트리 기반 회귀 모델 (Tree-Based Regression)

- 데이터 공간을 재귀적으로 분할하여 예측을 수행하는 알고리즘
- 데이터의 특성(feature)을 기반으로 스무고개와 같은 질문(분기 규칙)을 연속적으로 만들어나가며, 최종적으로 각 데이터가 속하는 영역(리프 노드, leaf node)을 결정
- 회귀 문제에서는 해당 리프 노드에 속한 훈련 데이터들의 타겟 값의 평균으로 예측값을 결정

1. **결정 트리 회귀 (Decision Tree Regressor)**:
    - 단일 트리를 사용하여 예측을 수행
    - 각 분기점(노드)에서는 모든 특성에 대해 분할 조건을 탐색하며,
      분산(variance) 또는 평균 제곱 오차(MSE)를 가장 많이 감소시키는 최적의 분할을 탐색
    - 이 과정을 반복하여 트리를 성장시키고, 더 이상 분할할 수 없거나 특정 정지 조건(e.g. 최대 깊이, 최소 샘플 수)에 도달하면 정지
    - **장점**: 모델이 직관적이고 시각화하여 해석하기 쉽습니다.
    - **단점**: 과적합(Overfitting)되기 매우 쉬워 일반화 성능이 떨어질 수 있습니다.
2. **랜덤 포레스트 회귀 (Random Forest Regressor)**:
    - 결정 트리의 과적합 문제를 해결하기 위해 등장한 **앙상블(Ensemble)** 기법
    - **배깅(Bagging)**: 원본 훈련 데이터에서 중복을 허용하여 여러 개의 샘플(부트스트랩 샘플)을 만듭니다.
    - **무작위 특성 선택**: 각 샘플로 결정 트리를 훈련시키되, 각 노드에서 분할을 위한 최적의 특성을 찾을 때 모든 특성을 고려하는 것이 아니라, 무작위로 선택된 일부 특성 중에서만 탐색합니다.
    - **결과 취합**: 이렇게 만들어진 수많은(e.g., 100개) 결정 트리들의 예측값들을 **평균**내어 최종 예측값으로 사용합니다.
    - **장점**: 개별 결정 트리보다 훨씬 안정적이고 높은 예측 성능을 보이며, 과적합에 강합니다.
    - **단점**: 단일 트리보다 모델이 복잡해져 해석이 어렵고, 훈련 속도가 느립니다.

적용 가능한 상황

- 데이터의 비선형 관계나 특성 간의 복잡한 상호작용을 모델링해야 할 때 매우 효과적입니다.
- 특성의 스케일에 영향을 받지 않으므로, 별도의 스케일링 전처리가 필수는 아닙니다.
- 이상치에 비교적 덜 민감하게 반응합니다.

구현 방법

`scikit-learn`의 `tree` 모듈에 있는 `DecisionTreeRegressor`와 `ensemble` 모듈의 `RandomForestRegressor`를 사용합니다.

