ENCM 509

Fundamentals of Biometric Systems Design

Department of Electrical and Software Engineering

Schulich School of Engineering

Laboratory Project #2

Face Recognition

1 Introduction

This lab project is to investigate and report the results of face recognition using feature extraction and classification procedures.

There are 2 variations of this project. For your Lab Project, choose one:

a) Face recognition in Python using LBP, Neural Networks and SVM (comparative analysis).

b) Face recognition in Python using BoB biometrics package (Linux or Mac platform).

---

2 Project Report

In this project, a face recognition and analysis of the results will be performed using the following procedures:

- face detection,
- facial features extraction,
- classification, and
- analysis of the results.

The total Project mark is 20. The lab report shall include:

- Description of the implemented project (introduction, procedure and analysis, conclusion) (6 marks).
- The project flow block-diagram, illustrations, graphs (such as DET, ROC) and their analysis (6 marks).
- Code or any modifications to the code (8 marks).

3 Pre-processing using OpenCV library (Python or C++)

You can implement Face detection in Python language using OpenCV library. The following code load the face image and convert it to gray scale:

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')


img = cv2.imread('YourImage.jpg')
img_gray = cv2.cvtColor(img, cv2.color_bgr2gray)
```

Next the Haar-Cascades is used for the face detection:

```
face_box = face_cascade.detectMultiScale(img_gray, 1.1, 4)
```

Here, 1.1 is a scale factor, and 4 is the number of neighbours to retain for the rectangular around the face.

An alternative approach to the frontal face cascade approach is LPB-based Haar cascades: https://github.com/opencv/opencv/tree/master/data/lbpcascades Two good full-run examples of face detection can be found here:

https://www.datacamp.com/community/tutorials/face-detection-python-opencv#face-detection

and

https://scikit-image.org/docs/dev/auto_examples/applications/plot_face_detection.html

For a lot of the project, you may want to create your own custom load function. Here is an example:

```
import cv2
import numpy as np
from glob import glob
# custom load_data function to load images and record subject labels
def get_data(path):
        paths = glob(path, recursive=True)

        data = [] #list of images
        label = [] #list of labels

        for path in paths:
                img = cv2.imread(path,0) # read image
                subject_label = path[path.rfind('s')+1:path.rfind('\\')] #extract the label

                # pre-processing step
                # can resize, rescale, normalize
                img = cv2.resize(img,(1,len(img)*len(img[0]))) # reshape image to a 1D vector
                img = np.float32(np.array(img)/255.0)   #normalize to 0-1 value

                # can apply LBP, PCA or other forms of feature extraction

                # append images and labels
                data.append(img)
                # decrease all labels by 1 since subject_labels start from 1
                label.append(int(subject_label)-1)

        return np.array(data)[:,:,0], np.array(label)
```

For classification (face recognition) with SVM, you can still use the version implemented in OpenCV; however, you will not get the probability values required to do the analysis required. Check the following link for the documentation and sample code:
https://docs.opencv.org/4.x/d0/dcc/tutorial_non_linear_svms.html

```
#ORL dataset contains 40 subjects with 10 images per subject
#5 images are used for training and the remaining 5 images are used for testing
train_path = r'ORL\\Train_Data\\*\\*'
test_path = r'ORL\\Test_Data\\*\\*'

train_data, train_label = get_data(train_path)
test_data, test_label = get_data(test_path)

svm = cv2.ml.SVM_create()
svm.setType(cv2.ml.SVM_C_SVC)
svm.setKernel(cv2.ml.SVM_LINEAR)
svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 100, 1e-6))

svm.train(train_data, cv2.ml.ROW_SAMPLE, train_label)
prediction = svm.predict(test_data)[1]

result = prediction[:,0] == test_label
accuracy = np.sum(result)/len(result)
print("Accuracy", accuracy)
```

4 Subproject 1 - Face recognition in Python using LBP, Neural Networks and SVM (comparative analysis)

The LBP is applied to an image using the function `local_binary_pattern` from the Scikit-Image library[1]. This function, besides the image to be processed, has the parameter such as the number of pixels in the neighborhood and the 'radius' for such search to be considered. The best parameters change according to the problem to be solved. You can try several sets of parameters to find the best one. The code below shows how to apply the LBP to the first image of Subject #1:

```
# Read one image of subject 1 from dataset
img = imread("ATT dataset/s1/1.pgm", as_gray=True)

# Extract LBP feature from the image
# P: Number of circularly symmetric neighbor set points = 12
# Q: Radius of circle = 3
lbp = local_binary_pattern(img, 12, 3)
```

The output of `local_binary_pattern` is an image of the same size as `img`, which represents the texture pattern found by the LBP. Fig. 1 shows the input image and the result of applying the LBP.

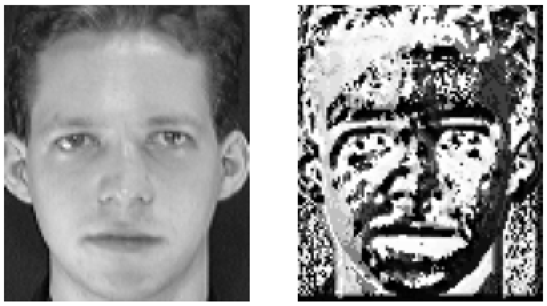


Figure 1: The result of the LBP feature extraction.

To this the LBP image into a feature vector you need to create the histogram of it using a reasonable number of bins. Experiments should be done to see if applying LBP is beneficial to the face recognition task, as well as how many bin provides the optimal performance.

4.1 Face classification using SVM and MLP

Classification is understood here as a process of comparing the feature vectors to find similarities. The most basic classifier is K-nearest-neighbor based on Euclidean distance. The current state-of-the-art technique is the Support Vector Machine. In this project, you will evaluate two of the most used classifiers: the Support Vector Machines (SVM) and; the Multi-layer Perceptron (MLP) neural network.

[1] https://scikit-image.org/docs/stable/api/skimage.feature.html?#skimage.feature.local_binary_pattern

4.1.1 Support Vector Machine (SVM) classifier

SVM calculates a boundary for every trained feature, in order to optimally divide the data into categories. The Scikit-Learn library has the SVM already implemented[2]. At the bottom of the documentation, you will see several tutorials on how to use this classifier.

The SVM has mainly three hyper-parameters that must be specified before the training processing: the 'kernel' function, the regularization parameter 'C' and the kernel coefficient called 'gamma'. You need to experiment with different combinations to find the best set for your application.

```
from sklearn.svm import SVC
import numpy as np

train_path = r'ORL\\Train_Data\\*\\*'
test_path = r'ORL\\Test_Data\\*\\*'

train_data, train_label = get_data(train_path) # use the previous custom get_data function
test_data, test_label = get_data(test_path)      # use the previous custom get_data function

svm = SVC(C=5.0, gamma=0.001, probability=True) #experiment with different C and gamma
svm.fit(train_data, train_label)

# probability matrix NxM where N is number of samples and M is the number of classes
probability_matrix = svm.predict_proba(test_data)

# calculate accuracy
prediction = np.argmax(probability_matrix,1)
result = prediction == test_label
accuracy = np.sum(result)/len(result)
```

4.1.2 Multi-layer Perceptron (MLP) classifier

The MLP is the classical artificial neural network, also known as fully connected network. It is composed of at least three layers of neurons, or perceptrons, called: input layer, hidden layer and output layer. The learning process occurs in the hidden and output layers. The Scikit-Learn library has implemented the MLP[3].

As well as the SVM, the MLP has several hyper-parameters. The main ones are: the size of the hidden layer, the activation function and the learning rate. You can search on the internet about the meaning of each one and how to wisely tune them.

