COSE362

Final Project

Kaggle Spotify Dataset을 이용한 인기있는 음원의 분류

정보대학 컴퓨터학과

2020320010

전병우

2022.12.21

0. 실험 내용에 대한 전체 요약

본 실험은 5개의 서로 다른 모델에 대해 default, feature selection method, hyperparameter search를 진행한 뒤 cross-validation을 수행하여 성능이 가장 좋은 모델을 선택하였고 이는 test set에서 약 80%의 accuracy를 기록하였다. 실험 과정 및 결과 분석을 통해 본 실험의 데이터에 대해 앙상블 모델이 가장 효과적임을 알 수 있었다.

1. 데이터 출처 및 데이터 설명

Data URL : https://www.kaggle.com/datasets/theoverman/the-spotify-hit-predictor-dataset

데이터는 Kaggle(https://www.kaggle.com) 데이터셋 중 하나로 음원 스트리밍 및 미디어 서비스 제공 업체인 스포티파이(Spotify)의 1960년부터 2019년 사이 발매된 곡에 대한 정보를 담고 있는 데이터셋이다. Raw data는 10년 단위(e.g. 1960~1969)로 묶인 csv 파일로 제공되었으며 본 실험을 위해서 각각의 csv파일을 모두 합치고 전처리 후 train, valid, test 데이터셋을 분리한 뒤 classification을 수행하였다.

데이터셋에는 총 19개의 attribute가 있으며 38977개의 instance로 구성되어 있다. 각 attribute의 상세 설명은 다음과 같다.

1) Track : 음원(곡)의 이름이다.
2) Artist : 음원을 발매한 아티스트의 이름이다.
3) Uri : 트랙의 리소스 식별자이다.
4) Danceability : 템포, 리듬 안정성, 비트강도, 전반적 규칙성에 기반하여 춤을 추기 적합한지 설명하는 정도로 0.0~1.0 사이의 값을 갖는다.

5) Energy : 음악의 강도로 빠르고 시끄러울수록 1.0에 가깝고 느리고 음량이 작을 경우 0.0에 가깝다.

6) Key : 트랙의 추정된 전체 키로 정수가 표준 pitch class notation에 mapping된다. 검색된 key값이 없으면 -1이다.

7) Loudness : 음량값이다.

8) Mode : 트랙의 modality이다. Major는 1, Minor는 0이다.

9) Speechiness : 트랙에서 말하는 단어의 존재값이다. 0.66 이상은 구어에 가까운 트랙이며 0.33~0.66이 일반적인 범주에 속한다.

10) Accousticness : 어쿠스틱 정도에 따른 0.0~1.0의 신뢰 측정값이다.

11) Instrumentalness : 트랙에 보컬이 포함되어 있지 않은 지의 여부이다. "우"와 "아" 같은 소리는 악기로 취급되며 1.0에 가까울수록 보컬 컨텐츠가 포함되지 않을 가능성이 높다.

12) Liveness : 음원 내 청중의 존재를 감지한다. 라이브 음원인지 판별할 수 있는 값으로 0.8이상에서 활성 상태일 가능성이 높다.

13) Valence : 음원을 통해 전달되는 음악적 긍정성으로 0.0부터 1.0 사이의 값을 갖는다. 낮은 값을 가질수록 슬픔, 우울, 분노에 가깝다.

14) Tempo : 트랙의 전체 예상 속도(분당 비트 수)이다.

15) Duration_ms : 트랙의 길이이다. (단위 : ms)

16) Time_signature : 음원의 예상 signature로 각 bar에 얼마나 많은 비트가 있는지 의미하는 값이다.

17) Chorus_hit : 언제 후렴구가 시작될 것인지에 대한 최선의 추정치로 타임스탬프이다. (단위 : ms)

18) Section : 음원에 있는 섹션의 수이다.

19) Target : 음원의 target variable이다. 노래가 발매되고 10년 동안 빌보드

발행 Hot-100의 주간 목록에 적어도 한번 실리면 히트(1)이며 그 외에는 플롭(0)이다.

