

# End-to-end Machine Learning project [2]

Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hankyong.ac.kr>

이 장에서는 부동산 회사에 최근에 고용된 데이터 과학자인 것처럼 가장하여 예제 프로젝트를 단계별로 진행해 본다.

수행할 주요 단계는 아래와 같다:

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.


# Prepare the Data for ML Algorithms

ML 알고리즘을 위한 데이터 준비

- 수작업 보다는 데이터 준비를 위한 함수를 작성하여 자동화하는 것이 좋다.

먼저 clean training set으로 변환

# drop labels for training set



```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

- `drop()`은 데이터의 사본을 생성하지만, `strat_train_set`에는 영향을 주지 않는다.

# Data Cleaning

대부분 ML 알고리즘 => missing 특성이 있는 경우 동작되지 않음 => 이를 다룰 수 있는 함수 필요

예: total\_bedrooms 속성 => 약간의 **missing** 값들을 갖는다. 수정해 보자.

- 3가지 옵션:

- ① 해당 블록(구역) 제거
- ② 속성 자체 제거
- ③ 어떤 값(zero, 평균, 메디안 등)으로 대체

=> DataFrame의 `dropna()`, `drop()`, `fillna()` 메소드로 쉽게 해결:

```
housing.dropna(subset=["total_bedrooms"])    # option 1
housing.drop("total_bedrooms", axis=1)       # option 2
median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

**Option 3:** training set에 대한 median 값 계산 필요

```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662	<1H OCEAN

**Scikit-Learn** => missing 값을 처리하는 편리한 클래스 제공: SimpleImputer

사용 방법: 먼저 각 속성의 missing 값을 해당 속성의 median으로 바꾸도록 지정하는 SimpleImputer 인스턴스를 생성해야 한다:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

Median은 수치 특성에서만 계산 가능하므로 텍스트 특성인 ocean\_proximity 제거:

```
housing_num = housing.drop("ocean_proximity", axis=1)

imputer.fit(housing_num)

>>> imputer.statistics_
array([ -118.51 ,  34.26 ,  29. , 2119.5 , 433. , 1164. , 408. ,  3.5409])
>>> housing_num.median().values
array([ -118.51 ,  34.26 ,  29. , 2119.5 , 433. , 1164. , 408. ,  3.5409])
```

새 데이터에 missing 값이 어느 시점에 생길지 모르므로 모든 수치 속성에 적용하는 것이 좋음

이제 "학습된" `imputer`를 이용하여 missing 값을 학습된 median으로 치환 가능:

```
X = imputer.transform(housing_num) ← 결과: NumPy 배열
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,  
                           index = list(housing.index.values)) ← 다시 원래대로 pandas DataFrame으로  
                                                                되돌리고 싶다면
```

```
housing_tr.loc[sample_incomplete_rows.index.values]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	4.6328
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662

# Handling Text and Categorical Attributes

지금까지 수치 속성만 다루었다. 이제 텍스트 속성을 다루보자: `ocean_proximity`

```
>>> housing_cat = housing["ocean_proximity"]
>>> housing_cat.head(10)
17606    <1H OCEAN
18632    <1H OCEAN
14650    NEAR OCEAN
3230     INLAND
3555     <1H OCEAN
19480     INLAND
8879     <1H OCEAN
13685     INLAND
4937     <1H OCEAN
4861     <1H OCEAN
Name: ocean_proximity, dtype: object
```

← Categorical 속성



그냥 임의의 텍스트가 아니다: => 각 값은 1개의 카테고리를 의미

- 대부분 ML 알고리즘 => 텍스트보다 숫자 선호 => 앞 카테고리를 텍스트에서 숫자로 변환해 보자!
- Scikit-Learn의 OrdinalEncoder 사용:

```
>>> from sklearn.preprocessing import OrdinalEncoder
>>> ordinal_encoder = OrdinalEncoder()
>>> housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
>>> housing_cat_encoded[:10]
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

카테고리 리스트 출력:

```
>>> ordinal_encoder.categories_  
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],  
      dtype=object)]
```

표현 방법의 문제점:

- ML 알고리즘 => 서로 가까이 있는 값들이 서로 멀리 떨어진 값들보다 더 유사하다고 가정
- 예: 카테고리 0과 2는 카테고리 0과 4보다 더 유사하다!!

이 문제를 해결하기 위한 일반적인 방법은 카테고리 당 1개의 이진 속성을 만드는 것이다!

카테고리가 "<1H OCEAN"인 경우의 속성을 1로 하고 나머지 다른 모든 속성들은 0으로 설정한다.

카테고리가 "NEAR OCEAN"인 경우의 속성을 1로 하고 나머지 다른 모든 속성들은 0으로 설정한다 등

- 하나의 속성만 1(hot)이 되고 다른 속성은 모두 0(cold)이 되기 때문에 이를 **one-hot 인코딩**이라고 한다.

Scikit-Learn은 카테고리형 값을 one-hot 벡터로 변환해주는 OneHotEncoder 인코더 제공:

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> cat_encoder = OneHotEncoder()
>>> housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
>>> housing_cat_1hot
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
  with 16512 stored elements in Compressed Sparse Row format>
```

