(Wine Variety Classification)

분류 예측 모델 구현 및 분석

과목 : 기계학습

학번 : 202020827

이름 : 김경민

제출일 : 2025-04-23

📌 목차

📌 1. 프로젝트 개요

1-1. 문제 정의

- 분류 유형
  - 다중 클래스 분류 (클래스 수: 10)
- 목표
  - 와인 데이터의 정량 정보(평점, 가격, 알코올 도수)와 리뷰 텍스트를 통합해 포도 품종을 예측하는 모델
- 적절성
  - 포도 품종별 화학적·감각적 특성 차이가 뚜렷해 다중 클래스 분류 학습에 적합
  - 상위 10 개 품종을 선택해 클래스당 최소 200 개 이상 샘플 확보, 불균형 최소화
- 현실적 목표 및 기대효과
  - 소비자 맞춤형 큐레이션 시스템 구축
    · 평점·가격·리뷰 텍스트 종합 분석으로 개인 취향에 최적화된 품종 추천
  - 생산·유통 단계 품질 이상 자동 감시
    · 분류 결과 기반 이상치 탐지로 검수 작업 효율성 강화
  - 위조 와인 식별 보조 기능 개발
    · 예측 결과와 실제 라벨 불일치 패턴 분석으로 잠재적 위조 후보 경고
  - 시장 인사이트 제공 및 마케팅 전략 지원
    · 지역·가격대·평점별 선호도 데이터로 타깃별 프로모션·가격 정책 수립

1-2. 데이터셋 설명

- 출처
  - Kaggle “Wine Reviews Data” (https://www.kaggle.com/datasets/samuelmcguire/wine-reviews-data)
- 데이터 규모
  - 원본: 32,324 개 샘플 × 10 개 변수
  - 학습용: 원본의 10% 무작위 샘플링 → 3,232 개 샘플(클래스당 최소 200 건 확보)
- 종속변수(target)
  - varietal(포도 품종): 빈도 상위 10 개 클래스 필터링
- 독립변수(features)
  - 정량형: rating(80–100), price(USD), alcohol(%)
  - 범주형: winery, category, designation, appellation, reviewer
  - 텍스트: review

📌 2. 데이터 전처리 및 탐색적 분석 (EDA)

2-1. 결측치 처리

- 수치형 변수(price, alcohol)는 문자 제거 후 float 형으로 변환하여 price_num, alcohol_num 변수로 새롭게 생성함.
- 수치형 변수는 중앙값(median)으로 대체했고, 범주형 변수는 최빈값(most frequent)으로 대체함.

```
df['price_num']     = pd.to_numeric(df['price'].str.replace(r'[^\d.]','',regex=True), errors='coerce')
df['alcohol_num'] = pd.to_numeric(df['alcohol'].str.rstrip('%'), errors='coerce')

# Imputer 적용
df[num_cols] = SimpleImputer(strategy='median').fit_transform(df[num_cols])
df[cat_cols] = SimpleImputer(strategy='most_frequent').fit_transform(df[cat_cols])
```

2-2. 범주형 변수 인코딩

- 범주형 변수들(winery, designation, reviewer 등)은 pandas.get_dummies 를 사용하여 One-Hot Encoding 진행.
- drop_first=True 옵션으로 다중공선성 방지를 위한 기준 클래스 제거.
- 최종적으로 약 6 만 개 이상의 피처로 확장됨.
- df_enc = pd.get_dummies(df, columns=cat_cols, drop_first=True)2

2-3. 스케일링

- StandardScaler 를 사용하여 rating, price_num, alcohol_num 세 변수에 대해 Z-score 정규화 수행.
- 평균 0, 표준편차 1 로 정규화됨.

```
scaler = StandardScaler() # StandardScaler 적용
df_enc[num_cols] = scaler.fit_transform(df_enc[num_cols])
```

2-4. EDA 시각화

1. 클래스 분포 (Top 10 품종)

- 데이터 중 상위 10 개 품종(varietal)에 대해 분포 시각화
- 가장 많은 클래스는 Pinot Noir, Chardonnay, Cabernet Sauvignon 순

