Yeast Dataset

<모델 개요>

- "Yeast Dataset"은 효모 유전자의 **microarray expression data**와 **phylogenetic profiles**를 포함한 **multi-label classification**용 데이터셋.

- 각 인스턴스(유전자)가 수행하는 여러 생물학적 기능을 예측하는 것이 모델의 목표.

<Dataset 특징>

1. Instance: 하나의 유전자(gene)을 나타냄.

2. Features: microarray expression data와 phylogenetic profiles로 총 103개가 구성됨.

- microarray expression data: 유전자 발현 수준을 나타내는 numerical 데이터

- phylogenetic profiles: 여러 생물종에서 해당 유전자의 존재 유무를 나타내는 symbolic 데이터

3. Labels: 하나의 유전자에 대해 14개의 label(multi-label)로 할당됨.

- 각 label은 생물학적 기능을 의미하며, binary value(0 또는 1)를 가짐.

- 하나의 gene은 여러 개의 기능을 가질 수 있음. Ex.[1, 0, 0, 1, ... ,0]

- physiological process, molecular function, cellular component 등으로 14개의 label이 구성됨.

이를 통해 gene function prediction과 생물학적 데이터 분석에서 주로 활용됨.

유전자가 갖는 여러 생물학적 기능을 예측하는 모델의 개발 절차

.1. 데이터 분석 및 탐색(EDA)

- 데이터의 분포, 상관관계 등을 살펴보며 각 feature가 target에 어떤 영향을 미치는지 파악함.
- OpenML에서 data를 로드해오기 위해 fetch_openml로 Yeast Dataset을 가져와 데이터 구조를 확인함.

<주어진 dataset의 결측치 여부, 데이터 타입, 기초 통계 요약 확인>

```python
# 사이킷런 0.22 버전 이전의 경우 fetch_mldata
from sklearn.datasets import fetch_openml
yeast = fetch_openml('yeast', version=4, as_frame=True)
# as_frame=True -> pandas DataFrame 형태로 로드
X = yeast['data']
Y = yeast['target']

# 데이터 구조 확인하기
print("Feature Shape:", X.shape)
print("Label Shape:", Y.shape)

print(X.head())
print(Y.head())

print(X.info())
print(Y.info())

print(X.describe())
print(Y.describe())
```

```
Feature Shape: (2417, 103)
Label Shape: (2417, 14)
      Att1      Att2      Att3      Att4      Att5      Att6      Att7  \
0  0.004168 -0.170975 -0.156748 -0.142151  0.058781  0.026851  0.197719
1 -0.103956  0.011879 -0.098986 -0.054501 -0.007970  0.049113 -0.030580
2  0.509949  0.401709  0.293799  0.087714  0.011686 -0.006411 -0.006255
3  0.119092  0.004412 -0.002262  0.072254  0.044512 -0.051467  0.074686
4  0.042037  0.007054 -0.069483  0.081015 -0.048207  0.089446 -0.004947

      Att8      Att9     Att10  ...     Att94     Att95     Att96     Att97  \
0  0.041850  0.066938 -0.056617  ...  0.006166 -0.012976 -0.014259 -0.015024
1 -0.077933 -0.080529 -0.016267  ...  0.007680  0.027719 -0.085811  0.111123
2  0.013646 -0.040666 -0.024447  ...  0.096277 -0.044932 -0.089470 -0.009162
3 -0.007670  0.079438  0.062184  ... -0.083809  0.200354 -0.075716  0.196605
4  0.064456 -0.133387  0.068878  ... -0.060467  0.044351 -0.057209  0.028047

      Att98     Att99    Att100    Att101    Att102    Att103
0 -0.010747  0.000411 -0.032056 -0.018312  0.030126  0.124722
1  0.050541  0.027565 -0.063569 -0.041471 -0.079758  0.017161
2 -0.012010  0.308378 -0.028053  0.026710 -0.066565 -0.122352
3  0.152758 -0.028484 -0.074207 -0.089227 -0.049913 -0.043893
4  0.029661 -0.050026  0.023248 -0.061539 -0.035160  0.067834

[5 rows x 103 columns]
  Class1 Class2 Class3 Class4 Class5 Class6 Class7 Class8 Class9 Class10  \
0  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE  FALSE   FALSE
1  FALSE  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE   FALSE
2  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   FALSE
3  FALSE  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE   FALSE
4  FALSE  FALSE   TRUE   TRUE   TRUE   TRUE  FALSE  FALSE  FALSE   FALSE

  Class11 Class12 Class13 Class14
0   FALSE    TRUE    TRUE   FALSE
1   FALSE   FALSE   FALSE   FALSE
2   FALSE    TRUE    TRUE   FALSE
3   FALSE   FALSE   FALSE   FALSE
4   FALSE   FALSE   FALSE   FALSE
```

