NATIONAL CENTRE FOR SCIENTIFIC RESEARCH "DEMOKRITOS"

UNIVERSITY OF THE PELOPONNESE

MSc in Data Science
Multimodal information processing and analysis

P6: Lecture speech analytics

February 2020

Dimitros Konstantinos, Karozis Stelios, Karousis Konstantinos, Mastrapas Anastasios, Nikoloutsakos Nikos

1. Introduction

The current project is assigned within the framework of the “Multimodal information processing and analysis” course of MSc in Data Science. The goal of the project is to process video and audio data, extract features, apply a Machine Learning algorithm and create an end-to-end process. Moreover, it is mented to be a simulation of work of a real work project, with collaboration of partners, assignment of roles, time management etc.

1.1 Business cases

If we wanted to give business dimensions to our project, we think that such a project could be part of several business cases, which would be intended to translate speech features into value. For instance, a company that organises conferences could use such an application for the assessment of the speakers so as to improve the quality of the conferences. Furthermore, a University could use speech analytics to make assessment of the lecturers. Thinking broader, speech analytics are able to help corporations deliver a great customer experience. Speech analytics gives a view into what customers actually think about company's brand, products or services, and the overall experience they have. Having access to this vital information, a company can take proactive measures to make changes and improvements. Speech analytics has become an essential tool for providing insights and advantages for all departments within a company. For instance, speech analytics could be deployed to the call center, since it is often the initial contact a customer has with a company. Speech analytics can analyze conversations via phone, email, text, webchat and social and turns them into searchable, structured data that businesses can use to gain insight into what customers feel or think.

1.2 Program language and tools

The project was created in Python 3 programming language and the code is available in a github public repository[1]. The Trello[2] task manager used in order the team member to collaborate more efficiently and a Nextcloud[3] server was used to exchange big data.

Figure 1.2.1: Logos of tools and programming language used

[1] https://github.com/cjd1884/pyLectureMultiModalAnalysis
[2] https://trello.com/b/79lO6VWx/multimodal
[3] https://mssg.ipta.demokritos.gr/owncloud

1.3 Datasets

The datasets used for training, test and demonstration are either video lectures from Youtube or videos recorded during the lectures. The latter were used only in the demonstration of the assignment.
For the training and test case, the Youtube video lectures were segmented and annotated by hand (see Appendix A & B).

Project Stage	**Datasets**
Training	Youtube Lectures
Testing	Youtube Lectures
Demonstration	“Multimodal information processing and analysis” Lectures

Table 1.3.1: Datasets used in each stage of the project

2. Methodology

The project is divided into two Parts.
Part 1 includes the construction of the dataset and then both training and testing of the model. In Part 1 our goal is to build a classifier using video lectures found on Youtube. The methodology followed in this section includes the following steps:

1. Find on Youtube and download 5 video lectures from different speakers. These videos are the input of our system.
2. Apply segmentation on the input videos taking as output 50 video lecture segments per speaker. The duration of these segments is approximately 15 seconds.
3. Annotate the video segments giving to each segment one of the three labels: boring, neutral, exciting. Each speaker was assigned to each group member.
4. Extract video features from each video segment of each speaker.
5. Extract audio features, after extracting audio from video (using ffmpeg).
6. Merge features in a common dataframe. At the end of this step we had a first version of our dataset completed, including both features and labels.
7. Apply data pre-processing to the dataset. More specifically, standardisation was applied to the features so as to have mean 0 and standard deviation 1.
8. Build the classifier. SVM algorithm was selected while ove-vs-all validation was applied to the dataset for the training of the model. Finally, the scoring (accuracy) of the model was produced followed by a confusion matrix per speaker.

Part 2 includes the demonstration stage of the project. The input in this part is an unseen video lecture. Steps 2, 4, 5 and 6 of Part 1 are repeated to construct the new dataset. Annotation in this Part is implemented using the classifier of Part A, with which we assign (predict) labels to the unseen instances. Finally, a summarization video is created: the user inserts the desired video duration (e.g. 2 min) and takes as output a piece of video which displays the predicted label on each video segment.

3. Results

In this section the results of the experiments in Part 1 of the project are presented.

3.1 Accuracy

The overall accuracy of the model is **0.316,** which is very low.

We are thinking of two possible reasons that could explain the low performance of the model:

1. There has been possibly bad annotation in the 3rd step of the process. We had considered in advance the annotation task as a difficult one, given that 5 different annotators would assign labels to the video segments. Maybe the application of a clustering algorithm to distinguish the instances in different groups or even the use of NLP algorithms to take also into account the video lectures' content could help.
2. We have a dataset with 250 instances and 168 features. It is possible that the number of the examples is not sufficient and does not help the classifier to be trained well leading it to underfit.

