SUR documentation

Martin Tomašovič

May 5, 2025

1 Introduction

In this document I describe how I created classifiers for audio and images to classify 31 persons from SUR 2024/2025 dataset. For each experiment, you can find a Python script in the /SRC/experiments/ folder. The final scripts to train and run best classifiers are in /SRC/ folder.

2 Image classifier

For images, I started with convolutional neural network, then I superficially tried to use PCA with LDA (this reached 41-45% accuracy), k-nearest neighbors, GMM, PCA with GMM and finally SVM. With SVM I experimented deeper.

2.1 Convolutional neural network

I tried a convolutional neural network (CNN) with three convoltional layers to classify the data. It did not perform well on this small dataset. This network was quickly over-trained. I tried to use augmentated data, which improved the accuracy. The best result I got was 31% and that was not stable. Weight initialization techniques were ineffective in improving the result.

2.2 Support Vector Machine

2.2.1 Experiment SVM version 1

I started with linear SVM and original dataset. The images are standardized using StandardScaler from sklearn. First I tried different image sizes. I found out that with 30x30 image size, the classifier performed best. I kept this setting in further experiments up to certain point, which I will describe later. The best results I got were 38.7% with 40x40 image and 40.32-41.94% with 30x30 image.

2.2.2 Experiment SVM version 2

Switching to radial basis function (rbf) kernel SVM with gamma set to scale and regularization parameter C set to 1 I got worse results 20% accuracy. Further experiments with rbf will be done later.

2.2.3 Experiment SVM version 3

I came back to linear SVM and used grid search to search for suitable regularization parameter (C). This proved to be effective rising to 42% accuracy.

2.2.4 Experiment SVM version 4

After loading images I extracted Histogram of Oriented Gradients (HOG) features, then applied standardization and linear SVM because that was the best I had in that moment. After grid search for best parameters I got 40.32% accuracy.

2.2.5 Experiment SVM version 5

I added PCA between standardization and SVM. I set the PCA to keep 95% variance. I also tried to search for best parameters here. Still I got 40.32%.

2.2.6 Experiment SVM version 6

I tried again rbf kernel, but now with HOG, standardization and PCA applied to data beforehand. I got 43.55%.

2.2.7 Experiment SVM version 7

I kept all settings as in version 6, but now I again experimented with image size. I found out that after applying all techniques, now it is more beneficial to keep the image size bigger. For 50x50 and 80x80 I got 58.6% accuracy.

2.2.8 Experiment SVM version 8

In this version, I tried to change processing. Including LANZOS resampling for aspect ratio preservation; intelligent padding to maintain square dimension and set the target image size to 80x80. In feature extraction I added gamma correction to HOG. And in SVM grid search I focused parameter grid around best-known values; increased maximal iterations and used tighter tolerance for stopping criteria (tol) for better convergence. It resulted in 62.90% accuracy.

2.2.9 Experiment SVM version 9

I added histogram equalization which improves local contrast and enhances feature visibility in low-contrast regions. In HOG I increased cells per block to 3x3 for context. These changes improved accuracy to 69.35%.

2.2.10 Experiment SVM version 10 and 11

In version 10 I used Jackknife to test how well the model would work when trained on all data – train and dev. I iterated over all data, trained a model on all data except one sample. I used this sample to evaluate the model. After all iterations I got accuracy 73.39%. Version 11 just trains the model on all data.

2.2.11 Experiment SVM version 12, 13 and 14

I decided to try to add augmentation to introduce geometry, color, and brightness variations observable in real world. This step improved accuracy of a model trained on just train part to 79% and that just with adding 10 images per original image.

Even better is that on Jackknife (version 13) I got 96.22% accuracy. I need to mention that in Jackknife I used even augmented dev set. The augmentation was performed by adding two combined transformations, which I kept after a few experiments. The first one consists of horizontal flip, random brightness contrast and a small rotation. The second one includes Gaussian blur, some affine transfomations, hue saturation change and also the horizontal flip but with smaller probability.

In version 14 the model is trained on all data with augmentation. The result of the 14th experiment is the final classifier I submit.

2.3 Further steps

To further improve the classifier, more experiments with different augmentations and more parameter tuning could be performed.

3 Audio classifier

For audio I decided to use Gaussian Mixture of Models (GMM) because it is suitable for small datasets and it can model complex probability distribution of audio features. First, I trained a GMM per class. Later, when I found out it did not perform as well as expected, I switched to one Universal Background Model (UBM) with GMM for each class, which did not outperform the initial model.

3.1 Gaussian Mixture of Models

First, I describe experiments with a GMM for each class.

3.1.1 Experiment GMM version 1

Initial setup. I cut first 1.25 second from each record and then trained a GMM on all features from a class. The model for each class had 16 components and diagonal covariance matrix type. It had around 52-56% accuracy. With a simple change of number of components, the accuracy improved. For 8 components cca 48%, and for 12 = 52% accuracy decreased. But for 32 = 53-60% and 64 = 53-63% the accuracy increased. I also tried to switch from diagonal to full covariance matrix. The results of full covariance matrix are in table 1.

