

**DATA SCIENCE**

Mini Project

on

# [Gender Recognition by Voice using Python](https://copyassignment.com/gender-recognition-by-voice-using-python/)

NAME : NATARAJ.C

REGISTER NO : 121012012756

DEGREE : B.TECH

BRANCH : CSE –3rd YEAR

SPECIALIZATION : DATA SCIENCE

SUBJECT : APPLIED ARTIFICAL INTELLIGENCE

SUB.CODE : XCSHD3

# [Gender Recognition by Voice using Python](https://copyassignment.com/gender-recognition-by-voice-using-python/)

# Project Overview :

|  |  |
| --- | --- |
| **Project Name:** | **Gender Recognition by Voice using Python** |
| **Abstract** | This will be an ML-based recognition system that can be used to recognize the gender of a person using his/her voice. |
| **Language/s Used:** | Python and Machine Learning Libraries |
| **IDE** | Jupyter Book or Google Colab |
| **Python version (Recommended):** | **Python 3** |

## Steps involved in Gender Recognition by Voice :

1. **Collection of data**
2. **Exploring the data**
3. **Audio feature extraction**
4. **Splitting the data**
5. **Building the model**
6. **Evaluating the model**

### 1. Collection of data

The very first step is to choose the dataset for our model. We can get a lot of different datasets from Kaggle and search for any voice dataset for the project.

The entire dataset is large .We can use the Kaggle notebook feature so that we can work with the data without downloading them all to the local storage. Since the total size of the dataset is huge, what we can do next is to load only the data from the cv-valid-train folder in which the corresponding audio details are stored in the cv-valid-train.csv file.

### 2. Exploring the data

As we have decided which data files to work with, now we can get even deeper into the dataset to find out details in the dataset. To start with, we are going to load all the required modules for this project. Before that, we have to install the python\_speech\_features module.

we have to load the data into our project. After all modules and the dataset are loaded, now we are able to check the first 5 rows of the dataset using df. *head()*.

We can see that the dataset got plenty of missing values, including the gender itself. Since the values in the gender column are going to be our target, it is important to filter out the *NaN*values in that column. In order to do so, we can create two new data frames called *df\_male* and *df\_female* which are used to store male and female voice details respectively.

We got another problem that is the dataset is extremely unbalanced. We can solve this problem by performing the under-sampling method. This is basically done by taking only a small portion of the available data such that the class distribution is going to be equal.

he audio file names in our dataset are actually having the extension of mp3. Actually, we need to convert these into wav because the Librosa module is just unable to read digital signals stored in mp3 format. In order to do that, we can use a function called convert\_to\_wav() with the help of the AudioSegment object taken from the Pydub module

After running the code above, now we should have new files stored in the working directory of the Kaggle notebook. We have to keep in mind that all these files might be saved in another directory instead depending on your specified path, especially when you use your local machine to do this project.

Up to this step, we already got 600 new wav files in which each of those are already having the prefix of either “male” or “female”. Therefore, we can ignore our data frame df that we used earlier and just focus on these new files as now we can extract the target label directly from the file name.

Furthermore, since the files are already having the extension of wav, then we can just employ the Librosa module to actually store the digital signal values into Python variables. Now we define the *load\_audio()* function to take the raw audio data and directly use it.important to keep in mind that the values stored in both male\_voices and female\_voices consist of the raw digital wave itself followed by the sample rate

### 3. Audio feature extraction

The machine learning algorithm works by labeling samples based on given features. We can think of the labels as y (dependent variable) and features as X (independent variables). In the case of voice recognition, feature extraction plays a big role since basically raw audio data is not quite informative and machine learning algorithms may unable to detect patterns in it.

There are actually plenty of audio feature extraction methods existing out there, and we will use MFCC due to its easy implementation and high recognition accuracy. The entire feature extraction process is wrapped in the extract\_features() function.

### 4. Splitting the data

Up to this, we already got a male\_concatenated and female\_concatenated array in which both of them store all male and female voices in two long arrays. Next, we are also going to concatenate the two arrays and store the result in X. y variable is necessary to be created as well in order to store all feature labels. Additionally, it is important to know that the labels here are in form of encoded values where 0 represents male and 1 denotes female. After that, we are going to split this X-y pair into training and testing data.