⇨ Dataset을 pd dataframe 형태로 로드함. X.shape()와 Y.shape()를 통해 전체 entity가 2417, feature 103개, label 14개인 것을 파악함.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2417 entries, 0 to 2416
Columns: 103 entries, Att1 to Att103
dtypes: float64(103)
memory usage: 1.9 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2417 entries, 0 to 2416
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Class1   2417 non-null   category
 1   Class2   2417 non-null   category
 2   Class3   2417 non-null   category
 3   Class4   2417 non-null   category
 4   Class5   2417 non-null   category
 5   Class6   2417 non-null   category
 6   Class7   2417 non-null   category
 7   Class8   2417 non-null   category
 8   Class9   2417 non-null   category
 9   Class10  2417 non-null   category
 10  Class11  2417 non-null   category
 11  Class12  2417 non-null   category
 12  Class13  2417 non-null   category
 13  Class14  2417 non-null   category
dtypes: category(14)
memory usage: 34.9 KB
```

```
              Att94         Att95         Att96         Att97         Att98  \
count  2417.000000  2417.000000  2417.000000  2417.000000  2417.000000
mean     -0.000773     0.000464    -0.000515     0.000667     0.000324
std       0.093316     0.096684     0.096209     0.096635     0.096280
min      -0.455191    -0.283594    -0.279408    -0.226420    -0.225374
25%      -0.054133    -0.056415    -0.056414    -0.059382    -0.058025
50%      -0.012893    -0.023595    -0.024313    -0.023059    -0.021942
75%       0.027977     0.034937     0.036057     0.041430     0.035730
max       0.609175     0.542867     0.547134     0.385928     0.540493

              Att99        Att100        Att101        Att102        Att103
count  2417.000000  2417.000000  2417.000000  2417.000000  2417.000000
mean     -0.001483    -0.001047    -0.001539     0.000284     0.007605
std       0.094369     0.096900     0.094211     0.093154     0.099368
min      -0.501572    -0.236589    -0.267052    -0.194079    -0.237752
25%      -0.053591    -0.063318    -0.059542    -0.054078    -0.077191
50%      -0.018216    -0.033623    -0.023519    -0.012007     0.022126
75%       0.019583     0.038901     0.025408     0.028087     0.103185
max       0.569250     0.509963     0.587358     0.700340     0.163431

[8 rows x 103 columns]
       Class1 Class2 Class3 Class4 Class5 Class6 Class7 Class8 Class9 Class10  \
count    2417   2417   2417   2417   2417   2417   2417   2417   2417    2417
unique      2      2      2      2      2      2      2      2      2       2
top     FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   FALSE
freq     1655   1379   1434   1555   1695   1820   1989   1937   2239    2164

       Class11 Class12 Class13 Class14
count     2417    2417    2417    2417
unique       2       2       2       2
top      FALSE    TRUE    TRUE   FALSE
freq      2128    1816    1799    2383
```

⇨ X.info(), Y.info()로 feature와 label의 data type, 결속치가 없는 것을 확인함. X.describe(), Y.describe()로 feature와 label을 이루는 전반적인 data의 분포에 대한 이해를 도움.

.2. 데이터 전처리 및 Train/Test Split

1. Feature Scaling

- Numerical 데이터: 정규화(Standardization) 필요.

- Symbolic 데이터: One-Hot Encoding 또는 Label Encoding으로 처리.

1) One-Hot Encoding

- 고유한 symbolic 값을 새로운 이진 벡터로 변환.

- 값이 범주에 해당하면 1, 그렇지 않으면 0.

장점: 범주 순서가 없는 데이터에 적합

단점: feature 수가 많으면 급격히 차원 증가(차원의 저주)

2) Label Encoding

- 고유한 symbolic 값을 정수로 변환.

- ex. ["A", "B", "C"] -> [0,1,2]

장점: feature 수가 많아도 차원을 증가시키지 않아 급격한 데이터 크기 증가X.

단점: 범주 순서 존재할 시 적절하지 않음. (순서, 크기 관계를 잘못 학습할 가능성 존재)

그러나 우리는 openml로 data를 로드했기 때문에 Y는 dataframe 형태로 저장된 상태임. 따라서 Y.applymap()을 이용해서 T/F를 이진벡터 0/1로 변환함.

