교과명 : 머신러닝

담당 교수 : 윤직혁 교수님

음악의 음향적 특징을 활용한 인기도 예측

- Spotify 데이터 분석 -

학과 : AI융합학부

학번 : 20241992

이름 : 최희우

문제 정의

이번 프로젝트의 최종 목표는 음악의 음향적 특성과 메타데이터를 분석하여 노래의 인기도를 사전에 예측할 수 있는 모델을 구축하는 것이다.

즉, 과거에 존재하는 수많은 노래들의 특성과 인기도 데이터를 학습하여 새로운 노래가 발매되었을 때 그 노래가 대중적으로 성공할 가능성을 수치적으로 예측하고자 한다.

이를 통해 음악 제작자나 프로듀서는 청취자들에게 인기가 높을 것으로 예상되는 음악적 특성을 사전에 파악하고 음악 기획이나 마케팅 전략을 보다 데이터 기반으로 수립할 수 있을 것이다.

연구 가설은 다음 세가지를 바탕으로 진행하려 한다.

첫째, 노래의 핵심 음향적 특징인 'danceability', 'energy', 'valence', 'loudness', 'tempo' 등은 노래의 인기도에 유의미한 영향을 미칠 것이다. 곡의 리듬감, 강도, 밝기 등의 요인은 청취자들의 감정적 반응과 반복 재생 여부에 직접적인 영향을 줄 가능성이 높기 때문이다.

둘째, 노래의 장르와 발매 시기는 음향적 특징과 결합하여 인기도 예측의 정확도를 높이는 보조적 요인으로 작용할 것이다. 이는 음악의 시대적 트렌드와 장르별 특징이 대중적 선호도에 영향을 줄 수 있기 때문이다.

셋째, 인기가 높은 곡들의 음향적 특징 분포는 장르에 따라 다를 것이며 각 장르별로 대표적인 음향적 조합이 존재할 것이다. 예를 들어 팝 장르는 일반적으로 valence와 danceability가 높고 재즈는 acousticness와 instrumentalness가 높은 경향을 보일 것이다.

위의 목표와 연구 가설을 바탕으로 살펴본 구체적인 문제들은 아래와 같다.

곡의 인기도에 가장 큰 영향을 미치는 음향적 특징은 무엇인지, 장르별 인기 곡의 음향적 특징 분포는 어떻게 다르며 음향적 특징과 장르, 발매 연도 정보를 함께 사용할 경우 인기도 예측 정확도는 얼마나 향상되는지 회귀 접근법과 분류 기반 접근 법 중 어느 방식이 더 높은 성능을 보이는 지이다. 이들을 알아보려고 한다.

머신러닝적으로 task는 크게 회귀와 분류를 사용하려 한다.

회귀 접근은 노래의 인기도를 0~100사이의 연속적인 값으로 예측한다. 이를 위해 선형 회귀, 랜덤 포레스트 회귀, XGBoost회귀 등의 모델을 적용할 수 있을 것이다.

분류접근은 인기도를 일정 기준을 기준으로 인기곡과 비인기곡으로 구분하여 예측하는 방식이다. 이 경우 로지스틱 회귀, 랜덤 포레스트, 그래디언트 부스팅 등의 모델을 활용할 수 있을 것이다.

따라서 이번 프로젝트에서는 회귀, 분류 두 가지 접근법을 병행하여 인기도 예측의 정량적, 정성적 분석을 모두 수행하고 각 모델의 예측 성능을 비교함으로써 종합적인 결론을 도출하고자 한다.

데이터 정의

이 모델을 구축할 때 사용할 데이터는 kaggle에서 제공한 데이터로 "30000 Spotify Songs"(출처 : https://www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs?select=spotify_songs.csv) 데이터셋을 활용하려고 한다. 이 데이터는 Spotify의 오디오 특성 API를 통해 수집된 대규모 데이터로 총 약 30000개의 노래에 대한 정보가 포함되어 있다.

