변수 중요도: 트리 기반 모델의 `feature_importances_`

- 머신러닝 모델이 예측을 수행하는 데 있어 각 입력 변수(피처)가 얼마나 중요한 역할을 하는지를 나타내는 지표
- 특히 트리 기반 모델(예: 결정 트리, 랜덤 포레스트, 그래디언트 부스팅)에서는 모델 학습 과정에서 각 피처가 불순도(impurity)를 얼마나 감소시켰는지 또는 오차를 얼마나 줄였는지 등을 기반으로 `feature_importances_` 속성을 통해 각 변수의 중요도를 수치화하여 제공
- 일반적으로 0과 1 사이의 값을 가지며, 모든 피처 중요도의 합은 1이 됨

적용 가능한 상황

- **피처 선택(Feature Selection)**: 모델 성능에 크게 기여하지 않는 불필요한 피처를 제거하여 모델을 단순화하고 과적합을 방지하며, 학습 시간을 단축할 수 있습니다.
- **모델 해석(Model Interpretation)**: 어떤 피처가 모델의 예측에 가장 큰 영향을 미치는지 이해함으로써 모델의 의사결정 과정을 설명할 수 있습니다.
- **도메인 지식 강화**: 데이터에 대한 새로운 통찰력을 얻고, 비즈니스 문제 해결에 중요한 변수를 식별하는 데 활용할 수 있습니다.
- **데이터 전처리 전략 수립**: 중요도가 낮은 피처에 대한 추가적인 전처리나 변환의 필요성을 판단하는 데 도움을 줍니다.

주의사항

- **상관관계가 높은 피처**: 여러 피처가 서로 강한 상관관계를 가질 경우, 모델은 이들 중 하나만 선택하여 불순도를 감소시킬 수 있습니다. 이 경우, 다른 상관관계가 높은 피처들의 중요도가 낮게 평가될 수 있어 해석에 주의가 필요합니다.
- **카디널리티(Cardinality)가 높은 피처에 대한 편향**: `feature_importances_`는 카디널리티가 높은(즉, 고유한 값이 많은) 범주형 피처나 연속형 피처에 더 높은 중요도를 부여하는 경향이 있을 수 있습니다.
- **모델 의존성**: `feature_importances_`는 특정 모델(트리 기반)에 한정된 방법이므로, 다른 종류의 모델(예: 선형 모델, SVM)에는 직접 적용할 수 없습니다.
- **순열 중요도(Permutation Importance)와의 비교**: `feature_importances_`는 학습 데이터 내에서 계산되므로, 훈련 데이터에 대한 모델의 성능을 기반으로 합니다. 반면 순열 중요도는 모델이 학습된 후 특정 피처의 값을 무작위로 섞었을 때 모델 성능이 얼마나 저하되는지를 측정하여, 테스트 데이터에 대한 모델의 일반화 성능 관점에서 중요도를 평가합니다. 두 방법을 보완적으로 사용하는 것이 좋습니다.

하이퍼파라미터 설명

`feature_importances_`는 모델의 속성(attribute)이므로 직접적인 하이퍼파라미터는 아닙니다. 하지만 모델의 하이퍼파라미터 설정(예: `n_estimators`, `max_depth`, `min_samples_split` 등)은 `feature_importances_` 계산 결과에 영향을 미칠 수 있습니다.

- `n_estimators` (트리 개수): 앙상블 모델에서 트리의 개수가 많아질수록 중요도 추정치가 더 안정적이고 신뢰할 수 있게 됩니다.
- `max_depth` (트리 최대 깊이): 트리의 깊이가 깊어질수록 더 많은 피처가 분할에 사용될 기회를 얻게 되어 중요도 분포에 영향을 줄 수 있습니다.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.datasets import make_classification

# 1. 데이터 준비
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,
n_redundant=2, random_state=42)
feature_names = [f'feature_{i}' for i in range(X.shape[1])]
X_df = pd.DataFrame(X, columns=feature_names)

X_train, X_test, y_train, y_test = train_test_split(X_df, y, test_size=0.2,
random_state=42)

