FoCal Multi-class: Toolkit for Evaluation, Fusion and Calibration of Multi-class Recognition Scores
— Tutorial and User Manual —

Niko Brümmer
Spescom DataVoice
niko.brummer@gmail.com

June 2007

Contents

Chapter 1

Introduction

This document is the user manual for the **FoCal Multi-class Toolkit**, which is a collection of utilities, written in MATLAB, to help researchers with the problems of evaluation, fusion, calibration and decision-making in multi-class statistical pattern recognition.

Although applicable more widely, the toolkit has been designed primarily with the NIST LRE-07[1] *language recognition evaluation* in mind.

This toolkit does not form a complete pattern or language recognizer, rather it helps the creators of such recognizers to:

- Evaluate the quality of a recognizer under development.
- Fuse the outputs of multiple recognizers to form a better recognizer.
- Calibrate the recognizer output, so that it can be used to make cost effective Bayes decisions, over a wide range of multi-class recognition applications.

1.1 Availability

Documentation and MATLAB code for this toolkit are available at:

`http://niko.brummer.googlepages.com/focalmulticlass`

This toolkit is made freely available, for non-commercial use, with the understanding that the authors of the toolkit and their employers cannot be held responsible in any way for the toolkit or its use.

[1] `http://www.nist.gov/speech/tests/lang/2007/index.htm`

1.2 Relationship to original FoCal Toolkit

The original **FoCal Toolkit**[2], by the same author and also written in MATLAB, was designed with application to the NIST SRE[3] *speaker detection evaluations* in mind. Speaker detection is a *two-class* recognition problem.

In contrast, the *FoCal Multi-class* toolkit is designed for *multi-class* recognition problems, but it has the following in common with the original FoCal:

- Both are applicable to the evaluation, fusion and calibration of recognizer *scores*.
- Both strive to optimize the *information* delivered to the user by the recognizer, by (i) information-theoretic evaluation (C_{llr}) and (ii) by optimizing fusion and calibration via logistic regression.
- Both emphasize *application-independence* and strive to help researchers design recognizers that can be applied to a wide variety of recognition applications.

1.3 Does it work?

Most of this toolkit as it stands, has been tested only on synthetic data, but the central ideas have been proven in published language recognition experiments. In particular:

- The multi-class C_{llr} evaluation objective and its refinement/calibration decomposition have been demonstrated on NIST LRE'05 data [3].
- The generative Gaussian backend is well-known in language recognition and has been used by the author of this toolkit also on LRE'05 data [4].

1.4 Manual organization

The rest of this manual is organized as follows:

Part I is a tutorial introduction to the basic principles on which this toolkit is based. The following should be considered essential reading:

[2] `http://www.dsp.sun.ac.za/~nbrummer/focal/`

[3] `http://www.nist.gov/speech/tests/spk/index.htm`

- All of chapter 2 (except possibly for section 2.3, if it does not interest you).
- All of chapter 3.
- Relevant parts of chapter 4, if you are interested in using Gaussian backends.

Part II has installation instructions as well as a reference section for all of the functions in the toolkit.

Also keep in mind that MATLAB-style help comments is available (in the m-file source code) for most of the functions and provides extensive additional information.

1.5 Acknowledgements

The author wishes to thank Lukáš Burget for making his HLDA optimization code available for use in this toolkit and David van Leeuwen for research collaboration.

Part I

Tutorial: Basic Principles

Chapter 2

Application-independent multi-class recognition

This toolkit is designed to be applicable to very general multi-class recognition applications. In particular, these applications include the following 'flavours' of recognition problems: (i) both *closed-set* and *open-set* and (ii) both *identification* and *detection*. The toolkit design is grounded on the assumption that for every kind of multi-class recognition problem:

> There is given a list of N pattern classes, $H_1, H_2, \ldots, H_N$, for each of which a (i) *prior* probability is given and for each of which a (ii) *likelihood* can be calculated from the input data.

In the context of the NIST Language Recognition Evaluation, we note:

- When doing *detection* of a target class, say H_i, even though we need only to make the binary decision of whether H_i is present or not, we still require the above assumption about priors and likelihoods for the other $N-1$ classes to hold.
- In the *closed-set* language recognition case, $H_1, H_2, \ldots, H_N$ represent N different explicitly specified languages (or dialects). In the *open-set* case, $H_1, H_2, \ldots, H_{N-1}$ are explicitly specified languages, and H_N denotes the *none-of-the-above*, or the *out-of-set* language hypothesis.

2.1 Input

In all cases of multi-class recognition (detection or identification), we assume that the following four essential inputs are given:

1. **Input data** x. (In language recognition x is a speech segment.)

2. **Prior knowledge** in the form of a list of N possible classes to which x may belong, as well as a *prior* probability distribution,

$$\mathbf{P}_{\text{prior}} = [P_1, P_2, \ldots, P_N]^{\mathrm{T}}, \tag{2.1}$$

   over these classes.

3. A **recognition question**, such as:

   - Which of the N classes does x belong to? (Identification)
   - Does x belong to class H_i or to one of the other $N-1$ classes? (Detection of H_i.)