각 곡은 장르, 발매일 등의 메타데이터 뿐만 아니라 음향적 속성을 수치화한 변수도 함께 포함하고 있어 내적 특성, 외적 맥락이 인기도에 미치는 영향을 동시에 분석하기에 적합하다 판단하였다.

이 데이터의 변수들은 아래와 같다.

변수명	자료형	설명
track_id	character	노래의 고유 ID
track_name	character	노래 제목
track_artist	character	노래 아티스트 이름
track_popularity	double	노래의 인기도를 0~100 사이의 값으로 나타냄. 값이 높을수록 대중적 인기가 높음을 의미함.
track_album_id	character	앨범의 고유 ID
track_album_name	character	노래가 수록된 앨범 이름
track_album_release_date	character	앨범의 발매일
playlist_name	character	재생목록(플레이리스트) 이름
playlist_id	character	재생목록의 고유 ID
playlist_genre	character	재생목록의 장르
playlist_subgenre	character	재생목록의 하위 장르
danceability	double	곡이 춤추기에 얼마나 적합한지를 나타내는

		지표. 템포, 리듬 안정성, 비트 강도 등 여러 음악적 요소를 종합하여 산출하며, 0.0은 가장 덜 춤추기 적합하고 1.0은 가장 적합함.
energy	double	곡의 에너지(활동성, 강도)를 0.0~1.0 범위로 표현. 빠르고, 시끄럽고, 강한 곡일수록 값이 높음.
key	double	곡의 전체적인 조성을 숫자로 표현. 예: 0=C, 1=C♯/D♭, 2=D 등. 조성이 감지되지 않으면 -1로 표시.
loudness	double	곡 전체의 평균 음량(dB). 일반적으로 -60~0 dB 범위이며, 값이 높을수록 음량이 큰 곡임.
mode	double	곡의 조성 모드. 1은 장조(major), 0은 단조(minor)를 의미함.
speechiness	double	트랙 내 말소리의 비율을 측정. 값이 1에 가까울수록 말 위주의 녹음(예: 랩, 오디오북), 0.33 미만은 일반적인 음악을 의미함.
acousticness	double	곡이 어쿠스틱(자연음) 기반일 확률을 0.0~0.1으로 표현, 1.0은 매우 어쿠스틱한 곡을 의미함.
instrumentalness	double	곡에 보컬이 없는 정도를 나타냄. 값이 1.0에 가까울수록 악기 연주만으로 구성된 곡일 가능성이 높음. 0.5 이상이면 일반적으로 기악곡으로 간주됨.
liveness	double	청중(라이브 공연)의 존재 가능성을 나타내는 지표. 값이 0.8 이상이면 실제 공연에서 녹음된 곡일 확률이 높음.
valence	double	곡이 전달하는 감정적 긍정성. 0.0~1.0 사이의 값으로, 높을수록 밝고 행복한 분위기를, 낮을수록 슬프고 어두운 분위기를 나타냄.
tempo	double	곡의 전체적인 템포(BPM, 분당 박자 수). 음악의 빠르기나 리듬의 속도를 나타냄.
duration_ms	double	곡의 길이(재생 시간)를 밀리초 단위로 표현.

이 중 track_popularity는 예측 대상변수로 사용하며 나머지 변수들은 입력 변수로 활용될 것이다. 모델 구축 시 필요없는 ID는 분석에서 제외하며 다른 변수들도 전처리 과정을 통해 정리될 것이다.

데이터 처리 과정

이번 프로젝트에서는 Spotify 공개 데이터를 기반으로 음악적 특성과 노래 인기도 간의 관계를 분석하기 위해 모델링 과정에 앞서 다음과 같은 전처리 과정을 수행하였다.

데이터셋 전체(32,833행) 중 표본 5개에서 track_name, track_artist, track_album_name에만 결측치가 존재하였다. 이 세 변수는 모델링에서 직접 사용하지 않는 문자열형 변수이며 결측치 비율 또한 0.015%로 매우 낮아 분석에 미치는 영향이 적다고 판단하여 결측치를 포함한 행을 제거하였다.

