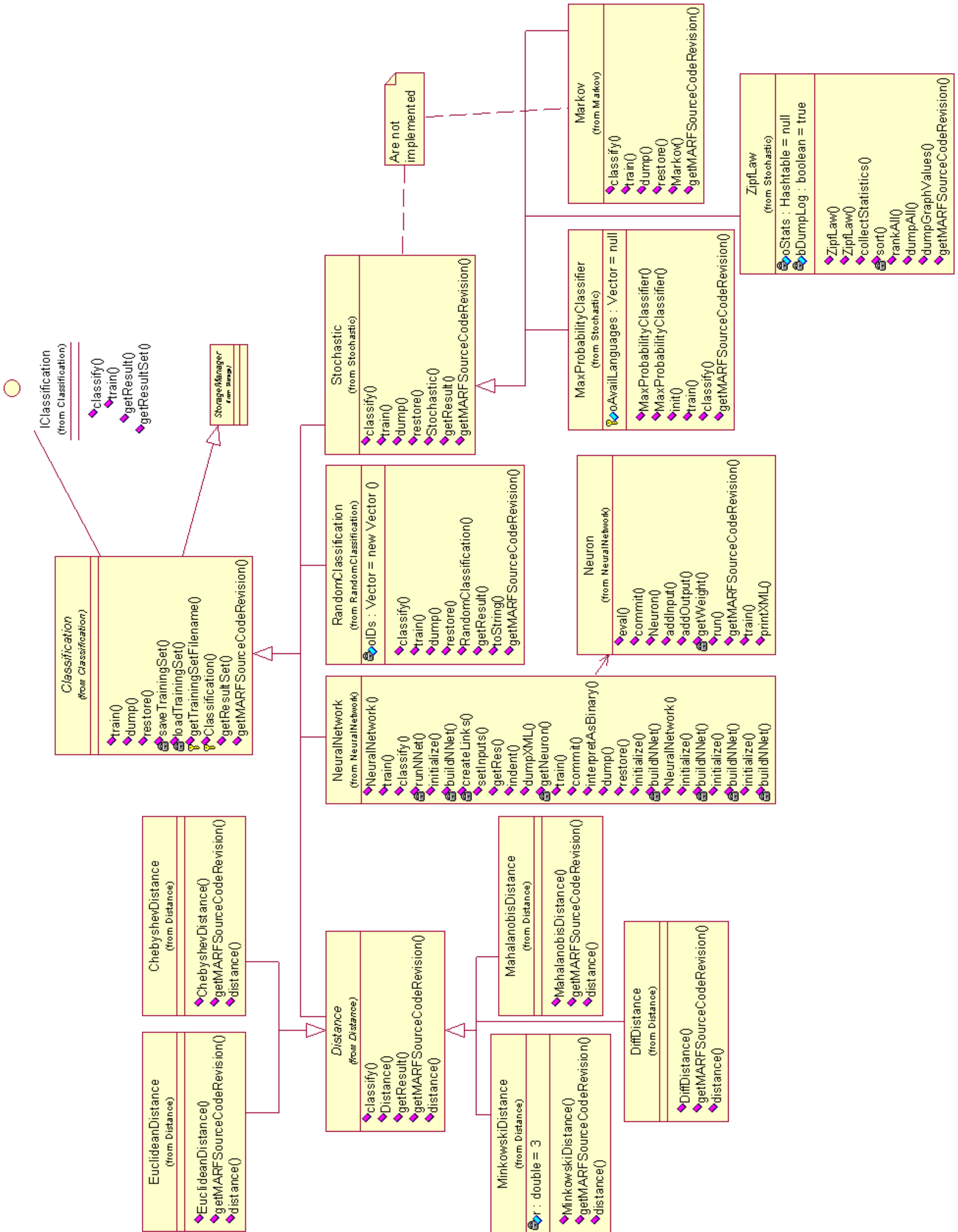

Figure 5.10: Classification

5.5.3 Minkowski Distance

Minkowski distance measurement is a generalization of both Euclidean and Chebyshev distances.

$$d(x,y) = \left(\sum_{k=1}^{n}(|x_k - y_k|)^r\right)^{\frac{1}{r}}$$

where r is a Minkowski factor. When $r = 1$, it becomes Chebyshev distance, and when $r = 2$, it is the Euclidean one. x and y are feature vectors of the same length n.

5.5.4 Mahalanobis Distance

$Revision : 1.7$

5.5.4.1 Summary

- Implementation: `marf.Classification.Distance.MahalanobisDistance`
- Depends on: `marf.Classification.Distance.Distance`
- Used by: `test`, `marf.MARF`, `SpeakerIdentApp`

5.5.4.2 Theory

This distance classification is meant to be able to detect features that tend to vary together in the same cluster if linear transformations are applied to them, so it becomes invariant from these transformations unlike all the other, previously seen distance classifiers.

$$d(x,y) = \sqrt{(x-y)C^{-1}(x-y)^T}$$

where x and y are feature vectors of the same length n, and C is a covariance matrix, learnt during training for co-related features.

In this release, namely 0.3.0-devel, the covariance matrix being an identity matrix, $C = I$, making Mahalanobis distance be the same as the Euclidean one. We need to complete the learning of the covariance matrix to complete this classifier.

5.5.5 Diff Distance

$Revision : 1.2$

5.5.5.1 Summary

- Implementation: `marf.Classification.Distance.DiffDistance`
- Depends on: `marf.Classification.Distance.Distance`
- Used by: `test`, `marf.MARF`, `SpeakerIdentApp`

5.5.5.2 Theory

When Serguei Mokhov invented this classifier in May 2005, the original idea was based on the way the `diff` UNIX utility works. Later, for performance enhancements it was modified. The essence of the diff distance is to count how one input vector is different from the other in terms of elements correspondence. If the Chebyshev distance between the two corresponding elements is greater than some error e, then this distance is accounted for plus some additional distance penalty p is added. Both factors e and p can vary depending on desired configuration. If the two elements are equal or pretty close (the difference is less than e) then a small "bonus" of e is subtracted from the distance.

$$d(x, y) = \sum_i |x_i - y_i| + p, if |x_i - y_i| > e, or(-e)$$

where x and y are feature vectors of the same length.

5.5.6 Artificial Neural Network

This section presents implementation of the Neural Network Classification module.

One method of classification used is an Artificial Neural Network. Such a network is meant to represent the neuronal organization in organisms. Its use as a classification method lies is in the training of the network to output a certain value given a particular input [RN95].

5.5.6.1 Theory

A neuron consists of a set of inputs with associated weights, a threshold, an activation function ($f(x)$) and an output value. The output value will propagate to further neurons (as input values) in the case where the neuron is not part of the "output" layer of the network. The relation of the inputs to the activation function is as follows:

$output \longleftarrow f(in)$

where $in = \sum_{i=0}^{n}(w_i \cdot a_i) - t$, "vector" a is the input activations, "vector" w is the associated weights and t is the threshold of the network. The following activation function was used:

$sigmoid(x; c) = \frac{1}{(1+e^{-cx})}$

where c is a constant. The advantage of this function is that it is differentiable over the region $(-\infty, +\infty)$ and has derivative:

$\frac{d(sigmoid(x;c))}{dx} = c \cdot sigmoid(x;c) \cdot (1 - sigmoid(x;c))$

The structure of the network used was a Feed-Forward Neural Network. This implies that the neurons are organized in sets, representing layers, and that a neuron in layer j, has inputs from layer $j-1$ and output to layer $j+1$ only. This structure facilitates the evaluation and the training of a network. For instance, in the evaluation of a network on an input vector I, the output of neuron in the first layer is calculated, followed by the second layer, and so on.

5.5.6.2 Training

Training in a Feed-Forward Neural Network is done through the an algorithm called Back-Propagation Learning. It is based on the error of the final result of the network. The error the propagated backward throughout the network, based on the amount the neuron contributed to the error. It is defined as follows:

$w_{i,j} \longleftarrow \beta w_{i,j} + \alpha \cdot a_j \cdot \Delta_i$

where

$\Delta_i = Err_i \cdot \frac{df}{dx(in_i)}$ for neuron i in the output layer

and

$\Delta_i = \frac{df}{dt(in_i)} \cdot \sum_{j=0}^{n} (\Delta_j)$ for neurons in other layers

The parameters α and β are used to avoid local minima in the training optimization process. They weight the combination of the old weight with the addition of the new change. Usual values for these are determined experimentally.

The Back-Propagation training method was used in conjunction with epoch training. Given a set of training input vectors Tr, the Back-Propagation training is done on each run. However, the new weight vectors for each neuron, "vector" w', are stored and not used. After all the inputs in Tr have been trained, the new weights are committed and a set of test input vectors Te, are run, and a mean error is calculated. This mean error determines whether to continue epoch training or not.

5.5.6.3 Usage as a Classifier

As a classifier, a Neural Network is used to map feature vectors to speaker identifiers. The neurons in the input layer correspond to each feature in the feature vector. The output of the network is the binary interpretation of the output layer. Therefore the Neural Network has an input layer of size m, where m is the size of all feature vectors and the output layer has size $\lceil(\log_2(n))\rceil$, where n is the maximum speaker identifier.

A network of this structure is trained with the set of input vectors corresponding to the set of training samples for each speaker. The network is epoch trained to optimize the results. This fully trained network is then used for classification in the recognition process.

5.5.7 Random Classification

That might sound strange, but we have a random classifier in MARF. This is more or less testing module just to quickly test the PR pipeline. It picks an ID in the pseudo-random manner from the list of trained IDs of subjects to classification. It also serves as a bottom-line of performance (i.e. recognition rate) for all the other, slightly more sophisticated classification methods meaning performance of the aforementioned methods must be better than that of the Random; otherwise, there is a problem.

Chapter 6

Natural Language Processing (NLP)

Revision : 1.1

This chapter will describe the NLP facilities now present in MARF. This includes:

1. Probabilistic Parsing of English
2. Stemming
3. Collocations
4. Related N-gram Models, Zipf's Law and Maximum Probability Classifiers, Statistical Estimators and Smoothing.

TODO

Chapter 7

GUI

Revision : 1.6

[UPDATE ME]

Even though this section is entitled as GUI, we don't really have too much GUI yet (it's planned though, see TODO, E). We do have a couple of things under the `marf.gui` package, which we do occasionally use and eventually they will expand to be a real GUI classes. This tiny package is in Figure 7.1.

TODO

7.1 Spectrogram

Revision : 1.4

Sometimes it is useful to visualize the data we are playing with. One of the typical thing when dealing with sounds, specifically voice, people are interested in spectrograms of frequency distributions. The `Spectrogram` class was designed to handle that and produce spectrograms from both FFT and LPC algorithms and simply draw them. We did not manage to make it a true GUI component yet, but instead we made it to dump the spectrograms into PPM-format image files to be looked at using some graphical package. Two examples of such spectrograms are in the Appendix A.

We are just taking all the `Output[]` for the spectrogram. It's supposed to be only half ([O'S00]). We took a hamming window of the waveform at 1/2 intervals of 128 samples (ie: 8 kHz, 16 ms). By half intervals we mean that the second half of the window was the first half of the next. O'Shaughnessy in [O'S00] says this is a good way to use the window. Thus, any streaming of waveform must consider this.

What we did for both the FFT spectrogram and LPC determination was to multiply the signal by the window and do a `doFFT()` or a `doLPC()` coefficient determination on the resulting array of N windowed samples. This gave us an approximation of the stable signal at $s(i \cdot N/2)$. Or course, we will have to experiment with windows and see which one is better, but there may be no definitive best.

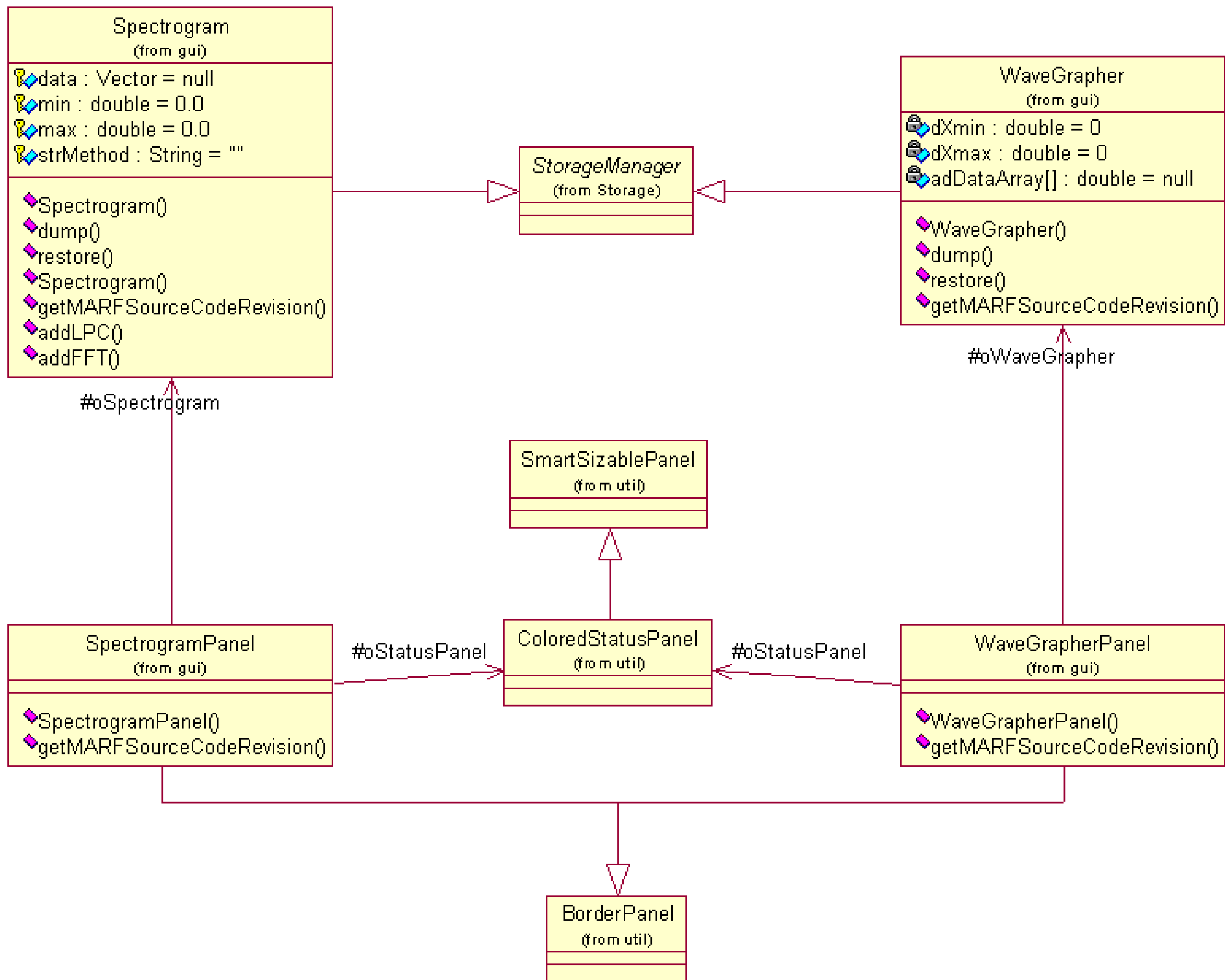

Figure 7.1: GUI Package

7.2 Wave Grapher

Revision : 1.3

`WaveGrapher` is another class designed, as the name suggests, to draw the wave form of the incoming/preprocessed signal. Well, it doesn't actually draw a thing, but dumps the sample points into a tab-delimited text file to be loaded into some plotting software, such as `gnuplot` or `Excel`. We also use it to produce graphs of the signal in the frequency domain istead of time domain. Examples of the graphs of data obtained via this class are in the Preprocessing Section (5.3).

