BEAT

Biometrics Evaluation and Testing

http://www.beat-eu.org/

Funded under the 7th FP (Seventh Framework Programme)

Theme SEC-2011.5.1-1
[Evaluation of identification technologies, including Biometrics]

D3.4: Metrics for the evaluation of biometric performance

Due date: 30/09/2013 **Submission date:** September 27, 2013
Project start date: 01/03/2012 **Duration:** 48 months
WP Manager: Julian Fierrez **Revision:** 0

Author(s): Javier Galbally (UAM), Julian Fierrez (UAM), Chi Ho Chan (UNIS), Norman Poh (UNIS), Josef Kittler (UNIS)

Project funded by the European Commission in the 7th Framework Programme (2008-2010)		
Dissemination Level		
PU	Public	Yes
RE	Restricted to a group specified by the consortium (includes Commission Services)	No
CO	Confidential, only for members of the consortium (includes Commission Services)	No

D3.4: Metrics for the evaluation of biometric performance

Abstract:

The present deliverable contains the documentation related to the use of the software functions developed to implement the different performance evaluation metrics described in D3.3. The functions have been implemented to be fully compatible with the BEAT platform so that they can be used within the final outcome tool of the project in order to carry out practical performance evaluations. Each function is accompanied by a link to its specific source code (in python). Therefore, this deliverable is fully practical and requires some programming background from the reader. The deliverable also represents a key contribution to the prototyping and development work to be carried out in WP7 where further documentation and manuals will be generated for the practical use, installation and deployment of the platform.

Contents

1 Introduction

This deliverable is an inventory of a number of software functions which implement the different performance evaluation metrics previously described in D3.3, both for the verification and the identification operation modes.

Therefore, in this deliverable, and in the specified links for each metric, we only provide the definition, usage information and source code of the functions which implement those metrics. As such, these document should be understood as a pure practical and technical support for those people who want to make use these software tools.

Although some basic concepts about the different biometric operating modes are given throughout the deliverable, this is only done for completeness and review purposes. For a clear and thorough explanation of the different types of errors and evaluation metrics the reader is always referred to D3.3.

In the two sections of the deliverable Sect. 2 and Sect. 3 (one dedicated to verification metrics and the other to identification metrics) the reader will find:

- Documentation related to the functions that implement the performance measures described in D3.3. This documentation includes: The objective of the function, expected input parameters, output of the function, and some details about specific software implementation in case some reference to other external functions is required.
- A link specifying the webpage where this function and its actual source code may be found. Due to the length of these links and in order to make them more easily readable, all of them start with the code COMMON-URL, which corresponds to:

  http://www.idiap.ch/software/bob/docs/releases/last/sphinx/html/measure/generated

  Therefore to reach each function, this COMMON-URL has to be concatenated prior to the specific link given with the function.

All functions have been programmed in the framework provided by the general computer vision and machine learning toolbox BOB, as it is fully compatible in terms of programming and interfaces with the BEAT platform:

http://idiap.github.io/bob/

The deliverable is not intended as a programming manual (further documentation and manuals for users will be generated within WP7 in order to install and get started with the BEAT platform). Therefore, for this deliverable to be fully useful for potential readers it is very advisable to have some background knowledge on: *i*) general programming concepts and specifically in Python; *ii*) some experience using the BOB toolbox.

2 Verification Metrics

For completeness and as a form of review, we present here the basic notions about the verification operation mode. For further details and a deeper understanding of this operation

mode and the most widely used metrics to evaluate it, we refer the reader to D3.3.

Recall that, as specified in D3.3, a biometric system working under the verification mode receives two inputs: a biometric sample (usually referred to as *test sample*) and an identity claim. The system task is to compare the test sample to a previously stored sample associated to the identity claim (typically in the enrollment stage) and determine if both samples belong to the same person. Therefore, the question posed to the system is: Is this person who he claims to be? With only a binary answer possible (yes or no). As such, two types of error con occur: *i*) *false rejection* or *false non-match*, that is, falsely rejecting a genuine's user claim; or *ii*) *false acceptance* or *false match*, that is, falsely accepting the claim to be from the genuine user when the actual person is an impostor.

