ESpeechPy

An audio emotion recognition system

ESpeechPy è un sistema basato su un'agente intelligente che reagisce alle emozioni di una persona dopo averne ascoltato la voce.

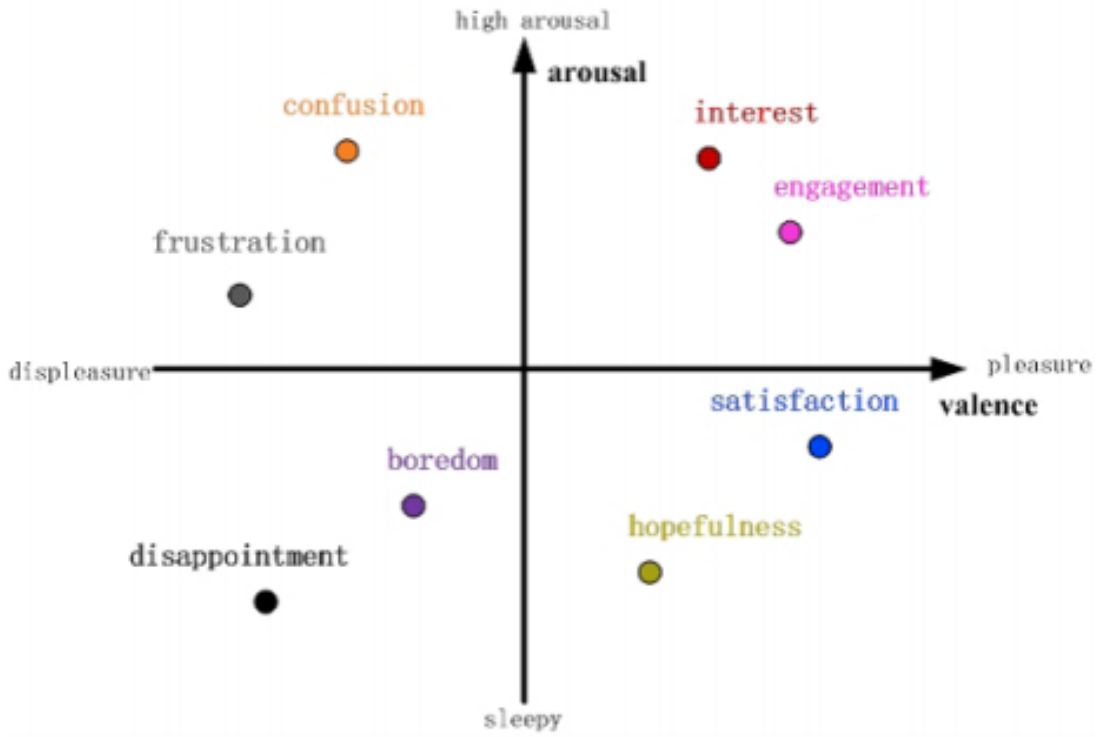
M. Metta, P. Sangermano, C. Pace, G. Semeraro, M. Mariano, N. Nargiso

07/06/2019

INDICE

1. INTRODUZIONE

Nel mondo informatico, spesso ci si trova a dover affrontare segnali analogici di diverso tipo. Uno di questi, quello che prenderemo in considerazione, è il segnale audio.
In informatica, questo può essere sfruttato in diversi ambiti che variano dalla semplice manipolazione dell'audio (equalizzazione, applicazione di filtri, ecc.), fino alle applicazioni che lo utilizzano come parametro biometrico per, ad esempio, l'identificazione di un individuo.
Il segnale audio, elemento fondamentale per la voce umana, può essere utilizzato per definire lo stato d'animo di colui che parla; le emozioni possono essere rilevate attraverso determinati tipi di features (caratteristiche che spiegheremo nel dettaglio in seguito) secondo due parametri specifici: l'*arousal* (stato generale di attivazione e reattività del sistema nervoso, in risposta a stimoli interni -soggettivi- o esterni -ambientali e sociali-) e la *valence*.
Arousal e Valence non sono parametri prettamente "informatici", ma provengono da studi psicologici e psicosomatici che delineano in modo preciso la sfera emozionale di appartenenza di ciascuna emozione.
Secondo il modello cartesiano, le emozioni possono rappresentate come punti in cui l'asse delle x rappresenta la *valence* e l'asse delle y rappresenta *l'arousal*. Il grafico si presenta approssimativamente così:



Facendo affidamento su questi studi e documenti scientifici, nel corso degli anni si è riusciti ad ottenere risultati sempre più che soddisfacenti in questo campo anche se solo in ambienti controllati

(senza rumore) e con audio che rasentavano la perfezione dal punto di vista acustico.
L'analisi di questi studi ha generato l'idea di un'agente intelligente in grado di racchiuderli e interpretarli.
Un'agente è un'entità intelligente dotata di pensiero, in grado di elaborare e prendere decisioni autonomamente.

