2장 머신러닝 프로젝트 처음부터 끝까지 (3부)

감사의 글

자료를 공개한 저자 오렐리앙 제롱과 강의자료를 지원한 한빛아카데미에게 진심어린 감사를 전합니다.

2.6 모델 선택과 훈련

- 목표 달성에 필요한 두 요소를 결정해야함
    - 학습 모델
    - 회귀 모델 성능 측정 지표

- 목표 달성에 필요한 두 요소를 결정해야함
    - 학습 모델
    - 회귀 모델 성능 측정 지표
- 목표: 구역별 중간 주택 가격 예측 모델

- 목표 달성에 필요한 두 요소를 결정해야함
    - 학습 모델
    - 회귀 모델 성능 측정 지표
- 목표: 구역별 중간 주택 가격 예측 모델
- 학습 모델: 회귀 모델
- 회귀 모델 성능 측정 지표: 평균 제곱근 오차(RMSE)를 기본으로 사용

2.6.1 훈련 세트에서 훈련하고 평가하기

- 지금까지 한 일
    - 훈련 세트 / 테스트 세트 구분
    - 변환 파이프라인을 활용한 데이터 전처리

2.6.1 훈련 세트에서 훈련하고 평가하기

- 지금까지 한 일
    - 훈련 세트 / 테스트 세트 구분
    - 변환 파이프라인을 활용한 데이터 전처리
- 이제 할 일
    - 예측기 모델 선택 후 훈련시키기
    - 예제: 선형 회귀, 결정트리 회귀

2.6.1 훈련 세트에서 훈련하고 평가하기

- 지금까지 한 일
    - 훈련 세트 / 테스트 세트 구분
    - 변환 파이프라인을 활용한 데이터 전처리
- 이제 할 일
    - 예측기 모델 선택 후 훈련시키기
    - 예제: 선형 회귀, 결정트리 회귀
- 예측기 모델 선택 후 훈련과정은 매우 단순함.
    - `fit()` 메서드를 전처리 처리가 된 훈련 데이터셋에 적용

선형 회귀 모델(4장)

- 선형 회귀 모델 생성: 사이킷런의 `LinearRegression` 클래스 활용

선형 회귀 모델(4장)

- 선형 회귀 모델 생성: 사이킷런의 **`LinearRegression`** 클래스 활용
- 훈련 및 예측
    - 훈련: `LinearRegression` 모델은 무어-펜로즈 역행렬을 이용하여 파라미터 직접 계산 (4장 참조)
    - 예측: (여기서는 연습 용도로) 훈련 세트에 포함된 몇 개 데이터를 대상으로 예측 실행

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

lin_reg.predict(housing_prepared)
```

선형 회귀 모델의 훈련 세트 대상 예측 성능

- RMSE(평균 제곱근 오차)가 68628.198 정도로 별로 좋지 않음.
- 훈련된 모델이 훈련 세트에 **과소적합** 됨.
    - 보다 좋은 특성을 찾거나 더 강력한 모델을 적용해야 함.
    - 보다 좋은 특성 예제: 로그 함수를 적용한 인구수 등
    - 모델에 사용되는 규제(regulaization, 4장)를 완화할 수도 있지만 위 모델에선 어떤 규제도 적용하지 않았음.

결정트리 회귀 모델(6장)

- 결정 트리 모델은 데이터에서 복잡한 비선형 관계를 학습할 때 사용
- 결정트리 회귀 모델 생성: 사이킷런의 `DecisionTreeRegressor` 클래스 활용

결정트리 회귀 모델(6장)

- 결정 트리 모델은 데이터에서 복잡한 비선형 관계를 학습할 때 사용
- 결정트리 회귀 모델 생성: 사이킷런의 `DecisionTreeRegressor` 클래스 활용
- 훈련 및 예측
    - 예측은 훈련 세트에 포함된 몇 개 데이터를 대상으로 예측 실행

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)

housing_predictions = tree_reg.predict(housing_prepared)
```

