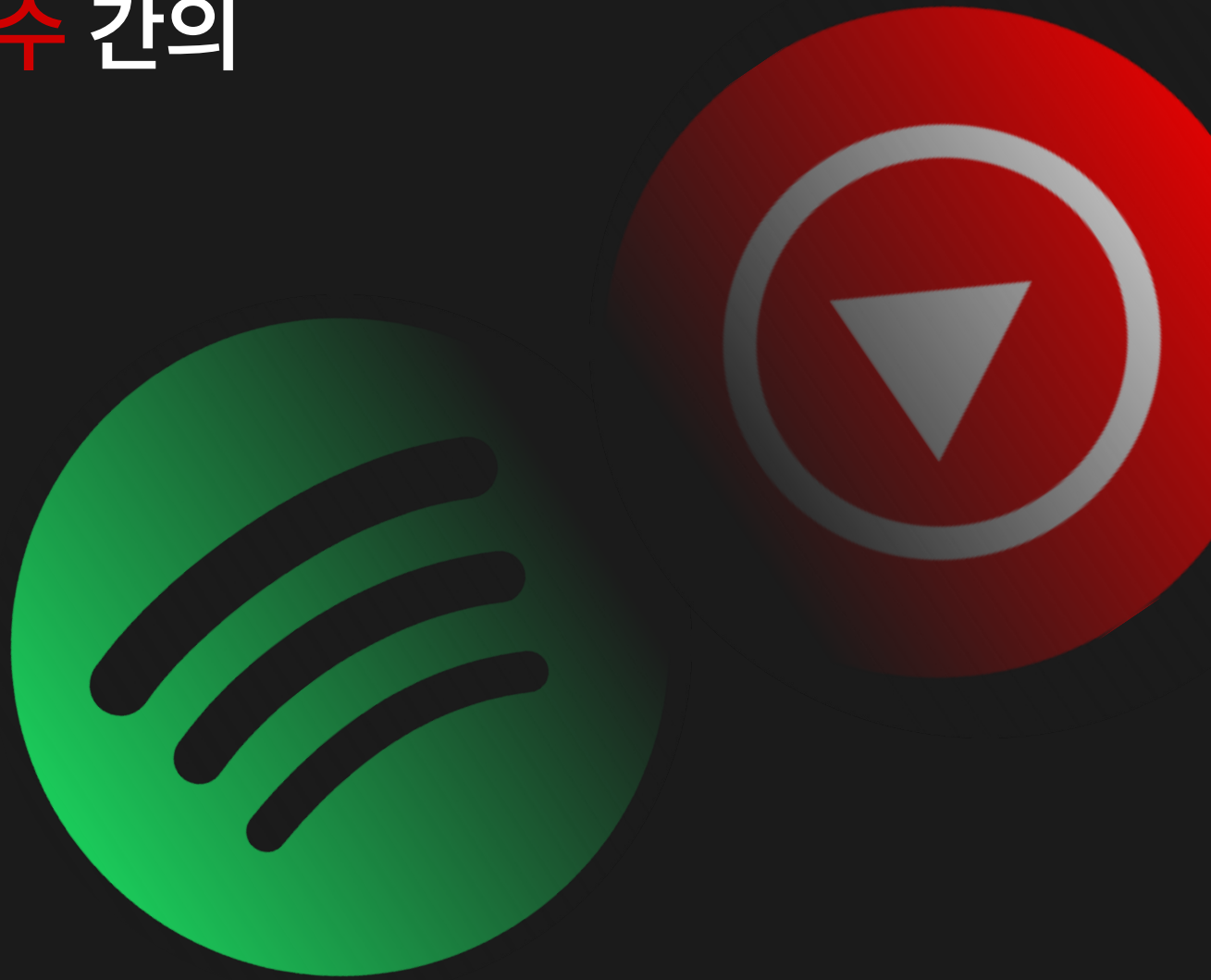
산업수학 기말 발표

스포티파이 DB – 유튜브 조회수 간의 상관관계 및 영향요인



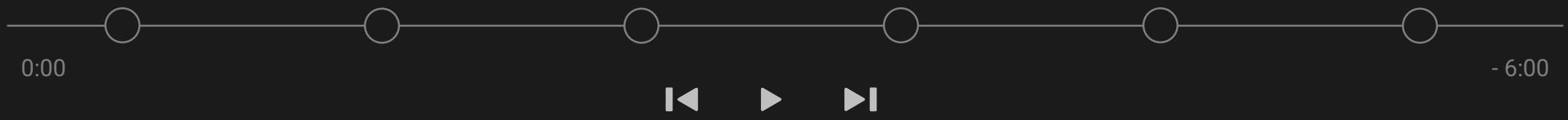
팀명	써리원 (팀원명 우측의 비율은 참여율을 나타낸 것임)		
20190311	**손유림 (25%)**	20211245	**설다현 (25%)**
20202076	**황대현 (25%)**	20221062	**온재현 (25%)**

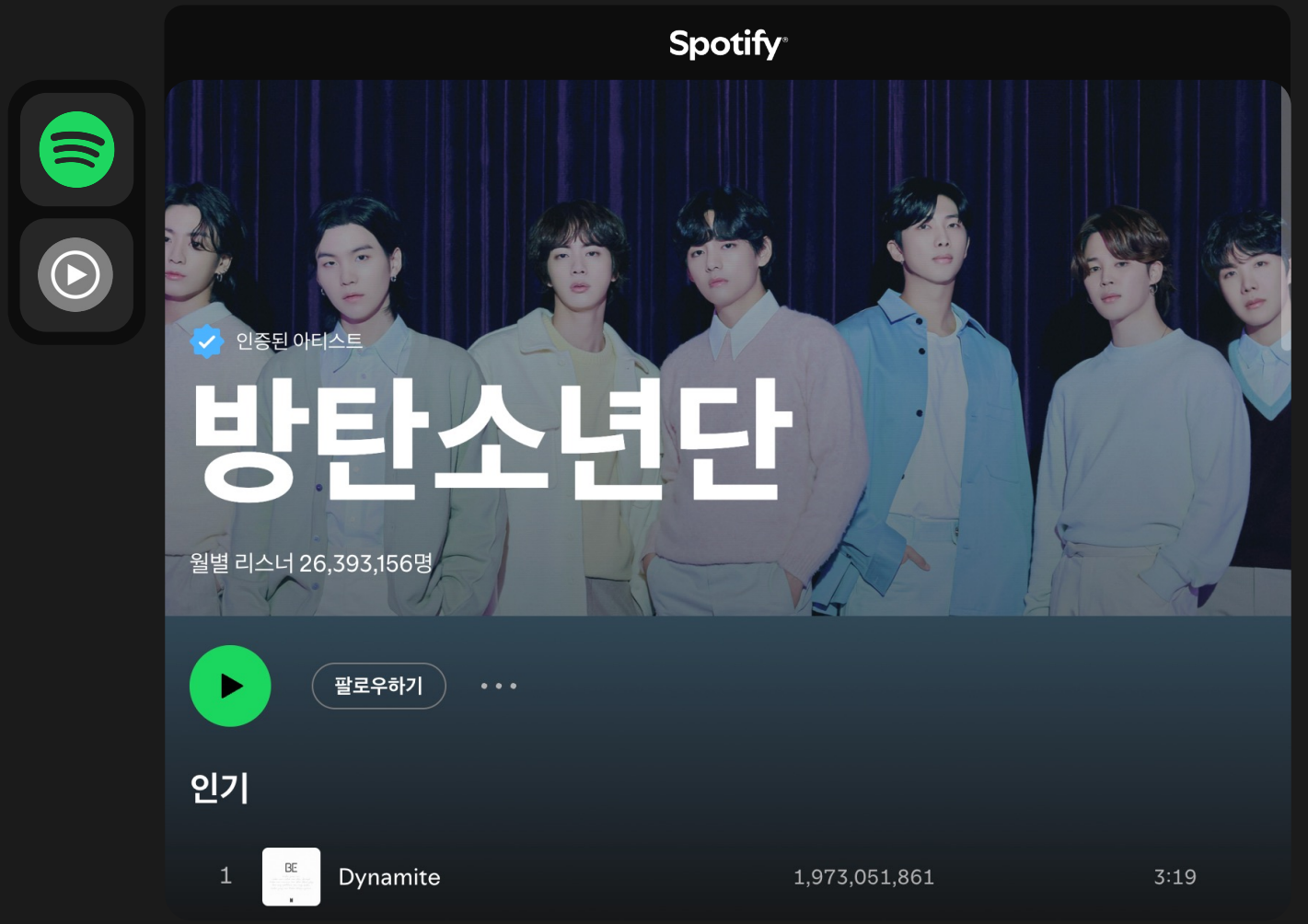
Contents ↘

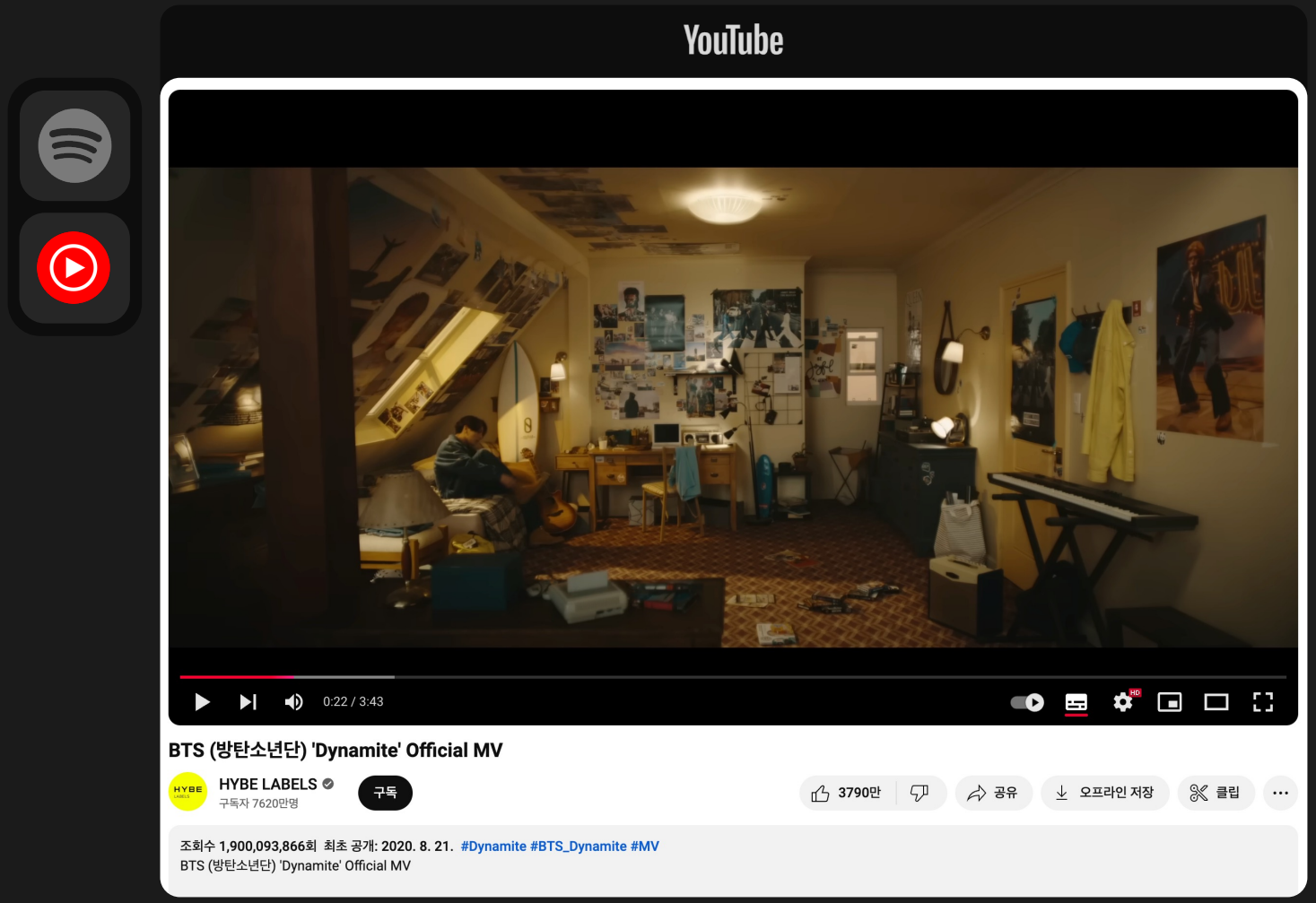
00

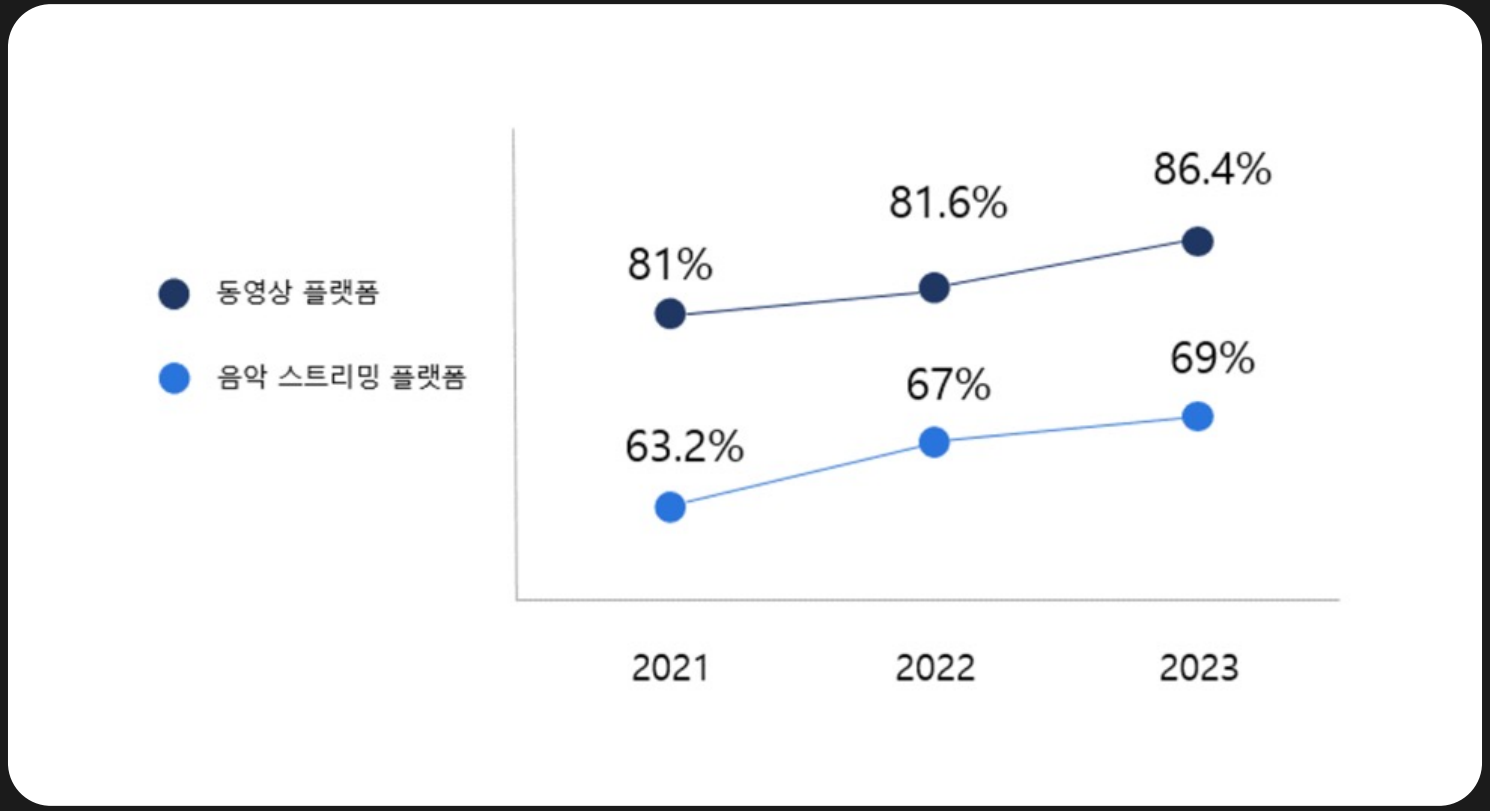
Introduction

서론




Spotify
인증된 아티스트
방탄소년단
월별 리스너 26,393,156명
팔로우하기
인기
1
Dynamite
1,973,051,861
3:19


YouTube
0:22 / 3:43
BTS (방탄소년단) 'Dynamite' Official MV
HYBE LABELS
구독자 7620만명
구독
3790만
공유
오프라인 저장
클립
조회수 1,900,093,866회 최초 공개: 2020. 8. 21. #Dynamite #BTS_Dynamite #MV
BTS (방탄소년단) 'Dynamite' Official MV



<한국콘텐츠진흥원, “2023 음악 이용자 실태조사”, 2023.11 (한국저작권위원회 재구성)>

주목할 점

Point 01

두 회사가 명확히
다른 서비스를 제공

Point 02

어떤 음악적 요소가
대중적 관심을 끄는가?

