



DACON - Sound Emotion Classification Competition

AI Competition held by DACON.

For details of the competition and DACON, please refer to:

데이터사이언티스트 AI 컴피티션

7만 AI 팀이 협업하는 데이터 사이언스 플랫폼. AI 경진대회와 대상 맞춤형 온/오프라인 교육, 문제 기반 학습 서비스를 제공합니다.


 <https://dacon.io/>



• 월간 데이콘 음성 감정 인식 AI 경진대회

월간 데이콘 음성 감정 인식 AI 경진대회

출처 : DACON - Data Science Competition

 <https://dacon.io/competitions/official/236105/overview/description>



This is my **second time** joining for a Data Science competition.

For this challenge, we are permitted to use only the given dataset (TRAIN, TEST).

There are 5001 .wav format files, 1881 .wav format files for each TRAIN and TEST, and .csv files with 3 columns `id`, `path`, `label` for each TRAIN and TEST.

`id` column contains the name of each `.wav` file.

`path` column contains the path to each `.wav` file.

`label` column contains the label of each `.wav` file ⇒ This label indicates which emotion each `.wav` file is showing.

Here are the specific requirements for the competition.

▼ Requirements

Background

Hello everyone! Welcome to the monthly Deacon Speech Emotion Recognition AI Competition.

Speech emotion recognition is a technology that identifies people's emotional states.

Create an AI model that can judge people's emotions based on acoustic data!

Theme

Developing emotion recognition AI models based on acoustic data

Description

Create a model to recognize emotions from acoustic data!

Evaluation

Judging Criteria: Accuracy

Primary Evaluation (Public Score): Scored on a randomly sampled 30% of the test data, publicly available during the competition.

Secondary Evaluation (Private Score): Scored on the remaining 70% of the test data, released immediately after the competition ends

The final ranking will be scored from the selected files, so participants must select the two files they would like to have scored in the submission window.

The final ranking is based on the highest score of the two selected files.

(If the final file is not selected, the first submitted file is automatically selected)
The Private Score ranking released immediately after the competition is not the final ranking; the final winners will be determined after code verification.
Determine the final ranking based on Private Score among submitting teams that have complied with the competition evaluation rules.

Individual or Team Participation Rules

You can participate as an individual or as a team.

Separate procedures are required for group or institutional participation. (More > Notice > Check the post)

How to participate as an individual: No need to apply for a team, you can submit freely in the submission window.

How to form a team: Check the team page for team formation instructions.

Maximum team size: 5 people

* The same person cannot register as an individual or multiple teams.

Number of submissions per day: 3 times

Data Acceptability

No use of external data other than competition-provided data

Among the competition-provided data, Test Dataset can only be used for inference for leaderboard submission (no model training)

Code may be requested in case of suspected data leakage (Note: Personal cleanup of data leakage) or rule violation

Pre-trained models allowed

You can use pre-trained models (Pretrained Model Weight) that have no legal restrictions on use and have been published in papers.

However, you cannot use pre-trained models that contain test datasets.

You cannot use pretrained models with CREMA-D dataset

If you are unable to check whether the pre-trained model you want to use contains a test dataset, please contact dacon@dacon.io.

User Evaluation

Teams wishing to receive an award certificate must first undergo user evaluation.

Upload your code to the code sharing page within the code submission period after your private ranking is released.

Include your private ranking, the model you used, and keywords for the code in the title

Example.) Private 1st place, LGBM model, utilizing 00 preprocessing technique

Competition participants evaluate the publicly available code
Comment on code errors, use of external data, etc. as comments

Notes

Maximum number of submissions per day: 3 times

Available languages: Python, R

Utilization of evaluation datasets in model training (Data Leakage) is not eligible for awards.

All learning, inference processes, and inference outputs must be based on legitimate code, and submissions obtained by abnormal means will be considered a rule violation if detected.

The final ranking will be scored from the selected files, so participants must select the files they want to be scored in the submission window.

Private rankings released immediately after the competition are not final and winners will be determined after code verification.

Daycon strictly prohibits fraudulent submissions, and if you have a history of fraudulent submissions to Daycon competitions, your evaluation will be restricted.

It was already D-5 from final submission date when I found out about this competition.

The remaining time was obviously insufficient, but I still decided to give it a try.

The purpose for me to join this competition was simple.

I heard the acoustic dataset is rare, I want to experience it!

For this competition what I have to do is to classify the TEST dataset into 6 different categories based on the TRAIN dataset which the categories are already marked for each data.

Categories are as below.

0: angry

1: fear

2: sad

3: disgust

4: neutral

5: happy

Let me first set up the `ipynb` file for this.

There could be more imported library the more I get into this.

▼ Codes

```
# Importing
import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import glob
from tqdm.auto import tqdm
from sklearn.model_selection import train_test_split
from custom_dataset import (CustomDataSet,
                             speech_file_to_array_fn as sfaf,
                             collate_fn,
                             create_data_loader,
                             validation,
                             train)

from transformers import (Wav2Vec2FeatureExtractor,
                          Wav2Vec2Model,
                          Wav2Vec2Config,
                          Wav2Vec2ConformerForSequenceClassification,
                          AutoModelForAudioClassification)

import evaluate
import librosa
import random
import torch
import torchaudio
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import Dataset, DataLoader
from sklearn.tree import DecisionTreeClassifier

from datasets import load_dataset, Audio
import warnings
warnings.filterwarnings(action='ignore')
os.environ['CUDA_LAUNCH_BLOCKING']='1'
```

▼ Decision Tree Classifier (Success)

Below is my theory.

1. Get the numerically expressed frequency data of each .wav TRAIN data
2. Find the trend of each category, and let the Machine Learn.
3. Then apply the trained machine to the TEST dataset.

4. Voila, here is the final submission .csv file.

As I was working alone for this competition, and no one around me had any experience in Acoustic data, I had to deal everything by myself.

Then I have first set each csv into variable and checked the information.

▼ Codes

```
train_df = pd.read_csv('./train.csv')
print(train_df.info())

test_df = pd.read_csv('./test.csv')
print(test_df.info())

-----

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 3 columns):
#   Column  Non-Null Count  Dtype
---  -
0    id      5001 non-null     object
1   path    5001 non-null     object
2   label   5001 non-null     int64
dtypes: int64(1), object(2)
memory usage: 117.3+ KB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1881 entries, 0 to 1880
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0    id      1881 non-null     object
1   path    1881 non-null     object
dtypes: object(2)
memory usage: 29.5+ KB
None
```

To finalize the setting-up process, I have created the folders that will be used later, and set paths as variable.

Also, have made dictionary to easily locate the files in each folders.