4. A **cost function**, which quantifies the consequences of wrong decisions.

2.2 Processing

The toolkit is designed to be applicable when the above-listed inputs are processed with the following four steps (of which the toolkit does steps 2 and 3):

1. Extract from input x basic recognition information in the form of multiple scores (not done by this toolkit).

2. Fuse and or calibrate these scores, to obtain the following canonical application-independent representation of the recognition information:

$$\vec{\ell}(x) = \begin{bmatrix} \ell_1(x) \\ \ell_2(x) \\ \vdots \\ \ell_N(x) \end{bmatrix} = \begin{bmatrix} \log P(\vec{\ell}(x)|H_1) - c \\ \log P(\vec{\ell}(x)|H_2) - c \\ \vdots \\ \log P(\vec{\ell}(x)|H_N) - c \end{bmatrix} \tag{2.2}$$

   where c is an unspecified constant. We shall refer to $\vec{\ell}(x)$ as the *log-likelihood-vector*. The calculation of $\vec{\ell}$ is the main purpose of the toolkit and it provides several different ways to achieve this. In the MATLAB code, the variable name `loglh`, for *log-likelihood* is used throughout to denote $\vec{\ell}$. Note:

- $\vec{\ell}$ is in a symmetrical, but redundant form. An asymmetrical, but non-redundant form can be obtained, e.g., by letting $c = P(\vec{\ell}(x)|H_N)$, so that the last component is always zero and then omitting that component. In this toolkit, we work with the symmetrical, redundant form.
- $P(\vec{\ell}(x)|H_N)$ is *not* the same as $P(x|H_N)$. All of the processing between input x and $\vec{\ell}(x)$ leads to some inevitable information-loss. Given this information loss, an optimal representation of the information that remains is in the *well-calibrated* form of equation (2.2).

3. Use the log-likelihoods, $\vec{\ell} = [\ell_1, \ell_2, \ldots, \ell_N]^{\mathrm{T}}$ and the prior, $\mathbf{P}_{\text{prior}} = [P_1, P_2, \ldots, P_N]^{\mathrm{T}}$, to obtain the *posterior* probability distribution, which is an optimal representation of both the prior information and the information $\vec{\ell}(x)$ that was extracted from the input x:

$$\mathbf{P}_{H_i|\vec{\ell}} = \begin{bmatrix} P(H_1|\vec{\ell}) \\ P(H_2|\vec{\ell}) \\ \vdots \\ P(H_N|\vec{\ell}) \end{bmatrix} = \left(\sum_{j=1}^{N} P_j e^{\ell_j} \right)^{-1} \begin{bmatrix} P_1 e^{\ell_1} \\ P_2 e^{\ell_2} \\ \vdots \\ P_N e^{\ell_N} \end{bmatrix}. \tag{2.3}$$

   The posterior may be calculated by the toolkit function `LOGLH2POSTERIOR`.

4. Apply the posterior and the cost-function to make a minimum-expected-cost Bayes decision. For example:

   - To make a minimum-probability-of-error *detection* decision, use:

$$\begin{aligned} &P(H_i|\vec{\ell}) \geq \tfrac{1}{2} \rightarrow \text{ accept } H_i, \\ &P(H_i|\vec{\ell}) \leq \tfrac{1}{2} \rightarrow \text{ reject } H_i. \end{aligned} \tag{2.4}$$

     Caution: In the NIST LRE, a *different* prior is prescribed for detecting each target class H_i. This means that step 3 must be repeated once for every target class. See toolkit function `LRE_DETECTION`.

   - To make a minimum-probability-of-error identification decision, just choose the hypothesis that maximizes the posterior, i.e., make a MAP decision.

These four steps can be grouped as follows:

- Steps 1 and 2 are *application-independent* steps to extract recognition information from the input and to format this information.
- Steps 3 and 4 are *application-dependent* steps to apply this information to make decisions.

2.3 Detection vs Identification

This section may be skipped without loss of continuity by the reader who is interested only in applying the toolkit. It contains some motivation for the design principles of this toolkit.

As noted, this toolkit is applicable to a wide range of different multi-class recognition applications, including both *detection* and *identification*. This section serves as a note to point out that these two flavours of recognition are closely related and that as long as recognition information is represented in log-likelihood form, it does not really matter all that much, whether performance is evaluated by (or optimized for) either one or the other flavour.

2.3.1 Hard decisions

Let us first consider the case of hard decisions. When there are N classes, then:

Identification is to decide: To which of the N classes does the input belong?

Detection of class H_i is to make the binary decision: Does the input belong to class H_i or not?

(When $N = 2$, detection and identification are the same.)

Take the case of three classes H_1, H_2, H_3 as example: There are three ways of writing the identification task as combinations of detection tasks:

1. Decide if it is H_1 or not. If not, decide between the other two.
2. Decide if it is H_2 or not. If not, decide between the other two.
3. Decide if it is H_3 or not. If not, decide between the other two.

For more hypotheses, this recipe is applied recursively and there is a combinatorial explosion of ways to write any N-class identification as combinations of 'one-against-K' detections, where K ranges from 1 to $N - 1$.