Chapter 8

Sample Data and Experimentation

Revision : 1.16

8.1 Sample Data

Revision : 1.15

We have both female and male speakers, with age ranging from a college student to a University professor. The table 8.1 has a list of people who have contributed their voice samples for our project (with the first four being ourselves). We want to thank them once again for helping us out.

8.2 Comparison Setup

The main idea was to compare combinations (in MARF: *configurations*) of different methods and variations within them in terms of recognition rate performance. That means that having several preprocessing modules, several feature extraction modules, and several classification modules, we can (and did) try all their possible combinations.

That includes:

1. Preprocessing: No-filtering, normalization, low-pass, high-pass, band-pass, and high-frequency boost, high-pass and boost filters.
2. Feature Extraction: FFT/LPC/Min-Max/Random algorithms comparison
3. Classification: Distance classifiers, such as Chebyshev, Euclidean, Minkowski, Mahalanobis, and Diff distances, as well as Neural Network and Random classification.

For this purpose we have written a `SpeakerIdentApp`, a command-line application for TI speaker identification. We ran it for every possible configuration with the following script, namely `testing.sh`:

ID	Name	Training Samples	Testing Samples
1	Serge	14	1
2	Ian	14	1
3	Steve	12	3
4	Jimmy	14	1
5	Dr. C.Y. Suen	2	1
6	Margarita Mokhova	14	1
7	Alexei Mokhov	14	1
8	Alexandr Mokhov	14	1
9	Graham Sinclair	12	2
10	Jihed Halimi	2	1
11	Madhumita Banerjee	3	1
13	Irina Dymova	3	1
14	Aihua Wu	14	1
15	Nick	9	1
16	Michelle Khalife	14	1
17	Shabana	7	1
18	Van Halen	8	1
19	RHCP	8	1
20	Talal Al-Khoury	14	1
21	Ke Gong	14	1
22	Emily Wu Rong	14	1
23	Emily Ying Lu	14	1
24	Shaozhen Fang	14	1
25	Chunlei He	14	1
26	Shuxin Fan	15	1
Total	**25**	**279**	**28**

Table 8.1: Speakers contributed their voice samples.

```
#!/bin/tcsh -f

#
# Batch Processing of Training/Testing Samples
# NOTE: Make take quite some time to execute
#
# Copyright (C) 2002-2005 The MARF Development Group
#
# $Header: /cvsroot/marf/apps/SpeakerIdentApp/testing.sh,v 1.27 2005/05/22 15:35:37 mokhov Exp $
#

#
# Set environment variables, if needed
#

setenv CLASSPATH .:marf.jar
setenv EXTDIRS

#
# Set flags to use in the batch execution
#

set java = 'java'
#set debug = '-debug'
set debug = ''
set graph = ''
#set graph = '-graph'
#set spectrogram = '-spectrogram'
set spectrogram = ''

if($1 == '--reset') then
    echo "Resetting Stats..."
    $java SpeakerIdentApp --reset
    exit 0
endif

if($1 == '--retrain') then
    echo "Training..."

    # Always reset stats before retraining the whole thing
    $java SpeakerIdentApp --reset

    foreach prep (-norm -boost -low -high -band -highpassboost -raw)
        foreach feat (-fft -lpc -randfe -minmax)

            # Here we specify which classification modules to use for
            # training. Since Neural Net wasn't working the default
            # distance training was performed; now we need to distinguish them
            # here. NOTE: for distance classifiers it's not important
            # which exactly it is, because the one of generic Distance is used.
            # Exception for this rule is Mahalanobis Distance, which needs
            # to learn its Covariance Matrix.

            foreach class (-cheb -mah -randcl -nn)
                echo "Config: $prep $feat $class $spectrogram $graph $debug"
                date
```

```
                # XXX: We cannot cope gracefully right now with these combinations --- too many
                # links in the fully-connected NNet, so run out of memory quite often; hence,
                # skip it for now.
                if("$class" == "-nn" && ("$feat" == "-fft" || "$feat" == "-randfe")) then
                    echo "skipping..."
                    continue
                endif

                time $java SpeakerIdentApp --train training-samples $prep $feat $class $spectrogram $graph $debug
            end

        end
    end

endif

echo "Testing..."

foreach file (testing-samples/*.wav)
    foreach prep (-norm -boost -low -high -band -highpassboost -raw)
        foreach feat (-fft -lpc -randfe -minmax)
            foreach class (-eucl -cheb -mink -mah -diff -randcl -nn)
                echo "=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-="
                echo "DOING FILE:"
                echo $file
                echo "Config: $prep $feat $class $spectrogram $graph $debug"
                date
                echo "============================================="

                # XXX: We cannot cope gracefully right now with these combinations --- too many
                # links in the fully-connected NNet, so run of memeory quite often, hence
                # skip it for now.
                if("$class" == "-nn" && ("$feat" == "-fft" || "$feat" == "-randfe")) then
                    echo "skipping..."
                    continue
                endif

                time $java SpeakerIdentApp --ident $file $prep $feat $class $spectrogram $graph $debug

                echo "---------------------------------------------"
            end
        end
    end
end

echo "Stats:"

$java SpeakerIdentApp --stats | tee stats.txt
$java SpeakerIdentApp --best-score | tee best-score.tex
date | tee stats-date.tex

echo "Testing Done"

exit 0

# EOF
```

See the results section (9) for results analysis.

8.3 What Else Could/Should/Will Be Done

There is a lot more that we realistically could do, but due to lack of time, these things are not in yet. If you would like to contribute, let us know, meanwhile we'll keep working at our speed.

8.3.1 Combination of Feature Extraction Methods

For example, assuming we use a combination of LPC coefficients and F0 estimation, we could compare the results of different combinations of these, and discuss them later. Same with the Neural Nets (modifying number of layers and number or neurons, etc.).

We could also do a 1024 FFT analysis and compare it against a 128 FFT analysis. (That is, the size of the resulting feature vector would be 512 or 64 respectively). With LPC, one can specify the number of coefficients you want, the more you have the more precise the analysis will be.

8.3.2 Entire Recognition Path

The `LPC` module is used to generate a mean vector of LPC coefficients for the utterance. `F0` is used to find the average fundamental frequency of the utterance. The results are concatenated to form the output vector, in a particular order. The classifier would take into account the weighting of the features: Neural Network would do so implicitly if it benefits the speaker matching, and stochastic can be modified to give more weight to the F0 or vice versa, depending on what we see best (i.e.: the covariance matrix in the Mahalanobis distance (5.5.4)).

8.3.3 More Methods

Things like F0, Endpointing, Stochastic, and some other methods have not made to this release. More detailed on this aspect, please refer to the TODO list in the Appendix.

Chapter 9

Experimentation Results

Revision : 1.17

9.1 Notes

Before we get to numbers, few notes and observations first:

1. We've got more samples since the demo. The obvious: by increasing the number of samples our results got better; with few exceptions, however. This can be explained by the diversity of the recording equipment, a lot less than uniform number of samples per speaker, and absence of noise and silence removal. All the samples were recorded in not the same environments. The results then start averaging after awhile.

2. Another observation we made from our output, is that when the speaker is guessed incorrectly, quite often the second guess is correct, so we included this in our results as if we were "guessing" right from the second attempt.

3. FUN. Interesting to note, that we also tried to take some samples of music bands, and feed it to our application along with the speakers, and application's performance didn't suffer, yet even improved because the samples were treated in the same manner. The groups were not mentioned in the table, so we name them here: Van Halen [8:1] and Red Hot Chili Peppers [10:1] (where numbers represent [training:testing] samples used).

9.2 Configuration Explained

Configuration parameters were extracted from the command line which `SpeakerIdentApp` was invoked with. They mean the following:

```
Usage:
    java SpeakerIdentApp --train <samples-dir> [options]  -- train mode
                         --ident <sample> [options]       -- identification mode
                         --stats                          -- display stats
                         --reset                          -- reset stats
                         --version                        -- display version info
                         --help | -h                      -- display this help and exit

Options (one or more of the following):

Preprocessing:

  -raw          - no preprocessing
  -norm         - use just normalization, no filtering
  -low          - use low-pass filter
  -high         - use high-pass filter
  -boost        - use high-frequency-boost preprocessor
  -band         - use band-pass filter

Feature Extraction:

  -lpc          - use LPC
  -fft          - use FFT
  -minmax       - use Min/Max Amplitudes
  -randfe       - use random feature extraction

Classification:

  -nn           - use Neural Network
  -cheb         - use Chebyshev Distance
  -eucl         - use Euclidean Distance
  -mink         - use Minkowski Distance
  -diff         - use Diff-Distance
  -randcl       - use random classification

Misc:

  -debug        - include verbose debug output
  -spectrogram  - dump spectrogram image after feature extraction
  -graph        - dump wave graph before preprocessing and after feature extraction
```

```
  <integer>     - expected speaker ID
```

9.3 Consolidated Results

Our ultimate results [1] for all configurations we can have and samples we've got are below. Looks like our best results are with "-raw -fft -mink", "-norm -fft -diff", "-raw -fft -eucl" and, "-raw -fft -mah" with the top result being around 75.00 % (see Table 9.1).

[1] as of Sat Jun 25 19:23:09 EDT 2005

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
1st	1	-raw -fft -mah	21	7	75.00
1st	2	-norm -fft -diff	21	7	75.00
1st	3	-raw -fft -eucl	21	7	75.00
1st	4	-raw -fft -mink	21	7	75.00
1st	5	-norm -fft -eucl	20	8	71.43
1st	6	-norm -fft -mah	20	8	71.43
1st	7	-norm -fft -cheb	20	8	71.43
1st	8	-raw -lpc -mah	19	9	67.86
1st	9	-boost -fft -mah	19	9	67.86
1st	10	-raw -fft -cheb	19	9	67.86
1st	11	-norm -lpc -cheb	19	9	67.86
1st	12	-boost -lpc -mink	19	9	67.86
1st	13	-norm -lpc -eucl	19	9	67.86
1st	14	-raw -lpc -cheb	19	9	67.86
1st	15	-norm -fft -mink	19	9	67.86
1st	16	-norm -lpc -diff	19	9	67.86
1st	17	-raw -lpc -eucl	19	9	67.86
1st	18	-raw -lpc -diff	19	9	67.86
1st	19	-norm -lpc -mah	19	9	67.86
1st	20	-boost -fft -eucl	19	9	67.86
1st	21	-low -fft -mah	18	10	64.29
1st	22	-boost -lpc -mah	18	10	64.29
1st	23	-boost -lpc -eucl	18	10	64.29
1st	24	-low -fft -cheb	18	10	64.29
1st	25	-low -fft -eucl	18	10	64.29
1st	26	-low -fft -diff	18	10	64.29
1st	27	-boost -fft -cheb	18	10	64.29
1st	28	-norm -lpc -mink	18	10	64.29
1st	29	-raw -lpc -mink	18	10	64.29

Table 9.1: Consolidated results, Part 1.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
1st	30	-boost -fft -diff	18	10	64.29
1st	31	-low -lpc -cheb	17	11	60.71
1st	32	-high -fft -mah	17	11	60.71
1st	33	-raw -fft -diff	17	11	60.71
1st	34	-high -fft -eucl	17	11	60.71
1st	35	-low -fft -mink	17	11	60.71
1st	36	-high -fft -mink	17	11	60.71
1st	37	-boost -lpc -cheb	17	11	60.71
1st	38	-boost -fft -mink	16	12	57.14
1st	39	-high -fft -cheb	16	12	57.14
1st	40	-norm -lpc -nn	16	12	57.14
1st	41	-low -lpc -mah	15	13	53.57
1st	42	-boost -lpc -diff	15	13	53.57
1st	43	-low -lpc -eucl	15	13	53.57
1st	44	-low -lpc -diff	15	13	53.57
1st	45	-highpassboost -lpc -cheb	14	14	50.00
1st	46	-high -lpc -cheb	14	14	50.00
1st	47	-low -lpc -mink	14	14	50.00
1st	48	-highpassboost -fft -diff	13	15	46.43
1st	49	-highpassboost -lpc -mink	13	15	46.43
1st	50	-band -fft -diff	13	15	46.43
1st	51	-band -lpc -mink	12	16	42.86
1st	52	-high -lpc -mah	12	16	42.86
1st	53	-highpassboost -lpc -eucl	12	16	42.86
1st	54	-raw -lpc -nn	12	16	42.86
1st	55	-highpassboost -lpc -diff	12	16	42.86
1st	56	-high -lpc -eucl	12	16	42.86
1st	57	-band -fft -eucl	12	16	42.86
1st	58	-band -lpc -mah	12	16	42.86
1st	59	-band -fft -mah	12	16	42.86