3. 실험 설계 및 방법

인기있는 음악은 어떠한 특징을 가질까? 연구자는 버스커버스커의 <벚꽃엔딩>을 들으며 히트곡들은 저마다 개성을 가지면서도 비슷한 속성을 가질 것이란 예상을 하였고 히트곡들이 가지는 특성이 궁금하였다. 본 탐구에서는 1960년부터 2019년까지의 음원으로부터 추출된 특징을 바탕으로 히트곡이 되기 위해서 어떠한 속성을 지녀야 하는지 다양한 기계학습 방법을 적용하여 분석한 후 최종적으로 과거 음원들의 데이터에 기반한 히트곡 분류기를 만들고자 한다.

3-1. 모델 선정 이유

본 탐구에서는 총 5개의 모델을 선정하였다.

1) Logistic Regression
2) Decision Tree
3) Naïve Bayesian
4) Support Vector Machine
5) Random Forest

이 중 5)의 랜덤 포레스트의 경우 앙상블 모델(Ensemble Model)이다.

위 모델들을 선정한 이유는 다음과 같다.

1) 로지스틱 회귀는 이벤트가 발생활 확률을 결정하는데 사용되는 클래식한 모델로 시그모이드를 이용해 baseline을 잘 잡을 수 있을 것이라 기대했기 때문이다.
2) 의사결정 나무는 비선형성을 고려하여 수학적 가정이 불필요한 비모수적 모형이므로 본 데이터들에 대한 explicit한 수학적 가정을 잘 회피할 수 있을 것이라 기대하였다.
3) 나이브 베이즈는 간단하며 빠르고 효율적인 알고리즘이란 측면에서

baseline을 잡을 수 있을 것이다.

4) SVM은 범주나 수치 예측에 강력하고 kernel technique을 통한 accuracy 향상을 기대해볼 수 있어서 선정하였다.

5) 랜덤 포레스트는 앙상블 모델로 overfitting 문제를 최소화하여 accuracy를 높일 수 있을 것이라 기대하여 채택하였다.

3-2. 데이터 전처리(Data Pre-processing)

먼저 attribute 중 track과 artist, uri는 각각 곡명과 아티스트명, 리소스 식별자이므로 다른 numerical attribute보다 영향력이 적다고 판단하여 전처리 과정에서 drop한다.

위와 같은 처리를 마치면 16개의 numerical attribute만 남게 되며 위 value들을 이용하여 데이터셋을 재가공하고 모델을 만든다.

```
def generate_dataset():
    drops = ['track', 'artist', 'uri']
    dataset_train, dataset_test, dataset_validate = get_dataset_seperated(train_size=0.8, validate_size=0.0, cols_to_drop=drops)

    return dataset_train, dataset_test, dataset_validate
```

Figure 1.

Figure 1의 generate_dataset 함수에서 전체 데이터셋 중에서 train, validation, test 데이터셋의 크기를 정의한다. Train 데이터셋은 전체 데이터셋 중 80%를 차지할 예정이며 validation은 이후 모델 training 과정에서 train 데이터셋에 대해 scikit-learn 함수 cross-validation을 사용할 것이므로 여기서는 비율을 갖지 않도록 만들었다. 따라서 test 데이터셋은 전체 데이터셋의 20%를 차지한다.

```python
def get_dataset_seperated(train_size = 0.8, validate_size = 0.0, cols_to_drop = ['track', 'artist', 'uri']):
    """
    For each dataset of each decade, get random 20% and add it to a dataset
    """

    total_index = ["60", "70", "80", "90", "00", "10"]
    total_index_full = ["1960", "1970", "1980", "1990", "2000", "2010"]

    temp_ds_train = []
    temp_ds_test = []
    temp_ds_validate = []

    for i in range(len(total_index)):

        dataset = pd.read_csv('data/dataset-of-{0}s.csv'.format(total_index[i]), header=0, delimiter=',')

        dataset_train = dataset.sample(frac=train_size, random_state=1, replace=True)
        dataset_test = dataset.drop(dataset_train.index)

        dataset_validate_sample = dataset_test.sample(frac=validate_size, random_state=1)
        dataset_validate = dataset_test.drop(dataset_validate_sample.index)

        dataset_train['decade'] = total_index_full[i]
        dataset_test['decade'] = total_index_full[i]
        dataset_validate['decade'] = total_index_full[i]

        temp_ds_train.append(dataset_train)
        temp_ds_test.append(dataset_test)
        temp_ds_validate.append(dataset_validate)

    dataset_train = pd.concat(temp_ds_train, axis=0, ignore_index=True)
    dataset_test = pd.concat(temp_ds_test, axis=0, ignore_index=True)
    dataset_validate = pd.concat(temp_ds_validate, axis=0, ignore_index=True)

    dataset_train.drop(cols_to_drop, axis=1, inplace=True)
    dataset_test.drop(cols_to_drop, axis=1, inplace=True)
    dataset_validate.drop(cols_to_drop, axis=1, inplace=True)

    dataset_train.to_csv('data/dataset_train_joined_flushed.csv')
    dataset_test.to_csv('data/dataset_test_joined_flushed.csv')
    dataset_validate.to_csv('data/dataset_validate_joined_flushed.csv')

    return dataset_train, dataset_test, dataset_validate
```

