機械学習レポート

1　線形回帰モデル

1.1　要点のまとめ

・ある入力から出力を予測する問題。そのうち、直線で予測できるもの。
・教師あり学習。
・入力と m 次元パラメータの線形結合（入力とパラメータの内積）を固有値力するモデル
・パラメータは最小二乗法により推定する。
　最小二乗法とは学習データの平均二乗誤差を最小とするもの。
　最小化は勾配が 0 となる点を求める。
・説明変数が 1 次元の場合は単回帰モデルと呼ぶ。
・データは回帰直線に誤差が加わり観測されていると仮定する。

1.2　実装演習結果

次ページ以降に演習の結果を掲載

線形回帰モデル-Boston Hausing Data-

1. 必要モジュールとデータのインポート

```
#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
```

```
# 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()
```

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

```
{'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, 1
       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, 4
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,
```



```
#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

    :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 Univer

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 regres
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on
```

```
#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']
```



```
#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]]
```

```
#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
```

```
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  2
 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]
```

2. データフレームの作成


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

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

# 最初の5行を表示
df.head(5)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

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

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


#dataリストの表示(1-5)
data[0:5]
```

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

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


target[0:5]
```

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

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


# 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)


# 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)
```

```
#予測
model.predict([[1]])
```

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

重回帰分析(2変数)

```
#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

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

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

```
# 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)
```



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

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

回帰係数と切片の値を確認

```
# 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
```

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

```
[9.10210898]
-34.67062077643857
```

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)))

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


# 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)


# 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()
```


```
# 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_er
# 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
```

✓ 0 秒 完了時間: 23:08

機械学習レポート

2　非線形回帰モデル

2.1　要点のまとめ

・ある入力から出力を予測する問題。そのうち、直線で表現できないもの。

・基底展開法は回帰関数として、基底関数と呼ばれる既知の非線形関数とパラメータベクトルの線型結合を使用する。よく使われる基底関数は下記のようなものがある。

　多項式関数

　ガウス型基底関数

　スプライン関数/B スプライン関数

・学習データに対して十分小さな誤差が得られないモデルは未学習である。対策としては表現力の高いモデルを利用する。

・小さな誤差であるが、テスト集合ござとの差が大きいモデルは過学習の可能性があり、下記の多様な対策を行う。

　学習データの数を増やす。

　不要な基底関数(変数)を削除して表現力を抑止

　正則化法を利用して表現力を抑止

・汎化性能が高いモデルとは、(学習誤差ではなく)汎化誤差(テスト誤差)が小さいモデル。本来目指すべきモデルとなる。

・クロスバリデーション(交差検証)とは、データを学習用と評価用に分割し学習・検証を行うもの。

2.2　実装演習結果

次ページ以降に演習の結果を掲載

Googleドライブのマウント

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

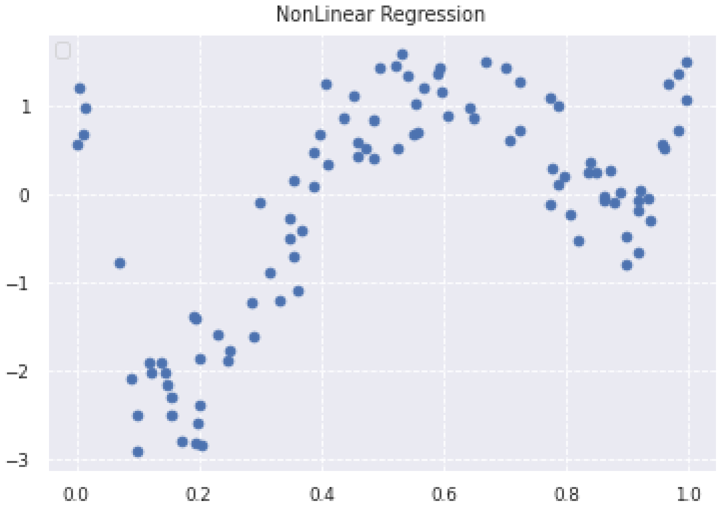
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

```
#seaborn設定
sns.set()
#背景変更
sns.set_style("darkgrid", {'grid.linestyle': '--'})
#大きさ(スケール変更)
sns.set_context("paper")
```