⇒ "sparse matrix" (행 당 1의 값을 가진 경우가 1개만 있고 나머지는 모두 0의 값으로 채워져 있다.)

⇒ 심각한 낭비!!

⇒ 따라서 보통 1의 값을 가진 원소의 위치만 저장한다! (마치 2D array처럼 사용 가능)

그러나, 정말로 2D array로 변환하고 싶다면 단순히 `toarray()` 메소드 호출:

```
>>> housing_cat_lhot.toarray()
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  1.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.]])
```

다시 한번 인코더의 `categories_` 인스턴스 변수를 이용하여 카테고리 리스트 출력:

```
>>> cat_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
      dtype=object)]
```

# Custom Transformers

자신의 커스텀 transformer 작성 가능 => 속성들을 combine하는 오퍼레이션을 추가할 수 있다!

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6


class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kwargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                          bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attris = attr_adder.transform(housing.values)
```

클래스 생성 및 3가지 메소드 생성:

- `fit()` (self 반환)
- `transform()`
- `fit_transform()`

`fit_transform()`은 `TransformerMixin()`을 base 클래스로 추가하면 자동으로 정의됨

```
housing_extra_attribs = pd.DataFrame(housing_extra_attribs,
                                     columns=list(housing.columns)+["rooms_per_household", "population_per_household"])

housing_extra_attribs.head()
```

```
housing_extra_attribs = pd.DataFrame(housing_extra_attribs, columns=list(housing.columns)+["rooms_per_household", "population_per_household"])
housing_extra_attribs.head()
```

housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	rooms_per_household	population_per_household
38	1568	351	710	339	2.7042	<1H OCEAN	4.62537	2.0944
14	679	108	306	113	6.4214	<1H OCEAN	6.00885	2.70796
31	1952	471	936	462	2.8621	NEAR OCEAN	4.22511	2.02597
25	1847	371	1460	353	1.8839	INLAND	5.23229	4.13598
17	6592	1525	4459	1463	3.0347	<1H OCEAN	4.50581	3.04785

# Feature Scaling

대부분의 ML 알고리즘 => 입력된 수치 특성들의 스케일이 서로 크게 다를 때 잘 작동하지 않는다.

- 예: housing 데이터

Total number of rooms: *from about 6 to 39,320*

Median income: *from 0 to 15*

모든 특성을 동일한 스케일로 만드는 방법:

- Min-max scaling (called normalization)
- Standardization

### Min-max scaling (*normalization*)

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))  
X_scaled = X_std * (max - min) + min
```

Scikit-Learn은 이를 위해 `MinMaxScaler`라는 transformer를 제공한다.

### Standardization

$$Z = (X - \mu) / \sigma$$

Scikit-Learn은 표준화를 위해 `StandardScaler`라는 transformer를 제공한다.

#### WARNING

As with all the transformations, it is important to fit the scalers to the training data only, not to the full dataset (including the test set). Only then can you use them to transform the training set and the test set (and new data).



# Transformation Pipelines

올바른 순서대로 실행되어야 하는 많은 transformation 단계들이 있다.

- 다행히도 Scikit-Learn은 이런 일련의 변환을 돕기 위해 Pipeline 클래스 제공:

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

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

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

- 마지막을 제외한 모든 estimator는 반드시 transformer이어야 함
- 마지막 estimator는 반드시 `fit_transform()` 메소드를 가져야 함

지금까지는 카테고리 컬럼과 수치 컬럼을 별개로 취급

⇒ 그러나 모든 컬럼을 처리할 수 있는 단일 transformer를 사용하는 것이 더 편리

- Scikit-Learn(0.20 버전)은 이러한 목적으로 ColumnTransformer를 도입
- 좋은 소식은 Pandas DataFrames와 함께 잘 작동한다는 것
- 이를 사용하여 모든 변환을 housing 데이터에 적용해 보자:

```
from sklearn.compose import ColumnTransformer
```

```
num_attribs = list(housing_num)  
cat_attribs = ["ocean_proximity"]
```

```
full_pipeline = ColumnTransformer([  
    ("num", num_pipeline, num_attribs),  
    ("cat", OneHotEncoder(), cat_attribs),  
])
```

```
housing_prepared = full_pipeline.fit_transform(housing)
```

# Select and Train a Model

드디어 ML 알고리즘을 위한 데이터 준비를 완료했다!!