```
from sklearn.neural_network import MLPClassifier
import numpy as np

train_path = r'ORL\\Train_Data\\*\\*'
test_path = r'ORL\\Test_Data\\*\\*'

# customize get_data function to include pre—processing methods (adding PCA or LBP)
```

[2]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
[3]https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

```
train_data, train_label = get_data(train_path) # use the previous custom get_data function
test_data, test_label = get_data(test_path)      # use the previous custom get_data function

# create MLP with 3 layers of perceptrons
# first layers has 128 neurons then 64 then another 128
# experiment with different layers/neurons
# experiment with different learning rate
mlp = MLPClassifier(hidden_layer_sizes=(128,64,128),
                                       learning_rate_init = 0.001,
                                       random_state=1)
mlp.fit(train_data, train_label)

# probability matrix NxM where N is number of samples and M is the number of classes
probability_matrix = mlp.predict_proba(test_data)

# calculate accuracy
prediction = np.argmax(probability_matrix,1)
result = prediction == test_label
accuracy = np.sum(result)/len(result)
```

4.1.3 Testing the classifier

Both classifiers, SVM and MLP, implementation in the Scikit-Learn has similar interface, where we call `.fit(...)` for training (using the training set) and `.predict(...)` to test (using the test set only). In the example, we used `.predict_proba(...)` to get the probability matrix. The matrix can be used to calculate additional metrics and graphs such as the ROC (FPR vs. TPR) or DET (FPR vs. FNR) curves.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, det_curve

# false positive rate (FPR)
# true positive rate (TPR)
# false negative rate (FNR)
# 40 subjects by 100 values
roc_fpr = np.zeros((40,100))
roc_tpr = np.zeros((40,100))

det_fpr = np.zeros((40,100))
det_fnr = np.zeros((40,100))

fig, [ax_roc, ax_det] = plt.subplots(1, 2, figsize=(11, 5))

# iterate through number of subjects (40)
# calculating the ROC and DET curve for each subject
for i in range(40):
        # find the fpr by tpr values for ROC
        fpr, tpr, _ = roc_curve(test_label[:]==i, probability_matrix[:,i])
        # interpolate rates so each curve contains 100 values
        roc_fpr[i] = np.linspace(min(fpr),max(fpr),100)
        roc_tpr[i] = np.interp(roc_fpr[i],fpr,tpr)

        # find the fpr by fnr values for DET
        fpr, fnr, _ = det_curve(test_label[:]==i, probability_matrix[:,i])
        # interpolate rates so each curve contains 100 values
        det_fpr[i] = np.linspace(min(fpr),max(fpr),100)
        det_fnr[i] = np.interp(det_fpr[i],fpr,fnr)

# average each subject's ROC curve to get the average ROC curve of system
roc_mid_fpr = np.mean(roc_fpr,0)
roc_mid_tpr = np.mean(roc_tpr,0)
ax_roc.plot(roc_mid_fpr, roc_mid_tpr)

# average each subject's DET curve to get the average DET curve of system
det_mid_fpr = np.mean(det_fpr,0)
det_mid_fnr = np.mean(det_fnr,0)
ax_det.plot(roc_mid_fpr, roc_mid_tpr)

ax_roc.set_xlabel('FPR')
ax_roc.set_ylabel('TPR')
ax_roc.set_title("Receiver Operating Characteristic (ROC) curves")

ax_det.set_xlabel('FPR')
ax_det.set_ylabel('FNR')
ax_det.set_title("Detection Error Tradeoff (DET) curves")

plt.show()
```

5 Subproject 2 - Face recognition in Python using BoB biometrics package (Linux or Mac platform)

This project goals are as follows:

- Investigate Gaussian Mixture Model (GMM) for face recognition. The GMM implemented in Bob's tool are Inter-Session Variability model (ISV) or Total Variability model (TV, also known as I-Vector).
- Investigate how to use bob.bio.gmm package

Bob is a free signal-processing and machine learning toolbox originally developed by the Biometrics group at IDIAP Research.

- The toolbox is written in Python and C++.
- The Biometric Framework is written entirely in Python, but only little Python programming experience is required.
- Currently, only Linux-based and Mac-OS-based systems are supported. For all other systems including Windows, a Virtual Machine with all the required software and data pre-installed is provided.

5.1 List of Bob packages

Bob's tool is organized in several independent Python packages, including

- bob.bio.face Tools to run face recognition experiments, such as face detection, facial feature extraction and comparison, and face image databases[4].
- bob.bio.gmm Algorithms based on Gaussian Mixture Modeling (GMM) such as Inter-Session Variability modeling (ISV) or Total Variability modeling (TV, aka. I-Vector).

Other packages are available from IDIAP[5].

5.2 Bob Installation Instructions

The first step is to install Conda. Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux.

1. Download Anaconda installer for Linux.
2. In your terminal window, run the following command.
```
$ bash Anaconda-latest-Linux-x 86\_64.sh
```
3. Follow the prompts on the installer screen.
4. To make sure the changes take effect, close and then re-open your Terminal window.

[4]https://www.idiap.ch/software/bob/docs/bob/bob.bio.face/master/
[5]https://www.idiap.ch/software/bob/docs/bob/bob/stable/list.html

5. To test your installation, in your Terminal window or Anaconda Prompt, run the command below, and for a successful installation, a list of installed packages appears.

```
$ conda list
```

Bob provides a Conda environment file, which creates a new Conda environment called Bob as follows:

```
$ conda env create -f bob.yml
$ source activate bob                     # activate the environment
$ python -c "import caffe; import bob.bio.base" # test the installation
```

Make sure that you have the latest version of conda installed by running the commands below:

```
$ conda update -n root conda
$ conda config --set show_channel_urls True
$ conda config --add channels defaults
$ conda config --add channels https://www.idiap.ch/software/bob/conda
```

Now you can create an environment and install Bob (`bob_py3`) in that environment:

```
$ conda create -n bob_py3 --override-channels \
  -c https://www.idiap.ch/software/bob/conda \
  -c defaults \
  python=3 bob
$ source activate bob_py3
$ python -c 'import bob.io.base'
```

You can install other Bob packages or database packages as follows:

```
$ conda install <package-name>
```

If there are no conda packages available for that package, use:

```
$ pip install <package-name>
```

Example:

```
$ source activate <bob_conda_environment>
$ conda install bob.bio.base \
                bob.bio.face \
                bob.bio.gmm \
                bob.bio.video \
                bob.db.youtube \
                gridtk
```

5.3 Testing your Installation

You can install a `nose` package to verify your installation:

```
$ conda install nose  # install nose
$ nosetests -vs bob.bio.base
$ nosetests -vs bob.bio.gmm
$...
```

Run the `nosetests` for each of the `bob.bio` packages separately.