```
n=100

def true_func(x):
    z = 1-48*x+218*x**2-315*x**3+145*x**4
    return z

def linear_func(x):
    z = x
    return z
```

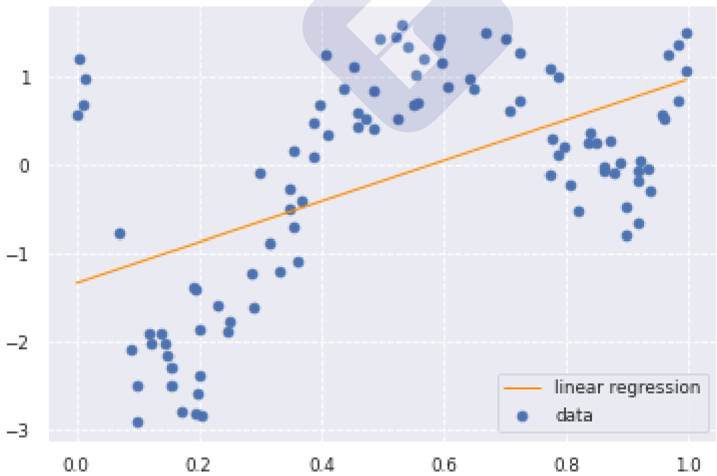
```
# 真の関数からノイズを伴うデータを生成

# 真の関数からデータ生成
data = np.random.rand(n).astype(np.float32)
data = np.sort(data)
target = true_func(data)

#  ノイズを加える
noise = 0.5 * np.random.randn(n)
target = target  + noise

# ノイズ付きデータを描画

plt.scatter(data, target)

plt.title('NonLinear Regression')
plt.legend(loc=2)
```

```
No handles with labels found to put in legend.
<matplotlib.legend.Legend at 0x7ff83f8de790>
```




```
from sklearn.linear_model import LinearRegression

clf = LinearRegression()
data = data.reshape(-1, 1)
target = target.reshape(-1, 1)
clf.fit(data, target)

p_lin = clf.predict(data)

plt.scatter(data, target, label='data')
plt.plot(data, p_lin, color='darkorange', marker='', linestyle='-', linewidth=1, markersize=6, labe
plt.legend()
print(clf.score(data, target))
```

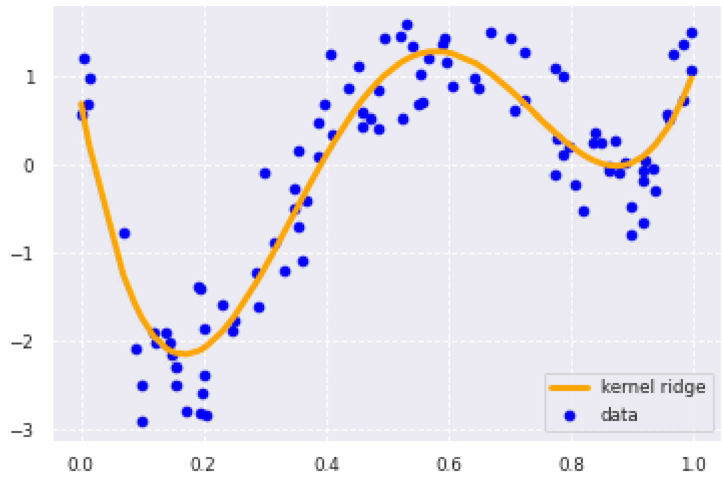
```
0.2971578022172682
```




```
from sklearn.kernel_ridge import KernelRidge

clf = KernelRidge(alpha=0.0002, kernel='rbf')
clf.fit(data, target)

p_kridge = clf.predict(data)

plt.scatter(data, target, color='blue', label='data')
```

```
plt.plot(data, p_kridge, color='orange', linestyle='-', linewidth=3,
plt.legend()
#plt.plot(data, p, color='orange', marker='o', linestyle='-', linewidth=1, markersize=6)
```

<matplotlib.legend.Legend at 0x7ff83ba88fd0>

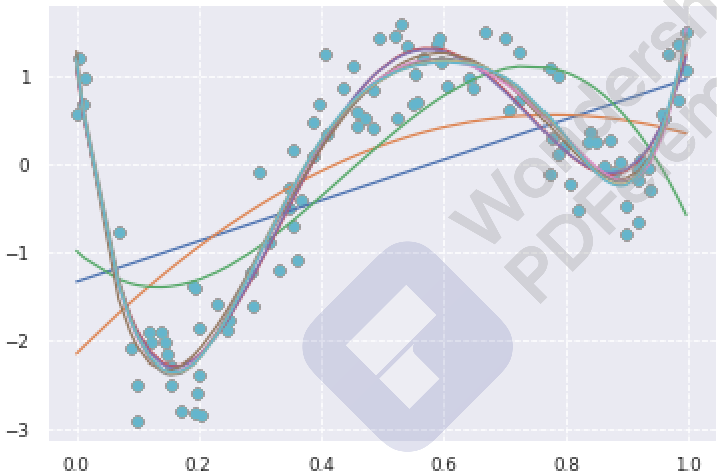