결정트리 회귀 모델의 훈련 세트 대상 예측 성능

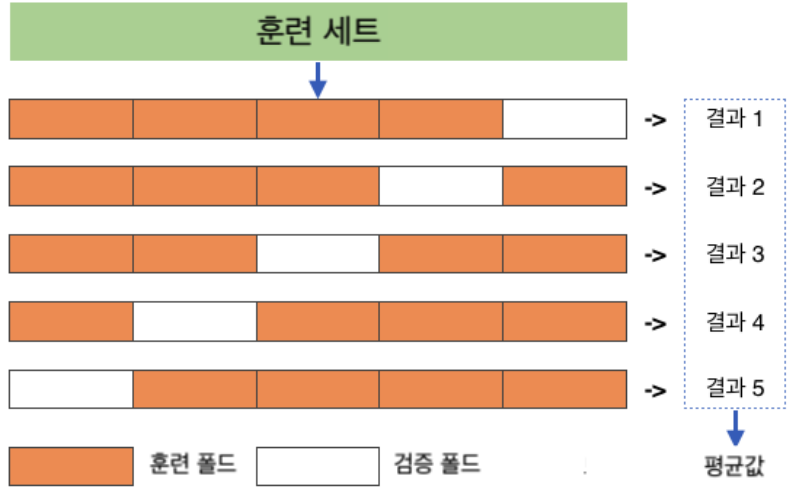
- RMSE(평균 제곱근 오차)가 0으로 완벽해 보임.
- 훈련된 모델이 훈련 세트에 심각하게 **과대적합** 됨.
    - 실전 상황에서 RMSE가 0이 되는 것은 불가능.
    - 훈련 세트가 아닌 테스트 세트에 적용할 경우 RMSE가 크게 나올 것임.

2.6.2 교차 검증을 사용한 평가

- 테스트 세트를 사용하지 않으면서 훈련 과정을 평가할 수 있음.
- **교차 검증** 활용

k-겹 교차 검증

- 훈련 세트를 **폴드**(fold)라 불리는 k-개의 부분 집합으로 무작위로 분할
- 총 k 번 지정된 모델을 훈련
    - 훈련할 때마다 매번 다른 하나의 폴드 선택하여 검증 데이터셋으로 활용
    - 다른 (k-1) 개의 폴드를 이용해 훈련
- 최종적으로 k 번의 평가 결과가 담긴 배열 생성
- k = 5인 경우



예제: 결정 트리 모델 교차 검증 (k = 10인 경우)

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)

tree_rmse_scores = np.sqrt(-scores)
```

예제: 결정 트리 모델 교차 검증 (k = 10인 경우)

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)

tree_rmse_scores = np.sqrt(-scores)
```

- k-겹 교차 검증의 모델 학습 과정에서 성능을 측정할 때 높을 수록 좋은 **효용함수** 활용
    - `scoring="neg_mean_squared_error"`
    - RMSE의 음숫값

예제: 결정 트리 모델 교차 검증 (k = 10인 경우)

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)

tree_rmse_scores = np.sqrt(-scores)
```

- k-겹 교차 검증의 모델 학습 과정에서 성능을 측정할 때 높을 수록 좋은 **효용함수** 활용
    - `scoring="neg_mean_squared_error"`
    - RMSE의 음숫값
- 교차 검증의 RMSE: 다시 음숫값(`-scores`) 사용
    - 평균 RMSE: 약 71407
    - 별로 좋지 않음.

예제: 선형 회귀 모델 교차 검증 (k = 10 인 경우)

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)

lin_rmse_scores = np.sqrt(-lin_scores)
```

- 교차 검증의 RMSE 평균: 약 69052
    - 결정트리 회귀 모델보다 좋음.

