```
#MOUNT TO SAVE AND LOAD df of built datasets
#to mount the personal 1612 gmail drive (IMP)
from google.colab import drive
drive.mount('/content/drive')
```

```
---------------------------------------------------------------------------
MessageError                                Traceback (most recent call last)
/tmp/ipython-input-495556469.py in <cell line: 0>()
      2 #to mount the personal 1612 gmail drive (IMP)
      3 from google.colab import drive
----> 4 drive.mount('/content/drive')

 3 frames 
/usr/local/lib/python3.12/dist-packages/google/colab/_message.py in
read_reply_from_input(message_id, timeout_sec)
    101     ):
    102       if 'error' in reply:
--> 103         raise MessageError(reply['error'])
    104       return reply.get('data', None)
    105

MessageError: Error: credential propagation was unsuccessful
```

```
df_path = '/content/drive/MyDrive/AiAudio_PR_Project/AiAudio_Dataset'
for_2sec_df_path = df_path+'/for_2sec' #will use it to train and select best model
for_2sec_rerec_df_path = df_path+'/for_2sec_rerec' #retrain, for our scenario, on the s

print("Path to dataset files:", for_2sec_df_path)
print("Path to dataset files:", for_2sec_rerec_df_path)
```

```
Start coding or generate with AI.
```

Getting data

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("mohammedabdeldayem/the-fake-or-real-dataset")
```

```
import os
cnn_path = os.path.join(path, "for-2sec/for-2seconds")
for_2sec_path=os.path.join(path, "for-2sec/for-2seconds/training")
for_2sec_test_path=os.path.join(path, "for-2sec/for-2seconds/testing")
for_2sec_valid_path=os.path.join(path, "for-2sec/for-2seconds/validation")



print("Path to dataset files:", path)
print("Path to dataset files:", for_2sec_path)
```

```
#RERECORDED FoR DATA, FOR INFERENCE
for_rerec_path_real = os.path.join(path, "for-rerec/for-rerecorded/validation/real")
for_rerec_path_fake = os.path.join(path, "for-rerec/for-rerecorded/validation/fake")

print("Path to rerec real files:", for_rerec_path_real)
print("Path to rerec fake files:", for_rerec_path_fake)
```

Feature Extraction

Function

```
import os
import librosa
import numpy as np
import pandas as pd

def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=16000)


    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)

    delta_mfccs = librosa.feature.delta(mfccs)

    spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)

    spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)

    chroma = librosa.feature.chroma_stft(y=y, sr=sr)

    zcr = librosa.feature.zero_crossing_rate(y)

    rms = librosa.feature.rms(y=y)


    features = []
    for feature_set in [mfccs, delta_mfccs, spec_cent, spec_bw, chroma, zcr, rms]:
        features.extend(np.mean(feature_set, axis=1))
        features.extend(np.std(feature_set, axis=1))

    return features


def build_dataset(base_dir):
    data = []
    for label_dir in ['real', 'fake']:
        folder = os.path.join(base_dir, label_dir)
        label = 0 if label_dir == 'real' else 1
        for fname in os.listdir(folder):
```

```
                fpath = os.path.join(folder, fname)
                try:
                    feats = extract_features(fpath)
                    data.append([fpath] + feats + [label])
                except Exception as e:
                    print(f"Error with {fpath}: {e}")
        return data


    feature_names = [f'mfcc{i}' for i in range(13)] + \
                    [f'delta_mfcc{i}' for i in range(13)] + \
                    ['spec_cent', 'spec_bw'] + \
                    [f'chroma{i}' for i in range(12)] + \
                    ['zcr', 'rms']

    feature_names = [f"{f}_{stat}" for f in feature_names for stat in ['mean', 'std']]
    df_cols = ['filename'] + feature_names + ['label']
```

for_2sec Dataset

No need to run Below again, just load pre-save(unprocessed)

The training dataset

```
dataset = build_dataset(for_2sec_path)
df = pd.DataFrame(dataset, columns=df_cols)
```

```
df.head()
```

```
df.tail()
```

```
df.shape
```

```
print(for_2sec_df_path+"/training_df.pkl")
```

```
df.to_pickle(f"{for_2sec_df_path}/training_df.pkl")
```

The testing set

↳ 5 cells hidden

Loading Dataset

for_2sec

```
df = pd.read_pickle(f"{for_2sec_df_path}/training_df.pkl")
df_test = pd.read_pickle(f"{for_2sec_df_path}/testing_df.pkl")
```

```
print(df.columns.tolist())
```

Preprocessing

↳ 4 cells hidden

Models Selection

↳ 34 cells hidden

MODEL

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import f1_score, accuracy_score, classification_report
```

```
model_path = '/content/drive/MyDrive/AiAudio_PR_Project/AiAudio_Model'
svm_m_path = model_path+'/svm' #will use it to train and select best model
svm_reduced_m_path = model_path+'/svm_reduced'
lr_m_path = model_path+'/lr'
lr_reduced_m_path = model_path + '/lr_reduced' #retrain, for our scenario, on the selecte
loris1_m_path = model_path + '/loris1'
loris2_m_path = model_path + '/loris2'

print("Path to SVM Model:", svm_m_path)
print("Path to SVM Reduced Model:", svm_reduced_m_path)
print("Path to LR Model:", lr_m_path)
```

Saving Model

SVM

```
import pickle

with open(svm_m_path, "wb") as f:
    pickle.dump(svm, f)

print(f"Model saved at: {svm_m_path}")
```

LR

```
import pickle

with open(lr_m_path, "wb") as f:
    pickle.dump(lr, f)

print(f"Model saved at: {lr_m_path}")
```

SVM_REDUCED (NO CHROMA)

```
import pickle

with open(svm_reduced_m_path, "wb") as f:
    pickle.dump(svm_reduced, f)

print(f"Model saved at: {svm_reduced_m_path}")
```

LR_Reduced

```
import pickle

with open(lr_reduced_m_path, "wb") as f:
    pickle.dump(lr_reduced, f)

print(f"Model saved at: {lr_reduced_m_path}")
```

Loris 1 and 2

```
import pickle

with open(loris1_m_path, "wb") as f:
    pickle.dump(loris1, f)

print(f"Model saved at: {loris1_m_path}")
```

```
import pickle

with open(loris2_m_path, "wb") as f:
    pickle.dump(loris2, f)

print(f"Model saved at: {loris2_m_path}")
```

Loading Model

```
import pickle

with open(svm_m_path, "rb") as f:
    svm = pickle.load(f)
```

```
import pickle

with open(svm_reduced_m_path, "rb") as f:
    svm_reduced = pickle.load(f)
```

```
import pickle

with open(lr_reduced_m_path, "rb") as f:
    lr_reduced = pickle.load(f)
```

```
import pickle

with open(loris1_m_path, "rb") as f:
    loris1 = pickle.load(f)
```

```
import pickle

with open(loris1_m_path, "rb") as f:
    loris1 = pickle.load(f)
```

```
import pickle

with open(loris2_m_path, "rb") as f:
    loris2 = pickle.load(f)
```