Number of Components	Accuracy (%)
64	44
16	61
32	53
8	63
4	61

Table 1: Accuracy by number of components in GMMs with full covariance matrix

3.1.2 Experiment GMM version 2

Futher, I experimented with 8 component GMM. I increased maximal iterations to 500 and set multiple initializations to 3. Now, I reached 63-66% accuracy.

3.1.3 Experiment GMM version 3

I added scaler. With adding scaler I measured 63% accuracy, that is same as without it, which was 63-66%. The scaler can improve robustness, and when it don't decrease accuracy I decided to keep it.

But with scaler and increase of MFCC components to 20 from default 13 I got 74% accuracy. This was great move forward.

3.1.4 Experiment GMM version 4

I set appendEnergy=False, so the log energy of the audio frame is not included as the first coefficient in the MFCC vector during mfcc extraction and I got 77% accuracy.

3.1.5 Experiment GMM version 5

I tried to add delta features with setting N = 2. The delta features are calculated based on preceding and following N frames. With N = 2 I got accuracy 71%. With double delta feats, meaning combining mffc, delta, and delta of delta features both with N = 2 I got 65%. Later I experimented with more N numbers but it still did not improve the results. Adding delta features decreased the accuracy so I decided not to use them.

3.1.6 Experiment GMM version 6

I test here different FFT sizes using nfft parameter change - the audio is split into short overlapping windows FFT is applied to each step and nfft controls how many points the FFT computes. Up to now I used default 512 value and the classifier has 77% accuracy. Now I try 1024 and 2048 2. 2048 likely causes over-smoothing in the frequency domain, blurring out speaker specific details. The best results 81% were obtained with nfft=1024. I kept it for next experiments.

NFFT	Accuracy (%)
512	77
1024	81
2048	76

Table 2: FFT size change

3.1.7 Experiment GMM version 7

I tried to add PCA. This was ineffective, as the measurements show 3, so I don't use it further.

Number of components	Accuracy (%)
12	61
16	74
18	76
20	81

Table 3: GMM with PCA

3.1.8 Experiment GMM version 8

Again, I tried to increase the number of cepstral coeficients (NUM CEPS) with NUM CEPS = 24 got 77% accuracy and with decreased NUM CEPS = 16 got 68% accuracy. So I set the NUM CEPS back to 20. In this version I tried to remove silent parts of the records. And it helped. The results of tuning threshold are in table 4, I will keep the silence removal threshold set to 25 dB. So below 25 dB it is considered silence.

Threshold dB	Accuracy (%)
15	81
20	84
25	87
30	84

Table 4: GMM with silence removal

3.1.9 Experiment GMM version 9

I applied cepstral mean normalization, which normalizes MFCCs accross recordings by removing channel/microphone bias. I got 90% accuracy.

3.1.10 Experiment GMM version 10

I tried to reintroduce delta and doble delta features. N here means number of neighbour coeficients deltas are computed from. The results are N=1 acc=74%, N=2 acc=71%, N=3 acc=68%. Deltas didn't help. I will use version 9 as the best version I got and use Jackknifing and train a GMMs on all data train and dev.

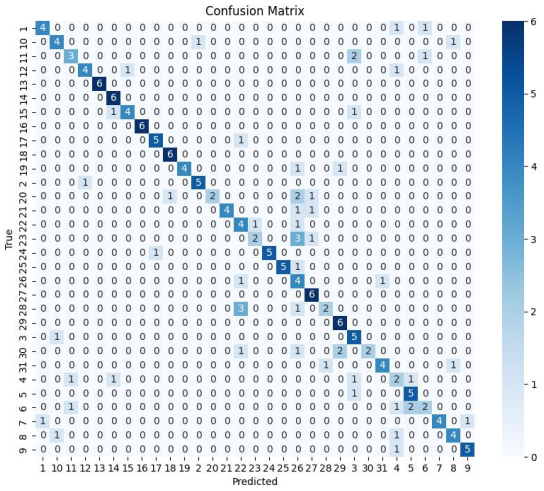

Figure 1: Confusion matrix of Jackknifing from experiment 11.

3.1.11 Experiment GMM version 11

Jackknifing resulted in 70% accuracy, in confusion matrix 1 it can be seen that the model learned to predict some classes better than others. This shows problems with generalization. Even though, I did further attempts to improve this score, I did not manage to improve it. With this setting I trained the model on all data train and dev I submitted it. The model is stored as audio_gmm_model.pkl.

3.2 Attempts to improve

I will not describe all further experiments in much detail, but I will summarize them here.