▼ Codes

```

# Folder Locations

dataset = "./"
TRAIN_WAV = dataset + "train/"
TEST_WAV = dataset + "test/"
PREPROCESSED = dataset + "preprocessed_data/"
TRAIN_LABEL_SEP = PREPROCESSED + "train_label_sep/"
WAV_TRAIN_LABEL_SEP = PREPROCESSED + "wav_train_label_sep/"
TEST_LABEL_SEP = PREPROCESSED + "test_label_sep/"
WAV_TEST_LABEL_SEP = PREPROCESSED + "wav_test_label_sep/"

if not os.path.exists(dataset + "preprocessed_data"):
    os.mkdir(dataset + "preprocessed_data")

if not os.path.exists(PREPROCESSED + "train_label_sep"):
    os.mkdir(PREPROCESSED + "train_label_sep")

if not os.path.exists(PREPROCESSED + "test_label_sep"):
    os.mkdir(PREPROCESSED + "test_label_sep")

if not os.path.exists(PREPROCESSED + "wav_train_label_sep"):
    os.mkdir(PREPROCESSED + "wav_train_label_sep")

if not os.path.exists(PREPROCESSED + "wav_test_label_sep"):
    os.mkdir(PREPROCESSED + "wav_test_label_sep")

wav_file_dict = {"train_wav" : TRAIN_WAV,
                 "test_wav" : TEST_WAV,
                 "wav_sep" : WAV_TRAIN_LABEL_SEP
                 }
wav_file_locations = {}
for key, value in wav_file_dict.items():
    wav_file_locations[key] = glob.glob(value + "*.wav")

csv_file_dict = {"train_label_sep" : TRAIN_LABEL_SEP,
                 "wav_train_label_sep" : WAV_TRAIN_LABEL_SEP
                 }

csv_file_location = {}
for key, value in csv_file_dict.items():
    csv_file_location[key] = glob.glob(value + "*.csv")

```

Before properly get started, I failed one attempt, and here is the record of it.

I have searched a bit about on how to deal with the acoustic data.

After searching for a while, I figured out that the

`librosa` library was the latest technology to convert the .wav data into numeric values.

`librosa.feature.mfcc` is what I used (<https://librosa.org/doc/latest/feature.html>)

MFCC stands for Mel-Frequency Cepstral Coefficient, which is an algorithms that converts the Acoustic data into characteristics vector (features).

MFCC takes each vector as 1 dimension.

▼ Failed Attempt

I tried to separate the train.csv, grouping by the classification, the `label` column.

I have successfully separated the train.csv into 6 different files and copy&separated the .wav files into different folders according to the segmentation on train.csv file.

Then I read each .wav file in different folders to process them to 6 different .csv files to let the machine learn the patter of each classification without any confusion.

▼ Codes

```
train_df = pd.read_csv("train.csv")

for label in train_df['label'].unique():
    filtered_df = train_df[train_df['label'] == label]
    filename = f"label_{label}.csv"
    filtered_df.to_csv(filename, index=False)

source_folder = TRAIN_WAV

for label in train_csv['label'].unique():
    filtered_csv_filename = f"label_{label}.csv"
    filtered_csv = pd.read_csv(os.path.join(TRAIN_LABEL_SEP, filtered_csv
_filename))

    target_folder = WAV_TRAIN_LABEL_SEP

    for path in filtered_csv['path']:
        wav_filename = os.path.basename(path)
        source_path = os.path.join(source_folder, wav_filename)
        target_path = os.path.join(target_folder, wav_filename)
        shutil.copyfile(source_path, target_path)

# <Dead code below as I though it was waste of the memory to read through
```



```

each wav files again.>

parent_folder = WAV_TRAIN_LABEL_SEP
features_df = pd.DataFrame()

for label_folder in tqdm(os.listdir(parent_folder), desc="Processing folders"):
    if label_folder == ".DS_Store":
        continue

    label_folder_path = os.path.join(parent_folder, label_folder)

    for wav_file in tqdm(os.listdir(label_folder_path), desc="Processing files"):
        wav_path = os.path.join(label_folder_path, wav_file)

        audio, sr = librosa.load(wav_path)
        mfcc = librosa.feature.mfcc(y=audio, sr=sr)
        mfcc_flattened = mfcc.flatten()
        row = pd.DataFrame([list(mfcc_flattened) + [label]])
        features_df = pd.concat([features_df, row])

features_df.reset_index(drop=True, inplace=True)

features_df.to_csv(dataset + "audio_features.csv", index=False)

# </Dead code until here>
# I decided to rather perform the function on the existing train folder
# (.wav files not separated)

parent_folder = TRAIN_WAV
features_df = pd.DataFrame()

for label in train_csv['label'].unique():
    filtered_csv_filename = f"label_{label}.csv"
    filtered_csv = pd.read_csv(os.path.join(TRAIN_LABEL_SEP, filtered_csv_filename))

    for path in filtered_csv['path']:
        wav_filename = os.path.basename(path)
        wav_path = os.path.join(parent_folder, wav_filename)

        audio, sr = librosa.load(wav_path)
        mfcc = librosa.feature.mfcc(y=audio, sr=sr)
        mfcc_flattened = mfcc.flatten()
        row = pd.DataFrame([list(mfcc_flattened) + [label]])
        features_df = pd.concat([features_df, row])

features_df.reset_index(drop=True, inplace=True)

features_df.to_csv(dataset + "audio_features.csv", index=False)

```

However, I realized that this was time and memory consuming process to separate all files in each classification and gather them again.

So I decided to cancel this method.

Also, for the Machine Learning Method, I decided to use Decision Tree Classifier. Simply, I thought it was easiest way to make it learn the trends per classification, and apply to the test .wav data.

In order to work on it, I had to first set up CFG dictionary.

SR refers to Sampling Rate

N_MFCC refers to number of MFCC (dimensions)

SEED refers to the seed for randomization

```
CFG = {  
    'SR': 16000,  
    'N_MFCC': 32,  
    'SEED': 42  
}
```

Then created a function definition

1. It takes the DataFrame, in my case, train.csv, as an input.
2. It takes the **path** column to get the path of each .wav file.
3. It load the .wav file and separates the **y** and **sr**

y here refers to the audio

4. Made **mfcc** a variable by applying **librosa.feature.mfcc** to **y**, by using **sr** and **n_mfcc**
5. Appended numerical mean of each vector of **mfcc** into a list **y_feature**
6. Appended the list **y_feature** into list **features** which now is a 2 dimension array
7. Placed list **features** into a DataFrame **mfcc_df**, so that it can be accessible.
8. Returned **mfcc_df**

▼ **Codes**

```
def get_mfcc_feature(df):
    features = []
    for path in tqdm(df['path']):
        y, sr = librosa.load(path, sr=CFG['SR'])
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=CFG['N_MFCC'])
        y_feature = []
        for e in mfcc:
            y_feature.append(np.mean(e))
        features.append(y_feature)

    mfcc_df = pd.DataFrame(features, columns=['mfcc_'+str(x) for x in range(1,
CFG['N_MFCC']+1)])
    return mfcc_df
```

To make the `train_df` and `test_df` into the form that I can later use in Machine Learning, I have applied the `get_mfcc_feature` to each df.

▼ Codes

```
train_x = get_mfcc_feature(train_df)
print(train_x)
test_x = get_mfcc_feature(test_df)
print(test_x)
```

Then checked how they looked like.