2.3.2 Soft decisions

The case of soft decisions is similar. Once the prior information has been incorporated, a soft N-class identification decision is just a (posterior) *probability distribution* over the possible answers, which can be written for the two cases as:

- For **identification** the soft decision is a probability distribution of the form: $[P(H_1), P(H_2), \ldots, P(H_N)]^{\mathrm{T}}$.
- For **detection** of H_i the soft decision is a probability distribution of the form: $[P(H_i), P(\neg H_i)]^{\mathrm{T}}$, where $\neg$ denotes *not* and where

$$P(\neg H_i) = 1 - P(H_i) = \sum_{j \neq i} P(H_j), \tag{2.5}$$

  which gives the relationship in one direction: *The detection posterior can be written in terms of the identification posterior.*

To show that the relationship holds in the other direction as well, consider the 3-class case again: There are three different ways to write the soft identification decision in terms of two soft detection decisions, of which the first is:

$$\begin{bmatrix} P(H_1) \\ P(H_2) \\ P(H_3) \end{bmatrix} = \begin{bmatrix} P(H_1) \\ P(H_2|\neg H_1)P(\neg H_1) \\ P(\neg H_2|\neg H_1)P(\neg H_1) \end{bmatrix}. \tag{2.6}$$

(The other two ways are similar.) Here the RHS has been written in terms of the detection posteriors $[P(H_1), P(\neg H_1)]^{\mathrm{T}}$ and $[P(H_2|\neg H_1), P(\neg H_2|\neg H_1)]^{\mathrm{T}}$. Note that the second posterior is obtained by simply removing the class H_1 from consideration and proceeding as with any other 2-class soft decision.

Applying this recipe recursively, any N-class *identification* posterior can be written[1] as the product of $N-1$ different 'one-against-K' *detection* posteriors, where K ranges from 1 to $N-1$.

2.4 Summary

The toolkit works on the principle of representing general milti-class recognition information as *log-likelihood-vectors*, where there is a likelihood for *each* possible class. If well-calibrated, these likelihoods can be used to make cost-effective Bayes decisions in a wide variety of recognition applications.

[1] Again, there is a combinatorial explosion of different ways to do this. If you need to know exactly how many ways there are, see sequence A109714 in the On-Line Encyclopedia of Integer Sequences: `http://www.research.att.com/~njas/sequences/A109714`.

Chapter 3

Cllr: Objective for Evaluation and Optimization

In the previous chapter, we showed how multi-class recognition information can be represented in *log-likelihood-vector* form. The subject of this chapter is *measurement and optimization of the information content* of these log-likelihood vectors.

In order to optimize information content, we need to be able to measure it. This measurement is implemented in the toolkit by an objective function that we call *multi-class* C_{llr}. This objective serves two useful purposes:

- It is an evaluation objective that researchers can use to judge the performance of their recognizers.
- It serves as optimization objective when the toolkit does supervised training of fusion and calibration coefficients.

3.1 Multi-class Cllr

Let there be N classes and let there be T supervised trials, where for every trail t:

- x_t is the input,
- $c(t) \in \{1, 2, \ldots, N\}$ denotes the true class,
- $\vec{\ell}(x_t)$ is the log-likelihood-vector computed by the evaluee. (Of course, the evaluee sees only x_t, not $c(t)$.)

The multi-class C_{llr} measure of goodness of these log-likelihoods is:

$$C_{\text{llr}} = -\frac{1}{T}\sum_{t=1}^{T} w_t \log_2 P_t \tag{3.1}$$

where P_t is the *posterior probability for the true class* of trial t, as calculated from the given log-likelihoods and the flat prior, $P(H_i) = P_{\text{flat}} = \frac{1}{N}$:

$$P_t = P\big(H_{c(t)}|\vec{\ell}\,(x_t)\big) = \frac{\exp\big(\ell_{c(t)}(x_t)\big)}{\sum_{j=1}^{N}\exp\big(\ell_j(x_t)\big)} \tag{3.2}$$

and where w_t is a weighting factor to normalize the class proportions in the evaluation trials[1]:

$$w_t = \frac{P_{\text{flat}}}{Q_{c(t)}}, \qquad\qquad Q_i = \frac{\text{no. of trials of class } H_i}{T}. \tag{3.3}$$

3.1.1 Properties

- C_{llr} has the sense of *cost*: lower C_{llr} is better.
- C_{llr} has units in *bits of information*.
- $0 \le C_{\text{llr}} \le \infty$.
- $C_{\text{llr}} = 0$ only for perfect recognition, where the posterior for the true class, $P_t = 1$, for *every* trial t.
- $C_{\text{llr}} = \infty$, if $P_t = 0$, for *any* trial t. Keep log-likelihoods finite for finite C_{llr}. More generally, keep log-likelihood magnitudes moderately small for small C_{llr}.
- $C_{\text{llr}} = \log_2 N$ is the *reference value* for a useless, but nevertheless *well-calibrated* recognizer, which extracts no information from the speech, and which acknowledges this by outputting *equal* log-likelihoods: $\ell_1(x_t) = \ell_2(x_t) = \cdots = \ell_N(x_t) = c_t$.
- $C_{\text{llr}} < \log_2 N$ indicates a useful recognizer which can be expected to help to make decisions which have lower average cost than decisions based on the prior alone.
- $C_{\text{llr}} > \log_2 N$ indicates a bad recognizer which can be expected to make decisions which have higher average cost than decisions based on the prior alone.

