Speech Emotion Recognition Project

Vaggelis Lamprou

0. Introduction

The repository presents a SER study on the "Crema D" dataset. The original dataset consists of 6 classes but for the purposes of this repo we focus on a 4-class variation, by considering only the "happy", "neutral", "sad" and "angry" emotions. The dataset consists of 48 males and 43 female speakers (91 in total); most of them contributing 14 "happy", "sad" and "angry" samples and 12 "neutral" samples. There are also a few speakers who contribute 13 and 11 samples per class respectively. The study adopts a speaker independent approach for the construction of the training, validation and test sets.

Kindly note that for the rest of the report, the section enumeration follows the "SER_classification_task.ipynb" Jupyter notebook section enumeration and focuses on presenting a detailed summary of each section.

1. Crema dataset

This section introduces the ***get_features_and_labels*** function which takes as input the Crema-D dataset directory and is responsible for :

- **Separating the 4 class samples** of interest per speaker from the remaining class samples.
- **Feature extraction** of the sample points by the "mid_feature_extraction" function of pyAudioAnalysis, at parameter levels : *mid_window = 0.5*, *mid_step = 0.25*, *short_window = 0.05* and *short_step = 0.05*. Each point is eventually described by a 136-dim vector.
- Assigning **labels** to the datapoints.

- Randomly splitting data into -speaker independent- **training** and **test sets**, of **82** and **9 speakers** respectively, corresponding to about **90%** and **10%** of the data.
- Learning a **standard scaler on the training data**.

We also note that the scaled training speaker features are stored in a 82-element list where each element corresponds to a np.ndarray with the features of a different speaker. Similarly, a training speaker labels list is defined (refer also to **figure 1.1** of the ipynb file for a visual description of these two lists). The reason for this list speaker-structure is that it enables construction of multiple speaker independent train/validation sets in section 5 (see *training_validation_sets* function).

On the other hand, the test data and labels are directly concatenated in np.arrays for testing purposes in section 6. For these data, scaling -with the learned scaler- will take place in section 6.

Finally, a ***multispeaker_data*** function is defined and throughout the study it is used for concatenating features and labels for different training speakers of interest.

2. Dataset visualization with PCA

At this point we attempt to get a visual perspective of the training dataset by transforming it in the direction of its **first 3 principal components**.

The resulting 3D plot, as per figure 2.1 in ipynb file, seems rather confusing as the points of the different classes seem to overlap a lot. However, one might guess that the classes **sad-angry** and **happy-sad** do not overlap as much as the rest classes. This becomes more clear in **figures 2.2** and **2.3** respectively, which focus only on the aforementioned combinations, and eventually aligns "well" with both the learned model results (section 5) and the test experiment results (section 6) as we see that the model does not make many wrong predictions when looking at these pairs only. The remaining two figures show overlap cases of more 2-class combinations.

3. Correlations

In this section we examine the training features correlations to the targets. For this purpose the ***correlation*** function is introduced and is responsible for finding those features that are at least correlated to the targets by the absolute value of the ***correlation_level*** parameter.

Overall we see that the correlations range in quite low values. The features which exceed the 0.2 threshold are spectral_entropy_mean, mfcc_2_mean and mfcc_2_std with correlation values 0.23, 0.21 and 0.22 respectively.

4. Features' Value Distributions

In continuation to section 3, we study a bit further the three aforementioned features. More specifically, in figures 4.1, 4.2 and 4.3, we draw the **feature boxplots** for the different classes and observe that in all cases, even when the medians per class are "not close" (see "sad" vs "angry" in all boxplots), there is still considerable overlap between the values of the features which prevents us from drawing safe classification conclusions.

5. Evaluate models and get optimal

Section 5 is concerned with finding a classifier that fits "well" to the training speakers features.

It consists of three main functions :

- The ***training_validation_sets*** function, which splits the training dataset into two **speaker independent training and validation datasets**, consisting of **66 and 16 speakers respectively**. This amounts to about **72%** and **18%** of the initial Crema dataset. The function performs this procedure ***n_exp*** **times**, as inserted by the user.

  The resulting training and validation datasets are stored in two lists of *n_exp* np.arrays each. Kindly refer to **figure 5.1** of the ipynb file

to see a visual description of these two lists. A similar structure also holds for the training and validation labels.