▼ Codes

```
print(train_x.info())
print('\n \n \n')
print(test_x.info())
```

▼ Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 32 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   mfcc_1      5001 non-null   float32
1   mfcc_2      5001 non-null   float32
2   mfcc_3      5001 non-null   float32
3   mfcc_4      5001 non-null   float32
```

```

4  mfcc_5    5001 non-null  float32
5  mfcc_6    5001 non-null  float32
6  mfcc_7    5001 non-null  float32
7  mfcc_8    5001 non-null  float32
8  mfcc_9    5001 non-null  float32
9  mfcc_10   5001 non-null  float32
10 mfcc_11   5001 non-null  float32
11 mfcc_12   5001 non-null  float32
12 mfcc_13   5001 non-null  float32
13 mfcc_14   5001 non-null  float32
14 mfcc_15   5001 non-null  float32
15 mfcc_16   5001 non-null  float32
16 mfcc_17   5001 non-null  float32
17 mfcc_18   5001 non-null  float32
18 mfcc_19   5001 non-null  float32
19 mfcc_20   5001 non-null  float32
20 mfcc_21   5001 non-null  float32
21 mfcc_22   5001 non-null  float32
22 mfcc_23   5001 non-null  float32
23 mfcc_24   5001 non-null  float32
24 mfcc_25   5001 non-null  float32
25 mfcc_26   5001 non-null  float32
26 mfcc_27   5001 non-null  float32
27 mfcc_28   5001 non-null  float32
28 mfcc_29   5001 non-null  float32
29 mfcc_30   5001 non-null  float32
30 mfcc_31   5001 non-null  float32
31 mfcc_32   5001 non-null  float32
dtypes: float32(32)
memory usage: 625.2 KB
None

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1881 entries, 0 to 1880
Data columns (total 32 columns):
#   Column      Non-Null Count  Dtype
---  -
0   mfcc_1      1881 non-null   float32
1   mfcc_2      1881 non-null   float32
2   mfcc_3      1881 non-null   float32
3   mfcc_4      1881 non-null   float32
4   mfcc_5      1881 non-null   float32
5   mfcc_6      1881 non-null   float32
6   mfcc_7      1881 non-null   float32
7   mfcc_8      1881 non-null   float32
8   mfcc_9      1881 non-null   float32
9   mfcc_10     1881 non-null   float32
10  mfcc_11     1881 non-null   float32
11  mfcc_12     1881 non-null   float32
12  mfcc_13     1881 non-null   float32
13  mfcc_14     1881 non-null   float32
14  mfcc_15     1881 non-null   float32
15  mfcc_16     1881 non-null   float32
16  mfcc_17     1881 non-null   float32

```

```
17 mfcc_18 1881 non-null float32
18 mfcc_19 1881 non-null float32
19 mfcc_20 1881 non-null float32
20 mfcc_21 1881 non-null float32
21 mfcc_22 1881 non-null float32
22 mfcc_23 1881 non-null float32
23 mfcc_24 1881 non-null float32
24 mfcc_25 1881 non-null float32
25 mfcc_26 1881 non-null float32
26 mfcc_27 1881 non-null float32
27 mfcc_28 1881 non-null float32
28 mfcc_29 1881 non-null float32
29 mfcc_30 1881 non-null float32
30 mfcc_31 1881 non-null float32
31 mfcc_32 1881 non-null float32
dtypes: float32(32)
memory usage: 235.2 KB
None
```

For future uses, I have assigned the `label` column of the `train_df` as `train_y`

▼ Codes

```
train_y = train_df['label']
print(train_y)
```

Preprocessing the data is done, and now it is time to let the Machine Learn!

I have first took `DecisionTreeClassifier` and made it variable.

Then trained the model with

`train_x` and `train_y`

▼ Codes

```
model = DecisionTreeClassifier(random_state=CFG['SEED'])
model.fit(train_x, train_y)
```

Predicted `test_x` with the trained model.

▼ Codes

```
preds = model.predict(test_x)
```

Made the predictions into csv file (codes) and submitted to the competition platform (manually) !

▼ Codes

```
submission = pd.read_csv(dataset + 'sample_submission.csv')
submission['label'] = preds
submission.to_csv(dataset + "baseline_submission.csv", index=False)
```

Here is the result

#	팀	팀 멤버	점수	제출수	등록일
183	swibeijason		0.32978	3	하루 전

I was originally 170 out of 180 competitors when I first submitted yesterday, but now I am ranked 183 out of 194 competitors.

After submitting twice more with different hyperparameters, I realized that the Decision Tree might not be the best-fit solution to this problem.

So I decided to try with other models.

▼ Transformer - wav2vec2 (Failed - Lack of PyTorch Knowledge)

After trying with DT Classifier, I decided to go for the Transformer model. It took a bit of research to find which model should I use, and figured out to use the

wav2vec2-large-xlsr-53 by facebook.

Here is the codes I modified after searching online.

▼ Codes

```

model = Wav2Vec2ForSequenceClassification.from_pretrained('facebook/wav2vec2-large-xlsr-53')

train_inputs = torch.tensor(train_x.values.astype(np.float32)).unsqueeze(1)
test_inputs = torch.tensor(test_x.values.astype(np.float32)).unsqueeze(1)

with torch.no_grad():
    train_logits = model(train_inputs).logits
    test_logits = model(test_inputs).logits

```

This was code was giving me so much identical error, no matter how much value I input into `.unsqueeze`

So I have deleted the `.unsqueeze` itself.

(Later found out that my

`mfcc` method already took the `batch` into account and didn't need to `unsqueeze` it in the first place)

After solving this problem, another error was just aroused.

I have googled, asked ChatGPT, but couldn't find solution to this problem.

▼ Error

```

RuntimeError: Calculated padded input size per channel: (2). Kernel size: (3).
Kernel size can't be greater than actual input size

```

Instead of moving forward with wav2vec2-large-xlsr-53, they recommended me to go with wav2vec2

It was Pickling Error, which I never encountered before.

I had to ask Google to solve this problem, and they told me to move all the classes into a separate .py file then import it from there.

Here is the full code I tried with, but had no luck moving forward from here.

▼ Codes (main.py)

```

from custom_dataset import (CustomDataSet,
                             speech_file_to_array_fn,
                             collate_fn,
                             create_data_loader,
                             validation,
                             train)

```

```

device = torch.device('cuda') if torch.cuda.is_available() else torch.device
('cpu')
CFG = {
    'SR':16_000,
    'BATCH_SIZE' : 8,
    'TOTAL_BATCH_SIZE': 42,
    'N_MFCC':32, # Melspectrogram 벡터를 추출할 개수
    'SEED':42,
    'EPOCHS' : 1,
    'LR' : 1e-4
}
train_df = pd.read_csv('./train.csv')
print(train_df.info())
print()
test_df = pd.read_csv('./test.csv')
print(test_df.info())

dataset = "./"
TRAIN_WAV = dataset + "train/"
TEST_WAV = dataset + "test/"

wav_file_dict = {"train_wav" : TRAIN_WAV,
                 "test_wav" : TEST_WAV,
                 "wav_sep" : WAV_TRAIN_LABEL_SEP
                 }

wav_file_locations = {}
for key, value in wav_file_dict.items():
    wav_file_locations[key] = glob.glob(value + "*.wav")

csv_file_dict = {"train_label_sep" : TRAIN_LABEL_SEP,
                 "wav_train_label_sep" : WAV_TRAIN_LABEL_SEP
                 }

csv_file_location = {}
for key, value in csv_file_dict.items():
    csv_file_location[key] = glob.glob(value + "*.csv")

def seed_everything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic=True
    torch.backends.cudnn.benchmark=True

seed_everything(CFG['SEED'])