Figure 2.

get_dataset_seperated는 앞 함수에서 넘겨주었던 데이터셋들의 비율에 따라 전체 데이터셋을 나누고, 지정한 attribute들을 drop한다. 앞서 언급한 바와 같이 raw data가 10년(decade) 단위의 csv 파일로 묶여 있으므로 이를 고려해 데이터셋을 재구성해야 한다.

3-3. 실험 분석법과 로직

최선의 classification을 위해 다음과 같은 로직을 통해 분석을 완료하였다.

1) 선정된 5개의 모델에 대해 default parameter를 갖는 모델을 생성한다. 이를 2)와 3)에 대한 baseline으로 삼는다.
2) 5개의 모델에 대해 RFE를 적용한 feature selection 적용 모델을 만든다.
3) 5개의 모델에 대해 hyperparameter search를 통해 tuning한 모델을 만든다.
4) 1)번 모델에 대한 2)와 3)의 validation 데이터셋에서 cross-validation을 수행한다.
5) 한 모델 타입(e.g. 의사결정나무)의 1)~3) 모델 중 더 높은 average score를 기록한 모델을 저장한다.
6) 최종적으로 서로 다른 모델 사이의 average score를 비교하고 실험 전체에서 가장 높은 average score를 기록한 모델을 채택한다.

각 과정에 대한 상세 설명과 **Code documentation**은 다음과 같다.

```python
def train_model(x_train, y_train, x_test, y_test, model_name):
    sc = StandardScaler()
    x_train = sc.fit_transform(x_train)
    x_test = sc.transform(x_test)

    model = RandomForestClassifier()
    if model_name == "LogisticReg":
        model = LogisticRegression(random_state=42)

    elif model_name == "DT":
        model = DecisionTreeClassifier(random_state=42)

    elif model_name == "MLP":
        model = MLPClassifier(max_iter=2000, random_state=42)

    elif model_name == "NB":
        model = GaussianNB()

    elif model_name == "RF":
        model = RandomForestClassifier(random_state=42)

    elif model_name == "SVM":
        scaler = MinMaxScaler()
        scaler.fit(x_train)
        x_train = scaler.transform(x_train)

        model = LinearSVC(random_state=42)

    else:
        print("Wrong Model!")

    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    print("Estimator: ", model)

    return model, y_pred
```

Figure 3.

train_model 함수는 과정 1)에 대한 것으로 각 모델들은 사이킷런 default 값을 hyperparameter로 갖는다. 그 후 각 모델은 train 데이터셋에 fit하게 된다. 이 모델을 baseline으로 이후 각각의 모델들과 비교할 것이다.

```python
def train_rfe_model(x_train, y_train, x_test, y_test, model_name):
    sc = StandardScaler()
    x_train = sc.fit_transform(x_train)
    x_test = sc.transform(x_test)

    model = RandomForestClassifier()
    if model_name == "LogisticReg":
        model = RFE(LogisticRegression(random_state=42), n_features_to_select=10)

    elif model_name == "DT":
        model = RFE(DecisionTreeClassifier(random_state=42), n_features_to_select=10)

    elif model_name == "NB":
        model = GaussianNB()

    elif model_name == "RF":
        model = RFE(RandomForestClassifier(random_state=42), n_features_to_select=10)

    elif model_name == "SVM":
        scaler = MinMaxScaler()
        scaler.fit(x_train)
        x_train = scaler.transform(x_train)

        model = RFE(LinearSVC(random_state=42), n_features_to_select=10)

    else:
        print("Wrong Model!")

    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    print("Estimator: ", model)

    return model, y_pred
```

Figure 4.

