ch02 앙상블 기법- RandomForest(3)

학습 내용

01. 랜덤 포레스트의 배경와 원리에 대해 알아본다.

02. 랜덤 포레스트의 파라미터에 대해 알아본다.

03. 트리에서 사용하는 변수의 중요도에 대해서 체크해보고 알아본다.

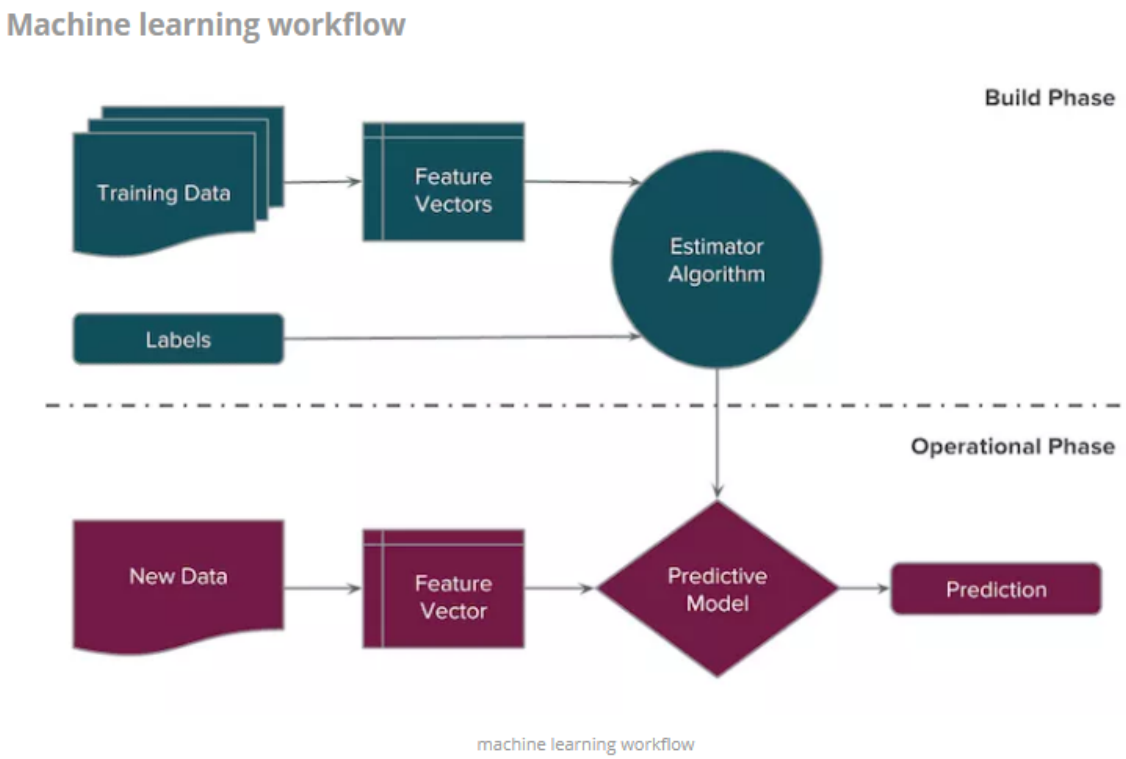
04. 의사결정트리와 선형회귀를 그래프를 통해 확인하고 알아본다.

In [1]:

```
from IPython.display import display, Image
```

In [2]:

```
## 머신러닝 작업 flow
display(Image(filename='img/machineWorkflow01.png'))
```



machine learning workflow

http://dataaspirant.com/2017/06/26/random-forest-classifier-python-scikit-learn/ (http://dataaspirant.com/2017/06/26/random-forest-classifier-python-scikit-learn/) 참조

01 앙상블(Ensemble) 기법

앙상블(ensemble)란?

- 여러 **머신러닝 모델을 연결**하여 더 강력한 모델을 만드는 기법이다.