[1] If there are an equal number of trials of each class, then $w_t = 1$.

3.1.2 Interpretations

For details, see [1].

- $\log_2 N - C_{\text{llr}}$ is the average effective amount of useful information, about the class of the input, that the evaluated recognizer delivers to the user, relative to the information given by the flat prior.

- The logarithmic cost for the posterior, $-\log_2(P_t)$ can be interpreted as an *expected cost* of using the recognizer, over a wide range of different identification and detection applications, and over a wide range of different cost coefficients.

- The logarithmic cost for the posterior, $-\log_2(P_t)$ can be interpreted as an integral, over a wide range of different identification and detection applications, of the *error-rate* of the recognizer.

3.1.3 Implementation in toolkit

See the toolkit function `MULTICLASS_CLLR`.

3.1.4 Relationship to other definitions of Cllr

The objective function C_{llr} was originally proposed in [2] for application in the *two-class* recognition problem of speaker detection and has subsequently been used in NIST SRE 2006[2].

Multi-class C_{llr} is a generalization of the two-class version, in the sense that the two versions are identical when the number of classes, $N = 2$. The multi-class variant was proposed in [3] for use in language recognition and is discussed in more detail in [1].

The reader is cautioned that some confusion may result between the multi-class and two-class versions of C_{llr}:

- The *two-class* version has been specified as alternative evaluation objective in the 2007 NIST Language Recognition Evaluation Plan[3]. The two-class version becomes applicable if the multi-class log-likelihood-vectors are converted to *detection log-likelihood-ratios* (see toolkit function `LOGLH2DETECTION_LLR`). These detection log-likelihood-ratios can be evaluated with two-class C_{llr}. For this purpose, the toolkit provides a

[2]See the SRE'06 Evaluation Plan at: `http://www.nist.gov/speech/tests/spk/2006/sre-06_evalplan-v9.pdf`

[3]See the LRE'07 Evaluation Plan at: `http://www.nist.gov/speech/tests/lang/2007/LRE07EvalPlan-v7e.pdf`

function that computes an *average two-class* C_{llr} over target languages (see toolkit function `AVG_CDET_AND_CLLR`).

- Although this toolkit is aimed at application in NIST LRE'07, we prefer to use the *multi-class* version rather than the averaged two-class version. The multi-class C_{llr} is employed in the all of following toolkit functions:
  - `MULTICLASS_CLLR` (for evaluation)
  - `MULTICLASS_MIN_CLLR` (for evaluation)
  - `CALREF_PLOT` (for evaluation)
  - `TRAIN_NARY_LLR_FUSION` (for fusion and calibration)

3.2 Refinement/calibration decomposition of Cllr

The toolkit further makes available a *refinement/calibration* decomposition of the multi-class C_{llr} metric. The decomposition works like this:

- The **total loss** is the sum of the *calibration loss* and the *refinement loss*. The total loss is just the *raw multi-class* C_{llr} as computed by (3.1).

$$C_{\text{llr}} = \mathsf{calibration\ loss} + \mathsf{refinement\ loss}. \tag{3.4}$$

- The **calibration loss** is the maximum amount by which the total loss can be reduced, by a simple calibration transformation of the log-likelihood-vectors of the evaluee. The calibration is optimized by the evaluator (toolkit function `MULTICLASS_MIN_CLLR`), while using the true class labels.
- The **refinement loss** is the optimized (minimum) value of C_{llr} as obtained with the above optimization of the calibration.

The total loss measures the actual effective performance of the recognizer. The aim of the whole exercise is to make the total loss as low as possible. The decomposition is to help the researcher understand where the problems lie that may be contributing to high total loss. For example, if a recognizer has low refinement loss, but high calibration loss, it indicates that the recognizer output does have the potential for good recognition, but that this potential is not being realized because of a calibration problem, which may be relatively easy to fix.

3.2.1 Calibration transformation

The *calibration transformation* which is used to perform the above-mentioned decomposition is:

$$\vec{\ell}\,'(x_t) = \alpha\vec{\ell}\,(x_t) + \vec{\beta} \tag{3.5}$$

where $\vec{\ell}\,(x_t)$ is the original log-likelihood-vector for trial t; $\vec{\ell}\,'(x_t)$ is the transformed (calibrated) log-likelihood-vector; α is a positive scalar; and $\vec{\beta}$ is an N-vector. Note the the calibration parameters $(\alpha, \vec{\beta})$ are constant for all the trials of an evaluation. This calibration transformation serves a dual purpose in this toolkit:

- As mentioned, it makes possible the refinement/calibration decomposition.
- If a recognizer is found during evaluation by this decomposition to suffer from a calibration problem, then this same transformation may be used to fix that problem. This is done by toolkit functions `TRAIN_NARY_LLR_FUSION` and `APPLY_NARY_LIN_FUSION`.