또한 발매일자(track_album_release_date)를 연도 단위로 변환하는 과정에서 일부 비정상적인 날짜 형식이 존재하여 release_year에서 NaN이 발생하였다. 이 변수는 모델 입력으로 반드시 필요하므로 release_year에 결측치가 포함된 1,166개 행은 제거하였다.

track_album_release_date는 문자열("YYYY-MM-DD") 형태로 제공되며 전체 날짜를 feature로 사용하는 것은 비효율적이라고 판단하였다. 따라서 pd.to_datetime()을 사용하여 날짜를 datetime 타입으로 변환하고 .dt.year를 이용해 연도만 추출하여 release_year 변수를 생성하였다. 이는 발매 연도에 따른 인기 변화 분석을 가능하게 하기 위해 수행하였다.

다음으로 학습에 의미를 갖지 않거나 모델에 직접 사용하기 어려운 문자열 변수들은 제거하였다.
제거한 컬럼은 다음과 같다.

- track_id, playlist_id 등 고유 식별자
- track_name, track_artist, playlist_name 등 텍스트 기반 식별 정보
- track_album_name 및 playlist_subgenre 등 문자열 기반 범주형 정보
- track_album_release_date

해당 변수들은 모델 학습에 직접적으로 기여하지 않으며 문자열 상태 그대로는 수치 모델에 입력할 수 없기 때문에 제거하였다.

Spotify의 장르 정보인 playlist_genre는 문자열 변수이므로 모델에서 직접 사용이 불가능하여 이를 해결하기 위해 원핫 인코딩(one-hot encoding)을 적용하였으며 원핫 인코딩을 하면 비슷한 정보가 여러 컬럼에 중복되어 모델이 혼란스러울 수 있어 drop_first=True 옵션을 사용해 하나의 컬럼을 제거하여 중복을 없애고 모델이 안정적으로 학습할 수 있도록 하였다.

음악적 특성 중 일부 변수에서 극단값이 확인되었다. 특히, loudness, speechiness, acousticness, liveness, valence 등의 분포에서 IQR(Interquartile Range) 기반 이상치가 존재했다. duration_ms와 tempo는 음악 특성상 분산이 큰 것이 자연스러우므로 제거 대상

에서 제외하였다.

제거한 이상치의 기준은 다음과 같다.

$$\text{Lower} = Q1 - 1.5 \times IQR, \quad \text{Upper} = Q3 + 1.5 \times IQR$$

이를 통해 전체 데이터 중 일부 극단값을 제거하였다.

머신러닝 모델의 튜닝 과정

데이터를 분석하기 위해 여러 머신러닝 모델을 적용하였다. 처음에는 회귀와 분류 모델을 모두 사용해 인기도를 다양한 방식으로 예측하는 것을 목표로 하였다. 그러나 실제 데이터를 분석해본 결과 음향적 특징과 인기도의 관계가 선형적이지 않아 선형 회귀 모델의 성능이 낮게 나타났다. 또한 XGBoost와 LightGBM과 같은 부스팅 모델은 데이터 규모에 비해 모델 복잡도가 크고 파라미터 튜닝 비용이 높아 baseline 대비 유의미한 성능 향상을 확인하기 어려웠다. 반면 RandomForest는 비선형 관계를 잘 반영하면서도 해석이 가능해 가장 높은 성능을 보여 핵심 모델로 선정하였고 인기도를 점수로 예측하는 회귀보다 인기/비인기 분류 방식이 연구 가설을 검증하는 데 더 적합해 분류 중심으로 분석 방향을 조정하였다. 따라서 최종적으로 Linear Regression, Logistic Regression, RandomForest 중심으로 분석을 진행하였다. 각 모델은 구조와 작동 방식이 서로 다르기 때문에 모델의 특징을 고려하여 적절한 전처리와 하이퍼파라미터 튜닝을 진행하였다.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Feature/Target 분리
X = df.drop("track_popularity", axis=1)
y = df["track_popularity"]