# Training and Evaluating on the Training Set

데이터 전처리 작업 덕분에 작업이 아주 수월해졌다!

Linear Regression 모델 학습:

```
from sklearn.linear_model import LinearRegression

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

Done!!

Linear Regression 모델 완성: Training set 이용 예측

```
>>> some_data = housing.iloc[:5]
>>> some_labels = housing_labels.iloc[:5]
>>> some_data_prepared = full_pipeline.transform(some_data)
>>> print("Predictions:", lin_reg.predict(some_data_prepared))
Predictions: [ 210644.6045  317768.8069  210956.4333  59218.9888  189747.5584]
>>> print("Labels:", list(some_labels))
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

- Scikit-Learn의 mean\_squared\_error 함수를 사용하여 전체 training set에서 회귀 모델의 RMSE 측정:

```
>>> 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
68628.198198489219
```

- 결과가 만족스럽지 못하다! => 더 복잡한 모델 사용

DecisionTreeRegressor 모델 학습:

```
from sklearn.tree import DecisionTreeRegressor

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

모델을 학습시켰다면 training set에 대해 평가해보자:

```
>>> 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
```

잠깐, What? No error at all?

# Better Evaluation Using Cross-Validation

Decision Tree 모델을 평가하기 위한 한 가지 방법: Scikit-Learn의 K-fold 교차 검증(K-fold cross-validation) 사용

예 : Training set를 fold라고 하는 10개의 서로 다른 서브셋으로 무작위로 분할한 다음 의사 결정 트리 모델을 10회 학습시키고 평가한다.

평가를 위해 매번 다른 fold를 선택하고 나머지 9개의 fold에 대해 학습시킨다.

결과는 10개의 평가 점수를 포함하는 배열:

```
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)
```

Scikit-Learn의 cross-validation scoring 함수 : cost 함수가 아니라 fitness 함수(greater is better)를 계산

⇒ Scoring 함수: MSE의 반대(음수 값)

⇒ **-scores**인 이유

Decision Tree 모델 결과:

```
>>> def display_scores(scores):  
...     print("Scores:", scores)  
...     print("Mean:", scores.mean())  
...     print("Standard deviation:", scores.std())  
...  
>>> display_scores(tree_rmse_scores)  
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782  
71115.88230639 75585.14172901 70262.86139133 70273.6325285  
75366.87952553 71231.65726027]  
Mean: 71407.68766037929  
Standard deviation: 2439.4345041191004
```



Linear Regression 모델에 대해서도 CV 적용:

```
>>> 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)
Scores: [66782.73843989 66960.118071    70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798348
```

Decision Tree 모델 결과가 Linear Regression 보다 더 나빠 보인다.

이유: Decision Tree 모델 => Overfitting!!

마지막 모델 => **RandomForestRegressor**

- RandomForest는 입력 특성들의 랜덤 서브셋에 대해 많은 Decision Tree를 학습시킨 다음 예측을 평균화하는 방식으로 작동
- 이처럼 다른 많은 모델 위에 모델을 구축하는 것을 **Ensemble Learning**이라고 하며 ML 알고리즘의 성능을 더욱 개선시키는 좋은 방법:

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> forest_reg = RandomForestRegressor()
>>> forest_reg.fit(housing_prepared, housing_labels)
>>> [...]
>>> forest_rmse
18603.515021376355
>>> display_scores(forest_rmse_scores)
Scores: [49519.80364233 47461.9115823 50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574 47585.73832311
 53490.10699751 50021.5852922 ]
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

- 결과가 훨씬 더 좋다!

# Fine-Tune Your Model

이제 모델을 미세 조정해 보자.

이를 수행할 수 있는 몇 가지 방법이 있다.

# Grid Search

방법 1:

- 최상의 hyperparameter 값의 조합을 찾을 때까지 hyperparameter를 수동으로 조정하는 방법
- 매우 지루한 작업, 많은 조합을 탐색하는데 오랜 시간 걸림

방법 2:

- Scikit-Learn의 **GridSearchCV**가 자동으로 찾도록 설정해 주는 방법
- Hyperparameter와 시도할 값을 알려주면 GridSearchCV가 CV를 사용하여 hyperparameter 값의 모든 가능한 조합을 평가

예를 들어, 다음 코드는 `RandomForestRegressor`에 대한 hyperparameter 값의 최상의 조합을 검색:

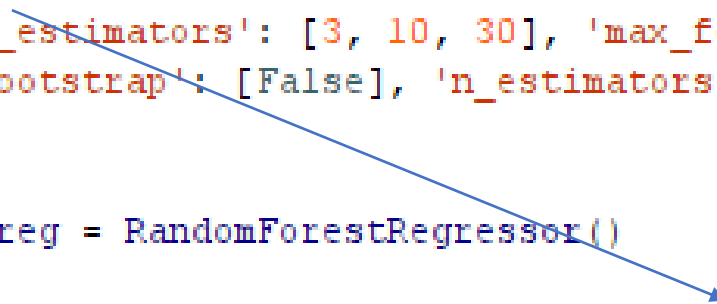
```
from sklearn.model_selection import GridSearchCV

