REPORT

과목: 기계학습
과제명: 분류 예측 모델 구현 및 분석
학번: 202020933
이름: 송주훈
제출일: 2025-04-19

1. 프로젝트 개요

● 문제 정의

본 프로젝트의 목적은 버섯의 다양한 형태적 특징(모양, 색, 등)을 기반으로, **버섯에 독이 있는지(poisonous) 혹은 먹을 수 있는지(edible)를 예측하는 분류 모델을 개발**하는 것이다.
타겟 변수인 'Mushroom_quality(p, e)'에 대해 예측 모델을 구축하고, 예측 성능을 비교·분석한다.

● 데이터셋 설명

- 출처: https://www.kaggle.com/datasets/rinichristy/uci-mushroom-dataset
- 데이터셋 크기: 8124개의 샘플 × 23개의 변수
- 종속변수 (Target):
    1. Mushroom_quality: 독성의 존재 여부를 판단
- 주요 독립변수 (Features):
    cap_shape, cap_surface, cap_color, bruises, odor, gill_attachment,
    gill_spacing, gill_size, gill_color, stalkshape, stalk_root,
    stalk_surface_above_ring, stalk_surface_below_ring, stalk_color_above_ring,
    stalk_color_below_ring, veil_color, ring_number, ring_type,
    spore_print_color, population, habitat

2. 데이터 전처리 및 탐색적 분석

● 결측치 처리

```
print("\n결측치 확인:")
print(df.isnull().sum())
```

```
결측치 확인:
Mushroom_quality            0
cap_shape                   0
cap_surface                 0
cap_color                   0
bruises                     0
odor                        0
gill_attachment             0
gill_spacing                0
gill_size                   0
gill_color                  0
stalkshape                  0
stalk_root                  0
stalk_surface_above_ring    0
stalk_surface_below_ring    0
stalk_color_above_ring      0
stalk_color_below_ring      0
veil_type                   0
veil_color                  0
ring_number                 0
ring_type                   0
spore_print_color           0
population                  0
habitat                     0
dtype: int64
```

모든 변수에서 결측치는 존재하지 않음 (isnull().sum() 결과 전체 0).
→ 하지만 직접 데이터를 확인해 보니 veil_type 변수의 경우 데이터 값이 모두 같다는 것을 확인했다.

```
# 현재 veil_type 특징의 경우 모두 같은 값이기 때문에 삭제 진행
print("---------")
print((df == 'p').sum())


df = df.drop(['veil_type'], axis=1)
```

```
veil_type                       8124
```

위 결과와 같이 전체 8124개의 데이터 중 veil_type 변수는 8124개의 'p' 값이 존재한다.
따라서 drop을 통해 veil_type 변수를 삭제하도록 하고 추가적으로 변수 중 '?'의 값을 갖는 변수 또한 아래와 같이 수정을 진행했다.

```
# 현재 stalk_root의 경우 총 8124개의 데이터 중 2480개의 '?' 값을 지니고 있음.
print("----------")
print((df == '?').sum())
```

```
stalk_root                      2480
```

위 결과를 보면 stalk_root 변수의 경우 전체 데이터의 1/4 값이 '?' 데이터로 구성되어 있다. 이를 해결하고자 '?'가 포함되어 있는 전체 행을 제거하게 되면은 너무 많은 양의 데이터가 손실되기 때문에 **최빈값**으로 대체를 진행하였다.

```
# '?' → Na로 변환 후 최빈값으로 대체
df['stalk_root'].replace('?', pd.NA, inplace=True)
# df['stalk_root'].mode()[0] => 최빈값 중 첫 번째 값만 반환
df['stalk_root'].fillna(df['stalk_root'].mode()[0], inplace=True)
```

이처럼 stalk_root 변수에 존재하는 모든 '?' 데이터를 최빈값으로 수정하였다.