train_rfe_model 함수는 과정 2에 대한 것으로 선정된 5개의 모델에 대해 feature selection 기법인 RFE를 적용하여 생성한 것으로 이때 n_feature_to_select는 10으로 설정하였다. 그 외에 train_model 함수와 동일하다. 추가적으로 NB에 대해서는 RFE를 적용하지 않았다.

```python
def hyperparameter_search(x_train, y_train, x_test, y_test, model_name, params=None):
    sc = StandardScaler()
    x_train = sc.fit_transform(x_train)
    x_test = sc.transform(x_test)

    opt = ""

    if model_name == "LogisticReg":
        opt = BayesSearchCV(
            LogisticRegression(),
            {
                'C': (1e-4, 2e+0),
                'tol': (1e-9, 1e-1)
            },
            n_iter=5,
            cv=5,
            #verbose=2,
            random_state=0
        )


    elif model_name == "DT":
        opt = BayesSearchCV(
            DecisionTreeClassifier(),
            {
                'max_depth': Integer(1, 20),
                'max_features': Categorical(['auto', 'sqrt', 'log2']),
                'min_samples_split': Integer(2, 20),
                'min_samples_leaf': Integer(1, 20),
                'min_impurity_decrease': (0.0, 1.0),
                'criterion': Categorical(['gini', 'entropy']),
            },
            n_iter=10,
            cv=5,
            #verbose=2,
            random_state=0
        )
```

Figure 5.

hyperparameter_search 함수는 과정 3에 대한 것으로 선정된 5개의 모델에 대해서 hyperparameter search를 수행하는 함수이다. 이때 hyperparameter search는 skopt의 BayesSearchCV 함수를 사용한다. 이 함수는 여러가지 benefit을 제공하는데, 첫째로 Grid search가 아닌 Bayesian optimization이므로 학습 이론에 의하면 더 좋은 파라미터를 찾을 확률이 올라간다. 둘째로 Cross-validation을 특별히 구현하지 않아도 된다는 점이다. 본 함수에서 cross-validation을 수행하면서 search 과정 중의 가장 optimal한 parameter를 찾을 수 있다. 따라서 BayesSearchCV 함수를 사용하였으며 각 parameter의 lower bound와 upper bound 등 range를 정해주었다.

```
    elif model_name == "NB":
        opt = BayesSearchCV(
            GaussianNB(),
            {
                "var_smoothing": (5e-10, 15e-10)
            },
            n_iter=10,
            cv=5,
            #verbose=2,
        )

    elif model_name == "RF":
        opt = BayesSearchCV(
            RandomForestClassifier(),
            {
                'n_estimators': Integer(1, 500),
                'max_depth': Integer(1, 20),
                'max_features': Categorical(['auto', 'sqrt', 'log2']),
                'min_samples_split': Integer(2, 20),
                'min_samples_leaf': Integer(1, 20),
                'bootstrap': Categorical([True, False]),
                'criterion': Categorical(['gini', 'entropy'])
            },
            n_iter=5,
            cv=5,
            #verbose=2,
            random_state=0
        )

    elif model_name == "SVM":
        scaler = MinMaxScaler()
        scaler.fit(x_train)
        x_train = scaler.transform(x_train)

        opt = BayesSearchCV(
            LinearSVC(),
            {
                'C': (1e-2, 10),
                'tol': (1e-9, 1e-1)
            },
            n_iter=32,
            cv=5,
            #verbose=2,
            random_state=0
        )
```

Figure 6. hyperparameter_search 함수

```
291     _ = opt.fit(x_train, y_train)
292     '''
293     print(opt.score(x_test, y_test))
294     print(opt.best_params_)
295     print(opt.best_score_)
296     print(opt.best_estimator_)
297     '''
298     print("Estimator: ", opt.best_estimator_)
299     print("Validation Avg Score of Best Model : ", opt.best_score_)
300     return opt.best_score_, opt.best_params_, opt.best_estimator_
```

