산업수학 최종 보고서

<Spotify DB - YouTube 조회수 간의 상관관계 및 영향요인>

팀명: 써리원

20190311 손유림 (참여 비율 : 25%)

20202076 황대현 (참여 비율 : 25%)

20211245 설다현 (참여 비율 : 25%)

20221062 온재현 (참여 비율 : 25%)

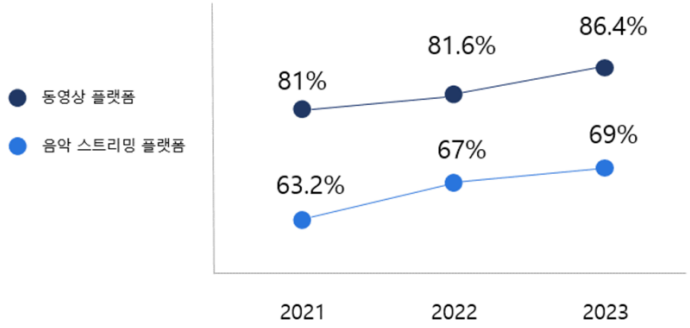
목차

1. 서론

다양한 플랫폼의 등장과 함께 소비자들의 음원 이용 방식은 과거와 달리 점차 다양해지고 있다. 테이프 및 CD와 같은 실물 매체에서 파일 및 스트리밍으로 그 형태가 달라졌지만, 스트리밍으로 음원 이용 방식이 완전히 넘어왔다고 평가받는 현재 시점에서도 그 방식은 계속해 변화하고 있다.

대표적으로, 음원 사이트가 아니라 동영상 플랫폼인 Youtube를 통해 음악을 감상하는 것은 이미 대중화 된 방식이다. 실제로, 방탄소년단의 곡 'Dynamite' 또한 Spotify에서 2022년 기준 16억회의 스트리밍 수를 기록했지만 Youtube에서도 Spotify 스트리밍 수의 약 4%에 해당하는 6200만의 조회수를 기록했다.

국내 소비자들이 음악을 감상하기 위해 이용하는 플랫폼을 조사한 결과를 보면 그 중요성은 더욱 크게 느껴진다. 한국콘텐츠진흥원이 2023년 11월 발표한 "2023 음악 이용자 실태조사"를 보면 2023년 음악 감상을 위해 동영상 플랫폼을 이용한다는 응답은 86.4%에 달했다. 이는 음악 감상을 위해 음악 스트리밍 플랫폼을 이용한다는 응답인 69%를 크게 웃도는 수치이다.



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

급성장하는 저작권 시장 속에서 음원이 차지하는 비율은 적지 않다. 이러한 상황에서 소비자들의 음원 소비 방식에 대한 이해는 콘텐츠 제작자들에게는 물론 음악 산업 전체에 발전에 있어서도 유용할 것으로 기대된다.

소비자들의 음원 소비 방식에 대해 분석하기 위해 음원 소비 방식의 두 개의 큰 갈래인 음원 스트리밍 서비스와 동영상 플랫폼을 기준으로 연구하기로 하였다. 이를 위해 세계 음원 스트리밍 시장의 30%를 차지하며 시장의 주도자로 평가받는 Spotify와 세계 최대 동영상 플랫폼인 Youtube를 대상으로 하였다. 이 두 업체가 제시한 데이터를 바탕으로 다양한 요소들에 다양한 머신러닝 및 딥러닝 기법을 적용해 "음악 데이터베이스와 조회수 간 상관관계 및 영향요인"을 분석하여 음악의 성공을 예측할 수 있는 모델 구축에 기여함과 동시에 대중의 음악 소비패턴에 대한 통찰을 얻어내고자 한다.

2. 데이터 전처리

2-1. 원본 데이터셋의 구조 및 데이터셋 생성

원본 데이터셋은 kaggle.com의 'Spotify and Youtube' 데이터셋이다. 데이터는 csv 형태로 저장되고 배포된다.

데이터셋은 Artist, Url_Spotify, Track, Album, Album_type, Uri, Danceability, Energy, Key, Loudness, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Tempo, Duration_ms, Url_youtube, Title, Channel, Views, Likes, Comments, Description, Licensed, official_video, Stream 의 27개 요소로 이루어져있다. 이러한 요소들을 비수치적 요소와 수치적 요소로 구분하고, 수치적 요소는 다시 1차적 요소, 2차적 요소로 구분하였다.

비수치적 요소란 Artist, Url_Spotify, Track, Album, Album_type, Uri, Url_youtube, Title, Channel, Description 과 같이 문자열로 이루어진 요소를 의미한다. 수치적 요소는 문자열이 아닌 수치로 이루어진 요소로 Danceability, Energy, Key, Loudness, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Tempo, Duration_ms, Views, Likes, Comments, Stream 등의 요소를 의미한다. 이 때, Licensed, official_video의 두 요소는 문자열로 이루어져 있지만 TRUE 또는 FALSE로 이루어져 있으므로 파이썬에서 dataframe 형태로 import 할 때 각각이 0과 1로 변환된다. 그러므로 본 항목에서는 Licensed, official_video는 수치적 요소로 구분하였다.

수치적 요소는 다시 1차적 요소와 2차적 요소로 구분된다. 1차적 요소는 Key, Loudness, Tempo, Duration_ms, Views, Likes, Comments, Stream와 같이 값 자체를 표현한 직접적이고 객관적인 수치이다. 2차적 요소는 Danceability(춤 추기에 적합한 트랙인지를 수치화), Energy(에너지가 높은 트랙인지를 수치화), Speechiness(트랙에서 말을 하는 정도를 수치화) , Acousticness(트랙이 어쿠스틱한지 정도를 수치화), Instrumentalness (트랙이 경음악으로 이루어져있는 정도를 수치화), Liveness(트랙이 음원 녹음 이외에도 청중 소리 등이 있는지를 통해 라이브일 가능성을 수치화), Valence(트랙이 긍정적인 정도를 수치화) 와 같이 스포티파이에서 음원 분석을 통해 그 정도를 수치화한 값들이다.

뮤직비디오와 음원의 상관관계를 알아보고자 하는 본 연구에서 뮤직비디오가 공식적으로 제작되었는지 여부는 그 자체가 큰 영향을 주기 때문에 초기 데이터셋 생성 중 official_video 열을 이용해 그 값이 TRUE(0)인 행만을 남겨두었다.

```
# pandas 를 이용해 csv 를 import 후 official_video 열이 TRUE 인 행만을 남기고 저장하는 코드
import pandas as pd
import numpy as np

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

데이터셋에 존재하는 Youtube 데이터와 현재 Youtube의 값을 비교하고, 존재하지 않는 열을 추가하기 위해 Youtube 데이터 크롤링을 진행하고자 하였다. 원본 데이터에는 존재하지 않았던 publishedAt이라는 열을 추가해 동영상이 업로드 된 시점을 받아오고, 조회수, 좋아요 수, 댓글 수는 기존 데이터셋의 생성 시점 (2024년 기준 약 2년 전)과 현재 시점의 값을 비교하고자 하였다. Youtube 데이터 크롤링을 위해 사용하는 API는 일일 10,000개의 요청 제한이 있다. 또한, 각 데이터당 두 번의 요청이 필요하기 때문에 csv 파일을 4,500개씩 분할하여 저장한다. 두 번의 요청이 필요한 이유는 조회수와 좋아요 수, 댓글 수는 statistics 영역에서 가져오고, 업로드 시점은 snippet 영역에서 가져오게 되는데, 각 영역에 대한 조회는 1회의 API 요청으로 처리되기 때문이다.

```
# pandas 를 이용해 csv 를 import 후 num 길이의 행으로 분리해서 저장하는 코드
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)
```

2-2. 데이터셋 크롤링 및 확장

데이터셋의 각 항목을 반복문으로 순회하며 크롤링을 진행하는 코드의 형태로, tqdm을 적용해 실행 중간에도 진행도 및 예상 잔여시간을 확인할 수 있도록 했으며, Youtube 영상이 삭제되어 있는 등의 에러에 대해서는 추가되는 데이터셋의 항목에 'Error'를 삽입하여 추후 데이터셋 이용에 문제가 없도록 하였다.

```
# 데이터셋 크롤링 및 확장 코드
import json
import requests
from tqdm import tqdm
import pandas as pd

file = input("csv directory: ")
api_key = '(API KEY)'
df = pd.read_csv(file)

# API 를 통해 데이터를 크롤링하는 함수 정의 코드
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]
```

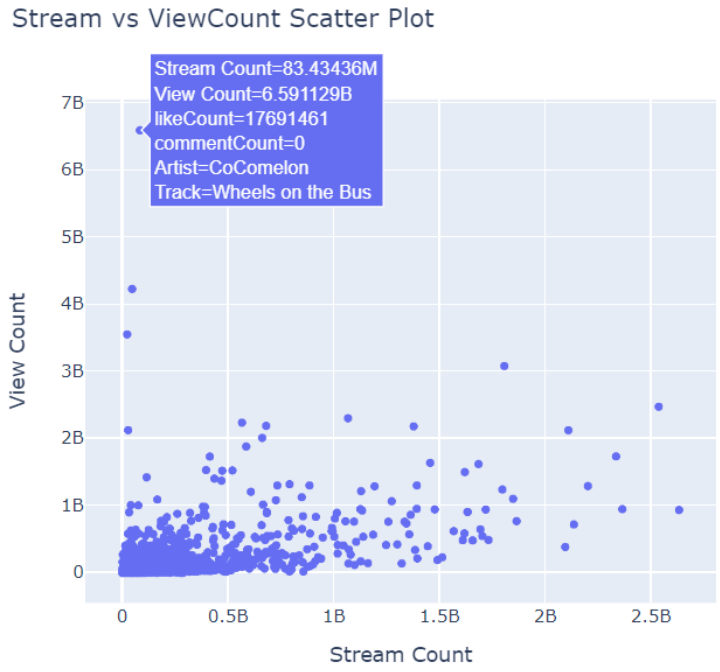
```
        except IndexError as f :
            data = dict({'publishedAt': 'Error', 'channelId': 'Error','viewCount': 'Error', 'likeCount': 'Error', 'commentCount': 'Error'})
            IndexerrorList.append(df['Unnamed: 0'][i])
        except KeyError as e :
            print(f'Error! Could not get {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
df.to_csv(f"{file.split('/')[-1]}_output.csv", index=False)
```

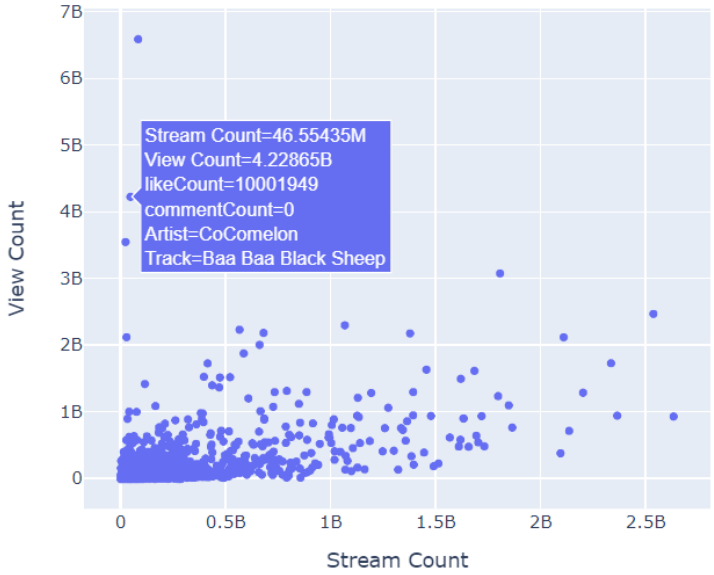
2-3. 이상치 및 결측치 처리

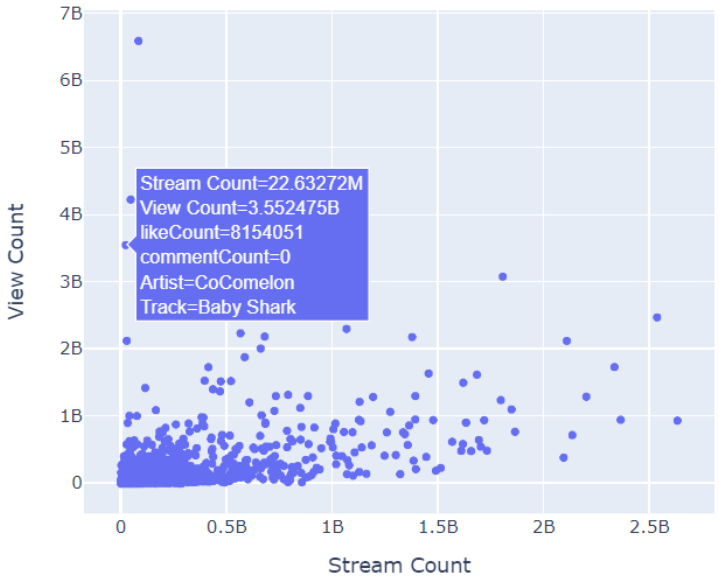
이후 API 사용량으로 인해 분리된 4개의 csv 파일을 하나로 합친 뒤, 크롤링한 결과물을 포함하여 하나로 합치는 작업을 진행하였다. 모든 파일을 읽어 하나의 데이터프레임으로 결합(concat)한 뒤, 그 결과를 1차적으로 저장하였다. 이후 저장된 csv 파일을 다시 pandas의 데이터프레임으로 읽어왔다.

```
# 데이터셋 결합 코드

import os
import pandas as pd

file_list = [(파일 리스트 목록)]
df_list = [pd.read_csv(file) for file in file_list]
df_combined = pd.concat(df_list, ignore_index = True)
df_combined.to_csv('combined_file.csv', index = False)
```

이후, 데이터셋 이용 전 기본적인 이상치와 결측치를 알아보기 위해 스포티파이 스트리밍 수와 Youtube 조회수 간의 scatter plot을 그려본 결과, 다음과 같은 이상치 데이터들이 발견되었다.







이러한 이상치 데이터들의 특징을 찾기 위해 각 점이 어떤 음악에 대응되는지 알아볼 수 있는 인터랙티브 산점도의 형태로 표현하여 분석을 진행하였다. 분석 결과, 동요나 아동용 음악 등의 경우 스포티파이 스트리밍 수보다 Youtube 조회수가 월등히 높았다. 이는 동요와 아동용 음악의 특성상 음원 스트리밍보다 Youtube를 통한 동영상 재생으로 재생되는 경우가 많다는 뜻이었고, 데이터셋 이용에 있어서 큰 영향을 줄 수 있다고 판단했다. 이 곡들의 Artist Name은 CoComelon으로 통일되어 있었기에 is_children 열을 추가하여 아동용 음악인지 여부를 one-hot encoding을 통해 Boolean 형태로 기록하였다.

```
# 동요 및 아동용 음악 여부 구분을 위한 열 추가 코드
df['is_children'] = df['Artist'].apply(lambda x : 1 if x == 'CoComelon' else 0)
```

동요 및 아동용 음악으로 구분된 데이터는 총 10개였다.

또한, 음악의 발매일자가 기존에 str형인 YYYY-MM-DD 구조로 되어 있었기에 연도를 int형 데이터로 이용하기 위해 publishedYear 열을 추가하였다.

```
# publishedYear 열 추가 코드
df['publishedYear'] = df['Artist'].apply(lambda x : 1 if x == 'CoComelon' else 0)
```

2-4. Feature Engineering

Feature Engineering이란 데이터셋에 대한 도메인 지식 등을 사용하여 딥러닝 및 머신러닝 등의 알고리즘이 효율적으로 작동할 수 있도록 하는 역할을 한다. 2-3의 이상치 및 결측치 처리도 그러한 Feature Engineering 기법 중 하나로 볼 수 있지만, 처리 된 데이터에 Feature Engineering을 적용한 것으로 생각하고 그 이후에 적용한 기법에 대해서 다루고자 한다.

Feature Engineering은 다시 크게 Feature Transformation과 Feature Interaction으로 나눌 수 있다. Feature Transformation은 단일 Feature에 대해 분포 및 스케일을 조정하는 과정 등을 의미하며, 정규화, 로그 변환 등을 이용하게 된다. Feature Interaction은 여러 개의 Feature에 대해 각 Feature 관계를 새롭게 추가하여 학습할 수 있도록 하는 기법이다.

Feature Engineering에 있어서는 회귀 작업에서 StandardScaler, MinMaxScaler 등을 사용하였다. StandardScaler는 표준화를 진행하는 Scaler로, 주어진 Feature를 평균이 0이고 분산이 1이 되도록 스케일링을 진행한다. 이렇게 스케일링이 진행된 변수는 가우시안 정규분포를 가지게 된다. MinMaxScaler는 정규화를 진행하는 Scaler로, 주어진 Feature 모든 값을 0과 1사이로 변환한다. 각 변수에서 최솟값을 빼고, 최댓값과 최솟값의 차이로 나누게 된다. 이러한 정규화를 진행하게 되면 모델의 학습과정에서 특정 변수에 많은 가중을 두어 학습하는 형태를 줄일 수 있다.

Feature Interaction 부분에 있어서는 새롭게 크롤링한 조회수, 좋아요 수, 댓글 수에 대해 증가량을 새로운 변수로 정의하였다.