Table 9.2: Consolidated results, Part 2.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
1st	60	-highpassboost -lpc -mah	12	16	42.86
1st	61	-band -lpc -cheb	12	16	42.86
1st	62	-band -lpc -eucl	12	16	42.86
1st	63	-highpassboost -fft -mink	11	17	39.29
1st	64	-high -fft -diff	11	17	39.29
1st	65	-band -lpc -diff	11	17	39.29
1st	66	-high -lpc -diff	10	18	35.71
1st	67	-band -fft -cheb	10	18	35.71
1st	68	-high -lpc -mink	10	18	35.71
1st	69	-highpassboost -fft -cheb	10	18	35.71
1st	70	-boost -lpc -nn	10	18	35.71
1st	71	-highpassboost -fft -eucl	9	19	32.14
1st	72	-raw -minmax -mah	9	19	32.14
1st	73	-raw -minmax -mink	9	19	32.14
1st	74	-highpassboost -fft -mah	9	19	32.14
1st	75	-raw -minmax -eucl	9	19	32.14
1st	76	-raw -minmax -cheb	8	20	28.57
1st	77	-band -fft -mink	7	21	25.00
1st	78	-norm -randfe -eucl	6	22	21.43
1st	79	-low -randfe -mink	6	22	21.43
1st	80	-norm -minmax -eucl	6	22	21.43
1st	81	-norm -randfe -mah	6	22	21.43
1st	82	-norm -randfe -mink	6	22	21.43
1st	83	-norm -minmax -mah	6	22	21.43
1st	84	-raw -minmax -nn	6	22	21.43
1st	85	-raw -minmax -diff	6	22	21.43
1st	86	-norm -minmax -cheb	6	22	21.43
1st	87	-high -minmax -eucl	5	23	17.86
1st	88	-raw -randfe -mink	5	23	17.86
1st	89	-high -minmax -mah	5	23	17.86

Table 9.3: Consolidated results, Part 3.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
1st	90	-high -minmax -diff	5	23	17.86
1st	91	-low -minmax -mink	5	23	17.86
1st	92	-boost -randfe -cheb	5	23	17.86
1st	93	-boost -randfe -diff	5	23	17.86
1st	94	-low -randfe -eucl	5	23	17.86
1st	95	-low -randfe -mah	5	23	17.86
1st	96	-high -randfe -eucl	4	24	14.29
1st	97	-low -lpc -nn	4	24	14.29
1st	98	-high -randfe -diff	4	24	14.29
1st	99	-low -minmax -mah	4	24	14.29
1st	100	-norm -randfe -diff	4	24	14.29
1st	101	-boost -randfe -eucl	4	24	14.29
1st	102	-high -randfe -mink	4	24	14.29
1st	103	-high -minmax -mink	4	24	14.29
1st	104	-norm -minmax -mink	4	24	14.29
1st	105	-low -randfe -cheb	4	24	14.29
1st	106	-low -minmax -cheb	4	24	14.29
1st	107	-raw -randfe -eucl	4	24	14.29
1st	108	-raw -randfe -mah	4	24	14.29
1st	109	-high -randfe -cheb	4	24	14.29
1st	110	-low -randfe -diff	4	24	14.29
1st	111	-low -minmax -eucl	4	24	14.29
1st	112	-high -randfe -mah	4	24	14.29
1st	113	-norm -randfe -cheb	4	24	14.29
1st	114	-high -minmax -cheb	4	24	14.29
1st	115	-low -minmax -diff	4	24	14.29
1st	116	-boost -randfe -mah	4	24	14.29
1st	117	-band -minmax -mink	3	25	10.71
1st	118	-high -minmax -nn	3	25	10.71
1st	119	-band -lpc -randcl	3	25	10.71

Table 9.4: Consolidated results, Part 4.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
1st	120	-low -minmax -nn	3	25	10.71
1st	121	-norm -minmax -diff	3	25	10.71
1st	122	-boost -minmax -cheb	3	25	10.71
1st	123	-boost -minmax -mah	3	25	10.71
1st	124	-band -lpc -nn	3	25	10.71
1st	125	-norm -minmax -nn	3	25	10.71
1st	126	-boost -minmax -eucl	3	25	10.71
1st	127	-band -minmax -mah	3	25	10.71
1st	128	-band -minmax -cheb	3	25	10.71
1st	129	-highpassboost -randfe -diff	3	25	10.71
1st	130	-boost -randfe -mink	3	25	10.71
1st	131	-highpassboost -lpc -nn	3	25	10.71
1st	132	-boost -minmax -mink	3	25	10.71
1st	133	-raw -randfe -cheb	3	25	10.71
1st	134	-band -minmax -eucl	3	25	10.71
1st	135	-band -minmax -diff	3	25	10.71
1st	136	-highpassboost -minmax -mink	3	25	10.71
1st	137	-highpassboost -randfe -randcl	3	25	10.71
1st	138	-raw -randfe -diff	3	25	10.71
1st	139	-high -lpc -nn	3	25	10.71
1st	140	-raw -minmax -randcl	2	26	7.14
1st	141	-high -fft -randcl	2	26	7.14
1st	142	-highpassboost -minmax -mah	2	26	7.14
1st	143	-band -randfe -mah	2	26	7.14
1st	144	-low -lpc -randcl	2	26	7.14
1st	145	-highpassboost -randfe -cheb	2	26	7.14
1st	146	-boost -minmax -diff	2	26	7.14
1st	147	-highpassboost -randfe -mah	2	26	7.14
1st	148	-band -fft -randcl	2	26	7.14
1st	149	-band -randfe -cheb	2	26	7.14

Table 9.5: Consolidated results, Part 5.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
1st	150	-highpassboost -randfe -eucl	2	26	7.14
1st	151	-highpassboost -minmax -eucl	2	26	7.14
1st	152	-highpassboost -minmax -diff	2	26	7.14
1st	153	-band -randfe -eucl	2	26	7.14
1st	154	-band -randfe -diff	2	26	7.14
1st	155	-norm -lpc -randcl	2	26	7.14
1st	156	-high -randfe -randcl	2	26	7.14
1st	157	-boost -minmax -nn	2	26	7.14
1st	158	-raw -lpc -randcl	1	27	3.57
1st	159	-band -minmax -nn	1	27	3.57
1st	160	-high -lpc -randcl	1	27	3.57
1st	161	-raw -randfe -randcl	1	27	3.57
1st	162	-highpassboost -minmax -cheb	1	27	3.57
1st	163	-highpassboost -minmax -randcl	1	27	3.57
1st	164	-boost -minmax -randcl	1	27	3.57
1st	165	-raw -fft -randcl	1	27	3.57
1st	166	-low -fft -randcl	1	27	3.57
1st	167	-low -randfe -randcl	1	27	3.57
1st	168	-highpassboost -fft -randcl	1	27	3.57
1st	169	-boost -fft -randcl	1	27	3.57
1st	170	-highpassboost -randfe -mink	1	27	3.57
1st	171	-band -randfe -randcl	1	27	3.57
1st	172	-low -minmax -randcl	1	27	3.57
1st	173	-highpassboost -lpc -randcl	0	28	0.00
1st	174	-boost -lpc -randcl	0	28	0.00
1st	175	-high -minmax -randcl	0	28	0.00
1st	176	-highpassboost -minmax -nn	0	28	0.00
1st	177	-band -minmax -randcl	0	28	0.00
1st	178	-norm -fft -randcl	0	28	0.00
1st	179	-norm -minmax -randcl	0	28	0.00

Table 9.6: Consolidated results, Part 6.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
1st	180	-boost -randfe -randcl	0	28	0.00
1st	181	-band -randfe -mink	0	28	0.00
1st	182	-norm -randfe -randcl	0	28	0.00
2nd	1	-raw -fft -mah	23	5	82.14
2nd	2	-norm -fft -diff	23	5	82.14
2nd	3	-raw -fft -eucl	23	5	82.14
2nd	4	-raw -fft -mink	24	4	85.71
2nd	5	-norm -fft -eucl	23	5	82.14
2nd	6	-norm -fft -mah	23	5	82.14
2nd	7	-norm -fft -cheb	23	5	82.14
2nd	8	-raw -lpc -mah	22	6	78.57
2nd	9	-boost -fft -mah	22	6	78.57
2nd	10	-raw -fft -cheb	21	7	75.00
2nd	11	-norm -lpc -cheb	22	6	78.57
2nd	12	-boost -lpc -mink	20	8	71.43
2nd	13	-norm -lpc -eucl	22	6	78.57
2nd	14	-raw -lpc -cheb	22	6	78.57
2nd	15	-norm -fft -mink	23	5	82.14
2nd	16	-norm -lpc -diff	22	6	78.57
2nd	17	-raw -lpc -eucl	22	6	78.57
2nd	18	-raw -lpc -diff	22	6	78.57
2nd	19	-norm -lpc -mah	22	6	78.57
2nd	20	-boost -fft -eucl	22	6	78.57
2nd	21	-low -fft -mah	22	6	78.57
2nd	22	-boost -lpc -mah	21	7	75.00
2nd	23	-boost -lpc -eucl	21	7	75.00
2nd	24	-low -fft -cheb	21	7	75.00
2nd	25	-low -fft -eucl	22	6	78.57
2nd	26	-low -fft -diff	20	8	71.43
2nd	27	-boost -fft -cheb	20	8	71.43

Table 9.7: Consolidated results, Part 7.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
2nd	28	-norm -lpc -mink	23	5	82.14
2nd	29	-raw -lpc -mink	23	5	82.14
2nd	30	-boost -fft -diff	20	8	71.43
2nd	31	-low -lpc -cheb	21	7	75.00
2nd	32	-high -fft -mah	21	7	75.00
2nd	33	-raw -fft -diff	21	7	75.00
2nd	34	-high -fft -eucl	21	7	75.00
2nd	35	-low -fft -mink	23	5	82.14
2nd	36	-high -fft -mink	19	9	67.86
2nd	37	-boost -lpc -cheb	20	8	71.43
2nd	38	-boost -fft -mink	20	8	71.43
2nd	39	-high -fft -cheb	20	8	71.43
2nd	40	-norm -lpc -nn	18	10	64.29
2nd	41	-low -lpc -mah	20	8	71.43
2nd	42	-boost -lpc -diff	20	8	71.43
2nd	43	-low -lpc -eucl	20	8	71.43
2nd	44	-low -lpc -diff	20	8	71.43
2nd	45	-highpassboost -lpc -cheb	18	10	64.29
2nd	46	-high -lpc -cheb	19	9	67.86
2nd	47	-low -lpc -mink	18	10	64.29
2nd	48	-highpassboost -fft -diff	17	11	60.71
2nd	49	-highpassboost -lpc -mink	16	12	57.14
2nd	50	-band -fft -diff	14	14	50.00
2nd	51	-band -lpc -mink	16	12	57.14
2nd	52	-high -lpc -mah	17	11	60.71
2nd	53	-highpassboost -lpc -eucl	18	10	64.29
2nd	54	-raw -lpc -nn	15	13	53.57
2nd	55	-highpassboost -lpc -diff	18	10	64.29
2nd	56	-high -lpc -eucl	17	11	60.71
2nd	57	-band -fft -eucl	15	13	53.57

Table 9.8: Consolidated results, Part 8.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
2nd	58	-band -lpc -mah	16	12	57.14
2nd	59	-band -fft -mah	15	13	53.57
2nd	60	-highpassboost -lpc -mah	18	10	64.29
2nd	61	-band -lpc -cheb	17	11	60.71
2nd	62	-band -lpc -eucl	16	12	57.14
2nd	63	-highpassboost -fft -mink	14	14	50.00
2nd	64	-high -fft -diff	18	10	64.29
2nd	65	-band -lpc -diff	18	10	64.29
2nd	66	-high -lpc -diff	19	9	67.86
2nd	67	-band -fft -cheb	14	14	50.00
2nd	68	-high -lpc -mink	14	14	50.00
2nd	69	-highpassboost -fft -cheb	15	13	53.57
2nd	70	-boost -lpc -nn	11	17	39.29
2nd	71	-highpassboost -fft -eucl	13	15	46.43
2nd	72	-raw -minmax -mah	11	17	39.29
2nd	73	-raw -minmax -mink	11	17	39.29
2nd	74	-highpassboost -fft -mah	13	15	46.43
2nd	75	-raw -minmax -eucl	11	17	39.29
2nd	76	-raw -minmax -cheb	10	18	35.71
2nd	77	-band -fft -mink	13	15	46.43
2nd	78	-norm -randfe -eucl	10	18	35.71
2nd	79	-low -randfe -mink	9	19	32.14
2nd	80	-norm -minmax -eucl	8	20	28.57
2nd	81	-norm -randfe -mah	10	18	35.71
2nd	82	-norm -randfe -mink	10	18	35.71
2nd	83	-norm -minmax -mah	8	20	28.57
2nd	84	-raw -minmax -nn	8	20	28.57
2nd	85	-raw -minmax -diff	10	18	35.71
2nd	86	-norm -minmax -cheb	9	19	32.14
2nd	87	-high -minmax -eucl	6	22	21.43