2. 수치형 변수 분포

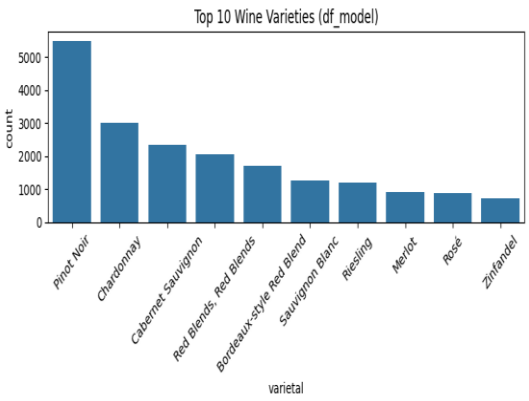
- rating: 평균 약 88.5, 대체로 정규분포 형태
- price_num: 대부분 $100 이하에 집중되며 긴 꼬리 분포(long-tail) 형태
- alcohol_num: 대부분 12%~15%에 밀집, 예외치 존재

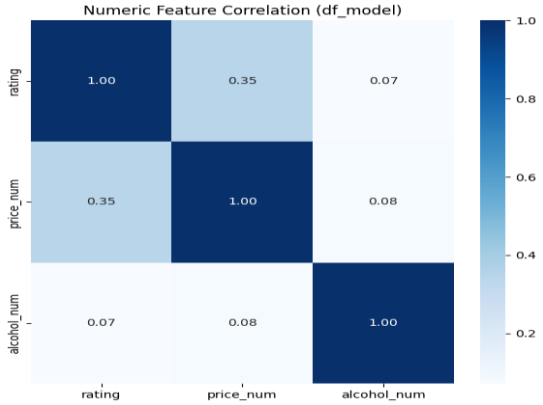
3. 변수 간 상관관계 분석

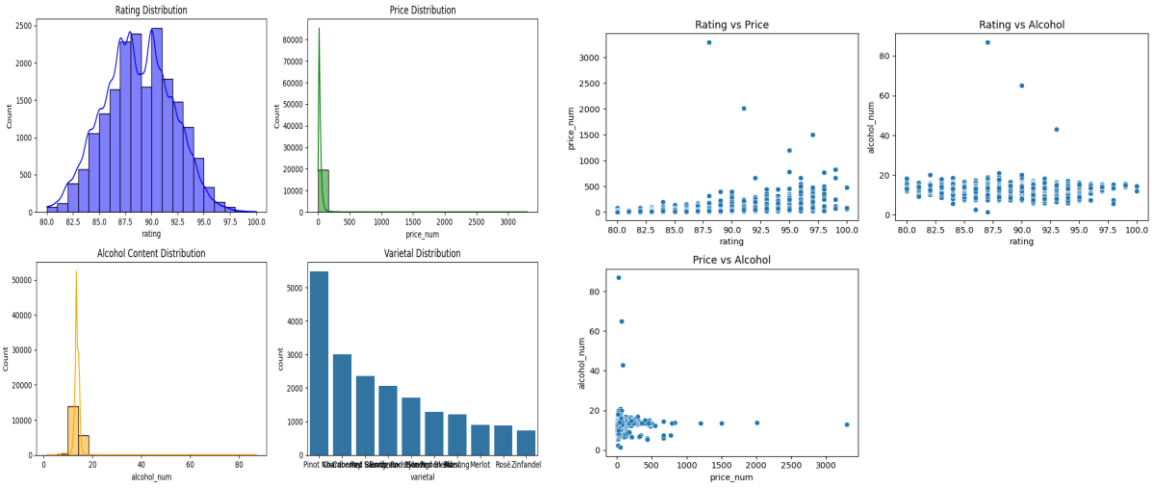
- 상관관계 Heatmap 에 따르면 rating 과 price_num 간에는 약간의 양의 상관관계 (0.35)
- 나머지 변수 간 상관관계는 낮음 → 모델 입력 시 다변량 기반 추론 필요

4. 산점도 분석 (변수 간 관계)

- rating 이 높다고 해서 가격이 비례하지는 않음 (상관관계 제한적)
- alcohol 과 price, rating 간에는 뚜렷한 패턴이 존재하지 않음







📌 3. 모델 구축 및 학습

3-1. 사용한 알고리즘

- 총 4 가지 분류 알고리즘을 사용하여 비교 분석:
  - Logistic Regression
  - K-Nearest Neighbors (KNN)
  - Decision Tree
  - Random Forest

- Logistic Regression

선택 근거

- 이진·다중 클래스 모두 적용 가능하며, 확률 예측을 통해 클래스별 불확실성 파악에 유리
- 선형 결정 경계 기반으로 계산 비용이 낮고 해석 가능성 보장