In version 14 after not as good Jackknife as expected, I try to use diagonal covariance and changed the number of GMM components, which resulted in increased accuracy to 92% 5. In version 16 I change covariance matrix to diagonal and I finetuned dB threshold for removing silence, the best I found is 15 dB and with GMM of 15 components, this setting reached 89% accuracy. I hoped that with the diagonal matrix the classifier would be more robust. Also in version 19, after silence removal I cut only a smaller part of the record, I found setting with 9 seconds to perform best, with 89% accuracy. On this new setting with diagonal matrix, added trimming and adjusted silence removal, I run Jackknifing and the result was 70% accuracy. This time however, for some classes it showed even smaller confidence, so I decided to keep the model from version 11.

When diagonal matrix experiments did not improve the accuracy. I tried to run Jackknife on models with full matrix that showed higher accuracy, first I tried model with 92% accuracy trained on train dataset. This model has increased GMM components to 10 and on Jackknife only 66% accuracy. Then, model with 95% accuracy on dev dataset. This model has just set 4 components, on Jackknife, so it should generalize better. But I got 69% on Jackknife.

The last thing I tried was to add Universal Background Model (UBM)[1]. First I tried separatly train UBM and GMMs for each class. This setup resulted at best in 90% accuracy tested on dev, tested on Jackknife I reached only 66%. Then, I tried to adapt the GMMs per class from UBM, but tested on dev I got 87% and on Jackknife only 65% accuracy, probably due to insufficient adaptation data per class or suboptimal adjustment of adaptation parameters. This approach may work, but it would need more parameter tuning.

3.3 Results

When training on train dataset and testing on dev dataset I achieved 95% accuracy best. However this model was overfit for dev dataset, which showed in jack knife test. So I settled with model achieving

[1] https://maelfabien.github.io/machinelearning/Speech1/#

Covariance Type	Number of Components	Accuracy (%)
diag	8	69
diag	12	73
diag	14	77
diag	16	74
full	6	92
full	8	90
full	10	92
full	12	92

Table 5: Accuracy by Covariance Type and Number of Components

90% on separate dataset training. This model achieved 70% accuracy on Jackknife.

3.4 Further steps

The accuracy could be improved by adding augmentated data to the audio part of the dataset. Further parameter tuning in UBM can also improve the accuracy. Or trying to switch to SVM model can improve the results.

4 Audio and image classifier fusion

I decided to combine the resulting scores for each class obtained from each classifier. The scores are normalized by Z-score normalization. The normalized scores are multiplied by weights for each classifier. The weights are computed from Jackknifing accuracy scores. Image classifier had 96% accuracy and audio classifier had 70% accuracy. I calculated weights this way:

$$\begin{aligned} \text{Image accuracy} &= 0.96 \\ \text{Audio accuracy} &= 0.70 \\ \text{Total accuracy} &= 0.96 + 0.70 = 1.66 \\ \text{Image weight} &= \frac{0.96}{1.66} \approx 0.58 \\ \text{Audio weight} &= \frac{0.70}{1.66} \approx 0.42 \end{aligned}$$

From the new scores I chose the predicted class using argmax. The results are stored in the file combined_audio_image.

5 Run code

Codes were developed with Python 3.12.3 on Windows 11.

5.1 Libraries

To use all the scripts, please install following libraries: scikit-learn, python_speech_features, librosa, seaborn, matplotlib, numpy.

For audio install wavfile.

For image augmentation install pillow and albumentations, which also installs open-cv2.

5.2 Run training and experiments

All scripts to run training of final classifiers are in SRC folder. All experiments are in folder

```
SRC\experiments\
```

They are named by the type of modality – image or audio, type of model – svm, gmm and order number of experiment.

The dataset from assignment needs to be preprocessed before using the scripts. In SRC folder, you need to download and add SUR_projekt2024-2025, SUR_projekt2024-2025_eval and generate Augumented_data_2_all, Separate_data and they need to be copied to experiments folder to reconstruct experiments. You can create the folders by following these steps. First place the original SUR_projekt2024-2025.zip file to the desired folder, unzip it and run separate_modality_folders.py. Then copy Separate_data folder, rename it to Augumented_data_2_all and run augumentation2.py to create augmentations.

The training scripts store the final model files in pickle respectively joblib formats in the same directory in which the scripts are, so they can be used later. To run training run scripts in SRC folder.

```
python image_svm_14_all_data_aug.py
python audio_gmm_v25_train_all.py
```

5.3 Run trained models

To run trained models ensure models image_classifier_model.joblib and audio_gmm_model.pkl is in the same folder and a folder SUR_projekt2024-2025_eval is in the same folder as the python script, so it can reach data, for example images

```
SUR_projekt2024-2025_eval\eval\eval_00001.png
```

and simply run scripts:

```
python run_image_classifier.py
python run_audio_classifier.py
```

The results are stored in files audio_gmm and image_svm files. To fuse the classifiers first compute results for audio and image, then run script combine_image_audio.py, the results will appear in file combined_audio_image.