```
#Ridge

from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Ridge

kx = rbf_kernel(X=data, Y=data, gamma=50)
#KX = rbf_kernel(X, x)

#clf = LinearRegression()
clf = Ridge(alpha=30)
clf.fit(kx, target)

p_ridge = clf.predict(kx)

plt.scatter(data, target, label='data')
for i in range(len(kx)):
    plt.plot(data, kx[i], color='black', linestyle='-', linewidth=1, markersize=3, label='rbf', alp

#plt.plot(data, p, color='green', marker='o', linestyle='-', linewidth=0.1, markersize=3)
plt.plot(data, p_ridge, color='green', linestyle='-', linewidth=1, markersize=3, label='ridge regres
#plt.legend()

print(clf.score(kx, target))
```

```
0.8250001801699819
```

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

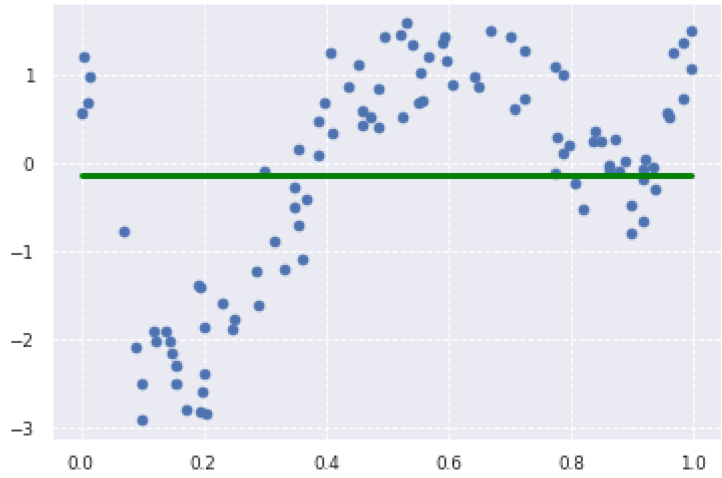
```
#PolynomialFeatures(degree=1)

deg = [1,2,3,4,5,6,7,8,9,10]
for d in deg:
    regr = Pipeline([
        ('poly', PolynomialFeatures(degree=d)),
        ('linear', LinearRegression())
    ])
    regr.fit(data, target)
    # make predictions
    p_poly = regr.predict(data)
    # plot regression result
    plt.scatter(data, target, label='data')
    plt.plot(data, p_poly, label='polynomial of degree %d' % (d))
```




```
#Lasso

from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Lasso

kx = rbf_kernel(X=data, Y=data, gamma=5)
#KX = rbf_kernel(X, x)

#lasso_clf = LinearRegression()
lasso_clf = Lasso(alpha=10000, max_iter=1000)
lasso_clf.fit(kx, target)

p_lasso = lasso_clf.predict(kx)

plt.scatter(data, target)

#plt.plot(data, p, color='green', marker='o', linestyle='-', linewidth=0.1, markersize=3)
plt.plot(data, p_lasso, color='green', linestyle='-', linewidth=3, markersize=3)
```

```
plt.plot(data, p_lasso, color='green', linestyle='-', linewidth=3,

print(lasso_clf.score(kx, target))
```

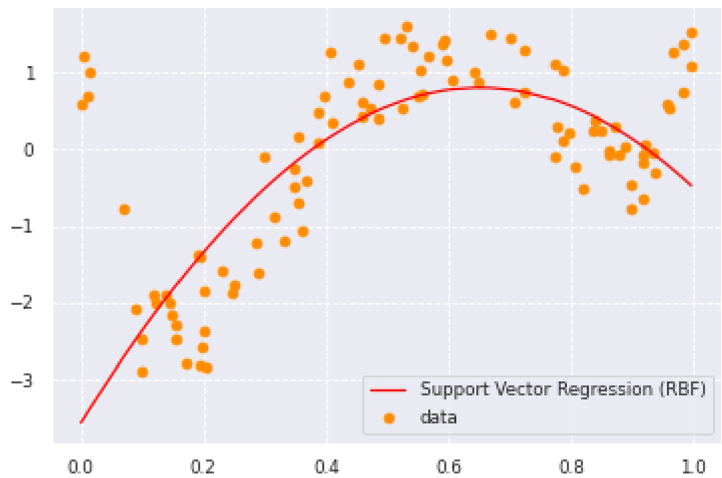
```
-4.440892098500626e-16
```