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()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error')

grid_search.fit(housing_prepared, housing_labels)
```



param\_grid:

⇒ Scikit-Learn에 먼저 첫 번째 dict에 지정된 n\_estimators와 max\_features hyperparameter 값의  $3 \times 4 = 12$  조합을 모두 평가한 다음, 두 번째 dict에서 hyperparameter 값의  $2 \times 3 = 6$  조합을 모두 시도하도록 한다.

GridSearchCV가 완료된 후 발견된 best hyperparameter 조합:

```
>>> grid_search.best_params_  
{'max_features': 8, 'n_estimators': 30}
```

Best estimator를 직접 볼 수도 있다:

```
>>> grid_search.best_estimator_  
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,  
                        max_features=8, 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,  
                        n_estimators=30, n_jobs=None, oob_score=False, random_state=None,  
                        verbose=0, warm_start=False)
```

평가 점수:

```
>>> cvres = grid_search.cv_results_  
>>> for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):  
...     print(np.sqrt(-mean_score), params)  
...  
63669.05791727153 {'max_features': 2, 'n_estimators': 3}  
55627.16171305252 {'max_features': 2, 'n_estimators': 10}  
53384.57867637289 {'max_features': 2, 'n_estimators': 30}  
60965.99185930139 {'max_features': 4, 'n_estimators': 3}  
52740.98248528835 {'max_features': 4, 'n_estimators': 10}  
50377.344409590376 {'max_features': 4, 'n_estimators': 30}  
58663.84733372485 {'max_features': 6, 'n_estimators': 3}  
52006.15355973719 {'max_features': 6, 'n_estimators': 10}  
50146.465964159885 {'max_features': 6, 'n_estimators': 30}  
57869.25504027614 {'max_features': 8, 'n_estimators': 3}  
51711.09443660957 {'max_features': 8, 'n_estimators': 10}  
49682.25345942335 {'max_features': 8, 'n_estimators': 30}  
62895.088889905004 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}  
54658.14484390074 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}  
59470.399594730654 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}  
52725.01091081235 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}  
57490.612956065226 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}  
51009.51445842374 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

# Randomized Search

Grid search 방법

- 적은 개수의 조합이 있을 때는 문제 없음
- 탐색 공간이 클 때 => **RandomizedSearchCV** 사용
- RandomizedSearchCV 클래스
  - ✓ GridSearchCV와 동일한 방식으로 사용됨
  - ✓ 단, 모든 가능한 조합이 시도되는 것이 아니라 랜덤하게 hyperparameter를 선택함



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

param_distributions = {
    '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_distributions,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)

RandomizedSearchCV(cv=5, error_score='raise-deprecating',
    estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
    max_features='auto', 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, n_estimators='warn', n_jobs=None,
    oob_score=False, random_state=42, verbose=0, warm_start=False),
    fit_params=None, iid='warn', n_iter=10, n_jobs=None,
    param_distributions={'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x000000
<scipy.stats._distn_infrastructure.rv_frozen object at 0x0000001F1F3ED5DA0>},
    pre_dispatch='2*n_jobs', random_state=42, refit=True,
    return_train_score='warn', scoring='neg_mean_squared_error',
    verbose=0)
```

```
cvres = rnd_search.cv_results_  
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):  
    print(np.sqrt(-mean_score), params)
```

```
49150.657232934034 {'max_features': 7, 'n_estimators': 180}  
51389.85295710133 {'max_features': 5, 'n_estimators': 15}  
50796.12045980556 {'max_features': 3, 'n_estimators': 72}  
50835.09932039744 {'max_features': 5, 'n_estimators': 21}  
49280.90117886215 {'max_features': 7, 'n_estimators': 122}  
50774.86679035961 {'max_features': 3, 'n_estimators': 75}  
50682.75001237282 {'max_features': 3, 'n_estimators': 88}  
49608.94061293652 {'max_features': 5, 'n_estimators': 100}  
50473.57642831875 {'max_features': 3, 'n_estimators': 150}  
64429.763804893395 {'max_features': 5, 'n_estimators': 2}
```

# Analyze the Best Models and Their Errors

Best 모델을 검사하여 문제에 대한 통찰력을 얻을 수 있다!

(예) RandomForestRegressor는 정확한 예측을 하기 위해 각 속성의 상대적 중요성을 나타낼 수 있다.