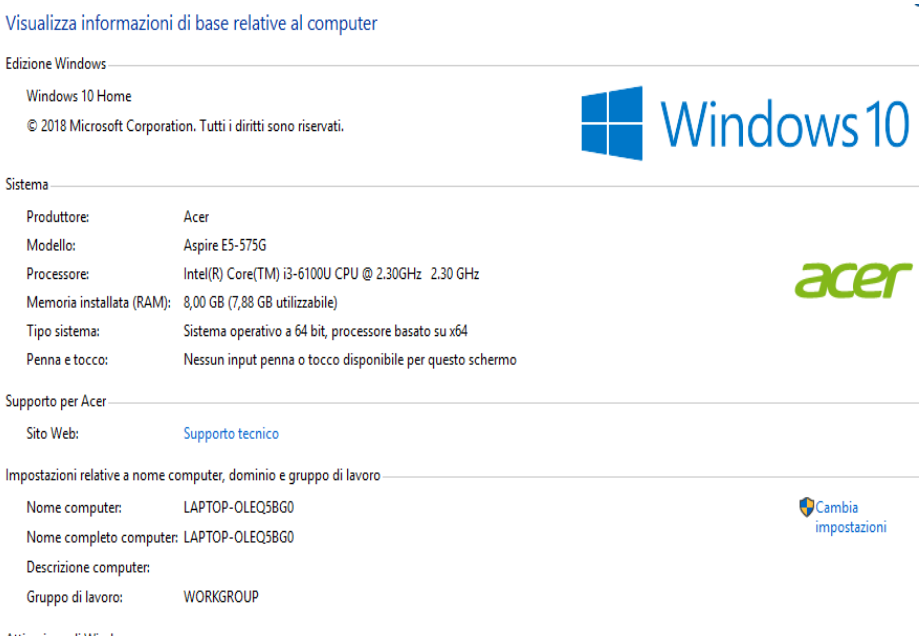
1. OBIETTIVI

ESpeechPy è un sistema basato su un'agente intelligente che, dopo aver riconosciuto un'emozione, restituisce un output in cui il reagisce in maniera consona a ciò che ha percepito.

L'interfaccia di ESpeechPy è costituita dalla figura di un "draghetto" che rappresenta sia il punto di partenza del sistema, sia l'output dell'agente: esprimerà la reazione all'emozione rilevata tramite una breve animazione.

2. DOCUMENTAZIONE TECNICA

a. SPECIFICHE DEL PC (ESEMPIO DI ARCHITETTURA)



b. DATASET

Il primo passo per il riconoscimento dell'emozioni attraverso l'audio è l'acquisizione di una fonte di conoscenza preesistente: nel nostro caso EMOVO, un dataset, completamente in italiano, costituito da 588 file audio (formato Wav, 48 MHz, 16 bit), diviso per interpreti (3 voci maschili e 3 femminili). Ciascun interprete ha registrato 14 audio per 7 emozioni che sono: *Rabbia, Gioia, Neutrale, Sorpresa, Disgusto, Tristezza e Paura.* Il nostro Team ha voluto ampliare il dataset estrapolando audio da ulteriori doppiatori: abbiamo raggiunto quota 963 audio.

c. LINGUAGGIO E LIBRERIE UTILIZZATE

Il sistema è stato sviluppato in linguaggio Python tramite *PyCharm*, un'ide basata sull'interprete *Python 3.7.*
Le librerie utilizzate sono le seguenti:

- *Wave* e *Array*: utilizzate per trasformare l'audio registrato da mono a stereo
- *Asyncio*: correlata a Spade
- *Spade*: utilizzata per la creazione dell'Agente in python
- *Python-opencv*: utilizzata per la manipolazione dei frame dei video, output dell'agente
- *Numpy* e *Scipy*: utilizzata come support per il classificatore
- *Piglet*: utilizzata per la realizzazione dell'interfaccia dell'applicazione
- sklearn: utilizzato per la creazione e l'addestramento del classificatore
- *pyAudioAnalysis.audioFeatureExtraction, Praat_feature*, *librosa*, *python_speech_features*: librerie usate per l'estrazione di ulteriori feature
- Sklearn: utilizzato per la creazione e l'addestramento del classificatore
- *Speech_recognition*: uno dei moduli utilizzati per l'estrazione delle feature dell'audio
- *Pickle*: utilizzato per la serializzazione del classificatore addestrato

d. CLASSIFICATORE

La libreria utilizzata per creare e addestrare il classificatore è *sklearn*. Questa fornisce una notevole quantità di classificatori che sono stati opportunamente addestrati (con le feature rilevate da altre librerie che verranno illustrate nel prossimo

paragrafo) e messi a confronto (per scegliere il migliore). In seguito, verranno forniti le differenze e il codice per l'addestramento di ciascun classificatore per le predizioni:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.naive_bayes import GaussianNB
```

a) Importare tutte le librerie necessarie sia per dare una forma ai dati estratti dal dataset (pandas) sia per utilizzare i classificatori (sklearn)