```
from sklearn import model_selection, preprocessing, linear_model, svm

# SVR-rbf
clf_svr = svm.SVR(kernel='rbf', C=1e3, gamma=0.1, epsilon=0.1)
clf_svr.fit(data, target)
y_rbf = clf_svr.fit(data, target).predict(data)

# plot

plt.scatter(data, target, color='darkorange', label='data')
plt.plot(data, y_rbf, color='red', label='Support Vector Regression (RBF)')
plt.legend()
plt.show()
```



```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning
  y = column_or_1d(y, warn=True)
```




```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size=0.1, random_state=0)
```

以下では，Googleドライブのマイドライブ直下にstudy_ai_mlフォルダを置くことを仮定しています．必要に応じて，パスを変更してください。

```
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint

cb_cp = ModelCheckpoint('/content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/weights.{epoch:
cb_tf  = TensorBoard(log_dir='/content/drive/My Drive/study_ai_ml/skl_ml/out/tensorBoard', histogra
```

```
def relu_reg_model():
    model = Sequential()
    model.add(Dense(10, input_dim=1, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='linear'))
#     model.add(Dense(100, activation='relu'))
#     model.add(Dense(100, activation='relu'))
#     model.add(Dense(100, activation='relu'))
#     model.add(Dense(100, activation='relu'))
    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer='adam')
    return model


from keras.models import Sequential
from keras.layers import Input, Dense, Dropout, BatchNormalization
from keras.wrappers.scikit_learn import KerasRegressor

# use data split and fit to run the model
estimator = KerasRegressor(build_fn=relu_reg_model, epochs=100, batch_size=5, verbose=1)

history = estimator.fit(x_train, y_train, callbacks=[cb_cp, cb_tf], validation_data=(x_test, y_test
```

```
    Epoch 00082: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
    Epoch 83/100
    18/18 [==============================] - 0s 6ms/step - loss: 0.2372 - val_loss: 0.2805

    Epoch 00083: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
    Epoch 84/100
    18/18 [==============================] - 0s 5ms/step - loss: 0.2374 - val_loss: 0.2914

    Epoch 00084: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
    Epoch 85/100
    18/18 [==============================] - 0s 6ms/step - loss: 0.3499 - val_loss: 0.4015

    Epoch 00085: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
    Epoch 86/100
    18/18 [==============================] - 0s 6ms/step - loss: 0.3179 - val_loss: 0.3155

    Epoch 00086: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
```

```
Epoch 87/100
18/18 [==============================] - 0s 5ms/step - loss: 0.3013 - val_loss: 0.4298

Epoch 00087: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 88/100
18/18 [==============================] - 0s 6ms/step - loss: 0.2728 - val_loss: 0.3113

Epoch 00088: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 89/100
18/18 [==============================] - 0s 6ms/step - loss: 0.3032 - val_loss: 0.2524

Epoch 00089: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 90/100
18/18 [==============================] - 0s 6ms/step - loss: 0.2207 - val_loss: 0.2911

Epoch 00090: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 91/100
18/18 [==============================] - 0s 5ms/step - loss: 0.2959 - val_loss: 0.2655

Epoch 00091: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 92/100
18/18 [==============================] - 0s 5ms/step - loss: 0.2292 - val_loss: 0.2871

Epoch 00092: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 93/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2647 - val_loss: 0.4099

Epoch 00093: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 94/100
18/18 [==============================] - 0s 7ms/step - loss: 0.2754 - val_loss: 0.3482

Epoch 00094: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 95/100
18/18 [==============================] - 0s 7ms/step - loss: 0.2159 - val_loss: 0.3610

Epoch 00095: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 96/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2304 - val_loss: 0.3580