3.2 Confusion matrices

In this section the confusion matrix of the classification of the video segments per speaker is presented.

Actual \ Predict	'boring'	'neutral'	'interesting'
'boring'	0	13	4
'neutral'	0	12	4
'interesting'	0	15	2

Table 3.2.1: Confusion matrix for Speaker 1

Actual \ Predict	'boring'	'neutral'	'interesting'
'boring'	1	0	7
'neutral'	2	0	26
'interesting'	0	0	14

Table 3.2.2: Confusion matrix for Speaker 2

Actual \ Predict	'boring'	'neutral'	'interesting'
'boring'	0	21	0
'neutral'	0	12	0
'interesting'	0	17	0

Table 3.2.3: Confusion matrix for Speaker 3

Actual \ Predict	'boring'	'neutral'	'interesting'
'boring'	0	5	5
'neutral'	0	15	12
'interesting'	1	6	6

Table 3.2.4: Confusion matrix for Speaker 4

Actual \ Predict	'boring'	'neutral'	'interesting'
'boring'	17	0	0
'neutral'	18	0	0
'interesting'	15	0	0

Table 3.2.5: Confusion matrix for Speaker 5

Appendix A: Functions

A.1 Video segmentation

Video files are partitioned into segments, approximately 20s long each. The first step is to determine the candidate segmentation points. To that end, we first apply the `silenceRemoval()` function from the pyAudioAnalysis library on the associated audio signal. This step is followed by a mild post-processing step, whereby longer segments are created, provided that their length is less than 20s, by merging successive candidate segmentation points. The second step consists of matching the associated video file to the audio signal and then applying the segmentation points to the video file using the ffmpeg tool.

segment_audio(audio_fn, audio_dir, st_win, st_step, up_bound)

Scope: Utility function to determine segmentation points of an audio signal.
Arguments:

- audio_fn = [string]
  - The filename of the audio signal
- audio_dir = [string]
  - Directory containing the audio signal
- st_win, st_step = [floats]
  - Short term window and step (arguments of `silenceRemoval()`)
- up_bound = [float]
  - Upper bound on the duration between successive segmentation points; only used when the time difference between two successive segmentation points is smaller than the bound.

segment_medium

Scope: Main function which segments a video file. The segmentation points are determined by invoking the `segment_audio()` function. The video file is matched to the input audio signal.
Arguments:

- audio_fn = [string]
  - The filename of the audio signal
- audio_dir = [string]
  - Directory containing the audio signals
- media_dir = [string]
  - Directory containing the video files
- out_dir =string]
  - Directory where the video segments will be stored
- st_win, st_step = [floats]
  - Short term window and step (arguments of `silenceRemoval()`)
- up_bound = [float]
  - Upper bound on the duration between successive segmentation points

input2seg(audio_dir, video_dir, output_folder)

Scope: The function combines all the A.1 Section's functions in order to be used from the main.
Arguments:

- audio_dir = [string]
  - Path to create the audio files that will be used in order to find the silent points
- video_dir = [string]
  - Path of video input(s)
- output_folder = [string]
  - Folder to put the segmented videos

A.2 Audio feature extraction

PyAudioAnalysis[4] library is used to extract audio mid-term features of the WAVE files.
It takes a list of audio files as inputs and returns a list of feature matrices. The audio files are listed in an index CSV file. The resulting feature vector is extracted by long-term averaging the mid-term features. Therefore ONE FEATURE VECTOR is extracted for each WAV file. The result is saved in a pickle dataframe format.

During development also the plot_feature_histograms from the MultimodalAnalysis examples[5] was also copied in the source code to help verify the audio features results.

video2audio

Scope: Utility function to convert video file segments to audio only WAVE files. Recursively convert all the video files in a directory to mono channel WAV and 16k bit rate file using the ffmpeg utility.
Arguments:

- data_path= [string]
  - Data path that contains the video files

audio_features_extraction

Scope: Create a dataframe with the audio features of all segments. This function extracts the mid-term features of the WAVE files from the index CSV.
Arguments:

- dir_name = [string]
  - Directory that contains the input audio files
- mt_win=[float] (defaul value=1.0), mt_step =[fload] (default value 1.0)
  - mid-term window length and step (in seconds)

[4] https://github.com/tyiannak/pyAudioAnalysis
[5] https://github.com/tyiannak/multimodalAnalysis