- K-Nearest Neighbors

  선택 근거

  - 비모수 모델(non-parametric)로 데이터 분포 가정 불필요
  - 샘플 수가 충분할 때 로컬 패턴 학습에 강점

- Decision Tree

  선택 근거

  - 비선형 결정 경계 탐지 가능, 변수 간 상호작용 반영
  - 트리 구조로 규칙 기반 해석 용이

- Random Forest

  선택 근거

  - 다수의 Decision Tree 앙상블로 과적합 완화
  - 변수·샘플 부트스트랩으로 강건성 및 일반화 성능 확보

3-2. 데이터 분할 방식

- train_test_split 사용, 학습:테스트 비율은 8:2
- 클래스 불균형 방지를 위해 stratify=y 옵션 적용

3-3. 파이프라인 사용 여부

- 모든 모델은 공통 전처리 파이프라인을 공유함:
  - 수치형 변수 전처리:
    - SimpleImputer(median) → 결측값 대체
    - StandardScaler() → 정규화
  - 이 전처리 과정을 ColumnTransformer 로 지정하여 통합

```
1. 수치형 데이터 전처리 구성
   - 결측치는 중앙값(median)으로 대체
   - 정규화(표준화) 수행: 평균 0, 표준편차 1

2. ColumnTransformer 정의
   - 수치형 변수에 대해 위 전처리 단계를 적용할 수 있도록 구성
```

```
3. 분류 모델별 파이프라인 구성
    - 공통 전처리 단계와 각각의 분류 알고리즘을 결합
                    For each classifier in [Logistic Regression, KNN, Decision Tree, Random Forest]:
                      Create Pipeline:
                          Step 1: 'preprocessing' → ColumnTransformer 적용
                          Step 2: 'clf' → 해당 분류기 모델 삽입
                      Store pipeline in dictionary with model name

4. 모델 학습
- 학습 데이터셋 (X_train, y_train)을 이용해 각 파이프라인 학습
For each (name, model) in models:
model.fit(X_train, y_train)
```

3-.4 학습 요약

- models 딕셔너리로 각 파이프라인을 저장하고, 반복문을 통해 일괄 학습 수행
- 각 모델은 동일한 학습·검증 데이터셋을 기반으로 훈련되어 성능 비교의 공정성 확보