```
# dataframe 형태의 label(FALSE/TRUE)를 이진벡터(0/1)로 변환
Y_binary = Y.applymap(lambda x: 1 if x == "TRUE" else 0).values
import numpy as np
Y_binary = np.array(Y_binary, dtype=int)
```

```
<ipython-input-3-ee8769c9314c>:2: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  Y_binary = Y.applymap(lambda x: 1 if x == "TRUE" else 0).values
```

- Y.applymap():

DataFrame의 각 원소에 함수(lambda)를 적용하는 함수.

"TRUE"는 1, "FALSE"는 0으로 변환.

- values:

변환된 DataFrame을 NumPy 배열로 추출.(모델학습을 위하여)

```
print(Y_binary)
```

```
[[0 0 0 ... 1 1 0]
 [0 0 1 ... 0 0 0]
 [0 1 1 ... 1 1 0]
 ...
 [0 0 0 ... 1 1 0]
 [0 0 0 ... 1 1 0]
 [0 1 1 ... 1 1 0]]
```

⇨ numpy 배열 형식으로 변환된 Y_binary 확인 가능.

```
[5]  # Numerical 데이터 정규화
     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
```

=> Numerical 데이터 regularization

2. Train/Test Split: 일정 비율로 랜덤하게 분할함.

```
# Train/Test Split as 0.8:0.2
# with random seed=42
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, Y_binary,
                                                    test_size=0.2, random_state=42)

print("Training Data Shape:", X_train.shape, y_train.shape)
print("Test Data Shape:", X_test.shape, y_test.shape)
```

```
Training Data Shape: (1933, 103) (1933, 14)
Test Data Shape: (484, 103) (484, 14)
```

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("Unique values in y_train:", np.unique(y_train))
```

```
X_train shape: (1933, 103)
y_train shape: (1933, 14)
Unique values in y_train: [0 1]
```

⇨ random 변수로 고정적 trainset, testset 분리.

⇨ Label value로 0,1 변환이 잘된 것을 확인가능.

