한 눈에 보는 머신 러닝 2장

P 107 ~ p115

2.5.4 특성 스케일링 ~ 2.6.2 교차 검증을 사용한 평가

2.5.4 특성 스케일링

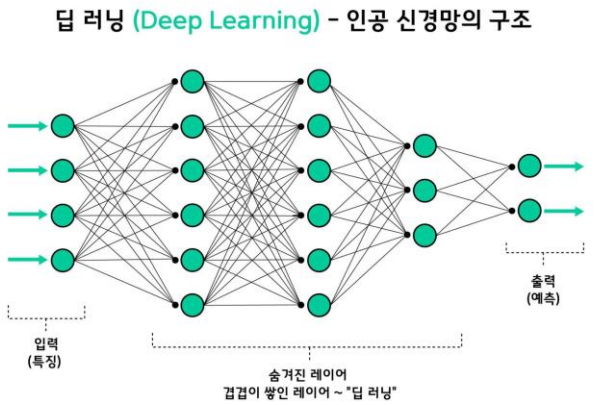
- 전체 방 개수의 범위 : 6 ~ 39,320 / 중간 소득의 범위 : 0 ~ 15
- 입력 숫자 특성들의 스케일이 많이 다름 -> 잘 작동하지 않음

- Min-max 스케일링, 표준화(standardization)

2.5.4 특성 스케일링

- Min-max 스케일링 (MinMAXScaler 변환기 사용)
  - 정규화(normalization)
  - 0~1 범위에 들도록 값을 이동하고 스케일 조정
  - 데이터 – 최솟값 / (최댓값 – 최솟값)

- 표준화(standardization) (StandardScaler 변환기 사용)
  - 평균을 먼저 뺌 (표준화를 하면 평균이 0이 됨)
    그 후 표준편차로 나누어 분포 분산이 1이 되도록 함.
  - 상한과 하한이 없어 어떤 알고리즘에서는 문제가 될 수 있음(ex. 신경망)
  - 이상치에 영향을 덜 받음



2.5.5 변환 파이프라인

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('attribs_adder',CombineAttributesAdder()),
        ('std_scaler',StandardScaler())
])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

✓ 0.0s

- Pipeline : 연속된 단계를 나타내는 이름/추정기 쌍의 목록 입력으로 받음
- 마지막 단계 : 변환기, 추정기 사용 가능, 그 외에는 변환기만 사용 가능
- StandardScaler(변환기) 사용 => 파이프라인 데이터에 대해 모든 변환을 순서대로 적용하는 transform() 메서드를 가지고 있음

2.5.5 변환 파이프라인

```python
from sklearn.compose import ColumnTransformer

#수치형 열 이름의 리스트, 범주형 열 이름 리스트 생성
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

#ColumnTransformer 클래스 객체 (변환기, 변환기가 적용될 열 이름의 리스트로 이루어짐)
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs), #수치형 열: num_pipeline사용하여 변환
    ("cat", OneHotEncoder(),cat_attribs), #범주형 열: OneHotEncoder 사용해 변환
])
#주택 데이터에 적용
housing_prepared = full_pipeline.fit_transform(housing)
```

- 하나의 변환기로 각 열마다 적절한 변환을 적용하여 모든 열 처리 -> ColumnTransformer
- OneHotEncoder : 희소행렬 반환, num_pipeline : 밀집행렬 반환
- ColumnTransfomer : 희소행렬, 밀집행렬 섞여있을 때 최종행렬의 밀집 정도 추정(0이 아닌 원소의 비율)

2.6.1 훈련 세트에서 훈련하고 평가하기

```
from sklearn.linear_model import LinearRegression

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

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

- 선형 회귀 모델 생성

```
[ ] # 연습 용도로 훈련 샘플 몇 개를 대상으로 예측 실행
    some_data = housing.iloc[:5]
    some_labels = housing_labels.iloc[:5]
    some_data_prepared = full_pipeline.transform(some_data)

    print("예측:", lin_reg.predict(some_data_prepared))
```

```
예측: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

- 예측

실제 중간 주택 가격은 다음과 같다.

```
[ ] print("레이블:", list(some_labels))
```

```
레이블: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

2.6.1 훈련 세트에서 훈련하고 평가하기

```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

```
68628.19819848923
```

평균 제곱근 오차(root mean square error, RMSE)