MODEL_NAME = 'facebook/wav2vec2-base'
train_df, valid_df = train_test_split(train_df, test_size = 0.2, random_state=
CFG['SEED'])
train_df.reset_index(drop=True, inplace=True)
valid_df.reset_index(drop=True, inplace=True)

def speech_file_to_array_fn(df):

```



```

        feature = []
        for path in tqdm(df['path']):
            speech_array, _ = librosa.load(path, sr=CFG['SR'])
            feature.append(speech_array)
        return feature
    train_x = speech_file_to_array_fn(train_df)
    valid_x = speech_file_to_array_fn(test_df)
    processor = Wav2Vec2FeatureExtractor.from_pretrained(MODEL_NAME)
    audio_model = AutoModelForAudioClassification.from_pretrained(MODEL_NAME)

    train_dataset = CustomDataSet(train_x, train_df['label'], processor)
    valid_dataset = CustomDataSet(valid_x, valid_df['label'], processor)

    train_loader = create_data_loader(train_dataset, CFG['BATCH_SIZE'], False, collate_fn, 16)
    valid_loader = create_data_loader(valid_dataset, CFG['BATCH_SIZE'], False, collate_fn, 16)

    class BaseModel(torch.nn.Module):
        def __init__(self):
            super(BaseModel, self).__init__()
            self.model = audio_model
            self.model.classifier = nn.Identity()
            self.classifier = nn.Linear(256, 8)

        def forward(self, x):
            output = self.model(x)
            output = self.classifier(output.logits)
            return output

    model = BaseModel()

    optimizer = torch.optim.Adam(model.parameters(), lr=CFG['LR'])
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max', factor=0.5, patience=3, verbose=True)
    inter_model = train(model, train_loader, valid_loader, optimizer, scheduler)

```

▼ Codes (custom_dataset.py)

```

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from tqdm import tqdm
import librosa
from transformers import Wav2Vec2FeatureExtractor
import numpy as np

CFG = {
    'SR': 16_000,
    'BATCH_SIZE': 8,
    'TOTAL_BATCH_SIZE': 42,
    'N_MFCC': 32, # Melspectrogram 벡터를 추출할 개수

```

```

    'SEED': 42,
    'EPOCHS': 1,
    'LR': 1e-4
}

device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

class CustomDataSet(torch.utils.data.Dataset):
    def __init__(self, x, y, processor):
        self.x = x
        self.y = y
        self.processor = processor

    def __len__(self):
        return len(self.x)

    def __getitem__(self, idx):
        input_values = self.processor(self.x[idx], sampling_rate=CFG['SR'], return_tensors='pt',
                                     padding=True).input_values
        if self.y is not None:
            return input_values.squeeze(), self.y[idx]
        else:
            return input_values.squeeze()

def speech_file_to_array_fn(df):
    feature = []
    for path in tqdm(df['path']):
        speech_array, _ = librosa.load(path, sr=CFG['SR'])
        feature.append(speech_array)
    return feature

def collate_fn(batch):
    x, y = zip(*batch)
    x = pad_sequence([torch.tensor(xi) for xi in x], batch_first=True)
    y = pad_sequence([torch.tensor([yi]) for yi in y], batch_first=True) # Convert scalar targets to 1D tensors
    return x, y

def create_data_loader(dataset, batch_size, shuffle, collate_fn, num_workers=0):
    return DataLoader(dataset,
                      batch_size=batch_size,
                      shuffle=shuffle,
                      collate_fn=collate_fn,
                      num_workers=num_workers
                      )

def validation(model, valid_loader, criterion):
    model.eval()
    val_loss = []

    total, correct = 0, 0

```

```

test_loss = 0

with torch.no_grad():
    for x, y in tqdm(iter(valid_loader)):
        x = x.to(device)
        y = y.flatten().to(device)

        output = model(x)
        loss = criterion(output, y)

        val_loss.append(loss.item())

        test_loss += loss.item()
        _, predicted = torch.max(output, 1)
        total += y.size(0)
        correct += predicted.eq(y).cpu().sum()

accuracy = correct / total
avg_loss = np.mean(val_loss)

return avg_loss, accuracy

def train(model, train_loader, valid_loader, optimizer, scheduler):
    accumulation_step = int(CFG['TOTAL_BATCH_SIZE'] / CFG['BATCH_SIZE'])
    model.to(device)
    criterion = nn.CrossEntropyLoss().to(device)
    best_model = None
    best_acc = 0

    for epoch in range(1, CFG['EPOCHS'] + 1):
        train_loss = []
        model.train()
        for i, (x, y) in enumerate(tqdm(train_loader)):
            x = x.to(device)
            y = y.flatten().to(device)

            optimizer.zero_grad()
            output = model(x)
            loss = criterion(output, y)
            loss.backward()

            if (i + 1) % accumulation_step == 0:
                optimizer.step()
                optimizer.zero_grad()

            train_loss.append(loss.item())

        avg_loss = np.mean(train_loss)
        valid_loss, valid_acc = validation(model, valid_loader, criterion)

        if scheduler is not None:
            scheduler.step(valid_acc)

        if valid_acc > best_acc:
            best_acc = valid_acc

```

```

        best_model = model

        print(f'epoch:[{epoch}] train loss:[{avg_loss:.5f}] valid_loss:[{valid_
_loss:.5f}] valid_acc:[{valid_acc:.5f}]')

        print(f'best_acc:[{best_acc:.5f}]')

        return (best_model)

```

▼ Error

I tried to search for the solution and modify the code the whole night, but couldn't get the right answer.

```

/Desktop/Competitions/DACON_Sound_Emotion/open/custom_dataset.py:48: UserWarni
ng: To copy construct from a tensor, it is recommended to use sourceTensor.clo
ne().detach() or sourceTensor.clone().detach().requires_grad_(True), rather th
an torch.tensor(sourceTensor).
    x = pad_sequence([torch.tensor(xi) for xi in x], batch_first=True)
53%|██████████████████████████████████████████████████████████████████████████| 125/236 [18:36<16:31, 8.93s/i
t]
-----
KeyError                                Traceback (most recent call last)
Cell In[17], line 5
      3 optimizer = torch.optim.Adam(model.parameters(), lr=CFG['LR'])
      4 scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode
='max', factor=0.5, patience=3, verbose=True)
----> 5 inter_model = train(model, train_loader, valid_loader, optimizer, sche
duler)

File ~/Desktop/Competitions/DACON_Sound_Emotion/open/custom_dataset.py:116, in
train(model, train_loader, valid_loader, optimizer, scheduler)
    113     train_loss.append(loss.item())
    115     avg_loss = np.mean(train_loss)
--> 116     valid_loss, valid_acc = validation(model, valid_loader, criterion)
    118     if scheduler is not None:
    119         scheduler.step(valid_acc)

File ~/Desktop/Competitions/DACON_Sound_Emotion/open/custom_dataset.py:70, in
validation(model, valid_loader, criterion)
    67     test_loss = 0
    69     with torch.no_grad():
--> 70         for x, y in tqdm(iter(valid_loader)):
    71             x = x.to(device)
    72             y = y.flatten().to(device)

File /opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/tqdm/s
td.py:1178, in tqdm.__iter__(self)
    1175     time = self._time
    1177     try:
-> 1178         for obj in iterable:
    1179             yield obj
    1180             # Update and possibly print the progressbar.

```

```

1181         # Note: does not call self.update(1) for speed optimisation.