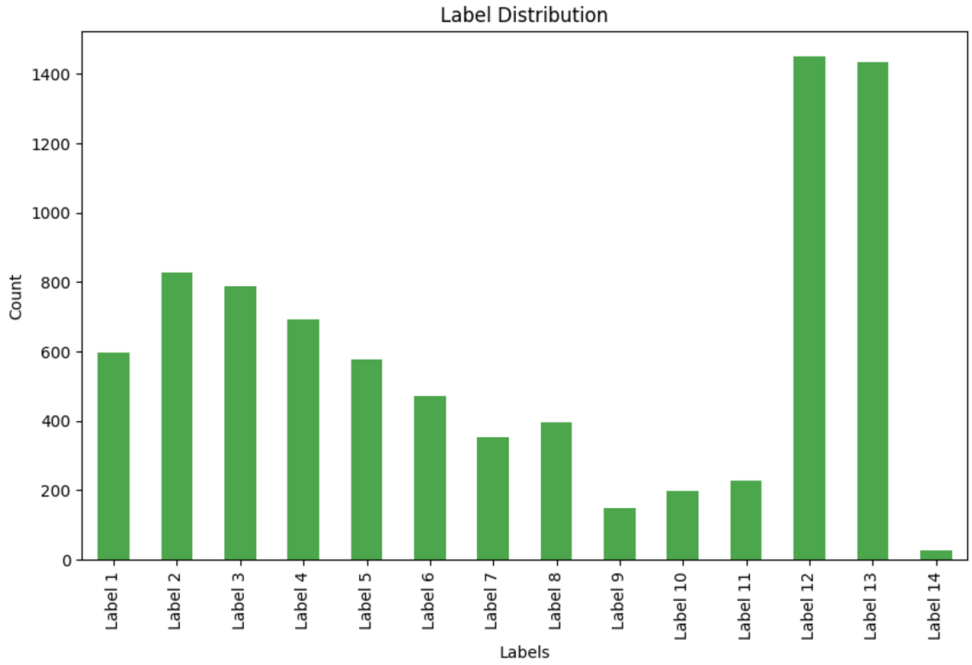
⇨ Feature가 14개로 늘어난 이유

: OneHotEncoder로 symbolic 데이터를 이진 벡터로 변환하고 각각 새로운 열로

확장하기 때문에 입력 데이터의 차원(= feature 개수)이 증가함.

각 라벨의 빈도수를 막대그래프 시각화로 파악

=> 클래스 불균형 감지 가능



Label 12와 Label 13의 경우 다른 라벨들에 비해 압도적으로 많은 빈도,

반면 Label 14는 거의 데이터가 없는 상황.

=> 빈도가 높은 라벨에 편향될 가능성이 크며, 빈도낮은 라벨 학습이 제대로 이루어지지 않을 확률이 존재함.

*** 불균형 데이터를 다루는 방법**

1. 데이터 증강

Oversampling (과샘플링): 빈도 낮은 라벨의 데이터를 증강.

* SMOTE (Synthetic Minority Oversampling Technique)

: 기존 데이터 특성 보존, 소수 클래스의 샘플 생성.

* Random Oversampling

: 소수 클래스 데이터를 단순 복제 -> 데이터 양 증가.

Undersampling (소샘플링)

: 빈도 높은 라벨의 데이터 일부 제거. (데이터 손실 유의 필요)

==========

2. 가중치 조정

- 클래스 가중치를 부여 -> 빈도 낮은 라벨일수록 더 큰 가중치 부여.

"class_weight='balanced'" -> 불균형 자동 보정

- Logistic Regression, SVM, Random Forest 등의 모델에서 지원

==========

3. 라벨 분포 조정

- 각 데이터 샘플의 라벨 수 조정.

- 너무 많은 라벨을 가진 샘플 제거

- 새로운 데이터 샘플 생성

==========

4. 알고리즘 선택

불균형 데이터에 특화된 모델 or 학습 전략 사용

XGBoost, CatBoost, LightGBM => 불균형 데이터에서도 좋은 성능

==========

⇨ Baseline 모델 설정 및 학습 시, class_weight 파라미터를 이용해서 불균형 데이터에 대한 균형적인 가중치 할당 방식을 선택함.

3. 모델 선택 및 학습

Baseline 모델

프로젝트에서 가장 기본적인 기준 성능이 되는 모델

복잡한 알고리즘 도입 전 단순한 방법으로 성능 측정

-> 향후 모델 개선 효과를 평가하는 기준점이 됨.

Baseline 종류

Random Forest: 앙상블 학습 기법(여러 개의 decision tree를 병렬 학습)

분류, 회귀, 다중 클래스/라벨 모두 효과적

over-fitting 발생가능성이 낮음 (트리 개별 성능에 의존X, 평균화)

==========

Logistic Regression: 선형 모델. 이진 분류 문제 해결을 위해 설계됨.

확률 기반으로 클래스 예측을 수행.

모델이 단순해 데이터 분석 or 기본적인 성능 기준을 세우기 적합.

다중 라벨 문제에는 확장 기법(OneVsRestClassifier)과 함께 사용하며

라벨별 '선형 결정 경계'를 학습함.

requlrization을 통해 과적합 방지 -> penalty 매개변수와 L1, L2 규제

Multi-Label Classification

: 하나의 샘플에 여러 개의 라벨이 할당될 수 있는 classification

다중 라벨: 한 샘플에 여러 클래스 할당 가능

<=> 다중 클래스: 한 샘플에 하나의 클래스만 할당함.