Figure 7. hyperparameter_search 함수

```
for model_type in model_types:
    print("< Analysis of Model Using Default Parameters >\n")
    model, y_pred = train_model(x_train=x_train, y_train=y_train, x_test=x_test, y_test=y_test, model_name=model_type)
    score1 = validate_model(x_train=x_train, y_train=y_train, model=model)

    print("\n=========================================\n")
    print("< Analysis of Model Using Feature Selection >\n")
    model_rfe, y_pred_rfe = train_rfe_model(x_train=x_train, y_train=y_train, x_test=x_test, y_test=y_test, model_name=model_type)
    score2= validate_model(x_train=x_train, y_train=y_train, model=model_rfe)

    #accuracy, precision, recall, f1 = evaluate_model(x_train, x_test, y_train, y_test, y_pred, model, cls)

    print("\n=========================================\n")
    print("< Analysis of Model After Parameter Search >\n")
    score3, params, parambest_model = hyperparameter_search(x_train=x_train, y_train=y_train, x_test=x_test, y_test=y_test, model_name=model_type)

    print("=========================================")

    models = [model, model_rfe, parambest_model]
    scores = [score1, score2, score3]
    goodmodel = models[scores.index(max(scores))]
    print("Best Model using(" + model_type + "): ", goodmodel)
    print("=========================================\n\n")

    goodmodels.append(goodmodel)
    goodscores.append(max(scores))
```

Figure 8.

Figure 8은 과정 4와 5에 대한 것이다. 각 모델에 대해서 과정 1과 2의 모델에 대해서 Cross-validation을 수행한 뒤 hyperparameter_search에서 이미 cross-validation을 수행한 과정 3의 모델과 비교한다. 이 때 변수 goodmodel은 과정 1~3 중 가장 좋은 스코어를 기록한 모델이 선택되고 마지막으로 goodmodels 라는 리스트에 추가되어서 다른 타입의 모델들과 비교될 것이다.

```
idx = goodscores.index(max(goodscores))
bestmodel = goodmodels[idx]
print("Best Model of My Project: ", bestmodel)

cls = bestmodel.classes_
accuracy, precision, recall, f1 = evaluate_model(x_train, x_test, y_train, y_test, bestmodel.predict(x_test), bestmodel, cls)
```

Figure 9.

마지막으로 Figure 9.는 과정 6에 대한 것으로 이를 통해 최종 모델을 선택하게 된다.

4. 실험 결과 분석

실험결과는 다음과 같다. 먼저 각 모델에 대한 validation을 살펴보겠다.

4-1. Cross-Validation 검증

1) Logistic Regression

```
< Analysis of Model Using Default Parameters >

Estimator:  LogisticRegression(random_state=42)
Validation Score:  [0.67416755 0.69849475 0.7550555  0.78607268 0.75794435]
Validation Avg Score:  0.7343469667021438

========================================

< Analysis of Model Using Feature Selection >

Estimator:  RFE(estimator=LogisticRegression(random_state=42), n_features_to_select=10)
Validation Score:  [0.67386346 0.69849475 0.75262278 0.78424814 0.75855253]
Validation Avg Score:  0.7335563326744716

========================================

< Analysis of Model After Parameter Search >

Estimator:  LogisticRegression(C=0.516817486779299, tol=0.09789036586373251)
Validation Avg Score of Best Model :  0.7344990117074655
========================================
Best Model using(LogisticReg):  LogisticRegression(C=0.516817486779299, tol=0.09789036586373251)
========================================
```

Figure 10.

Figure 10은 로지스틱 회귀의 결과이며 과정 1~3의 모델 중 hyperparameter를 튜닝한 과정 3의 모델이 Cross-validation에서 가장 높은 score를 기록하였다.

2) Decision Tree

```
< Analysis of Model Using Default Parameters >

Estimator:  DecisionTreeClassifier(random_state=42)
Validation Score:  [0.66063555 0.6513608  0.70366428 0.79747605 0.75566368]
Validation Avg Score:  0.7137600729816025

========================================

< Analysis of Model Using Feature Selection >

Estimator:  RFE(estimator=DecisionTreeClassifier(random_state=42), n_features_to_select=10)
Validation Score:  [0.63326745 0.66246809 0.70548882 0.78105519 0.73954691]
Validation Avg Score:  0.7043636916527293

========================================

< Analysis of Model After Parameter Search >

Estimator:  DecisionTreeClassifier(max_depth=11, max_features='log2',
                       min_impurity_decrease=0.02729496131386156,
                       min_samples_leaf=2, min_samples_split=6)
Validation Avg Score of Best Model :  0.584461000456135
========================================
Best Model using(DT):  DecisionTreeClassifier(random_state=42)
========================================
```

Figure 11.

