IOT PROJECT

GROUP MEMBER

Tahir Hussain
Mansoor Ashraf
Sania Kaleem

TITLE

“Service Dispatcher on Emergency call and Voice Analysis”

INTRODUCTION:

This project is design to handle emergencies, accidents and traumas which occur on daily besses
.
In our system there is no any system which handle multiple typs of accidents immeiatly because it handle manually . so that our reporting and implementation of that report is slow in our system
.
So that to overcome this problem we have design this project to handle the reporting of accident immediately.

PROJECT DESCRIPTION:

- Our project is reporting the accident by voice. and use the IVR an take user input explicitly.
- This system take users voice as input then it convert that into text by using NLP(natural language processing) to recognize the word.
- After reorganization of word, word will be match to the vocabulary an recognize the service which is user want ton.
- Also in our project there is age and gender recognition of the reporter will be predicted by its voice input.

Our system works in this way:

QUERY:
By user (in Speech) → Speech to Text → Machine Learning (NLP) with the help of spacy library "en_core_web_sm"(for Efficiency) and "en_core_web_trf" (For Accuracy) → Gets user intent, Now, once it understands the user intent, it's forward that message to emergency service and save all data they required.

ANSWER:
RESPONSE to User → your message forward to emergency services they will report you in few seconds.

WE DIVIDED OUR PROJECT INTO 3 MODULE WHICH ARE AS FOLLWING:

First Module:

Get user input into voice and convert into text then get the indent of user input and (optional save data as per client requirement) and forward their message to relevant Emergency service and tell the user your message will forward to XYZ emergency services.

SPEECH TO TEXT:

- We Used speech_recognition Library for speech to text Conversion.
- Speech recognition, also known as automatic speech recognition (ASR), computer speech recognition, or speech-to-text, is a capability which enables a program to process human speech into a written format.
- Speech recognition is **the process of converting spoken words to text**. Python supports many speech recognition engines and APIs, including Google Speech Engine, Google Cloud Speech API, Microsoft Bing Voice Recognition and IBM Speech to Text.
- A speech synthesizer is a computerized device that **accepts input, interprets data, and produces audible language**. It is capable of translating any text, predefined input, or controlled nonverbal body movement into audible speech.

Find Indent:

We use spacy library and used their trained pipeline which is "en_core_web_sm"(for Efficiency) and "en_core_web_trf" (For Accuracy), spacy support multi-languages.
These language are supported by spacy:

LANGUAGE	CODE	LANGUAGE DATA	PIPELINES
Catalan	ca	lang/ca	4 packages
Chinese	zh	lang/zh	4 packages
Danish	da	lang/da	4 packages
Dutch	nl	lang/nl	3 packages
English	en	lang/en	4 packages
French	fr	lang/fr	4 packages
German	de	lang/de	4 packages
Greek	el	lang/el	3 packages
Italian	it	lang/it	3 packages
Japanese	ja	lang/ja	4 packages
Lithuanian	lt	lang/lt	3 packages
Macedonian	mk	lang/mk	3 packages
Multi-language	xx	lang/xx	2 packages
Norwegian Bokmål	nb	lang/nb	3 packages
Polish	pl	lang/pl	3 packages
Portuguese	pt	lang/pt	3 packages