Epoch 00096: saving model to /content/drive/My Drive/study_ai_ml/skl_ml/out/checkpoints/w
Epoch 97/100
```

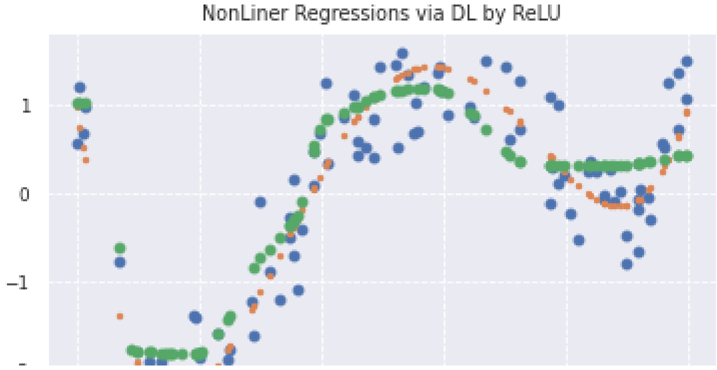


```
y_pred = estimator.predict(x_train)
```

```
18/18 [==============================] - 1s 2ms/step
```

```
plt.title('NonLiner Regressions via DL by ReLU')
plt.plot(data, target, 'o')
plt.plot(data, true_func(data), '.')
plt.plot(x_train, y_pred, "o", label='predicted: deep learning')
#plt.legend(loc=2)
```

```
[<matplotlib.lines.Line2D at 0x7ff7986ae4d0>]
```




```
print(lasso_clf.coef_)
```

```
[-0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.
 -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.
 -0. -0. -0. -0. -0. -0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
```


✓ 0 秒 完了時間: 23:13

機械学習レポート

3 ロジスティック回帰モデル

3.1 要点のまとめ

・分類問題を解くための教師あり機械学習モデル(教師データから学習)
・入力は m 次元のベクトル
・出力である目的変数は 0 か 1 の値
・入力と m 次元パラメータの線形結合をシグモイド関数に入力。出力は y=1 になる確率の値
・シグモイド関数は微分をシグモイド関数自身で表すことができるため、尤度関数の微分を行う際、計算が容易
・最尤推定とは、データからそのデータを生成したであろう尤もらしい分布(パラメータ)の推定。尤度関数を最大化するようなパラメータを選ぶ推定方法。
・最尤推定の際は、対数をとると微分の計算が簡易になる。
・最尤法では、対数尤度関数をパラメータで微分して 0 になる値を求める必要があるが、解析的に求めるのは困難であるため、反復学習による勾配降下法が用いられることがある。
・確率的勾配降下法では、勾配降下法のパラメータを 1 回更新するのと同じ計算量でパラメータを n 回更新できるので効率よく最適な解を探索可能

3.2 実装演習結果

次ページ以降に演習の結果を掲載

編集するにはダブルクリックするか Enter キーを押してください

Googleドライブのマウント

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

0. データ表示

```
#from モジュール名 import クラス名（もしくは関数名や変数名）
import pandas as pd
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#matplotlibをinlineで表示するためのおまじない (plt.show()しなくていい)
%matplotlib inline
```

以下では，GoogleドライブのマイドライブにStudy_ai_mlフォルダを置くことを仮定しています．必要に応じて，パスを変更してください。

```
# titanic data csvファイルの読み込み
titanic_df = pd.read_csv('/content/drive/My Drive/study_ai_ml/data/titanic_train.csv')
```

```
# ファイルの先頭部を表示し、データセットを確認する
titanic_df.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th	female	38.0	1	0	PC 17599	71

1. ロジスティック回帰

不要なデータの削除・欠損値の補完

```
#予測に不要と考えるからうをドロップ（本当はここの情報もしっかり使うべきだと思っています）
titanic_df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

#一部カラムをドロップしたデータを表示
titanic_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