```
>>> feature_importances = grid_search.best_estimator_.feature_importances_  
>>> feature_importances  
array([7.33442355e-02, 6.29090705e-02, 4.11437985e-02, 1.46726854e-02,  
       1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,  
       5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,  
       1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03])
```

해당 속성 이름 옆에 중요도 점수 표시:

```
>>> extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
>>> cat_encoder = full_pipeline.named_transformers_["cat"]
>>> cat_one_hot_attribs = list(cat_encoder.categories_[0])
>>> attributes = num_attribs + extra_attribs + cat_one_hot_attribs
>>> sorted(zip(feature_importances, attributes), reverse=True)
[(0.3661589806181342, 'median_income'),
 (0.1647809935615905, 'INLAND'),
 (0.10879295677551573, 'pop_per_hhold'),
 (0.07334423551601242, 'longitude'),
 (0.0629090704826203, 'latitude'),
 (0.05641917918195401, 'rooms_per_hhold'),
 (0.05335107734767581, 'bedrooms_per_room'),
 (0.041143798478729635, 'housing_median_age'),
 (0.014874280890402767, 'population'),
 (0.014672685420543237, 'total_rooms'),
 (0.014257599323407807, 'households'),
 (0.014106483453584102, 'total_bedrooms'),
 (0.010311488326303787, '<1H OCEAN'),
 (0.002856474637320158, 'NEAR OCEAN'),
 (0.00196041559947807, 'NEAR BAY'),
 (6.028038672736599e-05, 'ISLAND')]
```

# Evaluate Your System on the Test Set

모델을 잘 조율하면 충분히 좋은 시스템을 얻을 수 있다!

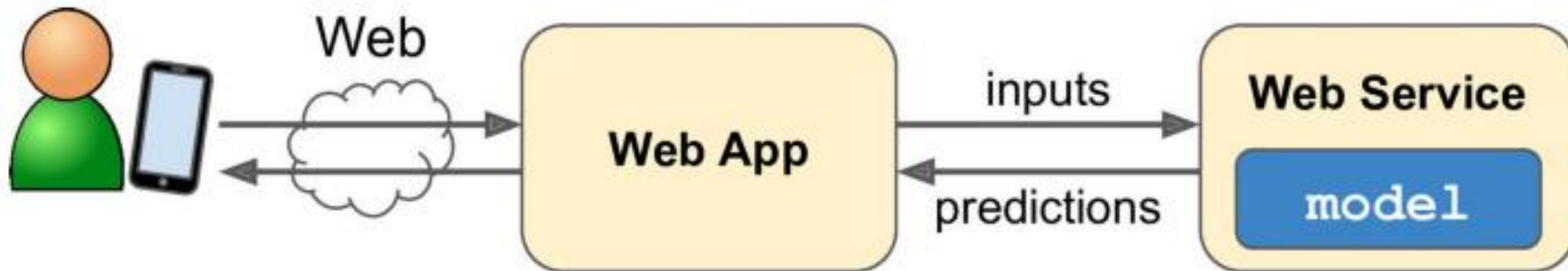
이제 테스트셋에서 최종 모델을 평가할 차례이다. 이 과정에는 특별한 것이 없다:

- 테스트셋에 대해 full\_pipeline을 실행하여 데이터를 변환하고
- 변환된 테스트셋에 대해 최종 모델을 평가:

```
final_model = grid_search.best_estimator_  
  
X_test = strat_test_set.drop("median_house_value", axis=1)  
y_test = strat_test_set["median_house_value"].copy()  
  
X_test_prepared = full_pipeline.transform(X_test)  
  
final_predictions = final_model.predict(X_test_prepared)  
  
final_mse = mean_squared_error(y_test, final_predictions)  
final_rmse = np.sqrt(final_mse)    # => evaluates to 47,730.2
```

# Launch, Monitor, and Maintain Your System

A model deployed as a web service and used by a web application



# 실습과제 3-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

참고: [제 03강 실습과제 #3 End-to-End Machine Learning Project \[2\].pdf](#)

