Speech Emotion Recognition Using

Get started Open in app

Dataset

Muriel Kosaka Oct 27, 2020 · 7 min read ★

Image by Tengyart on Unsplash

Through all the available senses, humans can sense the emotional state of their communication partner.

computers; although they can easily understand content based information, accessing the depth behind content is difficult and that's what speech emotion recognition (SER) sets out to do. It is a system through which various audio speech files are classified into different emotions such as happy, sad, anger and neutral by computers. Speech emotion recognition can be used in areas such as the medical field or customer call centers. My goal here is to demonstrate SER using the RAVDESS Audio Dataset provided on Kaggle.

Get started Open in app

```
%matplotlib inline
import matplotlib.pyplot as
plt
import librosa.display
from IPython.display import
Audio
import numpy as np
import tensorflow as tf
from matplotlib.pyplot import
specgram
import pandas as pd
from sklearn.metrics import
confusion_matrix
import IPython.display as ipd
# To play sound in the
notebook
import os # interface with
underlying OS that python is
running on
import sys
from sklearn.model_selection
import StratifiedShuffleSplit
from sklearn.preprocessing
import LabelEncoder
```

```
from keras.layers import
Conv1D, MaxPooling1D,
AveragePooling1D
from keras.layers import
Input, Flatten, Dropout,
Activation,
BatchNormalization, Dense
from sklearn.model_selection
import GridSearchCV
from
keras.wrappers.scikit_learn
import KerasClassifier
from keras.optimizers import
SGD
from keras.regularizers
import l2
import seaborn as sns
from keras.callbacks import
EarlyStopping,
ModelCheckpoint
from keras.utils import
to_categorical
from sklearn.metrics import
classification_report
```

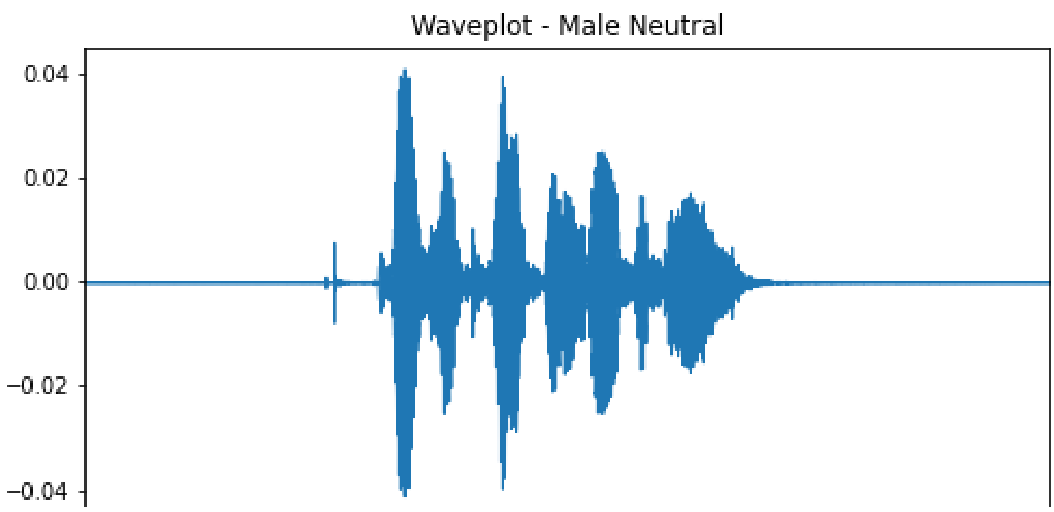
First, let's load in and play a sample audio file from the dataset using IPython.Display and Python's librosa library:

```
# LOAD IN FILE
x, sr = 
librosa.load('/Users/murielkosaka/Desktop/capstone_project/audio/audio_speech_actors_01-24/Actor_01/03-01-01-01-01-01-01.wav')

# PLAY AUDIO FILE
librosa.output.write_wav('ipd.Audio Files/MaleNeutral.wav', x, sr)
Audio(data=x, rate=sr)
```

Next let's look at a wave plot of this audio file using *librosa.display.waveplot*:

```
# DISPLAY WAVEPLOT
plt.figure(figsize=(8, 4))
librosa.display.waveplot(x, sr=sr)
plt.title('Waveplot - Male Neutral')
plt.savefig('Waveplot_MaleNeutral.png')
```



Wave plots plot a signals amplitude envelope over time, seeing the overall shape of an emotion can help determine which feature extraction method (MFCC, STFT, Log-Mel Spectograms, Zero-Crossing Rate, Spectral Centroid, etc.) to use for modeling. Feature extraction is important in modeling because it converts audio files into a format that can be understood by models.