```
#filename_glass = "temp.xlsx"
filename_glass = "dataset_temp.xlsx"
#filename_glass = "dataset_DEFINITIVO.xlsx"
df_glass = pd.read_excel(filename_glass)
```

b) Caricamento delle feature estratte dagli audio del dataset EMOVO (dataset_temp.xlsx) o EMOVO_ESTESO (dataset_DEFINITIVO.xlsx)

```
def get_train_test(df, y_col, x_cols, ratio):
    """
    This method transforms a dataframe into a train and test set, for this you need to specify:
    1. the ratio train : test (usually 0.7)
    2. the column with the Y values
    """
    #mask = np.random.rand(len(df)) &lt; ratio
    #mask = np.random.rand(len(df)); #la funzione np.random.rand restituisce un array di un certo numero di elementi
                                    #pari al valore intero passatogli come paramentro. Gli elementi dell'array sono
                                    #valori compresi tra 0 e 1
    #df_train = df[mask]
    #df_test = df[~mask]

    df_train, df_test = train_test_split(df, test_size=0.30)

    Y_train = df_train[y_col].values
    Y_test = df_test[y_col].values
    X_train = df_train[x_cols].values
    X_test = df_test[x_cols].values

    return df_train, df_test, X_train, Y_train, X_test, Y_test
```

c) Definizione della dimensione del training_set e del testing_set per i classificatori

---

```
y_col_glass = 'Emozione'
x_cols_glass = list(df_glass.columns.values)
x_cols_glass.remove(y_col_glass)

train_test_ratio = 0.7
df_train, df_test, X_train, Y_train, X_test, Y_test = get_train_test(df_glass, y_col_glass, x_cols_glass,train_test_ratio)

dict_classifiers = {

    "Logistic Regression": LogisticRegression(),
    "Nearest Neighbors": KNeighborsClassifier(),
    "Linear SVM": SVC(),
    "Gradient Boosting Classifier": GradientBoostingClassifier(n_estimators=1000),
    "Decision Tree": tree.DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(n_estimators=1000,random_state=0),
    "Random Forest Regressor": RandomForestRegressor(n_estimators=1000, random_state=0),
    "Neural Net": MLPClassifier(alpha = 1),
    "Naive Bayes": GaussianNB(),

}
```

d) Definizione del rapporto fra audio di “addestramento” - training - e audio di “test” – testing -: rapporto 70-30. Definizione del dizionario che conterrà i risultati della classificazione effettuata sugli audio del dataset.

```
def batch_classify(X_train, Y_train, X_test, Y_test, no_classifiers=5, verbose=True):
    """
    This method, takes as input the X, Y matrices of the Train and Test set.
    And fits them on all of the Classifiers specified in the dict_classifier.
    The trained models, and accuracies are saved in a dictionary. The reason to use a dictionary
    is because it is very easy to save the whole dictionary with the pickle module.
    Usually, the SVM, Random Forest and Gradient Boosting Classifier take quiet some time to train.
    So it is best to train them on a smaller dataset first and
    decide whether you want to comment them out or not based on the test accuracy score.
    """
    dict_models = {}
    for classifier_name, classifier in list(dict_classifiers.items())[:no_classifiers]:
        #t_start = time.clock()
        t_start = time.perf_counter()
        classifier.fit(X_train, Y_train)
        #t_end = time.clock()
        t_end = time.perf_counter()
        t_diff = t_end - t_start
        train_score = classifier.score(X_train, Y_train)
        test_score = classifier.score(X_test, Y_test)

        dict_models[classifier_name] = {'model': classifier, 'train_score': train_score, 'test_score': test_score,
                                        'train_time': t_diff}
        if verbose:
            print("trained {c} in {f:.2f} s".format(c=classifier_name, f=t_diff))
    return dict_models
```

e) Addestramento di ciascun classificatore e stampa del tempo impiegato per la classificazione

---

```

def display_dict_models(dict_models, sort_by='test_score'):

    cls = [key for key in dict_models.keys()]
    test_s = [dict_models[key]['test_score'] for key in cls]
    training_s = [dict_models[key]['train_score'] for key in cls]
    training_t = [dict_models[key]['train_time'] for key in cls]

    df_ = pd.DataFrame(data=np.zeros(shape=(len(cls), 4)),
                       columns=['classifier', 'train_score', 'test_score', 'train_time'])
    for ii in range(0, len(cls)):
        df_.loc[ii, 'classifier'] = cls[ii]
        df_.loc[ii, 'train_score'] = training_s[ii]
        df_.loc[ii, 'test_score'] = test_s[ii]
        df_.loc[ii, 'train_time'] = training_t[ii]