==========

OneVsRestClassifier(OvR)

- 각 라벨이 독립적인 이진 분류기로 학습됨.
- 다중 클래스/라벨을 여러 개의 이진 분류 문제로 전환
- 반드시 이진 분류 모델을 기반으로 학습함.
- 단점: 라벨 간 상관관계 고려가 불가함.
- 벙렬 처리 기본지원 X.
- 다중 클래스/라벨 제 해결 목적

==========

MultiOutputClassifier

- 각 라벨이 독립적으로 학습됨.

- 다중 라벨을 개별 분류기로 전환, 병렬적으로 학습함.

- 이진 분류/다중클래스 모델 모두 사용 가능함.

- 벙렬 처리 가능함 -> 라벨이 많은 경우 유리하게 적용.

- 단점: 라벨 간 상관관계 고려가 불가함.

- 다중 라벨 문제 해결 목적

고급 모델 및 튜닝

성능 지표 분석

: Precision, Recall, F1-Score: 각 라벨별/전체 성능 확인 가능.

ROC Curve 및 AUC Score: 각 클래스에 대한 구분 능력 평가.

- ROC: 라벨별 이진 분류 성능의 평가 지표

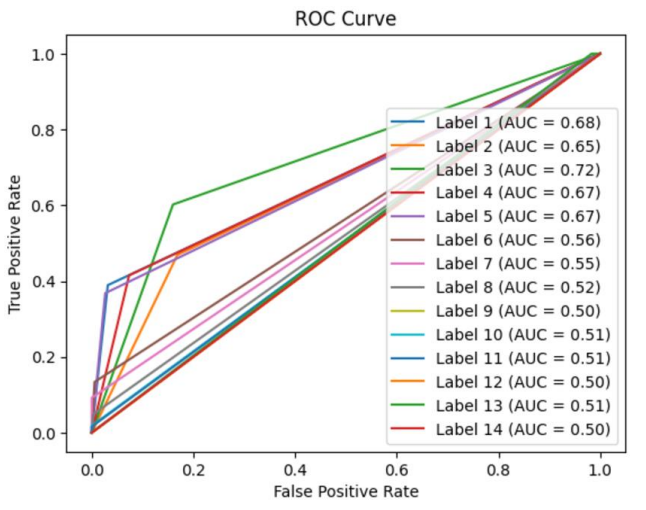
- X축 = False-positive: False를 True로 잘못 판단

- Y축 = True-positive: True를 올바르게 True로 판단

[1] OneVsRest로 RandomForest를 확장한 모델(class_weight 설정X)

```python
# OneVsRest로 R.F를 확장한 모델

from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import hamming_loss, accuracy_score

# 모델 정의하기
# BaseLine model로 Random forest를 선택.
# OneVsRest로 R.F를 확장하여 멀티 라벨 분류를 다룸.
model = OneVsRestClassifier(RandomForestClassifier(random_state=42))

# 학습 데이터셋으로 fitting
# OneVsRest -> 각 레벨에 대해 독립적으로 RF 학습 진행.
model.fit(X_train, y_train)

# fit한 model과 test data로 예측값 구하기
# 마찬가지로 label별 독립적인 결과 생성
y_pred = model.predict(X_test)

# 성능평가
print("Hamming Loss:", hamming_loss(y_test, y_pred))
# Hamming Loss: 샘플 내 라벨별 오답 비율
print("Accuracy:", accuracy_score(y_test, y_pred))
# Accuracy: 샘플별 예측 성공 비율
```

```
Hamming Loss: 0.18668831168831168
Accuracy: 0.1797520661157025
```



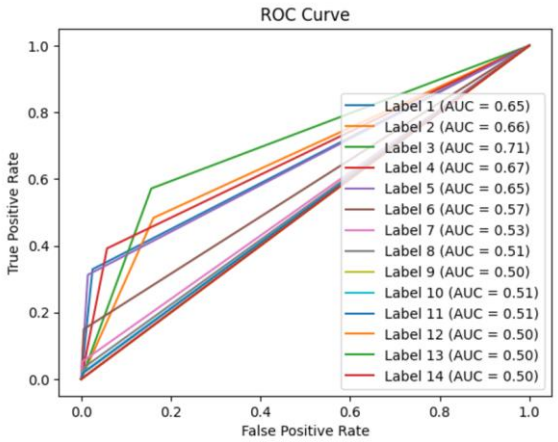
⇨ **0.5: 랜덤하게 예측하는 경우와 거의 동일한 정확도(예측정도)**