# Train/Test 분할
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 모델 생성
lr = LinearRegression()

# 학습
lr.fit(X_train_scaled, y_train)

# 예측
pred = lr.predict(X_test_scaled)

# 평가
print("MSE:", mean_squared_error(y_test, pred))
print("R2:", r2_score(y_test, pred))
```

✓ 0.0s

```
MSE: 568.4826815927083
R2: 0.10965456586728406
```

우선 연속적인 track_popularity를 직접 예측하기 위해 가장 기본 모델인 Linear Regression(선형 회귀)을 사용하였다. 선형 회귀는 입력 값들이 선형적으로 목표 변수에 영향을 준다고 가정하는 단순하고 직관적인 모델이다. 이 모델은 특성 값들의 크기가 서로 다르면 학습이 불안정해질 수 있어 먼저 StandardScaler를 이용해 값을 동일한 스케일로 맞춘 뒤 학습을 진행했다. 하지만 음악적 특성은 서로 복잡하게 얽혀 있고 완전히 선형적 관계만으로 설명되기 어렵기 때문에 선형 회귀는 비교적 낮은 성능을 보였다. 그럼에도 불구하고 선형 회귀는 가장 기본적인 회귀 모델이므로 이후 다른 모델들과 성능을 비교하는 기준으로 활용하였다.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score
import numpy as np

# 이진 클래스 생성
df['pop_class'] = (df['track_popularity'] > df['track_popularity'].median()).astype(int)

# X, y 분리
X = df.drop(['track_popularity', 'pop_class'], axis=1)
y = df['pop_class']

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 모델 생성
logi = LogisticRegression(
    max_iter=1000,
    solver='liblinear',   # 일반적으로 안정적
    C=1.0,                # regularization strength
)

# 학습
logi.fit(X_train_scaled, y_train)

# 예측
pred = logi.predict(X_test_scaled)

# 성능 평가
print("ACC:", accuracy_score(y_test, pred))
print("F1:", f1_score(y_test, pred))
```

✓ 0.0s

```
ACC: 0.6096272380393308
F1: 0.6074380165289256
```

다음으로, 인기도를 "높음/낮음" 두 그룹으로 나누어 예측하기 위해 Logistic Regression(로지스틱 회귀)을 적용하였다. 먼저 popularity의 중앙값을 기준으로 새로운 이진 타깃 변수(pop_class)를 만들고 마찬가지로 StandardScaler를 통해 모든 특성을 정

규화한 뒤 모델을 학습시켰다. 로지스틱 회귀는 선형 회귀와 비슷하게 직선 형태의 결정 경계를 사용하는 모델로 단순하지만 빠르게 학습되고 해석도 쉬운 장점이 있다. 다만 이 모델도 데이터가 비선형적인 구조를 갖는 경우에는 복잡한 패턴을 충분히 반영하지 못하는 특성이 있다. 따라서 로지스틱 회귀는 본 연구에서 기본적인 분류 성능을 확인하기 위한 baseline 모델로 사용하였다.

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

# X, y 정의
X = df.drop(["track_popularity", "pop_class"], axis=1)
y = df["pop_class"]

# train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# RandomForest는 스케일링 필요 없음
rf = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)

# 학습
rf.fit(X_train, y_train)

# 예측
rf_pred = rf.predict(X_test)

# 평가
print("ACC:", accuracy_score(y_test, pred))
print("F1:", f1_score(y_test, pred))
```

✓ 1.1s

```
ACC: 0.6096272380393308
F1: 0.6074380165289256
```

이후 음악적 특성 간의 복잡한 관계를 반영하기 위해 RandomForest Classifier를 활용하였다. 랜덤포레스트는 여러 개의 의사결정트리를 학습한 뒤 그 결과를 종합하여 예측하는 방식으로 비선형적인 패턴이나 변수 간의 상호작용을 자연스럽게 학습할 수 있다. 선형 모델과 달리 특징들 간의 곱, 비선형 변화 등 복잡한 패턴까지 반영해 성능이 크게 향상되었다. 기본 파라미터로 학습했을 때부터 로지스틱 회귀보다 더 높은 정확도와 F1-score를 보였기 때문에 이번 분석에서는 이 모델을 중심으로 성능 개선을 위한 튜닝을 진행하였다.