- 오차가 커질수록 값이 커짐 -> 예측에 얼마나 많은 오류가 있는지 가늠할 수 있음

RMSE를 측정할 데이터셋에 있는 샘플(구역) 수

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

- 기호 설명
  - $\mathbf{X}$: 훈련 데이터셋 전체 샘플들의 특성값들로 구성된 행렬, 레이블(타겟) 제외.
  - $\mathbf{x}^{(i)}$: i 번째 샘플의 전체 특성값 벡터. 레이블(타겟) 제외.
  - $y^{(i)}$: i 번째 샘플의 레이블
  - h: 예측 함수
  - $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$: i번째 샘플에 대한 예측 값

출처 : 경민이 ppt

- Mean_square_error 함수 사용하여 전체 훈련 세트에 대한 회귀모델의 RMSE 측정
- RMSE : 평균제곱근오차
- RMSE = $68628 , 대부분 중간 주택 가격 : $120,000~ $265,000 사이임에 비해 이 값은 만족스럽지 않음. -> 모델이 훈련 데이터에 과소 적합된 사례
- 특성들이 좋은 예측을 만들 만큼 충분한 정보 제공 x , 충분히 강력하지 x

2.6.1 훈련 세트에서 훈련하고 평가하기

- 과소 적합 해결하려면?
  - 더 강력한 모델 선택
  - 훈련 알고리즘에 더 좋은 특성 주입
  - 모델의 규제 감소

2.6.1 훈련 세트에서 훈련하고 평가하기

```
from sklearn.tree import DecisionTreeRegressor

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

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

- DecisionTreeRegressor를 훈련
  - 강력하고 데이터에서 복잡한 비선형 관계 찾을 수 있음

```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

```
0.0
```

- RMSE가 0이 나옴
  - 모델이 완벽하게 훈련세트에 적응
  - 과대적합이 너무 심하게 이루어짐

  -> 모델 성능 믿을 수 없다.

  훈련세트의 일부분으로 훈련하고 다른 일부분은 모델 검증에 사용해야한다.

2.6.2 교차 검증을 사용한 평가

```
[ ]  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)
```

- cv : 폴드 수, (10개)
- Scoring : 교차 검증에 사용되는 성능 평가 기준 지정. 여기서는 높을수록 좋은 효용함수 사용하므로 작을수록 좋은 비용함수의 음숫값을 사용한다.

- K-겹 교차 검증 (k-fold cross-validation) 사용
- 폴드(fold)라 불리는 10개의 서브셋으로 무작위 분할 후 다음 결정 트리 모델을 10번 훈련하고 평가
- 매번 다른 폴드 선택하고 평가에 사용, 나머지 9개 폴드는 훈련에 사용.
- 10개의 평가 점수가 담긴 배열이 결과가 된다.

2.6.2 교차 검증을 사용한 평가

```
[ ]  def display_scores(scores):
         print("점수:", scores)
         print("평균:", scores.mean())
         print("표준 편차:", scores.std())

     display_scores(tree_rmse_scores)
```

```
점수: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
평균: 71407.68766037929
표준 편차: 2439.4345041191004
```

- 결정 트리 결과

```
[ ]  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)
     display_scores(lin_rmse_scores)
```

```
점수: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
평균: 69052.46136345083
표준 편차: 2731.674001798344
```

- 선형 회귀 교차 검증결과

- 결정 트리 결과가 이전보다 좋지 않음. -> 앞선 결과가 완벽한 과대적합이었음.

2.6.2 교차 검증을 사용한 평가

```
[ ]  from sklearn.ensemble import RandomForestRegressor

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

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

- RandomForestRegressor 모델
- 특성을 무작위로 선택해서 많은 결정 트리를 만들고 예측을 평균 내는 방식으로 작동
- 앙상블 학습 : 여러 다른 모델을 모아서 하나의 모델로 만듬

-> 머신러닝 알고리즘 성능 극대화

2.6.2 교차 검증을 사용한 평가

```
[ ]  from sklearn.model_selection import cross_val_score

     forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                     scoring="neg_mean_squared_error", cv=10)
     forest_rmse_scores = np.sqrt(-forest_scores)
     display_scores(forest_rmse_scores)
```

```
점수: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
평균: 50182.303100336096
표준 편차: 2097.0810550985693
```

- 랜덤 포레스트를 대상으로 교차 검증 진행하면 선형회귀, 결정 트리 모델보다 성능이 더 좋게 나옴
- 하지만 훈련 세트에 대한 점수가 검증 세트에 대한 점수보다 훨씬 낮음. -> 여전히 훈련세트에 과대적합

2.6.2 교차 검증을 사용한 평가

- 과대 적합 해결하려면?
  - 모델을 더 간단히
  - 제한(규제)
  - 더 많은 훈련 데이터를 모으기