

2장 머신러닝 프로젝트 처음부터 끝까지 (3부)

개요

1. 실전 데이터 활용
2. 큰 그림 그리기
3. 데이터 훑어보기
4. 데이터 탐색과 시각화
5. 데이터 준비
6. 모델 선택과 훈련
7. 모델 미세 조정
8. 최적 모델 저장과 활용



2.6 모델 선택과 훈련

모델 훈련과 평가

- 지금까지 한 일
 - 훈련셋 / 테스트셋 구분
 - 변환 파이프라인을 활용한 데이터 전처리
- 이제 할 일
 - 회귀 모델 선택 후 훈련
 - 예제: 선형 회귀, 결정트리, 랜덤 포레스트

모델 선택

- 목표: 구역별 중간 주택 가격 예측 모델
- 학습 모델: 회귀 모델 지정
- 회귀 모델 성능 측정 지표 지정: 평균 제곱근 오차(RMSE)를 기본으로 사용

선형 회귀 모델

- 선형 회귀 모델 생성: 사이킷런의 `LinearRegression` 클래스 활용
- 훈련 및 예측: 4장에서 자세히 소개.

```
from sklearn.linear_model import LinearRegression  
  
lin_reg = make_pipeline(preprocessing, LinearRegression())  
lin_reg.fit(housing, housing_labels)  
lin_reg.predict(housing)
```

선형 회귀 모델 훈련 결과

- RMSE(평균 제곱근 오차): 68687 정도로 별로 좋지 않음.
- 훈련된 모델이 훈련셋에 **과소적합** 됨.
- 보다 좋은 특성을 찾거나 더 강력한 모델을 적용해야 함.

결정트리 회귀 모델

- 결정트리 회귀 모델 생성: 사이킷런의 `DecisionTreeRegressor` 클래스 활용
- 훈련 및 예측: 6장에서 자세히 소개

```
from sklearn.tree import DecisionTreeRegressor  
  
tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))  
tree_reg.fit(housing, housing_labels)  
housing_predictions = tree_reg.predict(housing)  
  
tree_rmse = mean_squared_error(housing_labels, housing_predictions,  
...                                squared=False)
```

결정트리 회귀 모델 훈련 결과

- RMSE(평균 제곱근 오차)가 0으로 완벽해 보임.
- 훈련된 모델이 훈련셋에 심각하게 **과대적합** 됨.
- 실전 상황에서 RMSE가 0이 되는 것은 불가능.
- 테스트셋에 대한 RMSE가 매우 높음

랜덤 포레스트 회귀 모델

- 결정트리 회귀 모델 생성: 사이킷런의 `RandomForestRegressor` 클래스 활용
- 훈련 및 예측: 100개의 결정트리 동시에 훈련. 7장에서 자세히 소개

```
from sklearn.ensemble import RandomForestRegressor  
forest_reg = make_pipeline(preprocessing.  
                           RandomForestRegressor(n_estimators=100, random_state=42))
```

교차 검증

- 테스트셋을 사용하지 않으면서 훈련 과정을 평가할 수 있음.

k-겹 교차 검증

- 훈련셋을 폴드_{fold}라 불리는 k-개의 부분 집합으로 무작위로 분할 후 k번 훈련 진행
 - 매 훈련마다 돌아가면 하나의 폴드를 선택하여 검증 데이터셋으로 지정.
 - 나머지 (k-1) 개의 폴드를 대상으로 훈련
 - 매 훈련이 끝날 때마다 선택된 검증 데이터셋을 이용하여 모델 평가
- 최종평가는 k-번 평가 결과의 평균값.
- k = 5인 경우



`cross_val_score` 함수

- 예제: 결정 트리 모델 교차 검증 ($k = 10$ 인 경우)

```
from sklearn.model_selection import cross_val_score
tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
...                             scoring="neg_root_mean_squared_error", cv=10)
```

- `scoring` 키워드 인자: k -겹 교차 검증의 모델 학습 과정에서 훈련 중인 모델의 성능 평가용 **효용함수** 지정
 - 문제는 RMSE는 낮을 수록 좋은 모델을 의미함. 따라서 효용함수로 RMSE의 음수값을 사용.
 - `scoring="neg_mean_squared_error"`
- 교차 검증의 RMSE: 계산된 `cross_val_score()` 반환값을 다시 양수로 전환.
 - 평균 RMSE: 약 71407
 - 별로 좋지 않음.

2.7 모델 미세 조정

- 가장 좋은 성능의 모델을 선정한 후에 모델의 세부 설정을 조정해서 모델의 성능을 보다 끌어 올릴 수 있음
- 모델 미세 조정을 위한 세 가지 방식
 - 그리드 탐색
 - 랜덤 탐색
 - 양상블 방법

그리드 탐색

- 지정한 하이퍼파라미터의 모든 조합을 교차검증하여 최선의 하이퍼파라미터 조합 찾기
- 사이킷런의 `GridSearchCV` 활용

예제: 랜덤 포레스트와 그리드 탐색

- 총 ($3 \times 3 + 2 \times 3 = 15$) 가지의 경우 확인
- 3-겹 교차검증(`cv=3`)이므로, 총 ($15 \times 3 = 45$)번 훈련함.