```
#nullを含んでいる行を表示
titanic_df[titanic_df.isnull().any(1)].head(10)
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
5	0	3	male	NaN	0	0	8.4583	Q
17	1	2	male	NaN	0	0	13.0000	S
19	1	3	female	NaN	0	0	7.2250	C
26	0	3	male	NaN	0	0	7.2250	C
28	1	3	female	NaN	0	0	7.8792	Q
29	0	3	male	NaN	0	0	7.8958	S
31	1	1	female	NaN	1	0	146.5208	C
32	1	3	female	NaN	0	0	7.7500	Q
36	1	3	male	NaN	0	0	7.2292	C
42	0	3	male	NaN	0	0	7.8958	C


```
#Ageカラムのnullを中央値で補完

titanic_df['AgeFill'] = titanic_df['Age'].fillna(titanic_df['Age'].mean())

#再度nullを含んでいる行を表示 (Ageのnullは補完されている)
titanic_df[titanic_df.isnull().any(1)]

#titanic_df.dtypes
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill
5	0	3	male	NaN	0	0	8.4583	Q	29.699118
17	1	2	male	NaN	0	0	13.0000	S	29.699118
19	1	3	female	NaN	0	0	7.2250	C	29.699118
26	0	3	male	NaN	0	0	7.2250	C	29.699118
28	1	3	female	NaN	0	0	7.8792	Q	29.699118
...	...	...	...	...	...	...	...	...	...
859	0	3	male	NaN	0	0	7.2292	C	29.699118
863	0	3	female	NaN	8	2	69.5500	S	29.699118
868	0	3	male	NaN	0	0	9.5000	S	29.699118
878	0	3	male	NaN	0	0	7.8958	S	29.699118
888	0	3	female	NaN	1	2	23.4500	S	29.699118

179 rows × 9 columns

1. ロジスティック回帰

実装(チケット価格から生死を判別)


```
#運賃だけのリストを作成
data1 = titanic_df.loc[:, ["Fare"]].values
```

```
#生死フラグのみのリストを作成
label1 =  titanic_df.loc[:,["Survived"]].values
```

```
from sklearn.linear_model import LogisticRegression
```

```
model=LogisticRegression()
```

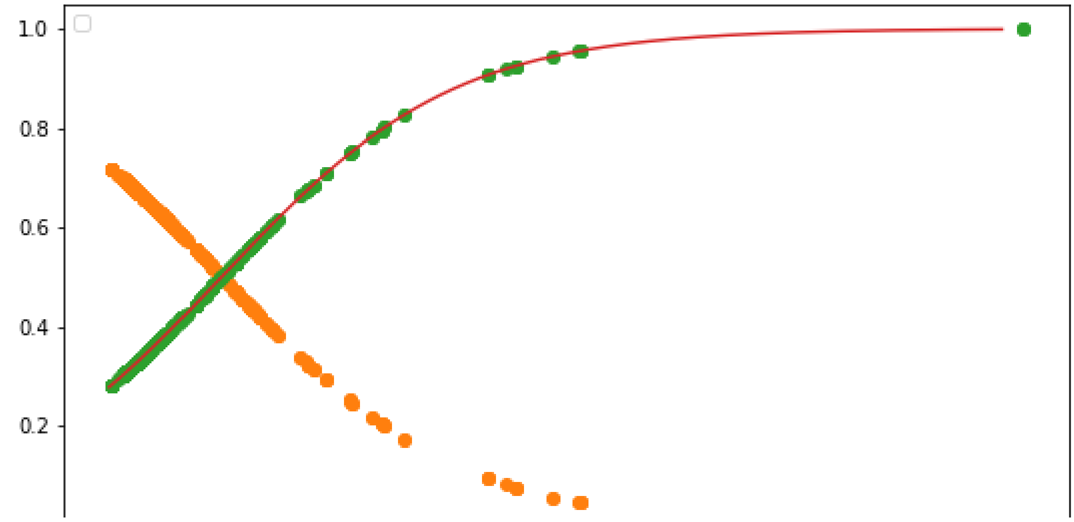
```
model.fit(data1, label1)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning
  y = column_or_1d(y, warn=True)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
model.predict([[61]])
```

```
    array([0])

model.predict_proba([[62]])

    array([[0.49978123, 0.50021877]])

X_test_value = model.decision_function(data1)

# # 決定関数値（絶対値が大きいほど識別境界から離れている）
# X_test_value = model.decision_function(X_test)
# # 決定関数値をシグモイド関数で確率に変換
# X_test_prob = normal_sigmoid(X_test_value)

print (model.intercept_)

print (model.coef_)

    [-0.94131796]
    [[0.01519666]]

w_0 = model.intercept_[0]
w_1 = model.coef_[0,0]

# def normal_sigmoid(x):
#     return 1 / (1+np.exp(-x))

def sigmoid(x):
    return 1 / (1+np.exp(-(w_1*x+w_0)))

x_range = np.linspace(-1, 500, 3000)

plt.figure(figsize=(9,5))
#plt.xkcd()
plt.legend(loc=2)