    #display(df_.sort_values(by=sort_by, ascending=False))
    print(df_.sort_values(by=sort_by, ascending=False))


```

f) Stampa del dizionario con nome del classificatore, accuratezza ottenuta sul training set, accuratezza ottenuta sul test set e tempo di training

g) RISULTATI CON DATASET EMOVO

	classifier	train_score	test_score	train_time
5	Random Forest	1.000000	0.672316	1.989251
3	Gradient Boosting Classifier	1.000000	0.627119	5.056969
4	Decision Tree	1.000000	0.531073	0.004600
0	Logistic Regression	0.401460	0.389831	0.018028
6	Random Forest Regressor	0.907852	0.347215	3.225850
8	Naive Bayes	0.391727	0.327684	0.001030
1	Nearest Neighbors	0.520681	0.305085	0.000628
7	Neural Net	0.236010	0.276836	0.193631
2	Linear SVM	1.000000	0.107345	0.024546

h) RISULTATI SU EMOVO_ESTESO

	classifier	train_score	test_score	train_time
5	Random Forest	1.000000	0.539792	2.850700
3	Gradient Boosting Classifier	1.000000	0.460208	9.308987
4	Decision Tree	1.000000	0.380623	0.011816
0	Logistic Regression	0.323442	0.318339	0.024586
8	Naive Bayes	0.298220	0.314879	0.001274
2	Linear SVM	1.000000	0.252595	0.061609
6	Random Forest Regressor	0.886688	0.211917	6.016947
1	Nearest Neighbors	0.405045	0.211073	0.000874
7	Neural Net	0.152819	0.155709	0.145770

A fronte dei risultati ottenuti si è deciso, per il nostro sistema, di utilizzare un classificatore di tipo *random forest* con il quale, nonostante non si raggiungano risultati ideali, ci si avvicina a una soglia di accettabilità tale da poter distinguere piuttosto chiaramente le emozioni.
N.B: il classificatore verrà in seguito serializzato (come si mostrato in seguito nel codice dell'agente) in modo da velocizzare la risposta dell'agente nella recezione dell'emozione.

i. STUDIO SULLE FEATURE

Nel nostro sistema, per poter addestrare i classificatori, sono state utilizzate 4 librerie: *pyAudioAnalysis, Praat_feature, librosa, python_speech_features*.
Librosa estrae 3 features:

- Media MFCC: è la media di tutti i valori MFCC (Mel Frequency Cepstral Coefficients). L'insieme di questi coefficienti costituisce un MFC (mel-frequency cepstrum): una rappresentazione a breve termine dello spettro della potenza di un suono;
