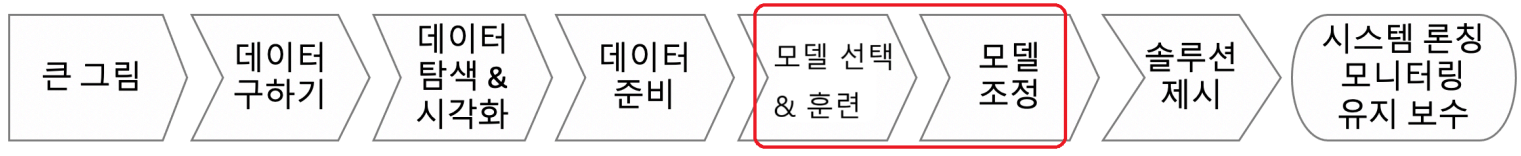
2장 머신러닝 프로젝트 처음부터 끝까지 (3부)

개요


큰 그림
데이터 구하기
데이터 탐색 & 시각화
데이터 준비
모델 선택 & 훈련
모델 조정
솔루션 제시
시스템 론칭 모니터링 유지 보수

2.7. 모델 선택과 훈련

2.7.1. 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 주택 중위가격 예측 모델
- 학습 모델: 회귀 모델 지정
- 회귀 모델 성능 측정 지표 지정: RMSE (평균 제곱근 오차)

RMSE

- Root-mean-square error (평균 제곱근 오차).
- 예측 오차의 제곱의 평균값에 루트를 씌운 값.
- 0에 가까울 수록 모델의 예측 성능이 좋음.

선형 회귀 모델

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

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(housing, housing_labels)
lin_reg.predict(housing)
lin_rmse = mean_squared_error(housing_labels, housing_predictions,
                              squared=False)
```

선형 회귀 모델 훈련 결과

- RMSE(평균 제곱근 오차): 68687.89 정도로 나쁨.
- 모델의 훈련셋에 대한 성능이 낮아서 **과소적합** 현상이 발생했다고 말할 수 있음.
- 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

- RMSE가 0으로 완벽해 보임.
- 모델이 훈련셋에 심각하게 **과대적합** 됨.
- 실전 상황에서 RMSE가 0이 되는 것은 불가능.
- 테스트셋에 대한 RMSE가 매우 높음

랜덤 포레스트 회귀 모델

- 랜덤 포레스트 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))
forest_reg.fit(housing, housing_labels)
housing_predictions = forest_reg.predict(housing)

forest_rmse = mean_squared_error(housing_labels, housing_predictions,
                                 squared=False)
```

랜덤 포레스트 회귀 모델 훈련 결과

- RMSE(`tree_rmse`)가 17474 정도로 선형회귀 모델 보다 헐씬 낮음.
- 테스트셋에 대한 RMSE가 보다 높기는 함.
- 하지만 결정트리 모델 보다 **과대적합** 현상이 매우 적게 발생함.

2.7.2. 교차 검증

k-겹 교차 검증

- 폴드 생성: 훈련셋을 **폴드**fold라 불리는 k-개의 부분 집합으로 무작위로 분할
- 모델 훈련: 총 k 번 훈련
    - 매 훈련마나다 하나의 폴드를 선택하여 검증 데이터셋으로 지정
    - 나머지 (k-1) 개의 폴드를 대상으로 훈련
    - 매 훈련이 끝날 때마다 선택된 검증 데이터셋을 이용하여 모델 평가
    - 매번 다른 폴드 활용
- 최종평가: k-번 훈련 평가 결과의 평균값 활용

예제: 5-겹 교차 검증

사이킷런의 `cross_val_score` 함수: k-겹 교차 검증

- 예제: 결정 트리 모델 10-겹 교차 검증
- `scoring="neg_mean_squared_error"` 옵션
    - 훈련중인 모델의 성능을 측정하는 **효용함수** 지정
    - 모델의 성능 측정값은 높을 수록 좋은 성능으로 평가되기에 회귀 모델의 경우 일반적으로 RMSE의 음숫값을 사용함.

```
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)
```

모델별 교차 검증 결과

- 결정트리 회귀 모델 교차 검증의 RMSE: 평균적으로 약 66868 정도
- 선형회귀 모델 교차 검증의 RMSE: 평균적으로 약 69858 정도로 결정트리 회귀 모델과 비슷.
- 랜덤 포레스트 회귀 모델 교차 검증의 RMSE: 평균적으로 약 47019 정도로 가장 좋음.