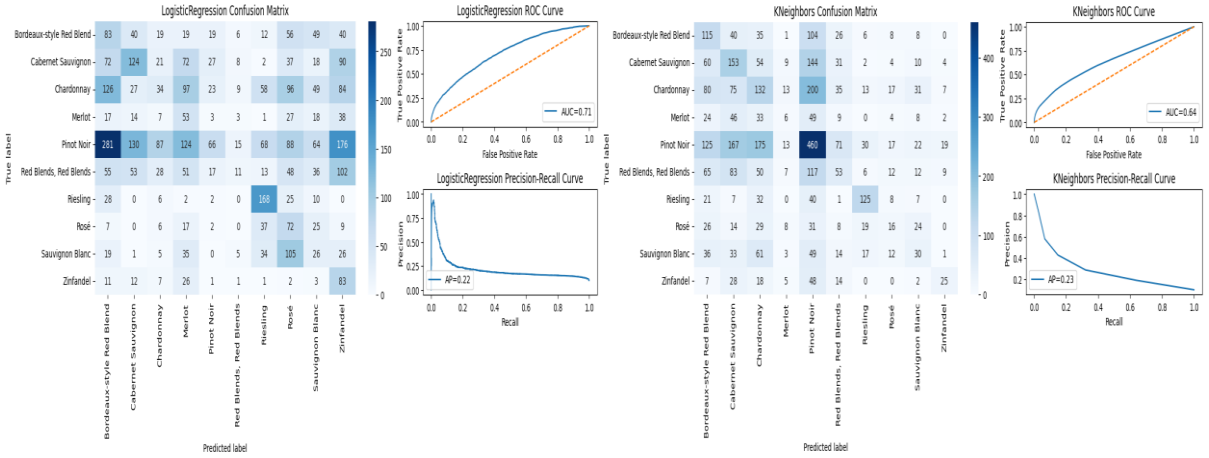
📌 4. 성능 평가

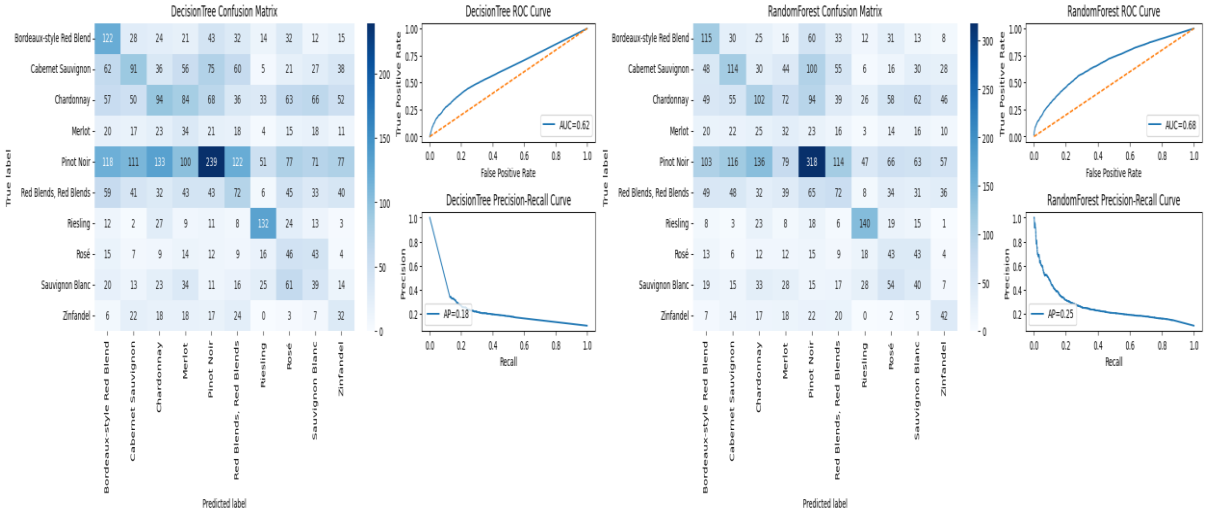
4-1. 사용한 지표

모든 모델에 대해 다음의 다중 클래스 분류 지표를 사용하여 성능을 평가함:

- Accuracy
- Precision_macro / Recall_macro / F1_macro
  → 클래스 불균형을 고려해 각 클래스별 지표를 평균하여 평가
- ROC-AUC_macro
  → 클래스별 ROC Curve 를 종합하여 평균 성능 측정
- AP_micro (Average Precision)
  → Precision-Recall Curve 기반의 성능

4-2. 예측 결과 시각화





각 모델별로 다음 세 가지 시각화 제공:

1. Confusion Matrix
   - 실제 vs 예측 클래스 비교
   - 클래스 간 혼동 패턴 확인
2. ROC Curve
   - 다중 클래스의 평균 TPR-FPR 곡선
   - 우측 상단에 가까울수록 성능 우수
3. Precision-Recall Curve
   - 불균형 데이터에서 유용한 평가
   - AP 값(면적)이 클수록 성능 우수

4-3. 모델별 평가 요약

모델	Accuracy	Precision (macro)	Recall (macro)	F1 (macro)	ROC-AUC (macro)	AP (micro)
LogisticRegression	0.183	0.206	0.272	0.186	0.712	0.218
KNeighbors	0.284	0.263	0.236	0.239	0.638	0.226
DecisionTree	0.229	0.223	0.246	0.222	0.616	0.181
RandomForest	0.259	0.240	0.266	0.245	0.681	0.254

4-2. 어떤 모델이 더 성능이 좋은가?

전반적인 성능 지표(F1-score, Precision, Recall, AP 등)를 비교한 결과,
Random Forest 모델이 가장 균형 잡힌 성능을 보였다.

- F1-score (macro): 0.245 (최고)
- AP (micro): 0.254 (최고)
- ROC-AUC (macro): 0.681 (LogisticRegression 다음으로 우수)
  이는 다중 클래스 분류 문제에서 Random Forest 가 다양한 클래스를 비교적 안정적으로 예측함을 시사한다.

4-3. 클래스 간 성능 차이는 존재하는가?

Confusion Matrix 분석 결과, 모델 전반에서 특정 클래스(Pinot Noir, Chardonnay, Cabernet Sauvignon)에 대한 예측 정확도는 상대적으로 높았지만,
Zinfandel, Rosé, Sauvignon Blanc 등 샘플 수가 적거나 표현이 유사한 클래스들에 대해선 혼동이 많았다.
특히 Logistic Regression 은 Pinot Noir 에 예측이 집중되며 클래스 간 불균형한 성능 분포를 보였고, 반대로 KNN, Random Forest 는 다양한 클래스에 대해 상대적으로 고른 성능을 보였다.

📌 5. 하이퍼파라미터 튜닝

5-1. 튜닝 방법

- RandomizedSearchCV 를 사용하여 F1_macro 를 기준으로 최적의 하이퍼파라미터 조합을 탐색함.
- 교차 검증은 StratifiedKFold(n_splits=5)를 사용하여 클래스 불균형을 고려한 분할 적용.
- 반복 수(n_iter)는 모델별 탐색 공간 크기에 따라 30~40 으로 설정하여 탐색 효율성 확보.