File /opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/torch/
utils/data/dataloader.py:633, in _BaseDataLoaderIter.__next__(self)
    630 if self._sampler_iter is None:
    631     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    632     self._reset() # type: ignore[call-arg]
--> 633 data = self._next_data()
    634 self._num_yielded += 1
    635 if self._dataset_kind == _DatasetKind.Iterable and \
    636     self._IterableDataset_len_called is not None and \
    637     self._num_yielded > self._IterableDataset_len_called:

File /opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/torch/
utils/data/dataloader.py:1345, in _MultiProcessingDataLoaderIter._next_data(se
lf)
    1343 else:
    1344     del self._task_info[idx]
-> 1345     return self._process_data(data)

File /opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/torch/
utils/data/dataloader.py:1371, in _MultiProcessingDataLoaderIter._process_data
(self, data)
    1369 self._try_put_index()
    1370 if isinstance(data, ExceptionWrapper):
-> 1371     data.reraise()
    1372 return data

File /opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/torch/
_utils.py:644, in ExceptionWrapper.reraise(self)
    640 except TypeError:
    641     # If the exception takes multiple arguments, don't try to
    642     # instantiate since we don't know how to
    643     raise RuntimeError(msg) from None
--> 644 raise exception

KeyError: Caught KeyError in DataLoader worker process 13.
Original Traceback (most recent call last):
  File "/opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/pa
ndas/core/indexes/range.py", line 391, in get_loc
    return self._range.index(new_key)
ValueError: 1001 is not in range

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/to
rch/utils/data/_utils/worker.py", line 308, in _worker_loop
    data = fetcher.fetch(index)
  File "/opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/to
rch/utils/data/_utils/fetch.py", line 51, in fetch
    data = [self.dataset[idx] for idx in possibly_batched_index]
  File "/opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/to
rch/utils/data/_utils/fetch.py", line 51, in <listcomp>
    data = [self.dataset[idx] for idx in possibly_batched_index]
  File "/Users/b05/Desktop/Competitions/DACON_Sound_Emotion/open/custom_datase

```

```
t.py", line 35, in __getitem__
    return input_values.squeeze(), self.y[idx]
File "/opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/pandas/core/series.py", line 981, in __getitem__
    return self._get_value(key)
File "/opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/pandas/core/series.py", line 1089, in _get_value
    loc = self.index.get_loc(label)
File "/opt/homebrew/Caskroom/miniconda/base/lib/python3.10/site-packages/pandas/core/indexes/range.py", line 393, in get_loc
    raise KeyError(key) from err
KeyError: 1001
```

I tried to catch the error by inserting print to every single line in the custom_dataset.py, but it didn't help.

▼ **Codes (custom_dataset.py with print)**

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from tqdm import tqdm
import librosa
from transformers import Wav2Vec2FeatureExtractor
import numpy as np

CFG = {
    'SR': 16_000,
    'BATCH_SIZE': 8,
    'TOTAL_BATCH_SIZE': 42,
    'N_MFCC': 32, # Melspectrogram 벡터를 추출할 개수
    'SEED': 42,
    'EPOCHS': 1,
    'LR': 1e-4
}

device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

def speech_file_to_array_fn(df):
    feature = []
    for path in tqdm(df['path']):
        speech_array, _ = librosa.load(path, sr=CFG['SR'])
        print(f"sfaf : speech_array, _ = librosa.load(path, sr=CFG['SR']), speech_array = {speech_array}")
        feature.append(speech_array)
        print("sfaf : feature.append(speech_array)")
    return feature

class CustomDataSet(torch.utils.data.Dataset):
```

```

def __init__(self, x, y, processor):
    self.x = x
    self.y = y
    self.processor = processor

def __len__(self):
    return len(self.x)

def __getitem__(self, idx):
    input_values = self.processor(self.x[idx], sampling_rate=CFG['SR'], re
turn_tensors='pt', padding=True).input_values
    if self.y is not None:
        print("CustomDataSet : __getitem__ : if self.y is not None:")
        return input_values.squeeze(), self.y[idx]
    else:
        print("CustomDataSet : __getitem__ : else:")
        return input_values.squeeze()

def collate_fn(batch):
    x, y = zip(*batch)
    print(f"collate_fn : x, y = zip(*batch), x = {x}, y = {y}")
    x = pad_sequence([torch.tensor(xi) for xi in x], batch_first=True)
    print("collate_fn : x = pad_sequence([torch.tensor(xi) for xi in x], batch
_first=True)")
    print(f"x = {x}")
    y = pad_sequence([torch.tensor([yi]) for yi in y], batch_first=True) # Co
nvert scalar targets to 1D tensors
    print("collate_fn : y = pad_sequence([torch.tensor([yi]) for yi in y], bat
ch_first=True)")
    print(f"y = {y}")
    return x, y

def create_data_loader(dataset, batch_size, shuffle, collate_fn, num_workers=
0):
    print("create_data_loader : Data Loaded")
    return DataLoader(dataset, batch_size=batch_size, shuffle=shuffle, collate
_fn=collate_fn, num_workers=num_workers)

def validation(model, valid_loader, criterion):
    model.eval()
    val_loss = []

    total, correct = 0, 0
    test_loss = 0

    with torch.no_grad():
        for x, y in tqdm(iter(valid_loader)):
            if y != None:
                print(f"validation : x = {x}")
                x = x.to(device)
                print(f"validation : x.to(device) = {x}")
                print(f"validation : y = {y}")

```

```

        y = y.flatten().to(device)
        print(f"validation : y.flatten().to(device) = {y}")
        output = model(x)
        print(f"validation : output = model(x) = {output}")
        loss = criterion(output, y)
        print(f"validation : loss = criterion(output, y) = {loss}")

        val_loss.append(loss.item())
        print("validation : val_loss.append(loss.item())")

        test_loss += loss.item()
        print("validation : test_loss += loss.item()")
        _, predicted = torch.max(output, 1)
        print("validation : _, predicted = torch.max(output, 1)")
        total += y.size(0)
        print("validation : total += y.size(0)")
        correct += predicted.eq(y).cpu().sum()
        print("validation : correct += predicted.eq(y).cpu().sum()")
    else:
        pass

    accuracy = correct / total
    print("validation : accuracy = correct / total")
    avg_loss = np.mean(val_loss)
    print("validation : avg_loss = np.mean(val_loss)")

    return avg_loss, accuracy

def train(model, train_loader, valid_loader, optimizer, scheduler):
    accumulation_step = int(CFG['TOTAL_BATCH_SIZE'] / CFG['BATCH_SIZE'])
    model.to(device)
    criterion = nn.CrossEntropyLoss().to(device)
    best_model = None
    best_acc = 0
    epoch_count = 1
    for epoch in range(1, CFG['EPOCHS']+1):
        print(f'train : epoch count = {epoch_count} th count')
        train_loss = []
        model.train()
        train_loader_count = 1
        for i, (x, y) in enumerate(tqdm(train_loader)):
            print(f"train : epoch_count = {epoch_count}, train_loader_count = {train_loader_count}")
            print(f"validation : x = {x}")
            x = x.to(device)
            print(f"train : x.to(device) = {x}")
            print(f"train : y = {y}")
            y = y.flatten().to(device)
            print(f"train : y.flatten().to(device) = {y}")
            print("train : x, y moved to device")
            optimizer.zero_grad()
            print("train : optimizer.zero_grad()")
            output = model(x)
            print(f"train : output = model(x) = {output}")
            loss = criterion(output, y)