● 데이터 분할 방식

```
X = df.drop('Mushroom_quality', axis=1)
y = df['Mushroom_quality']
```

```
from sklearn.model_selection import train_test_split

# 학습용과 테스트용 데이터 분할 (기본 75:25 비율, 랜덤시드 고정)
# stratify=y를 넣으면, 타겟 클래스의 비율이 train/test 양쪽에 동일하게 유지됨
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y  # stratify는 클래스 비율 유지
)
```

데이터 분할을 하기 전에 인코딩/스케일링을 하면, 훈련 데이터뿐만 아니라 테스트 데이터에 대한 정보도 인코더와 스케일러가 알게 되기 때문에 이는 시험 전에 시험지를 전달하는 것과 같은 상황이 되어버린다.
따라서 이전에 미리 train_test_split을 활용하여 사전에 데이터를 학습용 75%, 테스트용 25%로 나눠 데이터 분할을 진행했다.

● 범주형 변수 처리

```
# 범주형 변수가 존재하는지 확인
categorical_cols = X_train.select_dtypes(include='object').columns.tolist()
print(categorical_cols)
```

```
['cap_shape', 'cap_surface', 'cap_color', 'bruises', 'odor',
```

범주형 변수 인코딩을 진행하는 이유는 범주형 변수가 유한하고 고정된 수의 카테고리 또는 그룹을 나타내는 변수이기 때문이다. 따라서 범주형 데이터를 모델이 이해하고 처리할 수 있도록 수치 형태로 변환하는 과정이 필수적으로 진행되는데 이 과정이 바로 범주형 변수 인코딩 과정이다.
해당 데이터셋의 경우 모든 데이터가 범주형 변수로 구성되어 있기에 'OneHotEncoder'을 활용하여 인코딩을 진행하고자 한다.

```
# oneHotEncoder 준비
ohe = OneHotEncoder(
    drop='first',           # 첫 번째 카테고리 더미 하나를 제거해서 다중공선성 완화
    handle_unknown='ignore', # 학습 때 없던 카테고리가 오면, 모두 0으로 처리
    sparse_output=False             # 결과를 NumPy 배열로 반환 (희소행렬이 필요하면 True)
)
```

```
from sklearn.compose import ColumnTransformer

# ColumnTransformer를 활용하여 인코딩 적용
encoder = ColumnTransformer(
    transformers=[
        ('onehot', ohe, categorical_cols)
    ],
    remainder='passthrough'  # 나머지 컬럼은 그대로 두기
)
```

```
X_train_encoded = encoder.fit_transform(X_train)
```

```
X_test_encoded = encoder.transform(X_test)
```

위 과정은 train 데이터와 test 데이터에 대해서 인코딩을 진행하는 과정이다.
정리하면 X_train에 대해 fit을 통해 X_train을 기준으로 인코딩할 규칙을 학습하고 transform을 통해 학습한 규칙을 X_train 데이터에 적용하여 실제로 변환을 진행한 것이다. X_test에 대해서도 마찬가지로 이미 X_train에 대해서 학습된 규칙을 토대로 transform을 통해 X_test에 적용하도록 한 것이다.

이처럼 진행하는 이유는 다음과 같다.
- 모델이 학습 데이터만 보고 규칙을 정해야 하기 때문
- 똑같은 기준으로 테스트 데이터도 변환해야 훈련과 테스트가 같은 환경에서 비교할 수 있기 때문

일부 결과를 보면 다음과 같다.

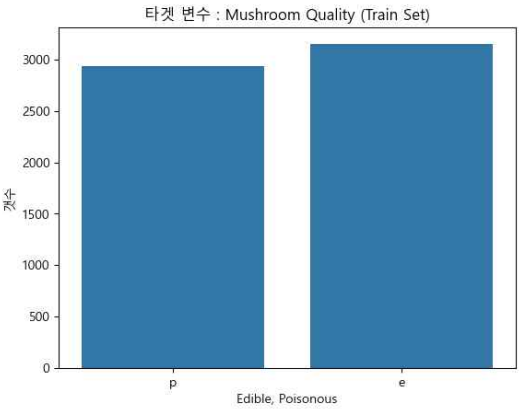
	cap_shape_c	cap_shape_f	cap_shape_k	cap_shape_s	cap_shape_x	cap_surface_g	cap_surface_s	cap_surface_y
0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0

● 스케일링

스케일링이란 수치형 변수에 대해서 데이터의 값 범위를 조정하는 과정을 말한다. 하지만 현재 데이터의 경우엔 모두 범주형 데이터이고 'OneHotEncoder'를 활용해 범주형 데이터를 0과 1로 만든 상태이다.
따라서 굳이 스케일링 과정을 진행하지 않고 넘어가도록 했다.