  대표적인 앙상블 기법인
  랜덤 포레스트(random forest)와 그래디언트 부스팅(gradient boosting)결정 트리는
  모델을 구성하는 기본 요소로 결정 트리를 사용한다.

의사결정트리는 과적합되는 현상이 발생한다. 어떻게 해야 할까?

가. 랜덤 포레스트(random forest)

배경 : 의사 결정 트리는 학습용 데이터에 **과대적합**되는 경향이 있다.
원리 : 조금씩 다른 여러 결정 트리의 묶음.
트리를 많이 만들고 각각의 모델에 대한 결과의 평균값을 구하면 일반화가 되어, 과대적합(Overfitting)을 줄일 수 있지 않을까?

이렇게 하면 트리 모델의 예측 성능이 유지되면서 과대적합이 줄어든다는 것이 수학적으로 증명됨.

아이디어

(1) 만들어지는 각각의 트리는 타깃 예측을 잘해야 한다. 그리고 다른 트리와 구별(조금 달라야)되어야 한다.
(2) 랜덤 포레스트는 각각의 트리 성격이 달라지도록 트리 생성 시에
데이터 샘플링의 무작위성을 주입한다.
(3) 트리를 랜덤하게 만드는 방법은 2가지
----- 데이터 포인터를 무작위로 선택하는 방법
----- 분할 테스트(노드 데이터 조건) feature(특성)을 무작위로 선택

나. 랜덤 포레스트 구축

(1) 생성할 트리의 개수(n_estimators)를 선택한다.
(2) 부트 스트랩 샘플(bootstrap sample)을 생성
    --- n_samples개의 데이터 포인터 중에서
    --- 무작위로 데이터 n_samples 횟수만큼 반복 추출.(중복 추출 될 수 있음)
    * 중복 추출로 인해(대략 1/3정도) 누락될 수 있다. 또는 중복된 데이터가 있을 수 있다.

(3) 생성된 데이터 셋으로 결정 트리를 만든다.
단, 여기서 특성(feature)는 무작위로 선택된 특성 중에서 최선의 테스트(조건)을 고른다.
(max_features 매개변수로 몇 개의 feature를 고를지는 선택이 가능하다.)
    ==> max_feature 값을 크게 하면 랜덤 포레스트의 트리들은 매우 비슷해지고, 가장 두드러진 특성을 이용해 데이터에 잘 맞춰진다.
    ==> max_feature를 낮추면 랜덤 포레스트 트리들은 많이 달라지고 각 트리는 데이터에 맞추기 위해 깊이가 깊어진다.

-----> 결과적으로 부트스트랩 샘플링은 랜덤 포레스트의 트리가
       조금씩 다른 데이터셋을 이용해 만들어지도록 한다.
-----> 각각의 트리는 전체 특성(feature)의 일부만을 사용한다.

다. 랜덤 포레스트

- 집값 데이터를 활용한 랜덤 포레스트 분석

5개의 랜덤 포레스트 모델을 생성

In [3]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
```

캐글 코리아 2차 대회 데이터 셋 데이터

- https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr/data (https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr/data)

ID : 집을 구분하는 번호
date : 집을 구매한 날짜
price : 집의 가격(Target variable)
bedrooms : 침실의 수
bathrooms : 화장실의 수
sqft_living : 주거 공간의 평방 피트(면적)
sqft_lot : 부지의 평방 피트(면적)
floors : 집의 층 수
waterfront : 집의 전방에 강이 흐르는지 유무 (a.k.a. 리버뷰)
view : 집이 얼마나 좋아 보이는지의 정도
condition : 집의 전반적인 상태
grade : King County grading 시스템 기준으로 매긴 집의 등급
sqft_above : 지하실을 제외한 평방 피트(면적)
sqft_basement : 지하실의 평방 피트(면적)
yr_built : 지어진 년도
yr_renovated : 집을 재건축한 년도
zipcode : 우편번호
lat : 위도
long : 경도
sqft_living15 : 2015년 기준 주거 공간의 평방 피트(면적, 집을 재건축했다면, 변화가 있을 수 있음)
sqft_lot15 : 2015년 기준 부지의 평방 피트(면적, 집을 재건축했다면, 변화가 있을 수 있음)

In [4]:

```
import pandas as pd

train = pd.read_csv("house_train.csv")
test = pd.read_csv("house_test.csv")
```

In [5]:

```
train.columns
```

Out[5]:

```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

- 예측하고자 하는 값(target)이 price

In [6]:

```
X_all = train.drop(['price'], axis=1)
y = train['price']

print(type(X_all), type(y), X_all.shape, y.shape)
```

```
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'> (15035, 2
0) (15035,)
```

In [7]:

```
X_all.columns
```

Out[7]:

```
Index(['id', 'date', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [8]:

```
X_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15035 entries, 0 to 15034
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             15035 non-null  int64
 1   date           15035 non-null  object
 2   bedrooms       15035 non-null  int64
 3   bathrooms      15035 non-null  float64
 4   sqft_living    15035 non-null  int64
 5   sqft_lot       15035 non-null  int64
 6   floors         15035 non-null  float64
 7   waterfront     15035 non-null  int64
 8   view           15035 non-null  int64
 9   condition      15035 non-null  int64
 10  grade          15035 non-null  int64
 11  sqft_above     15035 non-null  int64
 12  sqft_basement  15035 non-null  int64
 13  yr_built       15035 non-null  int64
 14  yr_renovated   15035 non-null  int64
 15  zipcode        15035 non-null  int64
 16  lat            15035 non-null  float64
 17  long           15035 non-null  float64
 18  sqft_living15  15035 non-null  int64
 19  sqft_lot15     15035 non-null  int64
dtypes: float64(4), int64(15), object(1)
memory usage: 2.3+ MB
```

In [9]:

```
train.corr()
```

Out[9]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	w
id	1.000000	0.020899	0.010520	0.104030	0.041725	-0.034077	0.182848	-
price	0.020899	1.000000	0.323672	0.525479	0.702899	0.096793	0.262588	
bedrooms	0.010520	0.323672	1.000000	0.530548	0.596974	0.033475	0.189532	-
bathrooms	0.104030	0.525479	0.530548	1.000000	0.755853	0.089308	0.508649	
sqft_living	0.041725	0.702899	0.596974	0.755853	1.000000	0.176500	0.363193	
sqft_lot	-0.034077	0.096793	0.033475	0.089308	0.176500	1.000000	0.001535	
floors	0.182848	0.262588	0.189532	0.508649	0.363193	0.001535	1.000000	
waterfront	-0.011775	0.265738	-0.004819	0.075452	0.108137	0.025584	0.031159	
view	-0.024360	0.400806	0.085703	0.187488	0.282821	0.080441	0.034511	
condition	-0.101618	0.039740	0.034885	-0.125907	-0.054213	-0.002099	-0.261016	
grade	0.078622	0.667211	0.375286	0.666278	0.762543	0.119906	0.462598	
sqft_above	0.073086	0.608577	0.494867	0.688255	0.878736	0.186242	0.529476	
sqft_basement	-0.050634	0.322218	0.315183	0.282642	0.434017	0.017818	-0.239350	
yr_built	0.202477	0.047290	0.158799	0.503964	0.315927	0.058686	0.490436	-
yr_renovated	-0.029810	0.140808	0.022729	0.065423	0.064893	-0.001451	0.009752	
zipcode	-0.005761	-0.051498	-0.162081	-0.207500	-0.200745	-0.127709	-0.059107	
lat	0.002588	0.301604	-0.011190	0.018110	0.051609	-0.082234	0.049004	-
long	0.014757	0.023547	0.135802	0.227669	0.245429	0.227451	0.126983	-
sqft_living15	0.029248	0.586419	0.407394	0.573541	0.760271	0.147562	0.287125	
sqft_lot15	-0.032269	0.086384	0.027242	0.088120	0.184176	0.728458	-0.010287	

In [10]:

```
from sklearn.preprocessing import MinMaxScaler
```

In [11]:

```
sel = ['sqft_living', 'sqft_lot', 'bedrooms']  # 'bedrooms' , 'bathrooms',
X = X_all[sel]
y = train['price']

nor_X = MinMaxScaler().fit_transform(X)  # 입력 데이터 정규화
print("정규화 : ", nor_X.shape, y.shape)
```

```
정규화 :  (15035, 3) (15035,)
```

In [12]:

```
#  정규화 데이터 사용
X_train, X_test, y_train, y_test = train_test_split(nor_X, y,
                                                    random_state=42)

# 정규화 데이터 사용 안함.
# X_train, X_test, y_train, y_test = train_test_split(X, y,
#                                                     random_state=42)
```

In [13]:

```
model = RandomForestRegressor(n_estimators=5, random_state=2)  # 5개의 트리
print( model.fit(X_train, y_train) )
print( model.score(X_train, y_train))
print( model.score(X_test, y_test))
```

```
RandomForestRegressor(n_estimators=5, random_state=2)
0.8922137121180739
0.37937640288308927
```

In [14]:

```
# 학습된 랜덤포레스트의 트리 모델
print(model.estimators_)
print(model.score)
print("WnWn{}".format(model.base_estimator))
```

```
[DecisionTreeRegressor(max_features='auto', random_state=1872583848), DecisionTreeRe
gressor(max_features='auto', random_state=794921487), DecisionTreeRegressor(max_feat
ures='auto', random_state=111352301), DecisionTreeRegressor(max_features='auto', ran
dom_state=1853453896), DecisionTreeRegressor(max_features='auto', random_state=21329
8710)]
<bound method RegressorMixin.score of RandomForestRegressor(n_estimators=5, random_s
tate=2)>


DecisionTreeRegressor()
```

In [15]:

```
print("부트스트랩 : {}".format(model.bootstrap))
print("노드 분할 기준 : {}".format(model.criterion))
```

```
부트스트랩 : True
노드 분할 기준 : mse
```

In [16]:

```
model.feature_importances_
```

Out[16]:

```
array([0.69818654, 0.25989234, 0.04192112])
```

In [17]:

```
model.n_features_
```

Out[17]:

```
3
```

In [18]:

```
n_features = X.shape[1]
n_features
```

Out[18]:

```
3
```

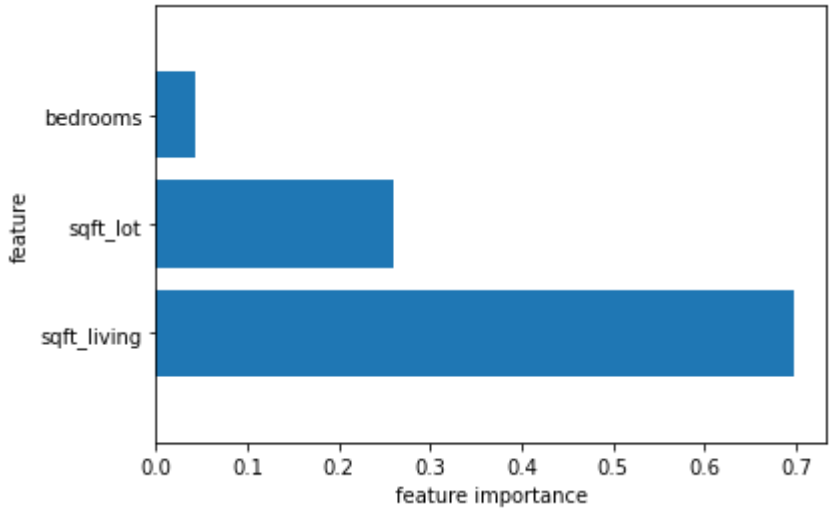
In [19]:

```
# model : 모델
# n_features : feature(변수의 개수)
# feature_names : 특성의 이름
def plot_feature_important_up(model, n_features, feature_names):
  imp = model.feature_importances_     # feature의 중요도
  plt.barh(range(n_features) , imp, align='center')  # 그래프(가로 막대 그래프)
  plt.yticks(np.arange(n_features), feature_names)  #y축의 축의 값
  plt.xlabel("feature importance")   # x축 레이블(제목)
  plt.ylabel("feature")              # y축 제목
  plt.ylim(-1, n_features)           # y축의 범위 지정
```

In [20]:

```
feature_names = sel

plot_feature_important_up(model, n_features, feature_names)
```



5개의 트리

- 다섯 개의 트리가 만든 결정 경계는 확연하게 다르다
- 랜덤 포레스트는 개개의 트리보다는 덜 과대적합된다.
- 실제로는 매우 많은 트리(수백, 수천개)를 사용하여 더 부드러운 경계가 만들어짐.

In [21]:

```
%%time
X_train, X_test, y_train, y_test = train_test_split(nor_X, y, random_state=42)
model_5 = RandomForestRegressor(n_estimators=5, random_state=2)  # 5개의 트리
model_5.fit(X_train, y_train)
```

```
Wall time: 166 ms
```

Out[21]:

```
RandomForestRegressor(n_estimators=5, random_state=2)
```

In [22]:

```
model_5
```

Out[22]:

```
RandomForestRegressor(n_estimators=5, random_state=2)
```

In [23]:

```
print( model_5.score(X_train, y_train))
print( model_5.score(X_test, y_test))
```

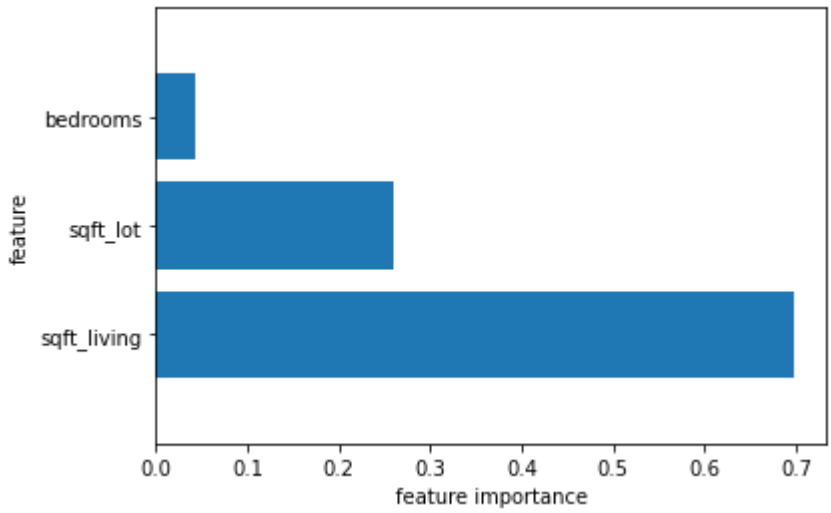
```
0.8922137121180739
0.37937640288308927
```

In [24]:

```
n_features = X.shape[1]
```

In [25]:

```
plot_feature_important_up(model, n_features, feature_names)
```



In [26]:

```
%%time
model_100 = RandomForestRegressor(n_estimators=100, random_state=2)  # 100개의 트리
model_100.fit(X_train, y_train)
print( model_100.score(X_train, y_train))
print( model_100.score(X_test, y_test))
```

```
0.9341193749716377
0.45105945624970734
Wall time: 3.14 s
```

In [27]:

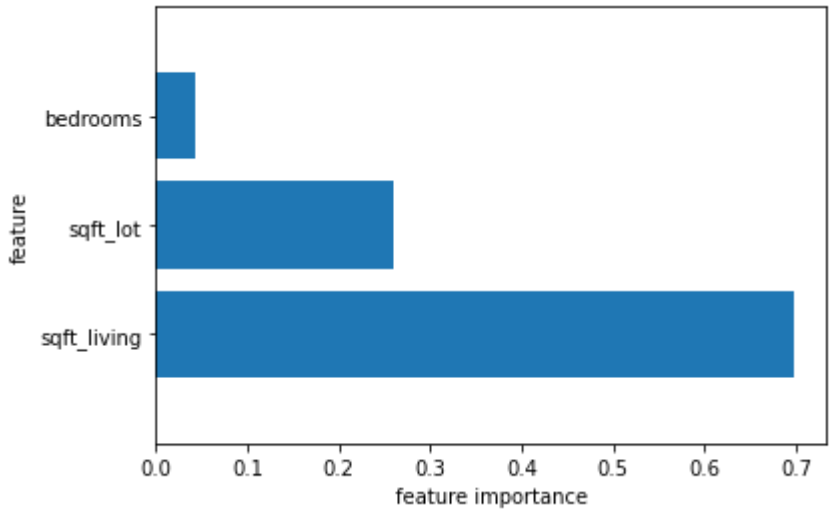
```
n_features = X_train.shape[1]
n_features
```

Out[27]:

3

In [28]:

```
plot_feature_important_up(model, n_features, feature_names)
```



5개의 모델에 대한 정확도 평가

In [29]:

```
for model in model_5.estimators_:
    model.fit(X_train, y_train)
    print("훈련 세트 정확도 : {:.3f}".format(model.score(X_train, y_train)))
    print("테스트 세트 정확도 : {:.3f}".format(model.score(X_test, y_test)))
```

```
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.074
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.056
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.087
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.059
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.069
```

100개 모델에 대한 정확도 평가

In [30]:

```
cnt = 1
for model in model_100.estimators_:
    model.fit(X_train, y_train)
    if cnt % 10 == 0:
        print("훈련 세트 정확도 : {:.3f}".format(model.score(X_train, y_train)))
        print("테스트 세트 정확도 : {:.3f}".format(model.score(X_test, y_test)))
    cnt += + 1
```

```
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.039
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.071
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.090
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.050
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.044
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.092
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.066
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.074
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.058
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.061
```

실습

- 여러가지 특성을 넣어 성능을 개선시켜보자.

Conclusion

(가) 회귀와 분류에 있어, 랜덤 포레스트는 현재 가장 널리 사용되는 머신러닝 알고리즘이다.
(나) 랜덤 포레스트는 성능이 매우 뛰어나고 매개변수 튜닝을 많이 하지 않아도 잘 작동한다.
(다) 만약 의사결정을 간소하게 해야 한다면 단일 트리를 사용할 수 있다.
(라) 랜덤 포레스트는 n_jobs를 이용하여 여러개의 코어를 이용하여 병렬 처리를 통해 속도 향상을 시킬 수 있다.
(마) 랜덤 포레스트는 트리가 많을 수록 random_state 값의 변화에 따른 변동이 적다.
(바) n_estimators는 클수록 좋다. (더 많은 트리는 과대 적합을 줄여준다.) 다만, 시간과 메모리의 문제가 발생
(사) max_features는 일반적으로 기본값을 쓰는 것이 좋다.
    * 분류는 max_features=sqrt(n_features), 회귀는 max_features=n_features

실습해 보기 3

데이터 셋 : 유방암 데이터 셋
```
from sklearn.datasets import load_breast_cancer
```

위의 데이터 셋을 이용하여 모델을 만들어보자.
(1) 랜덤 포레스트를 이용하여 훈련 세트 정확도, 테스트 세트 정확도를 확인해 보자.
(랜덤 포레스트 트리의 개수 = 5개, random_state=0, 최대 변수 선택 = 4)

도전 실습

- 2차 캐글 대회의 데이터 셋을 활용하여 랜덤 포레스트 모델을 구해보자.
- url : https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr의 (https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr%EC%9D%98) 대회
- 여러가지 변수 파라미터를 조절하여 RMSE가 가장 좋을때가 언제인가?