Go-to Guide for Regression

November 18, 2019

[1]:
```
%%html
<style>
.container{width: 100%}
</style>
```

```
<IPython.core.display.HTML object>
```

[2]:
```
%load_ext autoreload
%autoreload 2
```

[3]:
```
import warnings
warnings.filterwarnings("ignore")
```

[4]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

[5]:
```
import os
os.sys.path.insert(0, "../")
```

0.0.1 Load Data

[6]:
```
from tools import load_boston
data, desc = load_boston("../data_base")
data = data.rename(columns = {"target": "MEDV"})
features = data.drop("MEDV", axis = 1)
prices = data.MEDV
```

[7]:
```
print(desc)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------
```

Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM   per capita crime rate by town
        - ZN   proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS   proportion of non-retail business acres per town
        - CHAS   Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX   nitric oxides concentration (parts per 10 million)
        - RM   average number of rooms per dwelling
        - AGE   proportion of owner-occupied units built prior to 1940
        - DIS   weighted distances to five Boston employment centres
        - RAD   index of accessibility to radial highways
        - TAX   full-value property-tax rate per $10,000
        - PTRATIO pupil-teacher ratio by town
        - B   1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT   % lower status of the population
        - MEDV   Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980.   N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

```
   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

0.0.2 EDA

summarized statistics

[8]:
```
print(f"sample size: {data.shape[0]} - feature num: {features.shape[1]}")
```