● EDA 시각화

```
# 타겟 변수 비율 확인
# 클래스 비율 확인
sns.countplot(x=y_train)
plt.title("타겟 변수 : Mushroom Quality (Train Set)")
plt.xlabel("Edible, Poisonous")
plt.ylabel("갯수")
plt.show()
```



먼저 타겟 변수의 비율에 대해 시각화를 진행했다.

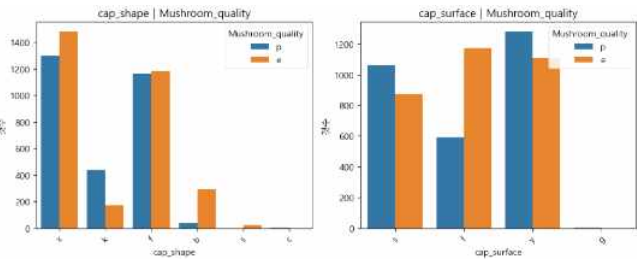
```
# 사용할 원본 데이터셋 (인코딩 전, 훈련 데이터 기준)
train_df = pd.concat([X_train, y_train], axis=1)

# 범주형 피처 리스트
cat_cols = X_train.columns

# 서브플롯 생성
n_cols = 3
n_rows = (len(cat_cols) + n_cols - 1) // n_cols
plt.figure(figsize=(5 * n_cols, 4 * n_rows))

for idx, col in enumerate(cat_cols, 1):
    plt.subplot(n_rows, n_cols, idx)
    sns.countplot(data=train_df, x=col, hue='Mushroom_quality')
    plt.title(f'{col} | Mushroom_quality')
    plt.xlabel(col)
    plt.ylabel("갯수")
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



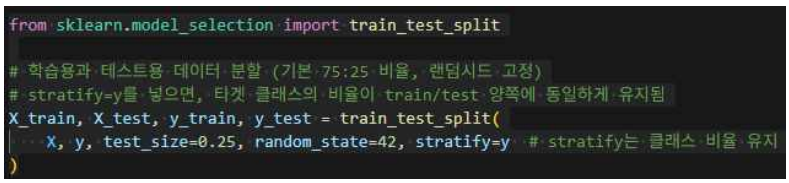
다음으로 위와 같이 veil_type를 제외한 21개의 특성과 타겟 데이터와의 관계를 표시하였다. 보고서에는 일부 특성에 대해서만 그래프를 첨부하였다.

3. 모델 구축 및 학습

● 사용한 알고리즘

- 종속변수 'Mushroom_quality'에 대해 분류 예측 모델을 구성하여 비교 분석을 진행한다:
    1. RandomForestClassifier
    2. DecisionTreeClassifier
    → 각 모델은 개별적으로 타겟에 대해 학습 및 예측을 수행함
- 해당 모델을 선택한 이유는 현재 데이터가 인코딩된 데이터이기 때문이다. 'RandomForest' 모델의 경우 과적합에 강하고, 인코딩된 데이터에 적합하다는 장점이 존재한다.
- 또한 'DecisionTree' 모델도 빠르고 해석이 강하며, 인코딩 데이터에 적합하다는 특징이 존재한다. 따라서 해당 두 모델을 선택하게 되었다.

● 데이터 분할 방식



```
from sklearn.model_selection import train_test_split

# 학습용과 테스트용 데이터 분할 (기본 75:25 비율, 랜덤시드 고정)
# stratify=y를 넣으면, 타겟 클래스의 비율이 train/test 양쪽에 동일하게 유지됨
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y  # stratify는 클래스 비율 유지
)
```

- train_test_split를 활용하여 학습 데이터와 테스트 데이터를 75:25 비율로 분할을 진행했다.

● 학습 코드 요약

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# 모델 생성
rf_clf = RandomForestClassifier(random_state=42)
dt_clf = DecisionTreeClassifier(random_state=42)

# 모델 학습
rf_clf.fit(X_train_encoded, y_train)
dt_clf.fit(X_train_encoded, y_train)

# 예측
y_pred_rf = rf_clf.predict(X_test_encoded)
y_pred_dt = dt_clf.predict(X_test_encoded)
```