# 2. RandomForestClassifier를 이용한 변수 중요도 확인
print("\n--- RandomForestClassifier Feature Importance ---")
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

rf_importances = rf_model.feature_importances_
rf_feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':
rf_importances})
rf_feature_importance_df = rf_feature_importance_df.sort_values(by='Importance',
ascending=False)
print(rf_feature_importance_df)
'''
     Feature  Importance
4  feature_4    0.206109
9  feature_9    0.167711
0  feature_0    0.159516
5  feature_5    0.143680
3  feature_3    0.120340
6  feature_6    0.069936
1  feature_1    0.065147
8  feature_8    0.025048
2  feature_2    0.021608
7  feature_7    0.020905
'''

# 3. XGBClassifier를 이용한 변수 중요도 확인
print("\n--- XGBClassifier Feature Importance ---")
xgb_model = XGBClassifier(n_estimators=100, use_label_encoder=False,
eval_metric='logloss', random_state=42)
xgb_model.fit(X_train, y_train)

xgb_importances = xgb_model.feature_importances_
xgb_feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':
xgb_importances})
xgb_feature_importance_df = xgb_feature_importance_df.sort_values(by='Importance',
```

```
ascending=False)
print(xgb_feature_importance_df)
'''
     Feature  Importance
9  feature_9    0.246176
4  feature_4    0.179213
5  feature_5    0.166644
3  feature_3    0.139662
0  feature_0    0.121020
1  feature_1    0.054325
6  feature_6    0.047176
8  feature_8    0.024042
7  feature_7    0.013442
2  feature_2    0.008300
'''

# 4. 변수 중요도 시각화 (RandomForest)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=rf_feature_importance_df)
plt.title('RandomForest Feature Importances')
plt.tight_layout()
plt.show()

# 5. 변수 중요도 시각화 (XGBoost)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=xgb_feature_importance_df)
plt.title('XGBoost Feature Importances')
plt.tight_layout()
plt.show()
```

결과 해석 방법

- `feature_importances_` 값은 각 피처의 상대적인 중요도를 나타냅니다. 값이 높을수록 해당 피처가 모델의 예측에 더 큰 영향을 미친다고 해석할 수 있습니다.
- 시각화(막대 그래프)를 통해 피처 중요도를 한눈에 파악하고, 중요도가 높은 피처들을 쉽게 식별할 수 있습니다.
- 중요도가 낮은 피처들은 모델에서 제거하는 것을 고려하여 피처 선택에 활용할 수 있습니다. 예를 들어, 중요도가 특정 임계값(예: 0.01)보다 낮은 피처들을 제거하거나, 상위 N개의 피처만 선택하여 모델을 재학습할 수 있습니다.

장단점 및 대안

장점	단점
- 구현이 간단하고 직관적	- 상관관계가 높은 피처들 사이에서 중요도 해석이 어려울 수 있음
- 모델 해석에 유용하며, 피처 선택의 근거를 제공	- 카디널리티가 높은 피처에 편향될 수 있음

장점	단점
- 트리 기반 모델에 내장되어 있어 쉽게 접근 가능	- 트리 기반 모델에만 적용 가능

대안

- **순열 중요도 (Permutation Importance)**: 모델 학습 후 특정 피처의 값을 무작위로 섞었을 때 모델의 성능이 얼마나 감소하는지를 측정하여 중요도를 평가합니다. 모델에 구애받지 않고(model-agnostic) 작동하며, 상관관계가 높은 피처에 대한 편향이 적다는 장점이 있습니다.
  - `sklearn.inspection.permutation_importance` 함수를 사용합니다.
- **SHAP (SHapley Additive exPlanations)**: 게임 이론에 기반하여 각 피처가 예측에 기여하는 정도를 공정하게 분배하여 중요도를 계산합니다. 개별 예측에 대한 피처 중요도와 전역적인 피처 중요도를 모두 제공하며, 모델에 구애받지 않습니다.
- **LIME (Local Interpretable Model-agnostic Explanations)**: 개별 예측에 대해 모델이 어떻게 작동하는지 설명하기 위해 지역적으로 해석 가능한 모델을 학습시켜 피처 중요도를 파악합니다.
- **재귀적 피처 제거 (Recursive Feature Elimination, RFE)**: 모델을 반복적으로 학습시키면서 중요도가 가장 낮은 피처를 하나씩 제거하여 최적의 피처 조합을 찾는 방법입니다.
  - `sklearn.feature_selection.RFE` 클래스를 사용합니다.