```



```

print(f"train : loss = criterion(output, y) = {loss}")
loss.backward()
print("train : loss.backward()")

if (i + 1) % accumulation_step == 0:
    print("train : if (i + 1) % accumulation_step == 0:")
    optimizer.step()
    print("train : optimizer.step()")
    optimizer.zero_grad()
    print("optimizer.zero_grad()")
    train_loader_count += 1
    train_loss.append(loss.item())
    print("train : train_loss.append(loss.item())")

avg_loss = np.mean(train_loss)
print(f"train : avg_loss = np.mean(train_loss) = {avg_loss}")
valid_loss, valid_acc = validation(model, valid_loader, criterion)
print(f"train : valid_loss, valid_acc = validation(model, valid_loader,
criterion)")

if scheduler is not None:
    scheduler.step(valid_acc)
    print(f"train : if scheduler is not None: scheduler.step(valid_acc)")

if valid_acc > best_acc:
    print(f"train : if valid_acc > best_acc:")
    best_acc = valid_acc
    print(f"train : best_acc = valid_acc = {best_acc}")
    best_model = model
    print(f"train : best_model = model = {best_model}")
    epoch_count += 1
    print(f'train : epoch:[{epoch}] train loss:[{avg_loss:.5f}] valid_loss:[{valid_loss:.5f}] valid_acc:[{valid_acc:.5f}]')

print(f'train : best_acc:{best_acc:.5f}')

return best_model

```

As I was lack of the `torch` knowledge, I decided to pause with the Transformation model, and to try with a different, but better understandable model.

▼ Machine Learning Models

By using the preprocess data I made before, I am trying various Machine Learning models.

- **Random Forest Classifier (Success)**

Below is the code, and I got 160th place with a score of 0.37411

▼ Codes

```
rf = RandomForestClassifier(n_estimators=50,
                           max_depth=4,
                           min_samples_split=2,
                           max_features=0.85,
                           n_jobs=-1,
                           random_state=CFG['SEED'])
X_train, X_val, y_train, y_val = train_test_split(train_x, train_y, test_
size=0.2, random_state=CFG['SEED'])
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)

rf.fit(train_x, train_y)
print("-- Random Forest --")
print("Train ACC : %.3f" % accuracy_score(y_train, rf.predict(X_train)))
print("Val ACC : %.3f" % accuracy_score(y_val, rf.predict(X_val)))

-----
# Output

-- Random Forest --
Train ACC : 0.423
Val ACC : 0.431

-----

X_test = pd.get_dummies(data=test_x)
pred = rf.predict(X_test)
pred

submission = pd.read_csv(dataset + 'sample_submission.csv')
submission['label'] = pred
submission.to_csv(dataset + "baseline_submission.csv", index=False)
```

Now I will try with other parameters and other models as well.

Then, I decided to use all 4 models I learned during the course.

As I felt modifying the hyperparameters every single time was too much of work, I added counter to modify all values.

As It was so hard to record all the data this `for` loop tried, and to prevent unexpected kernel shutdown, I added a code to automatically save the data to csv file. (Which later created over 10,000 files)

▼ Codes

```

count = 1

for depth in range(1, 6):
    for estimator in range(1, 1000):
        for features in range(1,100):
            rf = RandomForestClassifier(max_depth=depth,
                                       n_estimators=estimator,
                                       min_samples_split=2,
                                       max_features=features/100,
                                       n_jobs=-1,
                                       random_state=CFG['SEED']
                                       )

            dt = DecisionTreeClassifier(max_depth=depth,
                                       min_samples_split=2,
                                       max_features=features/100,
                                       random_state=CFG['SEED']
                                       )

            xgboost = XGBClassifier(max_depth=depth,
                                    n_estimators=estimator,
                                    grow_policy='depthwise',
                                    n_jobs=-1,
                                    random_state=CFG['SEED'],
                                    tree_method='auto'
                                    )

            # lgbm = LGBMClassifier(max_depth=depth,
            #                       n_estimators=estimator,
            #                       max_features=features/100,
            #                       n_jobs=-1,
            #                       num_leaves=2^(depth),
            #                       random_state=CFG['SEED']
            #                       )

            rf.fit(X_train, y_train)
            dt.fit(X_train, y_train)
            scaler = StandardScaler()
            x_trainScaled = scaler.fit_transform(X_train)
            xgboost.fit(x_trainScaled, y_train)
            # lgbm.fit(X_train, y_train)
            Results = pd.DataFrame([["%.3f" % accuracy_score(y_train, rf.predict(X_train)), "%.3f" % accuracy_score(y_val, rf.predict(X_val))],
                                   [("%.3f" % accuracy_score(y_train, dt.predict(X_train)), "%.3f" % accuracy_score(y_val, dt.predict(X_val))),
                                   [("%.3f" % accuracy_score(y_train, xgboost.predict(X_train)), "%.3f" % accuracy_score(y_val, xgboost.predict(X_val)))],
                                   [("%.3f" % accuracy_score(y_train, lgbm.predict(X_train)), "%.3f" % accuracy_score(y_val, lgbm.predict(X_val)))],
                                   index = ['Random Forest', "Decision Tree", "XG Boost"]

            )

            get_parameters = pd.DataFrame([rf.get_params(), dt.get_params(), xgboost.get_params()],
                                           index = ['Random Forest', "Decision Tree", "XG Boost"])

```

```
Results = Results.astype(float)
ComResults = pd.concat([Results, get_parameters], axis=1)

ComResults.to_csv(dataset + f"Results/Results{count}.csv")
count += 1
print(count)
```

After running the loops, I tried to find the best result.

Based on my understandings, the best result is the result that has the highest

`accuracy_score` for both `train` and `validation` sets.

Therefore, I read all files and found only the sets that had the highest

`accuracy_score` every round.

▼ Codes

```
result_files = glob.glob(dataset + "Results/*.csv")

max_temp = []
for file in result_files:
    filename = file.split(".")[1]
    filename = filename.split("/")[2]
    df = pd.read_csv(file)
    train_acc = pd.DataFrame(df["0"])
    val_acc = pd.DataFrame(df["1"])
    train_max = train_acc.idxmax()
    val_max = val_acc.idxmax()
    if train_max[-1] == val_max[-1]:
        max_temp.append([filename, train_acc.iloc[train_max[-1]][-1], val_acc.
            iloc[val_max[-1]][-1]])

max_temp_df = pd.DataFrame(max_temp)
max_temp_df = max_temp_df.rename(columns={0:'filename', 1:'train_acc', 2:'val_
acc'})
```

With `max_temp_df`, a set of the best scores of each round, I found the highest

`accuracy_score` among them again.

It was to find the top-of-top

`accuracy_score` to submit.

As the file was the combination of 4 different methods, I had to separate the `fit`,

`test` processes per method.

(I later found out that I was not fully understanding

`lgbm`, I had to mute the loop for `lgbm`.)