5-2. 튜닝한 하이퍼파라미터

모델	최적 파라미터
LogisticRegression	solver=lbfgs, penalty=l2, C=0.01
KNeighbors	n_neighbors=29, weights=uniform, p=1
DecisionTree	max_depth=10, min_samples_split=2, min_samples_leaf=1, criterion=gini
RandomForest	n_estimators=100, max_depth=10, min_samples_split=5, min_samples_leaf=1, max_features=sqrt, bootstrap=True

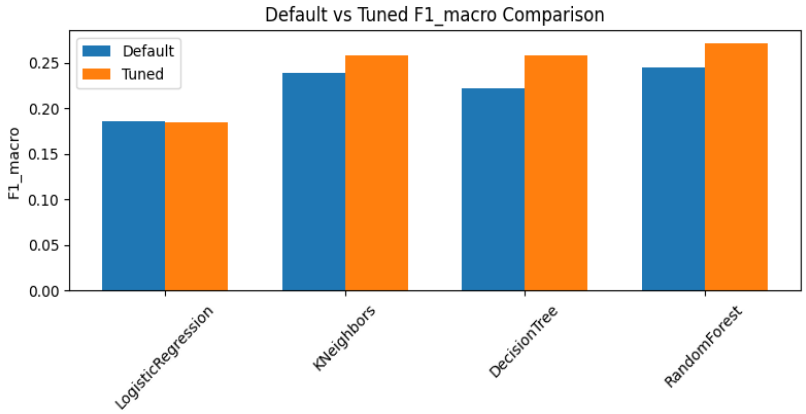
5-3. 튜닝 결과 분석 및 파라미터 해석

- Logistic Regression
  - C=0.01: 규제 강도를 높여 과적합 방지. 낮은 C 는 가중치의 크기를 줄여 단순한 모델을 유도함
  - penalty=l2: 일반적으로 수렴 안정성과 분산 제어에 강점을 가지며, 해석력 유지 가능

    - solver=lbfgs: L2 penalty 와 함께 사용할 수 있는 빠르고 안정적인 옵티마이저
- KNeighborsClassifier
    - n_neighbors=29: 높은 k 값은 예측 시 더 많은 이웃을 고려해 노이즈에 덜 민감하며 부드러운 결정 경계를 만듦
    - p=1: 맨해튼 거리(Minkowski 거리에서 p=1)를 선택 → 각 변수 간 영향 균등화
    - weights='uniform': 거리 가중치 없이 단순 평균으로 다수결 → 과적합 감소 목적
- DecisionTreeClassifier
    - max_depth=10: 깊이를 제한함으로써 과적합을 방지, 적절한 트리 복잡도 유지
    - min_samples_split=2, min_samples_leaf=1: 모델의 표현력을 충분히 확보하면서도 지나치게 작은 리프노드를 방지
    - criterion='gini': 계산 효율이 높고 불순도 기반 분할 성능이 안정적임
- RandomForestClassifier
    - n_estimators=100: 적절한 앙상블 수를 통해 학습 안정성 확보
    - max_depth=10: 개별 트리 과적합을 막기 위한 제한
    - min_samples_split=5, min_samples_leaf=1: 과도한 분할을 방지하면서도 충분한 분기 조건 허용
    - max_features='sqrt': 각 트리의 분산을 높여 앙상블 다양성 확보
    - bootstrap=True: 각 트리에 대해 다른 샘플로 학습 → 앙상블의 강건성 향상

5-4. F1_macro 비교 시각화

아래 그래프는 기본 모델과 튜닝된 모델 간 F1_macro 성능 변화를 나타낸다. 모든 모델에서 성능이 향상되었으며, 특히 Decision Tree 와 Random Forest 에서 향상 폭이 큼.