Table 9.9: Consolidated results, Part 9.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
2nd	88	-raw -randfe -mink	9	19	32.14
2nd	89	-high -minmax -mah	6	22	21.43
2nd	90	-high -minmax -diff	6	22	21.43
2nd	91	-low -minmax -mink	7	21	25.00
2nd	92	-boost -randfe -cheb	8	20	28.57
2nd	93	-boost -randfe -diff	9	19	32.14
2nd	94	-low -randfe -eucl	9	19	32.14
2nd	95	-low -randfe -mah	9	19	32.14
2nd	96	-high -randfe -eucl	6	22	21.43
2nd	97	-low -lpc -nn	6	22	21.43
2nd	98	-high -randfe -diff	7	21	25.00
2nd	99	-low -minmax -mah	5	23	17.86
2nd	100	-norm -randfe -diff	10	18	35.71
2nd	101	-boost -randfe -eucl	8	20	28.57
2nd	102	-high -randfe -mink	5	23	17.86
2nd	103	-high -minmax -mink	6	22	21.43
2nd	104	-norm -minmax -mink	9	19	32.14
2nd	105	-low -randfe -cheb	9	19	32.14
2nd	106	-low -minmax -cheb	5	23	17.86
2nd	107	-raw -randfe -eucl	8	20	28.57
2nd	108	-raw -randfe -mah	8	20	28.57
2nd	109	-high -randfe -cheb	7	21	25.00
2nd	110	-low -randfe -diff	9	19	32.14
2nd	111	-low -minmax -eucl	5	23	17.86
2nd	112	-high -randfe -mah	6	22	21.43
2nd	113	-norm -randfe -cheb	10	18	35.71
2nd	114	-high -minmax -cheb	5	23	17.86
2nd	115	-low -minmax -diff	5	23	17.86
2nd	116	-boost -randfe -mah	8	20	28.57
2nd	117	-band -minmax -mink	5	23	17.86

Table 9.10: Consolidated results, Part 10.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
2nd	118	-high -minmax -nn	3	25	10.71
2nd	119	-band -lpc -randcl	3	25	10.71
2nd	120	-low -minmax -nn	3	25	10.71
2nd	121	-norm -minmax -diff	8	20	28.57
2nd	122	-boost -minmax -cheb	5	23	17.86
2nd	123	-boost -minmax -mah	5	23	17.86
2nd	124	-band -lpc -nn	4	24	14.29
2nd	125	-norm -minmax -nn	4	24	14.29
2nd	126	-boost -minmax -eucl	5	23	17.86
2nd	127	-band -minmax -mah	5	23	17.86
2nd	128	-band -minmax -cheb	5	23	17.86
2nd	129	-highpassboost -randfe -diff	5	23	17.86
2nd	130	-boost -randfe -mink	5	23	17.86
2nd	131	-highpassboost -lpc -nn	4	24	14.29
2nd	132	-boost -minmax -mink	7	21	25.00
2nd	133	-raw -randfe -cheb	6	22	21.43
2nd	134	-band -minmax -eucl	5	23	17.86
2nd	135	-band -minmax -diff	6	22	21.43
2nd	136	-highpassboost -minmax -mink	6	22	21.43
2nd	137	-highpassboost -randfe -randcl	5	23	17.86
2nd	138	-raw -randfe -diff	6	22	21.43
2nd	139	-high -lpc -nn	5	23	17.86
2nd	140	-raw -minmax -randcl	3	25	10.71
2nd	141	-high -fft -randcl	2	26	7.14
2nd	142	-highpassboost -minmax -mah	6	22	21.43
2nd	143	-band -randfe -mah	5	23	17.86
2nd	144	-low -lpc -randcl	3	25	10.71
2nd	145	-highpassboost -randfe -cheb	5	23	17.86
2nd	146	-boost -minmax -diff	6	22	21.43
2nd	147	-highpassboost -randfe -mah	4	24	14.29

Table 9.11: Consolidated results, Part 11.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
2nd	148	-band -fft -randcl	3	25	10.71
2nd	149	-band -randfe -cheb	4	24	14.29
2nd	150	-highpassboost -randfe -eucl	4	24	14.29
2nd	151	-highpassboost -minmax -eucl	6	22	21.43
2nd	152	-highpassboost -minmax -diff	3	25	10.71
2nd	153	-band -randfe -eucl	5	23	17.86
2nd	154	-band -randfe -diff	4	24	14.29
2nd	155	-norm -lpc -randcl	3	25	10.71
2nd	156	-high -randfe -randcl	2	26	7.14
2nd	157	-boost -minmax -nn	3	25	10.71
2nd	158	-raw -lpc -randcl	5	23	17.86
2nd	159	-band -minmax -nn	3	25	10.71
2nd	160	-high -lpc -randcl	1	27	3.57
2nd	161	-raw -randfe -randcl	1	27	3.57
2nd	162	-highpassboost -minmax -cheb	5	23	17.86
2nd	163	-highpassboost -minmax -randcl	2	26	7.14
2nd	164	-boost -minmax -randcl	2	26	7.14
2nd	165	-raw -fft -randcl	1	27	3.57
2nd	166	-low -fft -randcl	1	27	3.57
2nd	167	-low -randfe -randcl	3	25	10.71
2nd	168	-highpassboost -fft -randcl	3	25	10.71
2nd	169	-boost -fft -randcl	2	26	7.14
2nd	170	-highpassboost -randfe -mink	2	26	7.14
2nd	171	-band -randfe -randcl	2	26	7.14
2nd	172	-low -minmax -randcl	3	25	10.71
2nd	173	-highpassboost -lpc -randcl	2	26	7.14
2nd	174	-boost -lpc -randcl	2	26	7.14
2nd	175	-high -minmax -randcl	1	27	3.57
2nd	176	-highpassboost -minmax -nn	1	27	3.57
2nd	177	-band -minmax -randcl	1	27	3.57

Table 9.12: Consolidated results, Part 12.

Guess	Run #	Configuration	GOOD	BAD	Recognition Rate,%
2nd	178	-norm -fft -randcl	1	27	3.57
2nd	179	-norm -minmax -randcl	2	26	7.14
2nd	180	-boost -randfe -randcl	1	27	3.57
2nd	181	-band -randfe -mink	1	27	3.57
2nd	182	-norm -randfe -randcl	5	23	17.86

Table 9.13: Consolidated results, Part 13.

Chapter 10

Applications

Revision : 1.12

This chapter describes the applications that employ MARF for one purpose or another. Most of them are our own applications that demonstrate how to use various MARF's features. Others are either research or internal projects of their own. If you use MARF or its derivative in your application and would like to be listed here, please let us know at `marf-devel@sf.lists.net`.

10.1 MARF Research Applications

10.1.1 SpeakerIdentApp - Text-Independent Speaker Identification Application

`SpeakerIdentApp` is an application for text-independent speaker identification that exercises the most of the MARF's features. `SpeakerIdentApp` is broadly presented through the rest of this manual.

10.2 MARF Testing Applications

10.2.1 TestFilters

Revision : 1.5

`TestFilters` is one of the testing applications in MARF. It tests how four types of FFT-filter-based preprocessors work with simulated or real wave type sound samples. From user's point of view, `TestFilters` provides with usage, preprocessors, and loaders command-line options. By entering `--help`, or `-h`, or when there are no arguments, the usage information will be displayed. It explains the arguments used in `TestFilters`. The first argument is the type of preprocessor/filter. These are high-pass filter (`--high`), low-pass filter (`--low`); band-pass filter (`--band`); high frequency boost preprocessor (`--boost`) to be chosen. Next argument is the type of loader to use to load initial testing sample. `TestFilters` uses two types of loaders, `SineLoader` and `WAVLoader`. Users should enter either `--sine` or `--wave` as the second argument to specify the desired loader. The argument `--sine` uses `SineLoader` that will generate a plain

sine wave to be fed to the selected preprocessor. While the `--wave` argument uses `WAVLoader` to load a real wave sound sample. In the latter case, users need to input the third argument – sample file in the WAV format to feed to `WAVLoader`. After selecting all necessary arguments, user can run and get the output of `TestFilters` within seconds.

The application is made to exercise the following MARF modules:

The main are the FFT-based filters in the `marf.Preprocessing.FFTFilter.*` package.

1. `BandpassFilter`
2. `HighFrequencyBoost`
3. `HighPassFilter`
4. `LowPassFilter`

Additionally, some other units were employed in this application:

1. `marf.MARF`
2. `marf.util.OptionProcessor`
3. `marf.Storage.Sample`
4. `marf.Storage.SampleLoader`
5. `marf.Storage.WAVLoader`
6. `marf.Preprocessing.Preprocessing`

The main `MARF` module enumerates these preprocessing modules as `HIGH_FREQUENCY_BOOST_FFT_FILTER`, `BANDPASS_FFT_FILTER`, `LOW_PASS_FFT_FILTER`, and `HIGH_PASS_FFT_FILTER`, and incoming sample file format as `WAV`, `SINE`. `OptionProcessor` helps maintaining and validating command-line options. `Sample` maintains incoming and processed sample data. `SampleLoader` provides sample loading interface for all the MARF loaders. It must be overridden by a concrete sample loader such as `SineLoader` or `WAVLoader`. `Preprocessing` does general preprocessing such as `preprocess()` (overridden by the filters), `normalize()`, `removeNoise()` and `removeSilence()` out of which for this application the former two are used. In the end, above modules work together to test the work of the filters and produce the output to STDOUT. The output of `TestFilters` is the filtered data from the original signal fed to each of the preprocessors. It provides both users and programmers internal information of the effect of MARF preprocessors so they can be compared with the expected output in the `expected` folder to detect any errors if the underlying algorithm has been changed.

10.2.2 TestLPC

Revision : 1.4

The `TestLPC` application targets the LPC unit testing as a preprocessing module. Through a number of options it also allows choosing between two implemented loaders – `WAVLoader` and `SineLoader`. To faciliate option processing `marf.util.OptionProcessor` used that provides an ability of maintaining and validating valid/active option sets. The application also utilizes the `Dummy` preprocessing module to perform the normalization of incoming sample.

The application supports the following set of options:

- `--help` or `-h` cause the application to display the usage information and exit. The usage information is also displayed if no option was supplied.
- `--sine` forces the use of `SineLoader` for sample data generation. The output of this option is also saved under `expected/sine.out` for regression testing.
- `--wave` forces the application to use the `WAVLoader`. This option requires a mandatory filename argument of a wave file to run the LPC algorithm against.

10.2.3 TestFFT

Revision : 1.4

`TestFFT` is one of the testing applications in MARF. It aims to test how the FFT (Fast Fourier Transform) algorithm works in MARFFeatureExtraction by loading simulated or real wave sound samples.

From user's point of view, `TestFFT` provides with usage and loaders command-line options. By entering `--help`, or `-h`, or even no arguments, the usage information will be displayed. It explains the arguments used in `TestFFT`. Another argument is the type of loader. `TestFFT` uses two types of loaders, `Sineloader` and `WAVLoader`. Users can enter `--sine` or `--wave` as the second argument. The argument `--sine` uses `SineLoader` that will generate a plain sine wave to be fed to a generic preprocessor. If the argument `--wave` is specified, the application uses `WAVLoader` to load a wave sample from file. In the latter case, users need to supply one more argument – sample file in the format of *.wav to be loaded by `WAVLoader`. After selecting all necessary arguments, user can run and get the output of `TestFFT` within seconds.

The application is made to exercise the MARF's FFT-based feature extraction algorithm located in the `marf.FeatureExtraction.FFT` package. Additionally, the following MARF modules are utilized:

1. `marf.MARF`
2. `marf.util.OptionProcessor`
3. `marf.Storage.Sample`
4. `marf.Storage.SampleLoader`

5. `marf.Storage.WAVLoader`
6. `marf.Preprocessing.Dummy`

The main `MARF` module enumerates incoming sample file format as `WAV`, `SINE`. `OptionProcessor` helps maintain and validate command-line options. `Sample` maintains and processed incoming sample data. `SampleLoader` provides sample loading interface for all the MARF loaders. It must be overridden by a concrete sample loader such as `SineLoader` or `WAVLoader`. `Preprocessing` does general preprocessing such with `preprocess()` (overridden by `Dummy`), `normalize()`, `removeNoise()` and `removeSilence()` out of which for this application the former two are used. Then, processed data will be serve for an parameter of `FeatureExtraction.FFT.FFT` to extract sample's features. In the end, above modules work together to test the work of the FFT algorithm and produce the output to STDOUT.

As we know, the output of `TestFFT` extracts the features data by FFT feature extraction algorithm. It gives both users and programmers direct information of the effect of MARF feature extraction, and can be compared with the expected output in the `expected` folder to detect any errors if the underlying algorithm has been changed.

10.2.4 TestWaveLoader

Revision : 1.4

`TestWaveLoader` is one of the testing applications in MARF. It tests functionality of the `WAVLoader` of MARF.

From user's point of view, `TestWaveLoader` provides with usage, input wave sample, output wave sample, and output textual file command-line options. By entering `--help`, or `-h`, or when there are no arguments, the usage information will be displayed. It explains the arguments used in `TestWaveLoader`. The first argument is an input wave sample file whose name is a mandatory argument. The second and the third arguments are the output wave sample file and output textual sample file of the loaded input sample. The names of the output files are optional arguments. If user does not provide any or both of the last two arguments, the output files will be saved in the files provided by `TestWaveLoader`.

The application is made to exercise the following MARF modules. The main module is the `WAVLoader` in the `marf.Storage.Loaders` package. the `Sample` and the `SampleLoader` modules in the `marf.Storage` help `WAVLoader` prepare loading wave files. `Sample` maintains and processes incoming sample data. `SampleLoader` provides sample loading interface for all the MARF loaders. It must be overridden by a concrete sample loader. For loading wave samples, `SampleLoader` needs `WAVLoader` implementation. Three modules work together to load in and write back a wave sample whose name was provided by users in the first argument, to save the loaded sample into a newly-named wave file, to save loaded input data into a data file referenced by `oDatFile`, and to output sample's duration and size to STDOUT.

As we know, the output of `TestWaveLoader` saves the loaded wave sample in a newly named output wave file. Its output also saves the input file data into a textual file. `TestWaveLoader` gives both users and programmers direct information of the results of MARF wave loader. The output sample can be compared with the expected output in the `expected` folder to detect any errors.

10.2.5 TestLoaders

Revision : 1.3

`TestLoaders` is another testing applications of MARF. It generalizes the testing machinery of `TestWaveLoader` for all possible loaders we may have. For now, it tests functionality of the `WAVLoader` and `SineLoader` of MARF, the only two implemented loaders; the others give non-implemented exceptions instead.