After examining waveplots for a sample of each emotion, I decided to use Log-Mel Spectrograms as the method of feature extraction. We can display the *librosa.display.specshow:*

```
# CREATE LOG MEL SPECTROGRAM
spectrogram =
librosa.feature.melspectrogra
m(y=x, sr=sr,
n_mels=128,fmax=8000)
spectrogram =
librosa.power_to_db(spectrogr
am)

librosa.display.specshow(spec
trogram, y_axis='mel',
fmax=8000, x_axis='time');
plt.title('Mel Spectrogram -
Male Neutral')
plt.savefig('MelSpec_MaleNeut
ral.png')
plt.colorbar(format='%+2.0f
dB');
```

Image by Author

Prepping the Data

Before modeling, I structured the data into a Pandas DataFrame by creating a directory of the audio files, then creating a function to extraction the emotion label and gender label for each file (although I was only interested in classifying emotion, I also extracted the gender label in case I should decide to the associated filepaths into a DataFrame *audio_df*.

```
# CREATE FUNCTION TO EXTRACT
EMOTION NUMBER, ACTOR AND
GENDER LABEL
emotion = []
gender = []
actor = []
file_path = []
for i in actor_folders:
    filename =
os.listdir(audio + i)
#iterate over Actor folders
    for f in filename: # go
through files in Actor folder
        part = f.split('.')
[0].split('-')

emotion.append(int(part[2]))

actor.append(int(part[6]))
```

```
        else:
            bg = "male"
        gender.append(bg)

file_path.append(audio + i +
'/' + f)

# PUT EXTRACTED LABELS WITH
FILEPATH INTO DATAFRAME
audio_df =
pd.DataFrame(emotion)
audio_df =
audio_df.replace({1:'neutral'
, 2:'calm', 3:'happy',
4:'sad', 5:'angry', 6:'fear',
7:'disgust', 8:'surprise'})
audio_df =
pd.concat([pd.DataFrame(gende
r),audio_df,pd.DataFrame(acto
r)],axis=1)
audio_df.columns =
['gender','emotion','actor']
audio_df =
pd.concat([audio_df,pd.DataFr
```

Feature Extraction

Next, most importantly I used librosa's *librosa.feature.melspectrogram* and *librosa.power_to_db* to obtain the log-mel spectrogram values of each audio file and then averaged the spectrogram values and loaded the data into a new Dataframe labeled *df*.

```
# ITERATE OVER ALL AUDIO
FILES AND EXTRACT LOG MEL
SPECTROGRAM MEAN VALUES INTO
DF FOR MODELING
df = pd.DataFrame(columns=
['mel_spectrogram'])

counter=0
```

```
librosa.load(path,
res_type='kaiser_fast',durati
on=3,sr=44100,offset=0.5)

    #get the mel-scaled
spectrogram (ransform both
the y-axis (frequency) to log
scale, and the “color” axis
(amplitude) to Decibels,
which is kinda the log scale
of amplitudes.)
    spectrogram =
librosa.feature.melspectrogra
m(y=X, sr=sample_rate,
n_mels=128,fmax=8000)
    db_spec =
librosa.power_to_db(spectrogr
am)
    #temporally average
spectrogram
    log_spectrogram =
np.mean(db_spec, axis = 0)
```

```
print(len(df))
df.head()
```

Since this created an array of values under one column, I used *pd.concat* to turn the array into a list and join with my previous DataFrame *audio_df*, and dropped the necessary columns to give us the final DataFrame.