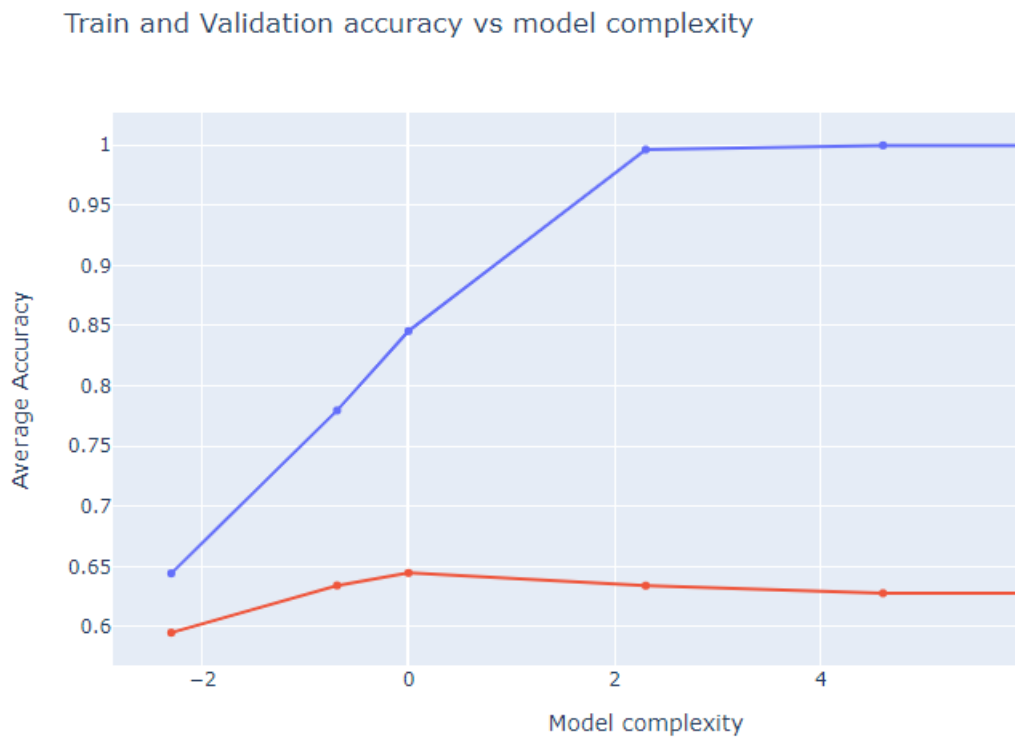
- The ***evaluate_classifiers*** function, which performs ***n_exp*** train/ validation experiments for multiple SVM, Random Forest and k-NN classifiers at a *fixed* ***correlation_level***. It eventually returns the "optimal" model in terms of the **average F1 score** over all the experiments.

  In addition this function stores in a dictionary many useful metric values for informative and plotting purposes. One may refer to the description of the function's outputs in the ipynb file for a detailed description.

- The ***evaluate_and_train_final_model*** function, which at first performs multiple $evaluate\_classifier$ function calls at different correlation levels and eventually **selects the optimal model**, again with respect to the **average F1 score** after ***n_exp*** experiments. It finally returns the optimal **trained model** over the **entire training speakers dataset**.

  In addition, the function provides summary information, plot and metric results for both the overall optimal model and the study performed among the different $correlation\_levels$. One may refer to the function's outputs in the ipynb file for a detailed description.

In the end, by applying the above functions to the training dataset for ***n_exp*=20** experiments we get that the best fitted classifier on our dataset is a **SVM** trained on all **136 features** ($correlation\_level = 0$) with **C=1** and **average F1 score equal to 0.64** .



blue : average train accuracy
red : average validation accuracy

6. Testing

In the last section of the study, we test the learned model on the 9-speaker test dataset constructed in section 1. After scaling the data with the learned scaler we get the following metric results :

- Confusion matrix :

	happy	neutral	sad	angry
happy	74	25	11	16
neutral	21	60	19	8
sad	13	21	91	1
angry	32	14	4	76

  Here we highlight that, as already discussed in section 2 of the report, the model tests well when looking at the pairs 'sad-angry' and 'happy-

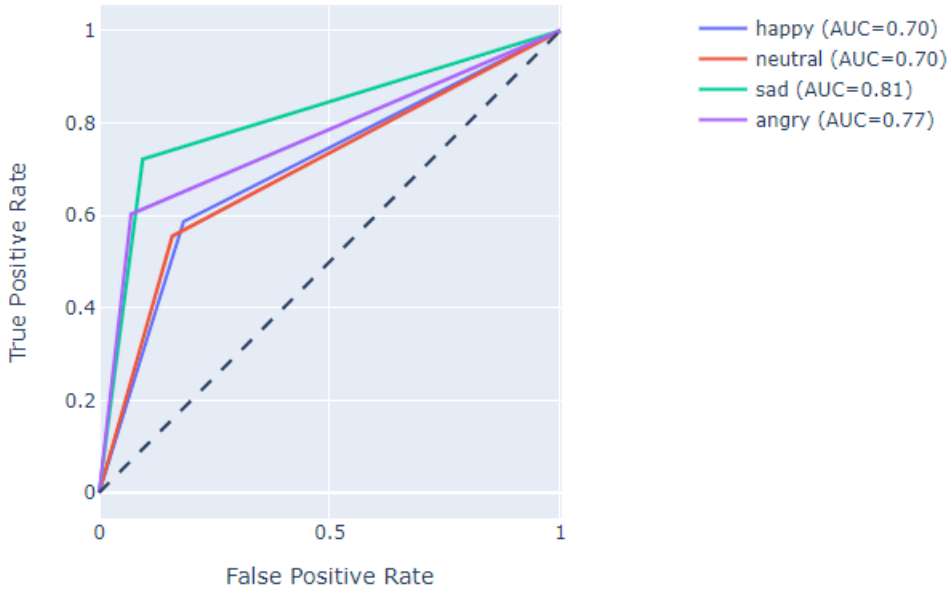
sad'. In other words, it does not confuse much sad with angry emotions and happy with emotions, compared to the rest 2-class combinations.

- Classification test report :

	precision	recall	f1-score	support
happy	0.53	0.59	0.56	126
neutral	0.50	0.56	0.53	108
sad	0.73	0.72	0.73	126
angry	0.75	0.60	0.67	126
accuracy			0.62	486
macro avg	0.63	0.62	0.62	486
weighted avg	0.63	0.62	0.62	486

Note that accuracy = 0.62, macro f1-score = 0.62

- ROC curves :



By the AUC scores we see that the "angry" and "sad" emotions are ones best classified by the model.
In additoin, recalling figure 2.2 of the training set, their values seem to overlap less than the rest combinations and are spread more towards

the edges of the 3D data distribution of figure 2.1. This proves to be in congruence with the test data results presented here as well.

- Precision-Recall curves :