線形回帰モデル（演習）

+ コード　+ テキスト

線形回帰モデル-Boston Hausing Data-

+ コード　+ テキスト

1. 必要モジュールとデータのインポート

[44]
```python
#from モジュール名 import クラス名（もしくは関数名や変数名）

from sklearn.datasets import load_boston
from pandas import DataFrame
import numpy as np
```

[45]
```python
# ボストンデータを"boston"というインスタンスにインポート
boston = load_boston()
```

[46]
```python
#インポートしたデータを確認(data / target / feature_names / DESCR)
print(boston)
# print(boston.feature_names)
```

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]]), 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]), 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'), 'DESCR': ".. _boston_dataset:¥n¥nBoston house prices dataset¥n--------------------------¥n¥n**Data Set Characteristics:**  ¥n¥n    :Number
```

+ コード　+ テキスト

[47]
```python
#DESCR変数の中身を確認
print(boston['DESCR'])
```

```
.. _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. M
```

```
#feature_names変数の中身を確認
#カラム名
print(boston['feature_names'])
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

[49]
```
#data変数(説明変数)の中身を確認
print(boston['data'])
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]
```

[50]
```
#target変数(目的変数)の中身を確認
print(boston['target'])
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
```

+ コード　+ テキスト

2. データフレームの作成

[51]
```
#  説明変数らをDataFrameへ変換
df = DataFrame(data=boston.data, columns = boston.feature_names)
```

[52]
```
# 目的変数をDataFrameへ追加
df['PRICE'] = np.array(boston.target)
```

[53]
```
# 最初の5行を表示
df.head(5)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

線形単回帰分析

```
#カラムを指定してデータを表示
df[['RM']].head()
```

	RM
0	6.575
1	6.421
2	7.185
3	6.998
4	7.147

[55]
```
# 説明変数
data = df.loc[:, ['RM']].values
```

[56]
```
#dataリストの表示(1-5)
data[0:5]
```

```
array([[6.575],
       [6.421],
       [7.185],
       [6.998],
       [7.147]])
```

[57]
```
# 目的変数
target = df.loc[:, 'PRICE'].values
```

[58]
```
target[0:5]
```

```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

[59]
```
## sklearnモジュールからLinearRegressionをインポート
from sklearn.linear_model import LinearRegression
```

[60]
```
# オブジェクト生成
model = LinearRegression()
#model.get_params()
#model = LinearRegression(fit_intercept = True, normalize = False, copy_X = True,  n_jobs = 1)
```

[61]
```
# fit関数でパラメータ推定
model.fit(data, target)
```

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

+ コード + テキスト

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

新しいセクション

[62]
```
#予測
model.predict([[1]])
```

```
array([-25.5685118])
```

重回帰分析(2変数)

[63]
```
#カラムを指定してデータを表示
df[['CRIM', 'RM']].head()
```

	CRIM	RM
0	0.00632	6.575
1	0.02731	6.421
2	0.02729	7.185
3	0.03237	6.998
4	0.06905	7.147

[64]
```
# 説明変数
data2 = df.loc[:, ['CRIM', 'RM']].values
# 目的変数
target2 = df.loc[:, 'PRICE'].values
```

[65]
```
# オブジェクト生成
model2 = LinearRegression()
```

[66]
```
# fit関数でパラメータ推定
model2.fit(data2, target2)
```

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

[67]
```
model2.predict([[0.2, 7]])
```

```
array([29.43977562])
```

回帰係数と切片の値を確認

[68]
```
# 単回帰の回帰係数と切片を出力
print('推定された回帰係数: %.3f, 推定された切片 : %.3f' % (model.coef_, model.intercept_))
```

```
推定された回帰係数: 9.102, 推定された切片 : -34.671
```

[69]
```
# 重回帰の回帰係数と切片を出力
print(model.coef_)
print(model.intercept_)
```

```
[9.10210898]
-34.67062077643857
```

+ コード + テキスト

モデルの検証

1. 決定係数

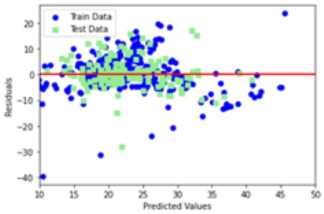
決定係数

print('単回帰決定係数: %.3f, 重回帰決定係数 : %.3f' % (model.score(data,target), model2.score(data2,target2)))