# plt.ylim(-0.1, 1.1)
# plt.xlim(-10, 10)

# plt.plot([-10,10],[0,0], "k", lw=1)
# plt.plot([0,0],[-1,1.5], "k", lw=1)
plt.plot(data1,np.zeros(len(data1)), 'o')
plt.plot(data1, model.predict_proba(data1), 'o')
plt.plot(x_range, sigmoid(x_range), '-')
#plt.plot(x_range, normal_sigmoid(x_range), '-')
#
```

```
No handles with labels found to put in legend.
[<matplotlib.lines.Line2D at 0x7fa779c1ded0>]
```



1. ロジスティック回帰

実装(2変数から生死を判別)

```
#AgeFillの欠損値を埋めたので
#titanic_df = titanic_df.drop(['Age'], axis=1)
```

```
titanic_df['Gender'] = titanic_df['Sex'].map({'female': 0, 'male': 1}).astype(int)
```

```
titanic_df.head(3)
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill	Gender
0	0	3	male	22.0	1	0	7.2500	S	22.0	1
1	1	1	female	38.0	1	0	71.2833	C	38.0	0
2	1	3	female	26.0	0	0	7.9250	S	26.0	0


```
titanic_df['Pclass_Gender'] = titanic_df['Pclass'] + titanic_df['Gender']
```

```
titanic_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill	Gender
0	0	3	male	22.0	1	0	7.2500	S	22.0	1
1	1	1	female	38.0	1	0	71.2833	C	38.0	0
2	1	3	female	26.0	0	0	7.9250	S	26.0	0
3	1	1	female	35.0	1	0	53.1000	S	35.0	0
4	0	3	male	35.0	0	0	8.0500	S	35.0	1

```
titanic_df = titanic_df.drop(['Pclass', 'Sex', 'Gender','Age'], axis=1)
```

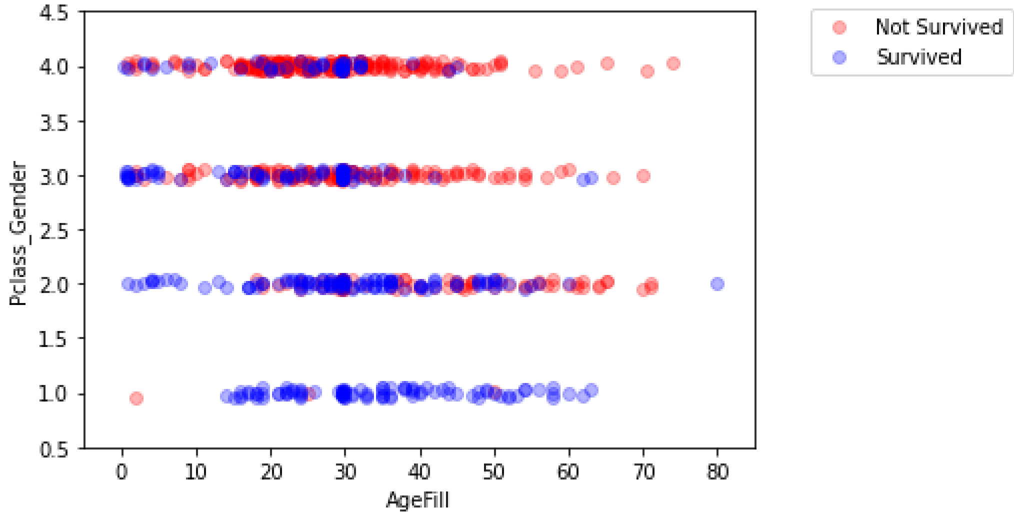
```
titanic_df.head()
```

	Survived	SibSp	Parch	Fare	Embarked	AgeFill	Pclass_Gender
0	0	1	0	7.2500	S	22.0	4
1	1	1	0	71.2833	C	38.0	1
2	1	0	0	7.9250	S	26.0	3
3	1	1	0	53.1000	S	35.0	1
4	0	0	0	8.0500	S	35.0	4

```
# 重要だよ！！！
# 境界線の式
#   w_1・x + w_2・y + w_0 = 0
#   ⇒ y = (-w_1・x - w_0) / w_2

# # 境界線 プロット
# plt.plot([-2,2], map(lambda x: (-w_1 * x - w_0)/w_2, [-2,2]))

# # データを重ねる
# plt.scatter(X_train_std[y_train==0, 0], X_train_std[y_train==0, 1], c='red', marker='x', label='t
# plt.scatter(X_train_std[y_train==1, 0], X_train_std[y_train==1, 1], c='blue', marker='x', label='
# plt.scatter(X_test_std[y_test==0, 0], X_test_std[y_test==0, 1], c='red', marker='o', s=60, label=
# plt.scatter(X_test_std[y_test==1, 0], X_test_std[y_test==1, 1], c='blue', marker='o', s=60, label
```