- Media ZCR: è la media di tutti i valori ZCR (Zero Crossing Rate). Lo ZCR è la frequenza del cambiamento del segno (+ o -) lungo il segnale audio;

- Media Rollof: è la media dei valori di Rollof. Questi indicano la porzione di spettro contenente una percentuale alfa (85% di default) dello spettro della potenza.

PyAudioAnalysis estrae l'entropia spettrale che definisce la densità dello spettro della potenza (Power Spectral Density (PSD) dei dati, compone l'arousal e la valence.

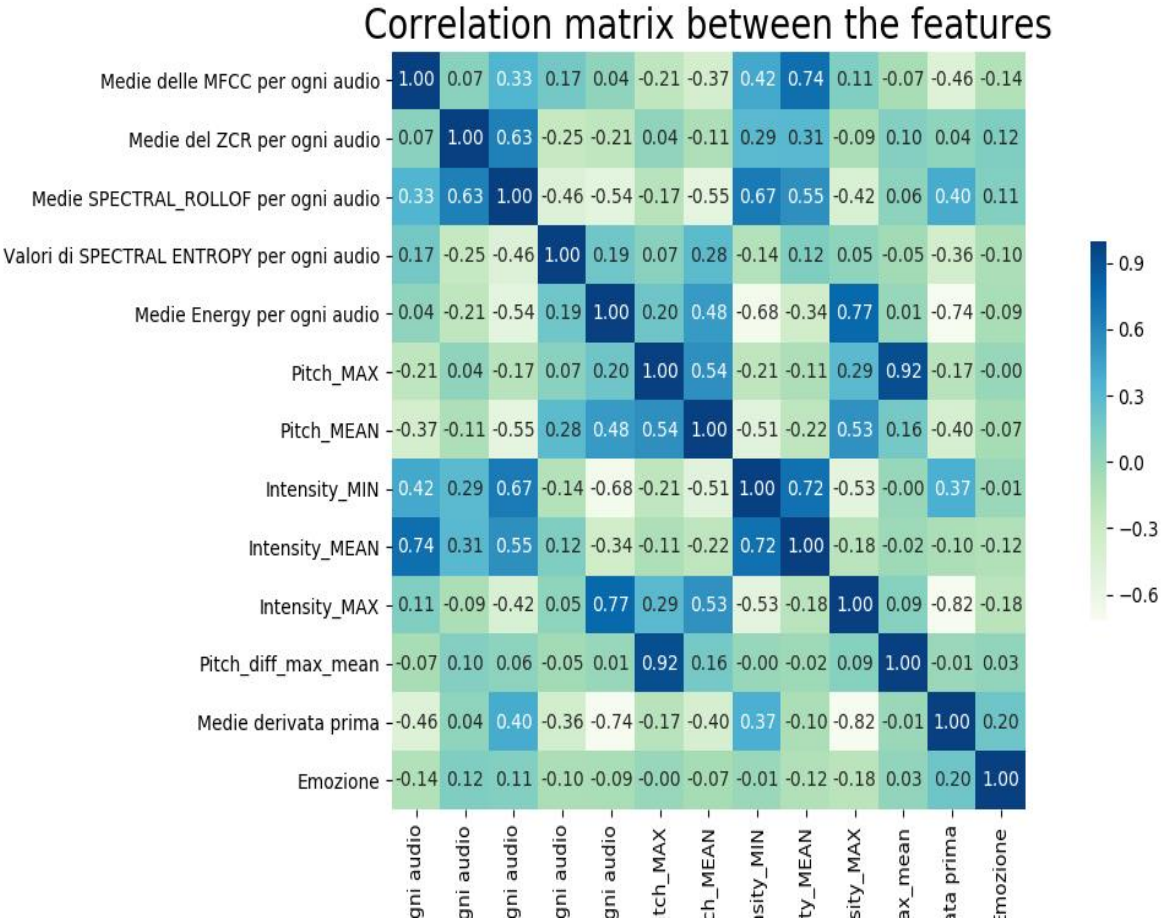
Python_speech_features viene utilizzata per calcolare la derivata sulla MFCC per capire le variazioni di questa in funzione del tempo.

Praat_feature estrae 41 feature "prosodiche" (correlate alle emozioni); abbiamo scelto:

- Energia: energia meccanica (o cinematica) che, partendo dalla sorgente, viene irradiata sotto forma di onde;
- Pitch: l'altezza relativa (o la bassezza relativa) di un tono percepito dall'orecchio umano che dipende dal numero di vibrazioni al secondo prodotte dalle corde vocali (in particolare pitch massimo, pitch_diff_max_mean e pitch_mean);
- Intensità: è la qualità acustica e psicoacustica associata alla forza di un suono, determinata dalla pressione esercitata dall'onda sonora (in particolare intensità minima, media e massima).

Di seguito verrà mostrata la matrice di correlazione calcolata con le emozioni precedentemente dichiarate, indicando, sull'ultima riga della matrice, la correlazione delle feature con le emozioni:

L'agente che abbiamo creato è *ESpeechPy*.



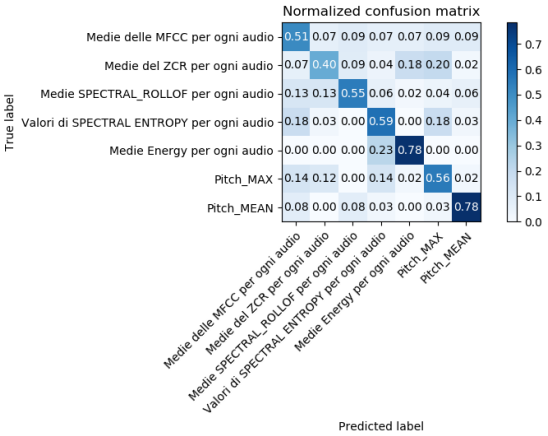
MATRICE DI CONFUSIONE

Testando diversi modelli (da noi serializzati) di classificatori, siamo riusciti a creare 2 modelli che risultano essere migliori:

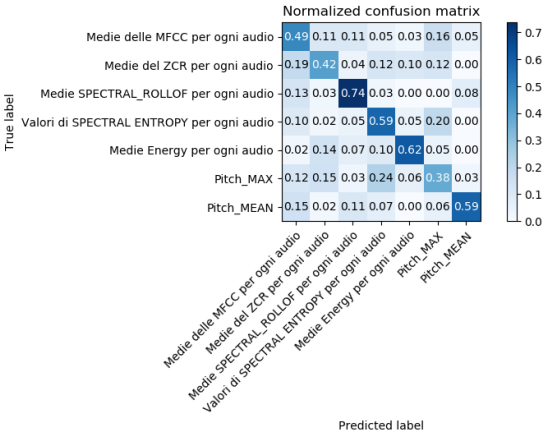
- *Modello3.pickle*
- *Modello5.pickle*

Questi sono stati addestrati esclusivamente su dataset “EMOVO” perché, integrando anche il nostro dataset, avremmo raggiunto una minore percentuale di accuratezza: nella fase di test, abbiamo rilevato un riscontro

negativo nelle predizioni eseguite dal sistema.



MODELLO3.PICKLE



MODELLO5.PICKLE (UTILIZZA GLI IPERPARAMETRI MODIFICATI)