The functions defined in this section are designed to implement the different common metrics used to evaluate the performance of biometric systems under the verification operation mode, that is, to estimate the two types of errors committed by the system and specified above.

2.1 FMR and FNMR

As specified in D3.3, the metrics False Match Rate (FMR) and False Non-Match Rate (FNMR) refer to error rates strictly of the matcher, while the FAR and FRR are performance metrics of the whole system where, in addition to the FMR and FNMR (which are responsible for most errors in a biometric system), there are other possible fail causes also considered such as the Failure to Enroll (FtE) or Failure to Acquire (FtA).

However, as these last errors (FtE and FtA) are very difficult to estimate in a laboratory evaluation where the database has already been acquired and is ready to be used for matching, in practice, in the vast majority of cases in the literature and technical evaluations, the False Match Rate (FMR) and the False Acceptance Rate (FAR) are used interchangeably. The same occurs with the False Non-Match Rate (FNMR) and the False Rejection Rate (FRR). In general, the terms FAR and FRR are far more used than their respective counterparts FMR and FNMR.

Given that, in practice, both pair of metrics are equivalent for the type of evaluations that will be carried out on the BEAT platform, only one of them has been implemented under the name FAR and FRR, as defined in Sect. 2.2.

2.2 FAR, FRR and EER

`bob.measure.eer_rocch(negatives, positives)`

Calculates the equal-error-rate (EER) given the input data. It returns a float with the value of the EER.

Input arguments:

- `negatives`: an array containing the score information for samples that are labeled to belong to the class usually known as "impostor".

- `positives`: an array containing the score information for samples that are labeled to belong to the class usually known as "client".

Link to source code: COMMON-URL/bob.measure.eer_rocch.html#bob.measure.eer_rocch

`bob.measure.farfrr(negatives, positives, threshold)`

Calculates the FA ratio and the FR ratio given positive and negative scores and a threshold.

Input arguments:

- `negatives`: an array containing the score information for samples that are labeled to belong to the class usually known as "impostor".
- `positives`: an array containing the score information for samples that are labeled to belong to the class usually known as "client".
- `threshold`: a float indicating the threshold (score) value where we want to compute the FAR and FRR.

It is expected that positive scores are greater than negative scores. So, every positive value that falls bellow the threshold is considered a false-rejection (FR). Negative samples that fall above the threshold are considered a false-accept (FA).

Positives that fall on the threshold (exactly) are considered correctly classified. Negatives that fall on the threshold (exactly) are considered incorrectly classified.

The threshold value does not necessarily have to fall in the range covered by the input scores (negatives and positives altogether), but if it does not, the output will be either (1.0, 0.0) or (0.0, 1.0) depending on the side the threshold falls.

The output is in form of two double-precision real numbers. The numbers range from 0 to 1. The first element of the pair is the false-accept ratio. The second element of the pair is the false-rejection ratio.

It is possible that in some setups the designer may have setup the system so that positive samples have a smaller score than the negative ones. In this case, make sure you normalize the scores so that positive samples have greater scores before feeding them into this function.

Link to source code: COMMON-URL/bob.measure.farfrr.html#bob.measure.farfrr

2.3 ROC and DET curves

`bob.measure.roc(negatives, positives, n_points)`

Calculates the ROC curve given a set of positive and negative scores and a desired number of points. Returns a two-dimensional array of doubles that express the X (FRR) and Y

(FAR) coordinates in this order. The points in which the ROC curve are calculated are distributed uniformly in the range [min(negatives, positives), max(negatives, positives)].