5.4 Database management

Bob.bio allows to perform the biometric recognition experiments using the collected databases[6]. After downloading the raw data for the databases, you will need to tell bob.bio where these databases can be found. A special file, where you can set your directories, is located in

```
~/.bob_bio_databases.txt
```

and it contains several lines like this:

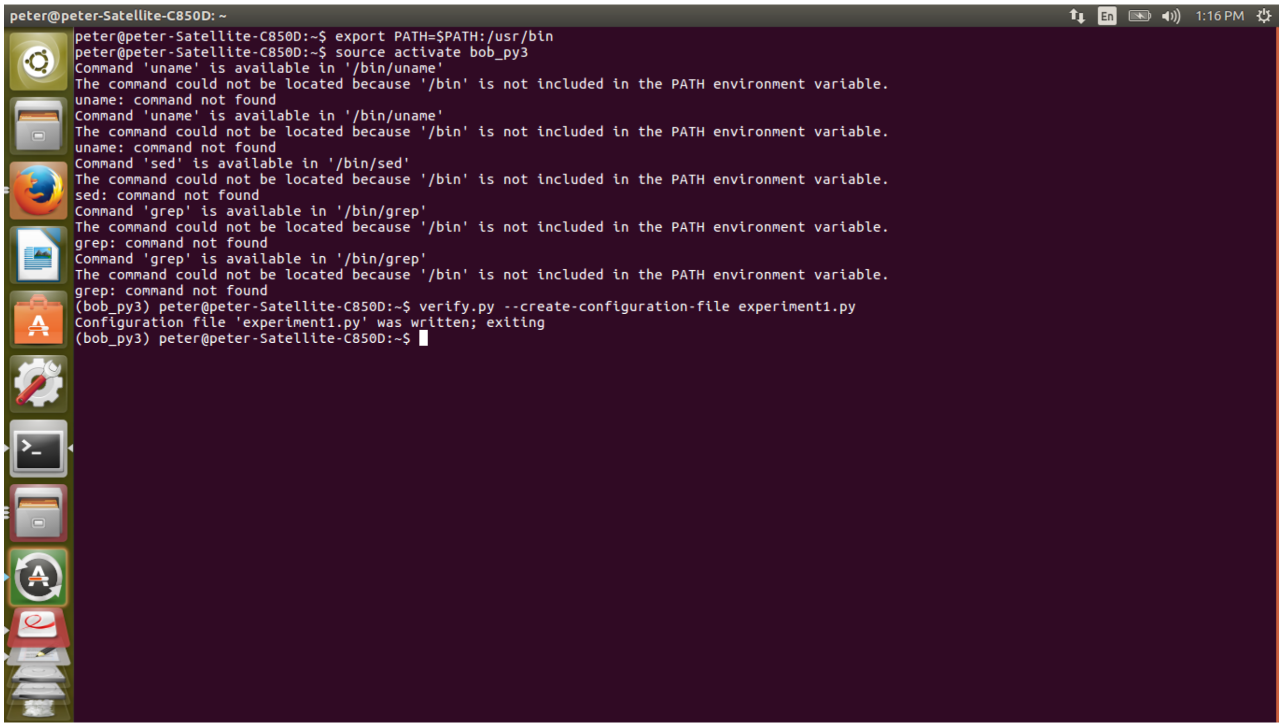
```
[YOUR_ATNT_DIRECTORY] = /path/to/your/directory
```

If this file does not exist, create and populate it yourself. In this project, you will use a small face recognition dataset, the AT&T Database of Faces, formerly known as the ORL database. Use the folowng commands:

```
$ wget http://www.cl.cam.ac.uk/Research/DTG/attarchive/pub/data/att_faces.zip
$ unzip att_faces.zip
```

5.5 Example of the experiment

5.5.1 Running Experiments (part I) –bob.bio.base



```
peter@peter-Satellite-C850D: ~
peter@peter-Satellite-C850D:~$ export PATH=$PATH:/usr/bin
peter@peter-Satellite-C850D:~$ source activate bob_py3
Command 'uname' is available in '/bin/uname'
The command could not be located because '/bin' is not included in the PATH environment variable.
uname: command not found
Command 'uname' is available in '/bin/uname'
The command could not be located because '/bin' is not included in the PATH environment variable.
uname: command not found
Command 'sed' is available in '/bin/sed'
The command could not be located because '/bin' is not included in the PATH environment variable.
sed: command not found
Command 'grep' is available in '/bin/grep'
The command could not be located because '/bin' is not included in the PATH environment variable.
grep: command not found
Command 'grep' is available in '/bin/grep'
The command could not be located because '/bin' is not included in the PATH environment variable.
grep: command not found
(bob_py3) peter@peter-Satellite-C850D:~$ verify.py --create-configuration-file experiment1.py
Configuration file 'experiment1.py' was written; exiting
(bob_py3) peter@peter-Satellite-C850D:~$
```

Figure 2: Screenshot of using the verify command

Bob's tool provides a generic script `verify.py`. As a default, `verify.py` accepts one or more configuration files that include the parametrization of the experiment to run. A configuration file contains one or more variables that define parts of the experiment. There is a shortcut to generate an empty configuration file that contains all possible variables as shown in Figure 3.

[6] https://gitlab.idiap.ch/bob/bob/wikis/Packages/#biometric-recognition-database-frontends

