IDIAP COMMUNICATION REPORT

SCORETOOLKIT DOCUMENTATION

André Anjos Sébastien Marcel

Idiap-Com-02-2012

Version of APRIL 20, 2012

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

ScoreToolKit Documentation

Release 1.0.2

André Anjos and Sébastien Marcel, Idiap Research Institute

April 20, 2012

CONTENTS

The Toolkit is conceived for these purposes:

1. Plot the DET curve for a particular system
2. Check the consistency between score files w.r.t. the filenames scores refer to

CHAPTER

ONE

INSTALLATION

To install from the command line on a machine you have access to the python installation tree (e.g., on a Windows machine):

```
$ easy_install trstk

# or

$ pip install trstk
```

If you don't have adminstrative rights on the Python installation directory, you can create an isolated virtual environment using *virtualenv*. Follow instructions there to download and create a virtual environment and then either *easy_install* or *pip install* this package.

Our PyPI page also contains a link to a Windows graphical installer. Unfortunately, it does not install the package dependencies like the command line installer does. You have to do it yourself. Here is the dependencies list:

- NumPy
- Matplotlib

Visit those webpages for more information.

CHAPTER

TWO

INPUT

Tools in this package accept score files in one single textual format. Each line in the file refers to one single sample in the database being analyzed. Each line is composed of 4 fields separated by spaces in this order:

1. Claimed identity: a string that defines the claimed identity of the subject being analyzed
2. Model label: contains a label/reference to the data used to make the model (filename <id>d<capture_number> used to make the model)
3. Real identity: a string that defines the real identity of the subject being analyzed (i.e. the output of the classification)
4. Test label: contains a label/reference to the data used to do the testing (filename <id>d<capture_number> of the test file)
5. Score: a floating-point value representing the score

Each of the above-mentioned fields **cannot have spaces in between**. Failing to comply will make the tools emit syntax errors pointing to the location in the file where problems seem to occur.

Here is a valid example score file:

```
02463 02463d547 02463 02463d653 0.623265
02463 02463d547 02463 02463d655 0.920861
02463 02463d547 02463 02463d657 0.938942
02463 02463d547 02463 02463d659 0.743715
02463 02463d547 02463 02463d661 0.397660
02463 02463d547 02463 02463d663 0.615722
02463 02463d547 02463 02463d665 0.613291
02463 02463d547 02463 02463d667 0.543184
02463 02463d547 02463 02463d669 0.829777
02463 02463d547 02463 02463d671 0.869681
02463 02463d547 02463 02463d673 0.806394
02463 02463d547 02463 02463d675 1.007791
02463 02463d547 04200 04200d75 0.257423
```

Here is an invalid example score file:

```
1 Bob_Jones bob-file-001 Bob_Jones bob-file-004 -37.643410
2 Susan Smith susan-file-001 Susan Smith susan-file-001 -33.393433
3 Joe joe-file-030 Joe joe-file-001 -72.295616
```

In this case, line 2 above will fail because the real identity field and the claimed identity fields contain spaces. Lines 1 and 3 do conform to the proposed scheme and will be parsed without problems.

2.1 Multi-modality Input

If you have multiple modalities you should build a single text file along the lines explained before, for each modality. The order of the *tags* within each file should be respected. Example *Hypothetical face verification experiment output*:

```
02463 02463d547 02463 02463d675 1.007791
02463 02463d547 04200 04200d75 0.257423
02463 02463d547 04201 04201d435 0.315074
02463 02463d547 04201 04201d437 0.347413
02463 02463d547 04201 04201d439 0.296383
02463 02463d547 04201 04201d443 0.371881
02463 02463d547 04201 04201d445 0.260964
```

Hypothetical speech verification experiment output:

```
02463 02463d547 02463 02463d675 0.9932
02463 02463d547 04200 04200d75  0.0027
02463 02463d547 04201 04201d435 0.0144
02463 02463d547 04201 04201d437 0.0159
02463 02463d547 04201 04201d439 0.1250
02463 02463d547 04201 04201d443 0.0031
02463 02463d547 04201 04201d445 0.0002
```

A set of working examples is included in the `example` directory of this package.

CHAPTER

THREE

DEPENDENCIES

To properly run the software in this package you must have the following packages installed:

- Python: is the scripting language used for the programs
- Matplotlib: is used for plotting
- Sphinx: if you need to *recompile* the documentation

CHAPTER

FOUR

USAGE

We describe a few scenarios for using the Toolkit in specific cases. In Section *API* we exemplify how to create your own scripts that can re-use the readout functionality available in the kit.

4.1 Example 1: Plotting a DET Curve

The following command will plot a single DET curve for a given input score file:

```
$ plotDET.py test.scores
```

This command should produce a single plot in PDF file named `det.pdf` calculated using the contents of the input score file `test.scores`. The plot title will be empty. You can change the output filename and its type (we support either *.png* files or *.jpg*) or add a plot title like this:

```
$ plotDET.py --title="My Test DET" --output=test.png test.scores
```

You can plot a series of overlayed DET curves in the following manner:

```
$ plotDET.py --title="My Test DET" --output=overlayed.pdf \
    --label=devel development.scores --label=test test.scores
```

This command will produce a single plot in a PDF file, with the overlayed DET curves generated using each of the score files given as input parameters. A legend will be drawn at a convenient location in the plot using the labels for each of the curves as determined by your input. By default the program generates black-and-white plots, but can be instructed to produce coloured plots using the `--colour` option (see `plotDET.py --help` message).

