Lab 13 & 14 - Keystroke Recognition

Classical Keystroke Recognition

In the classical keystroke recognition we will base on the following paper:

> Hosseinzadeh, Danoush, and Sridhar Krishnan. "Gaussian mixture modeling of keystroke patterns for biometric applications." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.6 (2008): 816-826.

1. First of all, download the Keystroke Dynamics Dataset: https://www.cs.cmu.edu/~keystroke/
   Here we will focus on the .csv file, but you can also use other versions.
2. Let's take a look at the dataset structure. What data does it contain? From how many people was the data collected? How many sequences do we have for each person?
3. Load data - you can use the standard csv Python package.
4. We need three features: KD (key down time), DDKL (down to down key latency) and UUKL (up to up key latency). First two are provided in the dataset (described as H.key1 and DD.key1.key2 values). The last one we can obtain as the following sum: UD.key1.key2 + H.key2. Extract vectors of each feature for each person ("subject") and each sequence ("sessionIndex" and "rep") from the dataset. Since every sequence was acquired from the same word (given password), each vector of a specific feature will have the same size.
   Let's think for the moment, how to store them for future use. Soon we will use bunch of vectors of specific features from the one person to train one GMM classifier, so it would be nice to access them via pair of keys: feature ('KD', 'DDKL', 'UUKL') and person ('sNNN'). One possible solution (not necessarily the best one!) is to use nested dictionaries. The outer will have feature as keys (and of course inner dictionaries as values), while inner dictionaries will have person as keys and list of feature vectors as values (in fact, each vector can also be a list, so here we will have list of lists as a dictionary value). For example, consider the subject "s002". We have 400 rows with this subject (from different sessions and reps). Assume that we want a vector of KD features. As it is mentioned above, the input dataset has equivalent features "H.key". So, for each row with subject "s002" we construct a vector with values from the following columns: H.period, H.t, H.i, ..., H.return (of course order is important here). In this manner we get 400 vectors for subject "s002" (in fact for each person we will have the same number of rows).
5. Split feature vectors into train and test sets, e.g. 80% to train and 20% to test. Remember that we have three different features and 51 different people (400 sequences for each person). To split them you can use indices (range(400)) sampled from a random distribution (e.g. using Python's random package).
6. Now we can create GMM models and LOOM thresholds database. Remember that we need one model and threshold for each combination of (feature, person) keys. As a GMM model you can use GaussianMixture (n_components=3, try also different values) from scikit-learn package with fit method for training. For the LOOM

threshold, write your own piece of code based on the article (there is a very good explanation of the LOOM method in section IV.C). Some remarks about it:

- *N* is the number of input vectors for one feature and person (e.g. using proposed split 80%-20% we obtain 320 inputs vectors for each feature KD, DDKL and UUKL for each person);
- for each person and feature we iterate over all input vectors (choose one at the time), construct GMM from remaining vectors (*fit()* method) and calculate score for this one selected vector (*score()* method);
- based on the obtained scores choose the model threshold according to equation (2) in the article.

NOTE: GMM models have a “dual” role here. For each person there are three different GMM models, one for each feature (KD, DDKL, UUKL), which should be fit to all vectors of a specific feature from one person. These models represent the current person in terms of a given feature and should be used during evaluation (point 7 below). Beside them, there are also *N* GMM models for each feature and person, fit to *N*-1 input vectors and used only to determine the LOOM threshold value (as described above).

7. Once trained, our models have to be tested. Iterate over test vectors for each feature and calculate the score given by GMM models (one for each person). Reject matching if score is lower than threshold for this person. From preserved matches choose one with the highest score - it is the final person prediction. To have measurable outputs we can use simple accuracy - count percentage of good predictions from the whole test inputs for the given feature. Which feature is the most discriminative one?