위 요약된 코드와 같이 다음과 같은 과정을 거쳤다. => 모델 생성, 모델 학습, 예측
먼저 모델을 생성한 후 사전에 준비한 X_train_encoded, y_train을 가지고 각 모델 학습을 진행했다. 이후 학습된 모델과 X_test_encoded을 활용하여 예측을 진행했다.

4. 성능 평가

● 사용한 지표

모델 성능 평가는 다음 여섯 가지 지표를 기반으로 수행됨:

- Accuracy(정확도)
- Precision(정밀도)
- Recall(재현율)
- F1-score
- Confusion Matrix
- ROC-AUC

● 모델별 성능 비교

☐ RandomForestClassifier 예측 성능

지표	Accuracy	Precision	Recall	F1-score	ROC-AUC
점수	1.0	1.0	1.0	1.0	1.0

☐ DecisionTreeClassifier 예측 성능

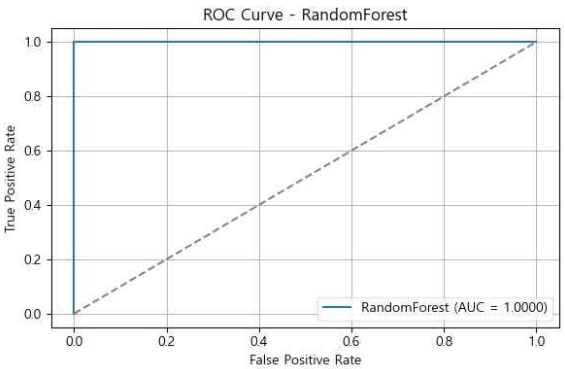
지표	Accuracy	Precision	Recall	F1-score	ROC-AUC
점수	1.0	1.0	1.0	1.0	1.0

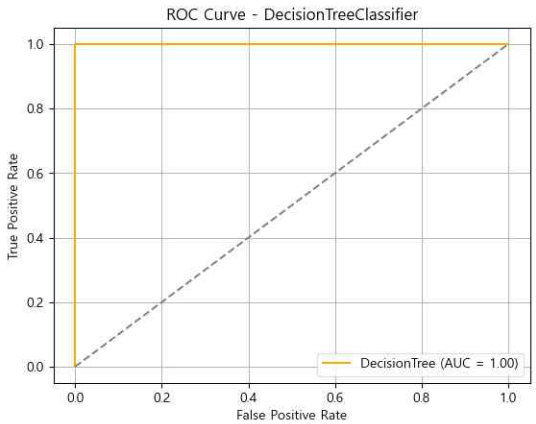
RandomForest 모델 :

```
Confusion Matrix:
 [[1052    0]
 [   0  979]]
```

DecisionTree 모델 :

```
Confusion Matrix:
 [[1052    0]
 [   0  979]]
```





위와 같이 현재 모델 학습 및 평가 결과 21개의 독립변수를 활용한 경우 모두 1의 정확도가 나오는 것을 확인했다. 이유를 찾아보니 현재 데이터셋의 높은 완성도로 인해 모든 독립변수를 활용하게 되면 1의 정확도가 나오는 것을 확인했다.

따라서 모델을 비교하고 추가적인 진행을 위해 모든 독립변수를 사용하는 것이 아닌 독립변수 중 작은 영향을 미치는 변수를 찾아 해당 변수만을 활용해 다시 학습을 진행하고 이후 파라미

터 튜닝까지 진행해 보고자 한다.

하나씩 대입을 진행한 결과 가장 큰 영향을 주는 독립변수는 'odor'로 해당 독립변수 하나만 존재해도 정확도가 0.987이 나오는 것을 확인했고, 이를 제외한 영향이 비교적 적은 'cap_shape', 'cap_surface', 'cap_color'(버섯의 갓 모양, 표면, 색)을 활용하여 남은 과정을 진행해 보고자 한다.