From user's point of view, `TestLoaders` provides with usage, loader types, input sample, output sample, and output textual file command-line options. By entering `--help`, or `-h`, or when there are no arguments, the usage information will be displayed. Entering `--version`, the `TestLoaders`' version and MARF's version is displayed. The usage info explains the arguments used in `TestLoaders`. The first argument is a type of loader that is a mandatory argument. The second argument is input file name that is mandatory for all loaders except for `SineLoader`. The third and forth arguments are the output wave sample file and output textual sample file names of the loaded input sample. The names of the output files are optional arguments. If user does not provide any or both of the last two arguments, the output files will be saved in the file names derived from the original.

The application is made to exercise the following MARF modules. The main modules for testing are in the `marf.Storage.Loaders` package. The `OptionProcessor` module in the `marf.util` helps handling the different loader types according to the users input argument. The `Sample` and the `SampleLoader` modules in the `marf.Storage` help `WAVLoader` prepare loading input files. `Sample` maintains and processes incoming sample data. `SampleLoader` provides sample loading interface for all the MARF loaders. It must be overridden by a concrete sample loader. The modules work together to load in and write back a sample, and save the loaded sample into a file, to save loaded input data into a data file referenced by `oDatFile`, and to output sample's duration and size to STDOUT. While for loading sine samples, it needs `SineLoader` implementation, and instead of saving data file, it saves a csv file referenced by `oDatFile`.

The output of `TestLoaders` saves the loaded wave sample in a newly named output wave file. Its output also saves the input file data into a textual file. `TestLoaders` gives both users and programmers direct information of the results of MARF loaders. Input sample can be compared with the expected output in the `expected` folder to detect any errors.

10.2.6 MathTestApp

Revision : 1.2

The `MathTestApp` application targets testing of the math-related implementations in the `marf.math` package. As of this writing, the application mainy exercises the `Matrix` class as this is the one used by the `MahalanobisDistance` classifier. It also does the necessary testing of the `Vector` class as well.

The application supports the following set of options:

- `--help` or `-h` cause the application to display the usage information and exit.
- `--version` displays the application's and the unrelying MARF's versions and exits.

Before running tests, the application validates the MARF version, and then produces the output to STDOUT of various linear matrix operations, such as inversion, identity, multiplication, transposal, scaling, rotation, translation, and shear. The stored output in the `expected` folder, `math.out`, contains expected output the authors believe is correct and which is used in the regression testing.

10.3 External Applications

10.3.1 GIPSY

GIPSY stands for General Intensional Programming System [RG05], which is a distributed research, development, and investigation platform about intensional and hybrid programming languages. GIPSY is a world on its own developed at Concordia University and for more information on the system please see [RG05] and [Mok05]. GIPSY makes use of a variety of MARF's utility and storage modules, as of this writing this includes:

- `BaseThread` for most threading part of the GIPSY for multithreaded compilation and execution.
- `ExpandedThreadGroup` to group a number of compiler or executor threads and have a group control over them.
- `Debug` for, well, debugging
- `Logger` to log compiler and executor activities
- `StorageManager` for binary serialization
- `OptionProcessor` for option processing in five core GIPSY utilities
- `Arrays` for common arrays functionality

There is a provision to also use MARF's NLP modules.

10.3.2 ENCSAssetsDesktop

This is a private little application developed for the Engineering and Computer Science faculty of Concordia University's Faculty Information Systems team. The application synchronize inventory data between a Palm device database and a relational database powering an online inventory application. This application primarily exercises:

- `BaseThread` for most threading part of the GIPSY for multithreaded compilation and execution.
- `ExpandedThreadGroup` to group a number of compiler or executor threads and have a group control over them.
- `Arrays` for common arrays functionality
- `ByteUtils` for any-type-to-byte-array-and-back conversion.

10.3.3 ENCSAssets

This is a web frontend to an inventory database developed by the same team as in Section 10.3.2. It primarily exercises the threading and arrays parts of MARF, namely `BaseThread` and `Arrays`.

10.3.4 ENCSAssetsCron

This is a cron-oriented frontend to an inventory database developed by the same team as in Section 10.3.2. It primarily exercises the `OptionProcessor` of MARF.

Chapter 11

Conclusions

Revision : 1.12

11.1 Review of Results

Our best configuration yielded 75.00 % correctness of our work when identifying subjects. Having a total of 27 testing samples (including the two music bands) that means 20 subjects identified correctly out of 27 per run on average.

The main reasons the recognition rate could be that low is due to ununiform sample taking, lack of preprocessing techniques optimizations, lack of noise/silence removal, lack or incomplete sophisticated classification modules (e.g. Stochastic models), and lack the samples themselves to train and test on.

Even though for commercial and University-level research standards 75.00 % recognition rate is considered to be very low as opposed to a required minimum of 95%-97% and above, we think it is still reasonably well provided that this was a school project and is maintained as a hobby now. That still involved a substantial amount of research and findings considering our workload and lack of experience in the area.

11.2 Acknowledgments

We would like to thank Dr. Suen and Mr. Sadri for the course and help provided. This LaTeX documentation was possible due to an excellent introduction of it by Dr. Peter Grogono in [Gro01].

Bibliography

[Ber05] Stefan M. Bernsee. *The DFT "à pied": Mastering The Fourier Transform in One Day*. DSPdimension.com, 1999-2005. `http://www.dspdimension.com/data/html/dftapied.html`.

[Fla97] D. Flanagan. *Java in a Nutshell*. O'Reily & Associates, Inc., second edition, 1997. ISBN 1-56592-262-X.

[Gro01] Peter Grogono. *A LaTeX2e Gallimaufry. Techniques, Tips, and Traps.* Department of Computer Science and Software Engineering, Concordia University, March 2001. `<http://www.cse.concordia.ca/~grogono/documentation.html>`.

[IJ02] Ifeachor and Jervis. *Speech Communications*. Prentice Hall. New Jersey, US., 2002.

[Mic05] Sun Microsystems. *The Java Website*. Sun Microsystems, Inc., 1994-2005. `<http://java.sun.com>`.

[Mok05] Serguei A. Mokhov. Towards Hybrid Intensional Programming with JLucid, Objective Lucid, and General Imperative Compiler Framework in the GIPSY. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, August 2005.

[O'S00] Douglas O'Shaughnessy. *Speech Communications*. IEEE Press. New Jersey, US., 2000.

[Pre93] William H. Press. *Numerical Recipes in C*. Cambridge University Press. Cambridge, UK., second edition, 1993.

[RG05] The GIPSY Research and Development Group. *The GIPSY Project*. Department of Computer Science and Software Engineering, Concordia University, 2002-2005. `http://newton.cs.concordia.ca/~gipsy/`.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall. New Jersey, US., 1995.

Appendix A

Spectrogram Examples

Revision : 1.14

As produced by the `Spectrogram` class.

Figure A.1: LPC spectrogram obtained for ian15.wav

Figure A.2: LPC spectrogram obtained for `graham13.wav`

Appendix B

MARF Source Code

You can download the code from <http://marf.sourceforge.net>, specifically:

- The latest unstable version:
  <http://marf.sourceforge.net/marf.tar.gz>
- Browse code and revision history online:
  <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/marf/>

API documentation in the HTML format can be found in the documentation distribution, or for the latest version please consult: <http://marf.sourceforge.net/api/>. If you want to participate in development, there is a developers version of the API: <http://marf.sourceforge.net/api-dev/>, which includes all the private constructs into the docs as well.

The current development version can also be retrieved via `CVS`. The process outlined in Appendix C.

Appendix C

The CVS Repository

The MARF source code is stored and managed using the CVS code management system at SourceForge. Anonymous CVS is available to pull the CVS code tree from the MARF package to your local machine.

C.1 Getting The Source Via Anonymous CVS

If you would like to keep up with the current sources on a regular basis, you can fetch them from SourceForge's CVS server and then use CVS to retrieve updates from time to time.

- You will need a local copy of CVS (Concurrent Version Control System), which you can get from `<http://www.cvshome.org/>` or any GNU software archive site. There is also WinCVS and CVS mode built in JBulider if you plan to use these products on Win32 platforms.
- Do an initial login to the CVS server:

  ```
  cvs -d:pserver:anonymous@cvs.sf.net:/cvsroot/marf login
  ```

  You will be prompted for a password; just press ENTER. You should only need to do this once, since the password will be saved in `.cvspass` in your home directory.
- Fetch the MARF sources:

  ```
  cvs -z3 -d:pserver:anonymous@cvs.sf.net:/cvsroot/marf co -P marf
  ```

  which installs the MARF sources into a subdirectory `marf` of the directory you are currently in.

  If you'd like to download sample applications which use MARF:

  ```
  cvs -z3 -d:pserver:anonymous@cvs.sf.net:/cvsroot/marf co -P apps
  ```

- Whenever you want to update to the latest CVS sources, `cd` into the `marf` or `apps` subdirectories, and issue

  ```
  cvs -z3 update -d -P
  ```

  This will fetch only the changes since the last time you updated.

- You can save yourself some typing by making a file `.cvsrc` in your home directory that contains

  ```
  cvs -z3
  update -d -P
  ```

  This supplies the `-z3` option to all `cvs` commands, and the `-d` and `-P` options to `cvs update`. Then you just have to say

  ```
  cvs update
  ```

  to update your files.