```python
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    'n_estimators': [200, 300, 400, 500, 700],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8],
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False]
}

rf = RandomForestClassifier(random_state=42)

random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=30,
    cv=3,
    scoring='f1',
    verbose=1,
    n_jobs=-1,
    random_state=42
)

random_search.fit(X_train, y_train)

print("Best Parameters from RandomSearchCV:")
print(random_search.best_params_)

best_rf = random_search.best_estimator_

pred = best_rf.predict(X_test)
print("ACC:", accuracy_score(y_test, pred))
print("F1:", f1_score(y_test, pred))
```

✓ 3m 7.0s

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best Parameters from RandomSearchCV:
{'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': 40, 'bootstrap': True}
ACC: 0.7067801584972117
F1: 0.7004497751124438
```

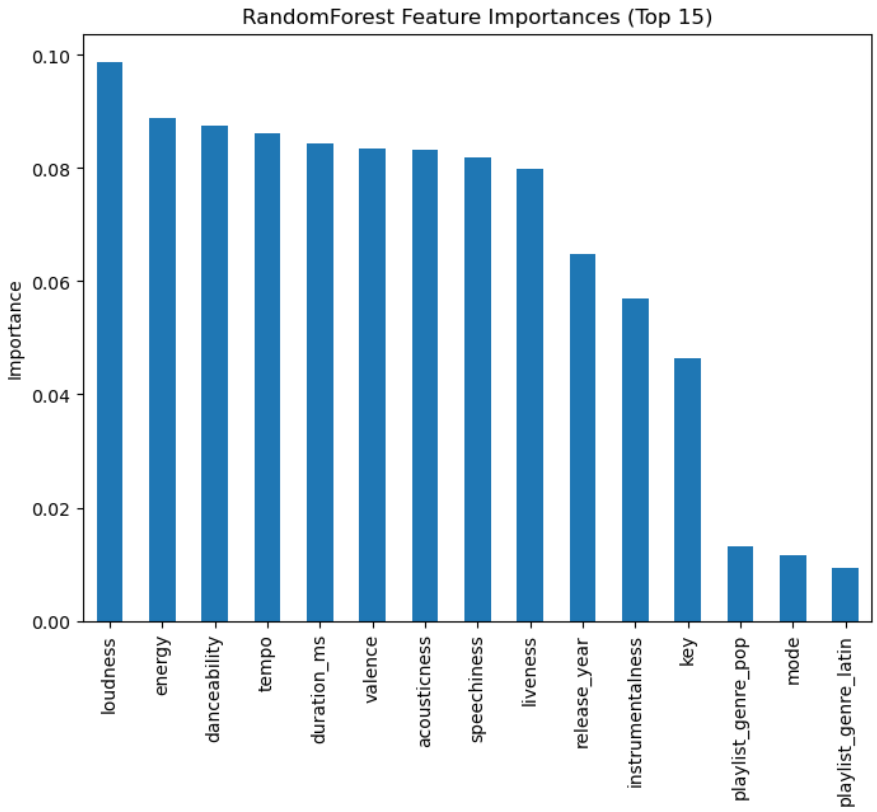
튜닝 과정에서는 랜덤포레스트의 주요 파라미터인 트리 개수(n_estimators), 트리 최대 깊이(max_depth), 노드를 나누기 위한 최소 샘플 수(min_samples_split), 리프 노드 최소 샘플 수(min_samples_leaf), 특성 선택 방식(max_features), 부트스트랩 샘플링 여부(bootstrap) 등을 조정하였다. 이 파라미터들은 모델이 과적합되는지 혹은 복잡도가 얼마나 되는지를 크게 좌우하는 요소이다. 모든 경우를 일일이 탐색하는 GridSearch는 연산량이 너무 크기 때문에 이번 분석에서는 RandomizedSearchCV를 사용해 지정된 범위 내에서 30개의 조합만 무작위로 선택하여 교차검증을 진행했다. 이 방식은 시간이 훨씬 적게 들면서도 성능이 좋은 파라미터 조합을 효율적으로 찾을 수 있는 장점이 있었다.