스포티파이 DB – 유튜브 조회수 간의 상관관계 및 영향요인

모델 구축

데이터셋을 바탕으로 여러 머신러닝 및 딥러닝 기법들을
시험하여 음악의 성공을 예측할 수 있는 모델 구축

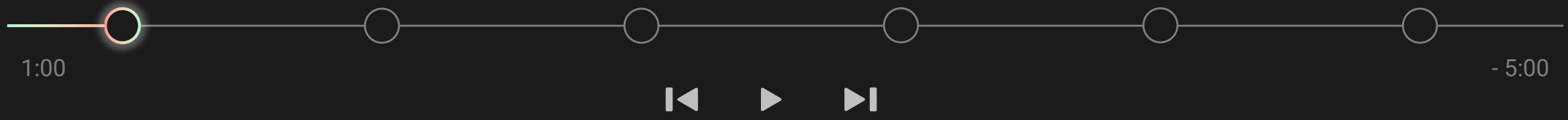
통찰 제공

음악의 각 요소에 따른
대중의 소비패턴에 관한 통찰을 제공

01

Data Preprocessing

데이터 전처리



데이터 전처리

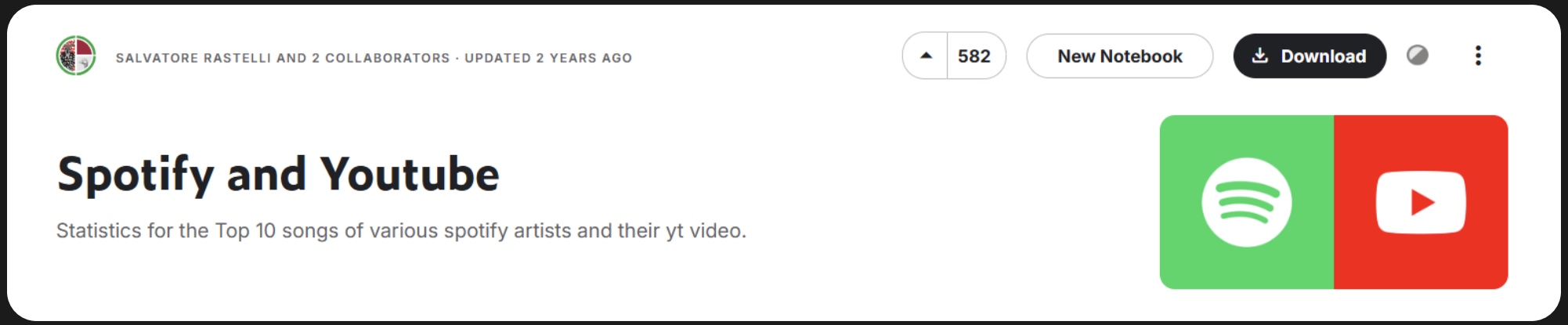
```python
# drop.py

import pandas as pd
import numpy as np

file = input("csv directory: ")

df = pd.read_csv(file)
print(df['official_video'])
df = df[ df['official_video'] == df['official_video'][0] ]
df.to_csv(f"{file.split('/')[-1]}_dropped.csv", index=False)
```

데이터 크롤링 및 확장

Feature Engineering

데이터 크롤링 및 확장

Youtube API, Spotify 크롤링



트랙 이름, 아티스트 이름, 아티스트 URL, Key, Tempo 등의
음원 사이트의 요소와 함께 **유튜브 뮤직비디오 링크, 조회수 등이 포함**

데이터 크롤링 및 확장

데이터 분류 및 분할

```python
# drop.py

import pandas as pd
import numpy as np

file = input("csv directory: ")

df = pd.read_csv(file)
print(df['official_video'])
df = df[ df['official_video'] == df['official_video'][0] ]
df.to_csv(f"{file.split('/')[-1]}_dropped.csv", index=False)
```

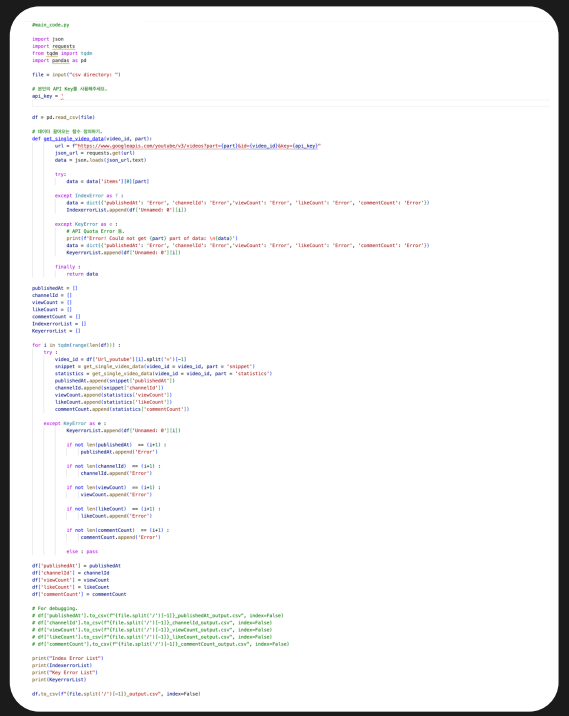
```python
#split.py

import pandas as pd
import numpy as np

file = input("csv directory: ")

df = pd.read_csv(file)

num = int(input("line size per file: "))

for i in range(int(len(df)/num) + 1) :
    data = df[num*i:num*(i+1)]
    data.to_csv(f"{file.split('/')[-1]}_{i+1}.csv", index=False)
```

공식 뮤직비디오만 남겨두고 저장

일일 10,000개의 API 요청제한으로 csv 파일을 4,500개씩 분할

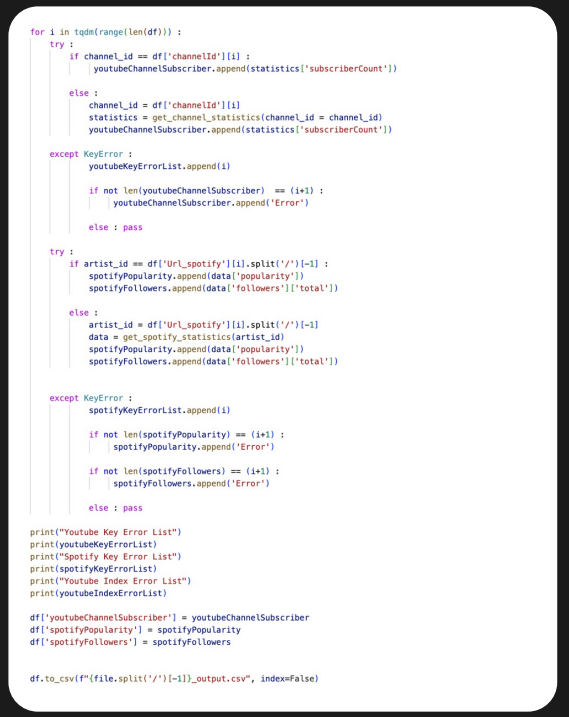
데이터 크롤링 및 확장

데이터 업데이트



```
#main_code.py

import json
import requests
from tqdm import tqdm
import pandas as pd

file = input("csv directory: ")

# 본인의 API Key를 사용하세요.
api_key = ''


df = pd.read_csv(file)

# 데이터 불러오는 함수 정의하기.
def get_single_video_data(video_id, part):
    url = f"https://www.googleapis.com/youtube/v3/videos?part={part}&id={video_id}&key={api_key}"
    json_url = requests.get(url)
    data = json.loads(json_url.text)

    try:
        data = data['items'][0][part]

    except IndexError as i :
        data = dict({'publishedAt': 'Error', 'channelId': 'Error',"viewCount': 'Error', 'likeCount': 'Error', 'commentCount': 'Error'})
        IndexerrorList.append(df['Unnamed: 0'][i])

    except KeyError as e :
        # API Quota Error 등.
        print(f"Error! Could not get {part} part of data: \n{data}")
        data = dict({'publishedAt': 'Error', 'channelId': 'Error',"viewCount': 'Error', 'likeCount': 'Error', 'commentCount': 'Error'})
        KeyerrorList.append(df['Unnamed: 0'][i])

    finally :
        return data

publishedAt = []
channelId = []
viewCount = []
likeCount = []
commentCount = []
IndexerrorList = []
KeyerrorList = []

for i in tqdm(range(len(df))) :
    try :
        video_id = df['Url_youtube'][i].split("=")[-1]
        snippet = get_single_video_data(video_id = video_id, part = 'snippet')
        statistics = get_single_video_data(video_id = video_id, part = 'statistics')
        publishedAt.append(snippet["publishedAt"])
        channelId.append(snippet['channelId'])
        viewCount.append(statistics['viewCount'])
        likeCount.append(statistics['likeCount'])
        commentCount.append(statistics['commentCount'])

    except KeyError as e :
        KeyerrorList.append(df['Unnamed: 0'][i])

        if not len(publishedAt)  == (i+1) :
            publishedAt.append('Error')

        if not len(channelId)  == (i+1) :
            channelId.append('Error')

        if not len(viewCount)  == (i+1) :
            viewCount.append('Error')

        if not len(likeCount)  == (i+1) :
            likeCount.append('Error')

        if not len(commentCount)  == (i+1) :
            commentCount.append('Error')

        else : pass

df['publishedAt'] = publishedAt
df['channelId'] = channelId
df['viewCount'] = viewCount
df['likeCount'] = likeCount
df['commentCount'] = commentCount

# For debugging.
# df['publishedAt'].to_csv(f"{file.split('/')[-1]}_publishedAt_output.csv", index=False)
# df['channelId'].to_csv(f"{file.split('/')[-1]}_channelId_output.csv", index=False)
# df['viewCount'].to_csv(f"{file.split('/')[-1]}_viewCount_output.csv", index=False)
# df['likeCount'].to_csv(f"{file.split('/')[-1]}_likeCount_output.csv", index=False)
# df['commentCount'].to_csv(f"{file.split('/')[-1]}_commentCount_output.csv", index=False)

print("Index Error List")
print(IndexerrorList)
print("Key Error List")
print(KeyerrorList)

df.to_csv(f"{file.split('/')[-1]}_output.csv", index=False)
```

```
import json
import requests
from tqdm import tqdm
import pandas as pd

file = input("csv directory: ")

(API_KEY_str)

df = pd.read_csv(file)

# 유튜브 채널 데이터 불러오는 함수
def get_channel_statistics(channel_id):
    url = f'https://www.googleapis.com/youtube/v3/channels?part=statistics&id={channel_id}&key={youtube_api_key}'
    json_url = requests.get(url)
    data = json.loads(json_url.text)

    try:
        data = data['items'][0]['statistics']

    except KeyError:
        print('Could not get channel statistics')
        data = dict({'subscriberCount': 'Error'})
        youtubeKeyErrorList.append(i)

    except IndexError :
        data = dict({'subscriberCount': 'Error'})
        youtubeIndexErrorList.append(i)

    return data

# 스포티파이 토큰 가져오는 함수 (1시간마다 초기화 됨)
def get_spotify_token(spotify_client_id, spotify_client_secret):
    headers = {
        'Content-Type': 'application/x-www-form-urlencoded'
        }

    body = {
        "grant_type" : "client_credentials",
        "client_id" : spotify_client_id,
        "client_secret" : spotify_client_secret
        }

    url = 'https://accounts.spotify.com/api/token'
    json_url = requests.post(url, data = body, headers = headers)
    data = json.loads(json_url.text)
    print(data['token_type'] + ' ' + data['access_token'])
    return data['token_type'] + ' ' + data['access_token']

token = get_spotify_token(spotify_client_id = spotify_client_id, spotify_client_secret = spotify_client_secret)

# 스포티파이 좋아요와 인기도 불러오는 함수
def get_spotify_statistics(artist_id):
    headers = {
        'Authorization': token
        }

    url = f'https://api.spotify.com/v1/artists/{artist_id}'
    json_url = requests.get(url, headers=headers)
    data = json.loads(json_url.text)

    return data

# 기본값
youtubeKeyErrorList = []
spotifyKeyErrorList = []
youtubeIndexErrorList = []
youtubeChannelSubscriber = []
spotifyPopularity = []
spotifyFollowers = []
channel_id = 'default'
artist_id = 'default'
```