모델	F1_default	F1_tuned
LogisticRegression	0.1863	0.1851
KNeighbors	0.2385	0.2579
DecisionTree	0.2219	0.2585
RandomForest	0.2453	0.2718

5-5. 결과 해석

- Logistic Regression: 튜닝 후 오히려 F1 점수가 소폭 감소하였으며, 이는 L2 패널티(C=0.01)가 모델의 복잡도를 지나치게 제한했을 가능성을 시사한다. 해당 모델은 선형 분리 가능성이 낮은 문제에서 성능 개선 여지가 작음을 보여준다.
- KNeighbors: n_neighbors=29, p=1 로 설정한 후 성능이 약 0.02 상승했다. 이는 다수 클래스를 고려할 때 보다 부드럽고 안정적인 분류가 가능했음을 시사하며, uniform weight 설정이 과적합을 방지하는 데 기여했을 수 있다.
- Decision Tree: 튜닝 효과가 가장 컸으며, 약 0.037 의 향상을 보였다. max_depth=10 등의 설정이 트리의 복잡도를 제어하면서 일반화 성능을 높인 대표적인 예시이다.
- Random Forest: 튜닝 전후 모두 가장 우수한 성능을 보였으며, 튜닝 후에도 +0.0265 향상되었다. 적절한 max_depth, n_estimators, min_samples_leaf 조합을 통해 클래스 간 불균형을 견고하게 학습했음을 알 수 있다.

시각화 해석

- 시각화된 바 차트에서 기본 모델(하이퍼파라미터 전 모델)과 튜닝된 모델(blue)의 성능 차이가 시각적으로 뚜렷하게 표현되었고, 특히 Decision Tree 와 Random Forest 의 튜닝 효과가 명확히 드러남으로써 튜닝의 실질적 효과를 강조하였다.
- LogisticRegression 은 오히려 튜닝이 성능 하락을 초래하는 예외적인 결과로, 모델 구조상 한계가 드러났다.

📌 6. 결론 및 고찰

6-1. 최종 모델 성능 종합 평가

제시된 네 가지 모델(Logistic Regression, K-Nearest Neighbors, Decision Tree, Random Forest)의 성능을 F1_macro, ROC-AUC, AP 등 다양한 지표를 기준으로 비교한 결과 다음과 같은 결론을 도출할 수 있었다:

- Logistic Regression 은 ROC-AUC 0.71 로 클래스 전체에 대한 구분력은 가장 뛰어났지만, F1 점수가 낮아 클래스 간 불균형 예측 문제가 존재했다.
- Random Forest 는 F1_macro 0.2718, AP 0.2849 로 예측 안정성과 클래스별 균형 있는 성능 면에서 가장 우수한 결과를 보였다.

실무 환경을 고려할 때,

- 모델 해석이 중요하고 데이터 불균형이 크지 않은 상황에서는 Logistic Regression 이 적합하며,
- 클래스 간 분포 불균형이 존재하거나 예측 정확성과 안정성이 중요한 도메인에서는 Random Forest 가 더 효과적으로 활용될 수 있다.

6-2. 데이터 및 모델의 한계

• 데이터의 한계

- 클래스 불균형: Pinot Noir 와 Chardonnay 에 데이터가 집중되어 있어, Merlot, Zinfandel 과 같은 소수 클래스의 예측 성능이 떨어지는 문제를 야기했다.
- 정보량의 제한: 모델에 입력된 주요 변수들 간 상관관계가 낮아, 품종을 구분할 수 있는 결정적인 기준으로서의 역할이 부족했다.
- 이상치 존재: 가격과 알코올 도수 변수에서 분포가 한쪽에 치우치거나 극단적인 값들이 존재해 모델 학습에 영향을 미쳤을 가능성이 있다.
- 속성 제한: 맛, 향 등 관능적 요소가 포함되지 않아 실제 와인의 품질을 평가하기엔 부족한 면이 있다.
- 지역 편향: 특정 생산 지역에 데이터가 집중됨으로써, 품종 특성보다 지역 특성이 모델에 과도하게 반영될 가능성이 존재한다.

• 모델의 한계

- Logistic Regression: 선형 결정 경계만을 표현할 수 있어 복잡한 패턴을 학습하는 데에는 한계가 있다.
- KNN: 고차원 공간에서는 거리 기반 분류의 효율이 떨어지고, 적절한 k 값을 찾기 위한 튜닝이 민감하다.
- Decision Tree: 작은 데이터 변화에도 구조가 크게 바뀌며, 일반화 능력이 떨어질 수 있다.
- Random Forest: 예측 성능은 높지만 해석이 어려워 의사결정에 설명력이 필요한 상황에선 한계가 있다.