```
# 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']
```

3. 클러스터링

본 탐구에서는 데이터의 음악적 요소 간 상관관계 및 영상 소비에 미치는 영향을 분석하고자 클러스터링 기법을 활용하였다. 이를 위해 타겟 변수를 Youtube 영상 조회수로 설정하였으며, 나머지 특성(feature) 중 음악 관련 feature(Danceability, Acousticness, Loudness 등)을 독립 변수로 설정하였다. 이때 원활한 군집화 수행을 위해 데이터 정규화(Dimensionality Reduction)를 수행함으로써, 특정 변수가 과도하게 영향을 미치는 것을 방지하였다.

또한 차원 축소 기법인 PCA(Principal Component Analysis)와 T-SNE 기법을 활용해, 데이터의 분산을 최대한 유지하면서 2차원으로의 변환을 시도하였다. 이를 통해 차원의 저주(curse of dimensionality) 문제를 해결하였다. 차원의 저주는 데이터의 차원이 증가함에 따라 데이터의 밀도가 희소해지고, 이로 인해 과적합(overfitting) 위험이 커지는 것을 의미한다. 머신러닝에서는 이런 문제를 피하기 위해 차원 축소를 시도할 수 있다. 이러한 차원 축소 기법은 크게 특징 제거(Feature Elimination), 특징 선택(Feature Selection), 특징 추출(Feature Extraction)로 나눌 수 있다.

PCA와 t-SNE 모두 특징 추출에 해당하는 차원 축소 기법이다. PCA(Principal Component Analysis)는 고차원 데이터의 분산을 최대한 보존하면서 차원을 축소하기 위한 선형 기법이다. 이 방법은 데이터의 공분산 행렬을 기반으로 주성분(Principal Components)을 계산하며, 이러한 주성분은 데이터의 분산이 최대화되는 방향을 나타낸다. PCA는 특이값 분해(SVD)를 사용하여 데이터의 주요 축을 정의하며, 각 축은 직교성을 가진다. 해당 기법은 데이터의 전반적인 구조를 파악하는 데 적합하며, 데이터의 주요 변동성을 설명하는 축을 따라 차원을 축소함으로써 노이즈 제거와 계산 효율성을 높이는 데 기여한다.

한편, t-SNE(t-Distributed Stochastic Neighbor Embedding)는 데이터의 국소적 구조를 강조하여 시각적으로 표현하는 데 특화된 비선형 차원 축소 기법이다. t-SNE는 고차원 데이터에서 두 데이터 포인트 간 유사성을 확률적으로 계산한 후, 이를 저차원 공간에서도 유사하게 유지하도록 매핑한다. 이 과정에서 고차원의 확률 분포와 저차원의 확률 분포 간의 쿨백-라이블러 발산(Kullback-Leibler divergence)을 최소화하는 최적화 알고리즘을 사용한다. 이러한 특성 덕분에 t-SNE는 데이터의 군집화와 같은 국소적 패턴을 명확히 시각화하며, 특히 데이터의 이웃 관계를 강조하는 데 유용하다. 다만, 전체적인 데이터 구조는 왜곡될 가능성이 있고, 계산 비용이 높다는 단점이 있다.

PCA는 데이터의 선형적 특성과 전반적인 분산을 효과적으로 유지하며, 데이터의 전체 구조를 분석하는 데 적합하다. 반면, t-SNE는 비선형적 관계를 강조하며, 국소적 군집과 데이터 포인트 간의 근접성을 시각적으로 탐색하는 데 유리하다. 본 탐구에서는 모델의 특성을 고려하여 두 기법을 활용하고, 데이터의 패턴과 관계를 다각적으로 조망하고자 하였다. 이를 통해 음악적 요소와 영상 소비의 연관성을 더욱 깊이 이해하며, 차원 축소 기법이 데이터 분석과 시각화에서 제공할 수 있는 다양한 관점을 실증적으로 검토하였다.

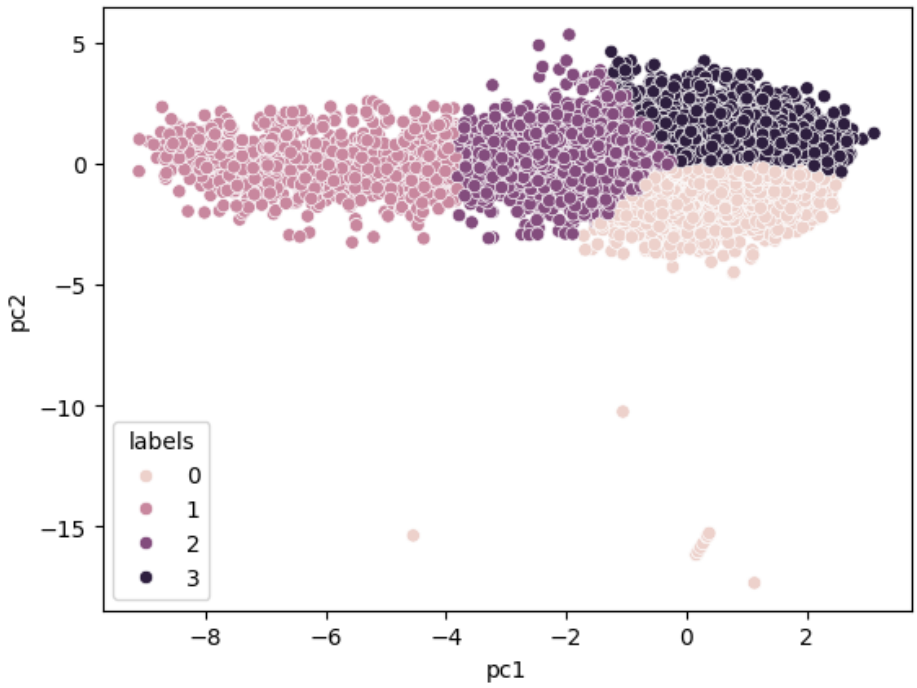
3-1. 머신러닝 기법을 활용한 방식

3-1-1. K-Means

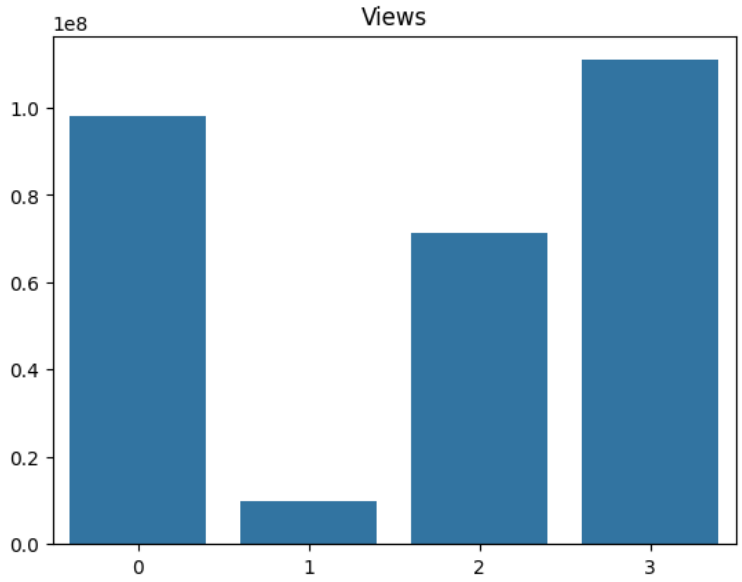
K-means 군집화 기법은 중심 기반 알고리즘으로, 사전에 설정한 클러스터 개수(K)를 기준으로 데이터 포인트를 K개의 클러스터로 나누는 방식이다. 해당 알고리즘은 다음과 같은 과정을 거쳐 군집을 형성한다.

1. 초기화: 처음에는 K 개의 클러스터 중심(centroids)을 무작위로 설정한다. 이는 K 개의 랜덤한 데이터 포인트를 선택하거나, 특정 알고리즘에 따라 초기 중심점을 지정할 수 있다.
2. 할당: 각 포인트와 중심 간의 유클리드 거리를 계산하여, 가장 가까운 중심에 속하는 클러스터로 포인트를 배정한다.
3. 계산 과정 반복: 각 클러스터에 속한 데이터 포인트들의 평균값을 계산하여, 클러스터 중심을 업데이트한다. 중심이 더 이상 변하지 않을 시 알고리즘이 수렴하며, 최종적으로 K 개의 클러스터가 형성된다.

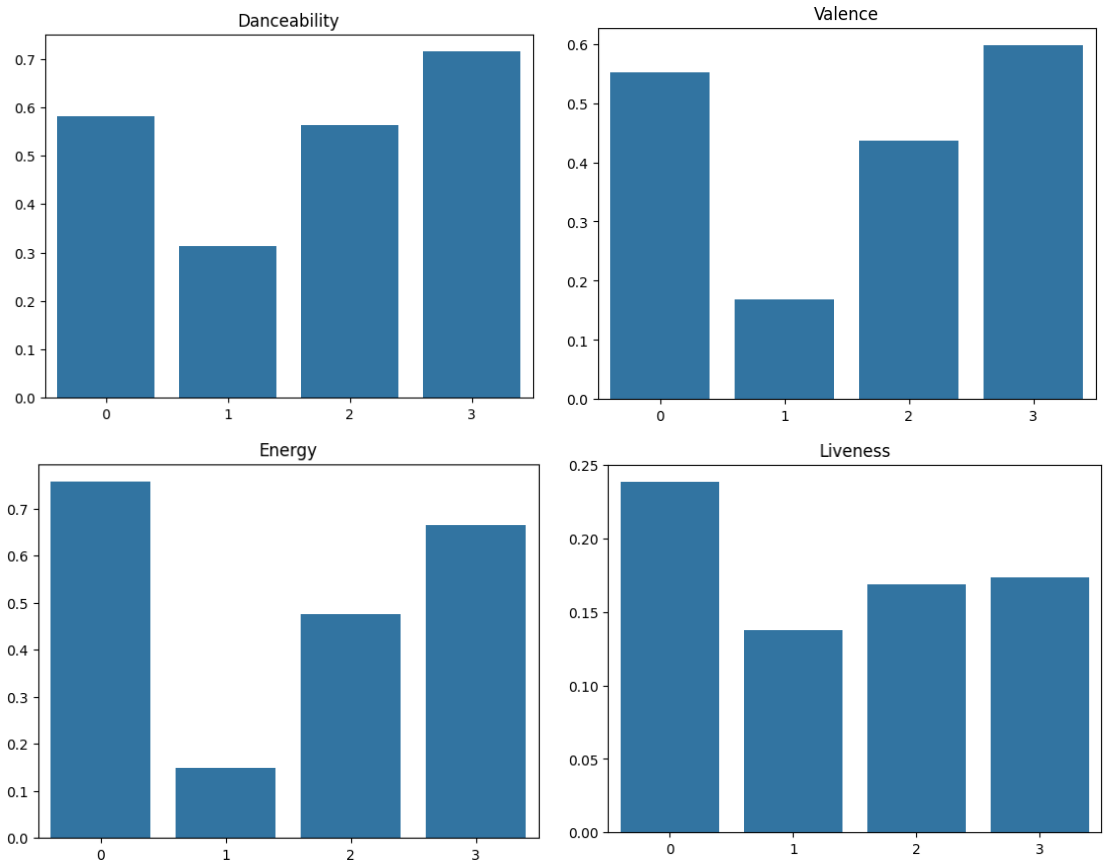
K = 4로 설정한 후 군집화를 수행하고, 이를 시각화한 결과는 다음과 같았다. 일부 이상치를 제외하고, 대체로 군집 간 분리가 잘 이루어지는 것을 확인할 수 있었다.



군집별 영상 조회수 (Views) 평균을 시각화한 결과는 다음과 같았다.



또한, 음악 관련 특성(feature)의 군집별 평균을 시각화한 결과는 다음과 같았다.



분석 결과, Views와 그래프 개형이 가장 유사한 것은 Danceability·Valence였다. 이외에도 Energy·Liveness가 서로 유사한 그래프 개형을 보이고 있었으며, 이러한 요소들 역시 조회수와 유의미한 상관관계를 보인다고 판단하였다.

Views와 Danceability의 상관관계의 경우, Danceability의 속성이 음악의 장르와 밀접한 관련이 있다는 점이 영향을 미쳤을 것이라고 판단하였다. 가령, EDM(Electronic Dance Music)이나 펑크(Funk), Pop 등의 음악은 일반적으로 Danceability가 높은 축에 속한다. 해당 장르 음원의 MV는 대체로 화려한 영상미를 바탕으로 시각적 몰입감을 중대시킨다는 특징을 가진다.

Valence의 경우 Danceability와 양의 상관관계를 가진다고 판단하였다. 대체로 Danceability가 높은 장르의 음악은 밝고 긍정적인 감정을 자극하는 경우가 많기 때문이다. Energy·Liveness의 속성이 서로 밀접하게 연결되어 있는 이유 역시 음악의 정서 및 리듬감은 생동감, 역동성 등과 유의미한 상관관계를 형성하기 때문인 것으로 해석하였다.

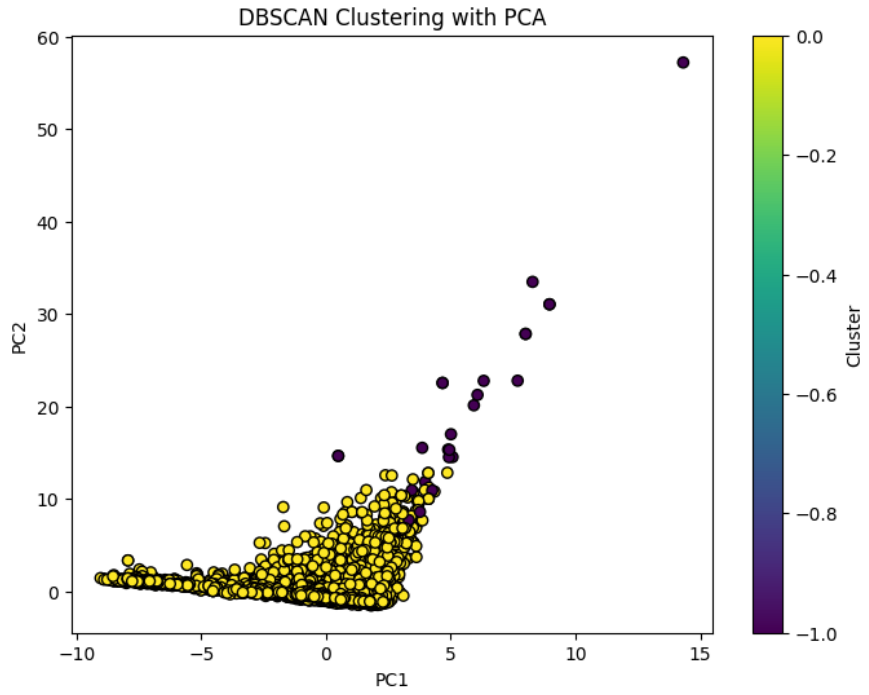
3-1-2. DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)은 밀도 기반 클러스터링 알고리즘으로, 데이터 포인트들이 얼마나 밀집되어 있는지를 기준으로 클러스터를 형성한다. 이는 클러스터의 형태가 구형이 아닌 경우에도 효과적으로 군집화할 수 있으며, K-means와 달리 클러스터 개수를 사전에 설정할 필요가 없다는 이점을 가진다. 해당 알고리즘은 다음과 같은 과정을 거쳐 군집을 형성한다.

1. 영역 정의: DBSCAN 은 두 가지 주요 하이퍼파라미터를 사용하여 밀집된 영역을 정의한다.
   - ε : 클러스터를 구성하는 최소 거리
   - minPts: 한 클러스터 내에 포함되어야 할 최소 데이터 포인트 개수

2. 포인트 분류: 위 하이퍼파라미터를 기준으로, 데이터 포인트를 세 가지로 분류한다.
   - 핵심 포인트(Core Point): ε 반경 내에 minPts 이상 데이터 포인트가 포함되면, 해당 포인트는 핵심 포인트로 간주된다.
   - 경계 포인트(Border Point): ε 반경 내에 minPts 미만의 포인트가 있더라도, 해당 포인트가 핵심 포인트의 ε 반경 내에 있으면 경계 포인트로 간주된다.
   - 노이즈(Noisy Point): 핵심 포인트도 아니고, 경계 포인트에도 속하지 않는 포인트는 노이즈로 간주된다.

3. 클러스터 형성: 밀집된 핵심 포인트를 서로 연결하여 클러스터를 형성한다. 이때 경계 포인트는 핵심 포인트에 속하는 클러스터에 포함되며, 새로운 클러스터를 형성하지 않는다. 또한, 밀도가 낮은 데이터 포인트는 노이즈로 분류되며, 이를 통해 데이터의 이상치를 알고리즘 내부에서 자동으로 처리한다.



클러스터링 시각화 결과를 확인한 결과, K-means대비 군집 구분이 명확하지 않음을 확인할 수 있었다. 앞서 언급한 바와 같이, 본 탐구에서는 이러한 결과의 원인을 데이터셋의 특성에서 찾았다. 음악 데이터는 본질적으로 연속적이고 복잡한 양상을 가지며, 이를 구성하는 요소들은 서로 밀접하게 연결되어 있는 경우가 많아 데이터 간 차이가 미세한 경우가 많다. 이에 따라, 데이터 간 명확한 밀도 차이를 찾는 것이 어려워, 밀도 기반 클러스터링 알고리즘이 본 데이터셋에는 적합하지 않았을 것이라는 결론을 내렸다.

3-2. 딥러닝 기법을 활용한 방식

3-2-1. Self-Supervised MLP와 K-Means를 활용한 클러스터링

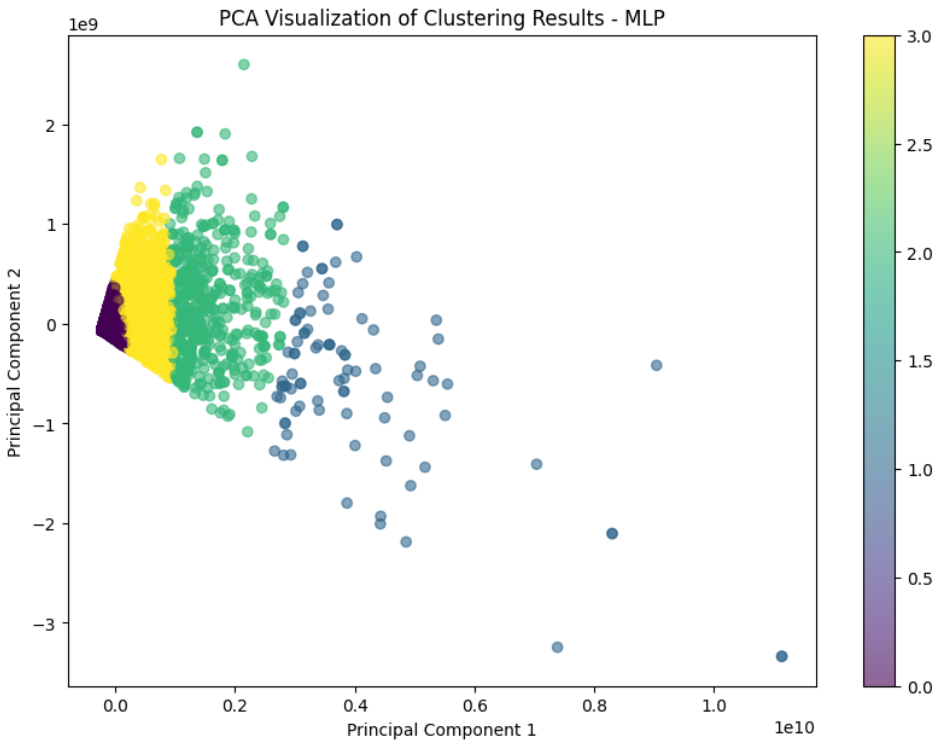
본 탐구에서는 자기 지도 학습(Self-Supervised Learning) 기반 MLP(Multi-Layer Perceptron) 모델과 비지도 학습 군집화 기법인 K-Means를 결합하여 데이터의 특징을 학습하고 군집화하였다. 이후 상술하였던 PCA와 t-SNE 기법을 통해 시각화를 수행하였다.

다층 퍼셉트론을 알고리즘으로 구현한 코드는 다음과 같다.

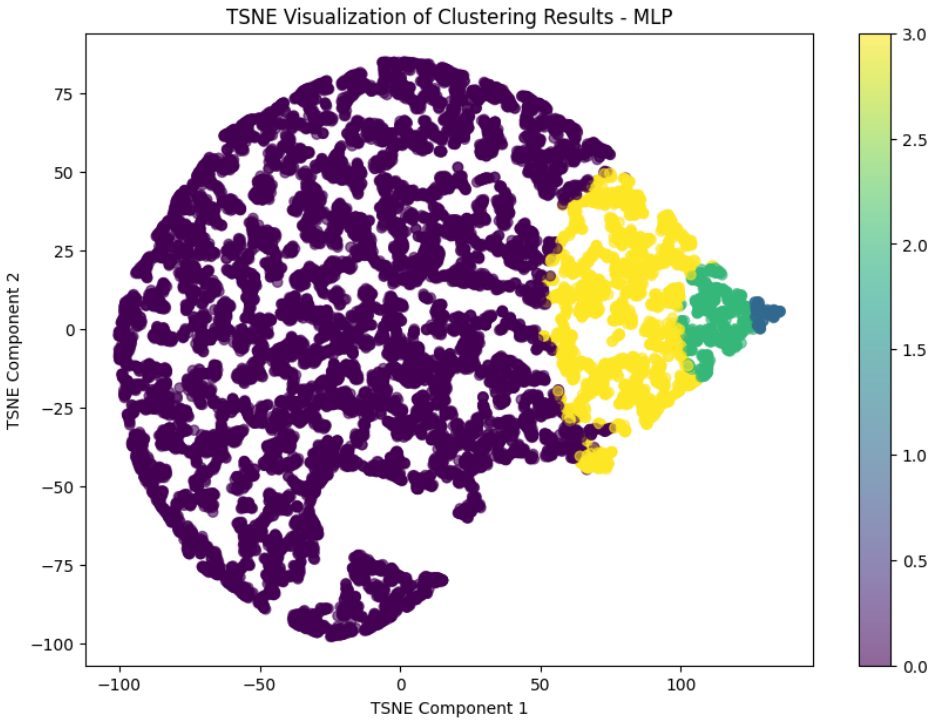
```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim=128):
        super(MLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 64),
            nn.ReLU(),
            nn.Linear(64,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

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

해당 객체는 입력 데이터의 차원(input_dim)을 시작으로, 128개의 은닉 뉴런(hidden_dim)과 64차원을 거친 후 다시 원래 차원으로 복원하는 구조를 가진다. 각 계층은 완전 연결층(nn.Linear)과 활성화 함수 ReLU(nn.ReLU)로 구성되어 있으며, 이를 통해 비선형적 데이터 관계의 효과적인 학습을 도모하였다. 또한, forward 메서드를 통해 순전파 신경망을 구현하였는데, 입력 데이터가 네트워크의 각 계층(완전 연결층과 활성화 함수)을 순차적으로 통과하도록 정의하였다. 이를 통해 데이터는 각 계층에서 선형 변환과 비선형 변환을 반복적으로 거치며, 모델이 입력 데이터의 숨겨진 패턴과 관계를 학습할 수 있도록 하였다.