We can now explicitly specify the refinement and calibration losses in terms of the calibration transformation (3.5):

$$\mathsf{refinement\ loss} = \min_{\alpha,\vec{\beta}} C'_{\text{llr}}, \tag{3.6}$$

$$\mathsf{calibration\ loss} = \max_{\alpha,\vec{\beta}} (C_{\text{llr}} - C'_{\text{llr}}), \tag{3.7}$$

where C'_{llr} is defined by rewriting (3.1) and (3.2) in terms of $\vec{\ell}\,'()$:

$$C'_{\text{llr}} = -\frac{1}{T}\sum_{t=1}^{T} w_t \log_2 P'_t, \tag{3.8}$$

$$P'_t = P\big(H_{c(t)}|\vec{\ell}\,'(x_t)\big). \tag{3.9}$$

3.3 Fusion

Multi-class C_{llr} has a third purpose in the toolkit, namely as logistic regression optimization objective for the *fusion* of multiple recognizers. Let there be K input recognizers, where the kth recognizer outputs its own (not necessarily well-calibrated) log-likelihood-vector $\vec{\ell}_{\,k}(x_t)$ for every trial t. Then

the fused (and calibrated) log-likelihood-vector is:

$$\vec{\ell}\,'(x_t) = \sum_{k=1}^{K} \alpha_k \vec{\ell}_k(x_t) + \vec{\beta} \tag{3.10}$$

This is implemented by toolkit function `APPLY_NARY_LIN_FUSION`. The fusion coefficients may be found as:

$$(\alpha_1, \alpha_2, \ldots, \alpha_K, \vec{\beta}) = \arg\max C'_{\text{llr}}, \tag{3.11}$$

where C'_{llr} is calculated for the fused $\vec{\ell}\,'()$ over a supervised training database.

This fusion is also a calibration, because:

- Equation (3.10) is a generalization of the calibration transformation (3.5), in the sense that these equations are identical for the case $K = 1$.
- There is no need to (i) separately pre-calibrate each input system, or to (ii) post-calibrate the fusion, because of the linearity of the fusion and the calibration: Additionally applying the calibration transformation would give no more genertality than just using the fusion on its own.

The training of fusion (or calibration when $K = 1$) is implemented by toolkit function[4] `TRAIN_NARY_LLR_FUSION`. This function is somewhat more general than equation (3.11), because it has options for (i) allowing $\vec{\beta}$ to remain zero, (ii) regularization penalties to keep coefficient sizes smaller, and (iii) using priors other than P_{flat} to define the optimization objective. For more information, type at the MATLAB prompt:
$\gg$ `help train_nary_llr_fusion`

[4]NARY ≡ multi-class; LLR ≡ Linear Logistic Regression.

Chapter 4

Generative Gaussian backends

This toolkit provides generative Gaussian 'backend' modelling as an alternative to (and a generalization of) the discriminative fusion and calibration of section 3.3. The basic principle of the generative backend is very simple:

- Classes are effectively recognized in score-space, by using (uncalibrated) recognition scores of one or more recognizers as features.
- There is a generative score-vector model for each class, in the form of one multivariate Gaussian probability density per class.
- During training, the parameters (mean-vector and covariance-matrix) of the Gaussians are estimated with the maximum-likelihood (ML) criterion, but subject to certain constraints.
- During recognition, the log-likelihood for each class is just the log Gaussian probability density of the score-vector, given the language.

If output scores of multiple recognizers are stacked into one score-vector, then this backend effectively performs a calibrated fusion of the recognizers. It therefore performs the same function as the discriminative fusion, but the backend is more general because it allows input of a more general form. (It also has many more parameters.)

Since the backend packages can do everything that the fusion package can do, it is up to the user to decide which to use. Under some circumstances, it may even be a good idea to follow the generative backend by discriminative calibration.

There are two main kinds of backend, *linear* and *quadratic*:

1. By assuming *homoscedastic* models (where all models share a common within-class covariance) a *linear* (or more correctly affine) transform results from score-space to log-likelihood-space.

2. *Heteroscedastic* modelling (with possibly different covariances for every class) leads to a *quadratic* backend transform.

4.1 Linear backends

For linear backends, we provide the following choices of covariance regularizing constraints, where the options differ according to how the d-by-d common-within-class covariance, C, is structured:

- The **PPCA** (Probabilistic Principal Component Analysis) covariance model is: $C = \sigma I + WW'$; where $\sigma > 0$; I is the identity matrix of size d; W is d-by-r; and $r < d$. See [6].

- The **FA** (Factor Analysis) covariance model is somewhat more general: $C = D + WW'$; where D is diagonal positive-definite matrix of size d; W is d-by-r; and $r < d$. See [5].

See the reference section 6.3 for more details.

4.2 Quadratic backends

For the quadratic backends, we provide the following choices of covariance regularizing constraints, where the options differ according to how the d-by-d within-class covariance, C_i, for each class i is structured:

- **PPCA**, see above.

- **FA**, see above.