Figure 11은 의사결정나무의 결과이며 과정 1~3의 모델 중 default parameter를 갖는 과정 1의 모델이 Cross-validation에서 가장 높은 score를 기록하였다.

3) Naïve Bayesian

```
< Analysis of Model Using Default Parameters >

Estimator:  GaussianNB()
Validation Score:  [0.68085753 0.69895089 0.69682226 0.72890376 0.69636612]
Validation Avg Score:  0.7003801125133039

========================================

< Analysis of Model Using Feature Selection >

Estimator:  GaussianNB()
Validation Score:  [0.68085753 0.69895089 0.69682226 0.72890376 0.69636612]
Validation Avg Score:  0.7003801125133039

========================================

< Analysis of Model After Parameter Search >

Estimator:  GaussianNB(var_smoothing=1.3979181910270596e-09)
Validation Avg Score of Best Model :  0.7003801125133039
========================================
Best Model using(NB):  GaussianNB()
========================================
```

Figure 13.

Figure 13은 가우시안 나이브 베이지안 모델의 결과이며 과정 1~3 중 과정 1의 모델이 Cross-validation에서 가장 높은 score를 기록하였다.

4) Support Vector Machine

```
Validation Score:  [0.67523187 0.6969743  0.75079824 0.78105519 0.75155846]
Validation Avg Score:  0.7311236125893263

========================================
```

```
Validation Score:  [0.67477573 0.69819066 0.75155846 0.78181542 0.75125437]
Validation Avg Score:  0.7315189296031626

========================================
```

```
Estimator:  LinearSVC(C=9.671640936594155, tol=0.07565529084765206)
Validation Avg Score of Best Model :  0.7321271096244488
========================================
Best Model using(SVM):  LinearSVC(C=9.671640936594155, tol=0.07565529084765206)
========================================
```

Figure 14.

Figure 14는 SVM 모델의 결과이며 과정 1~3 중 과정 3의 모델이 Cross-validation에서 가장 높은 score를 기록하였다.

5) Random Forest

```
< Analysis of Model Using Default Parameters >

Estimator:  RandomForestClassifier(random_state=42)
Validation Score:  [0.74638893 0.73909077 0.80720693 0.87136993 0.83031777]
Validation Avg Score:  0.7988748669606203

========================================

< Analysis of Model Using Feature Selection >

Estimator:  RFE(estimator=RandomForestClassifier(random_state=42), n_features_to_select=10)
Validation Score:  [0.66291622 0.72662308 0.7868329  0.85282043 0.81861031]
Validation Avg Score:  0.7695605899346205

========================================

< Analysis of Model After Parameter Search >

Estimator:  RandomForestClassifier(bootstrap=False, max_depth=18, max_features='log2',
                       min_samples_leaf=6, min_samples_split=5,
                       n_estimators=387)
Validation Avg Score of Best Model :  0.7923369317317925
========================================
Best Model using(RF):  RandomForestClassifier(random_state=42)
========================================
```

Figure 15.

Figure 15는 랜덤포레스트 모델의 결과이며 과정 1~3 중 과정 1의 모델이 Cross-validaton에서 가장 높은 score를 기록하였다.

4- 2. Test set 검증

```
Best Model of My Project:  RandomForestClassifier(random_state=42)
Accuracy(Test): 80.00432455808422
Precision: 77.3197909476383
Recall: 84.85930735930735
F1 score: 0.8091429750786853
Training MSE:  0.00012163600425726015
Test model MSE 0.1999567544191578
```

Figure 16.

Figure 16은 최종적으로 선택된 최적의 모델로 test set을 통해 평가하였다. 본 모델은 랜덤 포레스트의 과정 1 모델로, 사이킷런의 default parameter를 갖는다. 최종 모델은 test set에서 약 80%의 accuracy를 기록하였으며 이때의 precision은 77.3, recall은 84.85, F1 스코어는 0.8091을 기록하였다.