▼ Codes

```
filename = pd.DataFrame(max_temp_df['filename'])
train_acc = pd.DataFrame(max_temp_df['train_acc'])
val_acc = pd.DataFrame(max_temp_df['val_acc'])
train_max = train_acc.idxmax()
val_max = val_acc.idxmax()
if train_max[-1] == val_max[-1]:
    print(filename.iloc[train_max], train_acc.iloc[train_max], val_acc.iloc[val_max])
    val_max_filepath = dataset + "Results/" + filename.iloc[val_max].iat[0,0] + ".csv"
    maxfile = pd.read_csv(filepath)
    max_max_train = pd.DataFrame(maxfile["0"])
    max_max_val = pd.DataFrame(maxfile["1"])
    train_max_max = max_max_train.idxmax()[-1]
    val_max_max = max_max_val.idxmax()[-1]
    if train_max_max == 0: # Random Forest
        rf = RandomForestClassifier(max_depth=maxfile["max_depth"].iat[0],
                                   n_estimators=int(maxfile["n_estimators"].iat[0]),
                                   min_samples_split=2,
                                   max_features=maxfile["max_features"].iat[0],
                                   n_jobs=-1,
                                   random_state=CFG['SEED'])
        rf.fit(X_train, y_train)
        X_test = pd.get_dummies(data=test_x)
        rf_pred = rf.predict(X_test)
        rf_submission = pd.read_csv(dataset + 'sample_submission.csv')
        rf_submission['label'] = rf_pred
        rf_submission.to_csv(dataset + "rf_submission.csv", index=False)
    elif train_max_max == 1: # Decision Tree
        dt = DecisionTreeClassifier(max_depth=maxfile["max_depth"].iat[1],
                                   min_samples_split=2,
                                   max_features=maxfile["max_features"].iat[1],
                                   random_state=CFG['SEED'])
        dt.fit(X_train, y_train)
        X_test = pd.get_dummies(data=test_x)
        dt_pred = dt.predict(X_test)
        dt_submission = pd.read_csv(dataset + 'sample_submission.csv')
        dt_submission['label'] = dt_pred
        dt_submission.to_csv(dataset + "dt_submission.csv", index=False)
    elif train_max_max == 2: # XG Boost
        xgboost = XGBClassifier(max_depth=maxfile['max_depth'].iat[2],
                                n_estimators=int(maxfile["n_estimators"].iat[2]),
                                grow_policy='depthwise',
                                n_jobs=-1,
                                random_state=CFG['SEED'],
                                tree_method='auto')
```

```

    )
    xgboost.fit(X_train, y_train)
    X_test = pd.get_dummies(data=test_x)
    xgboost_pred = xgboost.predict(X_test)
    xgboost_submission = pd.read_csv(dataset + 'sample_submission.csv')
    xgboost_submission['label'] = dt_pred
    xgboost_submission.to_csv(dataset + "xgboost_submission.csv", index=False)
else:
    print("This is LGBM")

```

To be precautionous for the case where there is no top-of-top `accuracy_score` in this DataFrame, I added codes to select the best resulted `validation_accuracy_score`. I could also have added another elif or else statement to find `train_accuracy_score`, but I had no time to work on it as the deadline was only 30 minutes away.

▼ Codes

```

else:
    print(f"filename.iloc[val_max] = {filename.iloc[val_max]}, \n \n val_acc.i
loc[val_max] = {val_acc.iloc[val_max]}")
    print()
    print()
    # print(f"filename.iloc[val_max][1] = {filename.iloc[val_max][1]}")
    print(f"filename.iloc[val_max].info() = {filename.iloc[val_max].info()}")
    val_max_filepath = dataset + "Results/" + filename.iloc[val_max].iat[0,0]
    + ".csv"
    print(val_max_filepath)
    val_maxfile = pd.read_csv(val_max_filepath)
    val_max_train = pd.DataFrame(val_maxfile["0"])
    val_max_val = pd.DataFrame(val_maxfile["1"])
    train_max_max = val_max_train.idxmax()[-1]
    val_max_max = val_max_val.idxmax()[-1]
    if train_max_max == 0: # Random Forest
        rf = RandomForestClassifier(max_depth=val_maxfile["max_depth"].iat[0],
                                   n_estimators=int(val_maxfile["n_estimator
s"].iat[0]),
                                   min_samples_split=2,
                                   max_features=val_maxfile["max_features"].i
at[0],
                                   n_jobs=-1,
                                   random_state=CFG['SEED'])
        rf.fit(X_train, y_train)
        X_test = pd.get_dummies(data=test_x)
        rf_pred = rf.predict(X_test)
        rf_submission = pd.read_csv(dataset + 'sample_submission.csv')
        rf_submission['label'] = rf_pred
        rf_submission.to_csv(dataset + "rf_max_submission.csv", index=False)
    elif train_max_max == 1: # Decision Tree
        dt = DecisionTreeClassifier(max_depth=val_maxfile["max_depth"].iat[1],

```

```

min_samples_split=2,
max_features=val_maxfile["max_features"].iat[1],

at[1],

random_state=CFG['SEED']
)
dt.fit(X_train, y_train)
X_test = pd.get_dummies(data=test_x)
dt_pred = dt.predict(X_test)
dt_submission = pd.read_csv(dataset + 'sample_submission.csv')
dt_submission['label'] = dt_pred
dt_submission.to_csv(dataset + "dt_max_submission.csv", index=False)
elif train_max_max == 2: # XG Boost
    xgboost = XGBClassifier(max_depth=val_maxfile['max_depth'].iat[2],
                            n_estimators=int(val_maxfile["n_estimators"].iat[2]),

at[2]),

grow_policy='depthwise',
n_jobs=-1,
random_state=CFG['SEED'],
tree_method='auto'
)
xgboost.fit(X_train, y_train)
X_test = pd.get_dummies(data=test_x)
xgboost_pred = xgboost.predict(X_test)
xgboost_submission = pd.read_csv(dataset + 'sample_submission.csv')
xgboost_submission['label'] = dt_pred
xgboost_submission.to_csv(dataset + "xgboost_max_submission.csv", index=False)
else:
    print("This is LGBM")

```

Moreover, as I was not sure if selecting the top-of-top `accuracy_score` would increase my competition score, I have also added a code to find the bottom-of-bottom `accuracy_score`.

▼ Codes

```

min_temp = []
for file in result_files:
    filename = file.split(".")[1]
    filename = filename.split("/")[2]
    df = pd.read_csv(file)
    train_acc = pd.DataFrame(df["0"])
    val_acc = pd.DataFrame(df["1"])
    train_min = train_acc.idxmin()
    val_min = val_acc.idxmin()
    if train_min[-1] == val_min[-1]:
        min_temp.append([filename, train_acc.iloc[train_min[-1]][-1], val_acc.
iloc[val_min[-1]][-1]])

min_temp_df = pd.DataFrame(min_temp)
min_temp_df = min_temp_df.rename(columns={0:'filename', 1:"train_acc", 2:"val_
acc"})