해당 모델로 클러스터링을 수행하였을 때, PCA와 t-SNE를 활용한 시각화 결과는 각각 다음과 같았다. PCA 시각화는 데이터의 전반적인 분포와 전반적인 구조를 효과적으로 표현하였고, 군집의 응집도 또한 상대적으로 높은 편이었다. 한편, t-SNE는 데이터의 국소적 구조를 강조하여 군집 내부의 밀집도와 경계를 명확히 보여주었으며, 특정 군집이 분리된 형태로 나타나 군집화 결과를 시각적으로 확인하는 데 비교적 유리하다는 이점을 가지고 있었다.



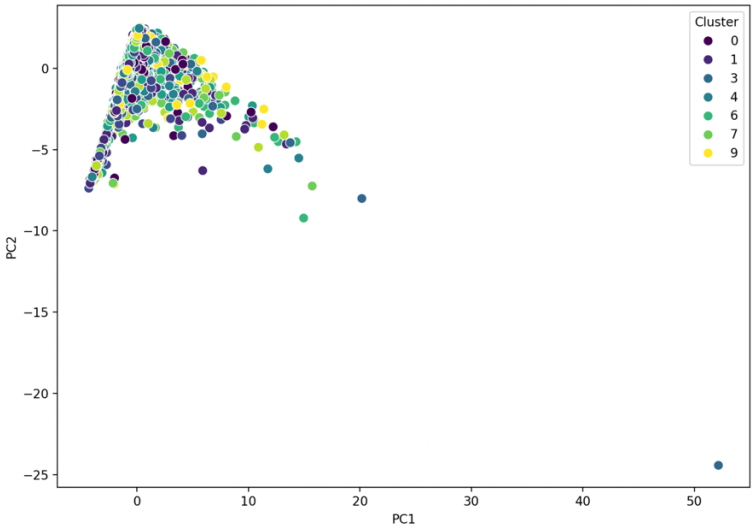
<PCA Visualization of MLP Clustering>



<t-SNE Visualization of MLP Clustering>

3-2-2. VAE

클러스터링에 있어서 딥러닝을 활용하기 위한 방법으로 VAE를 이용하고자 시도하였다. VAE란, Variational autoencoder의 준말로 generative model의 일종이다. Encoder를 통해 input을 저차원의 벡터로 인코딩 후, 이 벡터를 다시 input으로 Decoding하는 것을 학습하게 된다. 클러스터링에 이를 활용할 때, 데이터를 저차원으로 변환하는 과정에서 의미 손실을 줄임으로서 데이터 간 관계를 저차원의 벡터가 잘 보존하도록 할 수 있다. 또한, 노이즈가 포함된 데이터를 통해서도 저차원의 벡터로 이루어진 잠재 공간을 안정적으로 형성할 수 있도록 하고, 그러한 잠재 공간에서 새로운 데이터를 생성할 수 있다.



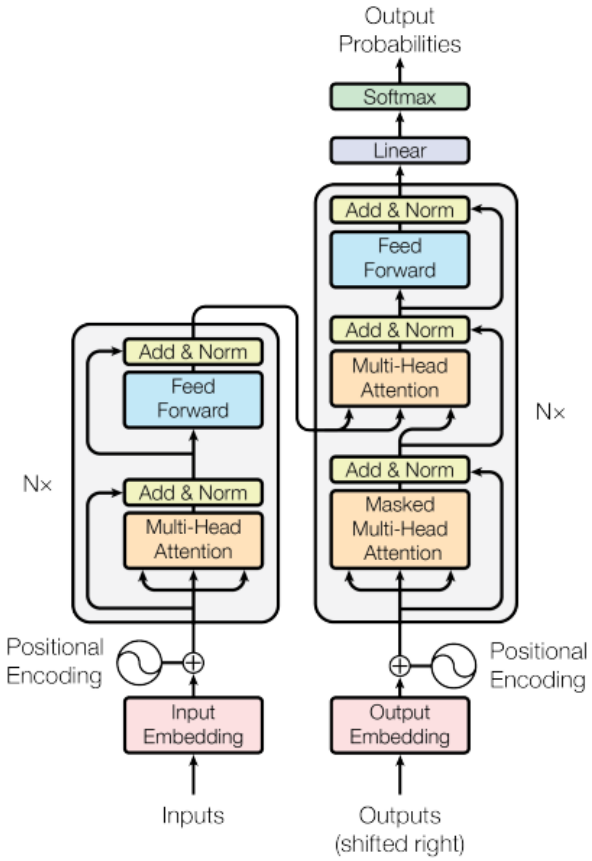
PCA를 통해 VAE로 클러스터링 한 결과이다. MLP에서의 클러스터링과 달리, 데이터들이 그룹화 되지 못하고 모여있고 색상을 보았을 때도 클러스터링 또한 제대로 되지 않은 모습을 볼 수 있다.

이를 통해, 기존 진행하였던 머신 러닝 (MLP)를 이용한 클러스터링에 비해 VAE를 이용한 클러스터링의 성능이 기말 과제 현재까지는 부족함을 알 수 있었다.

4. 회귀

본 탐구에서는 회귀 task를 통해 음악적 특성, 음원 스트리밍 횟수, 영상 조회수 및 인지도 간의 상관관계를 분석하고자 하였으며, 이를 위해 다양한 방식을 시도하였다. 선형 회귀 알고리즘 중 하나인 OLS Regression과, 정규화를 통해 Bias를 줄인 Ridge와 Lasso Regression 방식을 활용하여, 음원의 스트리밍 횟수와 조회수, 좋아요 수, 댓글 수 등의 회귀 분석을 진행하였다. 또한 KNN Regression을 통해 음악적 요인과 뮤직비디오 조회수 간 상관관계를 분석하고자 하였다.

한편, feature 간 상관관계를 보다 다각적으로 검토하기 위해, 회귀 task에도 딥러닝 기법을 적용하였다. 특히, Transformer의 Attention 메커니즘을 모델에 적극적으로 도입하였다. Transformer는 'Attention is All You Need (2017)'에서 소개된 모델 아키텍쳐[1]로, 논문 제목에서 알 수 있듯 Attention 메커니즘을 기반으로 데이터를 처리하는 것이 가장 큰 특징이다.



[1] **Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I.** (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 5998–6008. https://arxiv.org/abs/1706.03762

Attention은 입력 데이터 내에서 중요한 부분에 더 높은 가중치를 부여하여, 모델이 데이터의 상호작용과 관계를 학습할 수 있도록 돕는다. 특히, Self-Attention은 데이터의 모든 요소가 서로에게 영향을 주며, 특정 위치나 순서에 의존하지 않고 병렬 처리를 가능하게 한다. 이러한 구조는 긴 종속성을 효과적으로 학습하며, 복잡한 데이터의 특징을 파악하는 데 유리하다. 따라서 시계열 데이터, NLP 등 연속적인 데이터셋을 활용하는 분야에서 주로 사용되어 왔다.

그러나 근래에는 정형 데이터셋을 위한 딥러닝 모델에 Transformer을 적용하는 시도 또한 활발히 이루어지고 있다. Attention 메커니즘을 통해 feature 간 관계를 학습하고, 복잡한 비선형 관계를 효과적으로 모델링할 수 있기 때문이다. 특히, 기존의 Gradient Boosting 모델이나 단순 신경망과 비교했을 때, Transformer는 데이터 간의 복잡한 의존성을 학습하여 더 높은 예측 성능을 제공할 가능성이 높다. 또한, 범주형 데이터와 연속형 데이터를 모두 효과적으로 처리할 수 있다는 이점을 가진다. 이에 착안하여, 본 탐구에서는 TabNET[2] 및 TabTransformer을 회귀 task에 활용하였는데, 이 두 모델은 모두 Attention 메커니즘을 도입하면서 정형 데이터셋의 처리에 특화되어 있다.

4-1. 머신러닝 기법을 활용한 방식

4-1-1. OLS Regression

OLS(최소자승법)를 활용한 단순 선형 회귀 알고리즘부터 적용해보았다. 우선 통계학에서 선형회귀(Linear regression)는 종속 변수 y와 한 개 이상의 독립 변수 X와의 선형 상관관계를 모델링하는 회귀분석 기법이다. 최소 자승법(OLS, Ordinary Least Square)이란, 어떤 데이터가 주어졌을 시 최적의 추세선을 그리기 위한 방법 중 하나이다. 즉, 오차의 제곱을 최소화하는 회귀식을 추정하는 방식이다.

The Least Squares Method

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

단순 선형 회귀에서의 회귀식 추정은 다음과 같다.

$$\frac{\partial SSE}{\partial \beta} = 2\sum(y_i - \beta - \alpha x_i)(-1) = 0$$

$$\frac{\partial SSE}{\partial \alpha} = 2\sum(y_i - \beta - \alpha x_i)(-x_i) = 0$$

[2] 해당 모델은 후술할 분류 task에서도 활용된다.

$$\hat{\alpha} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sum x_i^2 - n\bar{x}^2} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

$$\hat{\beta} = \frac{\sum y_i - \alpha \sum x_i}{n}, \ \hat{\beta} = \bar{y} - \hat{\alpha}\bar{x}$$

다중 선형 회귀의 경우, 결과변수 y에 영향을 주는 원인변수 X가 여러 개인 것이므로 다음과 같이 행렬로 구해진다.

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} \cdots x_{1k} \\ 1 & x_{21} & x_{22} \cdots x_{2k} \\ \vdots & \vdots & \vdots \quad \vdots \\ 1 & x_{n1} & x_{n2} \cdots x_{nk} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

$$\begin{aligned} SSE &= (Y - X\beta)'(Y - X\beta) \\ &= Y'Y - Y'X\beta - \beta'X'Y + \beta'X'X\beta \\ &= Y'Y - 2\beta'X'Y + \beta'X'X\beta \end{aligned}$$

$$\frac{\partial SSE}{\partial \beta} = 2(X'X)\hat{\beta} - 2(X'Y) = 0$$

$$\therefore \hat{\beta} = (X'X)^{-1}X'Y$$

이와 같은 OLS Regression을 데이터에 직접 적용하였다.

```
# 데이터 타입 확인 및 변환
import pandas as pd
import numpy as np

# Stream, viewCount, likeCount, commentCount 열의 데이터 타입 확인
print(df[['Stream', 'viewCount', 'likeCount', 'commentCount']].dtypes)

# 숫자형 데이터로 변환 (변환할 수 없는 값은 NaN 으로 처리)
df['Stream'] = pd.to_numeric(df['Stream'], errors='coerce')
df['viewCount'] = pd.to_numeric(df['viewCount'], errors='coerce')
df['likeCount'] = pd.to_numeric(df['likeCount'], errors='coerce')
df['commentCount'] = pd.to_numeric(df['commentCount'], errors='coerce')

# Stream, viewCount, likeCount, commentCount 열의 데이터 타입 확인
print("₩n")
print(df[['Stream', 'viewCount', 'likeCount', 'commentCount']].dtypes)
```

```
# 실행 결과
Stream              float64
viewCount            object
likeCount            object
commentCount         object
dtype: object

Stream              float64
viewCount           float64
likeCount           float64
commentCount        float64
dtype: object
```

위와 같이 데이터 타입을 먼저 확인해본 후, 회귀를 위해 숫자형 데이터로의 변환을 거쳤다. 이후 결측치를 확인한 후 제거하였다.

```
# 결측치 확인
print(df[['Stream', 'viewCount', 'likeCount', 'commentCount']].isnull().sum())
# 결측치 제거 (결측치가 있는 행을 모두 제거)
df_clean = df.dropna(subset=['Stream', 'viewCount', 'likeCount', 'commentCount'])
```

최종적으로 OLS 회귀를 적용한 코드와 결과이다.

```
import statsmodels.api as sm

# 독립 변수와 종속 변수를 설정
X = df_clean['Stream']
y_view = df_clean['viewCount']
y_like = df_clean['likeCount']
y_comment = df_clean['commentCount']

# 상수항 추가
X = sm.add_constant(X)

# viewCount 에 대한 회귀 분석
model_view = sm.OLS(y_view, X).fit()
print(model_view.summary())
```

스트리밍 횟수와 뮤직비디오의 조회수, 좋아요수, 댓글수 각각을 y로 나타내어 각각의 회귀 분석을 진행하였다.

다음은 Spotify 스트리밍 횟수와 뮤직비디오 조회수의 관계를 OLS 회귀 분석한 결과이다.

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

다음은 Spotify 스트리밍 횟수와 뮤직비디오 좋아요 수 관계를 OLS 회귀 분석한 결과이다.

```
# likeCount 에 대한 회귀 분석
model_like = sm.OLS(y_like, X).fit()
print(model_like.summary())
```

OLS Regression Results

Dep. Variable:	likeCount	R-squared:	0.461
Model:	OLS	Adj. R-squared:	0.461
Method:	Least Squares	F-statistic:	1.299e+04
Date:	Sat, 12 Oct 2024	Prob (F-statistic):	0.00
Time:	00:07:05	Log-Likelihood:	-2.3804e+05
No. Observations:	15158	AIC:	4.761e+05
Df Residuals:	15156	BIC:	4.761e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	8.723e+04	1.5e+04	5.830	0.000	5.79e+04	1.17e+05
Stream	0.0056	4.92e-05	113.965	0.000	0.006	0.006

Omnibus:	21569.566	Durbin-Watson:	1.527
Prob(Omnibus):	0.000	Jarque-Bera (JB):	15441473.248
Skew:	8.137	Prob(JB):	0.00
Kurtosis:	158.512	Cond. No.	3.50e+08

마지막으로, Spotify 스트리밍 횟수와 뮤직비디오 좋아요 수 관계를 OLS 회귀 분석한 결과이다

```
# commentCount 에 대한 회귀 분석
model_comment = sm.OLS(y_comment, X).fit()
print(model_comment.summary())
```

OLS Regression Results

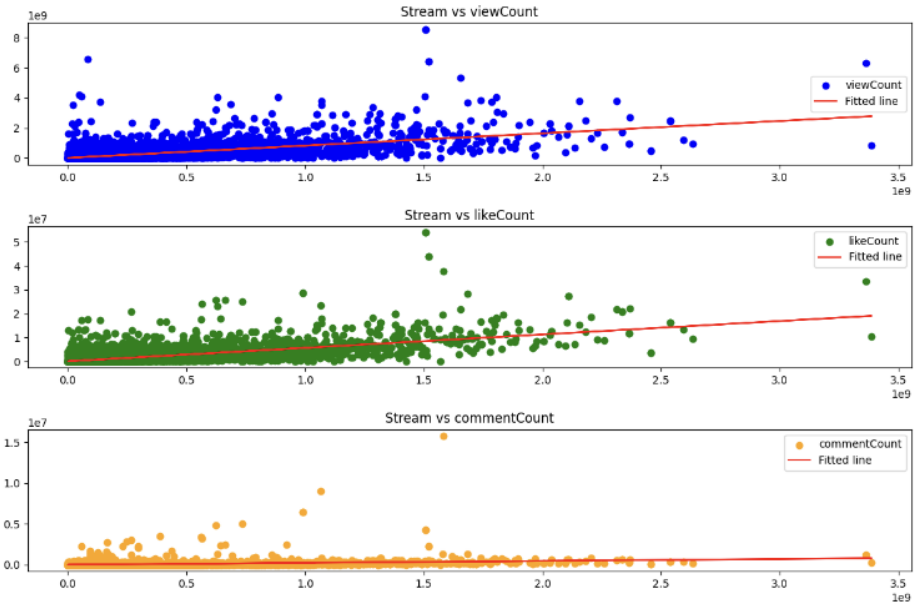
Dep. Variable:	commentCount	R-squared:	0.082
Model:	OLS	Adj. R-squared:	0.082
Method:	Least Squares	F-statistic:	1346.
Date:	Sat, 12 Oct 2024	Prob (F-statistic):	1.74e-282
Time:	00:07:05	Log-Likelihood:	-2.0708e+05
No. Observations:	15158	AIC:	4.142e+05
Df Residuals:	15156	BIC:	4.142e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	271.3415	1941.050	0.140	0.889	-3533.350	4076.033
Stream	0.0002	6.39e-06	36.694	0.000	0.000	0.000

Omnibus:	44260.587	Durbin-Watson:	1.530
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3701863234.435
Skew:	40.570	Prob(JB):	0.00
Kurtosis:	2422.640	Cond. No.	3.50e+08

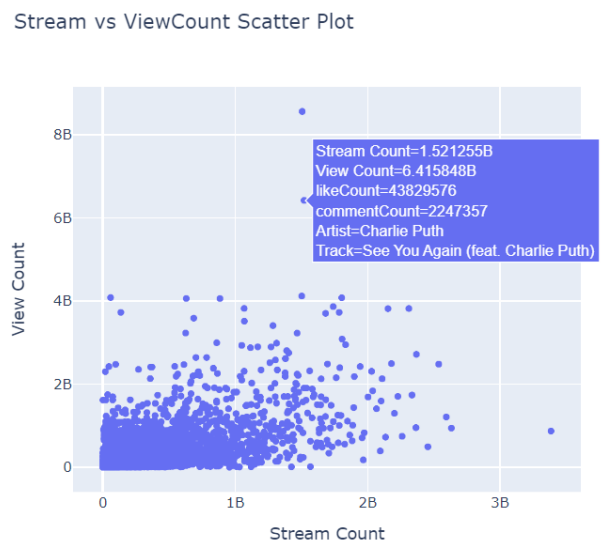
지금까지 스트리밍수와 유튜브의 조회수, 좋아요수, 댓글수 각각의 회귀분석을 한 결과이다. 선형 회귀의 유효성을 나타내는 결정계수를 기반으로 하면, 스트리밍 수와 유튜브 좋아요수가 0.461로 가장 높은 결정계수를 가짐을 알 수 있다. 반면 댓글수와의 관계는 0.082로 가장 낮았다.

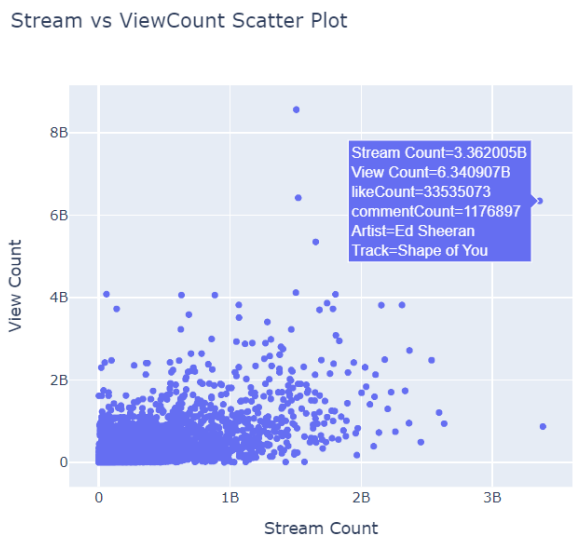
이러한 관계를 시각적으로 나타내고자 산점도와 fitted 회귀선을 그려보았다.



이상치 및 결측치 제거에서도 언급했지만, 산점도의 각 점, 특히 이상치를 나타내는 값들이 어떤 곡이고 어떤 정보를 가졌는지를 알아보고자 인터렉티브 산점도를 그려보았다.