Romanian	ro	lang/ro	3 packages
Russian	ru	lang/ru	3 packages
Spanish	es	lang/es	4 packages
Afrikaans	af	lang/af	*none yet*
Albanian	sq	lang/sq	*none yet*
Arabic	ar	lang/ar	*none yet*
Armenian	hy	lang/hy	*none yet*
Basque	eu	lang/eu	*none yet*
Bengali	bn	lang/bn	*none yet*
Bulgarian	bg	lang/bg	*none yet*
Croatian	hr	lang/hr	*none yet*
Czech	cs	lang/cs	*none yet*
Estonian	et	lang/et	*none yet*
Finnish	fi	lang/fi	*none yet*
Gujarati	gu	lang/gu	*none yet*
Hebrew	he	lang/he	*none yet*
Hindi	hi	lang/hi	*none yet*
Hungarian	hu	lang/hu	*none yet*
Icelandic	is	lang/is	*none yet*
Indonesian	id	lang/id	*none yet*
Irish	ga	lang/ga	*none yet*
Kannada	kn	lang/kn	*none yet*
Korean	ko	lang/ko	*none yet*
Kyrgyz	ky	lang/ky	*none yet*
Latvian	lv	lang/lv	*none yet*
Ligurian	lij	lang/lij	*none yet*
Luxembourgish	lb	lang/lb	*none yet*
Malayalam	ml	lang/ml	*none yet*
Marathi	mr	lang/mr	*none yet*
Nepali	ne	lang/ne	*none yet*
Persian	fa	lang/fa	*none yet*
Sanskrit	sa	lang/sa	*none yet*
Serbian	sr	lang/sr	*none yet*
Setswana	tn	lang/tn	*none yet*
Sinhala	si	lang/si	*none yet*
Slovak	sk	lang/sk	*none yet*
Slovenian	sl	lang/sl	*none yet*
Swedish	sv	lang/sv	*none yet*
Tagalog	tl	lang/tl	*none yet*
Tamil	ta	lang/ta	*none yet*
Tatar	tt	lang/tt	*none yet*
Telugu	te	lang/te	*none yet*
Thai	th	lang/th	*none yet*
Turkish	tr	lang/tr	*none yet*
Ukrainian	uk	lang/uk	*none yet*
Urdu	ur	lang/ur	*none yet*
Vietnamese	vi	lang/vi	*none yet*
Yoruba	yo	lang/yo	*none yet*

en_core_web_sm:

English pipeline optimized for CPU. Components: tok2vec, tagger, parser, senter, ner, attribute_ruler, lemmatizer.

LANGUAGE	ENEnglish
TYPE	COREVocabulary, syntax, entities
GENRE	WEBwritten text (blogs, news, comments)
SIZE	SM13 MB
COMPONENTS	tok2vec, tagger, parser, senter, attribute_ruler, lemmatizer, ner
PIPELINE	tok2vec, tagger, parser, attribute_ruler, lemmatizer, ner
VECTORS	0 keys, 0 unique vectors (0 dimensions)
SOURCES	OntoNotes 5 (Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue , Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, Ann Houston) ClearNLP Constituent-to-Dependency Conversion (Emory University) WordNet 3.0 (Princeton University)
AUTHOR	Explosion
LICENSE	MIT

The statistical components included in this model package assign the following labels. The labels are specific to the corpus that the model was trained on. To see the description of a label, you can use spacy.explain.

TOK2VEC	
TAGGER	$, '', ,, -LRB-, -RRB-, ., :, ADD, AFX, CC, CD, DT, EX, FW, HYPH, IN, JJ, JJR, JJS, LS, MD, NFP, NN, NNP, NNPS, NNS, PDT, POS, PRP, PRP$, RB, RBR, RBS, RP, SYM, TO, UH, VB, VBD, VBG, VBN, VBP, VBZ, WDT, WP, WP$, WRB, XX, ``
PARSER	ROOT, acl, acomp, advcl, advmod, agent, amod, appos, attr, aux, auxpass, case, cc, ccomp, compound, conj, csubj, csubjpass, dative, dep, det, dobj, expl, intj, mark, meta, neg, nmod, npadvmod

SENTER	I, S
ATTRIBUTE_RULER	
LEMMATIZER	
NER	CARDINAL, DATE, EVENT, FAC, GPE, LANGUAGE, LAW, LOC, MONEY, NORP, ORDINAL, ORG, PERCENT, PERSON, PRODUCT, QUANTITY, TIME, WORK_OF_ART

en_core_web_trf:

English transformer pipeline (roberta-base). Components: transformer, tagger, parser, ner, attribute_ruler, lemmatizer.