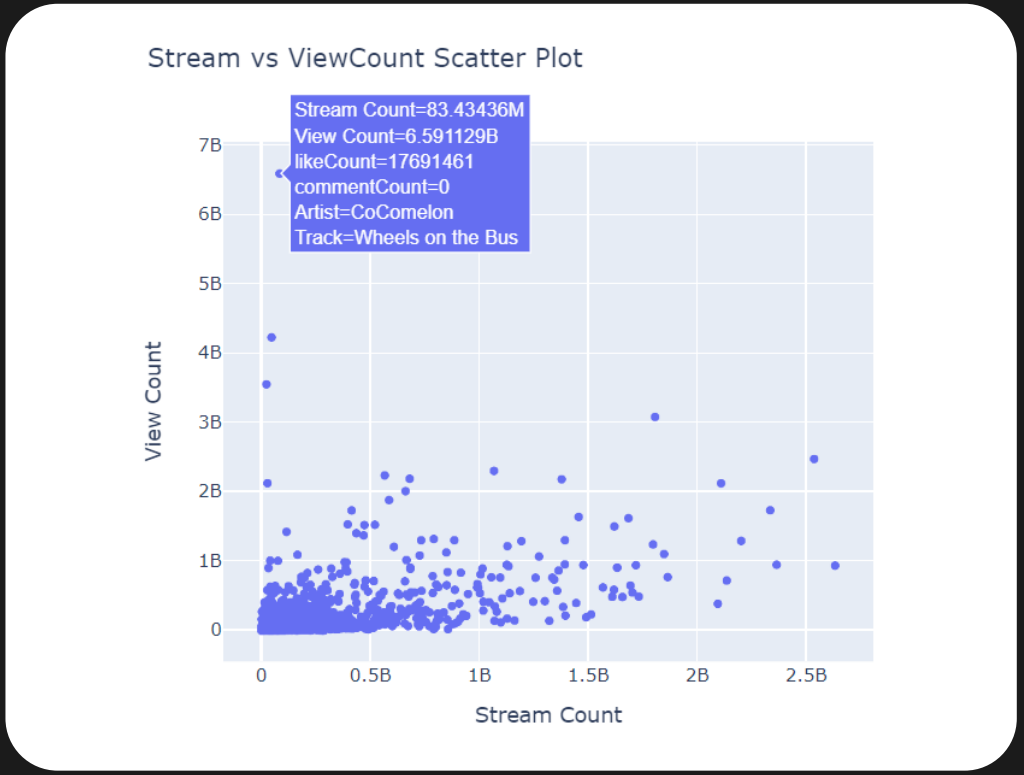
```
for i in tqdm(range(len(df))) :
    try :
        if channel_id == df['channelId'][i] :
            youtubeChannelSubscriber.append(statistics['subscriberCount'])

        else :
            channel_id = df['channelId'][i]
            statistics = get_channel_statistics(channel_id = channel_id)
            youtubeChannelSubscriber.append(statistics['subscriberCount'])

    except KeyError :
        youtubeKeyErrorList.append(i)

        if not len(youtubeChannelSubscriber)  == (i+1) :
            youtubeChannelSubscriber.append('Error')

        else : pass

    try :
        if artist_id == df['Url_spotify'][i].split('/')[-1] :
            spotifyPopularity.append(data['popularity'])
            spotifyFollowers.append(data['followers']['total'])

        else :
            artist_id = df['Url_spotify'][i].split('/')[-1]
            data = get_spotify_statistics(artist_id)
            spotifyPopularity.append(data['popularity'])
            spotifyFollowers.append(data['followers']['total'])


    except KeyError :
        spotifyKeyErrorList.append(i)

        if not len(spotifyPopularity) == (i+1) :
            spotifyPopularity.append('Error')

        if not len(spotifyFollowers) == (i+1) :
            spotifyFollowers.append('Error')

        else : pass

print("Youtube Key Error List")
print(youtubeKeyErrorList)
print("Spotify Key Error List")
print(spotifyKeyErrorList)
print("Youtube Index Error List")
print(youtubeIndexErrorList)

df['youtubeChannelSubscriber'] = youtubeChannelSubscriber
df['spotifyPopularity'] = spotifyPopularity
df['spotifyFollowers'] = spotifyFollowers


df.to_csv(f"{file.split('/')[-1]}_output.csv", index=False)
```

업로드 시점과 채널 ID를 크롤링하고, 2년 전에 업데이트되어 추가적인 업데이트가 필요하다고 판단되는

조회수, 좋아요 수, 댓글 수를 크롤링하여 csv 파일에 추가

Feature Engineering

이상치 및 결측치 처리



아동용 음악의 경우에서 스포티파이 스트리밍 수보다 유튜브 조회수가 월등히 높은 결과를 보임

Feature Engineering

이상치 관측 및 추출

```
df['is_children'].value_counts()
```

```
is_children
0    15713
1       10
Name: count, dtype: int64
```

동요에 해당하는 데이터는 10개로 추출

여기까지 데이터의 이상치와 결측치 제거 및 전처리를 마치며 데이터프레임을 csv로 저장

Feature Engineering

새로운 변수 정의

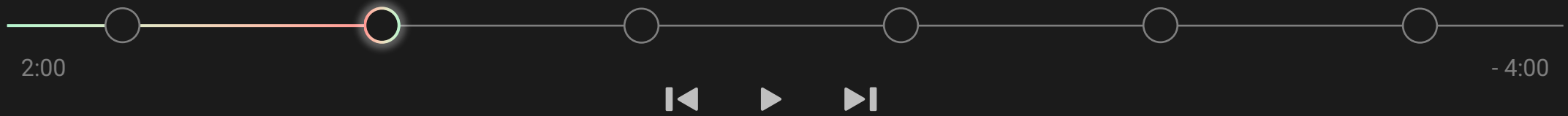
```
# delta_view, delta_like, delta_comment 열 추가 코드
df['delta_view'] = df['viewCount'] - df['Views']
df['delta_like'] = df['likeCount'] - df['Likes']
df['delta_comment'] = df['commentCount'] - df['Comments']
```

새롭게 크롤링한 조회수, 좋아요 수, 댓글 수에 대해 **증가량**을 새로운 변수로 정의

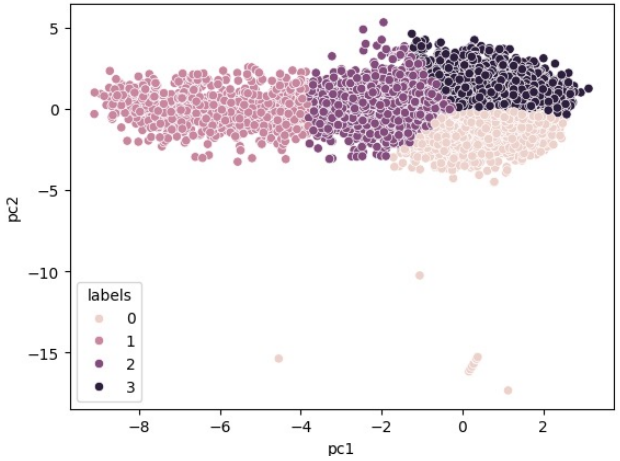
02

Clustering

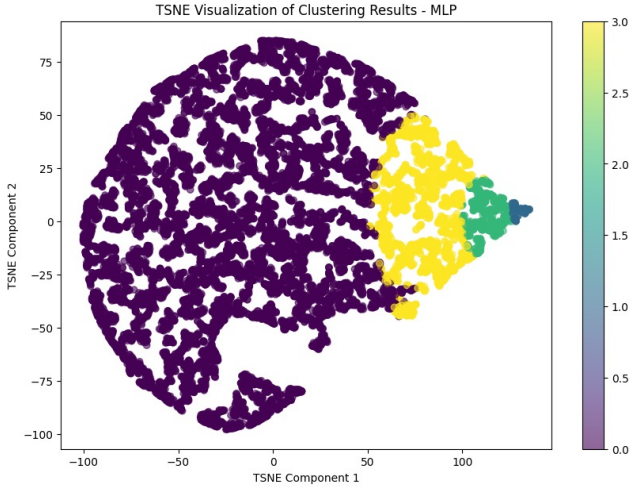
클러스터링



클러스터링



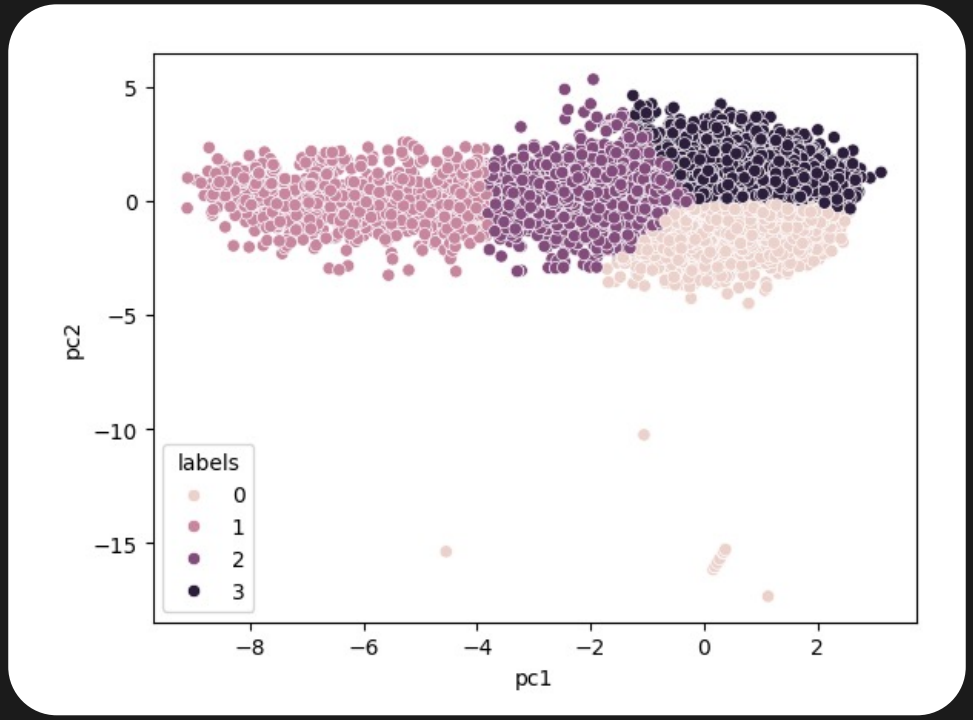
ML algorithms



DLL algorithms

ML algorithms (1)

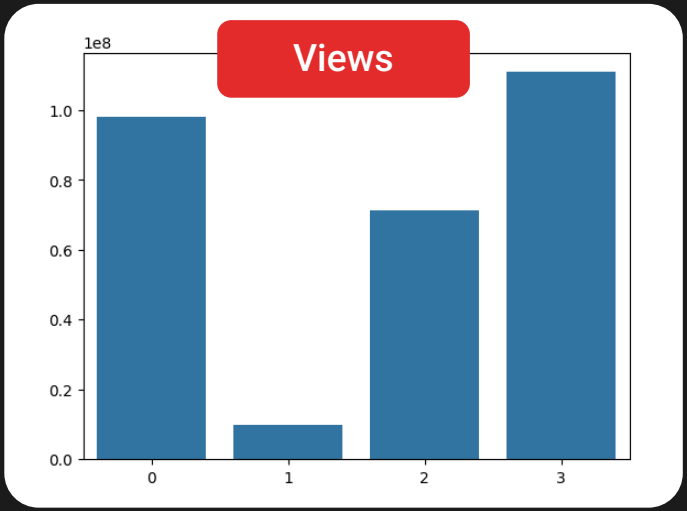
K – means clustering



k = 4 에 해당하는 클러스터링 시각화

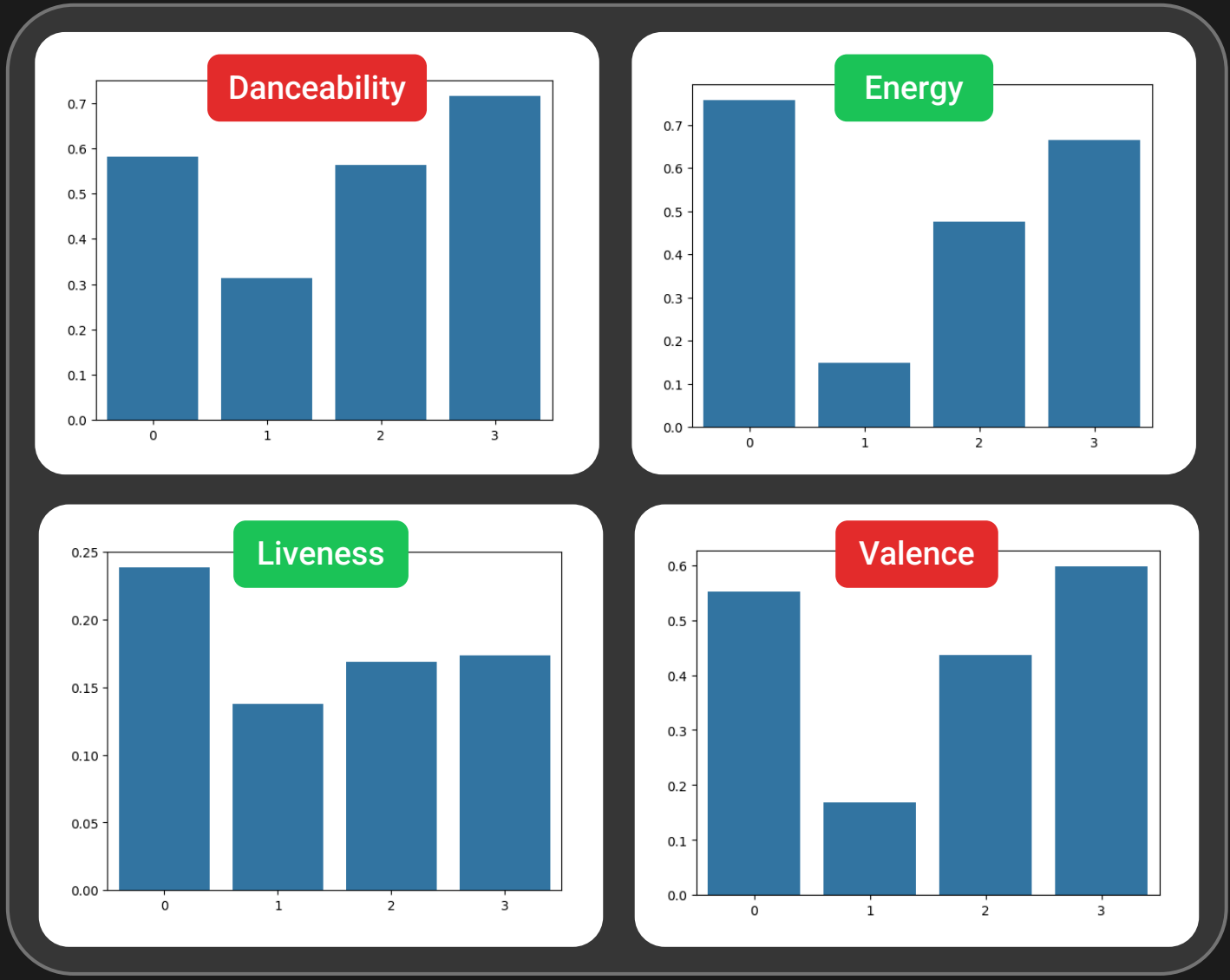
ML algorithms (1)

K-means Clustering



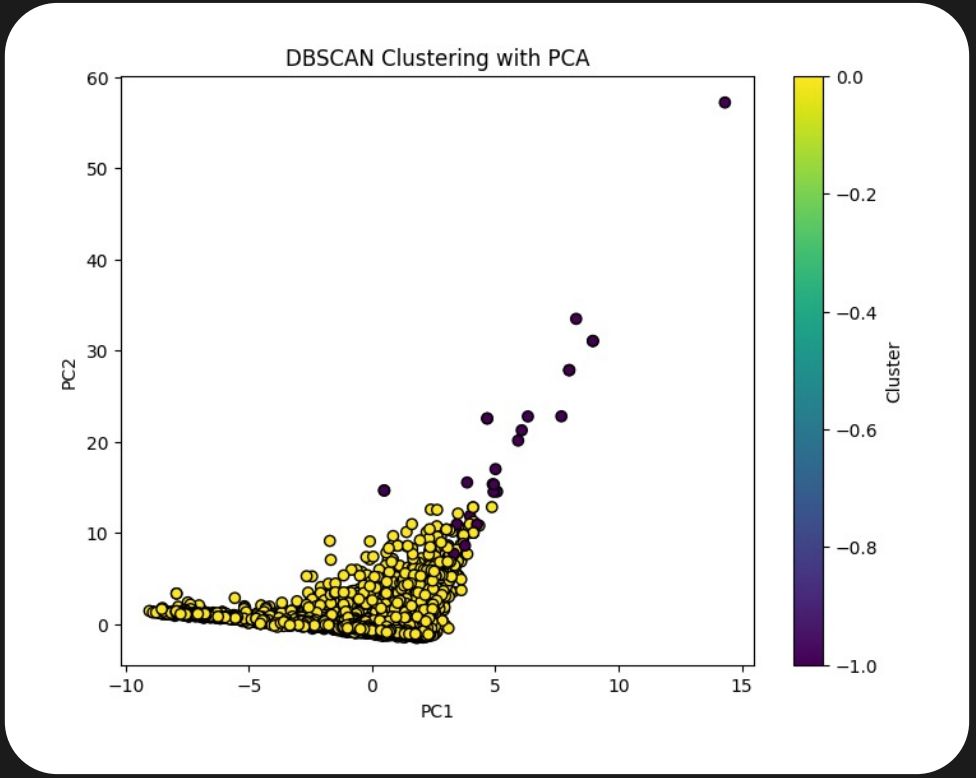
좌측 | Views 평균값

우측 | Danceability, Valence,
Energy, Liveness 평균값



ML algorithms (2)

DBSCAN(Density-Based Spatial Clustering of Applications with Noise)



K-means와 달리 클러스터 개수를 사전에 설정할 필요가 없다

K-means대비 군집 구분이 명확하지 않음

DLL algorithms (1)

Self-Supervised MLP와 K-means를 활용한 클러스터링

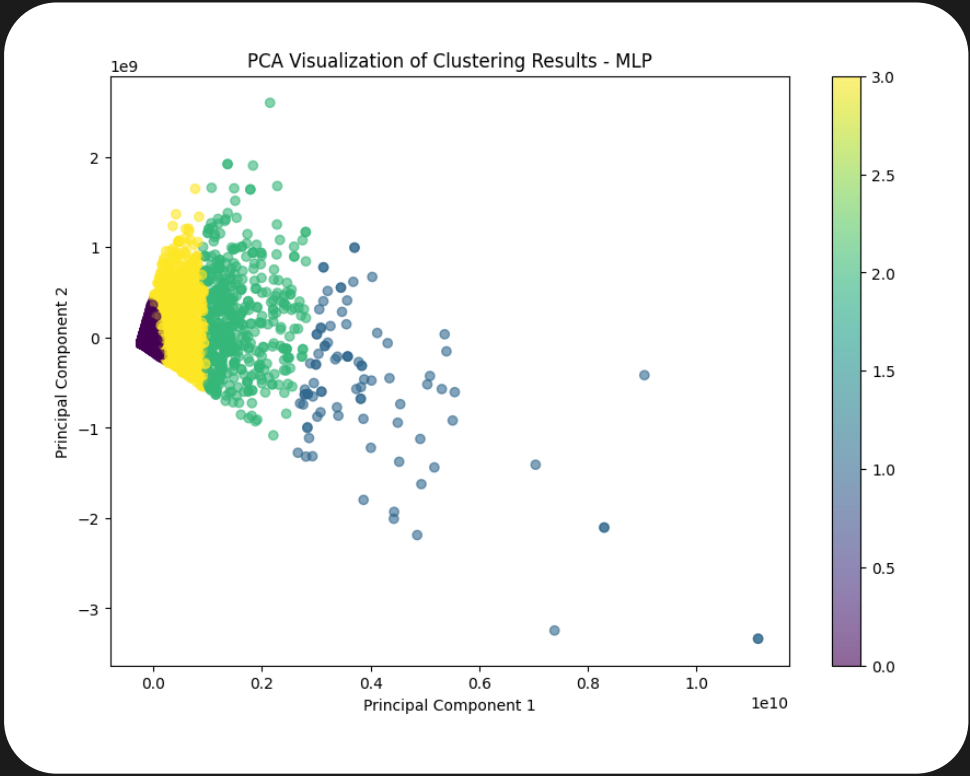
```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim=128):
        super(MLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 64),
            nn.ReLU(),
            nn.Linear(64, input_dim)
        )

    def forward(self, x):
        return self.model(x)
```

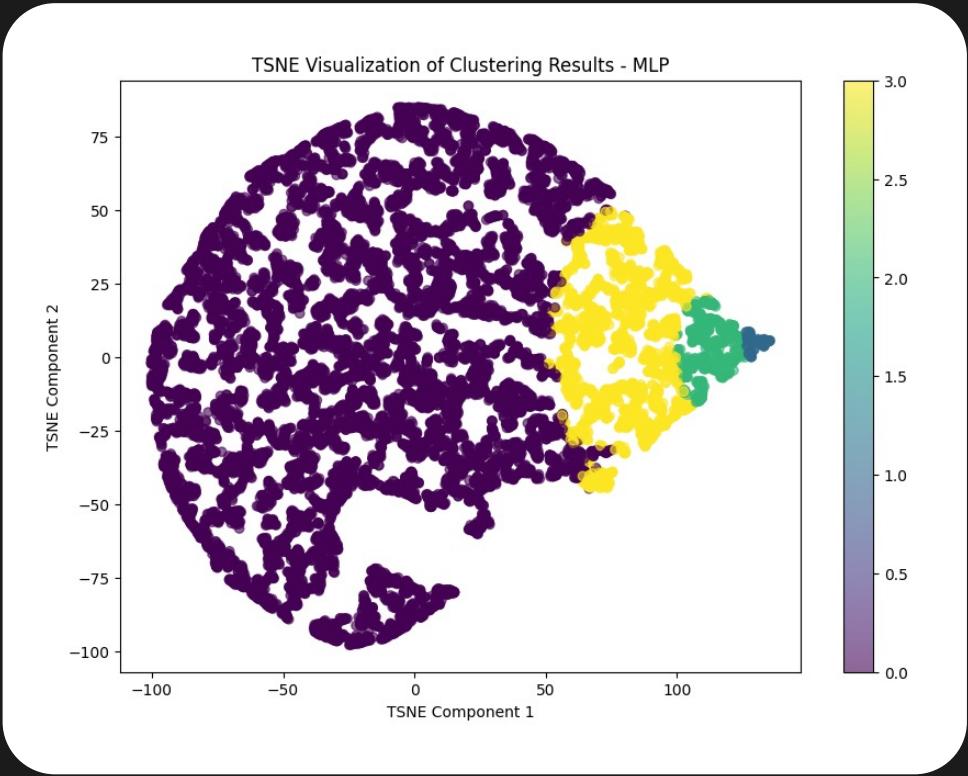
128개의 은닉 뉴런(hidden_dim)과 64차원을 거친 후 다시 원래 차원으로 복원하는 구조

활성화 함수로 ReLU 사용

DLL algorithms (1)

Self-Supervised MLP와 K-means를 활용한 클러스터링

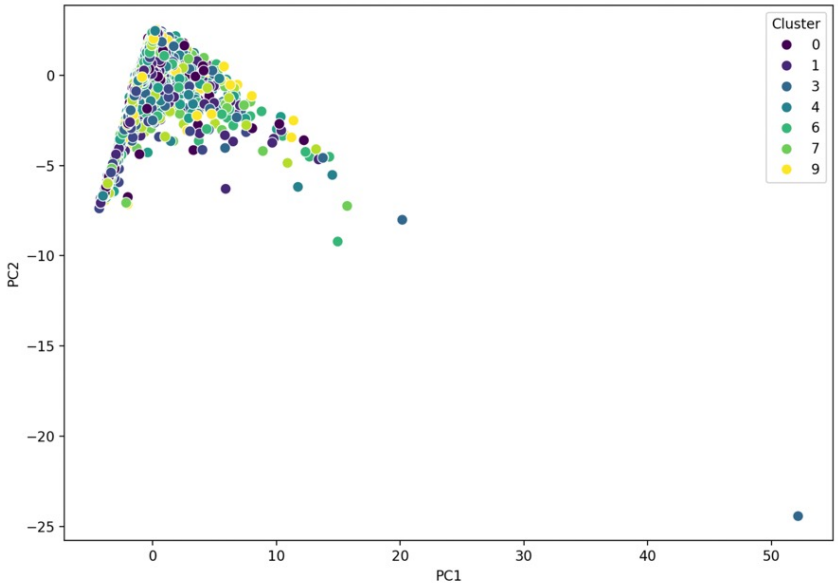




PCA와 t-SNE를 활용한 시각화 결과

DLL algorithms (2)

VAE(Variational autoencoder)

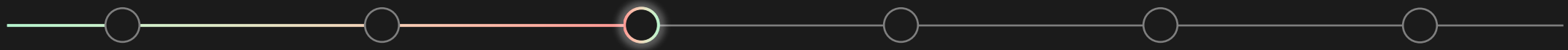


PCA를 활용한 시각화 결과

03

Regression

회귀

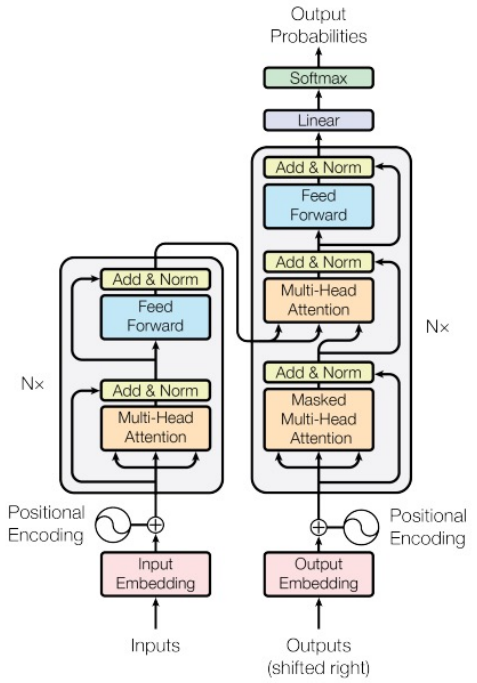


회귀

The Least Squares Method

$$SS_{(residuals)} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

ML algorithms



DLL algorithms

ML algorith

OLS Regression

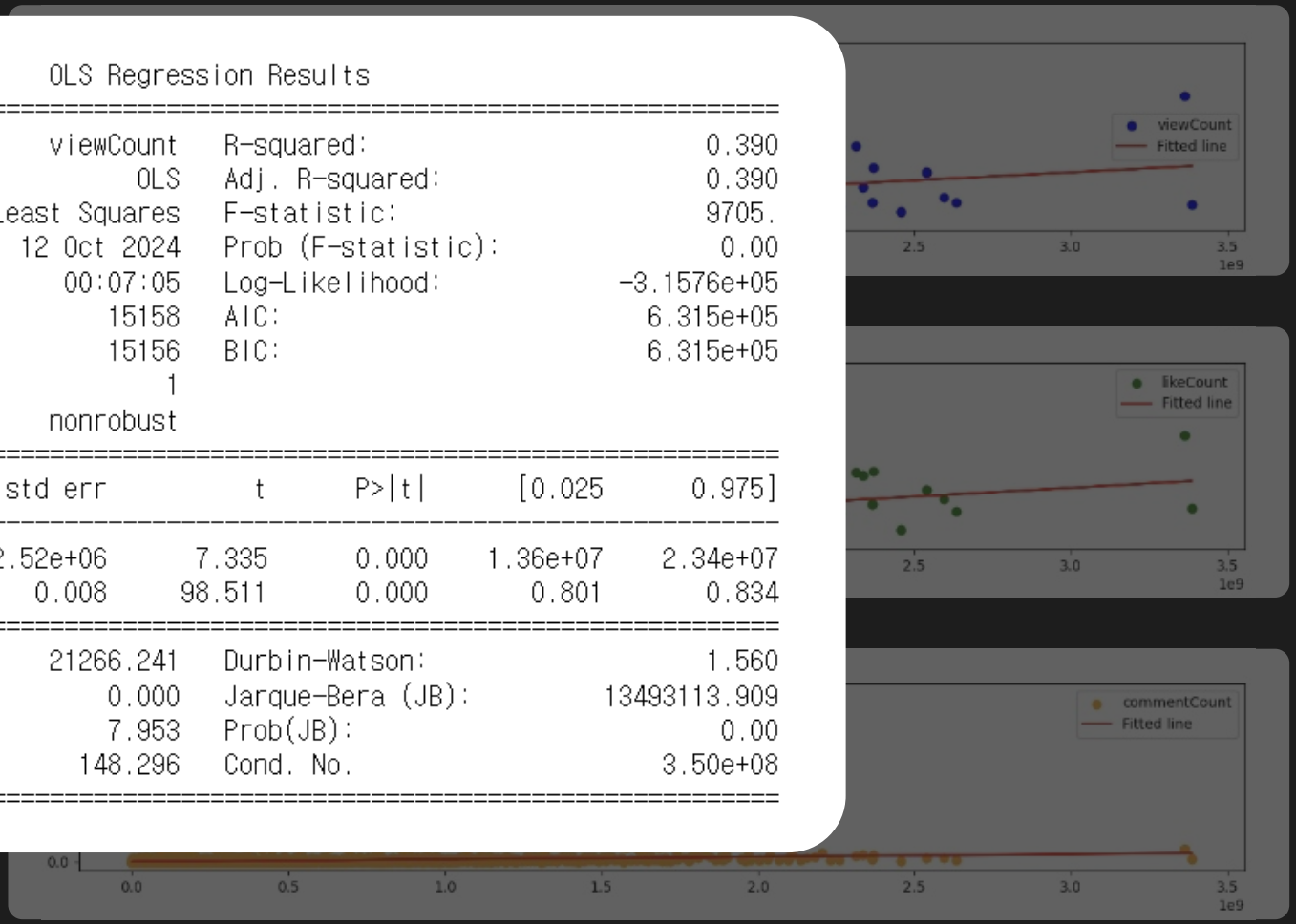
차례대로 스포티파이의 스
유튜브 조회수, 좋아요수, 댓
대응하는 선형회귀

OLS Regression Results

Dep. Variable:	viewCount	R-squared:	0.390
Model:	OLS	Adj. R-squared:	0.390
Method:	Least Squares	F-statistic:	9705.
Date:	Sat, 12 Oct 2024	Prob (F-statistic):	0.00
Time:	00:07:05	Log-Likelihood:	-3.1576e+05
No. Observations:	15158	AIC:	6.315e+05
Df Residuals:	15156	BIC:	6.315e+05
Df Model:	1		
Covariance Type:	nonrobust		

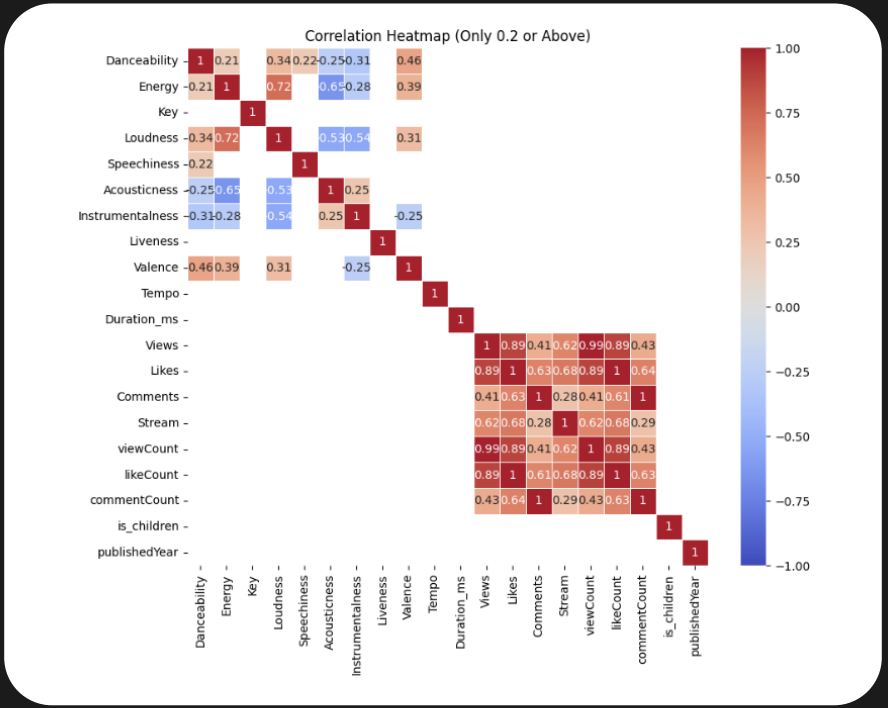
	coef	std err	t	P>|t|	[0.025	0.975]
const	1.85e+07	2.52e+06	7.335	0.000	1.36e+07	2.34e+07
Stream	0.8175	0.008	98.511	0.000	0.801	0.834

Omnibus:	21266.241	Durbin-Watson:	1.560
Prob(Omnibus):	0.000	Jarque-Bera (JB):	13493113.909
Skew:	7.953	Prob(JB):	0.00
Kurtosis:	148.296	Cond. No.	3.50e+08



ML algorithms (1)

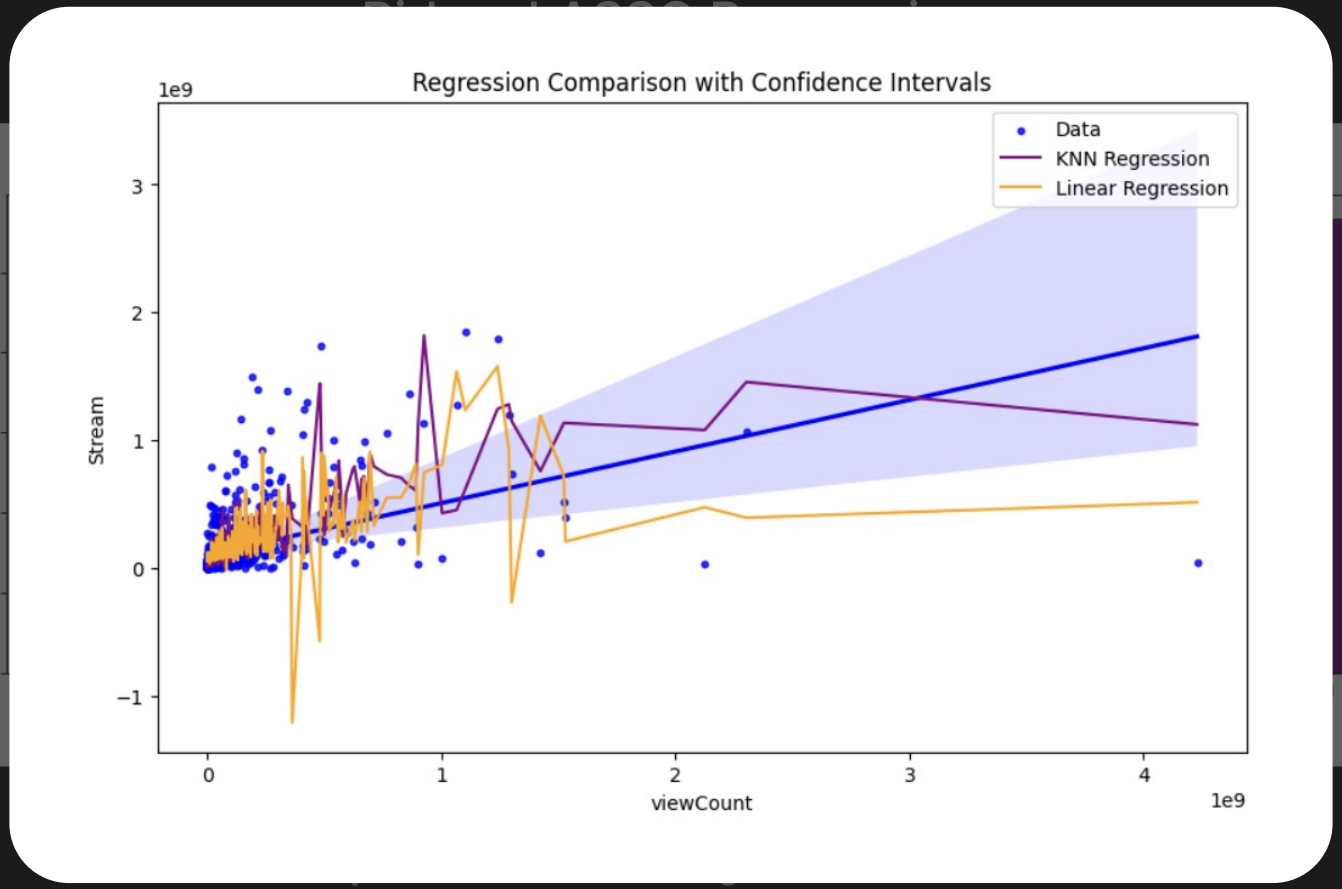
OLS Regression



수치형 특성 간의 상관계수 히트맵

숫자로 나타낼 수 있는 feature로는 **Danceability, Energy, Loudness** 등이 존재

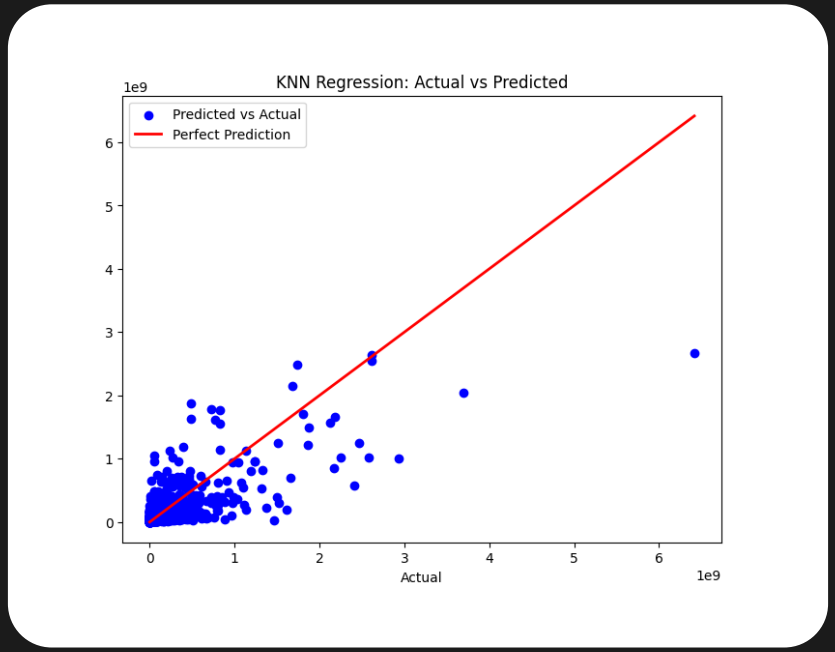
ML algorithms (2)



MSE: Linear = Ridge = LASSO < KNN

ML algorithms (3)

KNN Regression

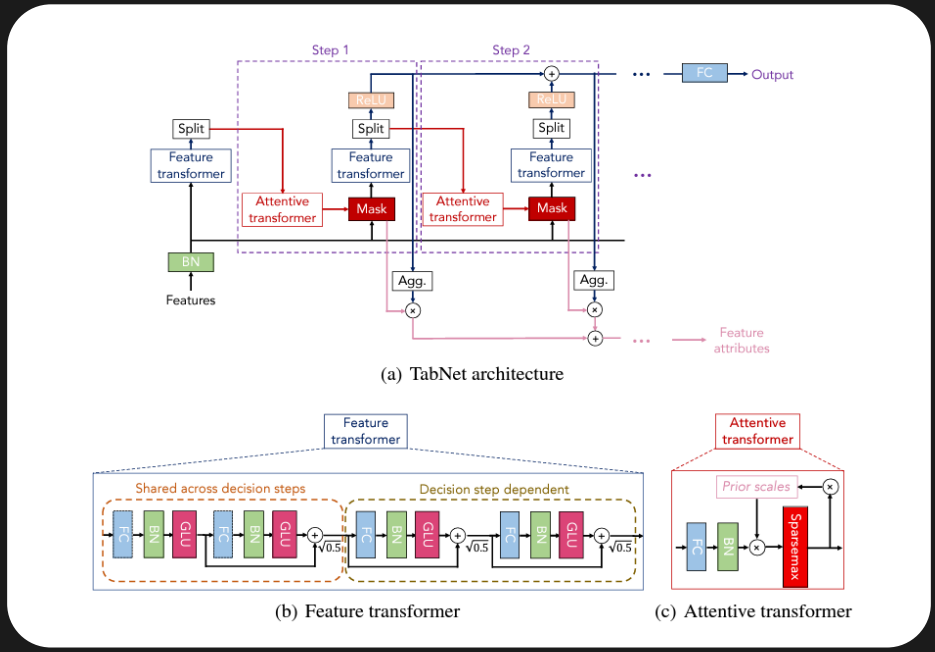


전체 feature 에 대한 학습 결과

(K = 3): 0.0382 (NRMSE) / 0.5071 (R-Squared)

DLL algorithms (1)

TabNetRegressor



Attention 메커니즘을 활용하여 학습 중 중요한 feature를 동적으로 선택하는 정형 데이터 특화 딥러닝 모델

Decision Step, Feature Transformer, Attentive Transformer 등을 활용하여

입력 데이터의 feature를 점진적으로 학습하고 이를 업데이트

DLL algorithms (1)

TabNetRegressor

```
# 하이퍼파라미터 설정
config = {
    "n_d": 8,                  # Decoder feature dimension size (8~64)
    "n_a": 8,                  # Attention feature dimension size (n_d = n_a)
    "n_steps": 5,              # Number of steps in the architecture. (3~10)
    "gamma": 1.3,              # Relaxation parameter for sparsity (1.0~2.0)
    "lambda_sparse": 1e-3,     # Sparsity regularization
    "learning_rate": 1e-4,     # Learning rate
    "max_epochs": 200,         # Maximum number of epochs for training
    "batch_size": 512,         # Batch size for training (64, 128, 256, 512, 1024, ...)
    "virtual_batch_size": 256, # Virtual batch size for gradient accumulation (should divide batch_size)
    "seed": 42                 # Random seed for reproducibility.
}
```

```
# 모델 초기화
model = TabNetRegressor(
    n_d=config["n_d"],
    n_a=config["n_a"],
    n_steps=config["n_steps"],
    gamma=config["gamma"],
    lambda_sparse=config["lambda_sparse"],
    optimizer_params=dict(lr=config["learning_rate"])
)

# 모델 학습
model.fit(
    X_train=X_train.values, y_train=y_train.values.reshape(-1, 1),
    eval_set=[(X_test.values, y_test.values.reshape(-1, 1))],
    eval_name=['val'],
    eval_metric=['rmse'],
    max_epochs=config["max_epochs"],
    patience=10,
    batch_size=config["batch_size"],
    virtual_batch_size=config["virtual_batch_size"],
    num_workers=0,
    drop_last=False,
```

결정계수: 0.4252, nrmse: 0.0385

DLL algorithms (2)

TabTransformer

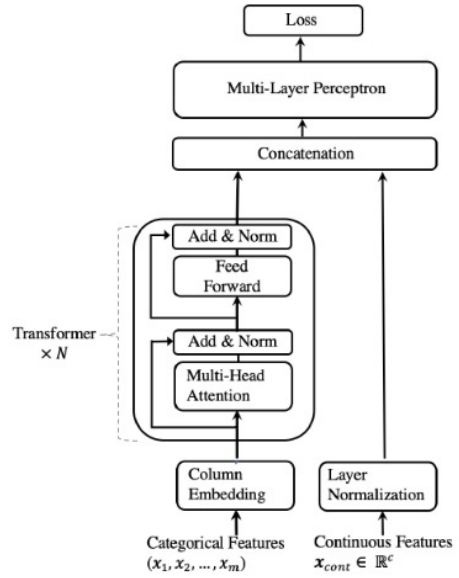


Figure 1: The architecture of TabTransformer.

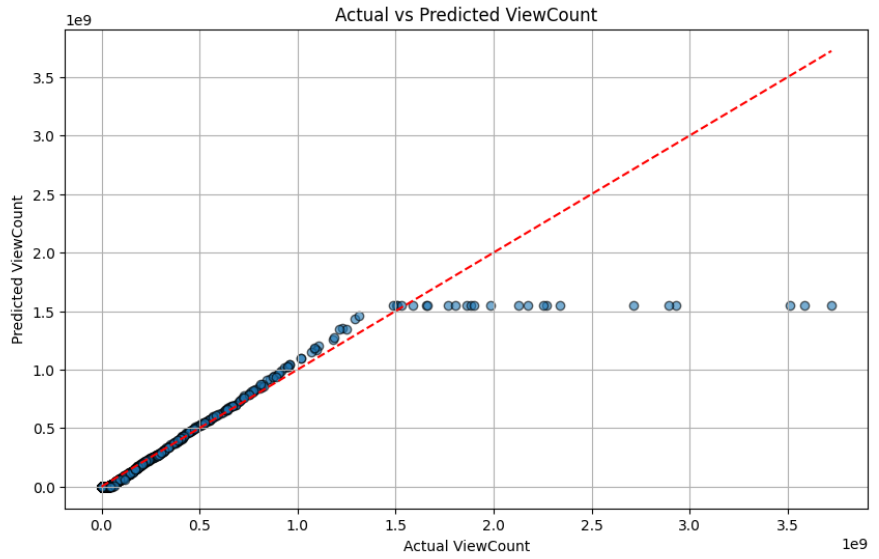
범주형 feature에 임베딩을 적용하여 연속형 벡터로 변환한 뒤, multi-head attention을 활용해 feature 간의 상호작용 관계를 학습

기존의 원핫 인코딩(one-hot encoding) 방식보다 더 풍부한 정보를 학습

DLL algorithms (2)

TabTransformer

```python
# TabTransformer 모델 정의
class TabTransformer(nn.Module):
    def __init__(self, input_dim, embed_dim=32, num_heads=4, num_blocks=4, hidden_dim=64):
        super(TabTransformer, self).__init__()
        self.embedding = nn.Linear(input_dim, embed_dim)
        self.transformer_blocks = nn.ModuleList([
            nn.TransformerEncoderLayer(d_model=embed_dim, nhead=num_heads)
            for _ in range(num_blocks)
        ])
        self.fc = nn.Sequential(
            nn.Linear(embed_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 1),
            nn.ReLU()
        )
    def forward(self, x):
        x = self.embedding(x)
        for block in self.transformer_blocks:
            x = block(x.unsqueeze(1)).squeeze(1)
        return self.fc(x)
```



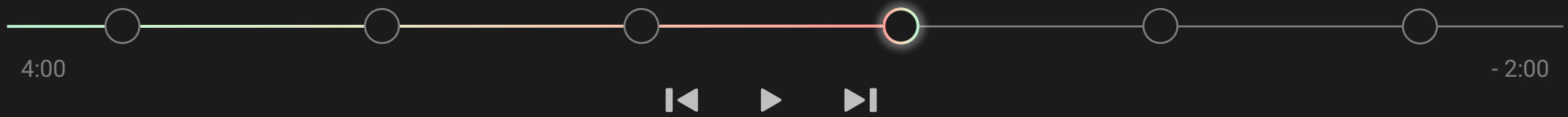
모델 정의 및 시각화

실제 결과값과 예측값의 분포가 유사하게 나타나지만, 조회수가 20억이 넘어가는 데이터들에 대해서는 추론 능력이 떨어짐

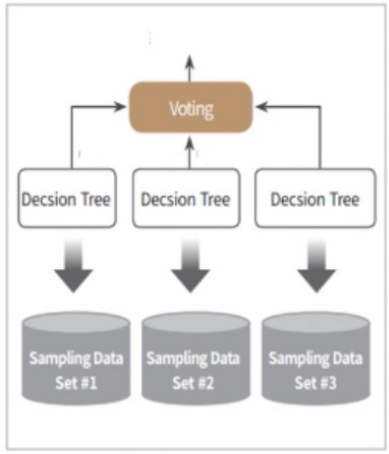
04

Classification

분류

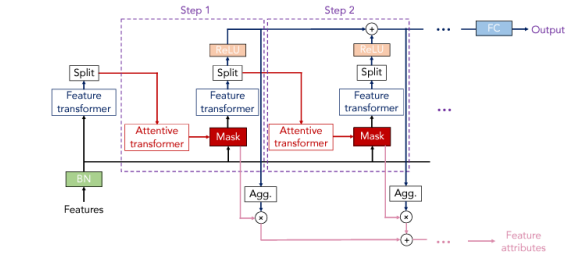


분류

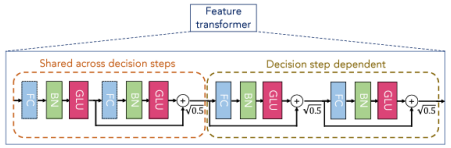


Bagging

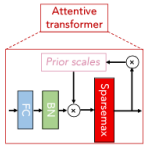
ML algorithms



(a) TabNet architecture



(b) Feature transformer



(c) Attentive transformer

DLL algorithms

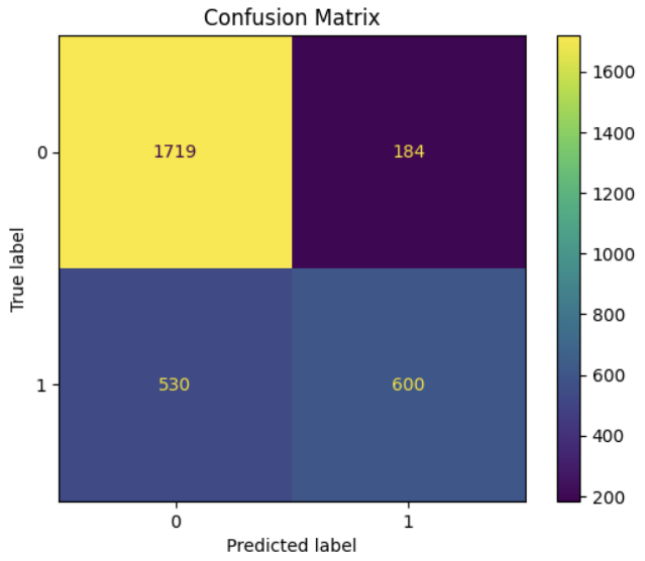
ML algorithms (1)

Random Forest

Accuracy: 0.7646

Classification Report:

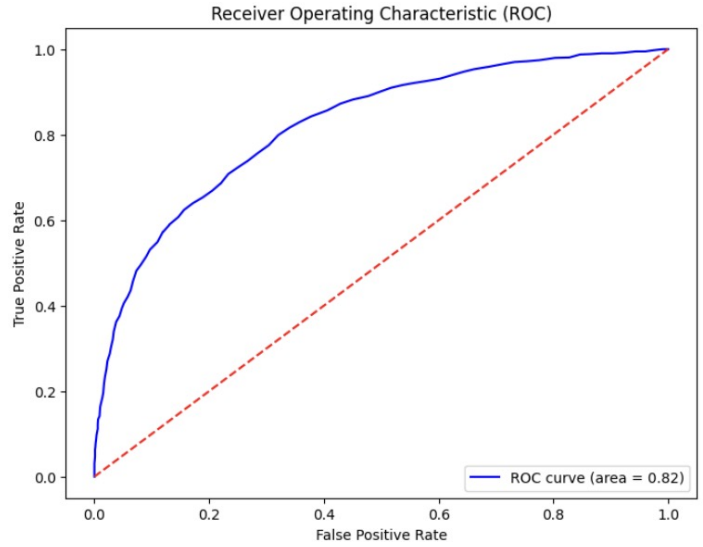
	precision	recall	f1-score	support
0	0.76	0.90	0.83	1903
1	0.77	0.53	0.63	1130
accuracy			0.76	3033
macro avg	0.76	0.72	0.73	3033
weighted avg	0.76	0.76	0.75	3033

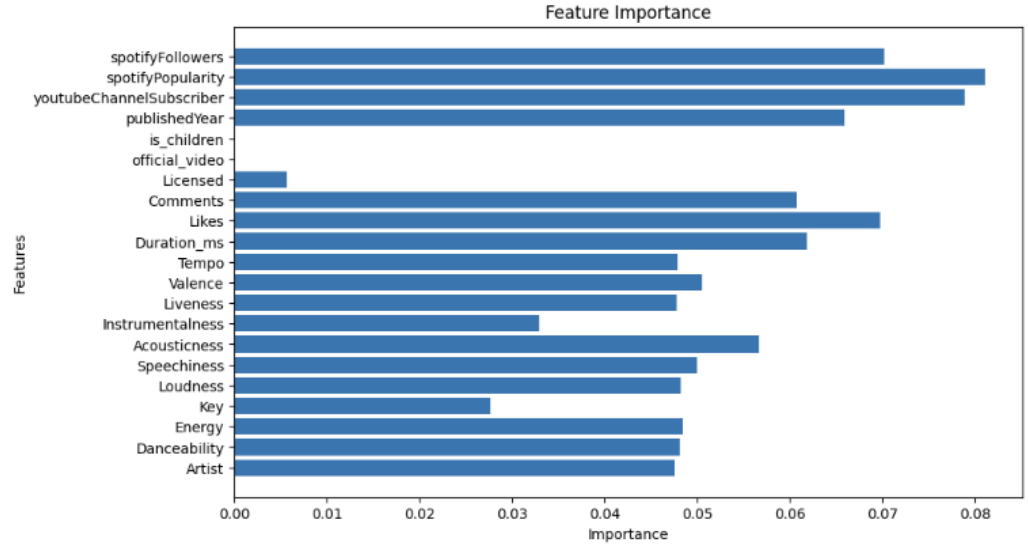


Train : Test 비율에 따른 정확도는 0.7646 (8 : 2) > 0.7501 (7 : 3) > 0.7462 (9 : 1) 로 나타남

ML algorithms (1)

Random Forest





'spotifyPopularity', 'youtubeChannelSubscriber', 'publishedYear', 'Likes', 'spotifyFollowers' 등이 주요 특성으로 관찰

이러한 결과에 근거하여 해당 특성으로 모델을 **재학습**

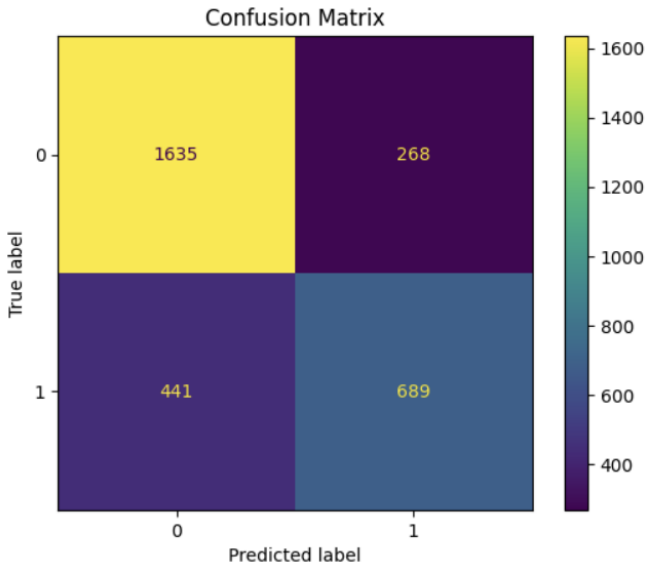
ML algorithms (1)

Random Forest

Accuracy: 0.7662

Classification Report:

	precision	recall	f1-score	support
0	0.79	0.86	0.82	1903
1	0.72	0.61	0.66	1130
accuracy			0.77	3033
macro avg	0.75	0.73	0.74	3033
weighted avg	0.76	0.77	0.76	3033



Confusion matrix 시각화

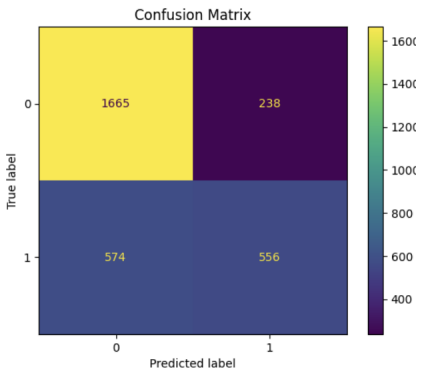
ML algorithms (2)

Gradient Boosting

Accuracy: 0.7323

Classification Report:

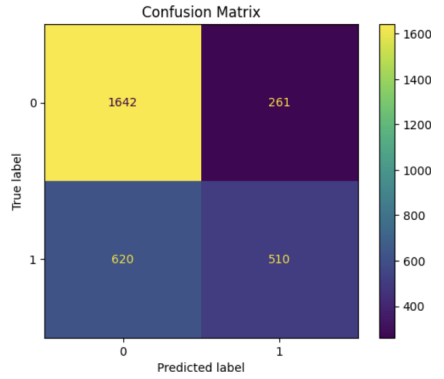
	precision	recall	f1-score	support
0	0.74	0.87	0.80	1903
1	0.70	0.49	0.58	1130
accuracy			0.73	3033
macro avg	0.72	0.68	0.69	3033
weighted avg	0.73	0.73	0.72	3033



Accuracy: 0.7095

Classification Report:

	precision	recall	f1-score	support
0	0.73	0.86	0.79	1903
1	0.66	0.45	0.54	1130
accuracy			0.71	3033
macro avg	0.69	0.66	0.66	3033
weighted avg	0.70	0.71	0.69	3033



정확도 비교 : 0.7323 (8 : 2) > 0.7274 (7 : 3) > 0.7264 (9 : 1)

'spotifyPopularity', 'youtubeChannelSubscriber', 'publishedYear' 특성을 추출 및 재학습했으나 성능이 **저하** (0.7323 -> 0.7095)

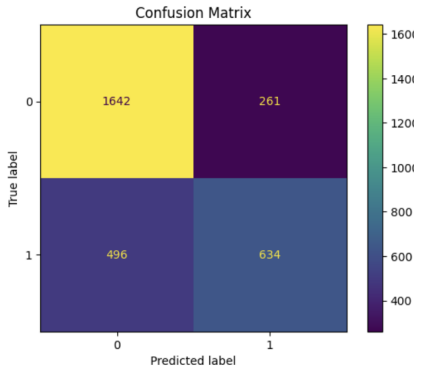
ML algorithms (3)

XGBoost

Accuracy: 0.7504

Classification Report:

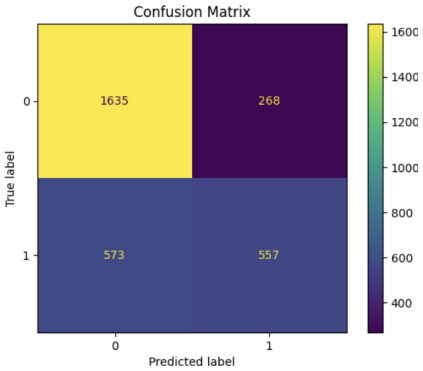
	precision	recall	f1-score	support
0	0.77	0.86	0.81	1903
1	0.71	0.56	0.63	1130
accuracy			0.75	3033
macro avg	0.74	0.71	0.72	3033
weighted avg	0.75	0.75	0.74	3033



Accuracy: 0.7227

Classification Report:

	precision	recall	f1-score	support
0	0.74	0.86	0.80	1903
1	0.68	0.49	0.57	1130
accuracy			0.72	3033
macro avg	0.71	0.68	0.68	3033
weighted avg	0.72	0.72	0.71	3033

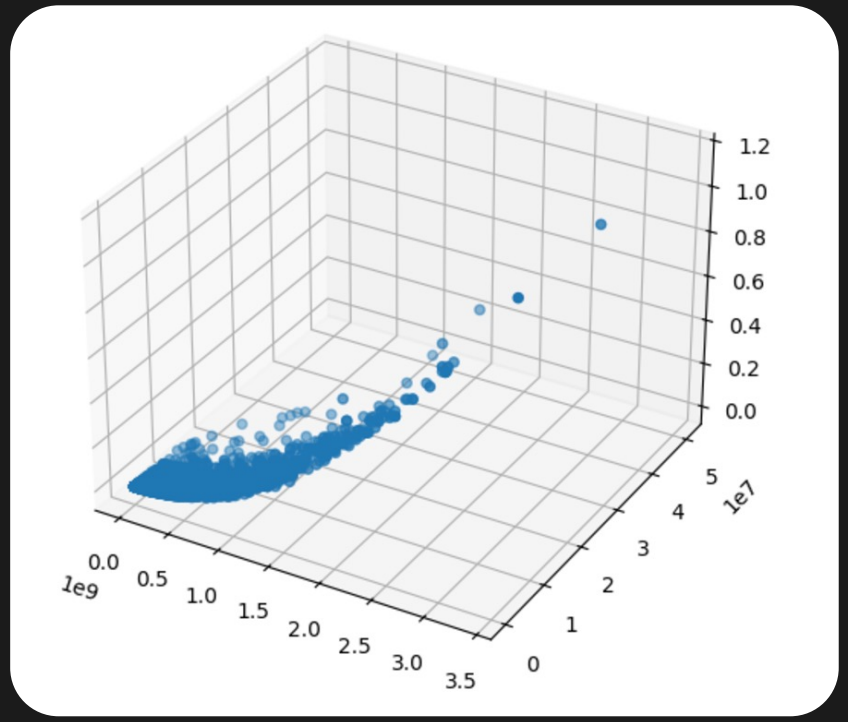


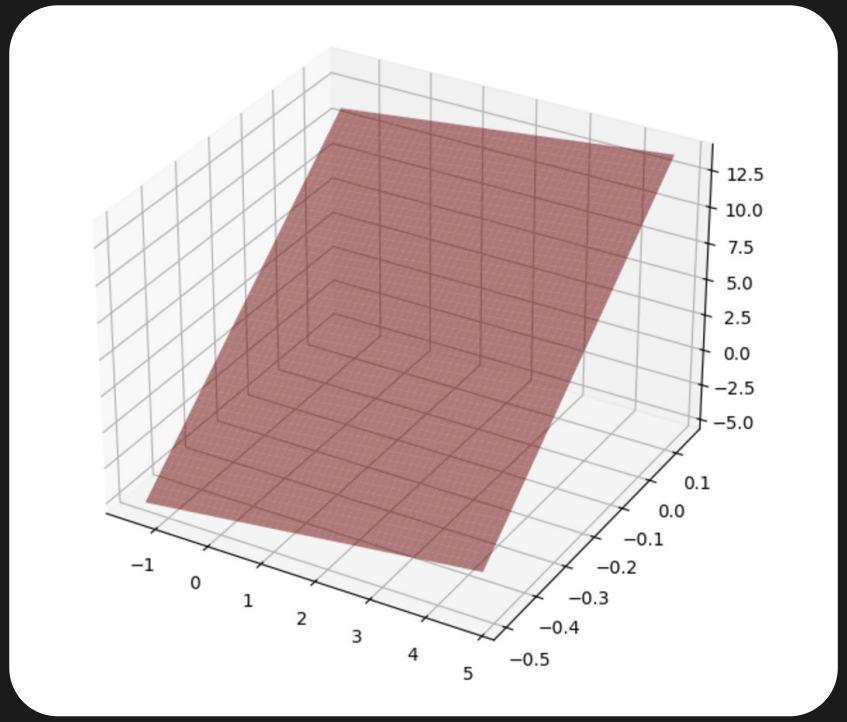
정확도 비교 : 0.7504 (8 : 2) > 0.7475 (9 : 1) > 0.7419 (7 : 3)

