음성파일에 감정과 상관관계에 있는 요소들이 무엇이 있을까 생각을 해보았습니다. 음높이, 강도, 템포, 음색 등등을 생각해 내었고 이 네 가지를 이용하여 모델을 만들면 좋을 것 같다고 판단하였습니다. 데이터 셋으로부터 이 특징들을 추출하기 위하여 numpy와 librosa를 이용하였습니다.

코드 설명

# Load the dataset

train\_df = pd.read\_csv('train.csv')

test\_df = pd.read\_csv('test.csv')

데이터셋을 로드합니다. 데이터 셋 형식은 리드미에 있습니다.

# Define a directory to store the extracted features

features\_dir = 'extracted\_features'

os.makedirs(features\_dir, exist\_ok=True)

데이터셋 전처리 결과를 저장할 폴더를 만듭니다. 전처리 과정이 너무 오래 걸려서 최초실행시에만 추출하고 그 뒤로는 저장된 값을 사용하게 했습니다.

def preprocess\_audio(file\_path):

    # Check if the features are already extracted and saved

    features\_file = os.path.join(features\_dir, os.path.splitext(os.path.basename(file\_path))[0] + '.npy')

    if os.path.exists(features\_file):

        print(f"Loading features from {features\_file}")

        return np.load(features\_file)

전처리를 담당하는 함수입니다. 먼저 추출된 특징이 이미 있으면 백업파일을 사용하게 했습니다.

    audio, sample\_rate = librosa.load(file\_path, res\_type='kaiser\_fast', mono=True)

    # Extract pitch/frequency, intensity/loudness, tempo/rhythm, and timbre/tone features

    pitch = librosa.yin(audio, fmin=100, fmax=1000)

    energy = np.mean(audio \*\* 2)

Librosa와 Numpy를 이용하여 pitch, intensity 를 추출합니다.

    onset\_env = librosa.onset.onset\_strength(y=audio, sr=sample\_rate)

    tempo = librosa.beat.tempo(onset\_envelope=onset\_env, sr=sample\_rate)[0]

각 시점에 발생할 수 있는 ‘onset’, 오디오 신호의 특별한 변화점들을 추출한 뒤, 이를 이용하여 tempo를 추출합니다.

    mfccs = librosa.feature.mfcc(y=audio, sr=sample\_rate, n\_mfcc=13)

    delta\_mfccs = librosa.feature.delta(mfccs)

    delta2\_mfccs = librosa.feature.delta(mfccs, order=2)

이번엔 톤을 추출하기 위해 mfccs를 추출합니다. Mfcc는 Mel-frequency cepstral coefficients로 노이즈를 제거하고 중요한 음색, 톤을 추출하는 기능을 합니다. 이렇게 추출한 톤의 변화율과 가속률을 또 추출합니다.

    # Pad or truncate the features to a fixed size

    max\_len = 1000

    pitch = np.pad(pitch, (0, max\_len - len(pitch)))

    mfccs = np.pad(mfccs, ((0, 0), (0, max\_len - mfccs.shape[1])))

    delta\_mfccs = np.pad(delta\_mfccs, ((0, 0), (0, max\_len - delta\_mfccs.shape[1])))

    delta2\_mfccs = np.pad(delta2\_mfccs, ((0, 0), (0, max\_len - delta2\_mfccs.shape[1])))

    features = np.concatenate((pitch, [energy], [tempo], mfccs.flatten(), delta\_mfccs.flatten(), delta2\_mfccs.flatten()))

    # Save the extracted features

    np.save(features\_file, features)

    print(f"Extracted features saved to {features\_file}")

    return features

이후 행렬계산을 위해 max\_len으로 패딩을 하고 추출값을 백업합니다.

train\_df['audio\_features'] = train\_df['path'].apply(preprocess\_audio)

test\_df['audio\_features'] = test\_df['path'].apply(preprocess\_audio)

데이터 셋을 전처리합니다.

# Prepare the data for machine learning

X = np.stack(train\_df['audio\_features'].values)

y = train\_df['label'].values

# Split the data

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Encode the labels

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_val\_encoded = label\_encoder.transform(y\_val)

추출한 특징들을 X에 정리하고 y에 label을 정리합니다. 그후 validation을 위해 0.2의 비율로 나눕니다. 그후 라벨을 숫자로 encode를 합니다.

print("Training the model...")

# Design and train a model

model = SVC()

# model = RandomForestClassifier()

#model = LogisticRegression()

model.fit(X\_train, y\_train\_encoded)

# Evaluate the model

val\_predictions = model.predict(X\_val)

val\_accuracy = accuracy\_score(y\_val\_encoded, val\_predictions)

print("Validation accuracy:", val\_accuracy)

학습을 시킵니다. SVC모델을 사용하였고 이후 Random Forest, Logistic Regression 으로도 해보았습니다.

# Predict emotions for the test set

X\_test = np.stack(test\_df['audio\_features'].values)

test\_predictions = model.predict(X\_test)

test\_df['predicted\_label'] = label\_encoder.inverse\_transform(test\_predictions)

# Save predictions or further analysis

test\_df.to\_csv('test\_predictions.csv', index=False)

학습된 모델을 바탕으로 test 데이터에 테스트를 합니다.

결과

Validation accuracy 같은 경우 SVC, RandomForestClassifier 모두 약 0.42로 나왔고 Logistic Regression은 0.38이 나왔습니다. test set에 의한 결과로는 최고 0.44798로 나왔습니다. 6지선다를 찍어서 맞을 확률이 0.16임을 감안하면 찍는 것에 비해 약 2.7배의 성능을 보여주지만 그래도 낮은 성능임을 알 수 있습니다. 모델을 고르는 것 보다 전처리가 중요하다는 것을 알았습니다.