LANGUAGE	**ENEnglish**	
TYPE	COREVocabulary, syntax, entities	
GENRE	WEBwritten text (blogs, news, comments)	
SIZE	TRF438 MB	
COMPONENTS	transformer, tagger, parser, attribute_ruler, lemmatizer, ner	
PIPELINE	transformer, tagger, parser, attribute_ruler, lemmatizer, ner	
VECTORS	0 keys, 0 unique vectors (0 dimensions)	
SOURCES	OntoNotes 5 (Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, Ann Houston) ClearNLP Constituent-to-Dependency Conversion (Emory University) WordNet 3.0 (Princeton University) roberta-base (Yinhan Liu and Myle Ott and Naman Goyal and Jingfei Du and Mandar Joshi and Danqi Chen and Omer Levy and Mike Lewis and Luke Zettlemoyer and Veselin Stoyanov)	
AUTHOR	Explosion	
LICENSE	MIT	

The statistical components included in this model package assign the following labels. The labels are specific to the corpus that the model was trained on. To see the description of a label, you can use

TRANSFORMER	
TAGGER	$, '', ,, -LRB-, -RRB- , ., :, ADD, AFX, CC, CD, DT, EX, FW, HYPH, IN, JJ, JJR, JJS, LS, MD, NFP, NN, NNP, NNPS, NNS, PDT, POS, PRP, PRP$, RB, RBR, RBS, RP, SYM, TO, UH, VB, VBD, VBG, VBN, VBP, VBZ, WDT, WP, WP$, WRB, XX, ``
PARSER	ROOT, acl, acomp, advcl, advmod, agent, amod, appos, attr, aux, auxpass, case, cc, ccomp, compound, conj, csubj, csubjpass, dative, dep, det, dobj, expl, intj, mark, meta, neg, nmod, npadvmod, nsubj, nsubjpass, nummod, oprd, parataxis, pcomp, pobj, poss, preconj, predet, prep, prt, punct, quantmod, relcl, xcomp
ATTRIBUTE_RULER	
LEMMATIZER	
NER	CARDINAL, DATE, EVENT, FAC, GPE, LANGUAGE, LAW, LOC, MONEY, NORP, ORDINAL, ORG, PERCENT, PERSON, PRODUCT, QUANTITY, TIME, WORK_OF_ART

Code:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 30 00:29:59 2021

@author: Mansoor
"""
import os
```

```
import sys
import pandas as pd
import spacy
import speech_recognition as sr
import pyttsx3
from datetime import date, datetime

global today
global current_time

today = date.today()
now = datetime.now()

print("Current Date =", today)
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)

jarvis = pyttsx3.init()
jarvis.setProperty('rate', 125)  # setting up new voice rate
voices = jarvis.getProperty('voices')
jarvis.setProperty('voice', voices[1].id)

sr.Microphone(device_index=1)
r3 = sr.Recognizer()

r3.energy_threshold = 8000
num = 0

with sr.Microphone() as source:

    jarvis.say("TELL YOUR Message")
    jarvis.runAndWait()
    jarvis.stop()

    print("TELL YOUR message \n")

    audio_name = r3.listen(source)

    try:
```

```
        text = r3.recognize_google(audio_name)

        print("NORMAL TEXT: \n",text)
        text = text.title()
        print("\n CAPTALIZED FIRST LETTER OF ALL STRINGS: \n",text,"\n")


        print("\n
========================================== \n")

        pd.set_option('display.max_colwidth', 200)
        nlp = spacy.load("en_core_web_sm")

        # text = "i am mansoor ashraf from karachi, pakistan i need an Ambulance its an Accident i am at Malir ."
        # text = "I live in Gulshan hadeed and my neighbour house was burning in fire and I need fire brigade urgently"
        text = "i am Mansoor Ashraf from karachi, pakistan i need Police, its an Accident i am at Malir"

        doc = nlp(text)
        print(text)
        print(" \n Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
        print("\n Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB \n"])