4.2 Example 2: Checking score set consistency

You can check the consistency between two (or more) score sets that are supposed to provide scores for multiple biometric modalities using the `checkModalities.py` script. This tool will compare two input files and will stop on the first error it finds:

```
$ checkModalities.py faceverif.scores speechverif.scores
```

If you sort all files before calling the program, huge score files can be checked in a much faster way as we will avoid the sorting step within the program. You can do this using the `sort` and `uniq` unix utilities to sort all score files before using `checkModalities.py` like this:

```
$ sort my-scores.txt | uniq > sorted-scores.txt
$ sort other-scores.txt | uniq > other-sorted-scores.txt
$ checkModalities.py --sorted sorted-scores.txt other-sorted-scores.txt
```

4.3 Example 3: Plotting a scores distribution

You can plot joint score distributions including impostors, clients and attacks using the *plotScores.py* script. to do so:

```
$ plotScores.py --title="My Score Distribution" --output=test.png legit.txt attack.txt
```

The input is expected to be divided among 2 files that contain the results of the baseline verification evaluation for the legit protocol and for the spoofing attack protocol. The routine will draw 3 histograms. The first 2 correspond to the clients and impostor groups found on the first file. The second histogram corresponds to the attacks found on the second file.

CHAPTER

FIVE

API

You can re-use part of the functionality of this code to input data into your own python scripts for fusing scores or any other task you might need to achieve. To do so, you need to be able to import the `trstk` library into your script. You should set the path leading to *trstk* so the python interpreter knows how to find it. There are two basic ways to do this.

1. Set the environment variable `$PYTHONPATH` to point to the directory *containing* the `trstk`:

   ```
   $ vim ./myScriptThatUsesStk.py #create the script
   $ export PYTHONPATH=/path/to/ScoreToolKit
   $ python ./myScriptThatUsesStk.py
   ```

2. Or, you can change directories to the ScoreToolKit root directory and have your scripts inside that directory. Python automatically searches the current working directory for imports:

   ```
   $ cd /path/to/ScoreToolKit
   $ vim ./myScriptThatUsesStk.py
   $ python ./myScriptThatUsesStk.py
   ```

5.1 Reference Manual

The ScoreToolKit (or simply "stk") provides functionality to load TABULA RASA conformant score files, for either plotting DET curves or for the validation of multi-file score matching.

`trstk.`**`checkModalities`**(*data1*, *filename1*, *data2*, *filename2*, *presorted=False*)

Double-checks score files for fusion.

This method checks two score files to make sure they match w.r.t. to the number of clients, imposter and models. It is equivalent to making sure the first 4 columns of such files contain the same fields, after ordering.

Parameters:

data1 The pre-loaded data set using load_file()

filename1 The first score file name (string)

data2 The (second) pre-loaded data set using load_file()

filename2 The second score file name (string)

presorted A flag indicating if the files have been pre-sorted (boolean)

Here is how to sort your score files using shell utilities `sort` and `uniq`:

```
$ sort my-scores.txt | uniq > my-sorted-scores.txt
```

Returns None.

trstk.**evalDET**(*negatives*, *positives*, *points*)
Evaluates the DET curve.

This method evaluates the DET curve given a set of positives and negatives, returning two numpy arrays containing the FARs and the FRRs.

trstk.**evalROC**(*negatives*, *positives*, *points*)
Evaluates the ROC curve.

This method evaluates the ROC curve given a set of positives and negatives, returning two numpy arrays containing the FARs and the FRRs.

trstk.**farfrr**(*negatives*, *positives*, *threshold*)
Calculates the FAR and FRR for a given set of positives and negatives and a threshold

trstk.**load_file**(*filename*, *no_labels=False*)
Loads a score set from a single file to memory.

Verifies that all fields are correctly placed and contain valid fields.

Returns a python list of tuples containg the following fields:

[0] claimed identity (string)

[1] model label (string)

[2] real identity (string)

[3] test label (string)

[4] score (float)

trstk.**plotDET**(*negatives*, *positives*, *filename='det.pdf'*, *points=100*, *limits=None*, *title='DET Curve'*, *labels=None*, *colour=False*)
Plots Detection Error Trade-off (DET) curve

Keyword parameters:

positives numpy.array of positive class scores in float64 format

negatives numpy.array of negative class scores in float64 format

filename the output filename where to save the plot. If not specified, we output to 'det.pdf'

points an (optional) number of points to use for the plot. Defaults to 100.

limits an (optional) tuple containing 4 elements that determine the maximum and minimum values to plot. Values have to exist in the internal desiredLabels variable.

title an (optional) string containg a title to be inprinted on the top of the plot

labels an (optional) list of labels for a legend. If None or empty, the legend is suppressed

colour flag determining if the plot is coloured or monochrome. By default we plot in monochrome scale.

trstk.**plotScores**(*data*, *filename='scores.pdf'*, *bins=None*, *limits=None*, *title=None*, *labels=None*, *colour=False*)
Plots the score distributions considering a population of impostors, clients and attacks.

Parameters:

data An iterable organized in the following way:

impostors The scores for the impostors (real-access, non-matching sample x model)

positives The scores for the clients (real-access, matching sample x model)

attacks The scores for the attacks that will be overlaid (matching attacked id sample x model)

filename The output filename for the plot

bins The number of bins to have for each of the bar plots (negatives, positives, attacks). If None, use the defaults.

limits The limits to be applied to the plot (leave the default value for the default behaviour)

title The plot title

labels A tuple with the labels associated with each score distribution

colour Shall I use color to generate the plot?

`trstk.`**`split`**(*data*)

Splits the input tuple list (as returned by load_file()) into positives and negative scores.

Returns 2 numpy arrays as a tuple with (negatives, positives)