[70]
```
# train_test_splitをインポート
from sklearn.model_selection import train_test_split
```

[71]
```
# 70%を学習用、30%を検証用データにするよう分割
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size = 0.3, random_state = 666)
# 学習用データでパラメータ推定
model.fit(X_train, y_train)
# 作成したモデルから予測（学習用、検証用モデル使用）
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

[72]
```
# matplotlibをインポート
import matplotlib.pyplot as plt
# Jupyterを利用していたら、以下のおまじないを書くとnotebook上に図が表示
%matplotlib inline
# 学習用、検証用それぞれで残差をプロット
plt.scatter(y_train_pred, y_train_pred - y_train, c = 'blue', marker = 'o', label = 'Train Data')
plt.scatter(y_test_pred, y_test_pred - y_test, c = 'lightgreen', marker = 's', label = 'Test Data')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
# 凡例を左上に表示
plt.legend(loc = 'upper left')
# y = 0に直線を引く
plt.hlines(y = 0, xmin = -10, xmax = 50, lw = 2, color = 'red')
plt.xlim([10, 50])
plt.show()
```



[73]

```python
# 平均二乗誤差を評価するためのメソッドを呼び出し
from sklearn.metrics import mean_squared_error
# 学習用、検証用データに関して平均二乗誤差を出力
print('MSE Train : %.3f, Test : %.3f' % (mean_squared_error(y_train, y_train_pred), mean_squared_error(y_test, y_test_pred)))
# 学習用、検証用データに関してR^2を出力
print('R^2 Train : %.3f, Test : %.3f' % (model.score(X_train, y_train), model.score(X_test, y_test)))
```

```
MSE Train : 44.983, Test : 40.412
R^2 Train : 0.500, Test : 0.434
```

[73]

```python

```

0 秒　完了時間: 10:34

+ コード + テキスト

0.0 0.2 0.4 0.6 0.8 1.0

学習

[21]
```python
#numpy実装の回帰
def train(xs, ys):
  cov = np.cov(xs, ys, ddof=0)
  a = cov[0, 1] / cov[0, 0]
  b = np.mean(ys) - a * np.mean(xs)
  return cov, a, b

cov, a, b = train(xs, ys)
print("共分散/分散：covarient:{}".format(cov))
print("回帰係数：coefficient:{}".format(a))
print("切片：intercept:{}".format(b))
```

```
共分散/分散：covarient:[[0.08501684 0.17298021]
 [0.17298021 0.39022914]]
回帰係数：coefficient:2.0346583042651076
切片：intercept:4.9831835621376035
```

[22]
```python
#skl実装の回帰
from sklearn.linear_model import LinearRegression
model = LinearRegression()
reg = model.fit(xs.reshape(-1, 1), ys.reshape(-1, 1))

print("回帰係数：coef_:{}".format(reg.coef_))
print("切片：intercept_:{}".format(reg.intercept_))
```

```
回帰係数：coef_:[[2.0346583]]
切片：intercept_:[4.98318356]
```

予測

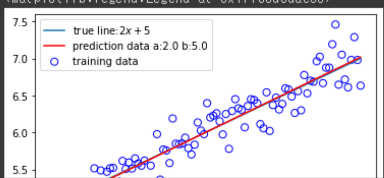
入力に対する値を$y(x) = ax + b$で予測する。

[23]
```python

xs_new = np.linspace(0, 1, n_sample)
#a,bは「学習済回帰係数」と「学習済切片値」
#今回、ys_predはランダム化していない。
ys_pred = a * xs_new + b

#学習値による回帰直性
plt.scatter(xs, ys, facecolor="none", edgecolor="b", s=50, label="training data")
plt.plot(xs_new, ys_true,label= "true line:$2x + 5$")
plt.plot(xs_new, ys_pred, color= "r", label="prediction data a:{:.2} b:{:.2}".format(a,b))
plt.legend()
```