튜닝을 통해 선정된 최적의 파라미터로 모델을 재학습한 결과 기본 설정으로 학습한 랜덤포레스트보다 정확도와 F1-score가 모두 상승하여 가장 높은 예측 성능을 보이는 모델을 구축할 수 있었다. 이 과정은 트리 기반 비선형 모델이 음악적 특성과 인기도의 복잡한 관계를 보다 잘 반영한다는 점을 고려하여 진행되었으며 튜닝을 통해 모델의 복잡도와 일반화 성능을 균형 있게 향상시켰다 볼 수 있었다.

제시된 문제에 대한 데이터 분석 및 결론

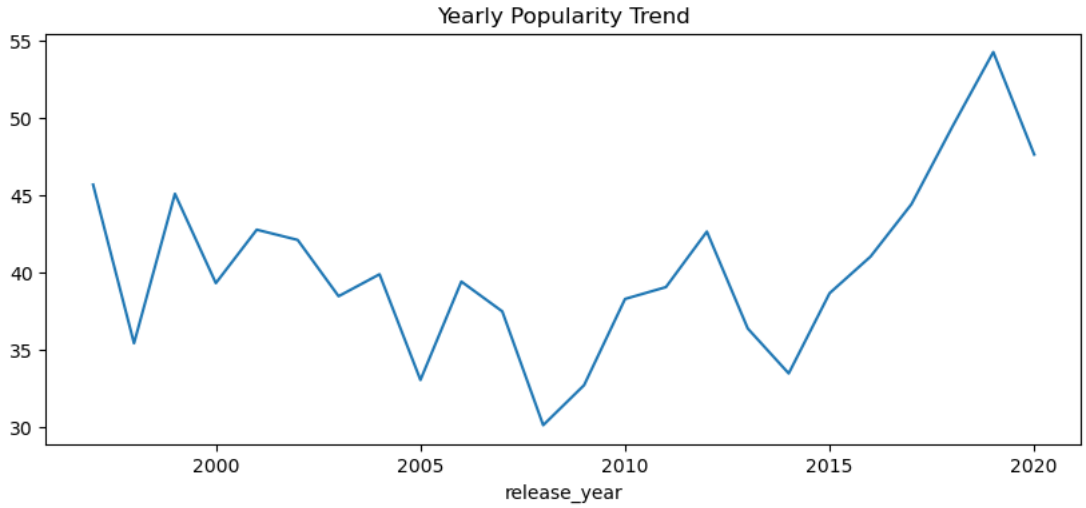
Linear Regression을 적용한 회귀 분석 결과 R^2는 0.11 수준으로 나타나 선형 모델의 설명력이 매우 제한적임을 확인하였다. 인기도를 상, 하위 그룹으로 구분한 분류 문제에서도 Logistic Regression은 ACC 0.61, F1 0.61의 낮은 성능을 보였다.