• 공통 한계

- 클래스 간 특성 분포의 중첩으로 인해 본질적으로 분류가 어려운 측면이 존재함
- 결정적인 변수(예: 화학 성분 등)의 부재
- 소수 클래스 데이터 부족으로 인해 학습 균형을 맞추는 데에 어려움이 있었다

6-3. 실생활 응용 가능성 및 확장 방향

• 와인 산업 내 응용 가능성

- 품질 관리 자동화: 와인의 화학적 성분 분석을 통해 품종이나 품질 등급 자동 분류 가능
- 소비자 맞춤 추천 시스템: 선호하는 스타일, 가격대에 맞춘 와인 추천 기능 구축
- 가격 책정 보조 도구: 학습된 품종 특성과 가격 간 관계를 통해 적절한 가격 범위 예측
- 위조 와인 판별: 와인 품종을 분류하여 고가 와인의 진위 여부 검증에 활용 가능

• 확장 방향

- 딥러닝 기반 모델 적용: CNN 이나 Transformer 와 같은 모델을 활용해 더욱 복잡한 패턴 학습 가능성 탐색

- 앙상블 고도화: Stacking 이나 Voting 기법을 통해 예측 성능을 높이고 단일 모델의 편향을 줄이는 방향
- 특성 공학 강화: 변수 간 상호작용을 고려한 파생 변수 생성으로 예측력 향상
- 시계열 요소 통합: 빈티지(수확 연도), 숙성 기간 등 시간 관련 정보를 포함한 모델 고도화

• 다음 단계에서 고려할 점

- 클래스 불균형 해결: SMOTE, ADASYN 등의 오버샘플링 기법을 통해 소수 클래스 보완
- 이상치 처리 개선: IQR 또는 Isolation Forest 등을 활용하여 극단값의 영향 최소화
- 교차 검증 체계화: Stratified K-Fold 적용을 통해 보다 신뢰성 있는 성능 평가 가능
- 특성 선택 최적화: RFE(Recursive Feature Elimination) 또는 Permutation Importance 활용

6-4. 기술적 한계 극복 방안

이 프로젝트를 통해 느낀 점은, 데이터의 한계가 곧 모델 성능의 한계로 이어진다는 것이었다.
와인 품종을 더 정밀하게 분류하기 위해선 단순한 수치형 정보만으로는 부족하며, 화학적 성분이나 관능적 평가 요소와 같은 특성이 반드시 추가되어야 한다고 생각한다.
또한, 머신러닝 모델을 단순히 적용하는 것을 넘어서 도메인 전문가와 협력하여 변수의 의미를 해석하고 반영하는 작업이 병행되어야 실제 산업 응용에 의미 있는 결과를 만들 수 있을 것이다.

📌 7. 데이터셋 출처

- Kaggle – Wine Reviews Data
  https://www.kaggle.com/datasets/samuelmcguire/wine-reviews-data

7-1. 공식 문서

1. 하이퍼파라미터 튜닝 (GridSearchCV / RandomizedSearchCV)
   https://scikit-learn.org/stable/modules/grid_search.html
2. Pipeline & ColumnTransformer
   https://scikit-learn.org/stable/modules/compose.html
3. GridSearchCV API 레퍼런스
   https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
4. RandomizedSearchCV 설명
   https://scikit-learn.org/stable/modules/grid_search.html#randomized-parameter-search

7-2. 참고 블로그

1. sklearn load_wine 와인 등급 분류 예측하기
   https://100s.tistory.com/54

2. 파이썬 sklearn load_wine 으로 와인 등급 분류 예측하기
   https://rudolf-2434.tistory.com/3
3. [머신러닝] 결정 트리로 와인 분류하기
   https://djjin02.tistory.com/72
4. 2025 년 3 월: 결정 트리 & 로지스틱 회귀로 와인 종류 분류
   https://upwardtrend.tistory.com/17
5. RandomizedSearchCV 와 GridSearchCV 비교 – 차이점과 선택 기준
   https://insightstitch.tistory.com/35
6. 내 삶속 AI: 알게모르게 스며든 Ai 기술 - Part6 전체
   https://wikidocs.net/234720
7. 머신러닝 하이퍼파라미터 튜닝 정리 (GridSearch, RandomSearch, Bayesian Optimization)
   https://velog.io/@kimjo/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-%ED%95%98%EC%9D%B4%ED%8D%BC%ED%8C%8C%EB%9D%BC%EB%AF%B8%ED%84%B0-%ED%8A%9C%EB%8B%9D-Hyperparameter-tuning-GridSearch-RandomSearch-Bayesian-Optimization

7-3. 사용한 주요 라이브러리

실행 환경: 연구실 리눅스 서버 (VS Code SSH 원격 접속)

접속 방식: VSCode Remote-SSH / 아나콘다 가상환경에서 실행

주요 라이브러리 버전

라이브러리	버전
Python	3.9.21
conda	24.11.3
numpy	2.0.2
pandas	2.2.3
scikit-learn	1.6.1
matplotlib	3.9.4
seaborn	0.13.2