- st_win= [float] (default value = 0.050), st_step= [float] (default value = 0.050)
  - short-term window and step (in seconds)
- features_audio_file=[string]
  - 'Audio2Features.pkl': the dataframe file name

A.3 Video feature extraction

The video feature extraction was based in pre-trained VGG16[6] deep neural network. The attributes of the last hidden layer multiplied with a random vector, were used as features in the current process. Due to the number of attributes the last hidden layers holds (over 64000) the classification would be dominated by the video features, hence we diminish the effect by multiply the feature vector with a random vector of smaller size. The random vector is kept for future use. Each segment produces many images, according to the desired frame rate, and for each video-part the average values of each feature is kept.
The aforementioned procedure is implemented via the use of three functions: RandomVector(), video2frame(), frame2features() and Video2feature()

RandomVector(trainmode, sz)

Scope: Create the random vector in order to diminish the size of video feature vector
Arguments:

- trainmode = [True or False]
  - If true a new vector is created, else the functions check if there is an existing one.
- sz = [integer]
  - The desired size of the vector

video2frame(count, sec, folderVID, file, folderIMG)

Scope: Create frame from video files with the desired frame rate
Arguments:

- count = [integer]
  - Counter of segment used
- sec = [float]
  - timestamp of video in seconds (indirect declaration of FPS)
- folderVID = [string]
  - path of video folder
- file = [string]
  - segment used
- folderIMG = [string]
  - path of image folder

frame2features(frame)

Scope: Image feature extraction using VGG16
Arguments:

[6] https://neurohive.io/en/popular-networks/vgg16/

- frame = [string]
    - image/frame used in feature extraction

Video2feature(pathIn, frameRate, save)

Scope: Main process that use all the above functions to create a dataframe with the video feature of all segments
Arguments:

- pathIn = [string]
    - path of segment
- frameRate = [float]
    - frame rate that produce frames
- save = [True or False]
    - If True the dataframe of video features is saved in pickle format

A.4 Classification

evaluate_training(df)

Scope: Evaluates the algorithm performance on learning the provided dataset. Run in rounds - each speaker versus all.
Arguments:

- df = [dataframe]
    - the dataframe with audio & video data

train(df, data_dir='data')

Scope: Trains an SVM model in the provided dataset.
Arguments:

- df = [dataframe]
    - the dataframe with audio & video data
- data_dir = [string]
    - the data directory

A.5 Labels2Summary

The summarization process creates a video output with the desired percentage of each label. The input is the classified instances of each video part. The user can choose the percentage of each label alongside the maximum duration of the video. Due to the fact that the segments are not of fixed length but rather are the result of silent segmentation, the average duration of video segments is taken into account during the creation of fixed video duration.

add_duration(df, folder)

Scope: Takes a dataframe with video segments and adds a column with the duration for each segment
Arguments:
- df = [dataframe]
- folder = [string]
  - name of input video folder of segments

labels2summary(df, col_cl, label, prc_l, col_d, duration_sec)

Scope: The function use the users preference of labels percentage and video duration and creates a dataframe with parts that will create the video summary.
Arguments:
- df = [dataframe]
  - A dataframe (derived from the SVM classification)
- col_cl = [string]
  - The column name of the classes
- label = [array]
  - Array of classes values
- prc_l = [array]
  - Array of percentage usage (in the final video) of each class
- col_d = [string]
  - The column name of the duration
- duration_sec = [integer]
  - Desirable duration of output video. If -1 is used, the final duration of video depends on the percentages of the labels (all segments are used)

summary2video(df, output_folder, output_name)

Scope: Concatenate video segments in one video file
Arguments:
- df = [dataframe]
  - dataframe that includes the segments that will be used for the video]
- output_folder = [string]
  - name of folder of output video file
- output_name = [string]
  - name of output video file with video suffix

df2summary(df, folderDATA, col_cl, label, prc_l, col_d, duration_sec, output_folder, output_name)

Scope: The function combines all the A.5 Section’s functions in order to be used from the main.
Arguments:
- df = [dataframe]

    - A dataframe (derived from the SVM classification)
- folderDATA = [string]
    - name of input video folder of segments
- col_cl = [string]
    - The column name of the classes
- label = [array]
    - Array of classes values
- prc_l = [array]
    - Array of percentage usage (in the final video) of each class
- col_d = [string]
    - The column name of the duration
- duration_sec = [integer]
    - Desirable duration of output video. If -1 is used, the final duration of video depends on the percentages of the labels (all segments are used)
- output_folder = [string]
    - name of folder of output video file
- output_name = [string]
    - name of output video file with video suffix