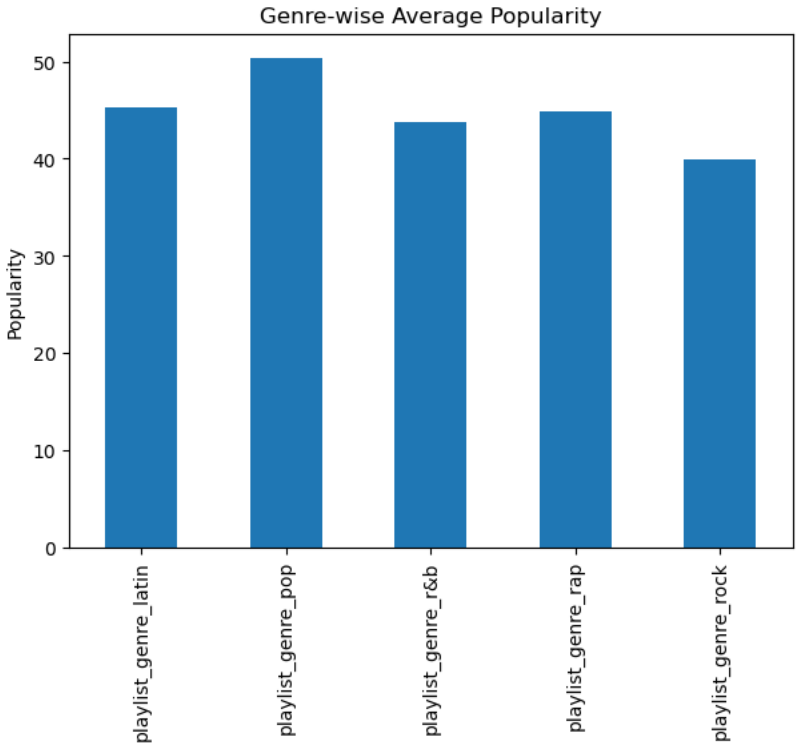
반면, 트리 기반 비선형 모델인 RandomForest Classifier를 적용했을 때는 ACC 0.70, F1 0.695로 분류 성능이 크게 향상되었다. 이는 음악의 인기라는 복잡한 현상이 단순한 선형 관계보다는 비선형적 상호작용을 통해 설명되며 분류 문제로 접근하는 것이 회귀 접근보다 인기도의 패턴을 더 명확히 포착할 수 있음을 알 수 있었다. 하이퍼파라미터 튜닝(RandomizedSearchCV) 후 성능은 ACC 0.7068, F1 0.7004로 조금 개선되었다.



Feature importance 분석 결과 loudness, energy, danceability 등이 인기도 예측에 가장 중요한 영향 요인으로 나타났다. 이 특성들은 곡의 소리 강도, 에너지감, 리듬감 등 청취자의 몰입도에 직접적으로 관여하는 음악적 요소라는 점에서 일반적인 음악적 해석과 일치한다. 이외에도 duration_ms와 valence 또한 일정 수준의 영향력을 보였으며 release_year는 대중의 취향 변화를 반영하는 보조적 트렌드 요인으로 작용하였다. 반면 instrumentalness나 speechiness와 같은 특성은 상대적으로 영향력이 낮게 나타났다. 장

르 더미 변수 역시 음향적 특성에 비해 상대적으로 낮은 중요도를 보였다.





연도와 장르는 각 범주별로 인기도 분포가 다르게 나타나 인기도에 영향을 주는 것으로 보이나 전체 feature 중요도 관점에서는 그 영향이 크지 않아 둘 모두 보조적 요인으로만 작용함을 확인할 수 있었다. 또한 장르 간 음향적 특성 차이가 있을 것이라는 초기 가설과 달리 실제로는 인기가 높은 곡들의 음향적 특성은 장르별로 크게 다르지 않음을 확인하였다.

종합적으로 이번 분석을 통해 비선형 모델이 선형 모델보다 인기도 예측에 훨씬 적합하며 음악적 특징 중에서는 loudness, energy, danceability 등 리듬, 강도 감정적 요소가 인기에 가장 큰 영향을 미친다는 점을 확인하였다. 반면 장르나 발매 연도는 음악적 핵심 요소에 비해 보조적인 역할에 그치는 것으로 나타났다.