- In **HLDA** (Heteroscedastic Linear Discriminant Analysis):

$$C_i = R' \begin{bmatrix} D & \mathbf{0} \\ \mathbf{0} & K \end{bmatrix} R \tag{4.1}$$

  where R is an orthogonal rotation matrix of size d, D is a positive-definite, *diagonal* matrix of size r; and K is positive-semi-definite of

size $d - r$. Additionally, the mean is also constrained: The mean μ_i, for class i, is constrained so that:

$$\mu_i = R' \begin{bmatrix} \mathbf{m}_i \\ \mathbf{m}_0 \end{bmatrix} \tag{4.2}$$

where $\mathbf{m}_i$ is r-dimensional and *different* for each class, while $\mathbf{m}_0$ is *constant* across classes. See [7, 8]. Under this model, the data can be rotated by R, after which the last $d - r$ components of the data may be discarded. In this reduced and rotated space, the covariances are heteroscedastic, but diagonal.

The quadratic backend additionally allows *smoothing* over the class covariances, where each class covariance is smoothed by convex combination with the average of the covariances. The degree of smoothing is controlled with the combination weight. A weight of 0 means no smoothing, while maximum smoothing with a weight of 1 gives a linear backend again.

See the reference sections 6.4 and 6.5 for more details on quadratic backends.

Part II

Toolkit User Manual

Chapter 5

How to get up and running

The toolkit is a collection of MATLAB m-files, arranged in a number of sub-directories.

1. To install the toolkit, copy these directories to an appropriate location and set your MATLAB path to include all of the directories.

2. To see if everything works and to get a demonstration of what the toolkit does, type at the MATLAB prompt:

   ≫ `multifocal_example1_fusion`

5.1 Software Versions

- This user manual is applicable to FoCal Multi-class Toolkit version 1.0. To see the version of the toolkit, type at the MATLAB prompt: ≫ `help_about_multifocal`

- This toolkit has been tested to run on versions 5 and 7 of MATLAB.

Chapter 6

Package Reference

All of the toolkit functions are written as MATLAB m-files and every function has the usual help comments in the m-file. *These comments form the detailed documentation for each function, which we do not repeat here.* Instead, we give a general overview of the functions and where and how to apply them.

The functions in the toolkit are grouped under a number of subdirectories according to functionality. We shall refer to each of these *groups of MATLAB funtions* as a *package*. The rest of this chapter documents each package in a separate section.

6.1 Multi-class Cllr evaluation

This package provides the primary evaluation mechanisms of this toolkit. It performs 'application-independent' evaluation of multi-class recognition scores that are intended to be calibrated in *log-likelihood-vector* form as defined in equation (2.2). Evaluation is performed on a set of supervised log-likelihood-vector outputs from the system under evaluation.

This package provides the following functions:

- `MULTICLASS_CLLR` for calculating C_{llr} of equation (3.1), which is the primary measure of goodness of a recognizer under evaluation.
- `MULTI_CLASS_MIN_CLLR`, which performs the refinement/calibration decomposition of C_{llr}, as explained in section 3.2.
- `CALREF_PLOT`, which evaluates and graphically compares the outputs of multiple recognizers by plotting the refinement/calibration decompositions in bar-graph form.

See figure 6.1, for an example of the plot produced by this function: There are 5 coloured columns which compare the performance of 5 different 3-class recognizers against the reference value[1] of $\log_2 3$ as represented by the leftmost black column. The user-supplied name of each system is printed under its corresponding column. The total height of each column represents the *total* C_{llr}, the green part represents *refinement loss* and the red part the *calibration loss*.

At the MATLAB prompt, type '`help function_name`' for more information on the above functions.

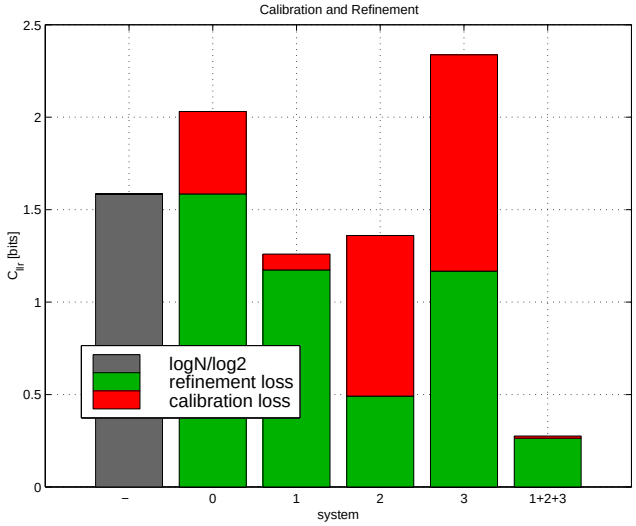

Figure 6.1: Example output of `CALREF_PLOT`

6.2 Fusion

This package performs discriminative (logistic regression) training to calibrate the scores of a single recognizer, or to form a calibrated fusion of the scores of multiple recognizers. Training is performed on a supervised database of scores.

The *scores* that serve as input to the calibration or fusion must be in *log-likelihood-vector* form (as defined by equation (2.2)), but need not be well-calibrated. See section 3.3, for more detail on fusion and calibration.

[1] See section 3.1.1

This package provides the following functions:

- `TRAIN_NARY_LLR_FUSION` for supervised training of calibration or fusion.
- `APPLY_NARY_LIN_FUSION` to apply the fusion to new scores.