```
sample size: 506 - feature num: 13
```

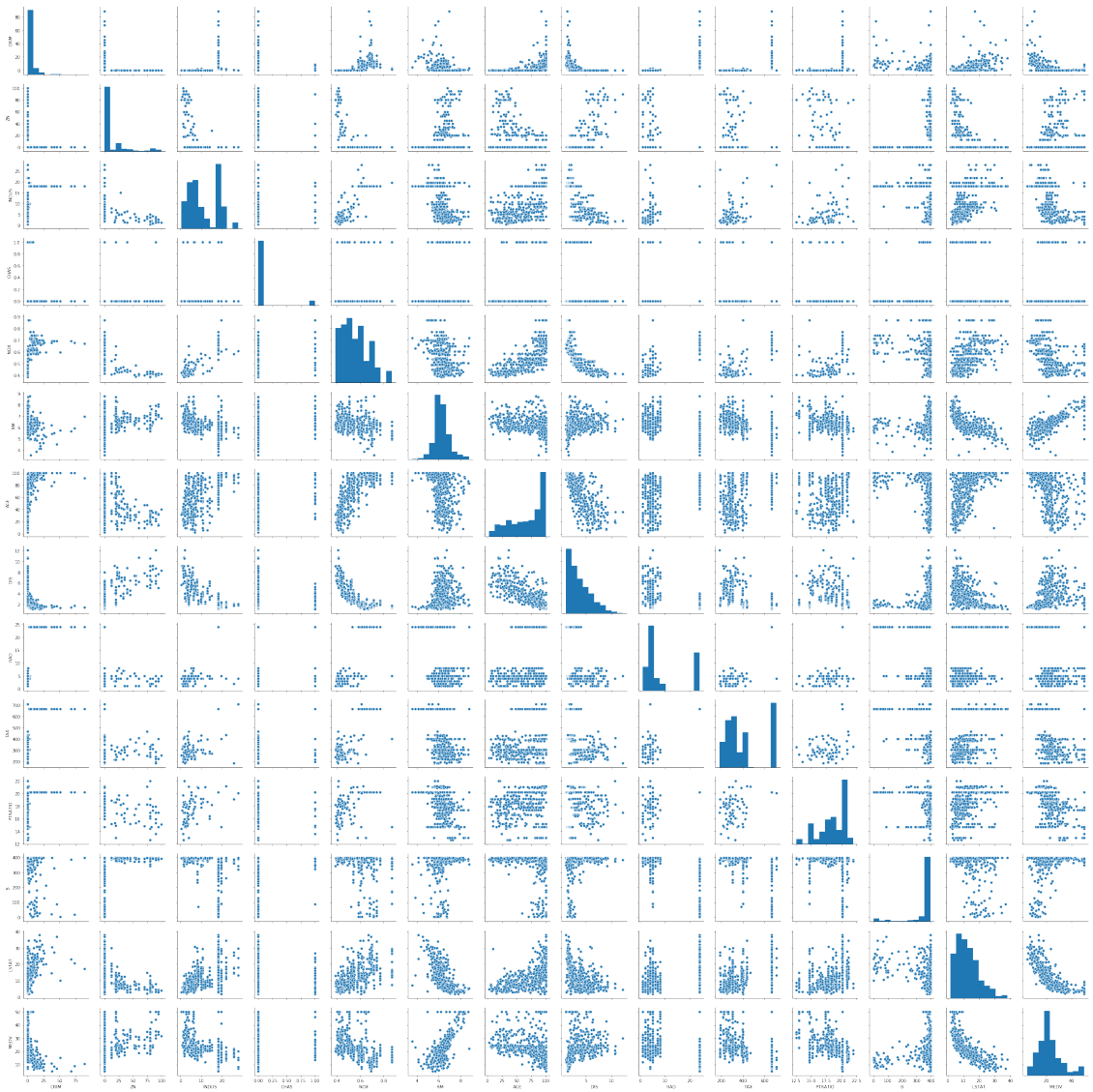
[9]:
```
prices.describe()
```

[9]:
```
count    506.000000
mean      22.532806
std        9.197104
min        5.000000
25%       17.025000
50%       21.200000
75%       25.000000
max       50.000000
Name: MEDV, dtype: float64
```

0.0.3 seaborn plot

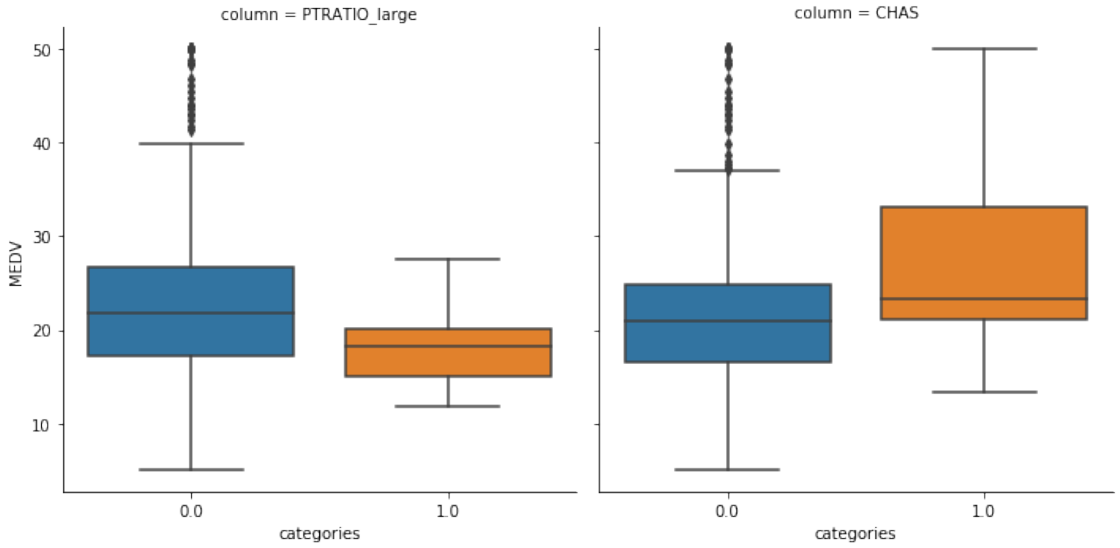
[10]:
```
import seaborn as sns
```

[11]:
```
_ = sns.pairplot(data)
```

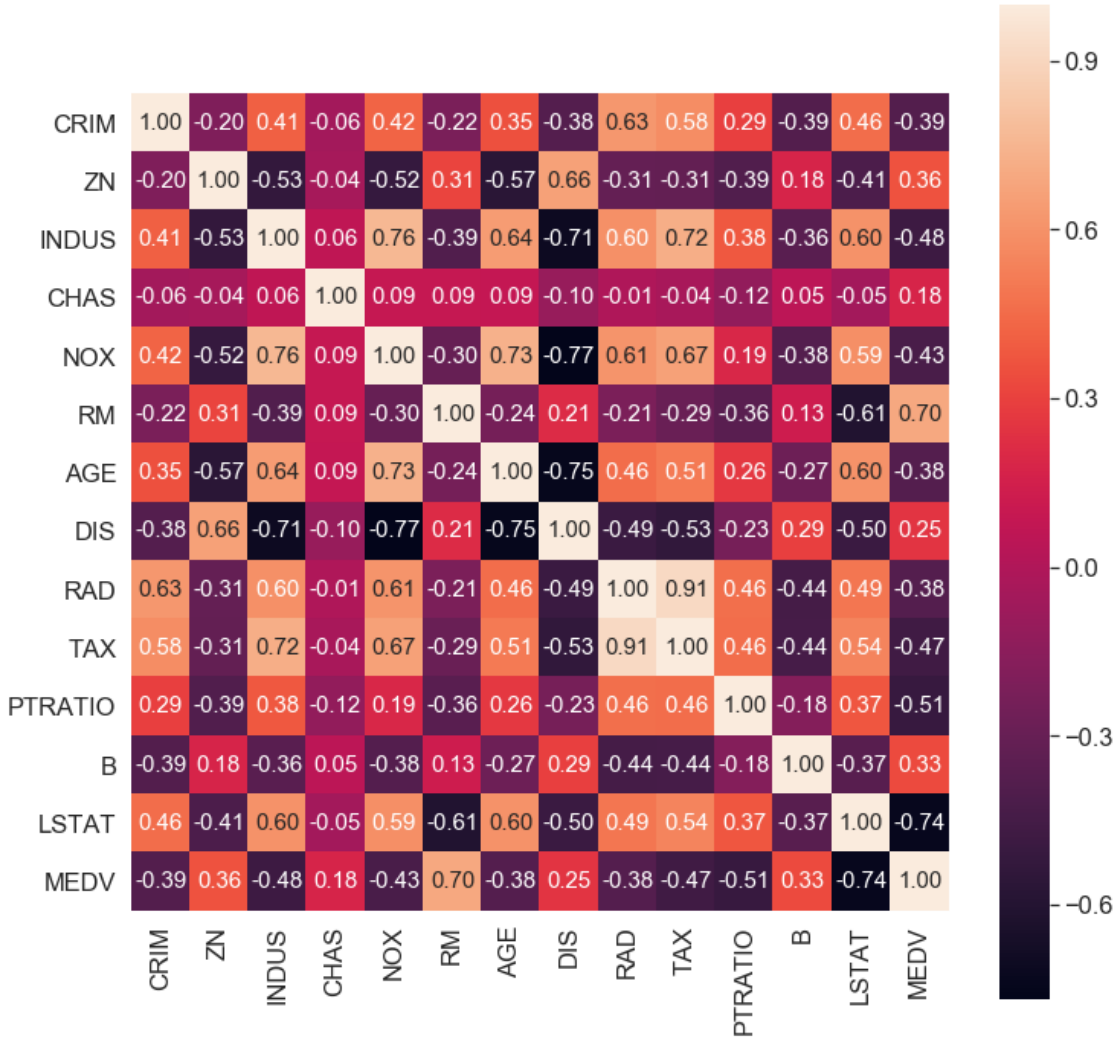


[12]:
```
def plot_cat_data(data, cat_names, target_name):
    plot_data = pd.DataFrame()
    for cat_name in cat_names:
        plot_data = pd.concat([plot_data, data.assign(column = cat_name).
↪rename(columns = {cat_name: "categories"})], axis = 0)
    sns.catplot(
        x = "categories",
        y = target_name,
        col = "column",
        kind = "box",
        data = plot_data,
        orient="v")
    return plot_data
```

[13]:
```
cat_names = ["PTRATIO_large", "CHAS"]
target_name = "MEDV"
data_ = data.assign(PTRATIO_large = data.PTRATIO > data.PTRATIO.quantile(.75))
plot_data = plot_cat_data(data_, cat_names, target_name)
```



[14]:
```
# Calculate and show correlation matrix
cm = np.corrcoef(data.values.T)
sns.set(font_scale=1.5)
cols = data.columns
fig, ax = plt.subplots(figsize = (12,12))
hm = sns.heatmap(cm,
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size': 15},
                 yticklabels=cols,
                 xticklabels=cols,
                 ax = ax)
```



0.0.4 Making a model

Train Test Splitting

[15]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, prices, test_size␣
 ↪= .3, random_state = 42)
```