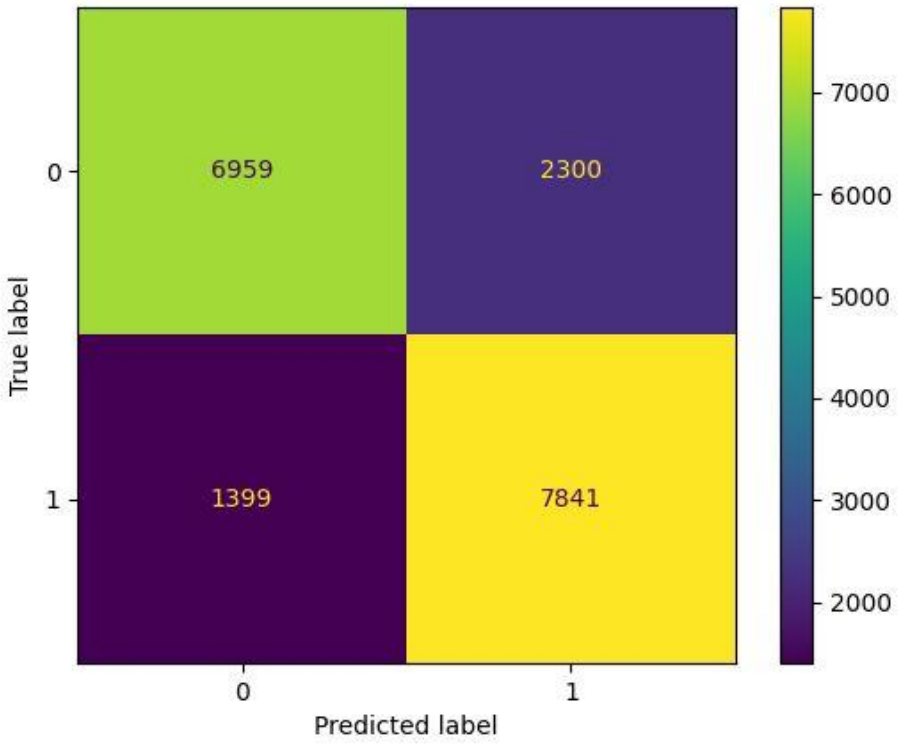

Figure 17. 최종 모델의 Confusion Matrix

이 모델은 validation set에서도 79.23의 accuracy를 기록했으며 test에서도 이와 유사하게 나왔음을 확인할 수 있다. 약 3만 9천 개에 달하는 전체 데이터셋 크기와 그것의 20%인 테스트셋을 고려했을 때 테스트셋 역시 치우쳐 있지 않았고 모델 역시 test set과 유사하며 적절한(편향되지 않은) train set에 대해 학습했기 때문에 다음과 같은 정확도를 기록할 수 있다고 분석할 수 있다.

4-3. 앙상블 방법과 성능 비교

본 실험에서는 앙상블 방법이 사용된 랜덤 포레스트가 최종 모델로 선택되었으며 validation set과 test set에서 가장 높은 정확도(약 79~80%)를 보였다. 이는 앙상블 모델이 voting 방식 등을 통해 단일 모델이 대응할 수 없는 오버피팅 문제에 유연하게 대응할 수 있기 때문이다.

4-4. Feature Selection 기법과의 비교

Feature selection은 과정 2에서 적용되었다. 그러나 대부분의 모델에서 feature selection 기법을 적용한 모델들이 높은 성능을 보이지 못했다. (feature selection을 적용한 기법 중 상위 5개의 모델에 든 모델이 없었다.) 이유는 본 실험의 attribute가 상대적으로 다른 실험에 비해 적을 뿐만 아니라 실험에 사용된 attribute들이 Spotify 기업의 다양한 전문가들에 의해 면밀히 분석된 attribute이기 때문에 각각의 feature가 곡의 히트에 끼치는 영향이 크고 중요했다는 것을 짐작해볼 수 있다.

5. 결론

지금까지 Kaggle의 Spotify Dataset을 이용해 히트곡 classification을 수행해보았다. 최종적으로 약 80%의 확률로 히트곡을 맞추는 분류기를 만들 수 있었다. 특히 5가지의 모델 중 앙상블 기법을 적용한 랜덤 포레스트가 가장 성능이 뛰어났

다. 그러나 몇 가지 후속 연구가 남아있다. 전처리 과정에서 drop한 artist attribute 역시 영향은 미미할 수 있지만 가수의 이름, 즉 명성에 따라서도 히트곡이 될 수도 있기 때문에 이 feature의 영향력에 대한 추가적인 실험이 필요하다. 추가적으로, 랜덤 포레스트 이외의 앙상블 기법을 이용해서 더 높은 정확도를 보이는 분류기를 만들 수도 있을 것이다.