At the MATLAB prompt, type '`help function_name`' for more information on the above functions.

6.3 Linear backend

This package provides functions to train and apply (homoscedastic) linear backends to transform general *score*-vectors to *log-likelihood*-vectors.

If output scores of multiple recognizers are stacked into one score-vector, then this backend effectively performs a calibrated fusion of the recognizers and is therefore a generalization of the fusion package. It is more general because is allows input of a more general form. It is up to the user to decide whether to use the discriminative or generative options, or indeed to even chain them.

The linear backend is less general than the quadratic ones and therefore more strongly regularized. It is therefore probably a safer backend to use when training data is scarce.

As noted in chapter 4, the linear backend can be further regularized by assuming structured PPCA or FA covariance models.

This package provides the following functions:

- `TRAIN_LINEAR_BACKEND` for supervised training.
- `APPLY_LINEAR_BACKEND` to apply the backend to new data.
- `COMPOSE_LINEAR_BACKEND` is an auxiliary function, which the uesr will probably not need to call.
- `TRAIN_LDA` This function is not presently used by any other part of this toolkit (and will probably not be needed by the user), but is included to document the relationship between the linear Gaussian backend and linear discriminant analysis (LDA).

At the MATLAB prompt, type '`help function_name`' for more information on the above functions.

6.4 Quadratic backend

This package provides functions to train and apply (heteroscedastic) quadratic backends to transform general *score*-vectors to *log-likelihood*-vectors.

If output scores of multiple recognizers are stacked into one score-vector, then this backend effectively performs a calibrated fusion of the recognizers and is therefore a generalization of the fusion package. It is more general because is allows input of a more general form. It is up to the user to decide whether to use the discriminative or generative options, or indeed to even chain them.

The linear backend is less general than the quadratic ones and therefore more strongly regularized. The linear is therefore probably a safer backend to use when training data is scarce.

As noted in chapter 4, the quadratic backend can be regularized by assuming structured PPCA or FA covariance models, or by smoothing between class covariances.

This package provides the following functions:

- `TRAIN_QUADRATIC_BACKEND` for supervised training.
- `APPLY_QUADRATIC_BACKEND` to apply the backend to new data.

At the MATLAB prompt, type '`help function_name`' for more information on the above functions.

6.5 HLDA backend

This package provides functions to train and apply the specialized HLDA quadratic backend, to transform general *score*-vectors to *log-likelihood*-vectors. This backend uses the HLDA assumption, rather than PPCA or FA to regularize the model estimates.

If output scores of multiple recognizers are stacked into one score-vector, then this backend effectively performs a calibrated fusion of the recognizers and is therefore a generalization of the fusion package. It is more general because is allows input of a more general form. It is up to the user to decide whether to use the discriminative or generative options, or indeed to even chain them.

The linear backend is less general than the quadratic ones and therefore more strongly regularized. The linear is therefore probably a safer backend to use when training data is scarce.

As noted in chapter 4, this HLDA backend can be further regularized by smoothing between class covariances.

This package provides the following functions:

- `TRAIN_HLDA_BACKEND` for supervised training.
- `APPLY_HLDA_BACKEND` to apply the backend to new data.

At the MATLAB prompt, type '`help function_name`' for more information on the above functions. Also see the `HLDA` package for more details on the HLDA implementation.

6.6 Application

This package provides functions to serve as a link between applications and the application-independent information carried by the log-likelihood-vectors used in the rest of this toolkit.

This package provides the following functions:

- `LOGLH2POSTERIOR` Implements equation (2.3), to convert from log-likelihood-vector form to posterior probability form.
- `LOGLH2DETECTION_LLR` Converts from log-likelihood-vector, $\vec{\ell}$, to *detection log-likelihood-ratio*, L_i, where for a given target class H_i, the detection log-likelihood-ratio is:

$$L_i = \log \frac{P(\vec{\ell}|H_i)}{P(\vec{\ell}|\neg H_i)} \tag{6.1}$$

At the MATLAB prompt, type '`help function_name`' for more information on the above functions.

6.7 Nist LRE

This package is specifically to apply this toolkit to the NIST LRE-07 speaker detection evaluation. It provides functions to convert from the log-likelihood-vector format as used in the rest of this toolkit to the required outputs for the LRE language detection, namely detection log-likelihood-ratio confidence scores as well as hard accept/reject decisions. It also

provides 'average C_{det}' and 'average C_{llr}' tools to evaluate these outputs.

This package provides the following functions:

- `LRE_DETECTION` Maps log-likelihood-vector inputs to detection log-likelihood-ratio confidence scores and to hard decisions. Applicable to both closed-set and open-set conditions.
- `AVG_DETECTION_COST` This evaluates decisions or log-likelihood-ratio scores respectively as average C_{det} or average C_{llr}.
- `AVG_CDET_AND_CLLR` This combines the above two functions, calling first the one and then the other. It therefore takes log-likelihood input and calculates average C_{det} and average C_{llr}.
- `CDET_CLLR_PLOT` Bar-graph comparison, in terms of C_{det} and C_{llr} of multiple recognizers.
- `CDET` and `SURPRISAL`, auxiliary functions, not called by the user.