### 5.Building the model

To use the SVM model as the voice classifier. We can also use any other model in this case, like logistic regression or even a neural network.

### 6. Evaluating the model

Now let’s evaluate this SVM classifier model. We can see that the model achieves 78.2% and 76.8% of accuracy towards training and testing data respectively. The confusion matrix on test data

The Support Vector Machine algorithm with Radial Basis Function is able to identify whether a voice is spoken by a male (0) or female (1) with better accuracy.

**Code for Gender Recognition by Voice using Python :**

**# Install python\_speech\_features module**

**!pip install python\_speech\_features**

**# Import all modules**

**import os**

**import librosa**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from tqdm import tqdm**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.svm import SVC**

**from sklearn.metrics import confusion\_matrix**

**from pydub import AudioSegment**

**from python\_speech\_features import mfcc**

**from time import time**

**# Load the csv file into data frame**

**df = pd.read\_csv('../input/common-voice/cv-valid-train.csv')**

**# Create two new data frames**

**df\_male = df[df['gender']=='male']**

**df\_female = df[df['gender']=='female']**

**# Find out the number of rows**

**print(df\_male.shape)**

**# output: (55029, 8)**

**print(df\_female.shape)**

**# output: (18249, 8)**

**# Take only 300 male and 300 female data**

**df\_male = df\_male[:300]**

**df\_female = df\_female[:300]**

**# Define the audio path**

**TRAIN\_PATH = '../input/common-voice/cv-valid-train/'**

**# The function to convert mp3 to wav**

**def convert\_to\_wav(df, m\_f, path=TRAIN\_PATH):**

**srcs = []**

**for file in tqdm(df['filename']):**

**sound = AudioSegment.from\_mp3(path+file)**

**# Create new wav files based on existing mp3 files**

**if m\_f == 'male':**

**sound.export('male-'+file.split('/')[-1].split('.')[0]+'.wav', format='wav')**

**elif m\_f == 'female':**

**sound.export('female-'+file.split('/')[-1].split('.')[0]+'.wav', format='wav')**

**return**

**# How to use the convert\_to\_wav() function**

**convert\_to\_wav(df\_male, m\_f='male')**

**convert\_to\_wav(df\_female, m\_f='female')**

**# Define a function to load the raw audio files**

**def load\_audio(audio\_files):**

**# Allocate empty list for male and female voices**

**male\_voices = []**

**female\_voices = []**

**for file in tqdm(audio\_files):**

**if file.split('-')[0] == 'male':**

**male\_voices.append(librosa.load(file))**

**elif file.split('-')[0] == 'female':**

**female\_voices.append(librosa.load(file))**

**# Convert the list into Numpy array**

**male\_voices = np.array(male\_voices)**

**female\_voices = np.array(female\_voices)**

**return male\_voices, female\_voices**

**# How to use load\_audio() function**

**male\_voices, female\_voices = load\_audio(os.listdir())**

**# The function to extract audio features**

**def extract\_features(audio\_data):**

**audio\_waves = audio\_data[:,0]**

**samplerate = audio\_data[:,1][1]**

**features = []**

**for audio\_wave in tqdm(audio\_waves):**

**features.append(mfcc(audio\_wave, samplerate=samplerate, numcep=26))**

**features = np.array(features)**

**return features**

**# Use the extract\_features() function**

**male\_features = extract\_features(male\_voices)**

**female\_features = extract\_features(female\_voices)**

**# The function used to concatenate all audio features forming a long 2-dimensional array**

**def concatenate\_features(audio\_features):**

**concatenated = audio\_features[0]**

**for audio\_feature in tqdm(audio\_features):**

**concatenated = np.vstack((concatenated, audio\_feature))**

**return concatenated**

**# How the function is used**

**male\_concatenated = concatenate\_features(male\_features)**