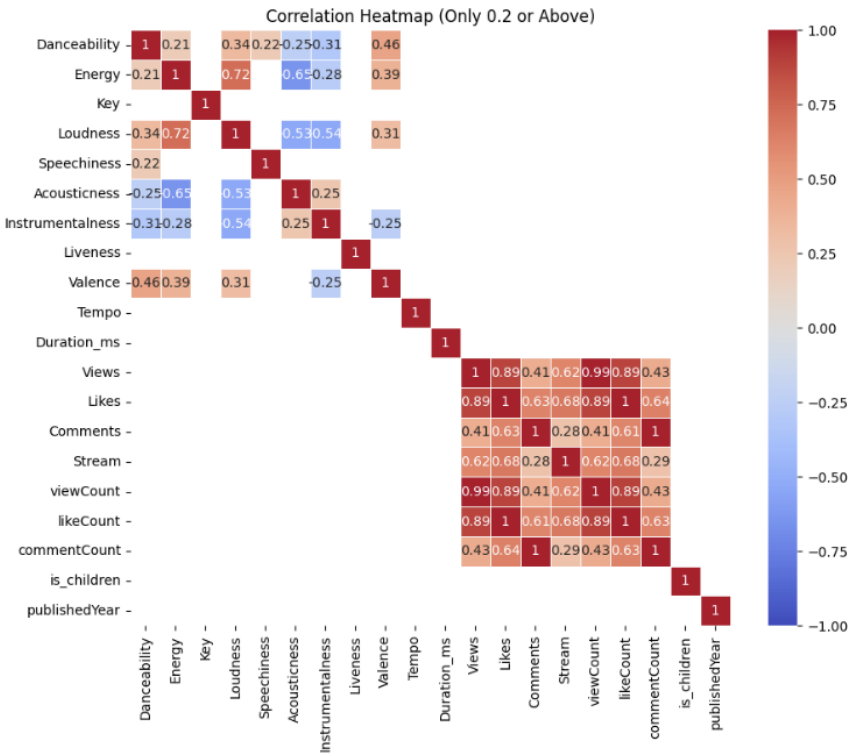
```
# 인터랙티브 산점도 만들기 - hover_data 로 추가 정보 표시
# Stream vs ViewCount

fig = px.scatter(
    df_clean,
    x='Stream',
    y='viewCount',
    hover_data=['likeCount', 'commentCount', 'Artist', 'Track'],  # 마우스를 올렸을 때 표시할 추가 정보
    labels={'Stream': 'Stream Count', 'viewCount': 'View Count'},  # 축 레이블 설정
    title='Stream vs ViewCount Scatter Plot'
)

# 그래프 출력
fig.show()
```





다음으로는 숫자형 데이터만 선택한 numeric_df를 만든 후, 각 feature들 간의 상관계수를 알아보기 쉽게 시각화하고자 히트맵을 그려보았다.

```
# 숫자형(int, float) 열만 선택
numeric_df = df_clean.select_dtypes(include=['number'])

# 결과 출력
#print(numeric_df)

import seaborn as sns

# 각 feature 간의 상관계수 계산
corr_matrix = numeric_df.corr()

# 상관계수가 0.2 이상인 값만 남기고 나머지는 NaN 으로 변환
filtered_corr_matrix = corr_matrix.mask(np.abs(corr_matrix) < 0.2)

# 히트맵 그리기
plt.figure(figsize=(10, 8))
sns.heatmap(filtered_corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1, linewidths=0.5, mask=filtered_corr_matrix.isnull())

# 그래프 제목과 축 레이블 설정
plt.title('Correlation Heatmap (Only 0.2 or Above)')
plt.show()
```



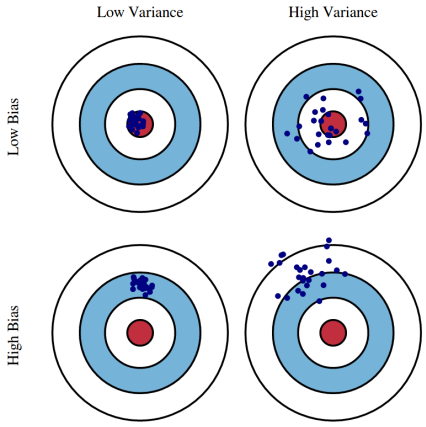
4-1-2. Ridge & Lasso Regression

이전의 Linear Regression 모델을 보면, least squre solution을 구하게 될 때 모델이 다음과 같이 과도하게 복잡해진다.

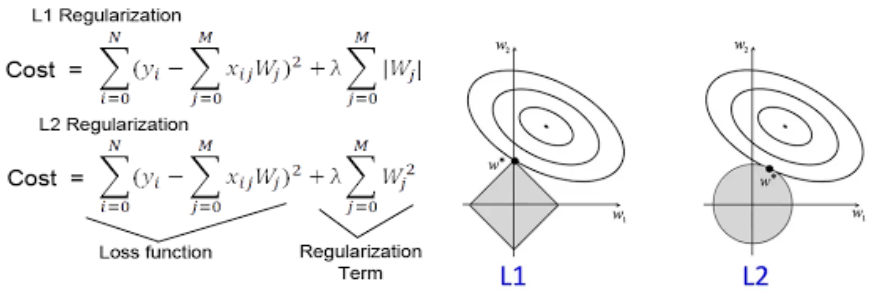
$$\hat{\beta}^{LS} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (y_i - x_i\beta)^2 \right\} = (X^T X)^{-1} X^T y$$

즉, 가중치가 너무 커져서 모델이 데이터에 쉽게 편향되거나 요동칠 수 있다는 의미이다. 데이터가 많아지면 과적합(overfitting)이 해결되는 데에 도움이 되지만, 수집 가능한 데이터의 수는 한정적이기 때문에, 제한적인 데이터를 활용하여 유연한 모델을 만들기 위해서는 정규화(regularization) 과정이 필수적이다.

앞서 다룬 OLS 회귀에서는 가중치에 대해 분산(variance)을 최소화하는 값으로 고르게 된다. 그러나 bias와 variance는 trade-off 관계이므로, bias가 증가하더라도 variance를 더 줄이고자 정규화로 이를 구현하는 것이다.



선형 회귀 모델을 4차식으로 만들었다면, 각각의 계수(β값)이 존재할 것이고 이 때 높은 차수의 베타값을 없애주면 모델을 단순화할 수 있다. 이 때 베타값을 없애는 대표적인 방식이 Ridge와 LASSO regression인 것이다.



L1을 LASSO, L2를 Ridge로 본다면 위와 같이 Cost로 표현된 식에서, Regularization term에 차이가 존재하는 것을 알 수 있다. 오른쪽 그림에서 제약조건과 MSE가 만나면, 제약조건을 만족하면서 에러가 최소

인 지점을 찾을 수 있고 마름모와 원은 각각 Regularization term에서 비롯되었다.

이를 데이터에 직접 적용하여 알고리즘별 성능을 비교해보고자 한다. 사용한 라이브러리는 다음과 같다.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

독립변수와 종속변수를 설정해주었고, 우선 train:test=9:1로 지정했다.

```
# 독립 변수(X)와 종속 변수(y) 설정
X = df_clean[['viewCount', 'likeCount', 'commentCount']]
y = df_clean['Stream']
# 데이터 분리 (훈련 세트와 테스트 세트)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

각 알고리즘을 비교해보고자 다양한 회귀 방법을 적용해보았고, 그 결과 선형 회귀, Ridge 회귀, Lasso 회귀 모두 결정계수가 0.5679로 같게 나오는 것을 확인할 수 있었으며, MSE 값에서 근소한 차이를 보였다.

```
# 1. 선형 회귀 (Linear Regression)
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)
r2_linear = r2_score(y_test, y_pred_linear)
mse_linear = mean_squared_error(y_test, y_pred_linear)
# 2. 릿지 회귀 (Ridge Regression)
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
y_pred_ridge = ridge_model.predict(X_test)
r2_ridge = r2_score(y_test, y_pred_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
# 3. 라쏘 회귀 (Lasso Regression)
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
y_pred_lasso = lasso_model.predict(X_test)
r2_lasso = r2_score(y_test, y_pred_lasso)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
```

```
Linear Regression: R²  = 0.5679, MSE = 32000491345920384.000000
Ridge Regression: R²  = 0.5679, MSE = 32000491345920380.000000
Lasso Regression: R²  = 0.5679, MSE = 32000491345920388.000000
```

4-1-3. KNN Regression

한편, K-Nearest Neighbors (KNN) 회귀 알고리즘을 활용하여 음악적 요인이 YouTube 조회수 차이에 미치는 영향을 분석하고자 하였다. 이를 위해 데이터셋 중 음악과 관련 있는 요인을 feature로 설정하여 모델에 적용한 후, 어떤 feature 조합이 가장 높은 성능을 보이는지 확인하였다. 평가지표로는 NRMSE(Normalized Root Mean Square Error)와 결정계수(R-Squared)를 사용하였다. 이때 NRMSE는 RMSE 값을 테스트 데이터셋의 최댓값과 최솟값의 차로 나누어 정규화한 지표로, 이를 통해 보다 용이한 결과 해석을 꾀하였다.

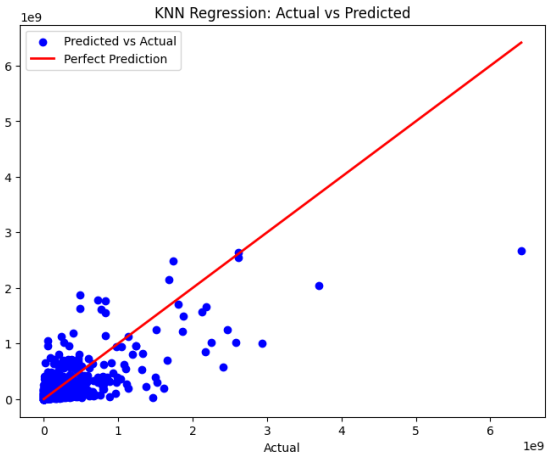
KNN 회귀는 비모수적 기계 학습 알고리즘으로, 예측하려는 값에 가장 가까운 K개의 이웃 데이터를 참조하여 해당 이웃들의 평균값을 통해 예측을 수행한다. 즉, 새로운 데이터 포인트와 기존 데이터 간의 유클리드 거리를 측정하여 가장 가까운 데이터 포인트들을 기준으로 예측 값을 산출한다.

우선, 기준점은 Artist, Danceability, Energy, Key, Loudness, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Tempo, Duration_ms, Licensed, official_video, Stream, is_children, publishedYear의 총 16가지 특성을 학습했을 때의 평가지표로 두었다. 이때 Artist는 기존 데이터셋에서는 문자열 형태의 데이터 타입을 가지나, 학습을 위해 각 아티스트별로 인덱스를 매겨 정수형 데이터타입으로 전환하였다.

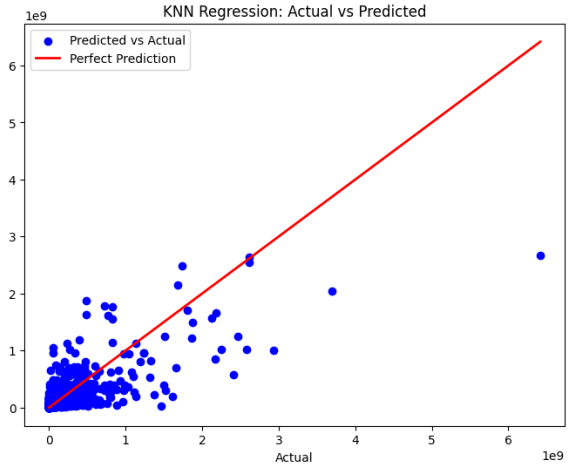
이후 특성 소거를 통해 대조실험을 진행하며, 기준점보다 더 높은 성능을 보이는 특성 조합을 찾고자 하였다. 실험 결과, 일부 특성들을 제거했을 때 기준점보다 높은 성능을 보이는 케이스를 발견할 수 있었다. 비교 예시는 다음과 같다.

- **특성 제거 X (K = 3): 0.0382 (NRMSE) / 0.5071 (R-Squared)**
- Danceability, Tempo 제거 (K = 3): 0.0377 / 0.5203
- Valence 제거 (K = 4): 0.0373 / 0.5294
- Instramentalness, Tempo 제거 (K = 5): 0.0376 / 0.5227
- **Danceability, Key, Instramentalness, Tempo 제거 (K = 7): 0.0366 / 0.5478**
- Danceability, Key, Valence, Instramentalness, Tempo 제거 (K = 14): 0.0375 / 0.5255

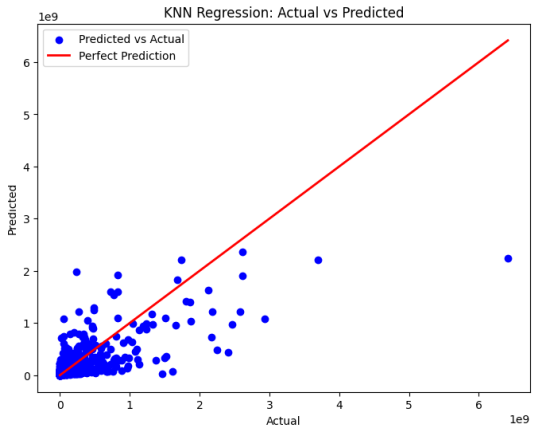
이때, 가장 높은 성능을 보였던 것은 Danceability, Key, Instramentalness, Tempo를 제거했을 때였다. 즉 음악적 특성 중 Energy, Loudness, Speechiness, Acousticness, Liveness, Duration_ms를 학습시켰을 때 가장 성능이 높은 것을 확인할 수 있었다.



<전체 Feature 학습>



<Danceability, Key, Instramentalness, Tempo 제거>



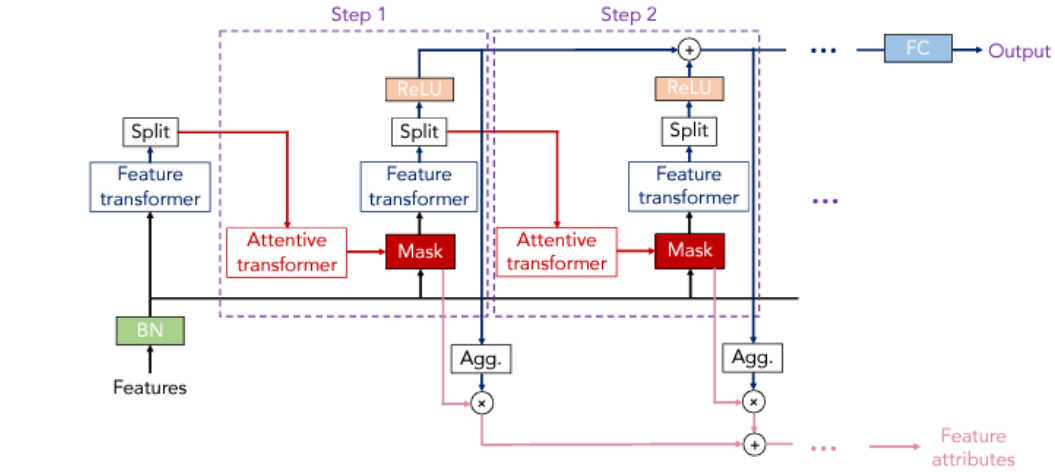
<Danceability, Key, Valence, Instramentalness, Tempo 제거>

그러나 이러한 결과를 단순히 어떤 특성만이 Youtube 영상 조회수에 영향을 미치거나, 상관관계를 갖지 않는다고 해석하는 것은 위험하다. 가령, 기준점에서 Valence만을 제거하였을 때는 0.5294의 결정계수 값을 가지는 것을 확인할 수 있다. 그러나 Danceability, Key, Instramentalness, Tempo를 제거했을 때(0.0366 / 0.5478)와, 이후 추가적으로 Valence를 제거했을 때(0.0375 / 0.5255)를 비교할 시, 전자가 더 높은 성능을 보이고 있음을 확인할 수 있다.

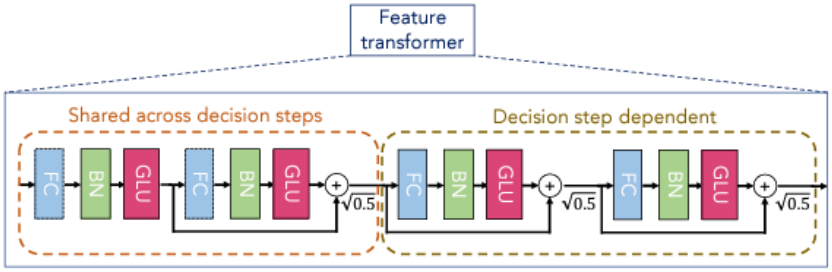
따라서 이러한 결과는 단일 특성과 조회수 간의 선형적 상관관계에서 비롯된 것이 아니며, 각 특성이 함의하는 속성과 다른 특성들 간의 유기적 관계에서 기인한 결과라고 해석할 수 있다. 이는 특성들 간의 상호작용이 조회수에 미치는 영향을 단순히 개별 특성만으로 설명하기 어렵다는 것을 보여준다.

4-2. 딥러닝 기법을 활용한 방식

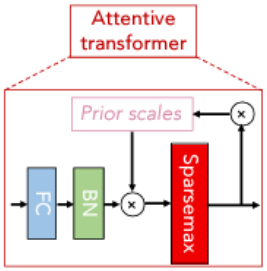
4-2-1. TabNetRegressor



(a) TabNet architecture



(b) Feature transformer



(c) Attentive transformer

TabNet은 'TABNET: Attentive Interpretable Tabular Learning (2019)'에서 발표된 모델[3]로, Attention 메커니즘을 활용하여 학습 중 중요한 feature를 동적으로 선택하는 정형 데이터 특화 딥러닝 모델이다. 입력 데이터를 여러 단계로 반복 처리하며, 중요한 정보는 보존하고 불필요한 정보는 억제하는 구조를 가지고 있다. 핵심 구성 요소로는 디시전 스텝(Decision Step), Feature Transformer, Attentive Transformer 등이 있으며, 이를 통해 입력 데이터의 feature를 점진적으로 학습하고 이를 업데이트한다. TabNet은 feature 중요도를 가중치 기반으로 해석할 수 있어 높은 모델 해석 가능성을 제공하며, Gradient Boosting Machine(GBM)과 유사한 성능을 테이블 데이터에서 구현할 수 있는 강점을 가지고 있다.

TabNetRegressor의 경우, pytorch에서 제공하는 라이브러리에서 해당 모델을 불러와서 사용할 수 있도록 지원하고 있다. 이를 포함하여 탐구에서 활용한 라이브러리는 다음과 같다.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from pytorch_tabnet.tab_model import TabNetRegressor
import pandas as pd
import numpy as np
import wandb
```

주요 하이퍼파라미터는 다음과 같이 config 객체로 관리하였다. 이후 Optuna를 활용하여 하이퍼파라미터 튜닝을 수행하였는데, 해당 부분은 '6-3. Optuna'에서 자세히 후술한다.

```
# 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.
}
```

[3] **Arik, S. Ö., & Pfister, T.** (2021). TabNet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8), 6679–6687. https://arxiv.org/abs/1908.07442

이후 모델 초기화 및 학습을 진행하였다.

```
# 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,
)
```

실험 결과, 결정계수: 0.4252, nrmse: 0.0385라는 값을 얻을 수 있었다. '4-1-3. KNN Regression'의 성능과 비교했을 때, TabNetRegressor 모델은 상대적으로 낮은 성능을 보인다. 본 보고서에서는 이러한 현상의 원인을 TabNet의 구조적 특성에서 찾았다. TabNet은 decision step을 사용한다는 점 때문에, 구조적 특성 때문에 범주형 데이터보다는 연속형 데이터에서 잘 동작하는 경향이 있다. 따라서, 범주형 데이터의 feature 간 상호작용을 더 잘 포착하는 모델을 탐색해야 한다는 결론을 얻었다. 이는 후술할 TabTransformer 모델을 활용한 연구로 이어지게 되었다.