```
&lt;matplotlib.legend.Legend at 0x7f786d3dae90&gt;
```

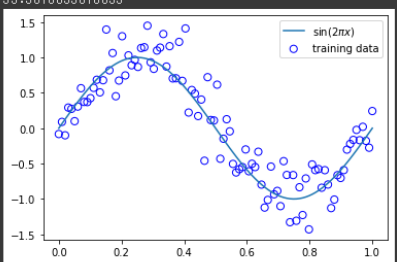


✓ 0 秒 完了時間: 21:27

+ コード　+ テキスト

RAM
ディスク

[25]
```
 0.36363636 0.37373737 0.38383838 0.39393939 0.4040404  0.41414141
 0.42424242 0.43434343 0.44444444 0.45454545 0.46464646 0.47474747
 0.48484848 0.49494949 0.50505051 0.51515152 0.52525253 0.53535354
 0.54545455 0.55555556 0.56565657 0.57575758 0.58585859 0.5959596
 0.60606061 0.61616162 0.62626263 0.63636364 0.64646465 0.65656566
 0.66666667 0.67676768 0.68686869 0.6969697  0.70707071 0.71717172
 0.72727273 0.73737374 0.74747475 0.75757576 0.76767677 0.77777778
 0.78787879 0.7979798  0.80808081 0.81818182 0.82828283 0.83838384
 0.84848485 0.85858586 0.86868687 0.87878788 0.88888889 0.8989899
 0.90909091 0.91919192 0.92929293 0.93939394 0.94949495 0.95959596
 0.96969697 0.97979798 0.98989899 1.        ]
1
[0.         0.01010101 0.02020202 0.03030303 0.04040404 0.05050505
 0.06060606 0.07070707 0.08080808 0.09090909 0.1010101  0.11111111
 0.12121212 0.13131313 0.14141414 0.15151515 0.16161616 0.17171717
 0.18181818 0.19191919 0.2020202  0.21212121 0.22222222 0.23232323
 0.24242424 0.25252525 0.26262626 0.27272727 0.28282828 0.29292929
 0.3030303  0.31313131 0.32323232 0.33333333 0.34343434 0.35353535
 0.36363636 0.37373737 0.38383838 0.39393939 0.4040404  0.41414141
 0.42424242 0.43434343 0.44444444 0.45454545 0.46464646 0.47474747
 0.48484848 0.49494949 0.50505051 0.51515152 0.52525253 0.53535354
 0.54545455 0.55555556 0.56565657 0.57575758 0.58585859 0.5959596
 0.60606061 0.61616162 0.62626263 0.63636364 0.64646465 0.65656566
 0.66666667 0.67676768 0.68686869 0.6969697  0.70707071 0.71717172
 0.72727273 0.73737374 0.74747475 0.75757576 0.76767677 0.77777778
 0.78787879 0.7979798  0.80808081 0.81818182 0.82828283 0.83838384
 0.84848485 0.85858586 0.86868687 0.87878788 0.88888889 0.8989899
 0.90909091 0.91919192 0.92929293 0.93939394 0.94949495 0.95959596
 0.96969697 0.97979798 0.98989899 1.        ]
1
33.5016835016835
```



学習

```
[26]
def polynomial_features(xs, degree=3):
    """多項式特徴ベクトルに変換
         X = [[1, x1, x1^2, x1^3],
                 [1, x2, x2^2, x2^3],
                  ...
                 [1, xn, xn^2, xn^3]]"""
    X = np.ones((len(xs), degree+1))
    X_t = X.T

    # print(X.shape)   #(10, 4)
    # print(X_t.shape) #(4, 10)

    for i in range(1, degree+1):
        X_t[i] = X_t[i-1] * xs  # [1,xs,xs^2, xs^3・・・・xs^(deg+1)]
    return X_t.T


```


```
[27]
Phi = polynomial_features(xs)
```


+ コード　+ テキスト　RAM ディスク

```
[27]
Phi = polynomial_features(xs)
Phi_inv = np.dot(np.linalg.inv(np.dot(Phi.T, Phi)), Phi.T)
w = np.dot(Phi_inv, ys)
```

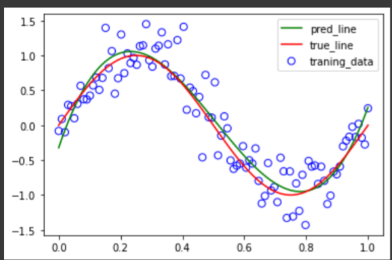
予測

入力を多項式特徴ベクトル$\phi(x)$に変換し、$y = \hat{\omega}(x)(y(x) = \phi\hat{\omega})$で予測する。

```
[28]
Phi_test = polynomial_features(xs)
ys_pred = np.dot(Phi_test, w)

```

```
plt.scatter(xs,ys, facecolor="none", edgecolor="b",s=50, label="traning_data")
plt.plot(xs,ys_pred, color="g", label="pred_line")
plt.plot(xs,ys_true, color="r", label="true_line")
plt.legend()
plt.show()
```



重回帰分析

データ生成（3次元入力）