```
Mean Absolute Error: 1.08843537414966
Mean Squared Error: 3.707482993197279
Root Mean Squared Error: 1.9254825351576885
```

MODELLO3.PICKLE

```
Mean Absolute Error: 1.5746887966804979
Mean Squared Error: 5.2012448132780085
Root Mean Squared Error: 2.2806237772324502
```

e. AGENTE

L'agente che abbiamo creato è *ESpeechPy*.

```
class DummyAgent(Agent):
    class MyBehav(CyclicBehaviour):
        async def on_start(self):
            print("Starting behaviour . . .")
            #self.counter = 0

        async def run(self):
            video("loading.mp4")
            video("start.mp4")
            while True:
                video("still.mp4")
                ##################
                microfono()
                ##################
                prediction = classifier()
                #print("ciao")
                # os.remove("audio_file.wav")
                print(prediction)
```

Le funzioni "on_start( )" e "run( )" permettono di avviare l'esecuzione dell'agente.

```
        async def on_end(self):
            print("Behaviour: {}.".format(self.exit_code))

    async def setup(self):
        print("Agent starting . . .")
        self.my_behav = self.MyBehav()
        self.add_behaviour(self.my_behav)

if __name__ == "__main__":
    dummy = DummyAgent("ourserver@404.city", "ourserver")
    dummy.start()

    dummy.stop()
```

La funzione "setup( )" permette di impostare il comportamento dell'agente.

f. MODULI UTILITY

In questa sezione, mostriamo il funzionamento (riportando il codice sorgente) delle seguenti funzioni:

- *microfono( )*
- *make_stereo( )*
- *video( )*

```
def microfono():

    r = sr.Recognizer()  # riconoscitore
    mic = sr.Microphone()  # classe dei microfoni
    lista_mic = mic.list_microphone_names()  # lista dei microfoni
    print(type(lista_mic[0]))
    #for i in range(0, len(lista_mic)):
        # if(lista_mic[i][0:len("Microfono")]=="Microfono" or lista_mic[i][0:len("Mic in at front ")]=="Mic in at front "):
        #print(lista_mic[i], i)
        # 5 o 9          11
    mic_scelto = sr.Microphone(device_index=6)  # il numero(6 per Claudio - 1 per Michele) è l'indice nella lista dei mic del device
    print('microfono mic_scelto')
    # prendere l'audio
    with mic as source:
        print('listen')
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)
        print('fine')

        # salvataggio audio
    with open("audio_file.wav", "wb") as file:
        file.write(audio.get_wav_data())
        file.close()
```

```
def make_stereo(file1, output):
    ifile = wave.open(file1)
    # (1, 2, 44100, 2013900, 'NONE', 'not compressed')
    (nchannels, sampwidth, framerate, nframes, comptype, compname) = ifile.getparams()
    assert comptype == 'NONE'  # Compressed not supported yet
    array_type = {1:'B', 2: 'h', 4: 'l'}[sampwidth]
    left_channel = array.array(array_type, ifile.readframes(nframes))[::nchannels]
    ifile.close()

    stereo = 2 * left_channel
    stereo[0::2] = stereo[1::2] = left_channel

    ofile = wave.open(output, 'w')
    ofile.setparams((2, sampwidth, framerate, nframes, comptype, compname))
    ofile.writeframes(stereo.tostring())
    ofile.close()
```

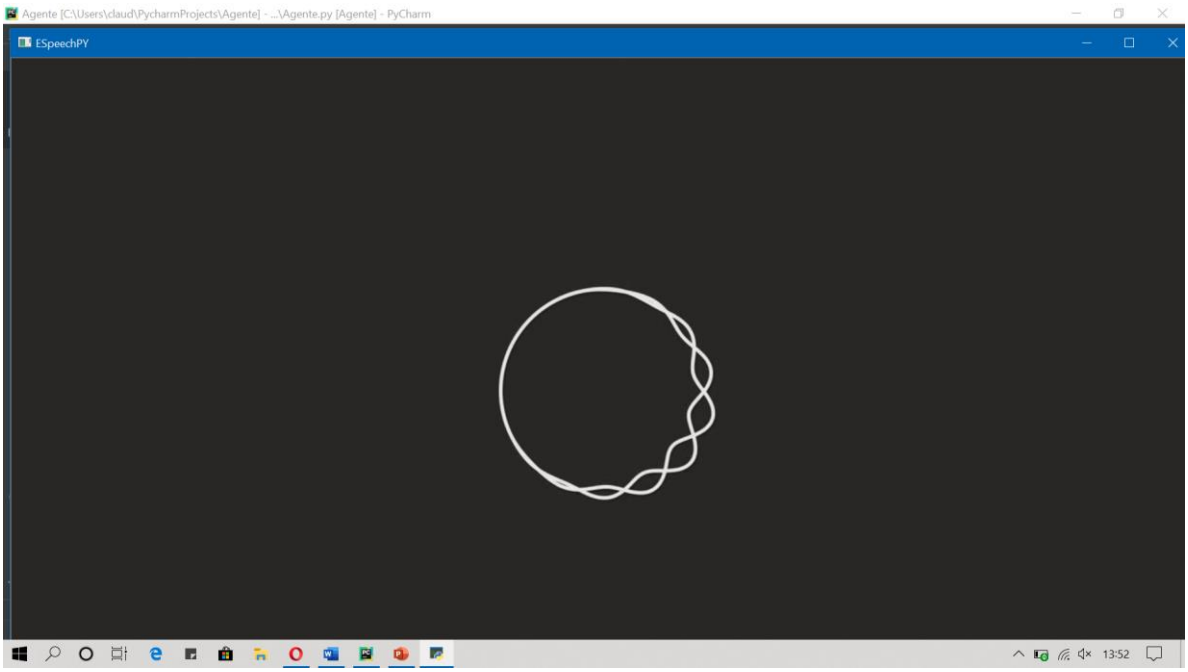
La funzione *"microfono( )"* consiste nel settare il sistema all'ascolto; un volta attivato il microfono, questo interromperà automaticamente l'acquisizione di audio quando non verranno più riprodotti suoni.

La funzione di supporto *"make_stereo( )"* viene utilizzata esclusivamente per trasformare l'audio acquisito da "mono" a "stereo".

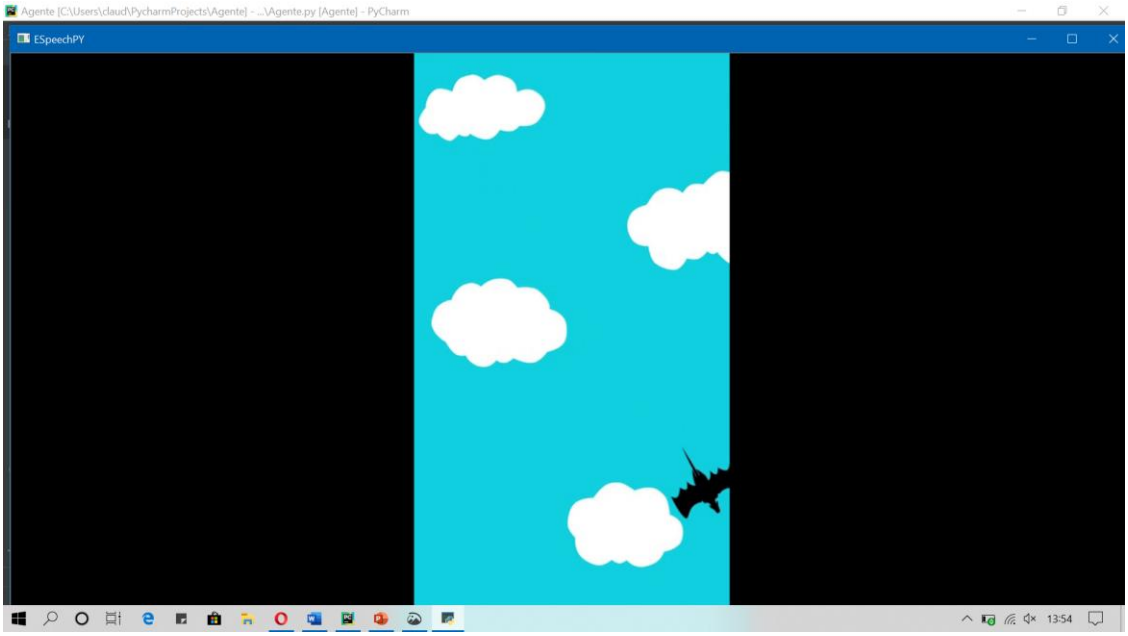
```
def video(video_name):
    # Create a VideoCapture object and read from input file
    cap = cv2.VideoCapture(video_name)
    # If the input is the camera, pass 0 instead of the video file name
    # Check if camera opened successfully
    if (cap.isOpened() == False):
        print("Error opening video stream or file")
    # Read until video is completed
    while (cap.isOpened()):
        # Capture frame-by-frame
        ret, frame = cap.read()
        if ret == True:
            # Display the resulting frame
            cv2.imshow('Agente', frame)
            # Press Q on keyboard to  exit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        # Break the loop
        else:
            break
    # When everything done, release the video capture object
    cap.release()
    # Closes all the frames
    # cv2.destroyAllWindows()
```

La funzione "*video( )*" consente la semplice apertura di file video.

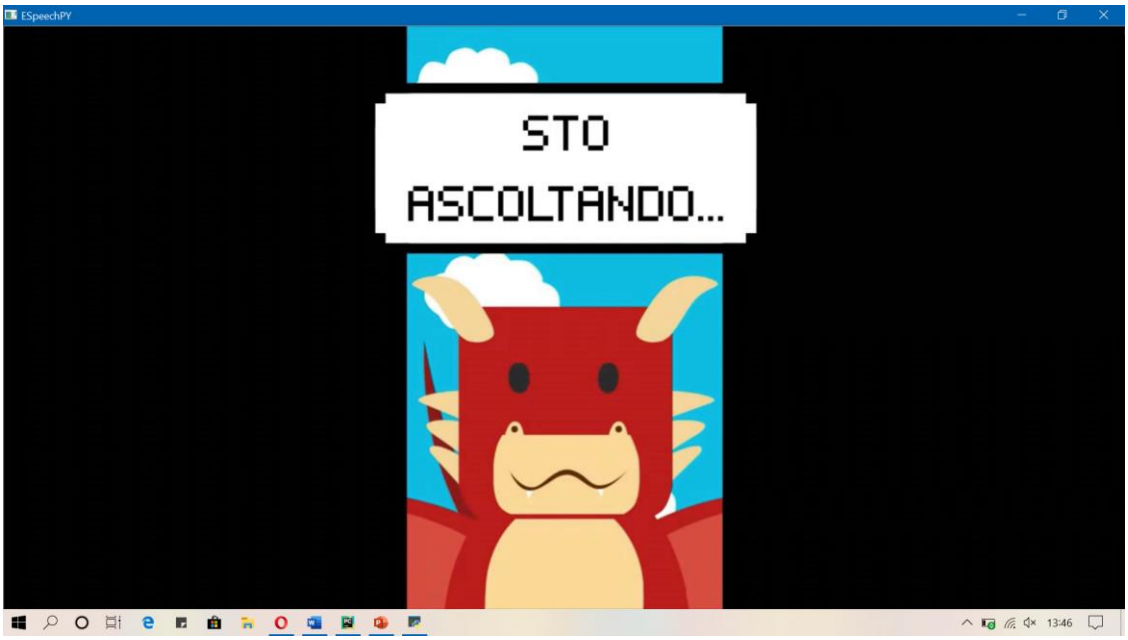
3. UNO SGUARDO AL SISTEMA



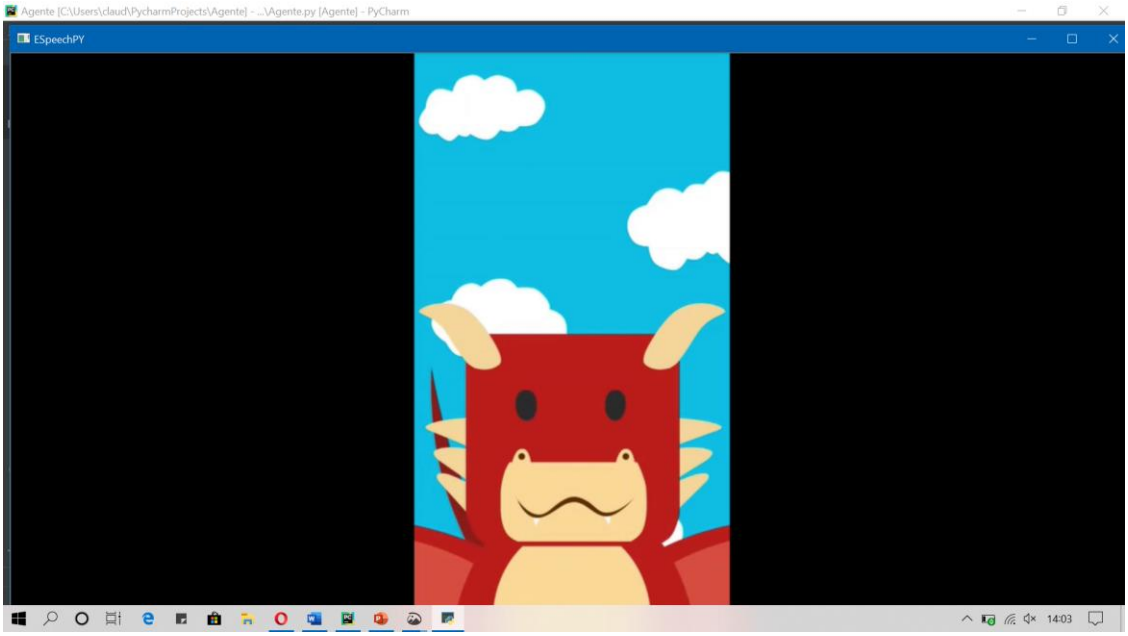
1) FASE DI LOADING



2) FASE DI AVVIO DI "*DRAGOBOT*"



3) SISTEMA IN ASCOLTO



4) FASE DI REAZIONE ALL'EMOZIONE