**female\_concatenated = concatenate\_features(female\_features)**

**print(male\_concatenated.shape)**

**# Output: (117576, 26)**

**print(female\_concatenated.shape)**

**# Output: (124755, 26)**

**# Concatenate male voices and female voices**

**X = np.vstack((male\_concatenated, female\_concatenated))**

**# Create labels**

**y = np.append([0] \* len(male\_concatenated), [1] \* len(female\_concatenated))**

**# Check whether X and y are already having the exact same length**

**print(X.shape)**

**# Output: (242268, 26)**

**print(y.shape)**

**# Output: (242268,)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=22)**

**# Initialize SVM model**

**clf = SVC(kernel='rbf')**

**# Train the model**

**start = time()**

**clf.fit(X\_train[:50000], y\_train[:50000])**

**print(time()-start)**

**# Output: 184.8018662929535 (seconds)**

**# Compute the accuracy score towards train data**

**start = time()**

**print(clf.score(X\_train[:50000], y\_train[:50000]))**

**# Output: 0.78204**

**print(time()-start)**

**# Output: 90.8693311214447 (seconds)**

**# Compute the accuracy score towards test data**

**start = time()**

**print(clf.score(X\_test[:10000], y\_test[:10000]))**

**# Output: 0.7679**

**print(time()-start)**

**# Output: 18.082067728042603 (seconds)**

**# Predict the first 10000 test data**

**svm\_predictions = clf.predict(X\_test[:10000])**

**# Create the confusion matrix values**

**cm = confusion\_matrix(y\_test[:10000], svm\_predictions)**

**# Create the confusion matrix display**

**plt.figure(figsize=(8,8))**

**plt.title('Confusion matrix on test data')**

**sns.heatmap(cm, annot=True, fmt='d',**

**cmap=plt.cm.Blues, cbar=False, annot\_kws={'size':14})**

**plt.xlabel('Predicted Label')**

**plt.ylabel('True Label')**

**plt.show()**

**# Performance comparison between different algorithms**

**index = ['SVM-RBF', 'SVM-Poly', 'SVM-Sigmoid', 'Logistic Regression']**

**# I record all the results below manually**

**values = [184.8, 137.0, 283.6, 0.7]**

**plt.figure(figsize=(12,3))**

**plt.title('Training duration (lower is better)')**

**plt.xlabel('Seconds')**

**plt.ylabel('Model')**

**plt.barh(index, values, zorder=2)**

**plt.grid(zorder=0)**

**for i, value in enumerate(values):**

**plt.text(value+20, i, str(value)+' secs', fontsize=12, color='black',**

**horizontalalignment='center', verticalalignment='center')**

**plt.show()**

**# set width of bar**

**barWidth = 0.25**

**index = ['SVM-RBF', 'SVM-Poly', 'SVM-Sigmoid', 'Logistic Regression']**

**# set height of bar**

**# I record all the results below manually**

**train\_acc = [78.2, 74.8, 74.8, 65.8]**

**test\_acc = [76.8, 74.3, 74.3, 65.8]**

**# Set position of bar on X axis**

**baseline = np.arange(len(train\_acc))**

**r1 = [x + 0.125 for x in baseline]**

**r2 = [x + 0.25 for x in r1]**

**# Make the plot**

**plt.figure(figsize=(16,9))**

**plt.title('Model performance (higher is better)')**

**plt.bar(r1, train\_acc, width=barWidth, label='Train', zorder=2)**

**plt.bar(r2, test\_acc, width=barWidth, label='Test', zorder=2)**

**plt.grid(zorder=0)**

**# Add xticks on the middle of the group bars**

**plt.xlabel('Model')**

**plt.ylabel('Accuracy')**

**plt.xticks([r + barWidth for r in range(len(train\_acc))], index)**

**# Create text**

**for i, value in enumerate(train\_acc):**

**plt.text(i+0.125, value-5, str(value), fontsize=12, color='white',**

**horizontalalignment='center', verticalalignment='center')**

**for i, value in enumerate(test\_acc):**

**plt.text(i+0.375, value-5, str(value), fontsize=12, color='white',**

**horizontalalignment='center', verticalalignment='center')**

**plt.legend()**

**plt.show()**

## Conclusion

The Gender Recognition by Voice using Python. We have used the SVM(support vector machine) algorithm which is a supervised learning algorithm used for classification.