Feature Selelction and Engineering

OLS

[16]:
```
import statsmodels.api as sm
mod = sm.OLS(y_train, X_train)
regr = mod.fit()
```

```
# print(regr.summary())
```

[17]:
```
from sklearn.metrics import mean_squared_error
y_test_hat = regr.predict(X_test)
benchmark = mean_squared_error(y_test_hat, y_test)
```

WLS

[18]:
```
mod = sm.WLS(y_train, X_train)
regr = mod.fit()
# print(regr.summary())
```

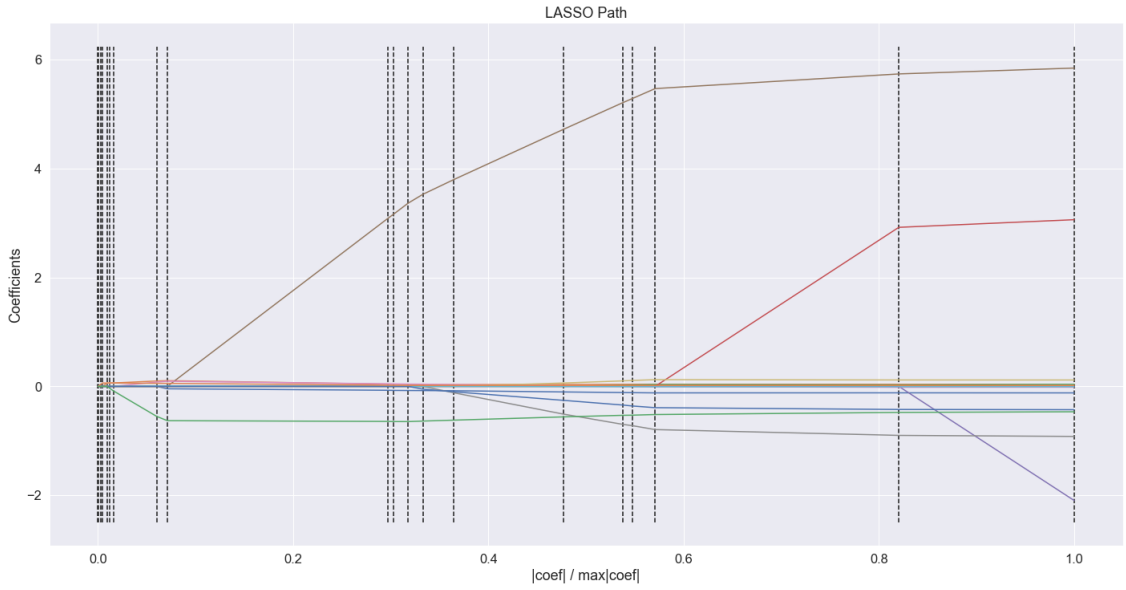
[19]:
```
from sklearn.metrics import mean_squared_error
y_test_hat = regr.predict(X_test)
print(mean_squared_error(y_test_hat, y_test) / benchmark)
```

```
1.0
```

Lasso

[20]:
```
from visualizations import plot_lasso_path

fig, ax = plt.subplots(1,1, figsize=(24, 12))
_ = plot_lasso_path(X_train.values, y_train.values, ax = ax)
```



[21]:
```
from visualizations import find_lar_coef

from sklearn.linear_model import lars_path
```

```
lmbda = .6
lambd_cutoffs, _, coefs = lars_path(X_train.values, y_train.values)

pd.DataFrame(
    data = find_lar_coef(0.2, lambd_cutoffs, coefs),
    index = X_train.columns, columns = [f"coef for lambda: {lmbda}"]
)
```

[21]:

	coef for lambda: 0.6
CRIM	0.000000
ZN	0.000000
INDUS	0.000000
CHAS	0.000000
NOX	0.000000
RM	0.000000
AGE	0.000000
DIS	0.000000
RAD	0.000000
TAX	0.133824
PTRATIO	0.000000
B	-1.927150
LSTAT	0.000000

[22]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
ALPHA = np.power(10, np.linspace(-1.5, 0.5, 20))
param_grid = {
    "alpha": ALPHA
}
regr =  mod = Lasso(fit_intercept = True)
grid = GridSearchCV(regr, param_grid, scoring="neg_mean_squared_error", cv=5,␣
 ↪n_jobs = 2, verbose = 1)
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    3.0s finished
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:813: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[22]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=1000, normalize=False, positive=False,
```