앙상블 학습 (7장)

- 여러 개의 다른 모델을 모아서 하나의 모델을 만드는 기법
- 머신러닝 알고리즘의 성능을 극대화는 방법 중 하나

랜덤 포레스트 회귀 모델 (7장)

- 앙상블 학습에 사용되는 하나의 기법
- 무작위로 선택한 특성을 이용하는 결정 트리 여러 개를 훈련 시킨 후 훈련된 모델들의 평균 예측값을 예측값으로 사용하는 모델

- 사이킷런의 `RandomForestRegressor` 클래스 활용
- 훈련 및 예측

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)

housing_predictions = forest_reg.predict(housing_prepared)
```

- 사이킷런의 `RandomForestRegressor` 클래스 활용
- 훈련 및 예측

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)

housing_predictions = forest_reg.predict(housing_prepared)
```

- 랜덤 포레스트 모델의 RMSE: 약 50182
    - 지금까지 사용해본 모델 중 최고
    - 하지만 여전히 과대적합되어 있음.

2.7 모델 세부 튜닝

- 살펴 본 모델 중에서 **랜덤 포레스트** 모델의 성능이 가장 좋았음
- 가능성이 높은 모델을 선정한 후에 **모델 세부 설정을 튜닝**해야함
- 튜닝을 위한 세 가지 방식
    - **그리드 탐색**
    - **랜덤 탐색**
    - **앙상블 방법**

2.7.1 그리드 탐색

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

예제: 그리드 탐색으로 랜덤 포레스트 모델에 대한 최적 조합 찾기

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

예제: 그리드 탐색으로 랜덤 포레스트 모델에 대한 최적 조합 찾기

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

- 총 (3x4 + 2x3 = 18) 가지의 경우 확인
- 5-겹 교차검증(`cv=5`)이므로, 총 (18x5 = 90)번 훈련함.

그리드 탐색 결과

- 최고 성능의 랜덤 포레스트 하이퍼파라미터가 다음과 같음.
    - `max_features` : 8
    - `n_estimators` : 30
    - 지정된 구간의 최고값들이기에 구간을 좀 더 넓히는 게 좋아 보임
- 최고 성능의 랜덤 포레스트에 대한 교차검증 RMSE: 49682
    - 하나의 랜덤 포레스트보다 좀 더 좋아졌음.

2.7.2 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 탐색으로 랜덤 포레스트 모델에 대한 최적 조합 찾기

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

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5,
scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

예제: 랜덤 탐색으로 랜덤 포레스트 모델에 대한 최적 조합 찾기

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

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5,
scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

- `n_iter=10` : 랜덤 탐색이 총 10회 진행
    - `n_estimators`와 `max_features` 값을 지정된 구간에서 무작위 선택
- `cv=5` : 5-겹 교차검증. 따라서 랜덤 포레스트 학습이 (10x5=50)번 이루어짐.

랜덤 탐색 결과

- 최고 성능의 랜덤 포레스트 하이퍼파라미터가 다음과 같음.
    - `max_features` : 7
    - `n_estimators` : 180
- 최고 성능의 랜덤 포레스트에 대한 교차검증 RMSE: 49150

2.7.3 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
- 7장에서 자세히 다룸

2.7.4 최상의 모델과 오차 분석

- 그리드 탐색과 랜덤 탐색 등을 통해 얻어진 최상의 모델을 분석해서 문제에 대한 좋은 통창을 얻을 수 있음

2.7.4 최상의 모델과 오차 분석

- 그리드 탐색과 랜덤 탐색 등을 통해 얻어진 최상의 모델을 분석해서 문제에 대한 좋은 통창을 얻을 수 있음
- 예를 들어, 최상의 랜덤 포레스트 모델에서 사용된 특성들의 중요도를 확인하여 일부 특성을 제외할 수 있음.
    - 중간 소득(median income)과 INLAND(내륙, 해안 근접도)가 가장 중요한 특성으로 확인됨
    - 해안 근접도의 다른 네 가지 특성은 별로 중요하지 않음
    - 중요도가 낮은 특성은 삭제할 수 있음.

```
[(0.36615898061813423, 'median_income'),
 (0.16478099356159054, 'INLAND'),
 (0.10879295677551575, 'pop_per_hhold'),
 (0.07334423551601243, 'longitude'),
 (0.06290907048262032, 'latitude'),
 (0.056419179181954014, 'rooms_per_hhold'),
 (0.053351077347675815, 'bedrooms_per_room'),
 (0.04114379847872964, 'housing_median_age'),
 (0.014874280890402769, 'population'),
 (0.014672685420543239, 'total_rooms'),
 (0.014257599323407808, 'households'),
 (0.014106483453584104, 'total_bedrooms'),
 (0.010311488326303788, '<1H OCEAN'),
 (0.002856474637320158, 'NEAR OCEAN'),
 (0.00196041559947807 06, 'NEAR BAY'),
 (6.0280386727366e-05, 'ISLAND')]
```