Input arguments:

- `negatives`: an array containing the score information for samples that are labeled to belong to the class usually known as "impostor".
- `positives`: an array containing the score information for samples that are labeled to belong to the class usually known as "client".
- `n_points`: an integer indicating the number of points where the FAR and FRR values will be computed.

Link to source code: COMMON-URL/bob.measure.roc.html#bob.measure.roc

`bob.measure.plot.roc(negatives, positives, npoints=100, CAR=False, kwargs)`

Plots Receiver Operating Charactaristic (ROC) curve.

This method will call matplotlib to plot the ROC curve for a system which contains a particular set of negatives (impostors) and positives (clients) scores.

The plot will represent the false-alarm on the vertical axis and the false-rejection on the horizontal axis.

The standard `matplotlib.pyplot.plot` command is used. All parameters passed with exception of the three first parameters of this method will be directly passed to the plot command. For further reference on these options please go to:
http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot

Input arguments:

- `negatives`: Array of negative class scores (impostor).
- `positives`: Array of positive class scores (client).
- `npoints`: Number of points to use when drawing the ROC curve CAR plot
- `CAR`: CAR over FAR in semilogx axis (CAR=True) or FAR over FRR linearly (CAR=False, the default)
- `kwargs`: a dictionary of extra plotting parameters, that is passed directly to matplotlib.pyplot.plot().

Note: This function does not initiate and save the figure instance, it only issues the plotting commands. Every user is responsible for setting up and saving the figure as it best fits his purposes.

Link to source code: COMMON-URL/bob.measure.plot.roc.html#bob.measure.plot.roc

bob.measure.det(negatives, positives, n_points)

Calculates the DET curve given a set of positive and negative scores and a desired number of points.
Returns a two-dimensional array of doubles that contain: X axis values in the normal deviate scale for the false-rejections, Y axis values in the normal deviate scale for the false-accepts.

Input arguments:

- `negatives`: Array of negative class scores (impostor).
- `positives`: Array of positive class scores (client).
- `n_points`: Number of points where the DET curve will be computed.

Link to source code: COMMON-URL/bob.measure.det.html#bob.measure.det

bob.measure.plot.det(negatives, positives, npoints=100, axisfontsize='x-small', kwargs)

Plots Detection Error Trade-off (DET) curve as defined in the paper:

Martin, A., Doddington, G., Kamm, T., Ordowski, M., and Przybocki, M., "The DET curve in assessment of detection task performance," in *Proc. European Conf. on Speech Communication and Technology (ECSCT)*, pp. 1895-1898, 1997.

The plot will represent the false-alarm on the vertical axis and the false-rejection on the horizontal axis.

This method will call matplotlib to plot the DET curve for a system which contains a particular set of negatives (impostors) and positives (clients) scores.

The standard `matplotlib.pyplot.plot` command is used. All parameters passed with exception of the three first parameters of this method will be directly passed to the plot command. For further reference on these options please go to:
http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot

Input arguments:

- `negatives`: Array of negative class scores (impostor).

- `positives`: Array of positive class scores (client).
- `npoints`: Number of points to use when drawing the DET curve.
- `axisfontsize`: The size to be used by x/y tick labels to set the font size on the axis.
- `kwargs`: A dictionary of extra plotting parameters, that is passed directly to `matplotlib.pyplot.plot`.

Note: This function does not initiate and save the figure instance, it only issues the plotting commands. Every user is responsible for setting up and saving the figure as it best fits his purposes.

Link to source code: COMMON-URL/bob.measure.plot.det.html#bob.measure.plot.det

`bob.measure.plot.det_axis(v, kwargs)`

Sets the axis in a DET plot.

This method wraps the `matplotlib.pyplot.axis` by calling `bob.measure.ppndf` on the values passed by the user so they are meaningful in a DET plot as performed by `bob.measure.plot.det`.

Input arguments:

- `v`: Python iterable (list or tuple) with the X and Y limits in the order (xmin, xmax, ymin, ymax). Expected values should be in percentage (between 0 and 100%). If `v` is not a list or tuple that contains 4 numbers it is passed without further inspection to `matplotlib.pyplot.axis`.
- `kwargs`: All remaining arguments will be passed to `matplotlib.pyplot.axis` without further inspection.

Link to source code: COMMON-URL/bob.measure.plot.det_axis.html#bob.measure.plot.det_axis

2.4 EPC curve

`bob.measure.epc(dev_negatives, dev_positives, test_negatives, test_positives, n_points)`

Calculates the EPC curve given a set of positive and negative scores and a desired number of points. Returns a two-dimensional array of doubles that express the X (cost) and Y coordinates in this order.

Please note that, in order to calculate the EPC curve, one needs two sets of data comprising a development set and a test set. The minimum weighted error is calculated

on the development set and then applied to the test set to evaluate the Half-Total Error Rate (HTER) in that threshold.

The EPC curve plots the HTER on the test set for various values of the score. For each value of the score, a threshold is found that provides the minimum weighted error on the development set. Each threshold is consecutively applied to the test set and the resulting HTER values are plotted in the EPC curve.

The cost points in which the EPC curve are calculated are distributed uniformly in the range [0.0, 1.0].

Input arguments:

- `dev_negatives`: Array of negative class scores (impostor) for the development set.
- `dev_positives`: Array of positive class scores (client) for the development set.
- `test_negatives`: Array of negative class scores (impostor) for the development set.
- `test_positives`: Array of positive class scores (client) for the development set.
- `npoints`: Number of points where the EPC values are computed.

Link to source code: COMMON-URL/bob.measure.epc.html#bob.measure.epc

`bob.measure.plot.epc(dev_negatives, dev_positives, test_negatives, test_positives, npoints=100, kwargs)`

Plots Expected Performance Curve (EPC) as defined in the paper:

Bengio, S., Keller, M., Marithoz, J., "The Expected Performance Curve," in *Proc. Int. Conf. on Machine Learning (ICML) Workshop on ROC Analysis in Machine Learning*, pp. 19631966, 2004.

This method will call matplotlib to plot the EPC curve for a system which contains a particular set of negative scores (impostors) and positive scores (clients) for both the development and test sets.

The plot will represent the minimum HTER on the vertical axis and the cost on the horizontal axis.

This method will call matplotlib to plot the DET curve for a system which contains a particular set of negatives (impostors) and positives (clients) scores.

The standard `matplotlib.pyplot.plot` command is used. All parameters passed with exception of the three first parameters of this method will be directly passed to the plot command. For further reference on these options please go to:
http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot

Input arguments:

- `dev_negatives`: Array of negative class scores (impostor) for the development set.
- `dev_positives`: Array of positive class scores (client) for the development set.
- `test_negatives`: Array of negative class scores (impostor) for the development set.
- `test_positives`: Array of positive class scores (client) for the development set.
- `npoints`: Number of points where the EPC values are computed and drawn.
- `kwargs`: A dictionary of extra plotting parameters, that is passed directly to `matplotlib.pyplot.plot`.

Note: This function does not initiate and save the figure instance, it only issues the plotting commands. Every user is responsible for setting up and saving the figure as it best fits his purposes.

Link to source code: COMMON-URL/bob.measure.plot.epc.html#bob.measure.plot.epc

3 Identification Metrics

For completeness and as a form of review, we present here the basic notions about the identification operation mode. For further details and a deeper understanding of this operation mode and the most widely used metrics to evaluate it, we refer the reader to D3.3.

Recall that, as specified in D3.3, a biometric system working under the identification mode receives only one input: a biometric sample (usually referred to as *test sample*). The system task is to compare the test sample to a database of stored samples and determine which of the samples in the database (if any) comes from the same individual as that of the test sample. Two different scenarios can be distinguished within the identification mode: *i*) *closed-set* when the person producing the test sample is assumed or known to be in the database; *ii*) *open-set* when the test person may not have been enrolled in the database.

In any case, the question posed to the system is: Who is the person presenting the test sample? With multiple possible answers: Any of the persons previously enrolled in the database, or "do not know" in case the subject is not in the gallery. As such, in this case the output of the systems is typically a ranked list of users from the database which are most likely to be the test-subject.

Therefore the error of the system is in general computed according to whether the correct identity appears or not among the first k identities in the ranked output list.

The functions defined in this section are designed to implement the different common metrics used to evaluate the performance of biometric systems under the identification operation mode.

3.1 Closed set: CMC curves

bob.measure.cmc(cmc_scores)

Calculates the cumulative match characteristic (CMC) from the given input.

For each test item the probability that the rank r of the positive score is calculated. The rank is computed as the number of negative scores that are higher than the positive score. If several positive scores for one test item exist, the highest positive score is taken. The CMC finally computes, the number of test items that have a rank r or higher.

Input arguments:

- `cmc_scores`: List of two-element tuples. Each of the tuples contains the negative and the positive scores for one test item..

Link to source code: COMMON-URL/bob.measure.cmc.html#bob.measure.cmc

bob.measure.plot.cmc(cmc_scores, logx=True, kwargs)

Plots the (cumulative) match characteristics curve and returns the maximum rank.

The standard `matplotlib.pyplot.plot` command is used. All parameters passed with exception of the three first parameters of this method will be directly passed to the plot command. For further reference on these options please go to:
http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot

Input arguments:

- `cmc_scores`: List of two-element tuples. Each of the tuples contains the negative and the positive scores for one test item.
- `logx`: Boolean input, if it is true, the x-axis is in log scale.
- `kwargs`: a dictionary of extra plotting parameters, that is passed directly to textttmatplotlib.pyplot.plot.

Note: This function does not initiate and save the figure instance, it only issues the plotting commands. Every user is responsible for setting up and saving the figure as it best fits his purposes.

Link to source code: COMMON-URL/bob.measure.plot.cmc.html#bob.measure.plot.cmc

3.2 Open set: DIR and FAR

bob.measure.plot.dir(cmc_scores, far_list, logx = True, kwargs)

Plots the Detection and Identification Rate from the given input and a vector of specified false acceptance rates.

The standard `matplotlib.pyplot.plot` command is used. All parameters passed with exception of the three first parameters of this method will be directly passed to the plot command. For further reference on these options please go to:
http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot

Input arguments:

- `cmc_scores`: List of two-element tuples. Each of the tuples contains the negative and the positive scores for one test item.
- `far_list`: Array of predefined false acceptance rates.
- `logx`: Boolean input, if it is true, the x-axis is in log scale.
- `kwargs`: a dictionary of extra plotting parameters, that is passed directly to textttmatplotlib.pyplot.plot.

Note 1: This function does not initiate and save the figure instance, it only issues the plotting commands. Every user is responsible for setting up and saving the figure as it best fits his purposes.

Note 2: At the moment of the deliverable submission this function is still going through the mandatory error-checking process in order to verify that it is fully compatible and consistent with the BEAT platform and the rest of implemented functions (in terms of inputs and outputs, naming conventions, etc.) The link given below is still not functional at the time of the deliverable release, but will be available before the deadline of milestone MS32 (April 2014).

Future link to source code: COMMON-URL/bob.measure.plot.cmc.html#bob.measure.plot.dir

4 Conclusions

The deliverable has presented all the practical information concerning the use and implementation of the number of software functions that have been developed to compute the biometric performance metrics described in D3.3. For each function its inputs and outputs were specified as well as a link to the actual source code (in Python). The reader may also find specifications related to any other relevant information about external functions used in their coding or any other relevant issues for their usage. The deliverable should be understood as a pure practical tool and the reader is referred to D3.3 for further details about the implemented metrics or to the outcomes of WP7 for instructions and user manuals of the BEAT platform were these software functions will be integrated.