Image by Author

Get started Open in app

occurred in five steps:

1.Train, test split the data

```
train,test =
train_test_split(df_combined,
test_size=0.2,
random_state=0,

stratify=df_combined[['emotio
n','gender','actor']])


X_train = train.iloc[:, 3:]
y_train =
train.iloc[:,:2].drop(columns
=['gender'])

X_test = test.iloc[:,3:]
y_test =
```

2. Normalize Data — To improve model stability and performance

```
# NORMALIZE DATA
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)
X_train = (X_train - mean)/std
X_test = (X_test - mean)/std
```

3. Transform into arrays for Keras

```
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
```

4. One-hot encoding of target variable

```
# CNN REQUIRES INPUT AND
OUTPUT ARE NUMBERS
lb = LabelEncoder()
y_train =
to_categorical(lb.fit_transfo
rm(y_train))
y_test =
to_categorical(lb.fit_transfo
rm(y_test))
```

5. Reshape data to include 3D tensor

```
X_train =
X_train[:,:,np.newaxis]
X_test =
X_test[:,:,np.newaxis]
```

with a simply dummy classifier, which generated predictions by respecting the class distribution of the training data and had a low accuracy score of 11.81%. I then tried a Decision Tree since this is a multi-classification problem using the average log-mel spectrograms and got an accuracy score of 29.17%.

```
dummy_clf = DummyClassifier(strategy="stratified")
dummy_clf.fit(X_train, y_train)
DummyClassifier(strategy='stratified')
dummy_clf.predict(X_test)
dummy_clf.score(X_test, y_test)
```

Get started Open in app

```
y_train)
clf.predict(X_test)
clf.score(X_test, y_test)
```

Initial Model

For my initial model, I trained a 1D CNN with three convolutional layers and one output layer and obtained an accuracy score of 48% on my test set, which is slightly better than my baseline decision tree (*insert sad face here*).

```
# BUILD 1D CNN LAYERS
model = Sequential()
model.add(Conv1D(64,
kernel_size=(10),
activation='relu',
input_shape=
```

```
(10),activation='relu',kernel
_regularizer=l2(0.01),
bias_regularizer=l2(0.01)))
model.add(MaxPooling1D(pool_s
ize=(8)))
model.add(Dropout(0.4))
model.add(Conv1D(128,
kernel_size=
(10),activation='relu'))
model.add(MaxPooling1D(pool_s
ize=(8)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(256,
activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(8,
activation='sigmoid'))
opt =
keras.optimizers.Adam(lr=0.00
01)
model.compile(loss='categoric
al_crossentropy',
optimizer=opt,metrics=
```