⇨ **그런 점에서 아직 Label5까지만 만족. 0.7을 넘기는 Label이 매우 적게 존재.**

[2] OneVsRest로 R.F를 확장한 모델(class_weight='balanced')

클래스 불균형 완화 - 클래스 가중치를 사용하는 Random Forest

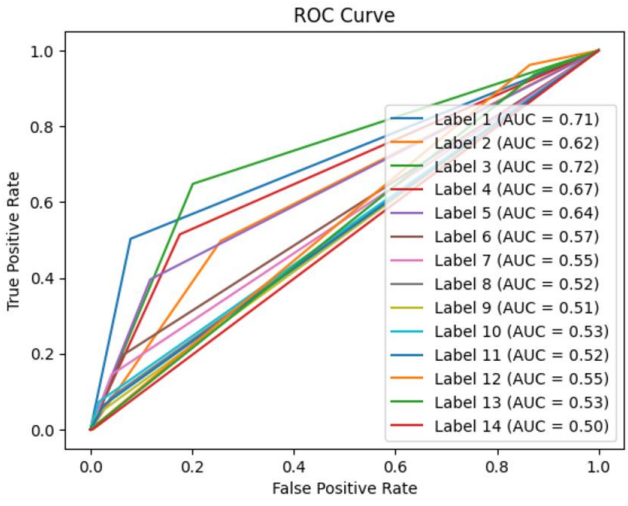
```
[41]  1 # OneVsRest로 R.F를 확장한 모델
      2
      3 from sklearn.ensemble import RandomForestClassifier
      4 from sklearn.multiclass import OneVsRestClassifier
      5 from sklearn.metrics import hamming_loss, accuracy_score
      6
      7 # 모델 정의하기
      8 # BaseLine model로 Random forest를 선택.
      9 # OneVsRest로 R.F를 확장하여 멀티 라벨 분류를 다룸.
     10 model = OneVsRestClassifier(RandomForestClassifier(class_weight="balanced", random_state=42))
     11
     12 # 학습 데이터셋으로 fitting
     13 # OneVsRest -> 각 레벨에 대해 독립적으로 RF 학습 진행.
     14 model.fit(X_train, y_train)
     15
     16 # fit한 model과 test data로 예측값 구하기
     17 # 마찬가지로 label별 독립적인 결과 생성
     18 y_pred = model.predict(X_test)
     19
     20 # 성능평가
     21 print("Hamming Loss:", hamming_loss(y_test, y_pred))
     22 # Hamming Loss: 샘플 내 라벨별 오답 비율
     23 print("Accuracy:", accuracy_score(y_test, y_pred))
     24 # Accuracy: 샘플별 예측 성공 비율

Hamming Loss: 0.18963990554899646
Accuracy: 0.1590909090909091
```



⇨ **여전히 0.7 넘기는 Label이 매우 적게 존재. AUC 증가한 Label과 감소한 Label 모두 공존(종합적인 성능의 변화 미미함)**

[3] OneVsRest로 logistic Regression를 확장한 모델(class_weight 설정X)

```
 1 # OneVsRest로 logistic Regression을 확장한 모델
 2
 3 from sklearn.linear_model import LogisticRegression
 4 from sklearn.multiclass import OneVsRestClassifier
 5 from sklearn.metrics import hamming_loss, accuracy_score
 6
 7 # 모델 정의하기
 8 # Logistic Regression 모델 정의 = BaseLine
 9 log_reg = LogisticRegression(solver='lbfgs', max_iter=1000, random_state=42)
10 # max_iter=최대 학습반복 횟수, random_state=재현을 위한 난수 설정
11 # OneVsRest로 logistic을 확장하여 멀티 라벨 분류를 다룸.
12 model = OneVsRestClassifier(log_reg)
13
14 # 학습 데이터셋으로 fitting
15 # OneVsRest -> 각 레벨에 대해 독립적으로 RF 학습 진행.
16 model.fit(X_train, y_train)
17
18 # fit한 model과 test data로 예측값 구하기
19 # 마찬가지로 label별 독립적인 결과 생성
20 y_pred = model.predict(X_test)
21
22 # 성능평가
23 print("Hamming Loss:", hamming_loss(y_test, y_pred))
24 # Hamming Loss: 샘플 내 라벨별 오답 비율
25 print("Accuracy:", accuracy_score(y_test, y_pred))
26 # Accuracy: 샘플별 예측 성공 비율

Hamming Loss: 0.20454545454545456
Accuracy: 0.15702479338842976
```