```
[30]
np.random.random((10, 3))
# help(np.random.random)
```

```
array([[0.21847938, 0.59071751, 0.37915162],
       [0.86171949, 0.03275228, 0.1470459 ],
       [0.22540672, 0.13998905, 0.17718038],
       [0.0248776 , 0.06876249, 0.20029515],
       [0.89959682, 0.61812503, 0.14794499],
       [0.57559898, 0.27428556, 0.60484786],
       [0.76465241, 0.28996051, 0.70600908],
       [0.15591789, 0.67347911, 0.73553966],
       [0.14340457, 0.52239257, 0.26799437],
       [0.49058151, 0.68330948, 0.69448236]])
```

```
[31]
n_sample = 100
var = .2

def mul_linear_func(x):
    ww = [1., 0.5, 2., 1.]
    return ww[0] + ww[1] * x[:, 0] + ww[2] * x[:, 1] + ww[3] * x[:, 2]

def add_noise(y_true, var):
    return y_true + np.random.normal(scale=var, size=y_true.shape)

```

✓ 0秒　完了時間: 21:27

+ コード + テキスト

RAM
ディスク

予測

入力に対する値を$\boldsymbol{y}(\boldsymbol{x}) = \hat{\boldsymbol{w}}^{\mathrm{T}}\boldsymbol{x}\ (\boldsymbol{y} = \boldsymbol{X}\hat{\boldsymbol{w}})$で予測する

[34]

```python
X_pred = np.random.random((n_sample, x_dim))
ys_pred_true = mul_linear_func(X_pred)
# ys_pred = add_noise(ys_pred_true, var)


# print(add_one(X_pred).shape)
# print(w)


#学習済w を用いて
ys_pred = np.dot(w,add_one(X_pred).T)

# print(add_one(X_pred)[0])
# print(w.T[0])
# print(ys_pred)

# np.dot(add_one(X_pred)[0],w.T[0])

for i in range(len(X_pred)):
    print("w{0}_true: {1:>5.2}   w{0}_estimated: {2:>5.2}".format(i, ys_pred_true[i], ys_pred[i]))

# print(X_pred.shape)
# print(ys_pred_true.shape)
# print(ys_pred_true.shape)
```

```
# plt_result(X_pred, ys_pred_true, ys_pred)
```

```
w25_true:   2.7   w25_estimated:   2.7
w26_true:   2.3   w26_estimated:   2.3
w27_true:   2.0   w27_estimated:   1.9
w28_true:   3.5   w28_estimated:   3.5
w29_true:   2.3   w29_estimated:   2.3
w30_true:   1.9   w30_estimated:   1.9
w31_true:   3.0   w31_estimated:   3.0
w32_true:   2.7   w32_estimated:   2.8
w33_true:   3.6   w33_estimated:   3.6
w34_true:   2.3   w34_estimated:   2.4
w35_true:   2.3   w35_estimated:   2.3
w36_true:   2.7   w36_estimated:   2.7
w37_true:   3.6   w37_estimated:   3.6
w38_true:   4.1   w38_estimated:   4.2
w39_true:   2.3   w39_estimated:   2.3
w40_true:   1.8   w40_estimated:   1.8
w41_true:   3.2   w41_estimated:   3.2
w42_true:   2.4   w42_estimated:   2.4
w43_true:   2.2   w43_estimated:   2.2
w44_true:   2.5   w44_estimated:   2.5
w45_true:   2.3   w45_estimated:   2.3
w46_true:   2.3   w46_estimated:   2.2
w47_true:   2.6   w47_estimated:   2.6
w48_true:   3.5   w48_estimated:   3.5
w49_true:   1.8   w49_estimated:   1.8
w50_true:   2.4   w50_estimated:   2.3
w51_true:   3.0   w51_estimated:   2.9
w52_true:   1.8   w52_estimated:   1.8
w53_true:   2.3   w53_estimated:   2.3
w54_true:   3.1   w54_estimated:   3.1
w55_true:   1.8   w55_estimated:   1.8
w56_true:   3.5   w56_estimated:   3.5
w57_true:   2.4   w57_estimated:   2.4
w58_true:   3.1   w58_estimated:   3.2
```

0 秒　完了時間: 21:27