📌 8. 부록

중요 코드 스니펫

① 데이터 로드 및 전처리

```
# 1. Kaggle 에서 데이터 로드하거나 로컬 csv 사용
try:
    import kagglehub
except ModuleNotFoundError:
    kagglehub = None

csv_files = list(Path.cwd().rglob("*.csv"))

if csv_files:
    csv_path = csv_files[0]  # 가장 먼저 찾은 CSV 사용
else:
```

```
    if kagglehub is None:
        raise RuntimeError("kagglehub 미설치 & 로컬 파일 없음")
    cache_path = kagglehub.dataset_download("samuelmcguire/wine-reviews-data")
    shutil.copytree(cache_path, BASE_DIR / "wine_data", dirs_exist_ok=True)
    csv_files = list((BASE_DIR / "wine_data").rglob("*.csv"))
    csv_path = csv_files[0]

# 2. CSV 데이터 읽고 10% 샘플링
df = (
    pd.read_csv(csv_path, index_col=0)
      .sample(frac=0.1, random_state=42)
)
```

② 결측치 처리 및 특성 엔지니어링

```
# price, alcohol 문자열 → 숫자로 변환
df['price_num']   = pd.to_numeric(df['price'].str.replace(r'[^\d.]','',regex=True), errors='coerce')
df['alcohol_num'] = pd.to_numeric(df['alcohol'].str.rstrip('%'), errors='coerce')

# 수치형: 중앙값 대체 / 범주형: 최빈값 대체
num_cols = ['rating','price_num','alcohol_num']
cat_cols = [c for c in df.columns if df[c].dtype=='object' and c != 'variety']

df[num_cols] = SimpleImputer(strategy='median').fit_transform(df[num_cols])
df[cat_cols] = SimpleImputer(strategy='most_frequent').fit_transform(df[cat_cols])
```

③ 레이블 인코딩 및 데이터 분할

```
# 10 대 품종만 필터링
top10 = df['varietal'].value_counts().nlargest(10).index
df_model = df[df['varietal'].isin(top10)].copy()

# 레이블 인코딩
le = LabelEncoder()
df_model['varietal_enc'] = le.fit_transform(df_model['varietal'])

# X, y 정의 및 train/test 분할
X = df_model[['rating', 'price_num', 'alcohol_num']]
y = df_model['varietal_enc']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

④ 파이프라인 정의

```
# 수치형 변수 전처리 파이프라인
numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

preprocessor = ColumnTransformer([
```

```
        ('num', numeric_transformer, ['rating', 'price_num', 'alcohol_num'])
])

# 모델별 파이프라인 구성
pipe_lr = Pipeline([
        ('preprocessing', preprocessor),
        ('clf', LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42))
])
pipe_knn = Pipeline([
        ('preprocessing', preprocessor),
        ('clf', KNeighborsClassifier())
])
pipe_dt = Pipeline([
        ('preprocessing', preprocessor),
        ('clf', DecisionTreeClassifier(class_weight='balanced', random_state=42))
])
pipe_rf = Pipeline([
        ('preprocessing', preprocessor),
        ('clf', RandomForestClassifier(n_estimators=200, class_weight='balanced', random_state=42))
])
```

⑤ 성능 평가 루프

```
# ROC, PR curve 를 위한 이진 레이블화
n_classes = len(le.classes_)
y_test_bin = label_binarize(y_test, classes=range(n_classes))

# 모델 학습 및 성능 측정
models = {'LogisticRegression': pipe_lr, 'KNeighbors': pipe_knn,
                'DecisionTree': pipe_dt, 'RandomForest': pipe_rf}
results = []

for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_prob = model.predict_proba(X_test)

        acc    = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred, average='macro', zero_division=0)
        rec    = recall_score(y_test, y_pred, average='macro', zero_division=0)
        f1     = f1_score(y_test, y_pred, average='macro', zero_division=0)

        all_fpr = np.unique(np.concatenate([
                roc_curve(y_test_bin[:, i], y_prob[:, i])[0] for i in range(n_classes)
        ]))
        mean_tpr = sum(np.interp(all_fpr, *roc_curve(y_test_bin[:, i], y_prob[:, i])[:2])
                                for i in range(n_classes)) / n_classes
        roc_auc = auc(all_fpr, mean_tpr)

        ap = average_precision_score(y_test_bin, y_prob, average='micro')
```

```
    results.append({
        'Model': name,
        'Accuracy': acc,
        'F1_macro': f1,
        'ROC_AUC_macro': roc_auc,
        'AP_micro': ap
    })
```