```

```

filename = pd.DataFrame(min_temp_df['filename'])
train_acc = pd.DataFrame(min_temp_df['train_acc'])
val_acc = pd.DataFrame(min_temp_df['val_acc'])
train_min = train_acc.idxmin()
val_min = val_acc.idxmin()
if train_min[-1] == val_min[-1]:
#     print(filename.iloc[train_min], train_acc.iloc[train_min], val_acc.iloc
[val_min])
    val_min_filepath = dataset + "Results/" + filename.iloc[val_min].iat[0,0]
+ ".csv"
    minfile = pd.read_csv(filepath)
    min_min_train = pd.DataFrame(minfile["0"])
    min_min_val = pd.DataFrame(minfile["1"])
    train_min_min = min_min_train.idxmin()[-1]
    val_min_min = min_min_val.idxmin()[-1]
    if train_min_min == 0: # Random Forest
        rf = RandomForestClassifier(max_depth=minfile["max_depth"].iat[0],
n_estimators=int(minfile["n_estimators"].i
at[0]),
                                min_samples_split=2,
                                max_features=minfile["max_features"].iat
[0],
                                n_jobs=-1,
                                random_state=CFG['SEED']
                                )
        rf.fit(X_train, y_train)
        X_test = pd.get_dummies(data=test_x)
        rf_pred = rf.predict(X_test)
        rf_submission = pd.read_csv(dataset + 'sample_submission.csv')
        rf_submission['label'] = rf_pred
        rf_submission.to_csv(dataset + "rf_min_submission.csv", index=False)
    elif train_min_min == 1: # Decision Tree
        dt = DecisionTreeClassifier(max_depth=minfile["max_depth"].iat[1],
min_samples_split=2,
max_features=minfile["max_features"].iat
[1],
                                random_state=CFG['SEED']
                                )
        dt.fit(X_train, y_train)
        X_test = pd.get_dummies(data=test_x)
        dt_pred = dt.predict(X_test)
        dt_submission = pd.read_csv(dataset + 'sample_submission.csv')
        dt_submission['label'] = dt_pred
        dt_submission.to_csv(dataset + "dt_min_submission.csv", index=False)
    elif train_min_min == 2: # XG Boost
        xgboost = XGBClassifier(max_depth=minfile['max_depth'].iat[2],
n_estimators=int(minfile["n_estimators"].iat
[2]),
                                grow_policy='depthwise',
                                n_jobs=-1,
                                random_state=CFG['SEED'],
                                tree_method='auto'
                                )
        xgboost.fit(X_train, y_train)
        X_test = pd.get_dummies(data=test_x)
        xgboost_pred = xgboost.predict(X_test)

```



```

        xgboost_submission = pd.read_csv(dataset + 'sample_submission.csv')
        xgboost_submission['label'] = xgboost_pred
        xgboost_submission.to_csv(dataset + "xgboost_min_submission.csv", index=False)
    else:
        print("This is LGBM")
    else:
        print(filename.iloc[val_min], val_acc.iloc[val_min])
        val_min_filepath = dataset + "Results/" + filename.iloc[val_min].iat[0,0] + ".csv"
        val_minfile = pd.read_csv(val_min_filepath)
        val_min_train = pd.DataFrame(val_minfile["0"])
        val_min_val = pd.DataFrame(val_minfile["1"])
        train_min_min = val_min_train.idxmin()[-1]
        val_min_min = val_min_val.idxmin()[-1]
        if train_min_min == 0: # Random Forest
            rf = RandomForestClassifier(max_depth=val_minfile["max_depth"].iat[0],
                                      n_estimators=int(val_minfile["n_estimators"].iat[0]),
                                      min_samples_split=2,
                                      max_features=val_minfile["max_features"].iat[0],
                                      n_jobs=-1,
                                      random_state=CFG['SEED'])
            rf.fit(X_train, y_train)
            X_test = pd.get_dummies(data=test_x)
            rf_pred = rf.predict(X_test)
            rf_submission = pd.read_csv(dataset + 'sample_submission.csv')
            rf_submission['label'] = rf_pred
            rf_submission.to_csv(dataset + "rf_min_submission.csv", index=False)
        elif train_min_min == 1: # Decision Tree
            dt = DecisionTreeClassifier(max_depth=val_minfile["max_depth"].iat[1],
                                      min_samples_split=2,
                                      max_features=val_minfile["max_features"].iat[1],
                                      random_state=CFG['SEED'])
            dt.fit(X_train, y_train)
            X_test = pd.get_dummies(data=test_x)
            dt_pred = dt.predict(X_test)
            dt_submission = pd.read_csv(dataset + 'sample_submission.csv')
            dt_submission['label'] = dt_pred
            dt_submission.to_csv(dataset + "dt_min_submission.csv", index=False)
        elif train_min_min == 2: # XG Boost
            xgboost = XGBClassifier(max_depth=val_minfile['max_depth'].iat[2],
                                   n_estimators=int(val_minfile["n_estimators"].iat[2]),
                                   grow_policy='depthwise',
                                   n_jobs=-1,
                                   random_state=CFG['SEED'],
                                   tree_method='auto')
            xgboost.fit(X_train, y_train)
            X_test = pd.get_dummies(data=test_x)
            xgboost_pred = xgboost.predict(X_test)

```

```

xgboost_submission = pd.read_csv(dataset + 'sample_submission.csv')
xgboost_submission['label'] = xgboost_pred
xgboost_submission.to_csv(dataset + "xgboost_min_submission.csv", index=False)
else:
    print("This is LGBM")








```

For the final submission, I have used two sets of parameter that resulted top-of-top `accuracy_score` and one set of parameter that resulted bottom-of-bottom `accuracy_score`.








Which at last gave me these scores below.

856016	lgbm_max_submission.csv 7회제출 - LGBM 활용, librosa.feature.mfcc전처리 활용, val acc 최소 score 활용 edit	2023-06-05 03:08:18	0.3882978723	✓
856021	lgbm_max_submission (1).csv 8회제출 - LGBM 활용, librosa.feature.mfcc전처리 활용, trian acc 최대 score 활용 edit	2023-06-05 03:17:30	0.3865248227	✓
855981	xgboost_min_submission.csv 6회 제출 - XG boost 활용, librosa.feature.mfcc전처리 활용, trian val acc 최소 score 활용 edit	2023-06-05 01:53:19	0.3741134752	□

Below is my final Public and Private score.

PUBLIC PRIVATE RANKING CHART						순위기준
● WINNER ● 1% ● 4% ● 10%						전체 랭킹 >
#	팀	팀 멤버	점수	제출수	등록일	
181	swibeijson		0.38829	8	6일 전	
1	파이썬초보만		0.87943	5	7일 전	
2	Ldoun		0.84042	15	6일 전	
3	hangjoo		0.83333	15	12일 전	
4	중요한건책이지않는마음		0.82978	8	7일 전	
5	램이부족해		0.82801	13	6일 전	
6	킹니프사		0.82624	35	9일 전	

Public Score

PUBLIC PRIVATE RANKING CHART			순위기준		
● WINNER ● 1% ● 4% ● 10%			전체 랭킹 >		
#	팀	팀 멤버	최종점수	제출수	등록일
173	swibeljason		0.41457	8	6일 전
1	파이썬초보만		0.87851	5	7일 전
2	Ldoun		0.82915	15	7일 전
3	중요한건겅이지않는마음		0.81852	8	8일 전
4	hangjoo		0.81776	15	12일 전
5	oriko		0.80789	19	20일 전
6	램이부족해		0.80561	13	6일 전

Private Score

I couldn't even made it to top 10%.

However, this was my first time actually completed not only the preprocess process, but the Machine Learning Process as well.

I already feel that my skills are improved again, and I feel I will do better next time when I join a new competition.