        # Find named entities, phrases and concepts
        for entity in doc.ents:
            print("\n",entity.text, entity.label_)

    for entity1 in doc.ents:
        if 'police' in entity1.text or 'Police' in entity1.text:
            print("\n ''This Message Forward to Police'' \n")

        elif 'ambulance' in entity1.text or 'Ambulance' in entity1.text:
            print("\n ''This Message Forward to Ambulance'' \n")

        elif 'fire brigade' in entity1.text or 'Fire Brigade' in entity1.text:
            print("\n ''This Message Forward to FIRE BRIGADE'' \n")
```

```
        # else:
        #     print("I can't forword this message please try-again")



except Exception as ex:

    print(str(ex))
    print("\n
========================================== \n")

    try:
        nlp = spacy.load("en_core_web_trf")
        doc = nlp(text)
        print(" \n Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
        print("\n Verbs:", [token.lemma_ for token in doc if token.pos_ ==
"VERB \n"])


        for entity in doc.ents:
            print("\n",entity.text, entity.label_)

        for entity1 in doc.ents:
            if 'police' in entity1.text or 'Police' in entity1.text:
                print("\n ''This Message Forward to Police'' \n")

        elif 'ambulance' in entity1.text or 'Ambulance' in entity1.text:
            print("\n ''This Message Forward to Ambulance'' \n")

        elif 'fire brigade' in entity1.text or 'Fire Brigade' in entity1.text:
            print("\n ''This Message Forward to FIRE BRIGADE'' \n")


        # else:
        #     print("I can't forword this message please try-again")
```

```
except Exception as ex:

    print(str(ex))
```

2 Module:

In second module we detect Gender from voice, so first we extract some feature from voice and we take the data set from Mozilla's Common Voice large dataset is used here, and some preprocessing has been performed

- Filtered out invalid samples.
- Filtered only the samples that are labeled in genre field.
- Balanced the dataset so that number of female samples are equal to male.
- Used Mel Spectrogram feature extraction technique to get a vector of a fixed length from each voice sample, the data folder contain only the features and not the actual mp3 samples (the dataset is too large, about 13GB).

If you wish to download the dataset and extract the features files (.npy files) on your own, preparation.py is the responsible script for that, once you unzip it,
put preparation.py in the root directory of the dataset and run it.
This will take sometime to extract features from the audio files and generate new .csv files.

Output:

```
Result: male
Probabilities:     Male: 96.36%    Female: 3.64%
```

CODE :

```
import pandas as pd
import spacy
import speech_recognition as sr
import pyttsx3
from datetime import date, datetime

global today
global current_time

today = date.today()
```

```
now = datetime.now()

print("Current Date =", today)
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)

jarvis = pyttsx3.init()
jarvis.setProperty('rate', 125)  # setting up new voice rate
voices = jarvis.getProperty('voices')
jarvis.setProperty('voice', voices[1].id)

sr.Microphone(device_index=1)
r3 = sr.Recognizer()

r3.energy_threshold = 8000
num = 0
```

```
with sr.Microphone() as source:

    jarvis.say("TELL YOUR Message")
    jarvis.runAndWait()
    jarvis.stop()

    print("TELL YOUR message \n")

    audio_name = r3.listen(source)

    try:
        text = r3.recognize_google(audio_name)

        print('\n')
        pd.set_option('display.max_colwidth', 200)

        nlp = spacy.load("en_core_web_sm")


        text = "i am taha jilani from karachi, pakistan i need an Ambulance its an
Accident i am at Malir ."
        # text = "I live in Gulshan hadeed and my neighbour house was burning in fire
and I need fire brigade urgently"
        # text = "i am Taha Jilani from karachi, pakistan i need Police, its an
Accident i am at Malir"

        doc = nlp(text)
        print(text)
        print(" \n Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
        print("\n Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB
\n"])

        # Find named entities, phrases and concepts
        for entity in doc.ents:
            print("\n",entity.text, entity.label_)

    except Exception as ex:
```

```
        print(str(ex))

    print("\n  ================================================== \n")

    try:
        nlp = spacy.load("en_core_web_trf")
        doc = nlp(text)

        for entity in doc.ents:
            print("\n",entity.text, entity.label_)

    except Exception as ex:

        print(str(ex))

import pyaudio
import wave
import librosa
import numpy as np
from sys import byteorder
from array import array
from struct import pack
from keras.models import load_model


THRESHOLD = 8000
CHUNK_SIZE = 1024
FORMAT = pyaudio.paInt16
RATE = 16000

SILENCE = 3



def is_silent(snd_data):
    "Returns 'True' if below the 'silent' threshold"
    return max(snd_data) < THRESHOLD

def normalize(snd_data):
    "Average the volume out"
    MAXIMUM = 16384
    times = float(MAXIMUM)/max(abs(i) for i in snd_data)

    r = array('h')
    for i in snd_data:
        r.append(int(i*times))
    return r

def trim(snd_data):
    "Trim the blank spots at the start and end"
    def _trim(snd_data):
        snd_started = False
        r = array('h')
```

```
    for i in snd_data:
        if not snd_started and abs(i)>THRESHOLD:
            snd_started = True
            r.append(i)

        elif snd_started:
            r.append(i)
    return r

# Trim to the left
snd_data = _trim(snd_data)

# Trim to the right
snd_data.reverse()
snd_data = _trim(snd_data)
snd_data.reverse()
return snd_data

def add_silence(snd_data, seconds):
    "Add silence to the start and end of 'snd_data' of length 'seconds' (float)"
    r = array('h', [0 for i in range(int(seconds*RATE))])
    r.extend(snd_data)
    r.extend([0 for i in range(int(seconds*RATE))])
    return r

def record():
    """
    Record a word or words from the microphone and
    return the data as an array of signed shorts.
    Normalizes the audio, trims silence from the
    start and end, and pads with 0.5 seconds of
    blank sound to make sure VLC et al can play
    it without getting chopped off.
    """
    p = pyaudio.PyAudio()
    stream = p.open(format=FORMAT, channels=1, rate=RATE,
        input=True, output=True,
        frames_per_buffer=CHUNK_SIZE)

    num_silent = 0
    snd_started = False

    r = array('h')

    while 1:
        # little endian, signed short
        snd_data = array('h', stream.read(CHUNK_SIZE))

        if byteorder == 'big':
            snd_data.byteswap()
        r.extend(snd_data)

        silent = is_silent(snd_data)

        if silent and snd_started:
```

```
            num_silent += 1
        elif not silent and not snd_started:
            snd_started = True

        if snd_started and num_silent > SILENCE:
            break

    sample_width = p.get_sample_size(FORMAT)
    stream.stop_stream()
    stream.close()
    p.terminate()

    r = normalize(r)
    r = trim(r)
    r = add_silence(r, 0.5)
    return sample_width, r
```

```
def record_to_file(path):
    "Records from the microphone and outputs the resulting data to 'path'"

    sample_width, data = record()
    data = pack('<' + ('h'*len(data)), *data)
    # print(*data,"\n",data)
    wf = wave.open(path, 'wb')
    wf.setnchannels(1)
    wf.setsampwidth(sample_width)
    wf.setframerate(RATE)
    wf.writeframes(data)
    wf.close()
```

```
def extract_feature(file_name, **kwargs):
    """
    Extract feature from audio file `file_name`
        Features supported:
            - MFCC (mfcc)
            - Chroma (chroma)
            - MEL Spectrogram Frequency (mel)
            - Contrast (contrast)
            - Tonnetz (tonnetz)
    e.g:
    `features = extract_feature(path, mel=True, mfcc=True)`
"""
mfcc = kwargs.get("mfcc")
chroma = kwargs.get("chroma")
mel = kwargs.get("mel")
contrast = kwargs.get("contrast")
tonnetz = kwargs.get("tonnetz")
X, sample_rate = librosa.core.load(file)
if chroma or contrast:
    stft = np.abs(librosa.stft(X))
result = np.array([])
if mfcc:
```

```
        mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,
axis=0)
        result = np.hstack((result, mfccs))
    if chroma:
        chroma = np.mean(librosa.feature.chroma_stft(S=stft,
sr=sample_rate).T,axis=0)
        result = np.hstack((result, chroma))
    if mel:
        mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
        result = np.hstack((result, mel))
    if contrast:
        contrast = np.mean(librosa.feature.spectral_contrast(S=stft,
sr=sample_rate).T,axis=0)
        result = np.hstack((result, contrast))
    if tonnetz:
        tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),
sr=sample_rate).T,axis=0)
        result = np.hstack((result, tonnetz))
    return result


if __name__ == "__main__":

    # load the saved model (after training)
    model=load_model("C:/Users/Mansoor/Desktop/gender-recognition-by-voice-
master/results/model.h5")
    model.load_weights("C:/Users/Mansoor/Desktop/gender-recognition-by-voice-
master/results/model.h5")

    print("Please talk")
    file = "C:/Users/Mansoor/Desktop/gender-recognition-by-voice-
master/results/test.wav"

    # record the file (start talking)
    record_to_file(file)

    # extract features and reshape it
    features = extract_feature(file, mel=True).reshape(1, -1)

    # predict the gender!
    male_prob = model.predict(features)[0][0]
    female_prob = 1 - male_prob
    gender = "male" if male_prob > female_prob else "female"

    # show the result!
    print("\n Result:", gender,"\n")
    print(f"Probabilities:     Male: {male_prob*100:.2f}%    Female:
{female_prob*100:.2f}%")
```

Third module:

In Third Module we detect Age from voice input, we extract some feature with pyaudio and librosa etc. and we use similar dataset of Mozilla's Common Voice but do some enhancement and we make age groups like thirty, twenty, forty etc. and from input we extract feature again and we can detect age group from voice

Code:

```
# -*- coding: utf-8 -*-


import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # visualizing data
import seaborn as sns # visualizing data with stunning default theme
import sklearn # contain algorithms
import warnings
warnings.filterwarnings('ignore')

# load dataset from input directory
df = pd.read_csv("C:/Users/Mansoor/Desktop/AGE DATASET/cv-valid-train.csv")
df[df['age'].notna()].head()


sns.set(rc={'figure.figsize':(15, 5)})
sns.countplot(x="age",
        data=df[df['age'].notna()],
        order=['teens', 'twenties', 'thirties', 'fourties', 'fifties', 'sixties', 'seventies', 'eighties'])

plt.show()

#
==========================================================================
=========================================
#
==========================================================================
=========================================

sns.countplot(x="age",
        hue='gender',
        data=df[df['age'].notna()],
        order=['teens', 'twenties', 'thirties', 'fourties', 'fifties', 'sixties', 'seventies', 'eighties'])

plt.show()

#
==========================================================================
=========================================
#
==========================================================================
=========================================

sns.displot(x="accent",
        data=df[df['accent'].notna()],
        hue='gender',
```

```
        multiple='stack',
        height=5, aspect=18/5)

plt.show()

#
==========================================================================================
=========================================
#
==========================================================================================
=========================================


#del df['duration']
start=df.shape
#df.isna().sum()
end = df[df['age'].notna()& df['gender'].notna() & df['accent'].notna()].shape
print("initial: {} final: {}".format(start, end))
sns.countplot(x="age",
        hue='gender',
        data=df[df['age'].notna()& df['gender'].notna() & df['accent'].notna()],
        order=['teens', 'twenties', 'thirties', 'fourties', 'fifties', 'sixties', 'seventies', 'eighties'])

plt.show()

#
==========================================================================================
=========================================
#
==========================================================================================
=========================================


"""#### **First-step findings**
We understood the dataset and also found that the dataset is imbalance. To refine the dataset for ml-model,
we need to perform pre-processing in next-step.

### Data Pre-processing
#### Data Cleaning
In this step we drop entries(samples) with NaN values. The columns that doesn't contribute(unnecessary)
to the model are removed. The attributes are checked for its datatypes and changed to an approapriate
type.
"""



#we extract the columns that we think useful are
df = df[['filename','age','gender']]
#To clean the data we remove the sample with NaN attribute values.
data = df[df['age'].notna() & df['gender'].notna()]
data.reset_index(inplace=True, drop=True)
data.head()

#
==========================================================================================
=========================================
```

```
#
==========================================================================
========================================


#data['gender'] = pd.to_numeric(data['gender'],errors = 'coerce')
#data.dtypes
#if the below code fail to covert gender datatype and values then uncomment above code
cleanup_nums = {"gender": {"male":1,"female":0,"other":0.5}}
data = data.replace(cleanup_nums)
data.head()

#
==========================================================================
========================================
#
==========================================================================
========================================


"""### **Feature Engineering**
#### **Feature Extraction**

We extract the following features:
The following features are related to audio quality through which the model will learn more effectively. In this project it is not necessary to have good knowledge about the given audio features.
**Gender**
* **Spectral Centroid**: each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame
* **Spectral Bandwidth**: compute 2nd-order spectral bandwidth
* **Spectral Rolloff**: the center frequency for a spectrogram bin such that at least roll_percent (0.85 by default) of the energy of the spectrum in this frame is contained in this bin and the bins below
* **Mel Frequency Cepstral Coefficients (MFCCs)**: a small set of 20 features that describe the overall shape of a spectral envelope

**Librosa package**
Librosa is a Python package for music and audio analysis. It provides the building blocks necessary to create the music information retrieval systems. Librosa helps to visualize the audio signals and also do the feature extractions in it using different signal processing techniques.
"""

import librosa
import audioread
ds_path = "E:/"

#this function is used to extract audio frequency features
def feature_extraction(filename, sampling_rate=48000):
    path = "{}{}".format(ds_path, filename)
    features = list()
    audio, _ = librosa.load(path, sr=sampling_rate)

    gender = data[data['filename'] == filename].gender.values[0]
    spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=audio, sr=sampling_rate))
    spectral_bandwidth = np.mean(librosa.feature.spectral_bandwidth(y=audio, sr=sampling_rate))
    spectral_rolloff = np.mean(librosa.feature.spectral_rolloff(y=audio, sr=sampling_rate))
    features.append(gender)
```

```
    features.append(spectral_centroid)
    features.append(spectral_bandwidth)
    features.append(spectral_rolloff)

    mfcc = librosa.feature.mfcc(y=audio, sr=sampling_rate)
    for el in mfcc:
        features.append(np.mean(el))

    return features


features = feature_extraction(data.iloc[0]['filename'])
print("features: ", features)

#the function create dataframe to store the feature and label related to each other
def create_df_features(orig):
    new_rows = list()
    tot_rows = len(orig)-1
    stop_counter = 55001

    for idx, row in orig.iterrows():
        if idx >= stop_counter: break
        print("\r", end="")
        print("{}/{}".format(idx, tot_rows), end="", flush=True)
        features = feature_extraction(row['filename'])
        features.append(row['age'])
        new_rows.append(features)

    return pd.DataFrame(new_rows, columns=["gender", "spectral_centroid", "spectral_bandwidth",
"spectral_rolloff",
                                           "mfcc1", "mfcc2", "mfcc3", "mfcc4", "mfcc5", "mfcc6", "mfcc7", "mfcc8",
                                           "mfcc9", "mfcc10", "mfcc11", "mfcc12", "mfcc13", "mfcc14", "mfcc15", "mfcc16",
                                           "mfcc17", "mfcc18", "mfcc19", "mfcc20", "label"])

df_features = create_df_features(data)
df_features.head()

"""### **Feature Transformation**

Scaling the features with the scikit-learn StandardScaler.
"""

from sklearn.preprocessing import StandardScaler

def scale_features(data):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(np.array(data.iloc[:, 0:-1], dtype = float))
    # with data.iloc[:, 0:-1] we don't consider the label column

    return scaled_data, scaler

x, scaler = scale_features(df_features)

print("Before scaling:", df_features.iloc[0].values[:-1])
print("\nAfter scaling:", x[0])
```