At the MATLAB prompt, type '`help function_name`' for more information on the above functions.

6.8 Examples

This package provides a number of example scripts which serve the following purposes:

- To check that installation is correct.
- To demonstrate the functionality of the toolkit.
- To show how to call and how to use together several of the toolkit functions.
- As test code for the functions in the toolkit.

This package provides the following scripts (they are m-file scripts, not functions):

- `HELP_ABOUT_MULTIFOCAL` Displays versioning and licensing information for this toolkit.

- `MULTIFOCAL_EXAMPLE1_FUSION` This example serves as introduction and demonstration of the toolkit. It starts by generating synthetic training and test data in log-likelihood-vector format, for a number of different hypothetical, 3-class recognizers. The training data has been purposefully made scarce in order to create a difficult fusion problem. It then demonstrates multi-class C_{llr} evaluation of these recognizers as well as a (discriminative) fusion between them. Then it also demonstrates a variety of generative Gaussian backend fusions on the same data.

- `MULTIFOCAL_EXAMPLE2_QUADPPCA` The synthetic data for this example is in general score-vector format (not log-likelihood-vector format). The generative backends are applicable to this data, but not the discriminative fusion. The data is generated according to a *heteroscedastic PPCA* model. A variety of different generative backends are tested on this data.

- `MULTIFOCAL_EXAMPLE3_LINPPCA` The synthetic data for this example is in general score-vector format (not log-likelihood-vector format). The generative backends are applicable to this data, but not the discriminative fusion. The data is generated according to a *homoscedastic PPCA* model. A variety of different generative backends are tested on this data.

- `MULTIFOCAL_EXAMPLE4_LINFA` The synthetic data for this example is in general score-vector format (not log-likelihood-vector format). The generative backends are applicable to this data, but not the discriminative fusion. The data is generated according to a *homoscedastic FA* model. A variety of different generative backends are tested on this data.

- `MULTIFOCAL_EXAMPLE5_HLDA` The synthetic data for this example is in general score-vector format (not log-likelihood-vector format). The generative backends are applicable to this data, but not the discriminative fusion. The data is generated according to an HLDA model. A variety of different generative backends are tested on this data.

- `MULTIFOCAL_EXAMPLE6_FUSION` This is a repeat of the first part of `MULTIFOCAL_EXAMPLE1_FUSION`, with the same types of data and the same discriminative fusion. But in this example, evaluation is done with the function `CDET_CLLR_PLOT`, instead of `CALREF_PLOT`, which was used in the former example. (See section 6.7 and type '`help cdet_cllr_plot`' in MATLAB for more information.)

The above m-file scripts do not take or return any parameters and do not display any help. Just type the script-name at the command-prompt to execute. The user is encouraged to edit these scripts in order to experiment with the toolkit.

The data in all of these examples is *synthetic*, used here just to demonstrate how these tools work. Don't base conclusions about relative merit of the toolkit backends on this data. Instead, use the tools demonstrated here to base conclusions on your own real data.

6.9 Other packages

In addition to the above-documented packages there are a few more packages that will probably not be called directly by the user. These packages include:

- `UTILITIES` General utility functions.
- `COVARIANCE` Functions for making various flavours of covariance estimates.
- `PPCA` Functions for PPCA regularization of covariances.
- `FACTOR_ANALYSIS` Functions for factor-analysis regularization of covariances.
- `HLDA` Functions for training and applying HLDA.
- `DATA_SYNTHESIS` Functions for creating the synthetic data used by the `EXAMPLES` package.

See the comments in the MATLAB source code for more information.

Bibliography

[1] Niko Brümmer, "Measuring, refining and calibrating speaker and language information extracted from speech", Ph.D. dissertation, Stellenbosch University, to be submitted 2007.

[2] N. Brümmer and Johan du Preez, "Application-Independent Evaluation of Speaker Detection", Computer Speech and Language, **20**:2-3, April-July 2006, pp. 230-275.

[3] Niko Brümmer and David van Leeuwen, "On Calibration of Language Recognition Scores", Odyssey 2006.

[4] David van Leeuwen and Niko Brümmer, "Channel-dependent GMM and Multi-class Logistic Regression", Odyssey 2006.

[5] Z. Ghahramani and G. E. Hinton, "The EM algorithm for mixtures of factor analyzers,", Tech. Report CRG-TR-96-1, Univ. of Toronto, 1997. Online: `www.cs.toronto.edu/~hinton/absps/tr-96-1.pdf`

[6] M.E. Tipping and C. Bishop, "Mixtures of Probabilistic Principoal Component Analysers", Neural Conmputation 11(2), pp. 443-482, 1999. Online: `http://citeseer.ist.psu.edu/13839.html`

[7] N. Kumar, "Investigation of Silicon-Auditory Models and Generalization of Linear Discriminant Analysis for Improved Speech Recognition", Ph.D. Thesis, John Hopkins University,Baltimore, 1997.

[8] M.J.F. Gales, "Semi-tied covariance matrices for hidden Markov Models", IEEE Transaction Speech and Audio Processing, vol. 7, pp. 272-281, 1999.