```
                                precompute=False, random_state=None,
                                selection='cyclic', tol=0.0001, warm_start=False),
             iid='warn', n_jobs=2,
             param_grid={'alpha': array([0.03162278, 0.04029611, 0.05134833,
0.06543189, 0.08337822,
       0.10624678, 0.13538762, 0.17252105, 0.21983926, 0.28013568,
       0.35696988, 0.45487779, 0.5796394 , 0.73861998, 0.94120497,
       1.19935395, 1.52830673, 1.94748304, 2.48162892, 3.16227766])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=1)
```

[23]:
```
from sklearn.metrics import mean_squared_error
y_test_hat = grid.predict(X_test)
print(mean_squared_error(y_test_hat, y_test) / benchmark)
```

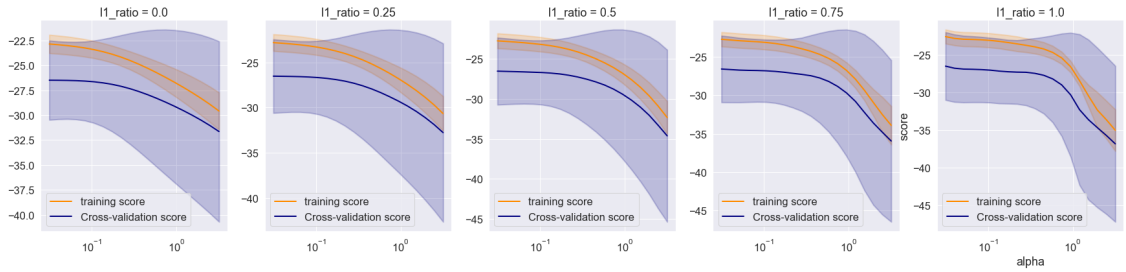
```
0.9001690886717025
```

Training Model

ElasticNet

[24]:
```
from sklearn.linear_model import ElasticNet
from visualizations import plot_validation_curves
```

[25]:
```
fig, axes = plt.subplots(1, 5, figsize = (30, 6))
L1_RATIO = np.linspace(0, 1, 5)
ALPHA = np.power(10, np.linspace(-1.5, 0.5, 20))
for i in range(5):
    mod = ElasticNet(l1_ratio = L1_RATIO[i], fit_intercept = True)
    _ = plot_validation_curves(X_train, y_train, mod, "alpha", ALPHA, n_jobs =␣
 ↪3, ax = axes[i])
    _.set_title(f"l1_ratio = {L1_RATIO[i]}")
```



[26]:
```
param_grid = {
    "l1_ratio": L1_RATIO,
    "alpha": ALPHA
}
regr =  mod = ElasticNet(fit_intercept = True)
```

```
grid = GridSearchCV(regr, param_grid, scoring="neg_mean_squared_error", cv=5,␣
 ↪n_jobs = 2, verbose = 1)
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits

[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done 500 out of 500 | elapsed:    1.2s finished
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:813: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[26]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True,
                                  l1_ratio=0.5, max_iter=1000, normalize=False,
                                  positive=False, precompute=False,
                                  random_state=None, selection='cyclic',
                                  tol=0.0001, warm_start=False),
             iid='warn', n_jobs=2,
             param_grid={'alpha': array([0.03162278, 0.04029611, 0.05134833,
0.06543189, 0.08337822,
       0.10624678, 0.13538762, 0.17252105, 0.21983926, 0.28013568,
       0.35696988, 0.45487779, 0.5796394 , 0.73861998, 0.94120497,
       1.19935395, 1.52830673, 1.94748304, 2.48162892, 3.16227766]),
                         'l1_ratio': array([0.  , 0.25, 0.5 , 0.75, 1.  ])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=1)
```

[27]:
```
from sklearn.metrics import mean_squared_error
y_test_hat = grid.predict(X_test)
print(mean_squared_error(y_test_hat, y_test) / benchmark)
```

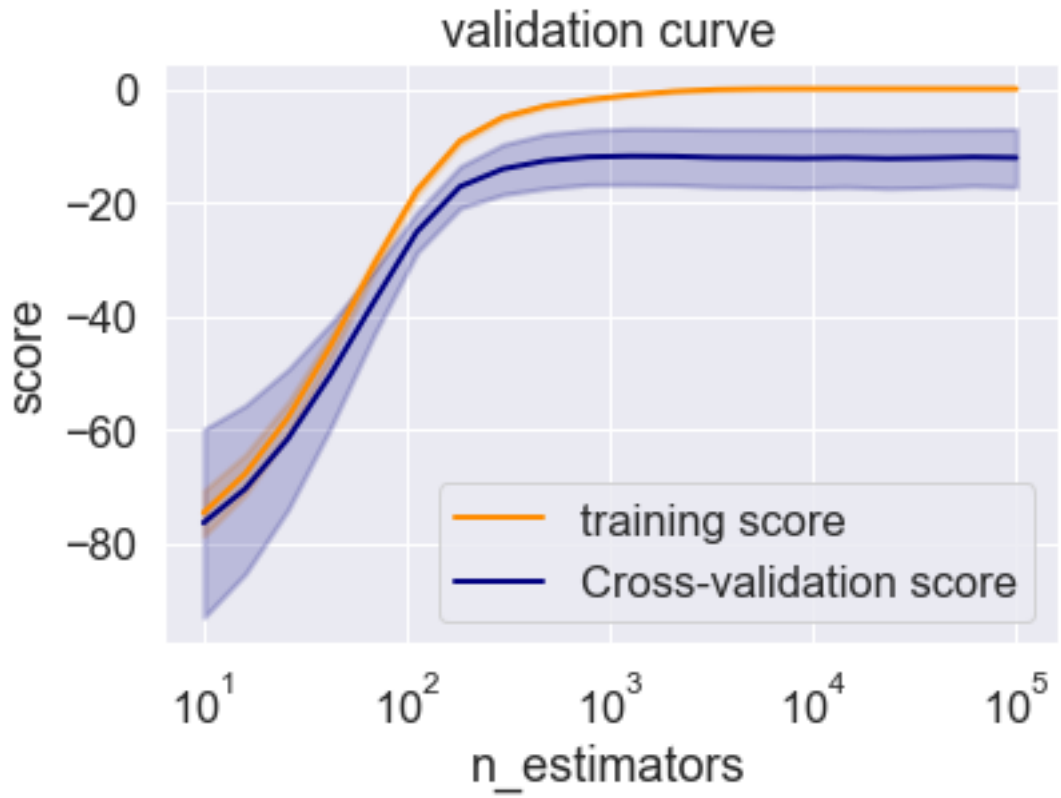
```
0.9001690886717025
```

Training and Plotting for Important Hypermaters

[28]:
```
from sklearn.ensemble import GradientBoostingRegressor
regr = GradientBoostingRegressor(loss = "ls", learning_rate = .01, max_depth =␣
 ↪3, )
```

[29]:
```
from visualizations import plot_validation_curves
plot_validation_curves(X_train, y_train, regr, "n_estimators", np.power(10, np.
 ↪linspace(1, 5, 20)).astype(int), n_jobs = 2)
```

[29]:
```
<module 'matplotlib.pyplot' from 'C:\\ProgramData\\Anaconda3\\lib\\site-
packages\\matplotlib\\pyplot.py'>
```



Grid Search for Optimal Hyperparameters

[30]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
regr = GradientBoostingRegressor(loss = "ls")
```

[31]:
```
param_grid = {
    "learning_rate": np.power(10, np.linspace(-5, -1, 5)),
    "max_depth": [1, 3, 5],
    "n_estimators": np.power(10, np.linspace(1, 5, 5)).astype(int)
}
grid = GridSearchCV(regr, param_grid, scoring="neg_mean_squared_error", cv=5,␣
 ↪n_jobs = 2, verbose = 1)
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 75 candidates, totalling 375 fits

[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done 100 tasks      | elapsed:  4.9min
[Parallel(n_jobs=2)]: Done 250 tasks      | elapsed: 12.4min
[Parallel(n_jobs=2)]: Done 375 out of 375 | elapsed: 123.7min finished
C:\ProgramData\Anaconda3\lib\site-
```

```
packages\sklearn\model_selection\_search.py:813: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[31]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.1,
                                                 loss='ls', max_depth=3,
                                                 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,
                                                 n_estimators=100,
                                                 n_iter...
                                                 random_state=None,
                                                 subsample=1.0, tol=0.0001,
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=False),
             iid='warn', n_jobs=2,
             param_grid={'learning_rate': array([1.e-05, 1.e-04, 1.e-03, 1.e-02,
1.e-01]),
                         'max_depth': [1, 3, 5],
                         'n_estimators': array([    10,    100,   1000,  10000,
100000])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=1)
```

[33]:
```
from sklearn.metrics import mean_squared_error
y_test_hat = grid.predict(X_test)
mean_squared_error(y_test_hat, y_test) / benchmark
```

[33]:
```
0.33308849866665063
```

[ ]:

[ ]:

[ ]: