```
import pandas as pd
import numpy as np
import os
import tqdm
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard, EarlyStopping
from sklearn.model_selection import train_test_split


df = pd.read_csv("/content/balanced-all.csv")
df.head()


df.tail()


# get total samples
n_samples = len(df)
# get total male samples
n_male_samples = len(df[df['gender'] == 'male'])
# get total female samples
n_female_samples = len(df[df['gender'] == 'female'])
print("Total samples:", n_samples)
print("Total male samples:", n_male_samples)
print("Total female samples:", n_female_samples)


label2int = {
    "male": 1,
    "female": 0
}




def load_data(vector_length=128):
    """A function to load gender recognition dataset from `data` folder
    After the second run, this will load from results/features.npy and results/labels.npy files
    as it is much faster!"""
    # make sure results folder exists
    if not os.path.isdir("results"):
        os.mkdir("results")
    # if features & labels already loaded individually and bundled, load them from there instead
    if os.path.isfile("/content/features.npy") and os.path.isfile("/content/labels.npy"):
        X = np.load("/content/features.npy")
        y = np.load("/content/labels.npy")
        return X, y
    # read dataframe
    df = pd.read_csv("/content/balanced-all.csv")
    # get total samples
    n_samples = len(df)
    # get total male samples
    n_male_samples = len(df[df['gender'] == 'male'])
    # get total female samples
    n_female_samples = len(df[df['gender'] == 'female'])
    print("Total samples:", n_samples)
    print("Total male samples:", n_male_samples)
    print("Total female samples:", n_female_samples)
    # initialize an empty array for all audio features
```

```
    X = np.zeros((n_samples, vector_length))
    # initialize an empty array for all audio labels (1 for male and 0 for female)
    y = np.zeros((n_samples, 1))
    for i, (filename, gender) in tqdm.tqdm(enumerate(zip(df['filename'], df['gender'])), "Loading data",
        features = np.load(filename)
        if len(features) == vector_length:
           X[i] = features
           y[i] = label2int[gender]
        elif len(features) < vector_length:
          padding = np.zeros(vector_length - len(features))
          X[i] = np.concatenate([features, padding])
          y[i] = label2int[gender]
        else:
          print(f"Truncating file {filename} with {len(features)} features.")
          X[i] = features[vector_length]
          y[i] = label2int[gender]
    # save the audio features and labels into files
    # so we won't load each one of them next run
    np.save("results/features", X)
    np.save("results/labels", y)
    return X, y


def split_data(X, y, test_size=0.1, valid_size=0.1):
    # split training set and testing set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=7)
    # split training set and validation set
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=valid_size, random_
    # return a dictionary of values
    return {
        "X_train": X_train,
        "X_valid": X_valid,
        "X_test": X_test,
        "y_train": y_train,
        "y_valid": y_valid,
        "y_test": y_test
    }


import numpy as np

def load_data():
    # load data from file or database
    X = np.random.rand(1000, 128)  # example input data
    y = np.random.randint(0, 2, size=1000)  # example target data
    return X, y
X, y = load_data()
X = X.reshape((1000 , 128))



data = split_data(X, y, test_size=0.1, valid_size=0.1)
```

Building the *model*

```
def create_model(vector_length=128):
    """5 hidden dense layers from 256 units to 64, not the best model."""
    model = Sequential()
```

```
    model.add(Dense(256, input_shape=(vector_length,)))
    model.add(Dropout(0.3))
    model.add(Dense(256, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(128, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(128, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(64, activation="relu"))
    model.add(Dropout(0.3))
    # one output neuron with sigmoid activation function, 0 means female, 1 means male
    model.add(Dense(1, activation="sigmoid"))
    # using binary crossentropy as it's male/female classification (binary)
    model.compile(loss="binary_crossentropy", metrics=["accuracy"], optimizer="adam")
    # print summary of the model
    model.summary()
    return model

# construct the model
model = create_model()
```

Training the model

```
# use tensorboard to view metrics
tensorboard = TensorBoard(log_dir="logs")
# define early stopping to stop training after 5 epochs of not improving
early_stopping = EarlyStopping(mode="min", patience=5, restore_best_weights=True)

batch_size = 64
epochs = 100
# train the model using the training set and validating using validation set
model.fit(data["X_train"], data["y_train"], epochs=epochs, batch_size=batch_size, validation_data=(data['
          callbacks=[tensorboard, early_stopping])
```

```
# save the model to a file
model.save("results/model.h5")
```

Testing the model

```
# evaluating the model using the testing set
print(f"Evaluating the model using {len(data['X_test'])} samples...")
loss, accuracy = model.evaluate(data["X_test"], data["y_test"], verbose=0)
print(f"Loss: {loss:.4f}")
print(f"Accuracy: {accuracy*100:.2f}%")
```

Testing the model with voice

```
pip install pyproject.toml
```

```
!apt-get install -y python-pyaudio python3-pyaudio
```

```
!apt install libasound2-dev portaudio19-dev libportaudio2 libportaudiocpp0 ffmpeg
```

```
pip install pyaudio
```

```
import pyaudio
import os
import wave
import librosa
import numpy as np
from sys import byteorder
from array import array
from struct import pack
```

```
THRESHOLD = 500
CHUNK_SIZE = 1024
FORMAT = pyaudio.paInt16
RATE = 16000

SILENCE = 30

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)

    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_name)
    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=(
        result = np.hstack((result, tonnetz))
    return result
```

```
pip install utils
```

```
pip install preprocessing
```

```
import librosa

def extract_feature(file_path, mfcc=True, chroma=True, mel=True):
    with open(file_path, 'rb') as f:
        x, sr = librosa.load(f, sr=None)

    if chroma:
        stft = np.abs(librosa.stft(x))
        result = np.array([])
        if mfcc:
            mfccs = np.mean(librosa.feature.mfcc(y=x, sr=sr, n_mfcc=40).T, axis=0)
            result = np.hstack((result, mfccs))
        if chroma:
            chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sr).T, axis=0)
            result = np.hstack((result, chroma))
```

```
        if mel:
            mel = np.mean(librosa.feature.melspectrogram(x, sr=sr).T, axis=0)
            result = np.hstack((result, mel))
    return result



import os
import sys

# Assuming that the directory containing the `preprocessing` package is `/path/to/your/package`
package_dir = '/content/results/model.h5'
sys.path.append(package_dir)

# Now you can import the `extract_feature` function from the `preprocessing` package
# from preprocessing import extract_feature



import pickle


pip install utils


!pip install --upgrade utils


pip install module_name


pip show module_name


import argparse
import os
import soundfile as sf
from utils import load_data, split_data, create_model
from preprocessing import extract_feature

def record_to_file(file):
    # Code for recording audio and saving it to the specified file
    # Implementation depends on your specific requirements

# if __name__ == "__main__":
    # load the saved model (after training)
    # model = pickle.load(open("result/mlp_classifier.model", "rb"))
    parser = argparse.ArgumentParser(description="""Gender recognition script, this will load the model y
                                    and perform inference on a sample you provide (either using your voic
    parser.add_argument("-f", "--file", help="The path to the file, preferred to be in WAV format")
    args = parser.parse_args()
    file = args.file
    # construct the model
    model = create_model()
    # load the saved/trained weights
    model.load_weights("results/model.h5")
    if not file or not os.path.isfile(file):
        # if file not provided, or it doesn't exist, use your voice
        print("Please talk")
        # put the file name here
        file = "test.wav"
        # record the file (start talking)
```

```
        record_to_file(file)
    else:
        # Validate the file format
        file_extension = os.path.splitext(file)[1].lower()
        if file_extension != ".wav":
            raise ValueError("Invalid file format. Please provide a WAV file.")

    # Load the audio file
    audio, sample_rate = sf.read(file)

    # Extract features and reshape it
    features = extract_feature(audio, sample_rate, 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("Result:", gender)
    print(f"Probabilities:     Male: {male_prob * 100:.2f}%    Female: {female_prob * 100:.2f}%")



import glob
import os
import pandas as pd
import numpy as np
import shutil
import librosa
from tqdm import tqdm


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_name)
    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=(
        result = np.hstack((result, tonnetz))
    return result

dirname = "data"

if not os.path.isdir(dirname):
    os.mkdir(dirname)


csv_files = glob.glob("*.csv")

for j, csv_file in enumerate(csv_files):
    print("[+] Preprocessing", csv_file)
    df = pd.read_csv(csv_file)
    # only take filename and gender columns
    new_df = df[["filename", "gender"]]
    print("Previously:", len(new_df), "rows")
    # take only male & female genders (i.e droping NaNs & 'other' gender)
    new_df = new_df[np.logical_or(new_df['gender'] == 'female', new_df['gender'] == 'male')]
    print("Now:", len(new_df), "rows")
    new_csv_file = os.path.join(dirname, csv_file)
    # save new preprocessed CSV
    new_df.to_csv(new_csv_file, index=False)
    # get the folder name
    folder_name, _ = csv_file.split(".")
    audio_files = glob.glob(f"{folder_name}/{folder_name}/*")
    all_audio_filenames = set(new_df["filename"])
    for i, audio_file in tqdm(list(enumerate(audio_files)), f"Extracting features of {folder_name}"):
        splited = os.path.split(audio_file)
        # audio_filename = os.path.join(os.path.split(splited[0])[-1], splited[-1])
        audio_filename = f"{os.path.split(splited[0])[-1]}/{splited[-1]}"
        # print("audio_filename:", audio_filename)
        if audio_filename in all_audio_filenames:
```