Appendix D

SpeakerIdentApp and SpeakersIdentDb Source Code

D.1 SpeakerIdentApp.java

```
import java.io.File;

import marf.MARF;
import marf.Storage.ModuleParams;
import marf.Storage.TrainingSet;
import marf.util.Debug;
import marf.util.MARFException;


/**
 * <p>Indentfies a speaker independently of text, based on the MARF framework.</p>
 *
 * <p>$Id: SpeakerIdentApp.java,v 1.42 2005/06/19 04:48:10 mokhov Exp $</p>
 *
 * @author The MARF Development Group.
 * @version 0.3.0
 * @since 0.0.1
 */
public class SpeakerIdentApp
{
    /*
     * ----------------
     * Apps. Versionning
     * ----------------
     */

    /**
     * Current major version of the application.
     */
    public static final int MAJOR_VERSION = 0;

    /**
     * Current minor version of the application.
```

```
     */
    public static final int MINOR_VERSION = 3;

    /**
     * Current revision of the application.
     */
    public static final int REVISION      = 0;

    /**
     * Main body.
     * @param argv command-line arguments
     */
    public static void main(String[] argv)
    {
        SpeakersIdentDb oDB = new SpeakersIdentDb("speakers.txt");

        try
        {
            // Since some new API is always introduced...
            validateVersions();

            /*
             * Database of speakers
             */
            oDB.connect();
            oDB.query();

            /*
             * If supplied in the command line,
             * the system when classifying will output next
             * to the guessed one
             */
            int iExpectedID = -1;

            /*
             * Default MARF setup
             */

            MARF.setPreprocessingMethod(MARF.DUMMY);
            MARF.setFeatureExtractionMethod(MARF.FFT);
            MARF.setClassificationMethod(MARF.EUCLIDEAN_DISTANCE);
            MARF.setDumpSpectrogram(false);
            MARF.setSampleFormat(MARF.WAV);

            Debug.enableDebug(false);

            // Parse extra arguments
            // XXX: maybe it's time to move it to a sep. method
            for(int i = 2; i < argv.length; i++)
            {
                try
                {
                    // Preprocessing

                    if(argv[i].equals("-norm"))
                        MARF.setPreprocessingMethod(MARF.DUMMY);
```

```
else if(argv[i].equals("-boost"))
    MARF.setPreprocessingMethod(MARF.HIGH_FREQUENCY_BOOST_FFT_FILTER);

else if(argv[i].equals("-high"))
    MARF.setPreprocessingMethod(MARF.HIGH_PASS_FFT_FILTER);

else if(argv[i].equals("-low"))
    MARF.setPreprocessingMethod(MARF.LOW_PASS_FFT_FILTER);

else if(argv[i].equals("-band"))
    MARF.setPreprocessingMethod(MARF.BANDPASS_FFT_FILTER);

else if(argv[i].equals("-highpassboost"))
    MARF.setPreprocessingMethod(MARF.HIGH_PASS_BOOST_FILTER);

else if(argv[i].equals("-raw"))
    MARF.setPreprocessingMethod(MARF.RAW);

// Feature Extraction

else if(argv[i].equals("-fft"))
    MARF.setFeatureExtractionMethod(MARF.FFT);

else if(argv[i].equals("-lpc"))
    MARF.setFeatureExtractionMethod(MARF.LPC);

else if(argv[i].equals("-randfe"))
    MARF.setFeatureExtractionMethod(MARF.RANDOM_FEATURE_EXTRACTION);

else if(argv[i].equals("-minmax"))
    MARF.setFeatureExtractionMethod(MARF.MIN_MAX_AMPLITUDES);

// Classification

else if(argv[i].equals("-nn"))
{
    MARF.setClassificationMethod(MARF.NEURAL_NETWORK);

    ModuleParams oParams = new ModuleParams();

    // Dump/Restore Format of the TrainingSet
    oParams.addClassificationParam(new Integer(TrainingSet.DUMP_GZIP_BINARY));

    // Training Constant
    oParams.addClassificationParam(new Double(1.0));

    // Epoch number
    oParams.addClassificationParam(new Integer(10));

    // Min. error
    oParams.addClassificationParam(new Double(4.27));

    MARF.setModuleParams(oParams);
}

else if(argv[i].equals("-eucl"))
    MARF.setClassificationMethod(MARF.EUCLIDEAN_DISTANCE);
```

```
            else if(argv[i].equals("-cheb"))
                MARF.setClassificationMethod(MARF.CHEBYSHEV_DISTANCE);

            else if(argv[i].equals("-mink"))
            {
                MARF.setClassificationMethod(MARF.MINKOWSKI_DISTANCE);

                ModuleParams oParams = new ModuleParams();

                // Dump/Restore Format
                oParams.addClassificationParam(new Integer(TrainingSet.DUMP_GZIP_BINARY));

                // Minkowski Factor
                oParams.addClassificationParam(new Double(6.0));

                MARF.setModuleParams(oParams);
            }

            else if(argv[i].equals("-mah"))
                MARF.setClassificationMethod(MARF.MAHALANOBIS_DISTANCE);

            else if(argv[i].equals("-randcl"))
                MARF.setClassificationMethod(MARF.RANDOM_CLASSIFICATION);

            else if(argv[i].equals("-diff"))
                MARF.setClassificationMethod(MARF.DIFF_DISTANCE);

            // Misc

            else if(argv[i].equals("-spectrogram"))
                MARF.setDumpSpectrogram(true);

            else if(argv[i].equals("-debug"))
                Debug.enableDebug(true);

            else if(argv[i].equals("-graph"))
                MARF.setDumpWaveGraph(true);

            else if(Integer.parseInt(argv[i]) > 0)
                iExpectedID = Integer.parseInt(argv[i]);
        }
        catch(NumberFormatException e)
        {
            // Number format exception should be ignored
            // XXX [SM]: Why?

            Debug.debug("SpeakerIdentApp.main() - NumberFormatException: " + e.getMessage());
        }
    } // extra args


    /*
     * --------------
     * Identification
     * --------------
     */
```

```
if(argv[0].equals("--ident"))
{
    /*
     * If no expected speaker present on the command line,
     * attempt to fetch it from the database by filename.
     */
    if(iExpectedID < 0)
        iExpectedID = oDB.getIDByFilename(argv[1], false);

    // Store config and error/sucesses for that config
    String strConfig = "";

    if(argv.length > 2)
        // Get config from the command line
        for(int i = 2; i < argv.length; i++)
            strConfig += argv[i] + " ";

    else
        // Query MARF for it's current config
        strConfig = MARF.getConfig();

    MARF.setSampleFile(argv[1]);
    MARF.recognize();

    int iIdentifiedID = MARF.queryResultID();

    // Second best
    int iSecondClosestID = MARF.getResultSet().getSecondClosestID();

    System.out.println("               Config: " + strConfig);
    System.out.println("         Speaker's ID: " + iIdentifiedID);
    System.out.println("   Speaker identified: " + oDB.getName(iIdentifiedID));

    /*
     * Only collect stats if we have expected speaker
     */
    if(iExpectedID > 0)
    {
        System.out.println("Expected Speaker's ID: " + iExpectedID);
        System.out.println("     Expected Speaker: " + oDB.getName(iExpectedID));
        System.out.println("       Second Best ID: " + iSecondClosestID);
        System.out.println("     Second Best Name: " + oDB.getName(iSecondClosestID));

        oDB.restore();
        {
            // 1st match
            oDB.addStats(strConfig, (iIdentifiedID == iExpectedID));

            // 2nd best: must be true if either 1st true or second true (or both :))
            boolean bSecondBest =
                iIdentifiedID == iExpectedID
                ||
                iSecondClosestID == iExpectedID;

            oDB.addStats(strConfig, bSecondBest, true);
        }
        oDB.dump();
```

```
    }
    else
    {
        System.out.println("       Second Best ID: " + iSecondClosestID);
        System.out.println("     Second Best Name: " + oDB.getName(iSecondClosestID));
    }
}

/*
 * Training
 */
else if(argv[0].compareTo("--train") == 0)
{
    try
    {
        // Dir contents
        File[] aFiles = new File(argv[1]).listFiles();

        String strFileName;

        // XXX: this loop has to be in MARF
        for(int i = 0; i < aFiles.length; i++)
        {
            strFileName = aFiles[i].getPath();

            if(strFileName.toLowerCase().endsWith(".wav"))
            {
                MARF.setSampleFile(strFileName);

                int iID = oDB.getIDByFilename(strFileName, true);

                if(iID == -1)
                    System.out.println("No speaker found for \"" + strFileName + "\" for training.");
                else
                {
                    MARF.setCurrentSubject(iID);
                    MARF.train();
                }
            }
        }
    }
    catch(NullPointerException e)
    {
        System.err.println("Folder \"" + argv[1] + "\" not found.");
        System.exit(-1);
    }

    System.out.println("Done training on folder \"" + argv[1] + "\".");
}

/*
 * Stats
 */
else if(argv[0].equals("--stats"))
{
    oDB.restore();
    oDB.printStats();
```

```
            }

            /*
             * Best Result with Stats
             */
            else if(argv[0].equals("--best-score"))
            {
                oDB.restore();
                oDB.printStats(true);
            }

            /*
             * Reset Stats
             */
            else if(argv[0].equals("--reset"))
            {
                oDB.resetStats();
                System.out.println("SpeakerIdentApp: Statistics has been reset.");
            }

            /*
             * Versionning
             */
            else if(argv[0].equals("--version"))
            {
                System.out.println("Text-Independent Speaker Identification Application, v." + getVersion());
                System.out.println("Using MARF, v." + MARF.getVersion());

                validateVersions();
            }

            /*
             * Help
             */
            else if(argv[0].equals("--help") || argv[0].equals("-h"))
            {
                usage();
            }

            /*
             * Invalid major option
             */
            else
                throw new Exception("Unrecognized option: " + argv[0]);
        }

        /*
         * No arguments have been specified
         */
        catch(ArrayIndexOutOfBoundsException e)
        {
            usage();
        }

        /*
         * MARF-specific errors
         */
```

```
        catch(MARFException e)
        {
            System.err.println(e.getMessage());
            e.printStackTrace(System.err);
        }

        /*
         * Invalid option and/or option argument
         */
        catch(Exception e)
        {
            System.err.println(e.getMessage());
            e.printStackTrace(System.err);
            usage();
        }

        /*
         * Regardless whatever happens, close the db connection.
         */
        finally
        {
            try
            {
                Debug.debug("Closing DB connection...");
                oDB.close();
            }
            catch(Exception e)
            {
                Debug.debug("Closing DB connection failed: " + e.getMessage());
                e.printStackTrace(System.err);
                System.exit(-1);
            }
        }
    }

    /**
     * Displays application's usage information and exits.
     */
    private static final void usage()
    {
        System.out.println
        (
            "Usage:\n" +
            "    java SpeakerIdentApp --train <samples-dir> [options]  -- train mode\n" +
            "                         --ident <sample> [options]       -- identification mode\n" +
            "                         --stats                          -- display stats\n" +
            "                         --reset                          -- reset stats\n" +
            "                         --version                        -- display version info\n" +
            "                         --help | -h                      -- display this help and exit\n\n" +

            "Options (one or more of the following):\n\n" +

            "Preprocessing:\n\n" +
            "  -raw         - no preprocessing\n" +
            "  -norm        - use just normalization, no filtering\n" +
            "  -low         - use low-pass filter\n" +
            "  -high        - use high-pass filter\n" +
```

```
            "  -boost        - use high-frequency-boost preprocessor\n" +
            "  -band         - use band-pass filter\n" +
            "\n" +

            "Feature Extraction:\n\n" +
            "  -lpc          - use LPC\n" +
            "  -fft          - use FFT\n" +
            "  -minmax       - use Min/Max Amplitudes\n" +
            "  -randfe       - use random feature extraction\n" +
            "\n" +

            "Classification:\n\n" +
            "  -nn           - use Neural Network\n" +
            "  -cheb         - use Chebyshev Distance\n" +
            "  -eucl         - use Euclidean Distance\n" +
            "  -mink         - use Minkowski Distance\n" +
            "  -diff         - use Diff-Distance\n" +
            "  -randcl       - use random classification\n" +
            "\n" +

            "Misc:\n\n" +
            "  -debug        - include verbose debug output\n" +
            "  -spectrogram  - dump spectrogram image after feature extraction\n" +
            "  -graph        - dump wave graph before preprocessing and after feature extraction\n" +
            "  <integer>     - expected speaker ID\n" +
            "\n"
        );

        System.exit(0);
    }

    /**
     * Retrieves String representation of the application's version.
     * @return version String
     */
    public static final String getVersion()
    {
        return MAJOR_VERSION + "." + MINOR_VERSION + "." + REVISION;
    }

    /**
     * Retrieves integer representation of the application's version.
     * @return integer version
     */
    public static final int getIntVersion()
    {
        return MAJOR_VERSION * 100 + MINOR_VERSION * 10 + REVISION;
    }

    /**
     * Makes sure the applications isn't run against older MARF version.
     * Exits with 1 if the MARF version is too old.
     */
    public static final void validateVersions()
    {
        if(MARF.getIntVersion() < (0 * 100 + 3 * 10 + 0))
        {
```

```
            System.err.println
            (
                "Your MARF version (" + MARF.getVersion() +
                ") is too old. This application requires 0.3.0 or above."
            );

            System.exit(1);
        }
    }
}

// EOF
```