● 모델별 성능 비교

□ 3개의 독립변수만을 활용한 RandomForestClassifier 예측 성능

지표	Accuracy	Precision	Recall	F1-score	ROC-AUC⊏
e	0.7232	0.73	0.74	0.73	0.8243
q		0.72	0.71	0.71	

□ 3개의 독립변수만을 활용한 DecisionTreeClassifier 예측 성능

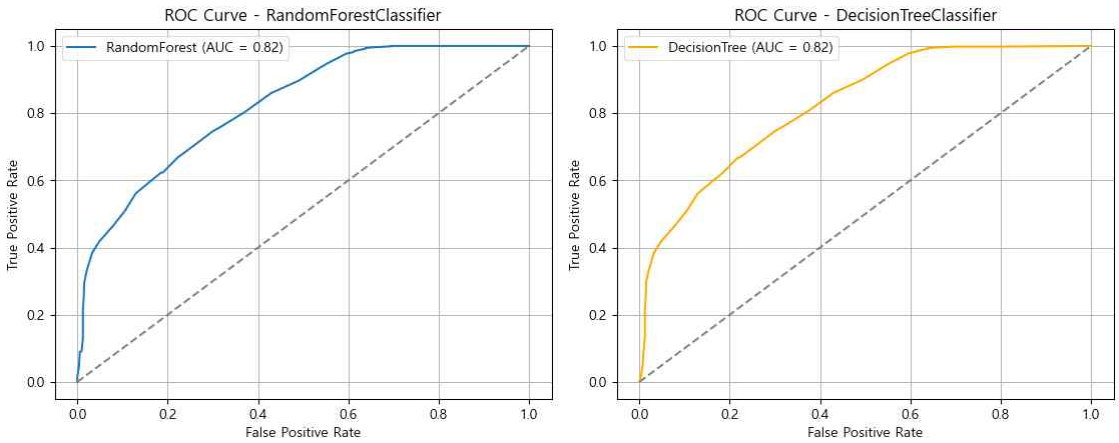
지표	Accuracy	Precision	Recall	F1-score	ROC-AUC⊏
e	0.7267	0.72	0.78	0.75	0.8239
q		0.74	0.67	0.70	

RandomForest 모델 :

```
Confusion Matrix:
 [[776 276]
 [286 693]]
```

DecisionTree 모델 :

```
Confusion Matrix:
 [[823 229]
 [326 653]]
```



위 모델의 경우 'cap_shape', 'cap_surface', 'cap_color' 3가지 독립변수에 대해서 모델 학습을 진행했고, 해당 모델에 대해 예측을 진행한 평가 지표이다.

'RandomForest' 모델과 'DecisionTree' 모델을 비교한 결과 'RandomForest' 모델이 비교적 더 높은 정확도와 안정적인 지표를 보여주었다. 하지만 두 모델 모두 정밀도, 재현율, F1 점수 모두 약간의 차이가 있지만, 전체적으로 균형 잡힌 모습을 보여주었다.
따라서 이제 하이퍼파라미터 튜닝을 통해 두 모델의 성능을 더 높이고자 한다.

5. 하이퍼파라미터 튜닝

● 튜닝 방법

- 사용 기법 : GridSearchCV
- 적용 모델 : DecisionTreeClassifier, RandomForestClassifier

● 파라미터 그리드 및 파라미터 범위

```
# 랜덤포레스트 튜닝 파라미터 후보
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

- RandomForest에서 튜닝할 수 있는 소수의 파라미터로 다음과 같이 조정하도록 했다.

1. n_estimators: 트리의 개수, 많을수록 성능은 올라가지만 학습 시간이 늘어남
2. max_depth: 개별 트리의 최대 깊이, None이면 리프 노드까지 분할
3. min_samples_split: 노드를 분할하기 위한 최소 샘플 수, 클수록 과적합 방지
4. min_samples_leaf: 리프 노드가 되기 위한 최소 샘플 수, 클수록 모델이 단순해짐
5. max_features: 각 트리를 만들 때 사용할 특성 수