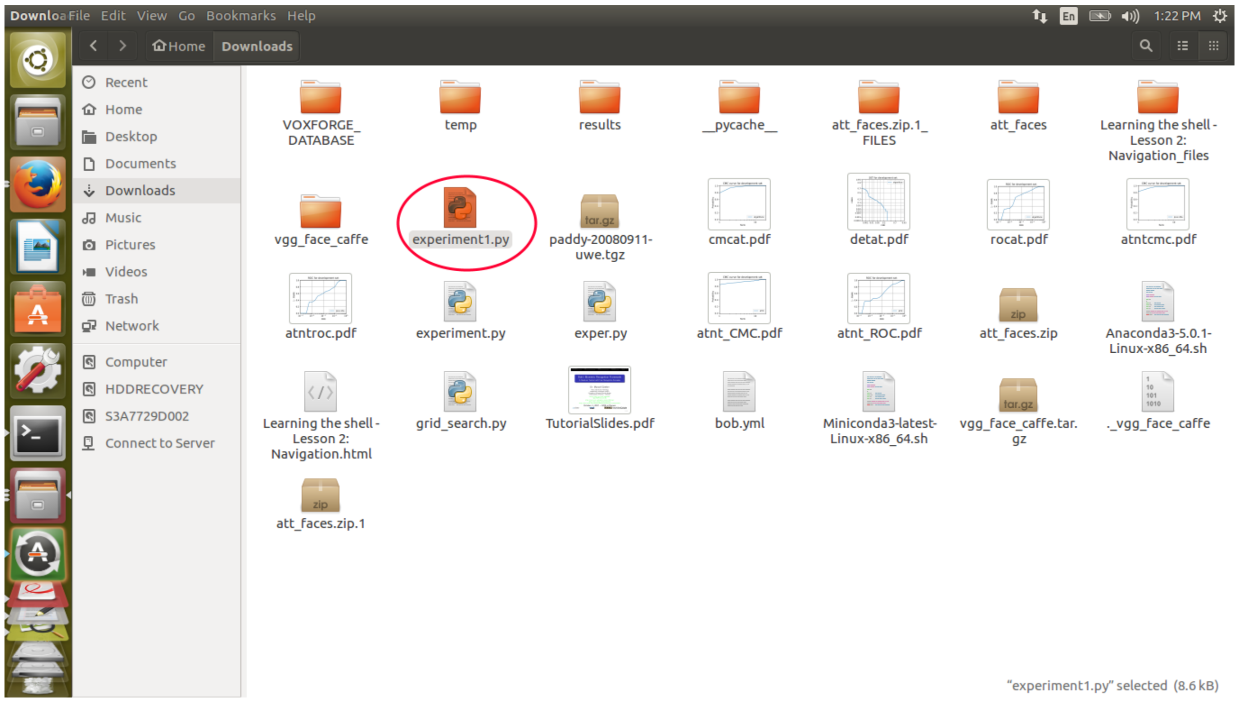
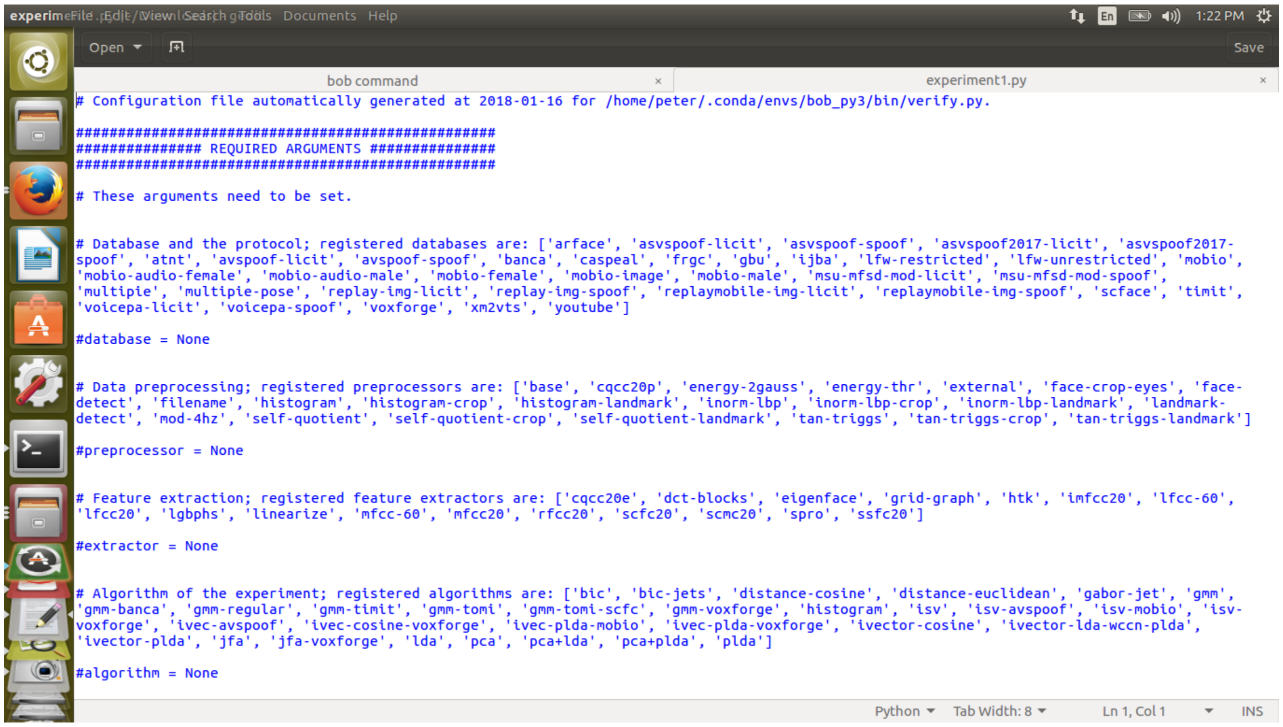


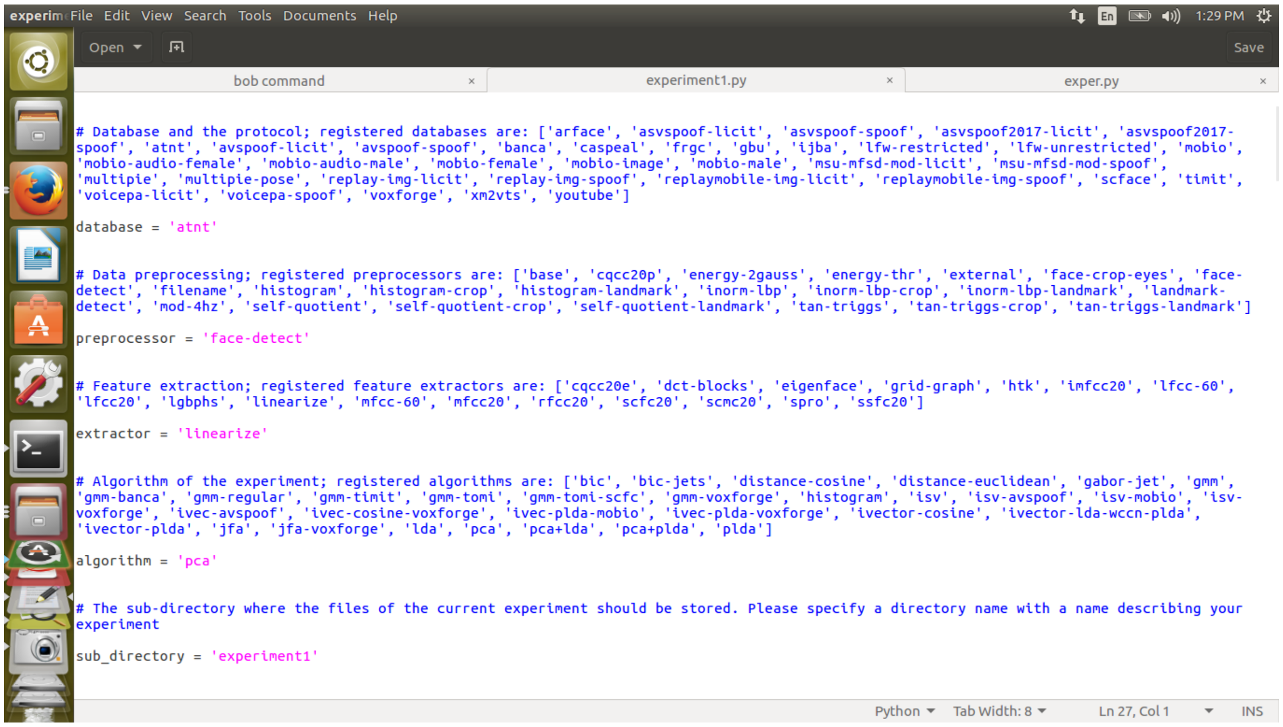
Figure 3: Screenshot of the python file



```
# Configuration file automatically generated at 2018-01-16 for /home/peter/.conda/envs/bob_py3/bin/verify.py.

##################################################
################ REQUIRED ARGUMENTS ###############
##################################################

# These arguments need to be set.


# Database and the protocol; registered databases are: ['arface', 'asvspoof-licit', 'asvspoof-spoof', 'asvspoof2017-licit', 'asvspoof2017-
spoof', 'atnt', 'avspoof-licit', 'avspoof-spoof', 'banca', 'caspeal', 'frgc', 'gbu', 'ijba', 'lfw-restricted', 'lfw-unrestricted', 'mobio',
'mobio-audio-female', 'mobio-audio-male', 'mobio-female', 'mobio-image', 'mobio-male', 'msu-mfsd-mod-licit', 'msu-mfsd-mod-spoof',
'multipie', 'multipie-pose', 'replay-img-licit', 'replay-img-spoof', 'replaymobile-img-licit', 'replaymobile-img-spoof', 'scface', 'timit',
'voicepa-licit', 'voicepa-spoof', 'voxforge', 'xm2vts', 'youtube']

#database = None


# Data preprocessing; registered preprocessors are: ['base', 'cqcc20p', 'energy-2gauss', 'energy-thr', 'external', 'face-crop-eyes', 'face-
detect', 'filename', 'histogram', 'histogram-crop', 'histogram-landmark', 'inorm-lbp', 'inorm-lbp-crop', 'inorm-lbp-landmark', 'landmark-
detect', 'mod-4hz', 'self-quotient', 'self-quotient-crop', 'self-quotient-landmark', 'tan-triggs', 'tan-triggs-crop', 'tan-triggs-landmark']

#preprocessor = None


# Feature extraction; registered feature extractors are: ['cqcc20e', 'dct-blocks', 'eigenface', 'grid-graph', 'htk', 'imfcc20', 'lfcc-60',
'lfcc20', 'lgbphs', 'linearize', 'mfcc-60', 'mfcc20', 'rfcc20', 'scfc20', 'scmc20', 'spro', 'ssfc20']

#extractor = None


# Algorithm of the experiment; registered algorithms are: ['bic', 'bic-jets', 'distance-cosine', 'distance-euclidean', 'gabor-jet', 'gmm',
'gmm-banca', 'gmm-regular', 'gmm-timit', 'gmm-tomi', 'gmm-tomi-scfc', 'gmm-voxforge', 'histogram', 'isv', 'isv-avspoof', 'isv-mobio', 'isv-
voxforge', 'ivec-avspoof', 'ivec-cosine-voxforge', 'ivec-plda-mobio', 'ivec-plda-voxforge', 'ivector-cosine', 'ivector-lda-wccn-plda',
'ivector-plda', 'jfa', 'jfa-voxforge', 'lda', 'pca', 'pca+lda', 'pca+plda', 'plda']

#algorithm = None
```

Figure 4: Screenshot of the python file content

```
$ verify.py --create-configuration-file experiment1.py
```



```
# Database and the protocol; registered databases are: ['arface', 'asvspoof-licit', 'asvspoof-spoof', 'asvspoof2017-licit', 'asvspoof2017-
spoof', 'atnt', 'avspoof-licit', 'avspoof-spoof', 'banca', 'caspeal', 'frgc', 'gbu', 'ijba', 'lfw-restricted', 'lfw-unrestricted', 'mobio',
'mobio-audio-female', 'mobio-audio-male', 'mobio-female', 'mobio-image', 'mobio-male', 'msu-mfsd-mod-licit', 'msu-mfsd-mod-spoof',
'multipie', 'multipie-pose', 'replay-img-licit', 'replay-img-spoof', 'replaymobile-img-licit', 'replaymobile-img-spoof', 'scface', 'timit',
'voicepa-licit', 'voicepa-spoof', 'voxforge', 'xm2vts', 'youtube']

database = 'atnt'


# Data preprocessing; registered preprocessors are: ['base', 'cqcc20p', 'energy-2gauss', 'energy-thr', 'external', 'face-crop-eyes', 'face-
detect', 'filename', 'histogram', 'histogram-crop', 'histogram-landmark', 'inorm-lbp', 'inorm-lbp-crop', 'inorm-lbp-landmark', 'landmark-
detect', 'mod-4hz', 'self-quotient', 'self-quotient-crop', 'self-quotient-landmark', 'tan-triggs', 'tan-triggs-crop', 'tan-triggs-landmark']

preprocessor = 'face-detect'


# Feature extraction; registered feature extractors are: ['cqcc20e', 'dct-blocks', 'eigenface', 'grid-graph', 'htk', 'imfcc20', 'lfcc-60',
'lfcc20', 'lgbphs', 'linearize', 'mfcc-60', 'mfcc20', 'rfcc20', 'scfc20', 'scmc20', 'spro', 'ssfc20']

extractor = 'linearize'


# Algorithm of the experiment; registered algorithms are: ['bic', 'bic-jets', 'distance-cosine', 'distance-euclidean', 'gabor-jet', 'gmm',
'gmm-banca', 'gmm-regular', 'gmm-timit', 'gmm-tomi', 'gmm-tomi-scfc', 'gmm-voxforge', 'histogram', 'isv', 'isv-avspoof', 'isv-mobio', 'isv-
voxforge', 'ivec-avspoof', 'ivec-cosine-voxforge', 'ivec-plda-mobio', 'ivec-plda-voxforge', 'ivector-cosine', 'ivector-lda-wccn-plda',
'ivector-plda', 'jfa', 'jfa-voxforge', 'lda', 'pca', 'pca+lda', 'pca+plda', 'plda']

algorithm = 'pca'


# The sub-directory where the files of the current experiment should be stored. Please specify a directory name with a name describing your
experiment

sub_directory = 'experiment1'
```

Figure 5: Screenshot of the python file content after changing variables

The generated experiment.py is a regular python file as shown in Figure 4. There are variables with default values as shown in Figure 5. There are five variables required to define the complete biometric recognition experiment:

- database: The database to run the experiments on.
- preprocessor: The data preprocessor.
- extractor: The feature extractor.
- algorithm: The recognition algorithm.
- subdirectory: A descriptive name for your experiment, which will serve as a sub-directory.