'spotifyPopularity', 'youtubeChannelSubscriber', 'publishedYear' 특성을 추출 및 재학습했으나 성능이 **저하** (0.7504 -> 0.7227)

ML algorithms (4)

SVM

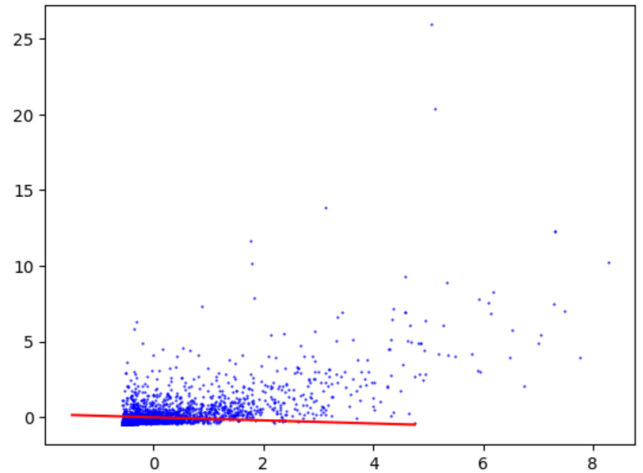




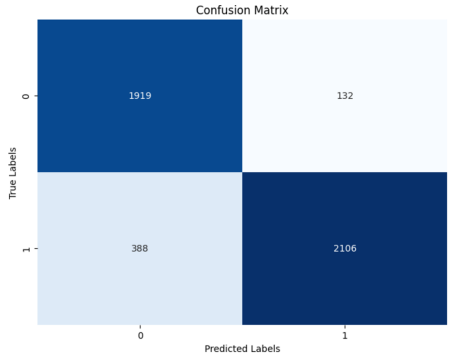
Kernel trick과 3차원 Hyperplane

ML algorithms (4)

SVM




```
Accuracy: 0.8855885588558856
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.94      0.88      2051
           1       0.94      0.84      0.89      2494

    accuracy                           0.89      4545
   macro avg       0.89      0.89      0.89      4545
weighted avg       0.89      0.89      0.89      4545

Confusion Matrix:
 [[1919  132]
 [ 388 2106]]
```



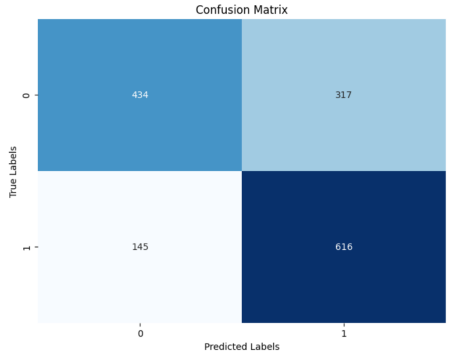
2차원 투영과 모델 추론 결과

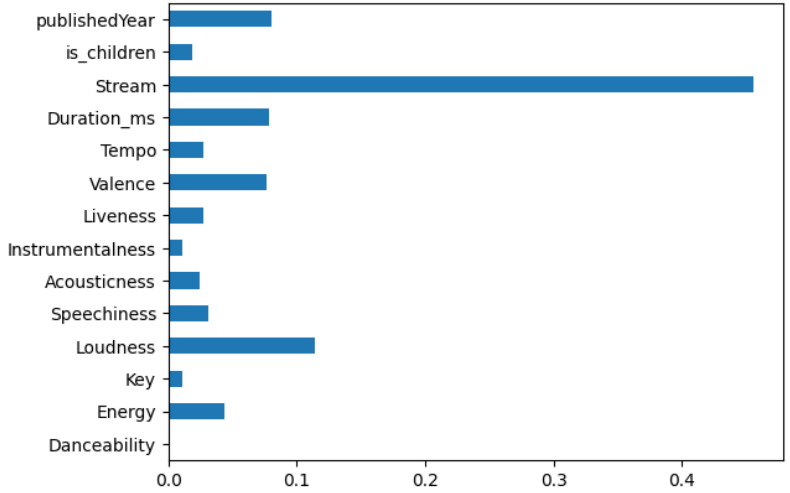
정확도: 0.8856 => SVM > Random Forest > XGBoost > Gradient Boost 순으로 높은 accuracy를 확인

DLL algorithms (1)

TabNetClassifier

```
Test Accuracy: 0.6944444444444444

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.58      0.65       751
           1       0.66      0.81      0.73       761

    accuracy                           0.69      1512
   macro avg       0.70      0.69      0.69      1512
weighted avg       0.70      0.69      0.69      1512
```



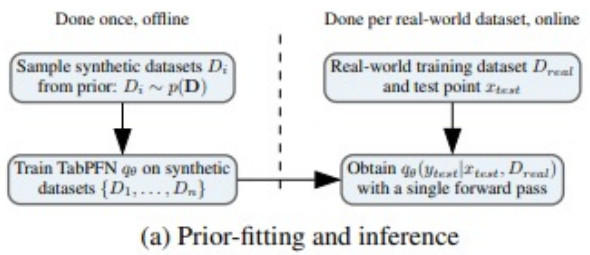


동일한 TabNet 모델을 사용한 모델 추론 결과와 feature importance 추출

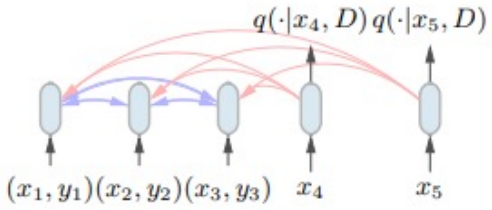
정확도: 0.6944

DLL algorithms (2)

TabPFN

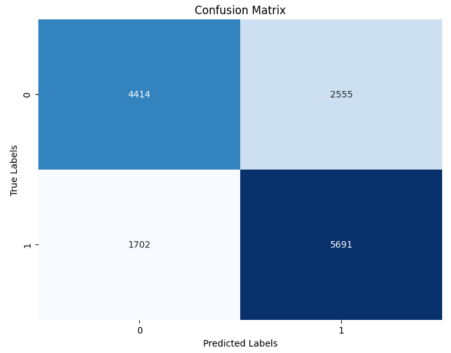


(a) Prior-fitting and inference



(b) Architecture and attention mechanism

```
Test Accuracy: 0.7035928143712575

Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.63      0.67      6969
           1       0.69      0.77      0.73      7393

    accuracy                           0.70     14362
   macro avg       0.71      0.70      0.70     14362
weighted avg       0.71      0.70      0.70     14362
```



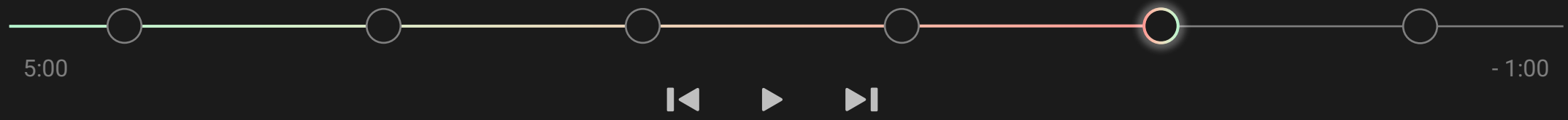
소규모 데이터셋에서 특히 뛰어난 성능을 발휘하며 복잡한 하이퍼파라미터 튜닝 없이도 높은 성능을 달성

정확도: 0.7036

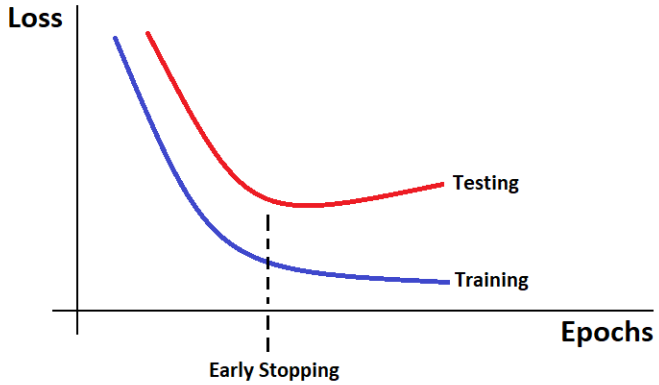
05

Hyperparameter Tuning

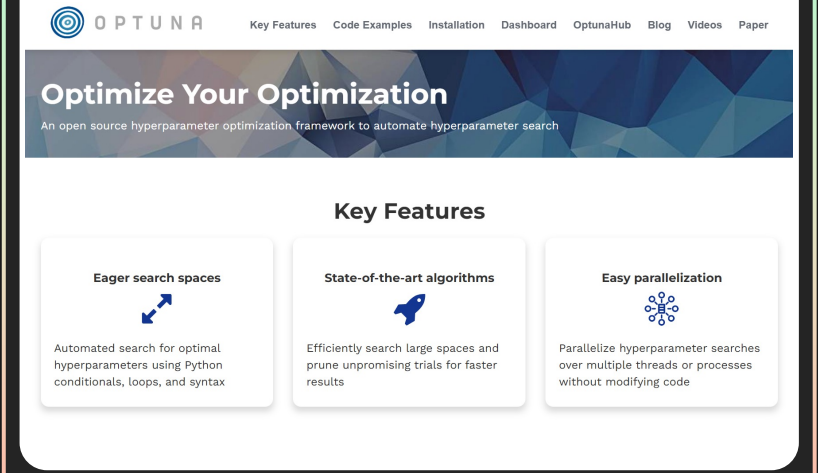
하이퍼파라미터 튜닝



하이퍼파라미터 튜닝



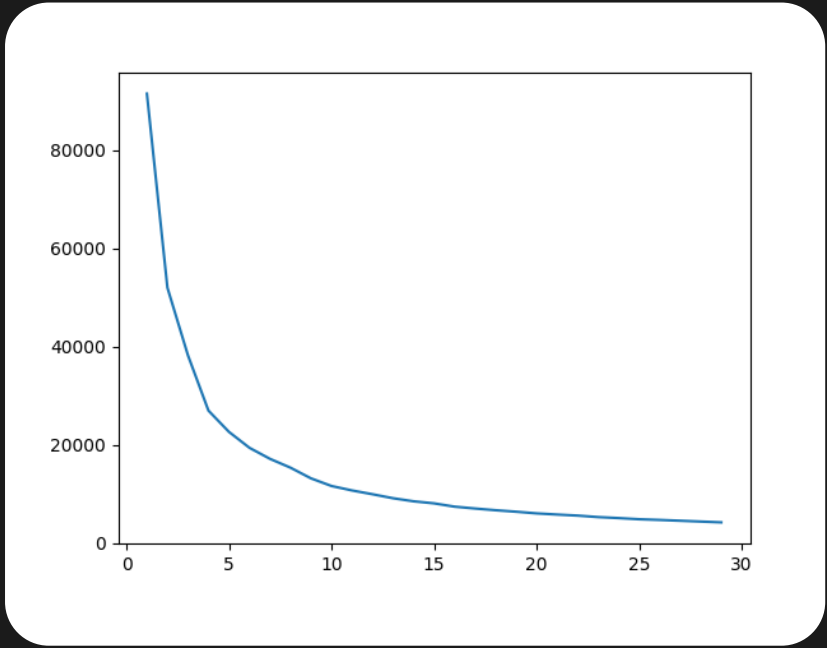
최적화 기법

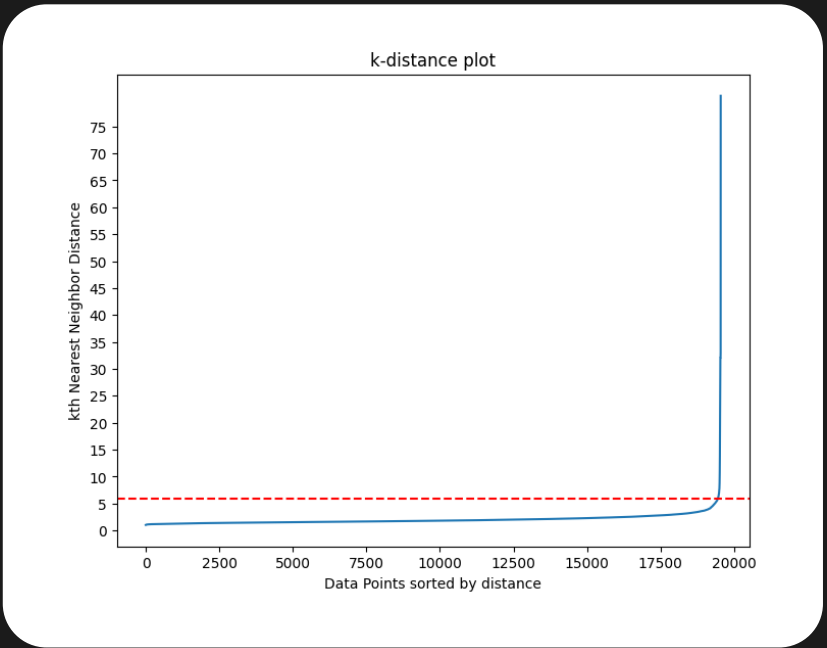


최적화 Library 사용

최적화 기법 (1)

Elbow Method





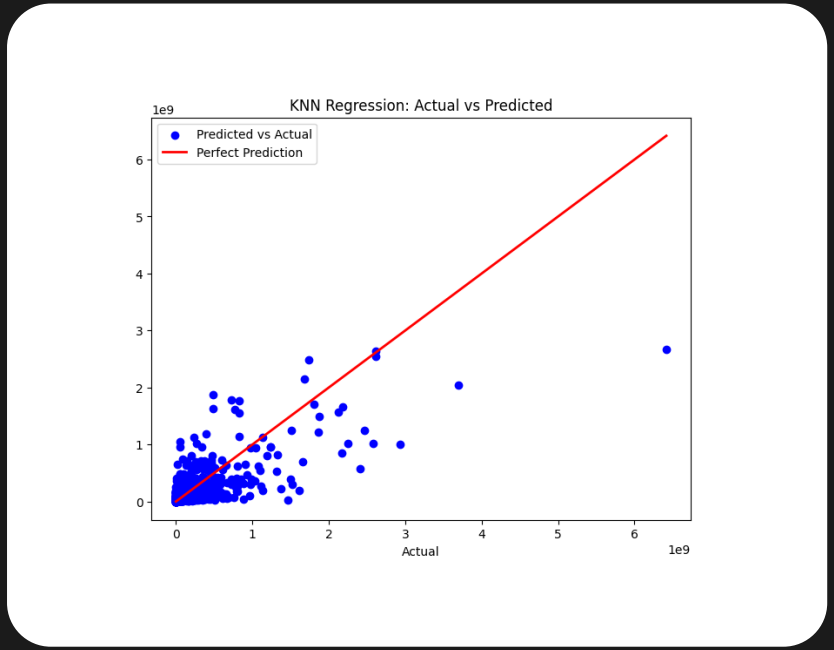
탐색하고자 하는 하이퍼파라미터 값을 증가시키면서 작업을 수행하고, 이를 그래프로 나타내어 최적해를 선택하는 방식

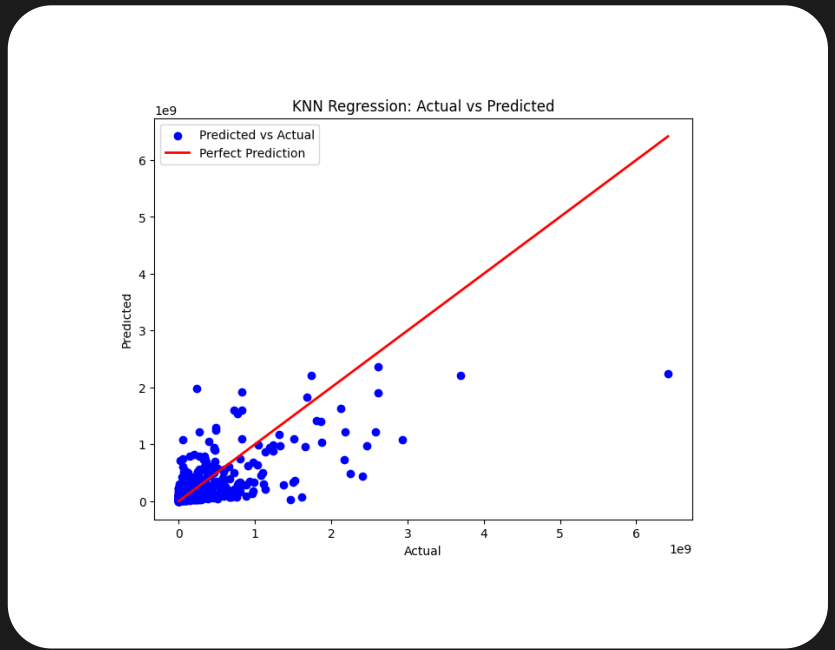
minPts = 100으로 설정한 후 k-거리 플롯을 수행, ε = 4 ~ ε = 7 범위에서 경사가 급격히 상승

최적의 값(ε = 6)을 채택

최적화 기법 (1)

Elbow Method





특성 소거를 통한 대조실험 (NRMSE, R-Squared)

좌: Danceability, Key, Instramentalness, Tempo 제거 (0.0366 / 0.5478)

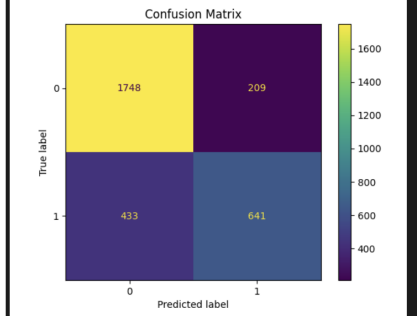
우: Danceability, Key, Valence, Instramentalness, Tempo 제거(0.0375 / 0.5255)

Random Forest

Accuracy: 0.7882

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.89	0.84	1957
1	0.75	0.60	0.67	1074
accuracy			0.79	3031
macro avg	0.78	0.75	0.76	3031
weighted avg	0.78	0.79	0.78	3031

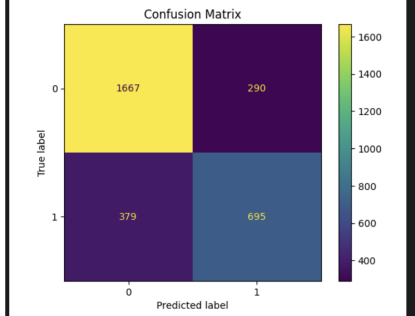


Gradient Boost

Accuracy: 0.7793

Classification Report:

	precision	recall	f1-score	support
0	0.81	0.85	0.83	1957
1	0.71	0.65	0.68	1074
accuracy			0.78	3031
macro avg	0.76	0.75	0.75	3031
weighted avg	0.78	0.78	0.78	3031

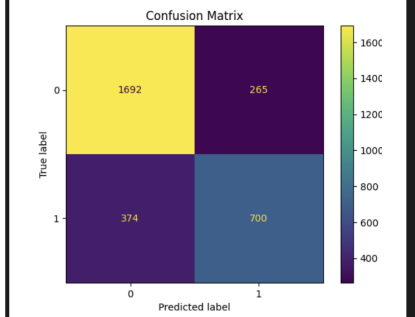


XG Boost

Accuracy: 0.7892

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.86	0.84	1957
1	0.73	0.65	0.69	1074
accuracy			0.79	3031
macro avg	0.77	0.76	0.76	3031
weighted avg	0.79	0.79	0.79	3031



최적화 기법 (2)

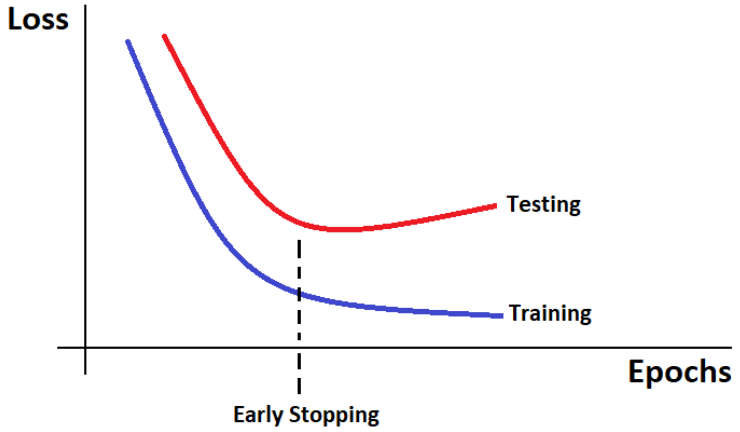
GridSearchCV

사용자가 직접 모델의 하이퍼파라미터의 값을 리스트로 입력

값에 대한 경우의 수마다 예측 성능을 비교하며 최적의 하이퍼파라미터 값을 찾는 과정을 진행

최적화 기법 (3)

Early Stopping



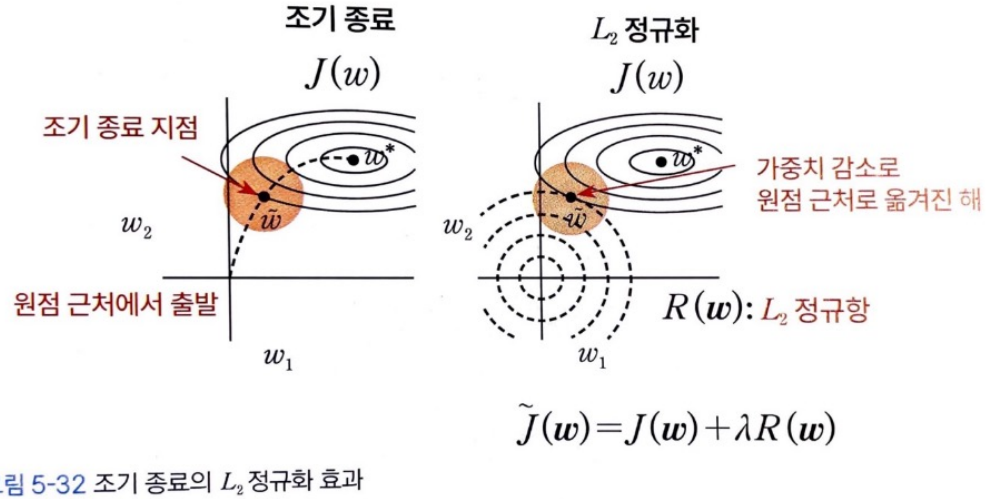


그림 5-32 조기 종료의 L_2 정규화 효과

과적합을 방지하기 위한 callback함수로, 적절한 시점에 학습을 조기종료하는 정규화 기법

별도의 API가 필요하지 않으며 L2 regularization과 유사한 효과

최적화 Library (1)

Optuna

```python
import optuna
from optuna.samplers import TPESampler
from sklearn.metrics import mean_squared_error
from pytorch_tabnet.tab_model import TabNetRegressor

# Objective 함수 정의
def objective(trial):
    # 탐색할 하이퍼파라미터 정의
    config = {
        "n_d": trial.suggest_int("n_d", 8, 64),
        "n_a": trial.suggest_int("n_a", 8, 64),
        "n_steps": trial.suggest_int("n_steps", 3, 10),
        "gamma": trial.suggest_float("gamma", 1.0, 2.0),
        "lambda_sparse": trial.suggest_float("lambda_sparse", 1e-6, 1e-3, log=True),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 1e-1, log=True),
        "max_epochs": 100,
        "batch_size": trial.suggest_categorical("batch_size", [64, 128, 256]),
        "virtual_batch_size": trial.suggest_categorical("virtual_batch_size", [32, 64, 128])
    }

    # TabNetRegressor 초기화
    model = TabNetRegressor(
        n_d=config["n_d"],
        n_a=config["n_a"],
        n_steps=config["n_steps"],
        gamma=config["gamma"],
        lambda_sparse=config["lambda_sparse"],
        optimizer_params=dict(lr=config["learning_rate"])
    )
```

```python
    # 모델 학습
    model.fit(
        X_train=X_train.values, y_train=y_train.values.reshape(-1, 1),
        eval_set=[(X_test.values, y_test.values.reshape(-1, 1))],
        eval_name=['val'],
        eval_metric=['rmse'],
        max_epochs=config["max_epochs"],
        patience=10,
        batch_size=config["batch_size"],
        virtual_batch_size=config["virtual_batch_size"],
        num_workers=0,
        drop_last=False
    )

    # 예측 및 성능 평가
    preds = model.predict(X_test.values)
    rmse = mean_squared_error(y_test.values.reshape(-1, 1), preds, squared=False)
    return rmse

# Optuna 스터디 생성 및 최적화 실행
study = optuna.create_study(direction="minimize", sampler=TPESampler())
study.optimize(objective, n_trials=50)

# 최적 하이퍼파라미터 출력
print("Best trial:")
trial = study.best_trial
print("  Value: ", trial.value)
print("  Params: ")
for key, value in trial.params.items():
    print(f"    {key}: {value}")
```

하이퍼파라미터를 자동으로 조정하고 최적화하는 오픈 소스 라이브러리

최적화 과정은 크게 트리 기반의 구조화된 파라미터 최적화(TPE)와 프루닝 메커니즘으로 구성

최적화 Library (1)

Optuna

Run history:

Final NRMSE	–
Final R2	–
Final RMSE	–
epoch	
val_rmse	

Run summary:

Final NRMSE	0.0385
Final R2	0.4252
Final RMSE	247234446.4438
epoch	99
val_rmse	247234446.44381

TabNetRegressor 개선 결과

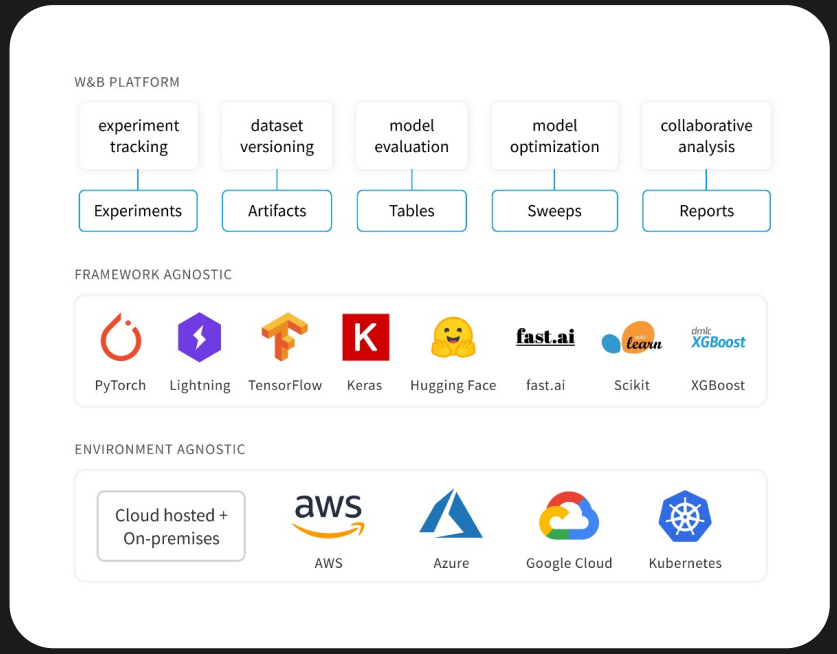
rmse: 247234446.4438

nrmse: 0.0385

R2: 0.4252

최적화 Library (2)

WandB




```
# wandB 로그인
wandb.login()
```

```
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: · · · · · · · · · · ·
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
True
```

```
# wandB 프로젝트 초기화
wandb.init(project="TabNET-classification-tuning")
```

```
wandb: Currently logged in as: studyinsogangu. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.18.5
Run data is saved locally in /content/wandb/run-20241106_074258-0g08lsl9
Syncing run classic-bird-3 to Weights & Biases (docs)
View project at https://wandb.ai/studyinsogangu/TabNET-classification-tuning
View run at https://wandb.ai/studyinsogangu/TabNET-classification-tuning/runs/0g08lsl9
Display W&B run
```

모델 학습과 동시에 학습 단계별로 모델 성능/손실함수 등을 추적하여, 최적의 하이퍼파라미터 조합을 찾아주는 도구

다음과 같이 WandB에 로그인을 하고, API key를 입력하여 세팅이 가능

최적화 Library (2)

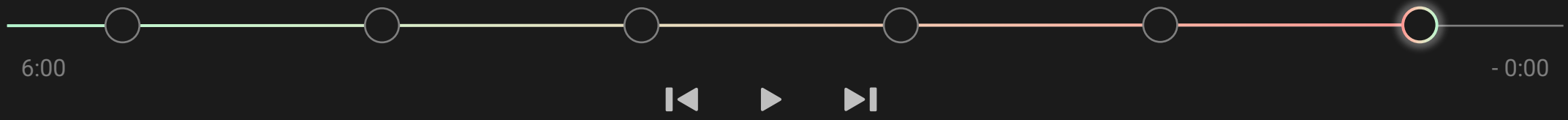
WandB

WandB Tracking을 통한 TabNet의 분류 작업 시행 및 결과 예시

06

In Conclusion

결론



01
탐구 의의

데이터 확보 및 처리를 통해 비교적 적은 연산량 및 자원으로도 준수한 퍼포먼스를 보이는 것에 **기여**
Transformer 모델과 다양한 최적화 방식을 통해 **정량적 분석**: 템포 및 키라는 구분점 추론
음악산업의 구조 변화 **재확인**, 음악 뿐만 아니라 **시각적 연출**이 조회수에 크게 **영향**

02
한계점

가설수립 및 변인통제 과정이 **미흡**
다양한 feature 간의 추상적 연관성은 파악하였으나 **새로운 인사이트를 발견하진 못함**
Learning rate scheduler, SHAP 등의 연구 도구에 대한 **숙련도 부족**

03
보완점

Danceability, Valence와 같은 정량적 수치가 조회수에 미치는 기제에 대한 **논리적 분석**
조회수 변화 추이에 영향을 미치는 새로운 음악적 분류 및 분석 기준을 **마련**

Thanks