```
# 결정트리 튜닝 파라미터 후보
param_grid_dt = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8],
    'criterion': ['gini', 'entropy']
}
```

- DecisionTree의 주요 파라미터는 총 4개로 다음과 같이 조정하도록 했다.

1. max_depth: 트리의 최대 깊이. None은 끝까지 분할
2. min_samples_split: 노드 분할을 위한 최소 샘플 수
3. min_samples_leaf: 리프 노드가 되기 위한 최소 샘플 수
4. criterion: 분할 기준. gini와 entropy는 가장 많이 사용

● 최적 하이퍼파라미터 및 성능

☐ Heating Load

- RandomForestClassifier

항목	값
최적 파라미터	'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 300
정확도	0.7243
정밀도, 재현율, F1 점수	0.7362 / 0.6670 / 0.6999
혼동 행렬	[[818 234] [326 653]]

- DecisionTreeClassifier,

항목	값
최적 파라미터	'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2
정확도	0.7253
정밀도, 재현율, F1 점수	0.7395 / 0.6639 / 0.6997
혼동 행렬	[[823 229] [329 650]]

● 튜닝 결과 분석

모델의 성능을 향상시키기 위해 GridSearchCV를 활용하여 RandomForestClassifier와 DecisionTreeClassifier 두 모델의 하이퍼파라미터 튜닝을 진행하였다.
튜닝 전후의 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1-score 등의 주요 지표를 비교한 결과는 다음과 같다.

□ RandomForestClassifier 튜닝 결과

- 튜닝 전 정확도: 0.7232
- 튜닝 후 정확도: 0.7243
- 튜닝 후 F1-score: 0.6999
- 재현율(Recall): 0.6670 → 클래스 'p'(독버섯)을 놓치지 않는 성능 지표
- 혼동 행렬:
  참양성(TP): 653
  거짓음성(FN): 326 → 여전히 'p'를 'e'로 잘못 판단하는 경우 존재

RandomForestClassifier는 튜닝 후 전체적인 성능(정확도, 재현율, F1-score)이 모두 소폭 상승하였으며, 특히 F1-score 향상이 눈에 띈다. 이는 정밀도와 재현율 사이 균형이 개선되었음을 의미한다. RandomForest의 앙상블 특성이 튜닝을 통해 보다 일반화된 예측력을 보이도록 도와준 것으로 판단된다.

□ DecisionTreeClassifier 튜닝 결과

- 튜닝 전 정확도: 0.7267
- 튜닝 후 정확도: 0.7253
- 튜닝 후 F1-score: 0.6997
- 재현율(Recall): 0.6639
- 혼동 행렬:
  참양성(TP): 650
  거짓음성(FN): 329

DecisionTreeClassifier는 튜닝 이후 성능 변화가 미미하거나 오히려 약간 하락하였다. 이는

단일 결정 트리 모델의 한계로, 특정 하이퍼파라미터 값에서 과적합(overfitting) 혹은 과소적합(underfitting)이 발생할 가능성이 높기 때문으로 보인다. 특히 재현율이 낮아지면서 F1-score에도 부정적인 영향을 주었다.

6. 결론 및 고찰

● 최종 모델 성능 종합 평가

본 프로젝트에서는 버섯의 여러 특성을 바탕으로 독버섯 여부(Mushroom_quality)를 분류하는 모델을 구축하였다. 그 결과, RandomForestClassifier 모델이 가장 우수한 성능을 보였으며, 하이퍼 파라미터 튜닝을 통해 정밀도와 재현율 간의 균형이 개선된 것을 확인할 수 있었다.

- 최종 선택 모델: RandomForestClassifier
- 최종 성능 요약 (튜닝 후):
  정확도 (Accuracy): 0.7243
  정밀도 (Precision): 0.7362
  재현율 (Recall): 0.6670
  F1 점수 (F1-score): 0.6999

이는 단순한 의사결정트리(DecisionTreeClassifier)에 비해 전체적으로 더 나은 예측 성능을 나타내며, 실전 적용 가능성이 높은 결과라고 할 수 있다.

● 데이터 또는 모델의 한계

- 데이터 전처리 한계: 원본 데이터에 존재하는 '?' 데이터에 대해 최빈값을 활용해 진행했지만, 이는 정확하지 않은 정보로 정보 손실의 가능성이 존재한다.
- 클래스 간 불균형 문제: Mushroom_quality의 클래스(e/p)가 거의 균형을 이루고는 있지만, 모델 성능 평가에 있어 재현율(Recall)이 낮은 것은 여전히 'p'를 충분히 포착하지 못하고 있다.

● 실생활 응용 가능성 또는 확장 방향

이 모델은 실제로 버섯 채집 어플리케이션, 식품 안전 경고 시스템 등에 적용될 수 있으며, 사용자 입력 또는 이미지 기반 데이터를 통해 독버섯 여부를 자동으로 분류하는 시스템의 핵심 구성 요소로 활용 가능하다.
또한 농업/식품 분야의 데이터 기반 안전성 판단 시스템 구축에도 확장 적용이 가능하다.

● 다음 단계에서 고려할 점

- 클래스 불균형 처리 및 결측치에 대한 고급 처리 진행
- 다양한 모델 실험 진행을 통해 고성능 앙상블 모델을 활용하여 성능 향상 가능성을 탐색

7. 참고자료

● 데이터셋 출처

https://www.kaggle.com/datasets/rinichristy/uci-mushroom-dataset

● 참고 자료

https://psystat.tistory.com/136 (OneHotEncoder)
https://scikit-learn.org/stable/modules/preprocessing.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://wikidocs.net/207524
https://velog.io/@changhtun1/%EA%B8%B0%EC%B4%88%EB%B6%80%ED%84%B0-%EC%8C%93%EC%95%84%EA%B0%80%EB%8A%94-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-3

● 사용한 주요 라이브러리

- python 3.13.2
- pandas 2.2.3
- numpy 2.2.4
- matplotlib 3.10.1
- seaborn 0.13.2
- scikit-learn 1.6.1

● 중요 코드 스니펫

- 데이터 누수 방지를 위한 분할 및 전처리

```
X = df.drop('Mushroom_quality', axis=1)
y = df['Mushroom_quality']
```

```
from sklearn.model_selection import train_test_split

# 학습용과 테스트용 데이터 분할 (기본 75:25 비율, 랜덤시드 고정)
# stratify=y를 넣으면, 타겟 클래스의 비율이 train/test 양쪽에 동일하게 유지됨
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y  # stratify는 클래스 비율 유지
)
```

```python
from sklearn.compose import ColumnTransformer

# ColumnTransformer를 활용하여 인코딩 적용
encoder = ColumnTransformer(
    transformers=[
        ('onehot', ohe, categorical_cols)
    ],
    remainder='passthrough'  # 나머지 컬럼은 그대로 두기
)
```

```python
# fit_transform을 fit, transform으로 나눠서 설명해보자
# fit = 학습 데이터를 기준으로 인코딩할 규칙을 학습
# transform = 위에서 학습한 규칙을 X_train 데이터에 적용
X_train_encoded = encoder.fit_transform(X_train)

# 이미 X_train에서 학습된 규칙을 토대로 transform을 통해
# 이때 X_test에 새로운 범주가 나타난다면 우리가 위해서 작성
X_test_encoded = encoder.transform(X_test)
```

- GridSearchCV 설정 코드

```python
# 랜덤포레스트 튜닝 파라미터 후보
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# GridSearchCV 객체 생성
rf_grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid_rf,
    cv=5,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)
```

```python
# 결정트리 튜닝 파라미터 후보
param_grid_dt = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8],
    'criterion': ['gini', 'entropy']
}

# GridSearchCV 객체 생성
dt_grid_search = GridSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_grid=param_grid_dt,
    cv=5,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)
```

- 모델 생성, 학습, 예측 진행

```python
# 모델 생성
dt_clf2 = DecisionTreeClassifier(random_state=42)

# 모델 학습
dt_clf2.fit(X_train_encoded2, y_train2)

# 예측
y_pred_dt2 = dt_clf2.predict(X_test_encoded2)
```