4-2-2. TabTransformer

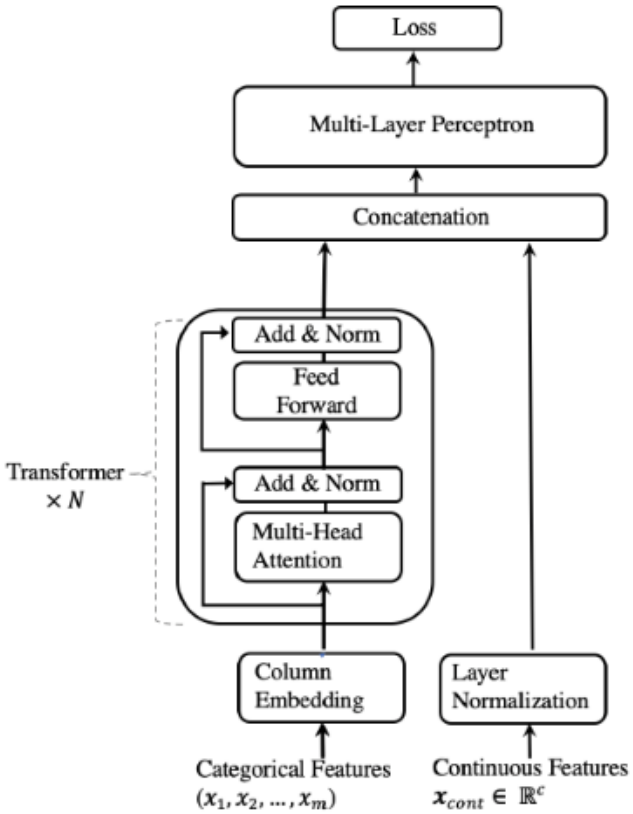


Figure 1: The architecture of TabTransformer.

TabTransformer는 'TabTransformer: Tabular Data Modeling Using Contextual Embeddings (2020)'에서 소개된 모델[4]로, 정형 데이터에서 범주형 feature를 효과적으로 처리하기 위해 Transformer 아키텍처를 적용한 구조를 가진다. 범주형 feature에 임베딩을 적용하여 연속형 벡터로 변환한 뒤, multi-head attention을 활용해 feature 간의 상호작용 관계를 학습한다. 이때 범주형 feature는 구조적으로 고차원 벡터 공간에 매핑되며, 연속형 feature는 변환 없이 결합되거나 추가적인 변환을 거쳐 함께 처리된다. 기존의 원핫 인코딩(one-hot encoding) 방식보다 더 풍부한 정보를 학습할 수 있으며, 범주형 데이터 간의 관계를 효과적으로 학습하여 테이블 데이터 문제에서 높은 성능을 보인다.

[4] **Huang, X., Khetan, A., Cvitkovic, M., & Karnin, Z.** (2020). TabTransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*. https://arxiv.org/abs/2012.06678

탐구에서는 다음과 같은 라이브러리를 활용하였다.

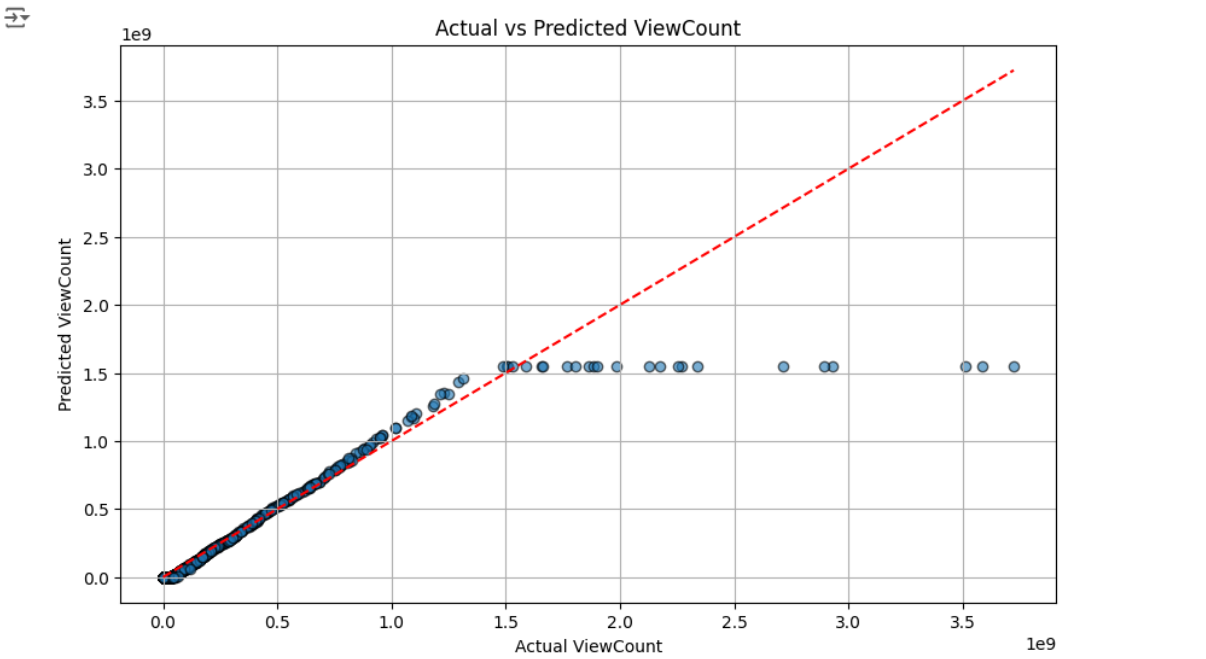
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import torch
from torch.utils.data import DataLoader, Dataset
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
```

또한 논문에서 소개하는 아키텍쳐를 바탕으로, 다음과 같이 TabTransformer 모델을 정의하였다.

```
# 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)
```

먼저, embedding 계층은 nn.Linear 모듈을 통해 입력 데이터를 embed_dim 크기의 고차원 벡터로 매핑한다. 이후 transformer_blocks는 여러 개의 TransformerEncoderLayer로 구성되며, 각 블록은 Multi-Head Attention을 통해 feature 간 관계를 학습한다. 마지막으로 완전 연결 계층(fc)은 Transformer 블록의 출력값을 받아 은닉 뉴런을 거치고, 이를 통해 최종적인 스칼라값을 출력한다. 이때 활성화 함수 ReLU를 도입하여 출력값의 범위를 제한함으로써 모델의 정확도를 높이고자 하였다. 순전파(forward) 과정에서는 입력 데이터를 임베딩하고, Transformer 블록을 반복적으로 통과시킨다. 이렇게 출력된 결과를 완전 연결 계층으로 전달하여 최종 출력을 생성한다.

실험 결과, 결정계수: 0.8487, nrmse: 0.0353라는 값을 얻을 수 있었다. 이는 앞에서 설명하였던 내용과 비교하였을 때, KNN Regression과 TabNetRegressor에 비해 상당히 개선된 결과라고 볼 수 있다. 이러한 결과는 TabTransformer가 전체적인 경향성을 파악하는 데 더 유리하다는 점을 시사한다. 실제로 TabTransformer의 학습 결과를 그래프로 시각화한 결과, 실제 결과값과 예측값의 분포가 유사하게 나타남을 확인할 수 있었다. 그러나 조회수가 20억이 넘어가는 데이터들에 대해서는 추론 능력이 떨어지는 현상을 발견하였는데, 이는 데이터셋 분포의 한계에 따른 결과인 것으로 추정된다.



5. 분류

본 탐구에서는 음원의 Spotify 스트리밍 횟수와 YouTube 조회수 차이에 영향을 미치는 요인을 분석하기 위해 이진 분류 작업을 수행하였다. 이를 위해 각 음원의 스트리밍 횟수와 조회수의 차이 및 평균을 계산한 후, 이 평균을 기준으로 값이 크면 1, 작으면 0으로 라벨링하였다. 이후 기존 데이터셋의 음악 관련 feature(Danceability, Acousticness, Loudness 등)을 독립변수로, 라벨 결과를 종속변수로 설정하여 모델 학습을 진행하였다. 또한, 추론 결과를 시각화하여 결과를 확인하고자 하였다.

5-1. 머신러닝 기법을 활용한 방식

머신러닝 기법을 활용한 분류 알고리즘으로는 결정 트리 기반 알고리즘을 활용하였다. 결정 트리 알고리즘은 데이터의 feature을 기반으로 트리 구조를 형성하여 의사 결정을 내리는 머신러닝 기법으로, 시각적으로 해석하기 쉬우며 데이터의 비선형 관계도 잘 처리할 수 있다는 장점을 지닌다. 또한 결정 과정이 명확해, 모델이 어떻게 결정을 내리는지 직관적으로 파악할 수 있다. 더하여 SVM은 딥러닝 등장 이전까지 많은 연구에서 선택된 분류 알고리즘으로, 현재도 복잡하지 않은 비선형 관계를 추론하는데 매우 높은 정확도를 보여주는 모델로 평가받는다.

최종적으로 Random Forest, Gradient Boosting, XGBoost, SVM의 4가지 모델을 선정하였다. 앞선 3개의 알고리즘은 앙상블 기반 알고리즘으로, 앙상블 학습은 여러 개의 개별 모델을 조합하여 새로운 최적의 모델을 형성하는 기법을 말하며 Bagging, Boosting 등의 방법으로 구분된다.

Bagging 기법은 Boostrap Aggregating의 줄임말로, 복원 추출을 통해 임의의 데이터를 생성하고, 이를 기반으로 여러 개의 독립적인 모델을 생성(Boostrap)한 후, 이를 집계(Aggregating)하여 최종적인 결과를 산출하는 기법이다. 한편 Boosting은 가중치를 활용하여 모델의 성능을 높이는 기법으로, 여러 개의 모델을 순차적으로 학습하며 가중치를 업데이트해나가는 방식을 의미한다.

Random Forest는 Bagging 기법을, Gradient Boosting과 XGBoost는 Boosting 기법을, SVM은 독자적인 margin optimization을 기반으로 하는 알고리즘에 속한다. 본 탐구는 이 네 가지 모델을 활용하여, 스트리밍 플랫폼 간 음원 소비 패턴의 차이를 예측하고, 음악적 특성이 이러한 차이에 미치는 영향을 비교분석하고자 하였다. 이때 학습 데이터셋과 테스트 데이터셋의 비율을 조정하고, Feature Importance를 추출하여 상위 특성을 확인하고 이를 모델 재학습에 활용하는 등의 방법을 활용하여 모델 성능 개선을 꾀하였다. 아울러, Confusion Matrix에 기반한 평가 지표를 활용하여 깊이 있는 결과 사후 분석을 진행하였다.

실험 결과 앙상블 모델은 Train: Test = 8:2 에서, SVM은 7:3 에서 가장 높은 성능을 보였다. 추가적으로 SVM > Random Forest > XGBoost > Gradient Boost 순으로 높은 accuracy를 확인하였다. 또한, gridsearchCV 등을 활용하여 보다 효율적으로 하이퍼파라미터를 탐색하는 방식을 통해 이후에 서술된 목차인 하이퍼파라미터 부분에서 앙상블 알고리즘들의 정확도를 높이는 성과를 얻었다.

5-1-1. Random Forest

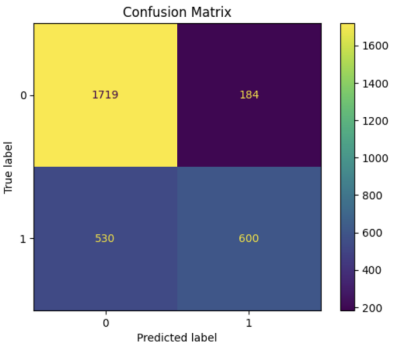
Random Forest는 Bagging 기법을 사용하여 여러 개의 Decision Tree를 학습하고 그 결과를 결합하여 최종 예측을 도출하는 앙상블 학습 방법이다. 먼저, 원본 데이터에서 n개의 샘플 데이터를 무작위로 복원 추출한 후, 이를 각 Decision Tree에 사용하여 독립적으로 학습을 진행한다. 각 트리는 학습 시 일부 특성(feature)을 무작위로 선택하여 분할 기준을 정하는데, 이를 통해 트리들 간의 상관관계를 줄이고, 모델의 다양성을 증가시킨다.

학습이 완료된 후, 각 트리의 예측을 취합하여 최종 결과를 결정한다. 분류 문제의 경우 다수결 투표 방식으로 최종 클래스를 예측하고, 회귀 문제의 경우 각 트리의 예측값의 평균을 최종 결과로 사용한다.

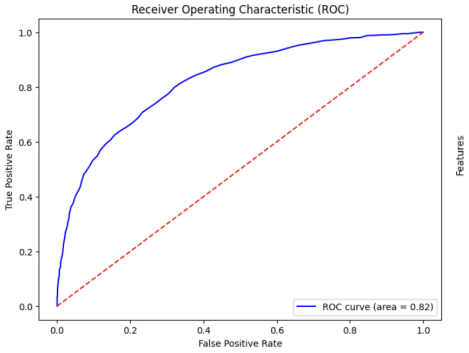
Random Forest는 여러 개의 독립적인 Decision Tree를 사용하기 때문에, 과적합(Overfitting)을 방지할 수 있는 장점이 있다. 여러 트리의 예측을 종합함으로써 개별 트리의 편향이 상쇄되면서, 보다 정확한 예측이 가능해지기 때문이다. 또한, 랜덤 포레스트는 병렬 처리가 가능하여 대규모 데이터에서도 빠른 학습 속도를 유지할 수 있다는 장점이 있다.

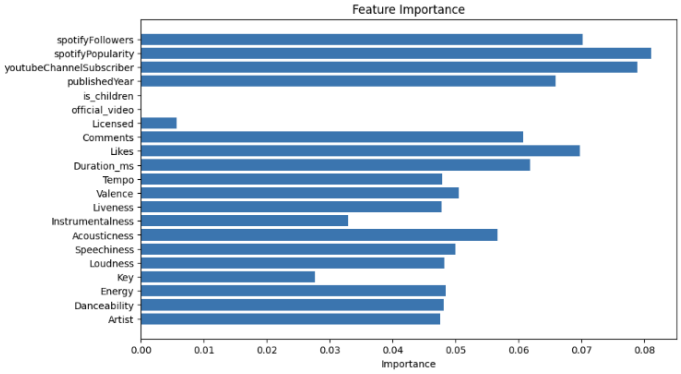
모델 학습 결과는 다음과 같았다.

```
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
```



추가적으로, Feature Importance를 추출한 결과, 'spotifyPopularity', 'youtubeChannelSubscriber', 'publishedYear', 'Likes', 'spotifyFollowers' 등이 주요 특성으로 관찰되었다. 이러한 결과에 근거하여 해당 특성으로 모델을 재학습한 결과, 근소한 accuracy 향상을 확인할 수 있었다.





5-1-2. Gradient Boosting

Gradient Boosting 은 Boosting 기법을 사용하는 알고리즘으로, 여러 개의 **Decision Tree** 를 순차적으로 학습시켜 가중치를 업데이트해나가는 방식이다. 이 알고리즘은 각 모델이 이전 모델의 오류를 보완하며 학습하는 특징이 있다.

초기 모델 학습에서는 주어진 모든 데이터에 대해 각 클래스의 예측 확률을 계산하고, 이 예측값과 실제 라벨 간의 차이, 즉 잔차(Residual)를 구한다. 다음 단계의 Decision Tree 는 이 잔차를 기반으로 학습을 진행한다. 이 과정에서 손실 함수를 최소화하는 방향으로 학습되며, 이를 최적화하기 위한 방법으로 경사하강법(Gradient Descent)이 사용된다.

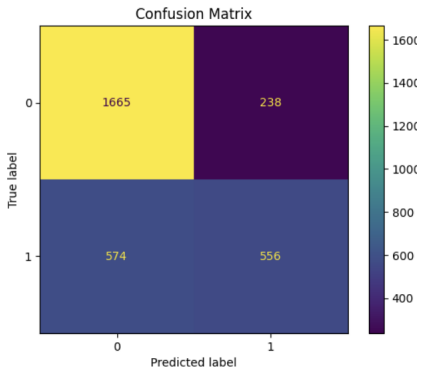
경사하강법은 손실 함수의 기울기(Gradient)를 계산하여 각 단계에서 모델이 어떻게 학습해야 하는지를 결정한다. 이 기울기는 잘못 예측한 데이터에 대해 더 큰 가중치를 부여하는 방식으로 조정된다. 즉, 모델은 예측에서 발생한 오류를 줄이기 위해 점진적으로 개선된다. 이때, 학습률(learning rate) 조정을 통해 모델을 최적화할 수 있다. 본 탐구에서는 learning rate=0.1 로 고정하여 작업을 수행하였다.

이 과정이 반복되면서 여러 개의 결정 트리가 순차적으로 학습되고, 각 모델은 이전 모델의 약점을 보완하여 최종 예측 성능을 높인다. Gradient Boosting 은 이처럼 순차적인 학습을 통해 예측값을 점진적으로 개선하며, 높은 예측 성능을 제공한다.

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



5-1-3. XGBoost

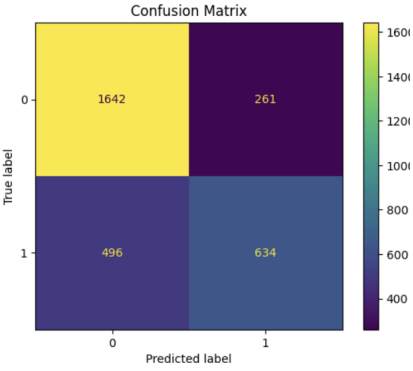
XGBoost(Extreme Gradient Boosting)는 Gradient Boosting을 확장한 기법으로, 핵심 알고리즘은 기존 Gradient Boosting과 동일하다. 각 데이터에 대한 각 클래스의 예측 확률을 계산하고 잔차를 구한 후, 경사하강법을 적용하여 손실함수를 최소화하는 방식으로 모델 성능을 최적화한다. 단, XGBoost의 경우, 손실함수를 최소화하는 과정에서 테일러 확장을 통한 2차 근사(Second-order Approximation)을 같이 진행한다.

또한 XGBoost에서는 L1, L2 정규화(Regularization)를 통해 모델의 복잡성을 제어한다. 이외에도 데이터 분할 과정에서 가지치기(pruning)을 활용함으로써 기여도가 낮은 노드를 제거하고 리프 노드의 페널티를 부여하며 트리가 지나치게 복잡해지거나 깊어지는 것을 방지한다. 이를 통해 기존 Gradient Boosting의 순차적 학습의 단점이었던 과적합(Overfitting) 문제를 완화하고, 모델의 일반화 성능을 높였다는 점에서 이 점을 가진다.

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



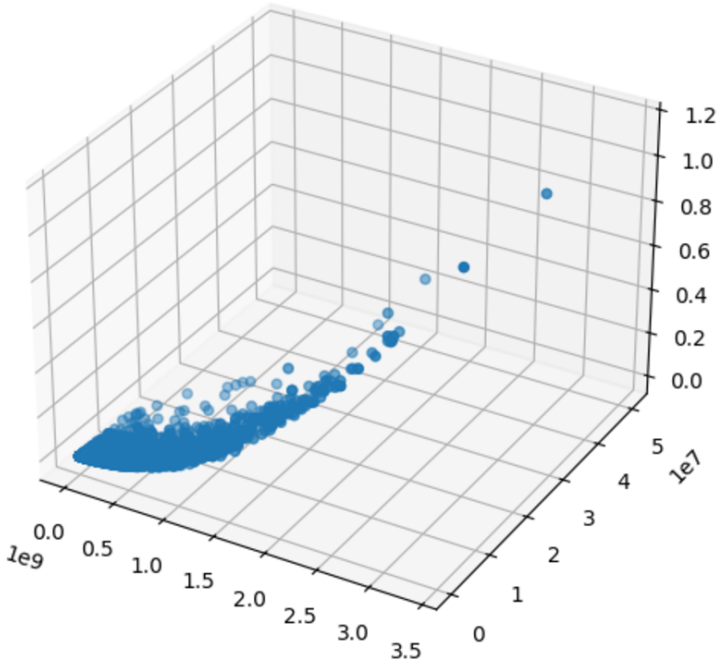
5-1-4. SVM

SVM(Support Vector Machine)은 데이터 분포 간의 margin($\frac{2}{\|w\|}$)을 가장 크게 만드는 decision boundary를 찾는 기법으로, 이 때 decision boundary와 가장 가까운 개별 집단의 데이터를 supprot vector라 부르기 때문에 이러한 명칭이 되었다.

$$\text{argmax}\ \frac{2}{|w|} \iff \text{argmin}\ |w| \text{ s.t. } w \cdot x_i - b \geq 1 \text{ for all } 1 \leq i \leq n$$

SVM은 딥러닝 이전에 자주 사용되던 분류 모델이며 큰 정확성을 자랑한다. 본 탐구에서도 딥러닝 모델을 활용한 분류 task에 앞서 성능 비교를 위하여 먼저 SVM을 이용한 분류 작업을 진행하였다. 다음은 코드 진행을 통해 데이터를 불러오고 이를 시각화하여 분포를 확인하는 과정이다.