D.2 SpeakersIdentDb.java

```
import java.awt.Point;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.text.DecimalFormat;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import marf.Storage.Database;
import marf.Storage.StorageException;
import marf.util.Debug;


/**
 * <p>Class SpeakersIdentDb manages database of speakers on the application level.</p>
 * <p>XXX: Move stats collection over to MARF.</p>
 *
 * <p>$Id: SpeakersIdentDb.java,v 1.21 2005/06/04 01:53:56 mokhov Exp $</p>
 *
 * @author Serguei Mokhov
 * @version $Revision: 1.21 $
 * @since 0.0.1
 */
public class SpeakersIdentDb
extends Database
{
    /**
     * Hashes "config string" -&gt; Vector(FirstMatchPoint(XSuccesses, YFailures),
     * SecondMatchPoint(XSuccesses, YFailures)).
```

```
     */
    private Hashtable oStatsPerConfig = null;

    /**
     * Array of sorted stats refs.
     */
    private Vector[] oSortedStatsRefs = null;

    /**
     * A vector of vectors of speakers info pre-loded on <code>connect()</code>.
     * @see #connect()
     */
    private Hashtable oDB = null;

    /**
     * "Database connection".
     */
    private BufferedReader oConnection = null;

    /**
     * Constructor.
     * @param pstrFileName filename of a CSV file with IDs and names of speakers
     */
    public SpeakersIdentDb(final String pstrFileName)
    {
        this.strFilename = pstrFileName;
        this.oDB = new Hashtable();
        this.oStatsPerConfig = new Hashtable();
    }

    /**
     * Retrieves Speaker's ID by a sample filename.
     * @param pstrFileName Name of a .wav file for which ID must be returned
     * @param pbTraining indicates whether the filename is a training (<code>true</code>) sample or testing (<code>false</code>)
     * @return int ID
     * @throws StorageException
     */
    public final int getIDByFilename(final String pstrFileName, final boolean pbTraining)
    throws StorageException
    {
        String str;

        // Extract actual file name without preceeding path (if any)
        if(pstrFileName.lastIndexOf('/') >= 0)
            str = pstrFileName.substring(pstrFileName.lastIndexOf('/') + 1, pstrFileName.length());
        else if(pstrFileName.lastIndexOf('\\') >= 0)
            str = pstrFileName.substring(pstrFileName.lastIndexOf('\\') + 1, pstrFileName.length());
        else
            str = pstrFileName;

        Enumeration oIDs = oDB.keys();

        // Traverse all the info vectors looking for sample filename
        while(oIDs.hasMoreElements())
        {
            Integer id = (Integer)oIDs.nextElement();
```

```
            //System.out.println("File: " + pstrFileName + ", id = " + id.intValue());

            Vector oSpeakerInfo = (Vector)oDB.get(id);
            Vector oFilenames;

            if(pbTraining == true)
                oFilenames = (Vector)oSpeakerInfo.elementAt(1);
            else
                oFilenames = (Vector)oSpeakerInfo.elementAt(2);

            // Start from 1 because 0 is speaker's name
            for(int i = 0; i < oFilenames.size(); i++)
            {
                String tmp = (String)oFilenames.elementAt(i);

                if(tmp.compareTo(str) == 0)
                    return id.intValue();
            }
        }

        return -1;
    }

    /**
     * Retrieves speaker's name by their ID.
     * @param piID ID of a person in the DB to return a name for
     * @return name string
     * @throws StorageException
     */
    public final String getName(final int piID)
    throws StorageException
    {
        //Debug.debug("getName() - ID = " + piID + ", db size: " + oDB.size());
        String strName;

        Vector oDBEntry = (Vector)oDB.get(new Integer(piID));

        if(oDBEntry == null)
            strName = "Unknown Speaker (" + piID + ")";
        else
            strName = (String)oDBEntry.elementAt(0);

        return strName;
    }

    /**
     * Connects to the "database" of speakers (opens the text file :-)).
     * @throws StorageException
     */
    public void connect()
    throws StorageException
    {
        // That's where we should establish file linkage and keep it until closed
        try
        {
            this.oConnection = new BufferedReader(new FileReader(this.strFilename));
            this.bConnected = true;
```

```
        }
        catch(IOException e)
        {
            throw new StorageException
            (
                "Error opening speaker DB: \"" + this.strFilename + "\": " +
                e.getMessage() + "."
            );
        }
    }

    /**
     * Retrieves speaker's data from the text file and populates
     * internal data structures.
     * @throws StorageException
     */
    public void query()
    throws StorageException
    {
        // That's where we should load db results into internal data structure

        String tmp;
        int id = -1;

        try
        {
            tmp = this.oConnection.readLine();

            while(tmp != null)
            {
                StringTokenizer stk = new StringTokenizer(tmp, ",");
                Vector oSpeakerInfo = new Vector();

                // get ID
                if(stk.hasMoreTokens())
                    id = Integer.parseInt(stk.nextToken());

                // speaker's name
                if(stk.hasMoreTokens())
                {
                    tmp = stk.nextToken();
                    oSpeakerInfo.add(tmp);
                }

                // training file names
                Vector oTrainingFilenames = new Vector();

                if(stk.hasMoreTokens())
                {
                    StringTokenizer oSTK = new StringTokenizer(stk.nextToken(), "|");

                    while(oSTK.hasMoreTokens())
                    {
                        tmp = oSTK.nextToken();
                        oTrainingFilenames.add(tmp);
                    }
                }
```

```
                oSpeakerInfo.add(oTrainingFilenames);

                // testing file names
                Vector oTestingFilenames = new Vector();

                if(stk.hasMoreTokens())
                {
                    StringTokenizer oSTK = new StringTokenizer(stk.nextToken(), "|");

                    while(oSTK.hasMoreTokens())
                    {
                        tmp = oSTK.nextToken();
                        oTestingFilenames.add(tmp);
                    }
                }

                oSpeakerInfo.add(oTestingFilenames);

                Debug.debug("Putting ID=" + id + " along with info vector of size " + oSpeakerInfo.size());

                this.oDB.put(new Integer(id), oSpeakerInfo);

                tmp = this.oConnection.readLine();
            }
        }
        catch(IOException e)
        {
            throw new StorageException
            (
                "Error reading from speaker DB: \"" + this.strFilename +
                "\": " + e.getMessage() + "."
            );
        }
    }

    /**
     * Closes (file) database connection.
     * @throws StorageException
     */
    public void close()
    throws StorageException
    {
        // Close file
        if(this.bConnected == false)
            throw new StorageException("SpeakersIdentDb.close() - not connected");

        try
        {
            this.oConnection.close();
            this.bConnected = false;
        }
        catch(IOException e)
        {
            throw new StorageException(e.getMessage());
        }
    }
```

```
/**
 * Adds one more classification statics entry.
 * @param pstrConfig String representation of the configuration the stats are for
 * @param pbSuccess <code>true</code> if classification was successful; <code>false</code> otherwise
 */
public final void addStats(final String pstrConfig, final boolean pbSuccess)
{
    addStats(pstrConfig, pbSuccess, false);
}

/**
 * Adds one more classification statics entry and accounts for the second best choice.
 * @param pstrConfig String representation of the configuration the stats are for
 * @param pbSuccess <code>true</code> if classification was successful; <code>false</code> otherwise
 * @param pbSecondBest <code>true</code> if classification was successful; <code>false</code> otherwise
 */
public final void addStats(final String pstrConfig, final boolean pbSuccess, final boolean pbSecondBest)
{
    Vector oMatches = (Vector)oStatsPerConfig.get(pstrConfig);
    Point oPoint = null;

    if(oMatches == null)
    {
        oMatches = new Vector(2);
        oMatches.add(new Point());
        oMatches.add(new Point());
        oMatches.add(pstrConfig);
    }
    else
    {
        if(pbSecondBest == false)
            oPoint = (Point)oMatches.elementAt(0); // Firts match
        else
            oPoint = (Point)oMatches.elementAt(1); // Second best match
    }

    int x = 0; // # of successes
    int y = 0; // # of failures

    if(oPoint == null) // Didn't exist yet; create new
    {
        if(pbSuccess == true)
            x = 1;
        else
            y = 1;

        oPoint = new Point(x, y);

        if(oPoint == null)
        {
            System.err.println("SpeakersIdentDb.addStats() - oPoint null! Out of memory?");
            System.exit(-1);
        }

        if(oMatches == null)
        {
```

```
                System.err.println("SpeakersIdentDb.addStats() - oMatches null! Out of memory?");
                System.exit(-1);
            }

            if(oMatches.size() == 0)
            {
                System.err.println("SpeakersIdentDb.addStats() - oMatches.size = 0");
                System.exit(-1);
            }

            if(pbSecondBest == false)
                oMatches.setElementAt(oPoint, 0);
            else
                oMatches.setElementAt(oPoint, 1);

            oStatsPerConfig.put(pstrConfig, oMatches);
        }

        else // There is an entry for this config; update
        {
            if(pbSuccess == true)
                oPoint.x++;
            else
                oPoint.y++;
        }
    }

    /**
     * Dumps all collected statistics to STDOUT.
     * @throws Exception
     */
    public final void printStats()
    throws Exception
    {
        printStats(false);
    }

    /**
     * Dumps collected statistics to STDOUT.
     * @param pbBestOnly <code>true</code> - print out only the best score number; <code>false</code> - all stats
     * @throws Exception
     */
    public final void printStats(boolean pbBestOnly)
    throws Exception
    {
        if(this.oStatsPerConfig.size() == 0)
        {
            System.err.println("SpeakerIdentDb: no statistics available. Did you run the recognizer yet?");
            return;
        }

        // First row is for the identified results, 2nd is for 2nd best ones.
        String[][] astrResults = new String[2][oStatsPerConfig.size()];

        this.oSortedStatsRefs = (Vector[])oStatsPerConfig.values().toArray(new Vector[0]);
        marf.util.Arrays.sort(oSortedStatsRefs, new StatsPercentComparator(StatsPercentComparator.DESCENDING));
```

```
        int iResultNum = 0;

        System.out.println("guess,run,config,good,bad,%");

        for(int i = 0; i < oSortedStatsRefs.length; i++)
        {
            String strConfig = (String)(oSortedStatsRefs[i]).elementAt(2);

            for(int j = 0; j < 2; j++)
            {
                Point oGoodBadPoint = (Point)(oSortedStatsRefs[i]).elementAt(j);
                String strGuess = (j == 0) ? "1st" : "2nd";
                String strRun = (iResultNum + 1) + "";
                DecimalFormat oFormat = new DecimalFormat("#,##0.00;(#,##0.00)");
                double dRate = ((double)oGoodBadPoint.x / (double)(oGoodBadPoint.x + oGoodBadPoint.y)) * 100;

                if(pbBestOnly == true)
                {
                    System.out.print(oFormat.format(dRate));
                    return;
                }

                astrResults[j][iResultNum] =
                    strGuess + "," +
                    strRun + "," +
                    strConfig + "," +
                    oGoodBadPoint.x + "," + // Good
                    oGoodBadPoint.y + "," + // Bad
                    oFormat.format(dRate);
            }

            iResultNum++;
        }

        // Print all of the 1st match
        for(int i = 0; i < astrResults[0].length; i++)
            System.out.println(astrResults[0][i]);

        // Print all of the 2nd match
        for(int i = 0; i < astrResults[1].length; i++)
            System.out.println(astrResults[1][i]);
    }

    /**
     * Resets in-memory and on-disk statistics.
     * @throws StorageException
     */
    public final void resetStats()
    throws StorageException
    {
        oStatsPerConfig.clear();
        dump();
    }

    /**
     * Dumps statistic's Hashtable object as gzipped binary to disk.
     * @throws StorageException
```

```
     */
    public void dump()
    throws StorageException
    {
        try
        {
            FileOutputStream fos = new FileOutputStream(this.strFilename + ".stats");
            GZIPOutputStream gzos = new GZIPOutputStream(fos);
            ObjectOutputStream out = new ObjectOutputStream(gzos);

            out.writeObject(this.oStatsPerConfig);
            out.flush();
            out.close();
        }
        catch(Exception e)
        {
            throw new StorageException(e);
        }
    }

    /**
     * Reloads statistic's Hashtable object from disk.
     * If the file did not exist, it creates a new one.
     * @throws StorageException
     */
    public void restore()
    throws StorageException
    {
        try
        {
            FileInputStream fis = new FileInputStream(this.strFilename + ".stats");
            GZIPInputStream gzis = new GZIPInputStream(fis);
            ObjectInputStream in = new ObjectInputStream(gzis);

            this.oStatsPerConfig = (Hashtable)in.readObject();
            in.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println
            (
                "NOTICE: File " + this.strFilename +
                ".stats does not seem to exist. Creating a new one...."
            );

            resetStats();
        }
        catch(ClassNotFoundException e)
        {
            throw new StorageException
            (
                "SpeakerIdentDb.retore() - ClassNotFoundException: " +
                e.getMessage()
            );
        }
        catch(Exception e)
        {
```

```
            throw new StorageException(e);
        }
    }
}

/**
 * <p>Used in sorting by percentage of the stats entries
 * in either ascending or descending order.</p>
 *
 * <p>TODO: To be moved to Stats.</p>
 *
 * @author Serguei Mokhov
 * @version $Revision: 1.21 $
 */
class StatsPercentComparator
extends marf.util.SortComparator
{
    /**
     * Mimics parent's constructor.
     */
    public StatsPercentComparator()
    {
        super();
    }

    /**
     * Mimics parent's constructor.
     * @param piSortMode either DESCENDING or ASCENDING sort mode
     */
    public StatsPercentComparator(final int piSortMode)
    {
        super(piSortMode);
    }

    /**
     * Implementation of the Comparator interface for the stats objects.
     */
    public int compare(Object poStats1, Object poStats2)
    {
        Vector oStats1 = (Vector)poStats1;
        Vector oStats2 = (Vector)poStats2;

        Point oGoodBadPoint1 = (Point)(oStats1.elementAt(0));
        Point oGoodBadPoint2 = (Point)(oStats2.elementAt(0));

        double dRate1 = ((double)oGoodBadPoint1.x / (double)(oGoodBadPoint1.x + oGoodBadPoint1.y)) * 100;
        double dRate2 = ((double)oGoodBadPoint2.x / (double)(oGoodBadPoint2.x + oGoodBadPoint2.y)) * 100;

        switch(this.iSortMode)
        {
            case DESCENDING:
                return (int)((dRate2 - dRate1) * 100);

            case ASCENDING:
            default:
                return (int)((dRate1 - dRate2) * 100);
        }
```

```
    }
}

// EOF
```