2.7.5 테스트 셋으로 시스템 평가하기

1. 최고 성능 모델 확인: 예를 들어, 그리드 탐색으로 찾은 최적 모델 사용

```
final_model = grid_search.best_estimator_
```

2.7.5 테스트 셋으로 시스템 평가하기

1. 최고 성능 모델 확인: 예를 들어, 그리드 탐색으로 찾은 최적 모델 사용

```
final_model = grid_search.best_estimator_
```

1. 테스트 세트 전처리
    - 전처리 파이프라인의 `transform()` 메서드를 직접 활용
    - **주의**: `fit()` 메서드는 전혀 사용하지 않음

2.7.5 테스트 셋으로 시스템 평가하기

1. 최고 성능 모델 확인: 예를 들어, 그리드 탐색으로 찾은 최적 모델 사용

```
final_model = grid_search.best_estimator_
```

1. 테스트 세트 전처리
    - 전처리 파이프라인의 `transform()` 메서드를 직접 활용
    - **주의**: `fit()` 메서드는 전혀 사용하지 않음

1. 최고 성능 모델을 이용하여 예측하기

2.7.5 테스트 셋으로 시스템 평가하기

1. 최고 성능 모델 확인: 예를 들어, 그리드 탐색으로 찾은 최적 모델 사용

```
final_model = grid_search.best_estimator_
```

1. 테스트 세트 전처리
    - 전처리 파이프라인의 `transform()` 메서드를 직접 활용
    - **주의**: `fit()` 메서드는 전혀 사용하지 않음

1. 최고 성능 모델을 이용하여 예측하기

1. 최고 성능 모델 평가 및 론칭

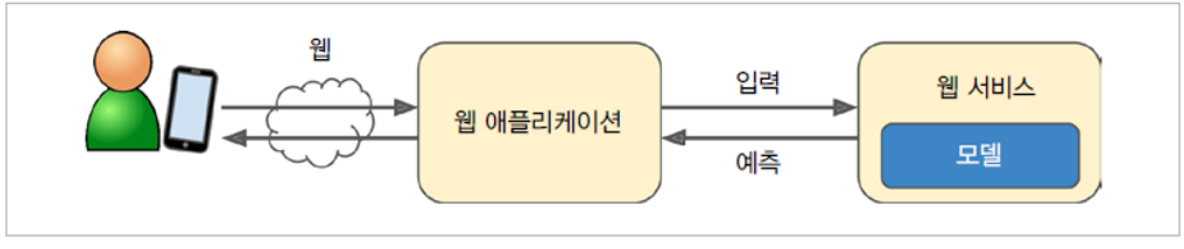
최상의 모델 성능 평가

- 테스트 세트에 대한 최고 성능 모델의 RMSE: 47730

최상의 모델 성능 평가

- 테스트 세트에 대한 최고 성능 모델의 RMSE: 47730

최상의 모델 성능 배포



데이터셋 및 모델 백업

- 완성된 모델은 항상 백업해 두어야 함. 업데이트된 모델이 적절하지 않은 경우 이전 모델로 되돌려야 할 수도 있음.
    - 백업된 모델과 새 모델을 쉽게 비교할 수 있음.
- 동일한 이유로 모든 버전의 데이터셋을 백업해 두어야 함.
    - 업데이트 과정에서 데이터셋이 오염될 수 있기 때문임.