```
# Load the CSV file
data = pd.read_csv('241029_nonchild_dataset.csv')

# Drop rows with missing values in specified columns
data_clean = data.dropna(subset=['Stream', 'Likes', 'viewCount', 'likeCount', 'commentCount']).copy()
# Apply log transformation
data_clean['viewCount_log'] = np.log1p(data_clean['viewCount'])

#print(data_clean['Stream'].shape)
#print(data_clean['Likes'].shape) >> (15148,)

# Set 2D input and kernel trick x^2 + y^2
X = np.vstack((data_clean['Stream'], data_clean['Likes']))
#print(X.transpose().shape) >> (15148,2)
x = X.transpose()[:,0]
y = X.transpose()[:,1]
trick = x**2 + y**2

# Show data cluster
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(1,1,1,projection = "3d")
ax.scatter(x,y,trick)
plt.show()
```

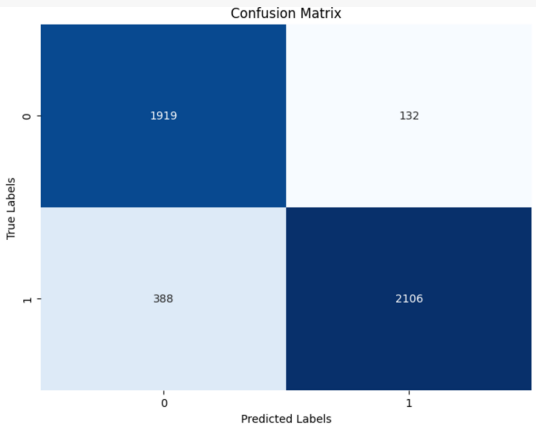


SVM에서 decision boundary는 확장된 데이터 공간에서의 초평면으로서 구해진다. 이 때 데이터 간의 비선형적 관계를 이용해 데이터의 사상을 효율적으로 해주는 것이 바로 kernel trick이다. 이렇게 확장된 데이터 분포를 3차원적으로 관찰하면 위와 같다. Kernel은 $x^2 + y^2$을 사용하였다.

다음으로 SVM 모델에 데이터를 이용한 학습과 추론을 진행하는 코드를 수행한다.

```
# Set the target variable
X = np.vstack((x,y, trick))
X = X.transpose()
m = data_clean['viewCount_log'].mean()
y_view = data_clean['viewCount_log'].apply(lambda x: 1 if x > m else 0)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_view, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the SVM model
#svm = SVC(kernel='poly', degree=2, C=1.0, class_weight = 'balanced')
svm = SVC(kernel='linear', C=1.0, gamma = 0.5, class_weight = 'balanced')
svm.fit(X_train, y_train)

# Predict and evaluate
y_pred = svm.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:₩n", classification_report(y_test, y_pred))
print("Confusion Matrix:₩n", confusion_matrix(y_test, y_pred))

cm = confusion_matrix(y_test,y_pred)
plt.figure(figsize=(8, 6))
sb.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

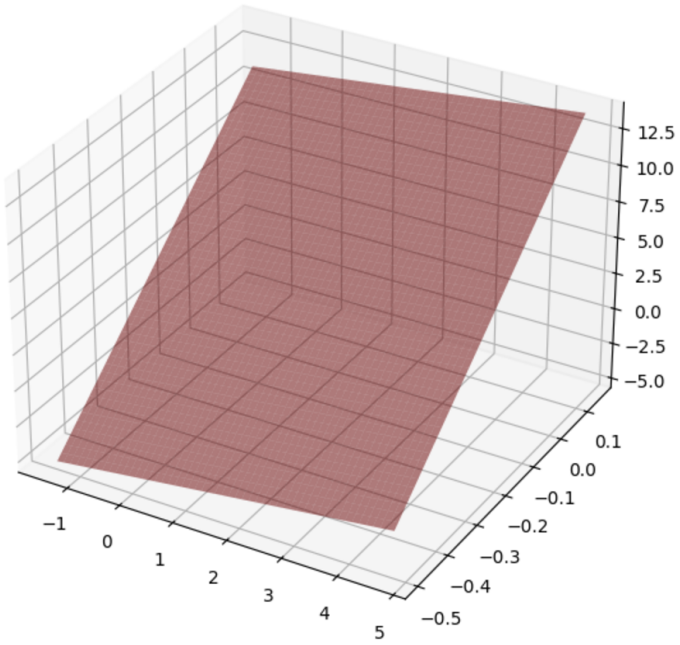
```
Accuracy: 0.8855885588558856
Classification Report:
              precision    recall  f1-score   support

           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]]
```



train, test, validation set을 적절히 설정해준 뒤, 모델의 학습 및 추론을 진행한다. Confusion matrix 및 추론 결과는 다음과 같다. 정확도는 약 0.89로 측정되었다.

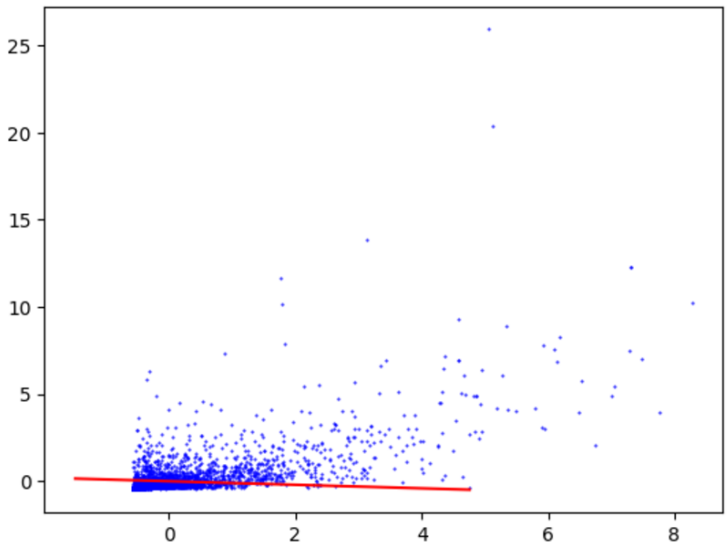
수학적으로 엄밀한 의미에서의 decision boundary를 구하고 출력하는 것은 많은 연산을 동반하며 비효율적이다. 따라서 support vector들 중 가장 극단에 위치한 두 점을 추출한 후, 두 벡터의 차(direction)를 법선벡터로 가지며 중점(d)를 지나는 3차원 평면을 시각화함으로써 원하는 방향의 결과를 얻을 수 있다. 다음으로 SVM 모델의 추론 과정에서 사용된 초평면을 근사적으로 시각화하는 코드를 수행한다.

```
# Infer the decision boundary plane in 3D
norm = svm.support_vectors_**2
norm = np.sqrt(norm[:,0] + norm[:,1])
idx_Max = norm.argmax()
idx_Min = norm.argmin()
s1 = svm.support_vectors_[idx_Max]
s2 = svm.support_vectors_[idx_Min]
d = (s1 + s2) / 2
# Create the decision line (extend it in both directions)
direction = s2 - s1
line_x = np.linspace(d[0] - direction[0], d[0] + direction[0], 15148)
line_y = np.linspace(d[1] - direction[1], d[1] + direction[1], 15148)

# Decision boundary (a plane in this case)
xx, yy = np.meshgrid(
    line_x, line_y
)

# Plane equation z = ax + by + c
a, b, c = direction[0], direction[1], direction[2]
zz = a * xx + b * yy + c
# Example coefficients from SVM
w = svm.coef_[0]  # Coefficients for the plane
b = svm.intercept_[0]  # Intercept for the plane
# Plane equation: w[0] * x + w[1] * y + w[2] * z + b = 0
# Solve for z: z = (-w[0]*x - w[1]*y - b) / w[2]
zz = (-w[0] * xx - w[1] * yy - b) / w[2]

# Plot the SVM decision boundary (as above)
# Plotting
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection="3d")
# Data cluster (points)
#ax.scatter(x, y, trick, color='red', label='Data Cluster')
ax.plot_surface(xx, yy, zz, alpha=0.5, color='red', label='Decision Boundary')
plt.show()
```



마지막으로 원래 데이터 입력값의 차원인 2차원에서도 어떤 식으로 모델이 구성되었는지를 확인할 수 있다. 2차원 도식을 나타내는 코드 진행은 다음과 같다.

```
# 2D projection
plt.plot(X_test[:,0],X_test[:,1], 'bo', markersize=0.5)
plt.plot(line_x, line_y, 'r')
```



5-2. 딥러닝 기법을 활용한 방식

회귀 task의 '4-2. 딥러닝 기법을 활용한 방식'에서 상술한 바와 마찬가지로, 분류 task에서도 Transformer의 구조를 활용한 알고리즘을 사용하였다. 그 중 TabNetClassifier은 회귀 task에서 사용하였던 TabNetRegressor와 같은 모델을 사용하는 분류기이며, pytorch에서 지원하는 모듈을 사용하였다. TabPFN은 tabpfn 라이브러리에서 지원하는 TabPFNClassifier 모듈을 사용하였으며, 해당 모듈은 소형 데이터셋에 특화되었다는 특성상 1024개 이하의 데이터셋을 학습할 것을 권고한다. 이에 Train: Test 데이터셋의 비율을 조정하여 실험을 진행하였다.

5-2-1. TabNetClassifier

앞서 설명한 바와 같이, 해당 분류기는 회귀 task의 TabNetRegressor와 동일한 모델을 기반으로 한 분류기이다. 이때 하이퍼파라미터 최적화에 도움을 주는 WandB를 사용하여 모델 학습 및 추론을 진행하였다. 또한 TabNet 내부적으로 제공되는 기능을 활용하여 feature importance를 직접 확인해볼 수 있었다.

사용한 라이브러리 및 데이터셋 전처리 과정은 다음과 같다.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from pytorch_tabnet.tab_model import TabNetClassifier
import pandas as pd
import numpy as np
import wandb
import matplotlib.pyplot as plt
import seaborn as sns

# 스트리밍 횟수 - 유튜브 조회수 차이 비교
df = df.dropna()
df_stream = list(df['Stream'])
df_view = list(df['viewCount'])
gap_sv = [i-j for i, j in zip(df_stream, df_view)]

# 평균 구하기
avg = sum(gap_sv) // len(gap_sv)
# 평균보다 크면 1, 작으면 0
y_binary = [int(i>avg) for i in gap_sv]
df['y_binary'] = y_binary
# 숫자형 데이터만 남기기 (TabNET)
numeric_df = df.select_dtypes(include=['number'])
#학습할 특징 설정
feature_list = list(numeric_df.columns[:11]) + ['Stream', 'is_children', 'publishedYear']
# 결측치 제거
numeric_df = numeric_df.dropna()
```

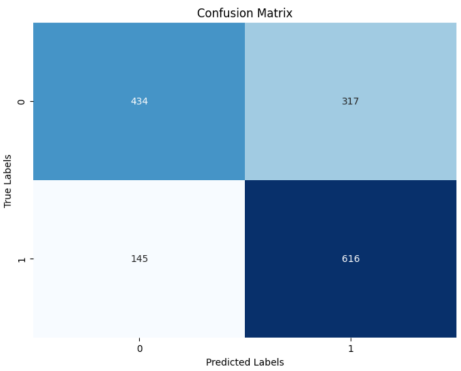
하이퍼파라미터 설정 및 모델 초기화 과정은 다음과 같다.

```
# 하이퍼파라미터 설정
config = {
    "n_d": 8,                       # Decoder feature dimension size
    "n_a": 8,                       # Attention feature dimension size
    "n_steps": 3,                   # Number of steps in the architecture
    "gamma": 1.5,                     # Relaxation parameter for sparsity
    "lambda_sparse": 1e-3,          # Sparsity regularization
    "learning_rate": 1e-3,          # Learning rate
    "max_epochs": 200,                # Maximum number of epochs for training
    "batch_size": 256,              # Batch size for training
    "virtual_batch_size": 128,      # Virtual batch size for gradient accumulation
    "seed": 42,                     # Random seed for reproducibility
    "momentum": 0.3,                  # Momentum for batch normalization
    "optimizer": "Adam",            # Optimizer type
    "scheduler_params": {
        "scheduler": "StepLR",      # Learning rate scheduler type
        "step_size": 10,            # Step size for learning rate decay
        "gamma": 0.9                  # Decay rate for learning rate
    },
    "verbose": 1,                   # Verbosity level (1 for basic info, 0 for silent)
    "device": "cuda"                # Device for training, can be "cuda" or "cpu"
}

# TabNET 모델 초기화
model = TabNetClassifier(
    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"])
)
```

모델의 학습 및 추론 과정은 다음과 같다. 학습 결과 정확도는 약 0.695로 측정되었다.

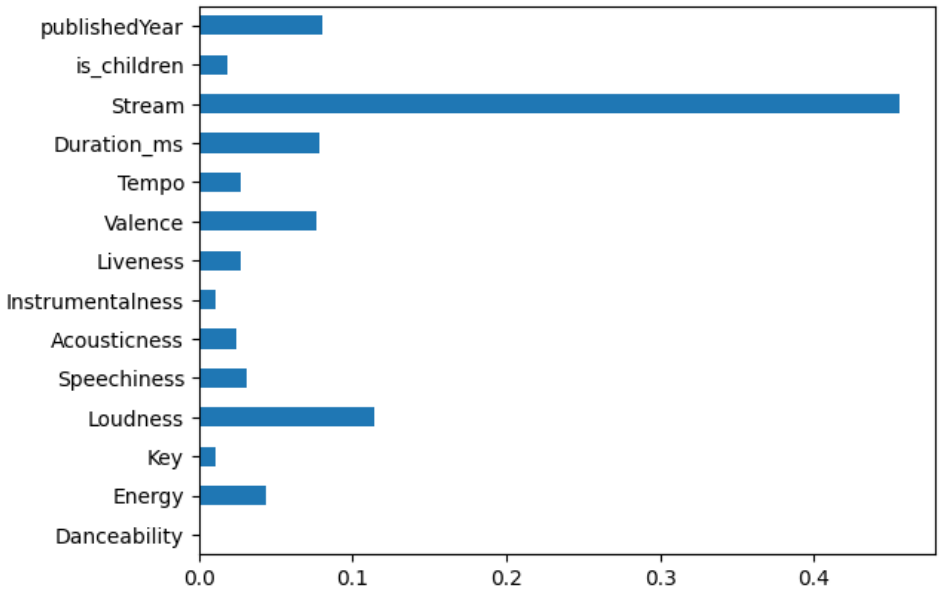
```
# 모델 학습
model.fit(
    X_train=X_train.values, y_train=y_train.values,
    eval_set=[(X_test.values, y_test.values)],
    eval_name=['val'],
    eval_metric=['accuracy', 'logloss'],
    max_epochs=config["max_epochs"],
    patience=10,
    batch_size=config["batch_size"],
    virtual_batch_size=config["virtual_batch_size"],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

# 학습 후 성능 기록
for epoch in range(len(model.history['val_accuracy'])):
    wandb.log({
        "epoch": epoch,
        "val_accuracy": model.history['val_accuracy'][epoch],
        "val_loss": model.history['val_logloss'][epoch],
    })
# 모델 예측
y_pred = model.predict(X_test.values)

# 평가 지표 계산
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Test Accuracy:", accuracy)
print("₩nClassification Report:₩n", class_report)
```

```
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
```

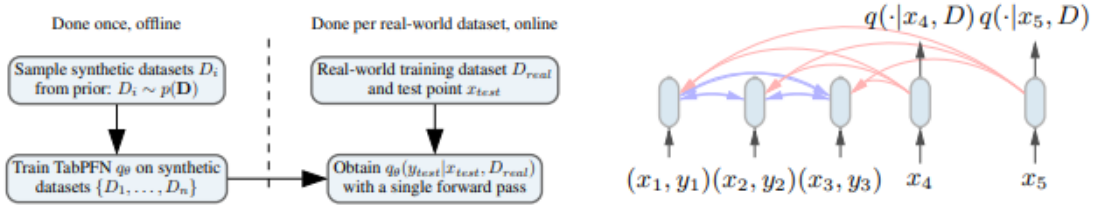


마지막으로, feature importance를 시각화한 결과는 다음과 같다. 음악적 특성 중에서는 Loudness, Valence, Energy 등이 모델 판단에 기여도가 높은 것을 확인할 수 있었다. 본 보고서에서는 이러한 양상을 '3-1-1. K-Means'에서 설명했던 바와 같이, 음악의 장르적 특성과 이에 따라 나타나는 뮤직비디오의 양상과 연관되어 있다고 해석하였다.

```
feat_importances = pd.Series(model.feature_importances_, index=feature_list)
feat_importances.plot(kind='barh')
```



5-2-2. TabPFN



(a) Prior-fitting and inference (b) Architecture and attention mechanism

TabPFN은 'TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second (2022)'에서 소개된 모델[5]로, 정형 데이터에 대해 소형 신경망과 사전 학습된 네트워크를 결합하여 빠르고 효율

[5] **Hollmann, N., Müller, S., Eggensperger, K., & Hutter, F.** (2022). TabPFN: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*. https://arxiv.org/abs/2207.01848

적인 학습을 제공한다. 해당 모델은 머신 러닝 문제에 대해 사전에 학습된 분포를 기반으로 새로운 테이블 데이터 문제를 처리하며, 학습 및 예측 시간을 수 초 내로 단축할 수 있다. 또한, Probabilistic 모델링을 활용해 예측 과정에서 불확실성을 정량화할 수 있다

TabPFN의 구조는 사전 학습된 네트워크를 활용해 데이터의 특성을 신속히 파악하고 최적의 매개변수를 자동으로 조정하도록 설계되어 있는데, 이러한 설계는 소규모 데이터셋에서 특히 뛰어난 성능을 발휘한다. 아울러, 복잡한 하이퍼파라미터 튜닝 없이도 높은 성능을 달성할 수 있다는 장점을 가진다.

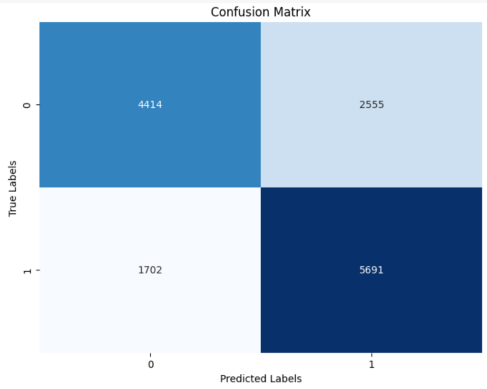
사용한 라이브러리는 다음과 같으며, 데이터 전처리 과정은 앞의 '5-2-1. TabNetClassifier'과 동일하다.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from tabpfn import TabPFNClassifier
import pandas as pd
import numpy as np
import wandb
import matplotlib.pyplot as plt
import seaborn as sns

# 스트리밍 횟수 - 유튜브 조회수 차이 비교
df = df.dropna()
df_stream = list(df['Stream'])
df_view = list(df['viewCount'])
gap_sv = [i-j for i, j in zip(df_stream, df_view)]

# 평균 구하기
avg = sum(gap_sv) // len(gap_sv)
# 평균보다 크면 1, 작으면 0
y_binary = [int(i>avg) for i in gap_sv]
df['y_binary'] = y_binary
# 숫자형 데이터만 남기기 (TabNET)
numeric_df = df.select_dtypes(include=['number'])
#학습할 특징 설정
feature_list = list(numeric_df.columns[:11]) + ['Stream', 'is_children', 'publishedYear']
# 결측치 제거
numeric_df = numeric_df.dropna()
```

모델 학습 및 추론 과정은 다음과 같다.

```
# 타겟 열 분리
X = numeric_df[feature_list]
y = numeric_df['y_binary']
# 데이터셋 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.95, random_state=42)
# 하이퍼파라미터 설정
config = {
    'device': 'cpu',
    'N_ensemble_configurations': 16,
}
# 모델 학습
model = TabPFNClassifier(
    device=config['device'],
    N_ensemble_configurations=config['N_ensemble_configurations'],
)

# 모델 학습
model.fit(X_train, y_train)

# 학습 후 성능 기록
wandb.config.update(config)
# 모델 예측
y_pred = model.predict(X_test.values[:100])

# 평가 지표 계산
accuracy = accuracy_score(y_test[:100], y_pred)
conf_matrix = confusion_matrix(y_test[:100], y_pred)
class_report = classification_report(y_test[:100], y_pred)
print("Test Accuracy:", accuracy)
print("₩nClassification Report:₩n", class_report)
```

```
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
```



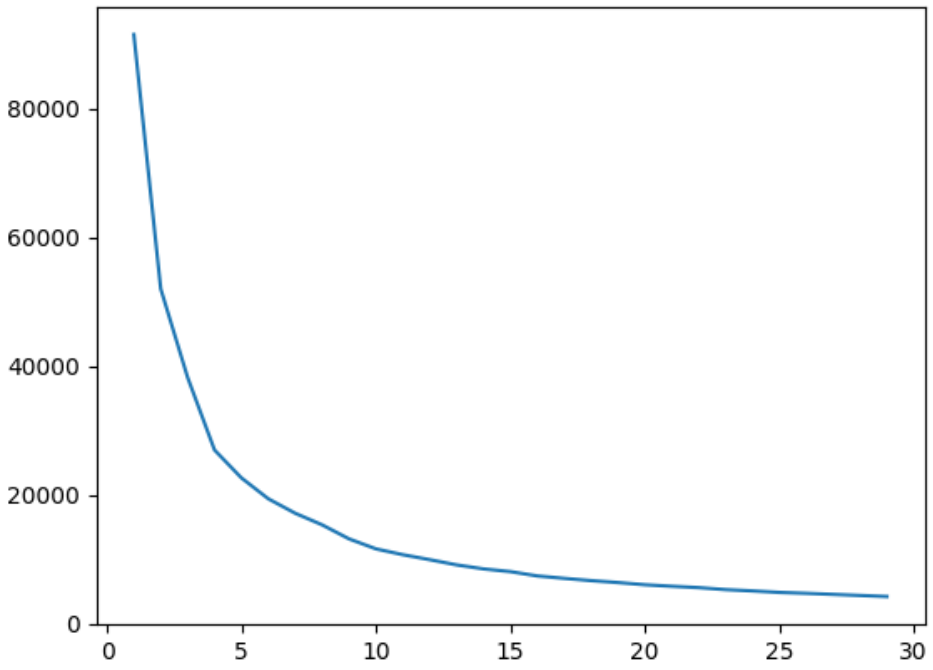
성능은 약 0.703으로 TabNetClassifier에 비해 근소하게 높은 accuracy 점수를 보여주었다. 그러나 학습에 사용된 데이터셋의 양이 압도적으로 적다는 점을 고려하였을 때, 본 보고서에서는 해당 모델이 효율성 측면에서 상당한 개선을 보였다고 해석하였다.