2.8. 모델 미세 조정

- 가장 좋은 성능의 모델을 선정한 후에 모델의 세부 설정(하이퍼파라미터)을 조정해서 모델의 성능을 보다 끌어 올릴 수 있음

- 모델 미세 조정을 위한 세 가지 방식
    - 그리드 탐색
    - 랜덤 탐색
    - 앙상블 방법

하이퍼파라미터 vs. 파라미터

- 사이킷런 클래스의 **하이퍼파라미터**hyperparameter: 해당 클래스의 객체를 생성할 때 생성자 메서드의 인자로 사용되는 값들.
- **파라미터**parameter: `fit()` 메서드가 데이터셋으로부터 추출한 정보에 해당하는 값들.
- 사이킷런의 모든 클래스는 적절한 하이퍼파라미터로 초기화되어 있으며, 데이터 변환과 값 예측에 필요한 모든 파라미터를 효율적으로 관리함.

2.8.1. 그리드 탐색

- 지정한 하이퍼파라미터의 모든 조합을 교차검증하여 최적의 모델을 생성하는 하이퍼파라미터 조합 찾기
- 사이킷런의 `GridSearchCV` 활용

예제: 랜덤 포레스트와 그리드 탐색

- 총 (3x3 + 2x3 = 15) 가지의 경우 확인

```
(군집수 3 가지) * (최대특성수 3 가지) + (군집수 2 가지) * (최대특성수 3 가지)
```

- 3-겹 교차 검증을 진행하기에 총 (15x3 = 45)번 모델 훈련 진행.

```
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]},
]
```

그리드 탐색 실행

```
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.
    - `preprocessing__geo__n_clusters`: 15
    - `random_forest__max_features`: 6
- 최고 성능의 랜덤 포레스트 회귀 모델에 대한 교차 검증의 RMSE는 44042 정도로 이전보다 좀 더 좋아짐.

2.8.2. 랜덤 탐색

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

예제: 랜덤 포레스트와 랜덤 탐색

- `n_iter=10`: 랜덤 탐색이 총 10회 진행
    - `geo__n_clusters`와 `max_features` 값을 지정된 구간에서 무작위 선택
- `cv=3`: 3-겹 교차검증. 따라서 랜덤 포레스트 학습이 (10x3=30)번 이루어짐.

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

param_distribs = {'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_distribs, n_iter=10, cv=3,
    scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
```

랜덤 탐색 결과

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

2.8.3. 앙상블 학습

- 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 포레스트에서 자세히 다룰 것임.

2.8.4. 최적 모델 활용

- 모델 미세 조정을 통해 구한 최적의 모델을 분석해서 훈련에 사용된 데이터셋에 대한 중요한 통찰을 얻을 수 있음.

- 예를 들어, 최적의 랜덤 포레스트 모델로부터 모델의 예측값에 영향을 주는 특성들의 상대적 중요도를 확인할 수 있음.

- `feature_importances_` 특성 확인
    - `log__median_income` 특성이 해당 구역의 집값 예측이 가장 중요함.
    - 해안 근접도 특성 중에서 특히 `INLAND` 특성이 집값 예측에 나름 중요한 역할을 수행함.

특성 중요도

```
# 최적 모델
final_model = rnd_search.best_estimator_
# 특성 중요도
feature_importances = final_model["random_forest"].feature_importances_
# 특성 중요도 내림차순 정렬
sorted(zip(feature_importances,
           final_model["preprocessing"].get_feature_names_out()),
           reverse=True)
```

```
[(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

```
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)
print(final_rmse)
```

```
41424.40026462184
```

2.9. 최적 모델 저장과 활용

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

- 저장하기

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

- 불러오기

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

- 활용

```
final_model_reloaded.predict(X_test)
```