Laboratory # 3

Biometric-based verification: Signature

1 Introduction

The purpose of this laboratory exercise is to learn handling and processing the acquired biometric data such as authentic (genuine) and forged (impostor) signatures of one individual, and preparing this data for usage in 1:1 matching. In this exercise, we will also look how to handle multiple-file input data in Python, and how to plot and analyze the normal distribution of scores.

The tablet data acquired in Lab 2 are represented by the extracted features: coordinates, time and pen pressure information. The extracted features will be used for matching based on score calculation.

In our exercise, the features (`coord`, `time` and `prs`) are already extracted. Data used in classification come from a proper feature extraction, with no missing values.

A simple classifier separates the data into two classes. The classifier has two main components: a training function and a classification function. The training is based on the Expectation Maximization (EM) algorithm.

To calculate scores, we will use a statistical classification method called *Gaussian Mixture Model* (GMM). Since the signature data are represented in the time domain. In the frequency domain, it translates into harmonics related to the period of the signal. What we need is the envelope of these harmonics to characterize the signature. This can be achieved by taking the peaks of the harmonics in the frequency domain. Such an envelope is represented by a mixture of Gaussian components.

GMM is a weighted sum of Gaussian probability density functions which are referred to as Gaussian components of the mixture model describing a class.

A Gaussian mixture density function is a weighted sum of M constituent functions,

$$p(x|\lambda) = \sum_{i=1}^{M} a_i f_i(x)$$

where x is a D-dimensional random vector, $f_i(x)$, $i = 1, 2, ..., M$ are the constituent Gaussian density functions, and a_i are the mixture weights. Each constituent Gaussian density is a D-variate normal density function. Note that for probabilities $p(x|\lambda)$, the constraint $\sum_{i=1}^{M} a_i = 1$ must be satisfied. Each individual has its own density function, that is $p(x|\lambda_s)$ where λ_s represent the different classes.

In this lab, we will two: genuine and impostor data. A maximum likelihood classifier can be used for this model. Given several classes, it is required to find the class with maximum posterior probability for the feature vectors X.

The probability of being the class λ_s given the feature vectors of the submitted signature is given by the Bayes formula:

$$P(\lambda_s|X) = \frac{P(X|\lambda_s)P(\lambda)}{P(X)}$$

Assume equal prior probabilities $P(\lambda)$ for all writers. $P(X)$ can be ignored as being a constant for all writers. Finally the criterion for the selection of one class as the correct one is:

$$max_s P(X|\lambda)$$

and assuming independence among the feature vectors, we get:

$$max_s \sum_t logP(x_t|\lambda),$$

in which $P(x_t|\lambda)$ for a feature x_t is calculated as shown above for $P(x|\lambda)$.

This classifier is already implemented in Python and available in the Scikit-Learn library[1].

In this Lab, we will focus on the analysis of the results, that is, score distributions, and evaluate FRR and FAR for those distributions.

2 The laboratory procedure

2.1 Acquisition of the sample data

Collect and save:

- 15 – 30 of your own (genuine) signatures,
- 15 – 30 of your "forged" (impostor) signatures (signed by someone else who try to replicate their shape),
- 15 – 30 some other signatures (also impostor, but some completely different written word, could be by the sme person).

After you collect signature files, it shall be saved in the same folder as the Jupyter Notebook `Lab3-SigVerif GMM.ipynb`.

An individual's signature acquisition using SigGet software results in a single `.csv` file (for example, `1.csv`). As in Lab2, the `.csv` file can be loaded into Python with the Pandas library's `read_csv` function:

```
Sig1 = pd.read_csv('1.csv')
```

the variable `Sig1` is a $P \times 4$ DataFrame[2]. Here, P is the quantity of points in that signature and the 4 columns are the features for each point: X, Y, *Pressure* and *Time*.

We have to load several "authentic" (genuine) signatures as well as several "impostor" ones. For example, if you have 30 signatures, then loading the 30 files (`0.csv, 1.csv,..., 9.csv`) can be managed by embedding the `read_csv` command in a `for` loop:

```
# this is the place where the folders "genuine" and "impostor" are
# change it to point to your dataset.
base_dir = '../signature_samples/biometrics/'
# note that if your dataset folder is in the same folder as the notebook,
# you don't need the "../" only "./"
base_dir = './signature_samples/biometrics/'

# "genuine" and "impostor" sub-directories
dir_authentic = base_dir + 'genuine/'
dir_impostor = base_dir + 'impostor/'
```

[1] https://scikit-learn.org/stable/modules/mixture.html
[2] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

```
# amount of signatures to be loaded
total_signatures = 30   # change this number

# lists to store the loaded signatures
auth_lst = []
imp_lst = []

# loading your signatures from both "authentic" and "impostor"
for i in range(total_signatures):
    # columns: [X, Y, Pressure, Time]
    csv_auth = pd.read_csv(dir_authentic + '%d.csv' % (i+1))
    csv_imp = pd.read_csv(dir_impostor + '%d.csv' % (i+1))

    # using .values will return only the DataFrame's values without the header
    # after this point, a 'ndarray' is added to the list
    auth_lst.append(csv_auth.values)
    imp_lst.append(csv_imp.values)
```

Each of the lists (`auth_lst` and `imp_lst`) has 30 (or your number) elements of the type `ndarray` (N-dimensional array of NumPy). Each element of this list has the size $P \times 4$. Here, P is the amount of points in the signature and each column corresponds to the fields: X, Y, *Pressure* and *Time*.

2.2 Classification procedure

The classification of classes (of individuals) include training a classifier, or a model, which is a Gaussian Mixture Model in this Lab. Once rained, the model shall be able to identify (classify) the new data presented to it. For the training, we are going to use the "authentic" (genuine) set of written data. Based on the previous code, our authentic set is stored in the variable `auth_list`. In the notebook provided, two variables/parameters are used for the training process:

`NUMTRAININGPATTERNS`: this is the number of signatures used for the training. It is usually set to no more than 80% of the given set size.

`NUMGCOMPONENTS`: quantity of Gaussian components used by the classifier. Usually this value is between 1 and 80, for example, 20, 40 and 60. The choice depends on your data, and can produce various matching score results.

After defining the reasonable values for these variables, the next step is to build a matrix that will store all the `NUMTRAININGPATTERNS` signatures together. In the code below, both `NUMGCOMPONENTS` and `NUMTRAININGPATTERNS` were set to 20. Thus, the following for-loop will load the first 20 signatures from the authentic set, variable `auth_lst`, and put all of them in the variable `train_set`:

```
# number of signature samples to use for training the models
NUMTRAININGPATTERNS = 20

# number of Gaussian mixture components in the GMM
NUMGCOMPONENTS = 20

# usually we don't know the final size of train_set because
# each signature has different number of points.
# to later on use 'concatenate', is necessary to take the first sample outside the loop
train_set = np.reshape(auth_lst[0], (-1,4))

# this for-loop put all the first "NUMTRAININGPATTERNS" in a huge matrix
# this matrix, stored in the varaible "train_set" will be used for the training
for i in range(1,NUMTRAININGPATTERNS):
        # taking from authentic/genuine
        sample =  np.reshape(auth_lst[i], (-1,4))
        train_set = np.concatenate((train_set, sample), axis=0)
```

At this point, all the data (20 authentic signatures), are grouped in the variable `train_set`.

The next step is to create and train the GMM model. In Python, with the Scikit-Learn library, we can do this in one single line:

```
# Training the GaussianMixture model from Scikit-Learn library
# the .fit(...) will perform the training step using the ``train_set''
gmm = GaussianMixture(n_components=NUMGCOMPONENTS).fit(train_set)
```

Note that the variable `NUMGCOMPONENTS`, defined previously as 20, is used during the creation of the model. The part `.fit(...)` of the previous code executes the training process. This function takes the variable `train_set`, with the 20 signatures that we prepared before.

To use the trained model later, we need to store it in some other variable. Let us use the variable `gmm`. Since we used 20 authentic samples for the training, there are only 10 left in our set or authentic signatures. In the next code, we use these remaining samples to calculate the log-likelihood score based on the trained (`gmm`) model using the function `.score(...)`. All the scores are collected into the variable `aScores` for later use.

```
# to store all the scores calculated
aScores = []
# going through the remaining signatures after removing the first 20 for training
for i in range(NUMTRAININGPATTERNS, total_signatures):
        # taking from authentic/genuine
        sample =  np.reshape(auth_lst[i], (-1,4))
        # compute the per-sample average log-likelihood of the given data.
        score = gmm.score(sample)
        print('%d: %.4f' % (i, score))
        # add the score calculated to the list aScores
        aScores.append(score)
```

In order to check how the trained GMM model distinguishes the classes of genuine and forged (impostor) signatures, we have to calculate the log-likelihood of the forged set. The process is very similar to what was done to the remaining samples of the authentic data. However, now we are storing the scores into the variable `fScores`:

```
fScores = []
for i in range(numForgedSigs):
        # taking from impostor
        sample =  np.reshape(imp_lst[i], (-1,4))
        score = gmm.score(sample)
        print('%d: %.4f' % (i, score))
        fScores.append(score)
```

Now we have both the genuine and impostor scores calculated. To plot a Normal distribution, we need two parameters, the mean value and the standard deviation. Thus, our next step is to calculate both values for the vectors, `aScore` and `fScore`:

```
# authentic set
aMu = np.mean(aScores)
aStd = np.std(aScores)

# impostor set
fMu = np.mean(fScores)
fStd = np.std(fScores)
```

Now we can plot the Normal distributions of the scores. The plots generated in the notebook are the normalized Probability Density Functions (PDFs) of `aProb` (green line) and `fProb` (red line), within the range defined by `np.arange(-60, 0, 0.01)`, as well as `aScores` (green squares) and `fScores` (red triangles). The green elements correspond to the genuine (authentic) data, while the red ones indicate the impostor. Keep in mind that the defined range might be different for your dataset.

3 Report

Your report in the form of a Jupyter Notebook / Python (file extension `.ipynb`) shall include the following graded components (10 marks total):

- Introduction (a paragraph about the purpose of the lab).
- (10 marks) Description of the result on each exercise with illustrations/graphs and analysis of the results (marks are distributed as shown in the Exercise section).
- Conclusion (a paragraph on what is the main take-out of the lab).

Your Notebook is your report that shall be saved (use menu "Download As") as `.ipynb`, and submitted through D2L dropbox for Lab 3, by the deadline (the following Thursday). Note that 10% of the lab grade will be deducted for each late submission day.

4 Exercise in Jupyter Notebook with Python

A detailed description of each exercise to be included in your report (10 marks total) is given below:

- **Exercise 1** (4 marks): Create your own data (three sets of signatures recorded as `.csv` files), or the data provided on D2L.

  Use the sample Notebook file `Lab3-SigVerif GMM.ipynb` to classify set 1 (genuine) and set 2 (impostor). Follow the instructions described in Section 2.2 and repeat this process for 2 different set of parameters (the number of signatures used for training, and the number of Gaussian mixture components, for example 20, 30, 40, ...), total 4 different pairs. In your report, include the illustrations of the distributions for these 4 sets of parameters. Include the 4 sets of distribution plots (figures) for your report. Include a brief analysis of how varying the above parameters influence the classification results.

  Plot the matching scores' PDFs of the "authentic" (genuine) and "forged" (impostor) sets. Find the mean (μ) and standard deviation (σ) for the genuine and impostor scores (variables `aMu`, `fMu`, and `aStd`, `fStd`, respectively). Use two other different sets of parameters, see the difference of plotting the pair of curves (authentic and impostor) together and separately.

- **Exercise 2** (4 marks): Consider the genuine set and the second set of impostor signatures that are very different (in shape, length, i.e. a different writing or another person signature). Repeat the process as described in Exercise 1, for 2 different set of parameters (the number of signatures used for training, and the number of Gaussian mixture components, for example 20, 30, 40, ...), total 4 different pairs. In your report, include the illustrations of the distributions for these 4 sets of parameters. Plot the matching scores' PDFs of the "authentic" (genuine) and "impostor" sets. Find the mean (μ) and standard deviation (σ) for the genuine and impostor scores (variables `aMu`, `fMu`, and `aStd`, `fStd`, respectively). Compare these results against the results of Exercise 1.

- **Exercise 3** (2 marks): Consider your plots from Exercise 1 or 2. Formulate a hypothesis H_0 about the μ and σ of the entire population of the authentic signatures, and analytically test this hypothesis based on your sample (your 10 or more signatures) given the level of the test significance equal to 0.05 ($\alpha = 0.05$, level of confidence: $1 - \alpha = 95\%$). Now, change the critical values (choose reasonable values) and evaluate the FRR value. Evaluate your FRR and FAR. You can do the calculations manually, and include in your report using the Markdown or taking a photo and adding this photo to the notebook (see Lab 1 on Markdown).

Upload your Jupyter Notebook (`.ipynb`) file containing your code on D2L.

Acknowledgments

The template for this Lab was developed by Dr. H. C. R. Oliveira (postdoc in the Biometric Technologies Laboratory in 2020-2022). We also acknowledge this course TAs, I. Yankovyi and M. Zakir, for verifying this lab code.

Dr. S. Yanushkevich
Biometric Technologies Laboratory
January 24, 2023.