6. 하이퍼파라미터 튜닝

6-1. Elbow Method

머신러닝 기법을 이용한 클러스터링 및 회귀 모델에서, 최적 해를 탐색하기 위해 Elbow Method 기법을 수행하였다. Elbow Method는 탐색하고자 하는 하이퍼파라미터 값를 증가시키면서 작업을 수행하고, 이를 그래프로 나타내어 최적해를 선택하는 방식을 의미한다. 이때 최적해는 주로 그래프가 꺾이는 지점에서 찾을 수 있는데, 이 모양이 마치 팔꿈치 모양과 유사해서 Elbow Method라는 이름이 붙게 되었다. 해당 기법은 K-means, DBSCAN, KNN Regression에서 사용되었다.

K-Means에서는 적절한 군집 개수(k)를 탐색하기 위해, y축을 Inertia value로 설정하였다. Inertia Value는 centroid와 해당 클러스터에 해당된 데이터 포인트 간 거리의 합을 의미한다. 이 지표는 군집이 얼마나 잘 응집되었는지 나타낸다. 그래프 결과에 따라, k = 4 ~ k = 6 범위에서 가장 효율적인 결과를 보였다고 판단하였고, 이후 실험을 통해 최적의 값(k=4)을 결정하였다.

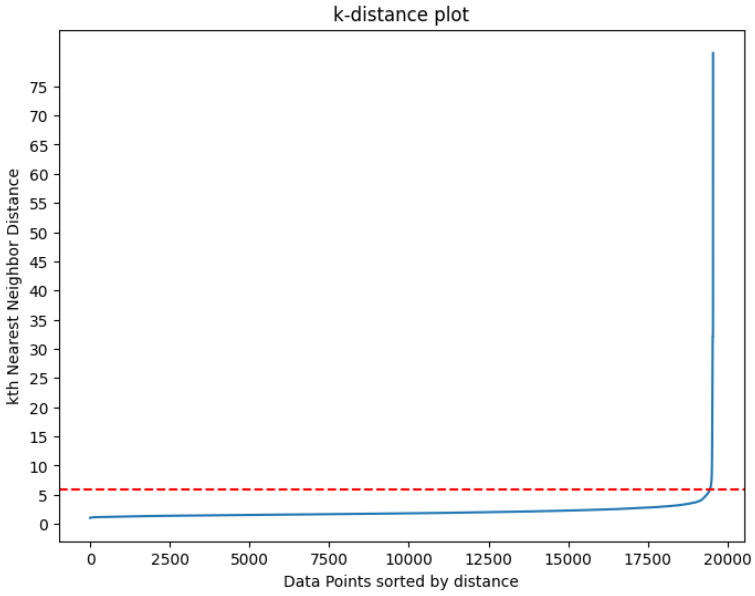


<Inertia Value Graph>

한편, DBSCAN에서는 ε을 탐색하기 위해, k-거리 플롯(k-distance plot)을 활용하여 최적해에 근사하는 것을 시도하였다. K-거리 플롯은 각 데이터 포인트에 대해 k(=minPts)번째로 가까운 이웃까지의 거리를 계산하며, 계산된 데이터 포인트의 k번째 이웃 거리값을 오름차순으로 정렬한다. 이후 데이터 포인트 인덱스를 x축으로, 데이터 포인트의 k번째 이웃 거리를 y축으로 설정하여 그래프로 나타낸다.

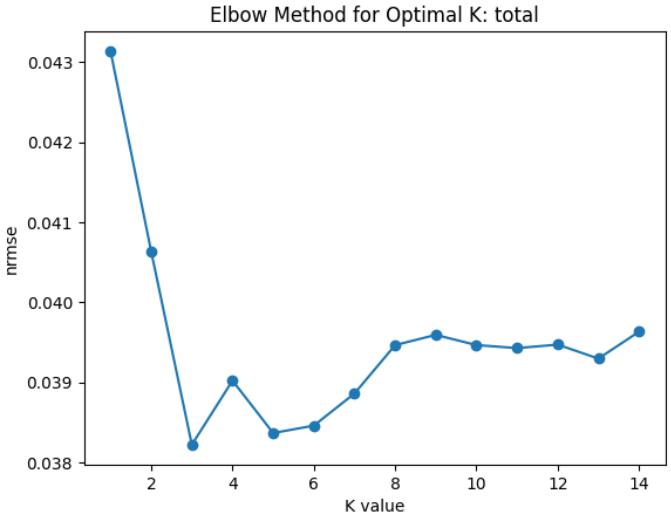
이 그래프를 통해 데이터의 밀집도를 시각적으로 확인할 수 있다. 그래프의 초기 부분은 밀도 높은 영역으로, 이웃 간 거리가 작아 비교적 완만한 경사를 보인다. 반대로, 그래프가 급격하게 상승하는 지점은 데이터 포인트들이 더 이상 가까운 이웃을 찾기 어려운 지점이다. 즉, 그래프의 급격한 경사 변화가 시작되는 지점을 찾으면 적절한 ε값을 선택할 수 있다.

본 탐구에서는 minPts = 100으로 설정한 후 k-거리 플롯을 확인하였다. 수행 결과, ε = 4 ~ ε = 7 범위에서 경사가 급격히 상승하는 것을 확인할 수 있었다. 이후 실험을 거쳐 최적의 값(ε = 6)을 채택하였다.

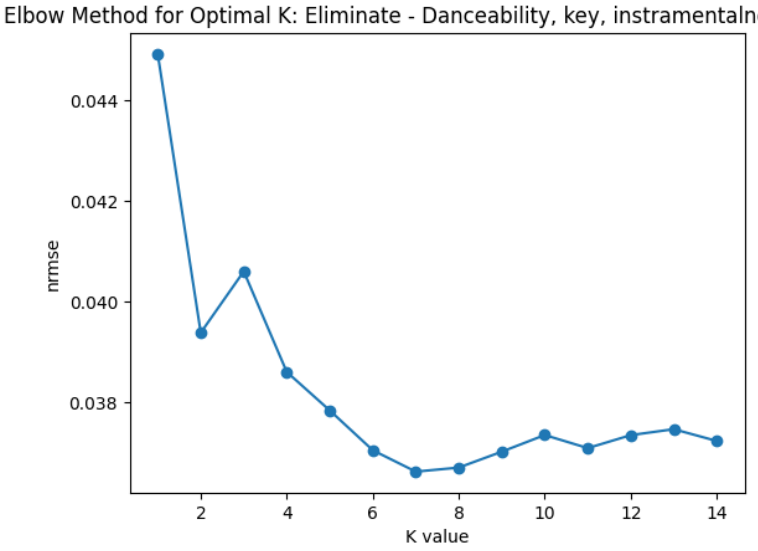


<K-거리 플롯>

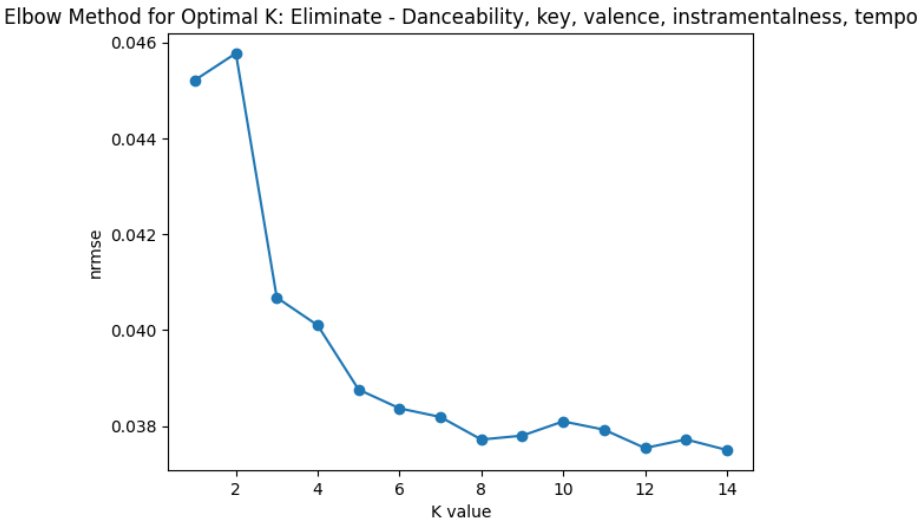
KNN 회귀에서는 참조 이웃 데이터 개수(K)를 탐색하기 위해, y값을 NRMSE(정규화된 RMSE)로 설정한 후 해당 기법을 적용하였다.



<전체 Feature 학습>



<Danceability, Key, Instramentalness, Tempo 제거>



<Danceability, Key, Valence, Instramentalness, Tempo 제거>

6-2. GridSearchCV

머신러닝에서 모델의 성능향상을 위해 쓰이는 기법으로, 사용자가 직접 모델의 하이퍼 파라미터의 값을 리스트로 입력하면 값에 대한 경우의 수마다 예측 성능을 측정 평가하여 비교하면서 최적의 하이퍼 파라미터 값을 찾는 과정을 진행한다.

사용하는 라이브러리는

```
from sklearn.model_selection import GridSearchCV
```

이며, 다음과 같이 하이퍼파라미터의 범위를 리스트로 지정한다.

```
param = {"learning_rate" : [0.01,0.1,0.2,0.3,0.4,0.5],
                "max_depth" : [25, 50, 75],
                "num_leaves" : [100,300,500,900,1200],
                "n_estimators" : [100, 200, 300,500,800,1000],
                "learning_rate" : [0.01,0.1,0.2,0.3,0.4,0.5]
                 }
```

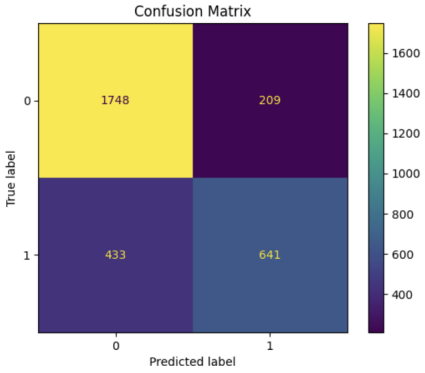
또한 분류 알고리즘에 대한 설명 중, '5-1.머신러닝 기법을 활용한 방식'에서 상술하였듯, 해당 기법은 머신러닝을 활용한 분류 기법 중 Random Forest, Gradient Boosting, XGBoost 에 적용되었다. Train: Test 데이터셋의 비율을 8:2 기준으로, 각 알고리즘별 최적의 파라미터 값은 다음과 같았으며, 세 경우 모두 Optuna 사용이 성능 향상에 기여한 것을 확인할 수 있었다.

```
# Random Forest: 0.7646 -> 0.7882
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

# Gradient Boosting
Best Parameters: {'learning_rate': 0.2, 'max_depth': 7, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200}

# XGBoost
Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 7, 'min_child_weight': 1, 'subsample': 0.8, 'n_estimators':
200}
```

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

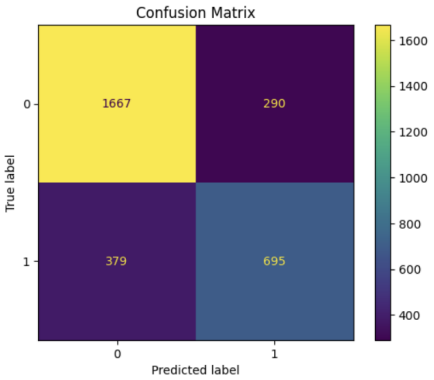


<GridSearchCV 적용 이후 Random Forest 결과>

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

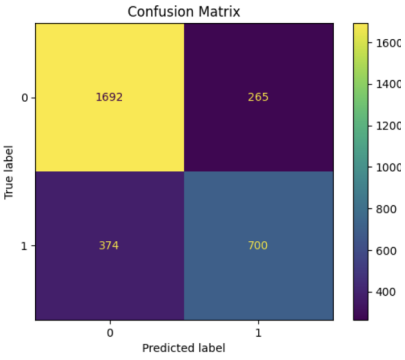


< GridSearchCV 적용 이후 Gradient Boosting 결과>

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



< GridSearchCV 적용 이후 XGBoost 결과>

6-3. Optuna

하이퍼파라미터를 자동으로 조정하고 최적화하는 오픈 소스 라이브러리로, 딥러닝 등 다양한 유형의 알고리즘에서 사용이 가능하다. Optuna의 핵심 기능은 "효율적인 하이퍼파라미터 최적화"에 있으며, 최적화 과정은 크게 두 가지 메커니즘으로 이뤄진다.

1. 트리 기반의 구조화된 파라미터 최적화 (TPE)
2. 프루닝 메커니즘

TPE는 베이지안 최적화의 일종으로, 기존의 grid 검색 방법보다 효율적으로 공간을 탐색하며, 모델 성능을 가장 잘 향상시킬 것으로 예상되는 하이퍼파라미터의 조합에 더 많은 자원을 할당한다.

프루닝 메커니즘은 계산 자원의 낭비를 줄이기 위해 비효율적인 시도를 조기에 중단시키는 기술이다. 훈련 중인 모델이 중간 평가에서 기대치에 미치지 못할 경우, 추가 훈련 없이 해당 시도를 중단시켜서, 최적화 프로세스의 속도를 높이고 불필요한 자원의 소모를 방지한다.

추가로, 분산 환경에서의 병렬 실행을 지원하며 사용자가 자신의 목적 함수를 자유롭게 정의할 수 있다는 점이 맞춤형 최적화를 가능하게 하는 optuna의 장점이다.

Optuna 를 TabNetRegresseor 에 적용한 코드는 다음과 같다.

```
import optuna
from optuna.samplers import TPESampler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from pytorch_tabnet.tab_model import TabNetRegressor
import numpy as np

# Objective 함수 정의
def objective(trial):
    # 탐색할 하이퍼파라미터 설정
    n_d = trial.suggest_int("n_d", 8, 64) # Decision layer size
    n_a = trial.suggest_int("n_a", 8, 64) # Attention layer size
    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)

    # TabNet 모델 초기화 및 학습
    model = TabNetRegressor(
        n_d=n_d, n_a=n_a, n_steps=n_steps, gamma=gamma,
        lambda_sparse=lambda_sparse, optimizer_params=dict(lr=learning_rate)
    )
```

```
    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,
    )
```

그 결과, 다음과 같은 결론을 얻었다.

Rmse(평균제곱근오차)가 351541851.9233에서 247234446.4438 로 감소하였으며, Nrmse(정규화된 평균제곱근오차)가 0.0548에서 0.0385로 감소하였다. 또한 R2(결정계수)가 음수인 -0.1622에서 0.4252로 증가하였다.

초기 하이퍼파라미터 설정값은 다음과 같았다.

```
config = {
    "n_d": 8,                   # Decoder feature dimension size (8~64)
    "n_a": 8,                   # Attention feature dimension size (n_d = n_a)
    "n_steps": 3,               # Number of steps in the architecture. (3~10)
    "gamma": 1.5,               # Relaxation parameter for sparsity (1.0~2.0)
    "lambda_sparse": 1e-3,      # Sparsity regularization
    "learning_rate": 1e-3,      # Learning rate
    "max_epochs": 300,          # Maximum number of epochs for training
    "batch_size": 256,          # Batch size for training (64, 128, 256, 512, 1024, ...)
    "virtual_batch_size": 128,  # Virtual batch size for gradient accumulation (should divide batch_size)
    "seed": 42                  # Random seed for reproducibility.
}
```

이 때의 결과값은 다음과 같다.