```
df_features['label'].unique()

from sklearn.preprocessing import LabelEncoder

def get_labels(data):
    labels = data.iloc[:, -1]
    encoder = LabelEncoder()
    labels = encoder.fit_transform(labels)
    return labels, encoder

y, encoder = get_labels(df_features)
classes = encoder.classes_
print("Before encoding:", df_features.iloc[0].values[-1])
print("\nAfter encoding:", y[0])
print("\nClasses:", classes)

"""### **Feature Selection**

We use the ANOVA (ANalysis Of VAriance) statistical technique (f_classif) to select the best 22 features.

"""

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

n_features = 22

f_selector = SelectKBest(f_classif, k=n_features).fit(x, y)
X_new = f_selector.transform(x)
scores = f_selector.scores_

indices = np.argsort(scores)[::-1]

features = []
for i in range(n_features):
    features.append(df_features.columns[indices[i]])

plt.figure(figsize=(22, 5))
plt.bar(features, scores[indices[range(n_features)]], color='g')
plt.xticks(fontsize=8)
plt.show()

"""### **Model Selection**

Here, we consider two classifiers:

    Support Vector Machine
    Random Forest

We evaluate them with the K-Fold Cross-Validation technique. At each iteration of this outer cross-validation process, we tune the hyper-parameters of the classifiers with another (inner) Cross-Validation process , that will further divide the training set into training and validation data.

At each iteration of the outer CV process, we print the F1-Score obtained by the tuned classifier on the validation data, but also the F1-Score computed on the test data. Finally, we print the average F1-Scores computed on the test data at each step of the outer CV process.
```

```
"""

from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

classifiers_and_params = [
    (SVC(), {'C': [200, 150, 100], 'gamma': ['auto', 'scale']}),
    (RandomForestClassifier(), {'n_estimators': [100, 150, 200]})
]

for tup in classifiers_and_params:
    print("{}".format(tup[0].__class__.__name__))

    # the main CV process
    outer_cv = KFold(n_splits=3, shuffle=True, random_state=0)
    fold_counter = 0

    results = list()
    for train_idx, test_idx in outer_cv.split(X_new):
        fold_counter += 1

        # split data in training and test sets
        X_train, X_test = X_new[train_idx], X_new[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]

        # the CV process used for the Grid Search
        inner_cv = KFold(n_splits=2, shuffle=True, random_state=0)

        # define and run the Grid Search CV process
        gs = GridSearchCV(tup[0], tup[1], scoring='f1_macro', cv=inner_cv, refit=True)
        res = gs.fit(X_train, y_train)

        # get the best model, re-fit on the whole training set
        best_model = res.best_estimator_

        # evaluation on the test set
        pred = best_model.predict(X_test)
        score = f1_score(y_test, pred, average='macro')
        results.append(score)

        print("\tFold {}, Best Params {} with F1 Score {:.3f}, F1 Score on Test data {:.3f}"
              .format(fold_counter, res.best_params_, res.best_score_, score))

    print('\tAverage F1 Score on Test Set: {:.3f}\n'.format(np.mean(results)))

import itertools
import matplotlib.pyplot as plt

def my_plot_confusion_matrix(cm, classes, normalize=False):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        title = "Normalized Confusion Matrix"
    else:
```

```
    title = "Confusion Matrix (without normalization)"

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.title(title)

thresh = cm.max() / 2.
fmt = "{:0.2f}" if normalize else "{:d}"
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, fmt.format(cm[i, j]),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2, random_state=0)

model = SVC(C=100, gamma='scale')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

plt.figure()
my_plot_confusion_matrix(cm, classes=classes)

plt.grid(False)
plt.show()

# Plot normalized confusion matrix
plt.figure()
my_plot_confusion_matrix(cm, classes=classes, normalize=True)

plt.grid(False)
plt.show()

import joblib
# save the model to disk
filename = 'finalized_model.sav'
joblib.dump(model, filename)
```