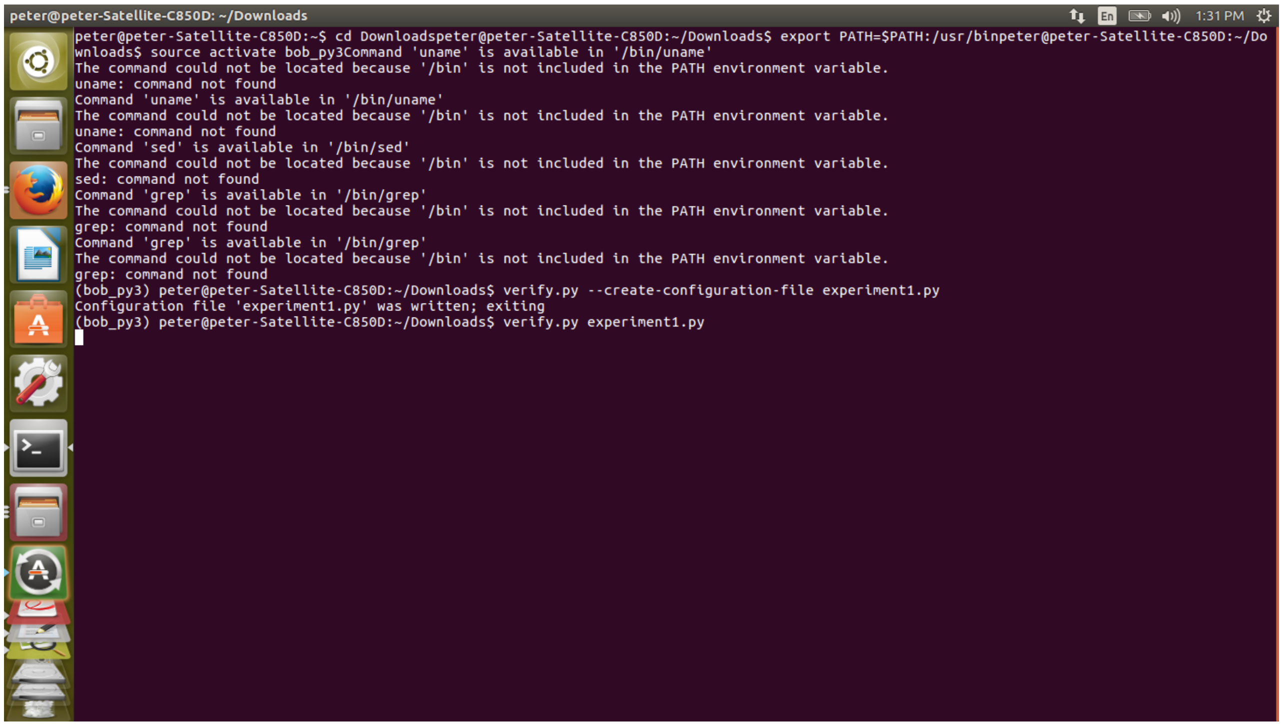
The first four variables can be specified in several different ways. We will use only the registered resources as shown in Figure 6. To get a list of registered resources, call:

```
$ resources.py
```

One variable, which is not required, but recommended, is "verbose". By default, the algorithms are set up to execute, and only errors are reported (logging.ERROR). To change this, you can set the verbose variable to show verbosity level 2. An example of minimal configuration file (experiment1.py) shall look as follows:

```
database = 'atnt'
preprocessor = 'face-detect'
extractor = 'linearize'
```

```
    algorithm = 'pca'
    sub_directory = 'experiment1'
    verbose = 2
```



```
peter@peter-Satellite-C850D: ~/Downloads
peter@peter-Satellite-C850D:~$ cd Downloadspeter@peter-Satellite-C850D:~/Downloads$ export PATH=$PATH:/usr/binpeter@peter-Satellite-C850D:~/Do
wnloads$ source activate bob_py3Command 'uname' is available in '/bin/uname'
The command could not be located because '/bin' is not included in the PATH environment variable.
uname: command not found
Command 'uname' is available in '/bin/uname'
The command could not be located because '/bin' is not included in the PATH environment variable.
uname: command not found
Command 'sed' is available in '/bin/sed'
The command could not be located because '/bin' is not included in the PATH environment variable.
sed: command not found
Command 'grep' is available in '/bin/grep'
The command could not be located because '/bin' is not included in the PATH environment variable.
grep: command not found
Command 'grep' is available in '/bin/grep'
The command could not be located because '/bin' is not included in the PATH environment variable.
grep: command not found
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$ verify.py --create-configuration-file experiment1.py
Configuration file 'experiment1.py' was written; exiting
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$ verify.py experiment1.py
```

Figure 6: Running the experiment

To run the experiment, use this command:

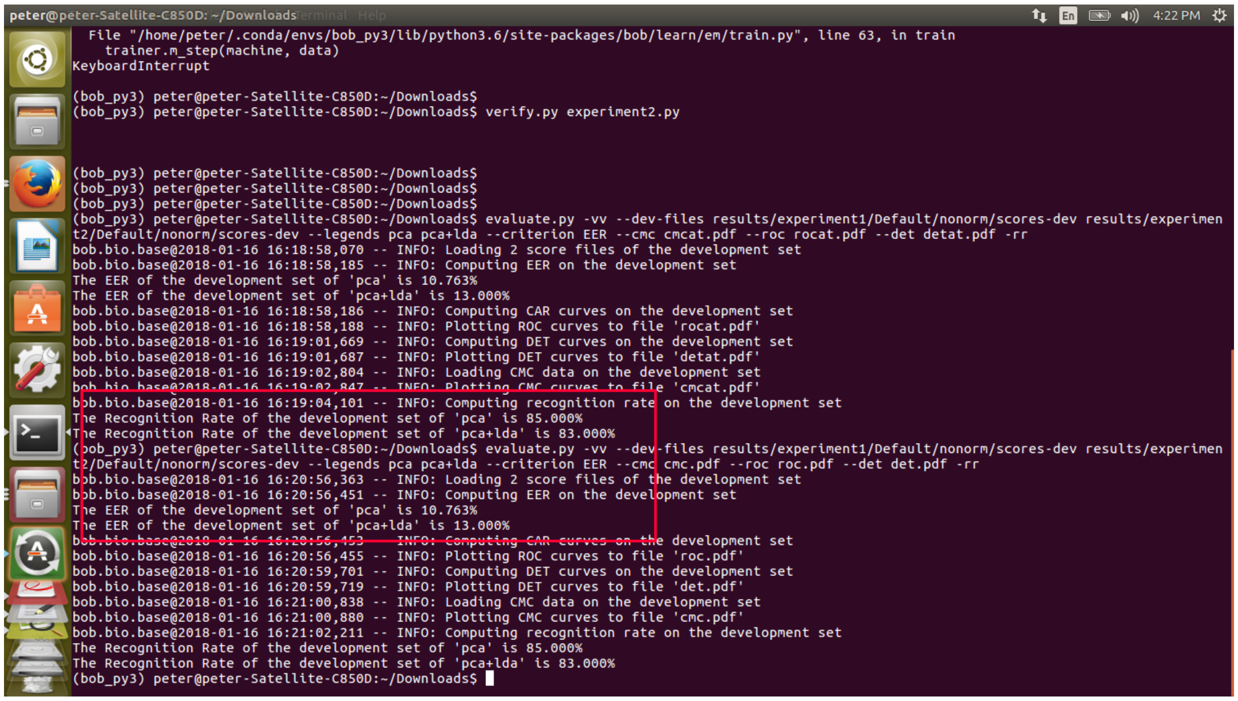
```
$ verify.py experiment1.py
```

To be able to run exactly the command line from above, it requires to have bob.bio.face installed. The final result of the experiment will be one (or more) score file(s). Usually, they will be called something like scores-dev. By default, you can find them in a sub-directory 'result_directory', but you can change this option using the result_directory variable.

5.6 Command Line Options

Each configuration can be specified as command line option of 'verify.py'. The same experiment as above can be executed using the commands below:

```
$ verify.py
    --database atnt
    --preprocessor face-detect
    --extractor linearize
    --algorithm pca
    --sub-directory PCA_ATNT {vv
```



```
peter@peter-Satellite-C850D: ~/Downloads
  File "/home/peter/.conda/envs/bob_py3/lib/python3.6/site-packages/bob/learn/em/train.py", line 63, in train
    trainer.m_step(machine, data)
KeyboardInterrupt

(bob_py3) peter@peter-Satellite-C850D:~/Downloads$
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$ verify.py experiment2.py



(bob_py3) peter@peter-Satellite-C850D:~/Downloads$
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$ evaluate.py -vv --dev-files results/experiment1/Default/nonorm/scores-dev results/experimen
t2/Default/nonorm/scores-dev --legends pca pca+lda --criterion EER --cmc cmcat.pdf --roc rocat.pdf --det detat.pdf -rr
bob.bio.base@2018-01-16 16:18:58,070 -- INFO: Loading 2 score files of the development set
bob.bio.base@2018-01-16 16:18:58,185 -- INFO: Computing EER on the development set
The EER of the development set of 'pca' is 10.763%
The EER of the development set of 'pca+lda' is 13.000%
bob.bio.base@2018-01-16 16:18:58,186 -- INFO: Computing CAR curves on the development set
bob.bio.base@2018-01-16 16:18:58,188 -- INFO: Plotting ROC curves to file 'rocat.pdf'
bob.bio.base@2018-01-16 16:19:01,669 -- INFO: Computing DET curves on the development set
bob.bio.base@2018-01-16 16:19:01,687 -- INFO: Plotting DET curves to file 'detat.pdf'
bob.bio.base@2018-01-16 16:19:02,804 -- INFO: Loading CMC data on the development set
bob.bio.base@2018-01-16 16:19:02,847 -- INFO: Plotting CMC curves to file 'cmcat.pdf'
bob.bio.base@2018-01-16 16:19:04,101 -- INFO: Computing recognition rate on the development set
The Recognition Rate of the development set of 'pca' is 85.000%
The Recognition Rate of the development set of 'pca+lda' is 83.000%
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$ evaluate.py -vv --dev-files results/experiment1/Default/nonorm/scores-dev results/experimen
t2/Default/nonorm/scores-dev --legends pca pca+lda --criterion EER --cmc cmc.pdf --roc roc.pdf --det det.pdf -rr
bob.bio.base@2018-01-16 16:20:56,363 -- INFO: Loading 2 score files of the development set
bob.bio.base@2018-01-16 16:20:56,451 -- INFO: Computing EER on the development set
The EER of the development set of 'pca' is 10.763%
The EER of the development set of 'pca+lda' is 13.000%
bob.bio.base@2018-01-16 16:20:56,453 -- INFO: Computing CAR curves on the development set
bob.bio.base@2018-01-16 16:20:56,455 -- INFO: Plotting ROC curves to file 'roc.pdf'
bob.bio.base@2018-01-16 16:20:59,701 -- INFO: Computing DET curves on the development set
bob.bio.base@2018-01-16 16:20:59,719 -- INFO: Plotting DET curves to file 'det.pdf'
bob.bio.base@2018-01-16 16:21:00,838 -- INFO: Loading CMC data on the development set
bob.bio.base@2018-01-16 16:21:00,880 -- INFO: Plotting CMC curves to file 'cmc.pdf'
bob.bio.base@2018-01-16 16:21:02,211 -- INFO: Computing recognition rate on the development set
The Recognition Rate of the development set of 'pca' is 85.000%
The Recognition Rate of the development set of 'pca+lda' is 83.000%
(bob_py3) peter@peter-Satellite-C850D:~/Downloads$
```

Figure 7: Evaluating the experiment

5.7 Analysis of the Experimental Results

After the experiment has finished successfully, one or more text file containing all the scores are stored. To analyze the experiment, you can use the generic evaluate.py script, which has properties for all performance evaluation types, such as Cumulative matching Score (CMC), DIR, Receiver Operating Curve (ROC) and DET plots, as well as computing recognition rates, Equal Error Rate (EER), HTER, Cllr and minDCF. Additionally, a combination of different algorithms can be plotted into the same files. Just specify all the score files that you want to evaluate using the –dev-files option, and possible legends for the plots (in the same order) using the –legends option, and the according plots will be generated. For example, to create a ROC curve for the experiment above, use:

```
$ evaluate.py --dev-files results/pca-experiment/male/nonorm/scores-dev
              --legend MOBIO
              --roc MOBIO_MALE_ROC.pdf -vv
```

There is another file called Experiment.info inside the 'result' directory. This file is a text file that contains a complete configuration of the experiment. It makes it possible to inspect all default parameters of the algorithms, and even to re-run the exact same experiment.

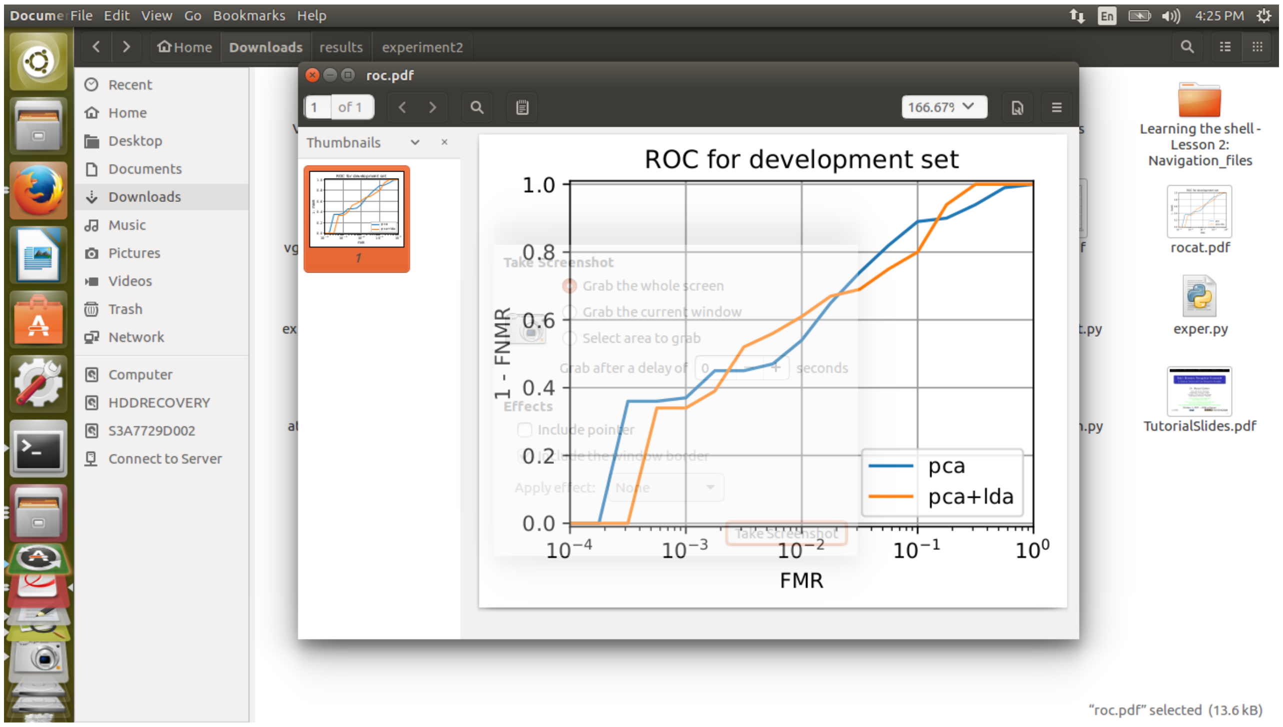


Figure 8: ROC curve

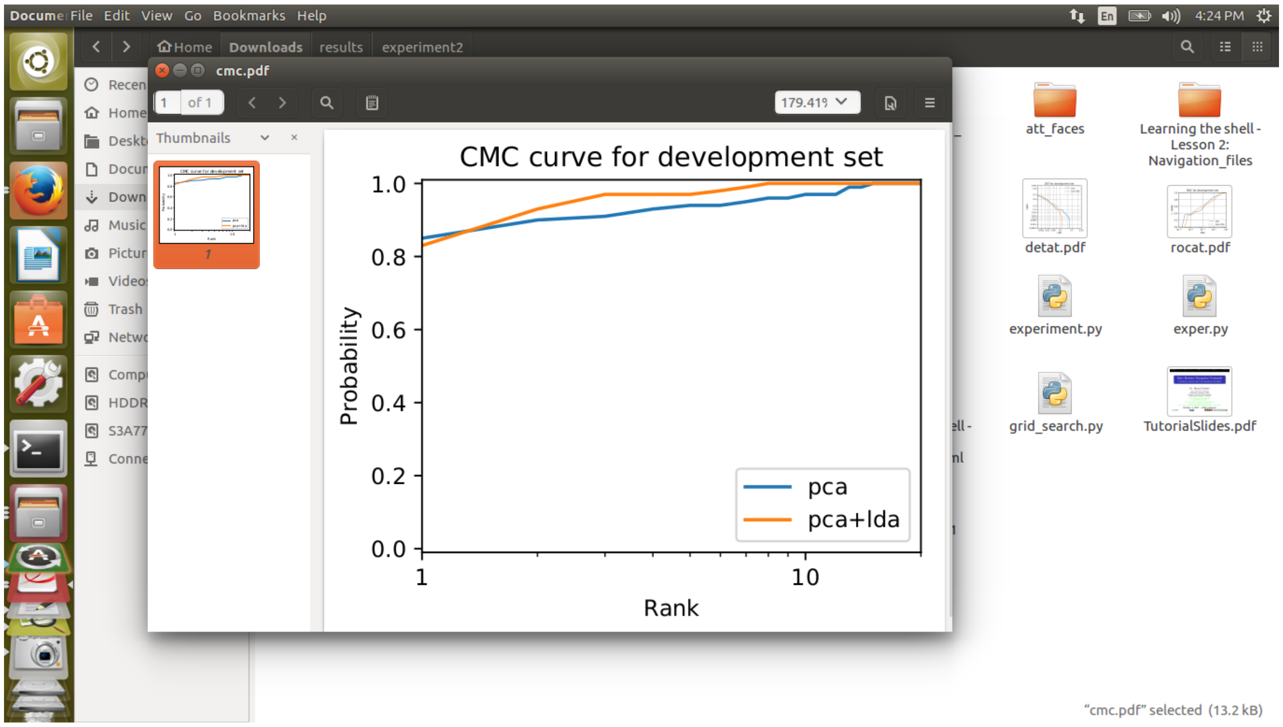


Figure 9: CMC curve

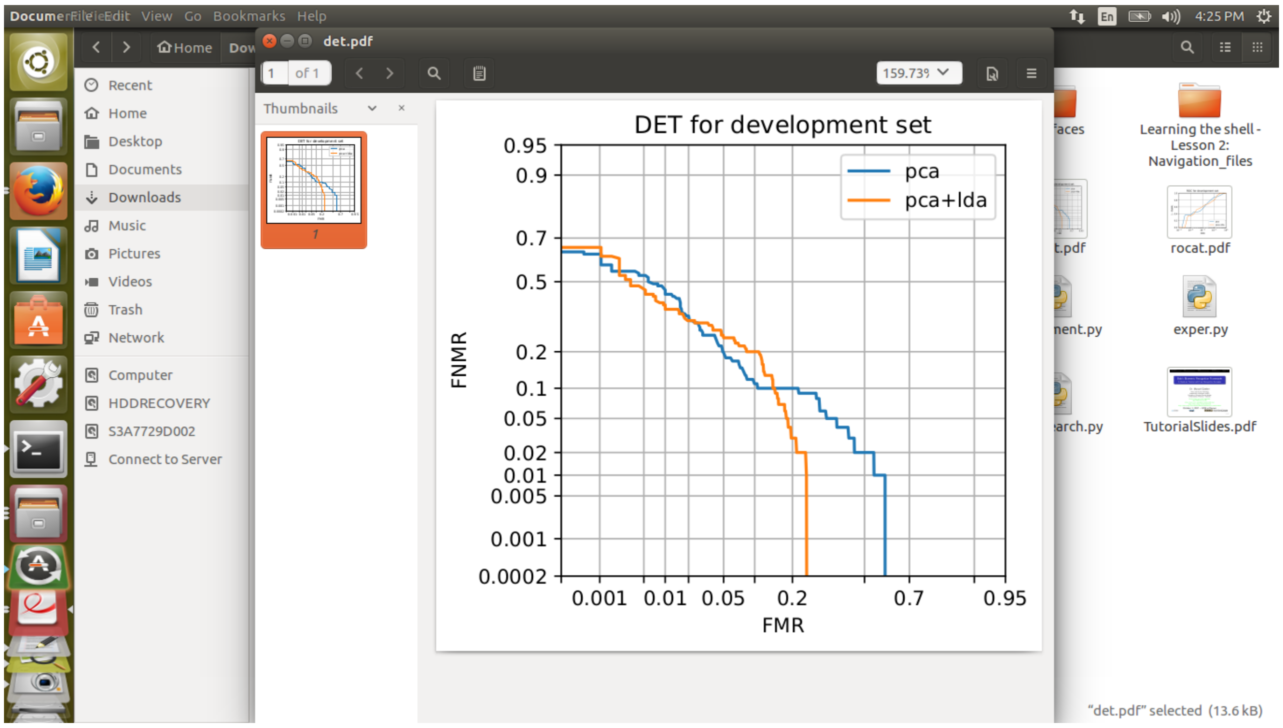


Figure 10: DET curve

5.8 Running Baseline Algorithms-bob.bio.face

This option executes one of the baseline face recognition algorithms that are implemented in bob.bio. To run the baseline experiments, you can use the baselines.py script:

```
$ baselines.py
```

The configuration file is similar to the one we used in `verify.py`.

5.9 Baseline Results

To evaluate the results, a wrapper call to evaluate.py is produced by the baselines.py –evaluate command. Several types of evaluation can be achieved: ROC curves, DET plots, CMC curves and the computation of EER/HTER. The complete set of results of the baseline experiments are generated using:

```
$ baselines.py --all -vv --evaluate ROC DET CMC HTER
```

6 Reference links

- http://vast.uccs.edu/public-data/IJCB.html
- https://www.idiap.ch/software/bob/docs/bob/bob.bio.base/master/

7 Acknowledgments

Projects 1 were co-developed by Dr. K. Lai and Dr. S. Yanushkevich, in the Biometric Technologies Laboratory. Project 2 is based on the open BoB package developed at IDIAP, Switzerland.

February 15, 2023