RMSE: 351541851.9233

NRMSE: 0.0548

R2: -0.1622

파라미터 최적화를 위한 optuna를 적용한 이후에는, 다음과 같은 최적화 결과를 얻었다.

```
Stop training because you reached max_epochs = 100 with best_epoch = 99 and best_val_rmse = 247234446.44381
{'n_d': 38, 'n_a': 42, 'n_steps': 7, 'gamma': 1.7387573518282637, 'lambda_sparse': 2.1150092752905284e-06, 'learning_rate': 0.05939016164949467, 'batch_size': 64, 'virtual_batch_size': 32}
```

최종적으로 설정한 하이퍼파라미터는 다음과 같다.

```
config = {
    "n_d": 38,                      # Decoder feature dimension size (8~64)
    "n_a": 42,                      # Attention feature dimension size (n_d = n_a)
    "n_steps": 7,                   # Number of steps in the architecture. (3~10)
    "gamma": 1.7387573518282637, # Relaxation parameter for sparsity (1.0~2.0)
    "lambda_sparse": 2.1150092752905284e-6,      # Sparsity regularization
    "learning_rate": 0.05939016164949467,        # Learning rate
    "max_epochs": 100,              # Maximum number of epochs for training
    "batch_size": 64,               # Batch size for training (64, 128, 256, 512, 1024, ...)
    "virtual_batch_size": 32,   # Virtual batch size for gradient accumulation
    "seed": 42                      # Random seed for reproducibility.
}
```

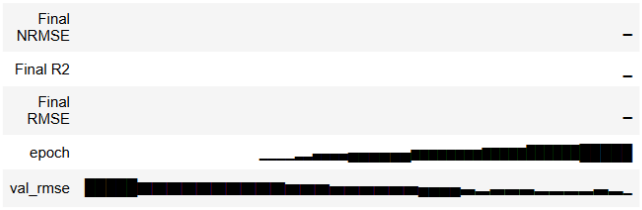
따라서 다음과 같이 향상된 성능의 결과값을 얻었다.

rmse: 247234446.4438

nrmse: 0.0385

R2: 0.4252

Run history:



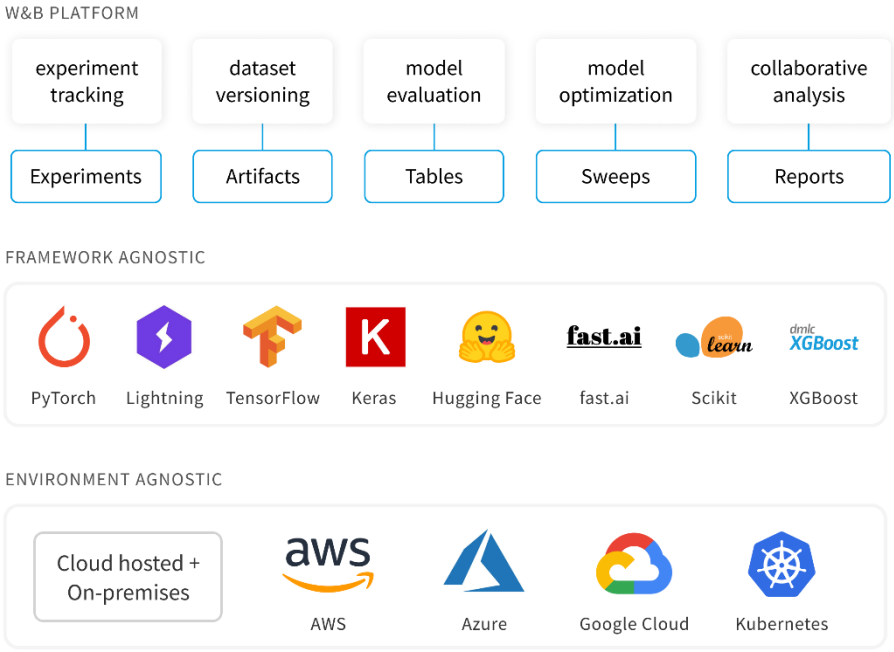
Run summary:

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

6-4. WandB

머신러닝 혹은 딥러닝 모델을 학습할 때에는, 모델을 최적화시키기 위하여 하이퍼파라미터의 값을 적절히 선택해주어야 하고, 원하는 결과를 얻기 위해서는 여러 가지 하이퍼파라미터 값을 변경하며 다양한 실험을 해야한다. 그러나 학습자가 직관적으로 값을 변경하는 경우에는, 매우 비효율적이며, 실험결과를 비교하기 위해 정리하는 과정이 번거롭고 기록이 누락될 위험성이 있다. 따라서 이를 해결하고 효율적인 학습을 할 수 있도록 하는 Tool 인 WandB 를 사용할 수 있다.

WandB(Weights&Biases)는 최적의 모델을 더욱 효율적으로 만들 수 있도록 돕는 Experiment tracking tool 이다. 다음과 같이 유용한 5 가지의 도구를 제공하고 있다.



Experiments : 모델 실험을 추적하기 위한 Dashboard 를 제공한다.

Artifacts : Dataset version 관리와 Model vertion 을 관리한다.

Tables : 데이터를 loging 하여 시각화하고 쿼리를 만드는 데에 사용한다.

Sweeps : 하이퍼파라미터를 자동으로 튜닝하여 최적화한다.

Reports : 실험을 정리하여 협업자들과 공유한다.

이를 통해 협업과 효율적인 프로젝트 관리가 가능하며, 여러 프레임워크와의 결합이 가능하므로 확장성이 뛰어나다는 장점도 있다. 다음과 같이 wandb 에 로그인을 하고, API key 를 입력하여 세팅이 가능하다.

```
# wandB 로그인
wandb.login()
```

```
True
```

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

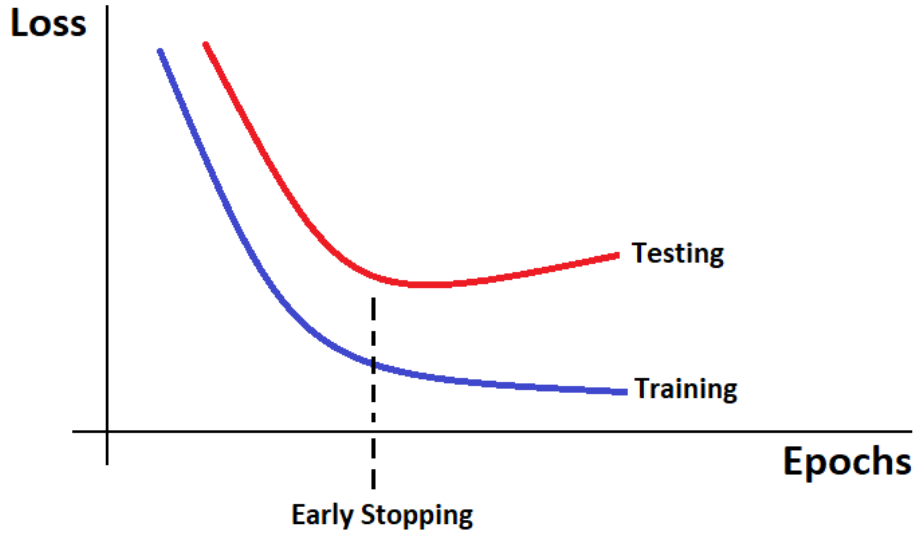
```
Tracking run with wandb version 0.18.7

Run data is saved locally in C:₩Users₩ekgus₩Desktop₩산업수학 팀플₩wandb₩run-20241115_220324-aewopktm

Syncing run ethereal-bee-2 to Weights & Biases (docs)

View project at https://wandb.ai/seoldh-sogang-university/TabNET-regression-tuning

View run at https://wandb.ai/seoldh-sogang-university/TabNET-regression-tuning/runs/aewopktm

Display W&B run
```

공식 홈페이지의 예시에 따르면, 모델 학습과 동시에 학습 단계별로 모델 성능과 손실함수도 추적이 가능하며, 이를 통해 최적의 하이퍼파라미터 조합을 찾을 수 있다.

본 프로젝트의 경우에는, WandB Tracking 을 통해서 TabNet 의 분류 작업을 시도해보았고 다음과 같은 결과를 얻을 수 있었다.

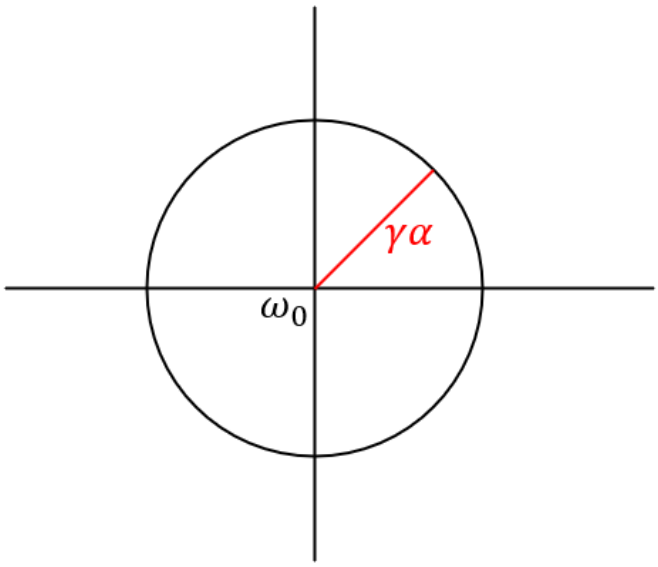
6-5. EarlyStopping

EarlyStopping은 과적합을 방지하기 위한 callback함수로, 적절한 시점에 학습을 조기종료하는 정규화 기법이다. 훈련 중 주기적으로 성능 검증을 하다가 성능의 개선이 보이지 않으면, 즉 더 좋아지지 않으면 과적합으로 판단하고 훈련을 멈춘다. 이 때 성능 검증의 단위는 epoch이며, epoch보다 자주 검증할 시에는 batch 실행 단위로 검증하기도 한다. 이 때 epoch란, 전체 훈련 데이터를 한 번 사용해서 훈련하는 주기를 의미한다. 보통 훈련할 때에는 여러 epoch에 걸쳐서 진행한다.



이처럼 조기종료를 할 때에는 그 기준도 중요한데, 모델의 성능이 즉각 향상되지 않는다고 종료하면 과소적합, 즉 학습이 제대로 되지 않을 수 있다. 신경망을 학습할 시 단계마다 batch로 근사한 gradient는 실제 gradient와 차이가 있기 때문에 성능이 요동칠 수 있다. 따라서 EarlyStopping을 위해서는 조건을 설정해야하는데, 학습 조기종료를 위해 관찰하는 항목인 val_loss 또는 val_accurcay, 개선을 판단하기 위한 최소 변화량, 개선을 위해 몇 번의 epoch를 기다릴지를 설정하는 양 등의 파라미터가 있다.

EarlyStopping의 정규화 효과는 파라미터 공간을 작게 한다는 것에 있다. 초기 파라미터 위치가 ω_0이고, 최적화 스텝 수와 learning rate를 각각 γ, α 라고 한다면 파라미터 공간은 다음과 같이 $\gamma\alpha$ 크기의 반경을 갖는 공간으로 제약된다.



이처럼 EarlyStopping으로 파라미터 공간 크기가 제약되면, L2 정규화인 Ridge regression과 동일한 효과를 가질 수 있다.

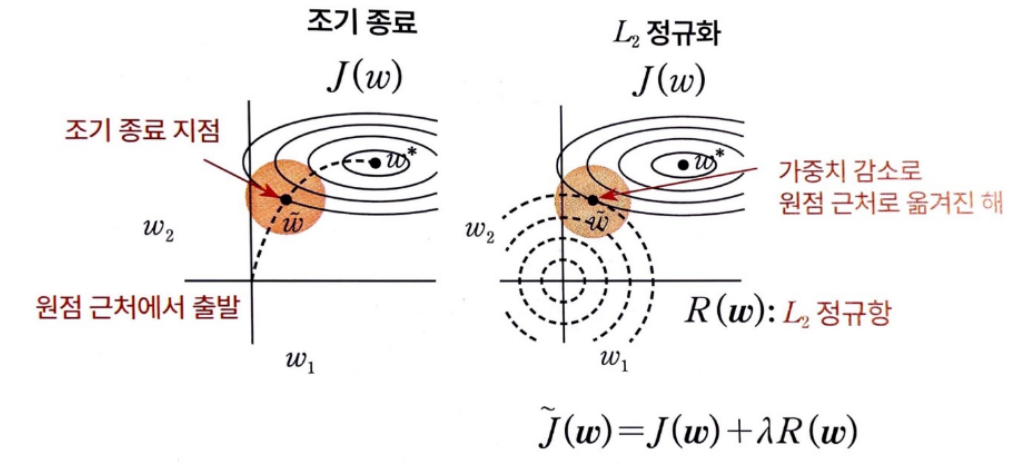


그림 5-32 조기 종료의 L_2 정규화 효과

왜냐하면 EarlyStopping을 할 때, 원점으로부터 출발하여 최적의 해가 있는 방향으로 진행하다가 도중에 멈추는데, EarlyStopping의 위치는 위 그림과 같이 L2 regularization을 했을 때 loss function과 regularization term이 만나는 지점과 유사한 위치이기 때문이다. 따라서 loss function이 2차 함수로 정의되는 선형 모델이라면, EarlyStopping과 L2 regularization는 동일하다는 것이 증명된다.

EarlyStopping은 별도 API 활용 없이 적용 가능하며, 적용한 코드는 다음과 같다.

```
patience = 5
best_val_loss = float('inf')
patience_counter = 0
epochs_max = 300

for epoch in range(epochs_max):
    model.train()
    running_loss = 0.0

    '''
    모델 학습 루프
    '''

    val_loss /= len(val_loader)
    print(f'Epoch [{epoch + 1}/{epochs_max}], ₩
          Loss: {running_loss / len(train_loader):.4f}, ₩
          Val Loss: {val_loss:.4f}')

    # 조기 종료 조건 확인
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        patience_counter = 0
    else:
        patience_counter += 1

    if patience_counter >= patience:
        print(f"Early stopping triggered after {epoch + 1} epochs")
        break
```

분류를 위한 TabNet 알고리즘 적용 시에는, EarlyStopping이 epoch 53에서 일어났고, best_epoch=43, best_val_logloss=0.56424임을 확인할 수 있었다.

7. 결론 및 제언

7-1. 탐구의 의의

크롤링을 통해 원본에는 없었던 동영상 업로드 시점 및 새로 업데이트 된 조회수, 좋아요수, 댓글수를 포함한 양질의 데이터를 확보하였다. 또한 이상치 데이터의 특징을 찾고 전처리하였으며, Feature Engineering을 통해 학습 데이터에 대한 도메인 지식을 통하여, 새롭게 크롤링한 조회수, 좋아요수, 댓글수의 증가량을 새로운 변수로 도입하여 음악적 특성 및 뮤직비디오의 주요 특성을 연관지어 분석하고자 하였다. 이는 기존 머신러닝 기법을 통하여 비교적 적은 연산량 및 자원으로도 준수한 퍼포먼스를 보이는 데에 기여하였다.

한편 딥러닝 기법에서는 Transformer의 Attention 메커니즘을 추가로 적용함으로써, 정형 데이터셋의 약점인, 딥러닝 학습 시 성능이 떨어진다는 점을 극복하고자 했다. 이 과정에서 optuna, WandB, EarlyStopping 등의 다양한 하이퍼파라미터 튜닝 기법을 시도하였다. 특히 WandB를 통해서는 각종 하이퍼파라미터의 변경점을 기록한 실험 결과를 트래킹함으로써 성능의 최적화 지점을 찾아내었다. 또한 시각화 및 결과를 사후 분석함으로써, 단순히 모델의 성능을 끌어올리는 것에 그치지 않고, PCA와 t-SNE를 활용한 군집화 결과를 시각적으로 확인하며 음악적 요소 중에서는 템포(tempo), 키(key)가 확실한 구분점임을 추론하였다. 이처럼 결과를 정량적으로 분석하고자 시도했다는 점에서 의의가 있다.

이를 통해 기존에 연구되어 왔던 음원과 뮤직비디오 간 상관관계에 대한 이론에 대해서도 다시 고찰해 볼 수 있었다. 특히 K-pop을 비롯한 음악들은 2004년을 분기점으로 음반에서 음원으로 이동하는 매체의 대전환이 이뤄지며, 음악산업에서 매체 구조의 변화와 뮤직비디오의 상관관계는 매우 크다.[6] 또한 음악산업에서도 뮤직비디오, 즉 Youtube라는 매체의 중요성이 대두되며, 유튜브 뮤직 비디오의 조회수가 음악 그 자체뿐 아니라, 시각적 연출에도 영향을 받고 있음[7]을 다시 한 번 검증할 수 있었다.

---

[6] 이종훈. 2021, "음악산업의 매체전환과 뮤직비디오 제작 양상의 변화", 문화콘텐츠연구, no.21, pp.137-162. Available from: doi:10.34227/tjocc.2021..21.137

[7] 강윤후, 박진호. (2020). 유튜브 뮤직 비디오의 조회수와 시각적 연출 특성의 관계에 대한 연구 : 케이팝 뮤직 비디오를 중심으로. 한국엔터테인먼트산업학회논문지, 14(8), 63-75.

7-2. 한계점

탐구를 진행함에 있어서, 가설을 수립하고 변인을 통제하는 과정이 체계적으로 이루어졌다고 보기 어려운 부분이 있다. 특히, 명확한 가설을 세우고 연구를 진행했다기 보다는, 데이터 전처리와 회귀, 분류, 클러스터링의 머신러닝 알고리즘을 적용해보며 데이터를 차차 이해하고 상관관계를 찾아가는 진행 방식에 가까웠다. 구체적인 연구 절차 및 방식에 철저한 계획을 세우지 않고 진행 경과를 지켜보며 유동적으로 탐구를 진행함에 따라, 연구의 전문성 및 신뢰도에서 미흡한 부분이 존재한다. 따라서 음악과 대중문화라는 분야에 대한 수준 높고 전문적인 지식을 갖추어 체계적으로 진행하였다면, 더욱 의미있고 완성도 높은 연구가 되었을 것으로 본다.

또한 데이터셋에 존재하는 음악 특성 및 조회수, 좋아요수, 댓글수라는 feature 간의 연관성(상관계수) 및 회귀적 연관성이 존재한다는 다소 추상적인 패턴은 파악하였으나, 더욱 심층적인 분석까지 나아가기는 어려웠다. 이에 따라, 연구에서 진행하고 해결하고자 하는 주제와 관련하여 창의적이거나 혁신적인 새로운 인사이트를 발견했다고 보기엔 어려운 부분이 있다.

마지막으로 딥러닝 모델을 사용함에 있어서, 이미 존재하고 사용되고 있는 하이퍼파라미터 튜닝 및 최적화 기법을 시도한 것에 그쳤다는 점이 아쉬운 부분이다. 추후 연구를 발전시킨다면, learning rate scheduler 적용 및 다양한 활성화 함수를 적용함으로써, 모델의 구조적 개선을 시도해볼 수 있을 것이다. 나아가 feature importance 추출 과정에서 SHAP이라는 새로운 시각화를 시도하였으나, 느린 속도나 정형 데이터와 모델 구조와의 호환성 문제, 낮은 효율성으로 인해 의미있는 시각화 결과를 도출하지 못한 점이 아쉬운 부분이다. 데이터셋의 충분한 확장과 SHAP을 다루는 기술이 숙련된다면 이후 충분한 모델 해석력을 확보할 수 있을 것이다.

7-3. 보완점

본 연구에서 시도하고 검증한 내용을 바탕으로 하여, 앞으로 추가적인 후속 연구가 진행된다면 다음 주제에 대하여 보다 심층적으로 분석할 것을 제언한다.

1. 각각의 음악적 특성이 뮤직비디오의 특성과 조회수에 어떠한 영향을 미치며, 그 양상은 독립적인지, 다른 특성들과도 연관되는지를 분석해볼 수 있다. 특히 서로 유기적 관계를 보이는 feature인 Danceability와 Valence와 같은 경우, 구체적으로 어떠한 기제를 통하여 뮤직비디오 조회수에 영향을 미치는지, 그 논리적인 과정을 검증하고 제시한다면 보다 신뢰도 높은 분석이 될 것이다.
2. 스포티파이와 유튜브, 즉 음원과 영상 사이에 서로 영향을 미치며 작용하는 요소들이 음악과 직접적으로 관련되어 있지 않은 특성은 없는지 검토해야 한다. 예를 들면 창작물을 업로드한 계정의 구독자 수 또는 음원을 발매한 시기의 시대적 상황과 같은 것들이다. 이때 단순히 조회수와 특성 간의 상관관계만을 분석하기 보다는, 그 관계를 정량적으로 분석할 수 있다면 더욱 타당한 분석이 될 것이다. 예를 들면, 세계적으로 유명한 창작자들이 제작한 뮤직비디오와, 비교적 인지도가 낮은 음원의 뮤직비디오 영상에 음악적 특성이 조회수에 영향을 미치는 방식은 달라질 수 있다는 점이다. 즉 대중음악의 흥행을 결정하는 요인과 공연성에는 다양한 요인들이 영향을 줄 수 있고 이를 규명해야 한다는 것이다[8]. 따라서 음악적 특성에 따른 조회수 변화 추이는 다른 요인에 따라 달라질 수 있음을 고려하여 새로운 분류 및 분석 기준을 마련해야 한다.

이를 통해 폭넓고 다양한 플랫폼에서 음원 창작자 및 관련업 종사자들에게 대중의 인기를 어느 정도 예측 가능하면서도 독창적이고 신선한 아이디어를 담은 창작 전략을 제시하고, 대중음악을 중심으로 한 문화의 생산 및 소비자들에게 실천적인 보탬이 되기를 기대한다.

[8] 이남미 외 3. 2019, " 대중음악 흥행결정요인과 공연성과와의 관계 ", 한국콘텐츠학회논문지, Vol.19 Issue7, pp.54-66. Available from: https://doi.org/10.5392/JKCA.2019.19.07.054