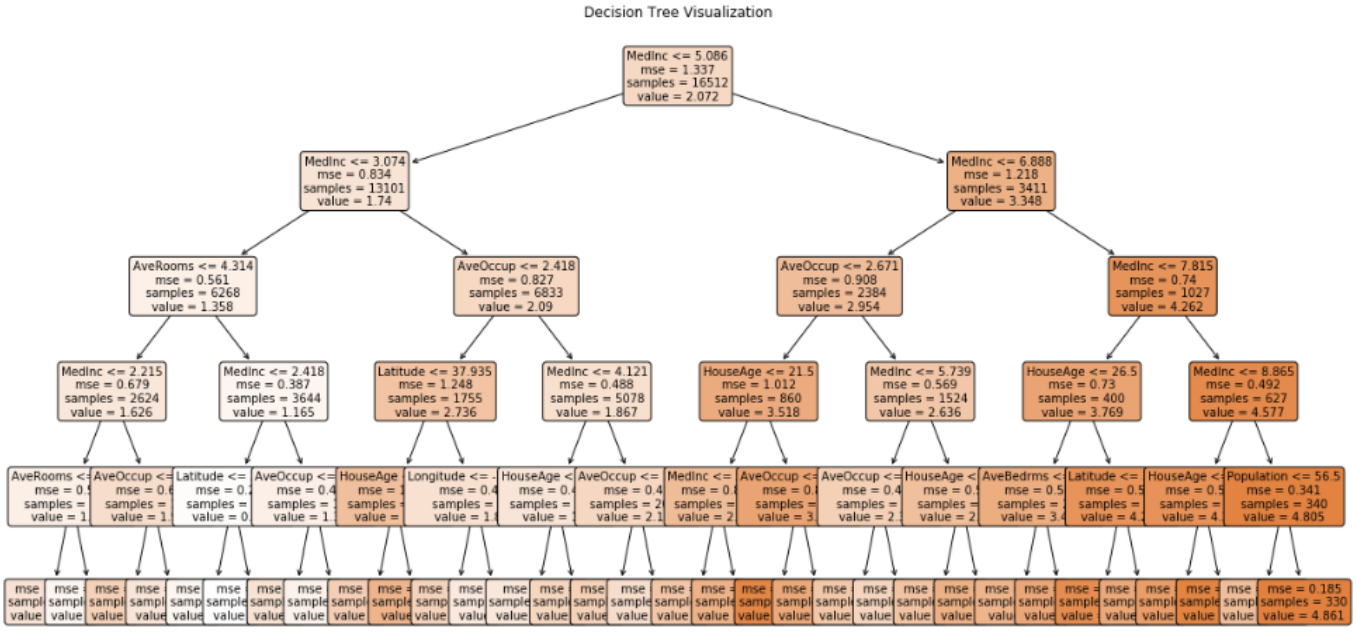
주의사항

- **하이퍼파라미터 튜닝**: 트리 기반 모델은 과적합을 방지하고 최적의 성능을 내기 위해 하이퍼파라미터 튜닝이 매우 중요합니다.
    - `max_depth`: 트리의 최대 깊이. 너무 깊으면 과적합됩니다.
    - `min_samples_split`: 노드를 분할하기 위한 최소 샘플 수.
    - `min_samples_leaf`: 리프 노드가 되기 위한 최소 샘플 수.
    - `max_features` (랜덤 포레스트): 각 분할에서 고려할 최대 특성의 수.

- **random_state**: DecisionTreeRegressor와 RandomForestRegressor 모두 무작위성을 포함하므로, 결과 재현을 위해 random_state를 고정하는 것이 좋습니다.

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# 1. 데이터 준비
housing = fetch_california_housing()
X = housing.data
y = housing.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 2. 결정 트리 회귀 (Decision Tree Regressor)
# 주요 하이퍼파라미터
# max_depth: 트리의 최대 깊이.
# min_samples_split: 노드를 분할하기 위한 최소 샘플 수.
# min_samples_leaf: 리프 노드가 되기 위한 최소 샘플 수.
dt_reg = DecisionTreeRegressor(max_depth=5, random_state=42)
dt_reg.fit(X_train, y_train)
dt_pred = dt_reg.predict(X_test)
print(f"Decision Tree MSE: {mean_squared_error(y_test, dt_pred):.3f}") # 0.525

# 3. 랜덤 포레스트 회귀 (Random Forest Regressor)
# 주요 하이퍼파라미터
# n_estimators: 생성할 트리의 개수.
# max_features: 각 분할에서 고려할 최대 특성 수.
# n_jobs: 병렬 처리에 사용할 CPU 코어 수. -1이면 모두 사용.
rf_reg = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42,
n_jobs=-1)
rf_reg.fit(X_train, y_train)
rf_pred = rf_reg.predict(X_test)
print(f"Random Forest MSE: {mean_squared_error(y_test, rf_pred):.3f}") # 0.297

# 4. 특성 중요도 (Feature Importance) 확인
# 랜덤 포레스트는 각 특성이 모델의 예측에 얼마나 중요한지를 측정하는
feature_importances_ 속성을 제공
importances = pd.Series(rf_reg.feature_importances_, index=housing.feature_names)
print("\nFeature Importances:\n", importances.sort_values(ascending=False))
'''
MedInc        0.593833
AveOccup      0.139737
Latitude      0.076563
Longitude     0.076109
HouseAge      0.047918
AveRooms      0.031580
```

```
Population    0.017259
AveBedrms     0.017000
'''

# 5. 모델 시각화 (결정 트리)
plt.figure(figsize=(20, 10))
plot_tree(dt_reg, feature_names=housing.feature_names, filled=True, rounded=True,
fontsize=10)
plt.title("Decision Tree Visualization")
plt.show()
```



결과 해석 방법

- **모델 시각화 (`plot_tree`)**: 결정 트리의 분기 규칙을 시각적으로 확인할 수 있습니다.
  - 각 노드에는 분기 조건, `mse`(해당 노드의 평균 제곱 오차), `samples`(해당 노드에 속한 샘플 수), `value`(해당 노드의 평균 타겟 값, 즉 예측값)가 표시됩니다.
  - 이 시각화를 통해 모델이 어떤 특성을 기준으로 어떻게 예측을 수행하는지 직관적으로 이해할 수 있습니다.
- **특성 중요도 (`feature_importances_`)**: 랜덤 포레스트나 결정 트리에서 제공하는 이 속성은 모델이 예측을 만들 때 각 특성이 얼마나 중요하게 사용되었는지를 나타냅니다. 값이 높을수록 중요한 특성입니다. 이를 통해 변수 선택이나 모델 해석에 도움을 받을 수 있습니다.

장단점 및 대안

- **장점**:
  - 비선형 관계와 변수 간 상호작용을 잘 모델링합니다.
  - 특성 스케일링이 필수가 아니며, 이상치에 비교적 강건합니다.
  - 결정 트리는 시각화를 통해 해석이 용이하고, 랜덤 포레스트는 높은 성능과 특성 중요도 정보를 제공합니다.
- **단점**:
  - 결정 트리는 과적합되기 매우 쉽습니다.

    - 랜덤 포레스트는 훈련 데이터 범위 밖의 값을 예측하기 어렵고, 모델이 복잡하여 해석이 어렵습니다.
    - 데이터가 작을 경우 성능이 불안정할 수 있습니다.
- **대안**:
    - **부스팅 계열 모델 (Gradient Boosting, XGBoost, LightGBM)**: 랜덤 포레스트와 유사한 앙상블 기법이지만, 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리를 만들어나가므로 일반적으로 더 높은 성능을 보입니다.
    - **SVR (Support Vector Regressor)**: 비선형 데이터에 대해 다른 접근 방식을 제공하며, 특히 고차원 데이터에서 좋은 성능을 낼 수 있습니다.
    - **다항 회귀/GAM**: 비선형성을 모델링하는 또 다른 방법으로, 모델의 해석이 더 중요할 때 고려할 수 있습니다.