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])

param_grid = [
    {'preprocessing__geo_n_clusters': [5, 8, 10],
     'random_forest__max_features': [4, 6, 8]},
    {'preprocessing__geo_n_clusters': [10, 15],
     'random_forest__max_features': [6, 8, 10]},
]

grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')

grid_search.fit(housing, housing_labels)
```

그리드 탐색 결과

- 최고 성능의 랜덤 포레스트 하이퍼파라미터가 다음과 같음.
 - `n_clusters`: 15
 - `max_features`: 6
- 최고 성능의 랜덤 포레스트에 대한 교차검증 RMSE: 44042
 - 하나의 랜덤 포레스트보다 좀 더 좋아졌음.

랜덤 탐색

- 그리드 탐색은 적은 수의 조합을 실험해볼 때 유용
- 조합의 수가 커지거나, 설정된 탐색 공간이 커지면 랜덤 탐색이 효율적
 - 설정값이 연속적인 값을 다루는 경우 랜덤 탐색이 유용
- 사이킷런의 `RandomizedSearchCV` 추정기가 랜덤 탐색을 지원

예제: 랜덤 포레스트와 랜덤 탐색

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distributions = {'preprocessing__geo__n_clusters': randint(low=3, high=50),
...                      'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
...     full_pipeline, param_distributions=param_distributions, n_iter=10, cv=3,
...     scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
```

예제: 랜덤 포레스트와 랜덤 탐색

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distributions = {'preprocessing__geo__n_clusters': randint(low=3, high=50),
...                      'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
...     full_pipeline, param_distributions=param_distributions, n_iter=10, cv=3,
...     scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
```

- `n_iter=10`: 랜덤 탐색이 총 10회 진행
 - `n_clusters` 와 `max_features` 값을 지정된 구간에서 무작위 선택
- `cv=3`: 3-겹 교차검증. 따라서 랜덤 포레스트 학습이 $(10 \times 3 = 30)$ 번 이루어짐.

랜덤 탐색 결과

- 최고 성능의 랜덤 포레스트 하이퍼파라미터가 다음과 같음.
 - `n_estimators`: 45
 - `max_features`: 9
- 최고 성능의 랜덤 포레스트에 대한 교차검증 RMSE: 41995

앙상블 방법

- 결정 트리 모델 하나보다 랜덤 포레스트처럼 여러 모델로 이루어진 모델이 보다 좋은 성능을 낼 수 있음.
- 또한 최고 성능을 보이는 서로 다른 개별 모델을 조합하면 보다 좋은 성능을 얻을 수 있음
- [앙상블 학습과 랜덤 포레스트](#)에서 자세히 다룸

최적 모델 활용

- 그리드 탐색과 랜덤 탐색 등을 통해 얻어진 최적의 모델을 분석해서 문제에 대한 좋은 통찰을 얻을 수 있음
- 예를 들어, 최적의 랜덤 포레스트 모델에서 사용된 특성들의 중요도를 확인하여 일부 특성을 제외할 수 있음.
 - 중간 소득(`log_median_income`)이 가장 중요
 - 해안 근접도 특성 중에선 `INLAND` 특성이 중요

특성 중요도

```
[ (0.18694559869103852, 'log__median_income'),
  (0.0748194905715524, 'cat__ocean_proximity_INLAND'),
  (0.06926417748515576, 'bedrooms_ratio__bedrooms_ratio'),
  (0.05446998753775219, 'rooms_per_house__rooms_per_house'),
  (0.05262301809680712, 'people_per_house__people_per_house'),
  (0.03819415873915732, 'geo__Cluster_0_similarity'),
  [...]
  (0.00015061247730531558, 'cat__ocean_proximity_NEAR BAY'),
  (7.301686597099842e-05, 'cat__ocean_proximity_ISLAND')]
```

테스트셋 활용 최종 평가

- 최고 성능 모델을 테스트셋에 적용하여 최종 평가

```
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set[ "median_house_value" ].copy()

final_predictions = final_model.predict(X_test)

final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
```

2.8. 최적 모델 저장 및 활용

- 긴 훈련으로 찾은 최고 성능의 모델을 저장하면 언제든 재 활용 가능
- 완성된 모델 저장

```
joblib.dump(final_model, "my_california_housing_model.pkl")
```

- 저장된 모델 불러오기

```
final_model_reloaded = joblib.load("my_california_housing_model.pkl")
final_model_reloaded.predict(X_test)
```