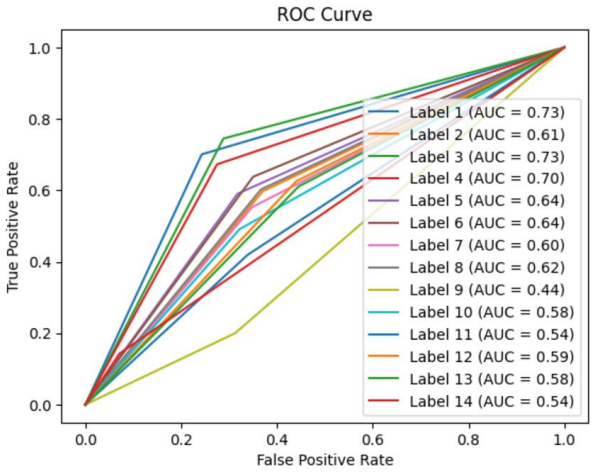


⇨ **AUC 0.7 넘기는 Label이 증가함. 전반적인 성능이 조금씩 상승함.**

[4] OneVsRest로 logistic Regression를 확장한 모델(class_weight='balanced')

```python
# OneVsRest로 logistic Regression을 확장한 모델 - balanced 가중치 사용

from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import hamming_loss, accuracy_score

# 모델 정의하기
# Logistic Regression 모델 정의 = BaseLine
log_reg = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)
# max_iter=최대 학습반복 횟수, random_state=재현을 위한 난수 설정
# OneVsRest로 logistic을 확장하여 멀티 라벨 분류를 다룸.
model = OneVsRestClassifier(log_reg)

# 학습 데이터셋으로 fitting
# OneVsRest -> 각 레벨에 대해 독립적으로 RF 학습 진행.
model.fit(X_train, y_train)

# fit한 model과 test data로 예측값 구하기
# 마찬가지로 label별 독립적인 결과 생성
y_pred = model.predict(X_test)

# 성능평가
print("Hamming Loss:", hamming_loss(y_test, y_pred))
# Hamming Loss: 샘플 내 라벨별 오답 비율
print("Accuracy:", accuracy_score(y_test, y_pred))
# Accuracy: 샘플별 예측 성공 비율
```

```
Hamming Loss: 0.32718417945690675
Accuracy: 0.0640495867768595
```

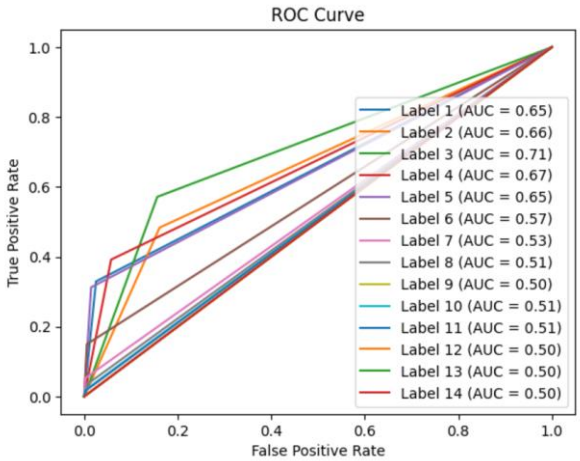


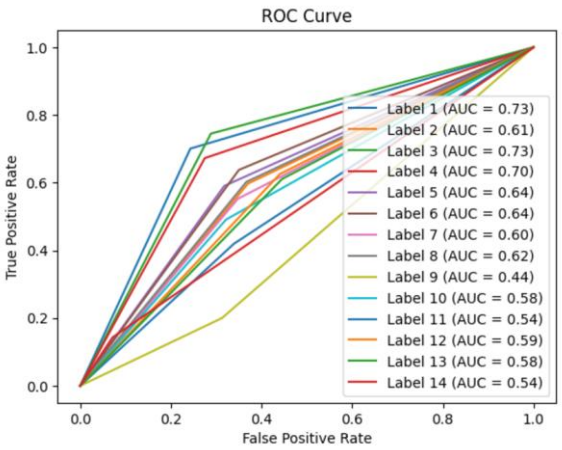
⇨ **AUC 0.7 넘기는 Label이 또 증가함. 전반적인 성능이 지속적으로 상승함.**

⇨ **[1]~[4]의 model fit을 통해 class_weight="balanced"로 class 분포별 가중치 완화 파라미터를 활용하는 것이 상대적으로 효과적인 모델 성능 구현에 영향을 줌.**

[5] MultiOutputClassifier로 RandomForest를 확장한 모델(class_weight='balanced')

[6] MultiOutputClassifier로 logistic Regression를 확장한 모델(class_weight='balanced')





=> [**5**]

=> [6] : similar with [4]

지금까지 가장 성능이 우수한 모델 조합: class_weight="balanced" with

- OneVsRest로 logistic Regression을 확장한 모델
- MultiOutputClassifier로 logistic Regression을 확장한 모델

불균형 분포 데이터셋에 효과적인 모델: Gradient Boosting 모델:

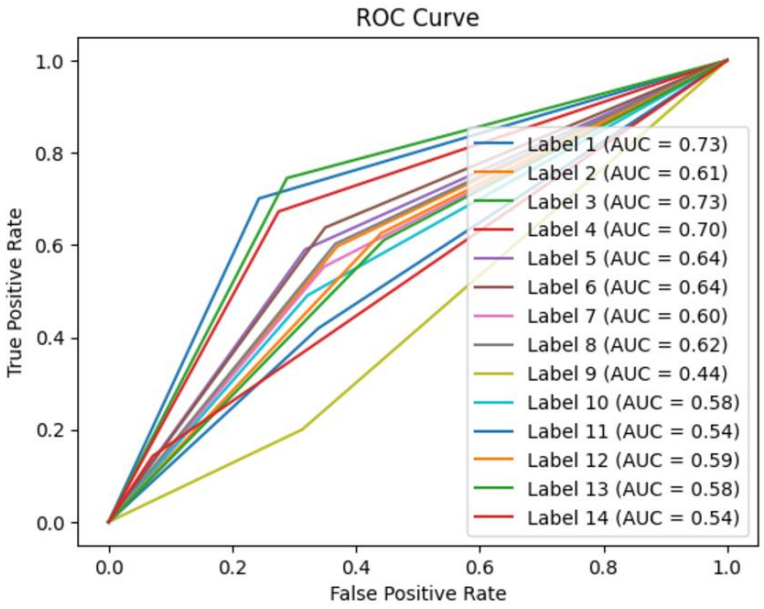
- XGBoost, LightGBM 알고리즘

- 트리 기반 모델(스케일링 불필요)의 확장으로 학습 속도와 정확도가 뛰어남.

```
from xgboost import XGBClassifier
from sklearn.multiclass import OneVsRestClassifier

model = OneVsRestClassifier(XGBClassifier(scale_pos_weight=3,
                                          use_label_encoder=False, eval_metric="logloss"))
model.fit(X_train, y_train)

```

숨겨진 출력 표시

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

for i in range(y_test.shape[1]):
    fpr, tpr, _ = roc_curve(y_test[:, i], y_pred[:, i])
    plt.plot(fpr, tpr, label=f'Label {i+1} (AUC = {auc(fpr, tpr):.2f})')

plt.xlabel('False Positive Rate') # False를 True로 잘못 판단
plt.ylabel('True Positive Rate') # True를 False로 잘못 판단
plt.title('ROC Curve')
plt.legend()
plt.show()
```



⇨ **불균형 분포에 최적인 XGBoost의 예측성능과 [4],[6]의 예측 성능이 비슷함.**

⇨ **[4],[6]이 최적의 파라미터 값을 가지고 주어진 데이터셋을 label에 분류하는 기술을 구현함.**

⇨ **최적의 파라미터를 찾아 학습하는 모델 발견**