Get started

Open in app

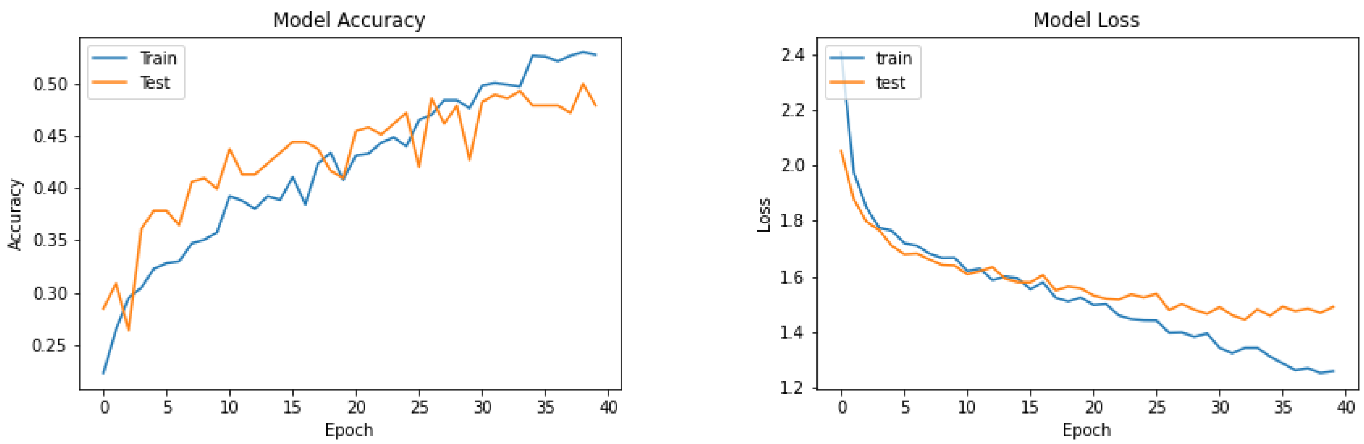


Image by Author

Data Augmentation

To improve the generalizability of my initial model, I explored Data Augmentation, but rather than augmenting images of the audio files, I augmented the audio file itself. Using custom functions provided by Eu Jin Lok on Kaggle, I added noise, stretch, speed also be done using Numpy and Librosa, explored here in this article.

As can be seen in the code above, applying data augmentation to the audio files and using feature extraction methods (also used log-mel spectograms), resulted in four Dataframes. These four DataFrames were combined in a similar manner as was done in the Data Preparation steps and followed the same preprocessing steps as described above. The Data Augmentation methods resulted in a much larger training set size of 5,760 images compared to 1,440 for my initial model.

```
model.add(Conv1D(64,
kernel_size=(20),
activation='relu',
input_shape=
(X_train.shape[1],1)))
model.add(Conv1D(128,
kernel_size=
(20),activation='relu',kernel
_regularizer=l2(0.01),
bias_regularizer=l2(0.01)))
model.add(MaxPooling1D(pool_s
ize=(8)))
model.add(Dropout(0.4))
model.add(Conv1D(128,
kernel_size=
(20),activation='relu'))
model.add(MaxPooling1D(pool_s
ize=(8)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(256,
activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(8,
```

```
keras.optimizers.Adam(lr=0.00
01)
```

Training a 1D CNN with three convolutional layers and one output layer resulted in a slightly higher accuracy score of 58% (*insert another sad face here*).

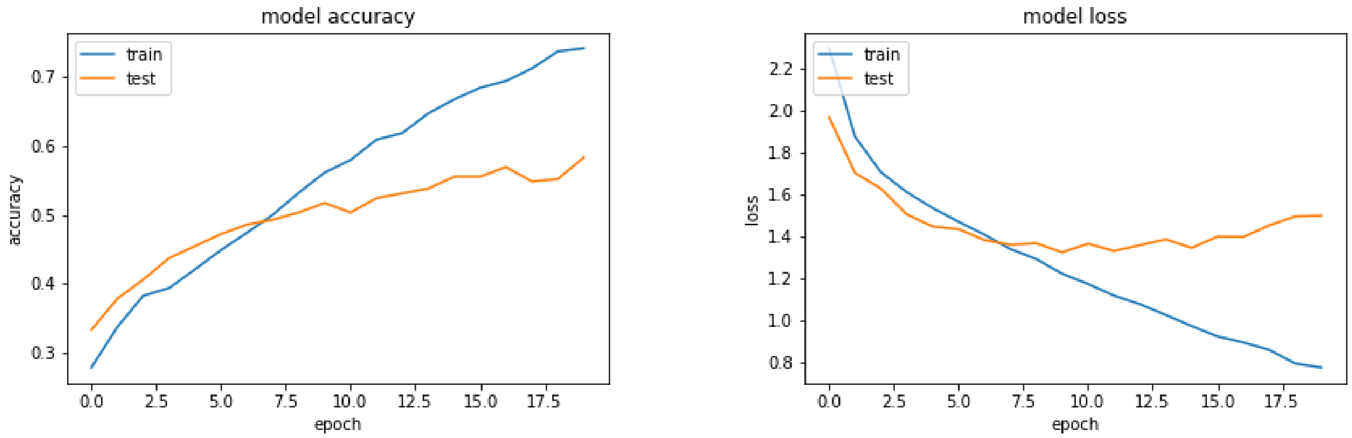


Image by Author

Next Steps

performance. Thank you for reading! :)

Full code is available on my GitHub.

Deep Learning