Appendix E

TODO

```
MARF TODO/Wishlist
------------------

$Header: /cvsroot/marf/marf/TODO,v 1.67 2005/08/07 23:49:12 mokhov Exp $

Legend:
-------

"-"  -- TODO
"*"  -- done (for historical records)
"+-" -- somewhat done ("+" is degree of completed work, "-" remaining).
"?"  -- unsure if feature is needed or how to proceed about it or it's potentially far away


THE APPS

- SpeakerIdentApp
  - GUI
  - Real-Time recording (does it belong here?)
  - Move dir. read from the app to MARF in training section {0.3.0}
  - Enhance batch recognition (do not re-load training set per sample) {0.3.0}
  - Enhance Option Processing
  - Add --batch-ident option {0.3.0}
    - Make use of OptionProcessor
    - Enhance options with arguments, e.g. -fft=1024, -lpc=40, -lpc=40,256, keeping the existing defaults
    - Add option: -data-dir=DIRNAME other than default to specify a dir where to store training sets and stuff
    - Add -mink=r
    - Add single file training option
    - Dump stats: -classic -latex -csv
  - Make binary/optimized distro
  - Generate data.tex off speakers.tex in manual.
  - Convert testing.sh to Java
  - Open up -randcl with -nn
  - Finish testing.bat
  * Make executable .jar
  * Improve on javadoc
  * ChangeLog
  * Sort the stats
  * Add classification methods to training in testing.sh
  * Implement batch plan execution
```

```
- LangIdentApp
  - Integrate
  - Add GUI
  - Make executable .jar
  - Release.

- ProbabilisticParsingApp
  - Integrate
  - Add GUI
  - Make executable .jar
  - Release.

- ZipfLawApp
  - Integrate
  - Add GUI
  - Make executable .jar
  - Release.

- TestFilters
  - The application has already a Makefile and a JBuilder project file.
    We would want to add a NetBeans project as well.
  - Add GUI
  * Release.
  * It only tests HighFrequencyBoost, but we also have BandpassFilter,
    HighPassFilter, and LowPassFilter.
    These have to be tested.
  * The output of every filter will have to be stored in a expected
    (needs to be created) folder for future regression testing. Like
    with TestMath.
  * Option processing has to be done and standartized
    by using marf.util.OptionProcessor uniformly in all apps. But we can
    begin with this one. The options then would be: --high, --low, --band,
    and --boost that will correspond to the appropriate filters I mentioned.
  * The exception handling has to be augmented to print the error message
    and the stack trace to System.err.
  * Apply coding conventions to naming variables, etc.
  * Make executable .jar

- TestNN
  +- Fix to work with new MARF API
  - Add use of OptionProcessor
  - Apply coding conventions
  - Add GUI
  - Make executable .jar
  - Release.

- TestLPC
  - Add GUI
  * Release.
  * Fix to work with new MARF API
  * Add use of OptionProcessor
  * Apply coding conventions
  * Make executable .jar

- TestFFT
  - Add GUI
```

```
  * Release.
  * Make executable .jar
  * Fix to work with new MARF API
  * Add use of OptionProcessor
  * Apply coding conventions

- MathTestApp
  - Add GUI
  * Release.
  * Make executable .jar

- TestWaveLoader
  - Add GUI
  * Release.
  * Fix to work with new MARF API
  * Apply coding conventions
  * Make executable .jar

- TestLoaders
  - Add GUI
  * Create a-la TestFilters with option processing.
  * Make executable .jar
  * Release.

- Regression
  - one script calls all the apps and compares new results vs. expected
  - Matrix ops
  +- Fix TestNN
  - Add GUI
  - Make executable .jar
  - Release.

- Graph % vs. # of samples used per method

- Release all apps as packages at Source Forge
  - Bundle
  ++- Individually

- SpeechRecognition
  - Define

- InstrumentIdentification
  - Define


THE BUILD SYSTEM

  - Perhaps at some point we'd need make/project files for other Java IDEs, such as
    - IBM Visual Age
    - Ant
    - Windoze .bat file(s)?
    * Sun NetBeans
  * A Makefile per directory
  * Make utility detection test
  * global makefile in /marf
  * fix doc's global makefile
  * Global Makefile for apps
```

```
    (will descend to each dir and build all the apps.)
  * Build and package distrubution
    * MARF
    * App


DISTROS

  ++++- Apps jars
  - Different JDKs (1.4/1.5)
  - rpm
    - FC
    - Mandrake
    - RH
  - deb
  - dmg
  - iso


THE FRAMEWORK

- All/Optimization/Testing

  - Implement text file support/toString() method for
    all the modules for regression testing

  - Threading and Parallelism
    - PR and FE are safe to run in ||
    - Fix NNet for ArrayLists->Vector

  * Implement getMARFSourceCodeRevision() method for CVS revision info.


- Preprocessing

  - Make dump()/restore() to serialize filtered output {0.3.0}
  - Fix hardcoding of filter boundaries
  - Implement
    - Enable changing values of frequency boundaries and coeffs. in filters by an app.
    - Endpoint {1.0.0}
    - "Compressor" [steve]
    - Methods:  {1.0.0}
      - removeNoise()
      - removeSilence()
      - cropAudio()
    * High-pass filter with high-frequency boost together {0.3.0}
    * Band-pass Filter {0.2.0}
  * Move BandpassFilter and HighFrequencyBoost under FFTFilter package with CVS comments
  * Tweak the filter values of HighPass and HighFrequencyBoost filters


- Feature Extraction

  - Make modules to dump their features for future use by NNet and maybe others {0.3.0}
  - Implement {1.0.0}
    - F0
      - fundamental frequency estimation,
```

```
      providing us with a few more features.
    - Cepstral
    - Segmentation
    * RandomFeatureExtraction {0.2.0}
  - Enhance
    - MinMaxAmplitudes to pick different features, not very close to each other

- Classification

  - Implement
    +++-- Mahalanobis Distance {0.3.0}
      - Learning Covariance Matrix
    - Stochastic [serge] {1.0.0}
      - Gaussian Mixture Models
      - Hidden Markov Models {1.0.0}
    - SimilarityClassifier {0.3.0}
      ? Boolean Model
      ? Vector Space Model
    * Minkowski's Distance {0.2.0}
    * RandomClassification {0.2.0}

  - Fully Integrate
    +- MaxProbabilityClassifier
      - Push StatisticalEstimator to Stochastic
    +- ZipfLaw

  - Fix and document NNet {0.*.*}
    - add % of correct/incorrect expected to train() {0.3.0}
    - ArrayList ---> Vector, because ArrayList is NOT thread-safe {0.3.0}
    - Second Best (by doubling # of output Neurons with the 2nd half being the 2nd ID)
    +- Epoch training
    * dump()/retore() {0.2.0}

  - Distance Classifiers
    - make distance() throw an exception maybe?
    * Move under Distance package
    * DiffDistance

- Sample Loaders

  - Create Loaders for Java-supported formats:
    +--- AIFC
    +--- AIFF
    +--- AU
    +--- SND
  +--- Add MIDI support
  * Create MARFAudioFileFormat extends AudioFileFormat
  * Enumerate all types in addition to ours from FileAudioFormat.Types

- marf.speech package
  - Recognition
  - Dictionaries
  - Generation


- NLP package
  ++++- Integrate
```

```
    +- Classification
    - Potter's Stemmer
    * Stats
    * Storage management
    * StatisticalEstimators
    * NLP.java
    * Parsing
    * Util
    * Collocations

? Integrate AIMA stuff


- Stats {0.3.0}

  - Move stats collection from the app and other places to StatsCollector
  - Timing
  - Batch progress report
  - Rank results
  - Results validation (that numbers add up properly e.g. sum of BAD and GOOD should be equal to total)


- Algos

  +- Algorithm decoupling to marf.algos or marf.algorithms or ... {0.4.0}
    * To marf.math.Algorithms {0.3.0}
  - marf.algos.Search
  - marf.util.DataStructures -- Node / Graph --- to be used by the networks and state machines
  * move out hamming() from FeatureExtraction


* marf.math

  * Integrate Vector operations
  * Matrix:
    * Add {0.3.0.3}
      * translate
      * rotate
      * scale
      * shear


- GUI {0.5.0}

  - Make them actual GUI components to be included into App
    - Spectrogram
      +----- Implement SpectrogramPanel
      - Draw spectrogram on the panel
      * Fix filename stuff (module_dirname to module_filename)
    - WaveGrapher
      +----- Implement WaveGrapherPanel
      - Draw waves on the panel
  - Fix WaveGrapher
    - Sometimes dumps files of 0 length
    - Make it actually ouput PPM or smth like that (configurable?)
    - Too huge files for samp output.
    - Have LPC understand it
```

```
  - Config tool
  - Web interface?


- MARF.java

  - Concurrent use of modules of the same type
    - FFT and F0 can both be applied like normalization and filtering
  - Implement
    - streamedRecognition()
    +- train()
       - Add single file training
    - Inter-module compatibility (i.e. specific modules can only work
      with some other specific modules and not the others)
      - Module Compatibility Matrix
      - Module integer and String IDs
  - Server Part {2.0.0}
  * enhance error reporting
  * Embed the NLP class


* MARF Exceptions Framework {0.3.0}
  * Propagate to NLP properly
    * NLPException
  * StorageException
  * Have all marf exceptions inherit from util.MARFException


- marf.util

  - complete and document
    +- Matrix
    +- FreeVector
    * Arrays
    * Debug
  ? PrintFactory
    - Move NeuralNetowork.indent()
  ? marf.util.upgrade
  * OptionProcessor
    * Integrate {0.3.0.2}
    * Add parsing of "name=value" options {0.3.0.3}
  * Add marf.util.Debug module. {0.3.0.2}
    * marf.util.Debug.debug()
    * Replicate MARF.debug() --> marf.util.Debug.debug()


- Storage

  - ModuleParams:
    - have Hashtables instead of Vectors
      to allow params in any order and in any number.
    - maybe use OptionProcessor(s) or be its extension?
  - Keep all data files under marf.data dir, like training sets, XML, etc {0.3.0}
  - Implement
    - Schema (as in DB for exports)
      - Common attribute names for
        - SQL
```

```
        - XML
        - HTML
        - CSV
      - Metadata / DDL
    - Dump/Restore Types
      +++- DUMP_BINARY (w/o compression) {0.3.0}
      - DUMP_XML {?.?.?}
      - DUMP_CSV {?.?.?}
      - DUMP_HTML {?.?.?}
      - DUMP_SQL {?.?.?}
  +- Revise TrainingSet stuff
    ? Cluster mode vs. feature set mode
    - TrainingSet
      - upgradability {?.?.?}
      - convertability: gzbin <-> bin <-> csv <-> xml <-> html <-> sql
  +--- Add FeatureSet {0.3.0}
  - Revise dump/restore implementation to check for unnecessary
    file writes
  * Integrate IStorageManager
  * Move DUMP_* flags up to IStorageManager
  * Add re-implemented StorageManager and integrate it


* Clean up
  * CVS:
    ? Rename /marf/doc/sgml to /marf/doc/styles
    x Remove /marf/doc/styles/ref
    * Remove --x permissions introduced from windoze in files:
      * /marf/doc/src/tex/sampleloading.tex
      * /marf/doc/src/graphics/*.png
      * /marf/doc/src/graphics/arch/*.png
      * /marf/doc/src/graphics/fft_spectrograms/*.ppm
      * /marf/doc/src/graphics/lpc_spectrograms/*.ppm
      * /marf/doc/arch.mdl
      * /marf/src/marf/Classification/Distance/EuclideanDistance.java
      * /marf/src/marf/Preprocessing/FFTFilter.java
      * /apps/SpeakerIdentApp/SpeakerIdentApp.jpx
      * /apps/SpeakerIdentApp/testing-samples/*.wav
      * /apps/SpeakerIdentApp/testing-samples/*.wav
      * /apps/TestFilters/TestFilters.*
    * Add NLP revisions directly to the CVS (by SF staff)
      * Parsing
      * Stemming
      * Collocations
    * Move distance classifiers with CVS log
      to Distance
    * remove uneeded attics and corresponding dirs
      * "Ceptral"
      * Bogus samples


THE CODE

* Define coding standards
+++- Propagate them throughout the code
* Document
```

```
THE SAMPLE DATABASES

- /samples/ -- Move all wav and corpora files there from apps
  - WAV/
    - training-samples/
    - testing-samples/
    - README
    - speakers.txt
    - <training-sets>

  - corpora/
    - training/
      - en/
      - fr/
      - ru/
      - ...
    - testing/
      - en/
      - fr/
      - ru/
      - ...
    - heldout/
    - README
    - <training-sets>

- Make releases


THE TOOLS

* Add module

- Add tools:
  - marfindent
  * stats2latex
  * cvs2cl
  * cvs2html

- upgrade/
  - MARFUpgrade.java
    - an app or a marf module?
    - cmd-line?
    - GUI?
    - Interactive?
  - v012/TrainingSet.java
  - v020/TrainingSet.java
         FeatureSet.java


THE DOCS

- docs [s]
  - report.pdf -> manual.pdf
  - autosync from the report
    - history.tex -> HISTORY
    - legal.tex -> COPYRIGHT
```

```
    - installation.tex -> INSTALL
  - Arch Update [serge]
    +- gfx model (rr)
      - gui: add StorageManager
      - add util package
      - add math package
      - add nlp package
    * update doc
    * newer images
  - MARF-specific exceptions
  - Account for dangling .tex files
    - old-results.tex
    - output.tex
    * installation.tex
    * training.tex
    * sample-formats.tex
    * cvs.tex
    * history.tex
    * f0.tex
    * notation.tex
    * sources.tex
  ++- better doc format and formulas
  - Results:
    - Add modules params used, like r=6 in Minkowski, FFT input 1024, etc
    - Add time took
  * fix javadoc 1.4/1.5 warnings
  * fix Appendix
  * split-out bibliography
  * index
  * ChangeLog
  * report components [serge]

- web site
  - Publish
    * TODO
    +- ChanageLog
      * Raw
      - HTML
    - Manual
      - Add HTML
  * Add training sets
  * CVS
  * autoupdate from CVS

EOF
```

Index

API
- Arrays, 80, 81
- BANDPASS_FFT_FILTER, 76
- BandpassFilter, 76
- BaseThread, 80, 81
- ByteUtils, 80
- Debug, 80
- doFFT(), 50
- doLPC(), 50
- Dummy, 77, 78
- ExpandedThreadGroup, 80
- F0, 57
- FeatureExtraction.FFT.FFT, 78
- HIGH_FREQUENCY_BOOST_FFT_FILTER, 76
- HIGH_PASS_FFT_FILTER, 76
- HighFrequencyBoost, 76
- HighPassFilter, 76
- Logger, 80
- LOW_PASS_FFT_FILTER, 76
- LowPassFilter, 76
- LPC, 57
- MahalanobisDistance, 79
- MARF, 8, 36, 76, 78
- marf.Classification.Distance.DiffDistance, 46
- marf.Classification.Distance.Distance, 45, 46
- marf.Classification.Distance.MahalanobisDistance, 45
- marf.FeatureExtraction.FFT, 77
- marf.gui, 50
- marf.MARF, 30, 45, 46, 76, 77
- marf.math, 79
- marf.math.Algortithms.Hamming, 37
- marf.Preprocessing.Dummy, 78
- marf.Preprocessing.Dummy.Dummy, 30
- marf.Preprocessing.Dummy.Raw, 30
- marf.Preprocessing.FFTFilter.*, 76
- marf.Preprocessing.FFTFilter.HighFrequencyBoost, 33
- marf.Preprocessing.FFTFilter.HighPassFilter, 33
- marf.Preprocessing.Preprocessing, 76
- marf.Storage, 78, 79
- marf.Storage.Loaders, 78, 79
- marf.Storage.Sample, 76, 77
- marf.Storage.SampleLoader, 76, 77
- marf.Storage.WAVLoader, 76, 78
- marf.util, 79
- marf.util.OptionProcessor, 76, 77
- MathTestApp, 79
- Matrix, 79
- NeuralNetwork, 18
- normalize(), 76, 78
- OptionProcessor, 76, 78–81
- preprocess(), 76, 78
- Preprocessing, 76, 78
- recognize(), 8
- removeNoise(), 76, 78
- removeSilence(), 76, 78
- Sample, 76, 78, 79
- SampleLoader, 76, 78, 79
- SINE, 76, 78
- SineLoader, 75–79
- Sineloader, 77
- SpeakerIdentApp, 3, 30, 45, 46, 59, 75, 87
- SpeakersIdentDb, 87
- Spectrogram, 50, 83
- Storage, 21

- StorageManager, 80
- test, 30, 45, 46
- TestFFT, 77, 78
- TestFilters, 75, 76
- TestLoaders, 79
- TestLPC, 77
- TestWaveLoader, 78, 79
- train(), 8
- Vector, 79
- WAV, 76, 78
- WaveGrapher, 5, 52
- WAVLoader, 16, 75–79

Applications, 75
- ENCSAssets, 81
- ENCSAssetsCron, 81
- ENCSAssetsDesktop, 80
- GIPSY, 80
- MathTestApp, 79
- SpeakerIdentApp, 75
- TestFFT, 77
- TestFilters, 75
- TestLoaders, 79
- TestLPC, 77
- TestWaveLoader, 78

Classification, 43
- Diff Distance, 45
- Mahalanobis Distance, 45

CLASSPATH, 20

Coding and Naming Conventions, 7

Distance
- Diff, 45
- Mahalanobis, 45

Experiments, 53

External Applications
- ENCSAssets, 81
- ENCSAssetsCron, 81
- ENCSAssetsDesktop, 80
- GIPSY, 80

F0, 41

Feature Extraction, 37
- F0, 41
- Min/Max Amplitudes, 42

Files
- ~/.emacs, 6
- ~/.vimrc, 7
- expected, 76, 78–80
- expected/sine.out, 77
- gmake, 18
- graham13.wav, 83
- INSTALL, 18
- make, 18
- marf-<version>, 19
- math.out, 80

Filters
- High Frequency Boost, 33
- High-Pass Filter, 33
- High-Pass High Frequency Boost Filter, 36

GIPSY, 80

GNU, 18, 19

GUI, 50

Hamming Window, 37
- Implementation, 37
- Theory, 37

High Frequency Boost, 33

High-Pass Filter, 33

High-Pass High Frequency Boost Filter, 36

Java, 3

Limiations, 16

LPC, 40

MARF
- Application Point of View, 8
- Applications, 75
- Architecture, 8
- Authors, 4
- Coding Conventions, 7
- Core Pipeline, 8, 10, 11

Current Limitaions, 16
External Applications, 80
GUI, 50
History, 5
Installation, 18
Introduction, 3
Packages, 11
Project Location, 6
Purpose, 3
Research Applications, 75
Source Code, 6
source code formatting, 6
Testing Applications, 75
Min/Max Amplitudes, 42

NLP, 49

Preprocessing, 30
Raw, 30

regression test application, 19
Research Applications
SpeakerIdentApp, 75
Results, 58

Sample Data, 53
Source code formatting, 6

Testing Applications
MathTestApp, 79
TestFFT, 77
TestFilters, 75
TestLoaders, 79
TestLPC, 77
TestWaveLoader, 78
Tools
CVS, 84
diff, 46
Excel, 52
gmake, 20
gmake clean, 20
gnuplot, 52
javac, 18
less, 7
make, 18, 19
more, 7
vim, 7

upgrading, 19