```
np.random.seed = 0

xmin, xmax = -5, 85
ymin, ymax = 0.5, 4.5

index_survived = titanic_df[titanic_df["Survived"]==0].index
index_notsurvived = titanic_df[titanic_df["Survived"]==1].index

from matplotlib.colors import ListedColormap
fig, ax = plt.subplots()
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
sc = ax.scatter(titanic_df.loc[index_survived, 'AgeFill'],
                titanic_df.loc[index_survived, 'Pclass_Gender']+(np.random.rand(len(index_survived)
                color='r', label='Not Survived', alpha=0.3)
sc = ax.scatter(titanic_df.loc[index_notsurvived, 'AgeFill'],
                titanic_df.loc[index_notsurvived, 'Pclass_Gender']+(np.random.rand(len(index_notsur
                color='b', label='Survived', alpha=0.3)
ax.set_xlabel('AgeFill')
ax.set_ylabel('Pclass_Gender')
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
ax.legend(bbox_to_anchor=(1.4, 1.03))
```

```
<matplotlib.legend.Legend at 0x7fa7788b5490>
```




```
#運賃だけのリストを作成
data2 = titanic_df.loc[:, ["AgeFill", "Pclass_Gender"]].values
```

```
data2
```

```
array([[22.        ,  4.        ],
       [38.        ,  1.        ],
       [26.        ,  3.        ],
       ...,
       [29.69911765,  3.        ],
       [26.        ,  2.        ],
       [32.        ,  4.        ]])
```

```
#生死フラグのみのリストを作成
label2 =  titanic_df.loc[:,["Survived"]].values
```

```
model2 = LogisticRegression()
```

```
model2.fit(data2, label2)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning
  y = column_or_1d(y, warn=True)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
model2.predict([[10,1]])
```

```
array([1])
```

```
model2.predict_proba([[10,1]])
```

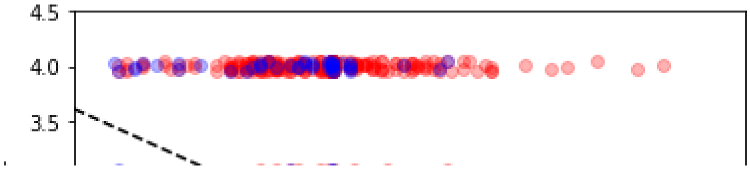
```
array([[0.03754749, 0.96245251]])
```

```
titanic_df.head(3)
```

	Survived	SibSp	Parch	Fare	Embarked	AgeFill	Pclass_Gender
0	0	1	0	7.2500	S	22.0	4
1	1	1	0	71.2833	C	38.0	1
2	1	0	0	7.9250	S	26.0	3

```
h = 0.02
xmin, xmax = -5, 85
ymin, ymax = 0.5, 4.5
xx, yy = np.meshgrid(np.arange(xmin, xmax, h), np.arange(ymin, ymax, h))
Z = model2.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
Z = Z.reshape(xx.shape)

fig, ax = plt.subplots()
levels = np.linspace(0, 1.0)
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
#contour = ax.contourf(xx, yy, Z, cmap=cm, levels=levels, alpha=0.5)

sc = ax.scatter(titanic_df.loc[index_survived, 'AgeFill'],
                titanic_df.loc[index_survived, 'Pclass_Gender']+(np.random.rand(len(index_survived)
                color='r', label='Not Survived', alpha=0.3)
sc = ax.scatter(titanic_df.loc[index_notsurvived, 'AgeFill'],
                titanic_df.loc[index_notsurvived, 'Pclass_Gender']+(np.random.rand(len(index_notsur
                color='b', label='Survived', alpha=0.3)

ax.set_xlabel('AgeFill')
ax.set_ylabel('Pclass_Gender')
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
#fig.colorbar(contour)

x1 = xmin
x2 = xmax
y1 = -1*(model2.intercept_[0]+model2.coef_[0][0]*xmin)/model2.coef_[0][1]
y2 = -1*(model2.intercept_[0]+model2.coef_[0][0]*xmax)/model2.coef_[0][1]
ax.plot([x1, x2] ,[y1, y2], 'k--')
```

```
[<matplotlib.lines.Line2D at 0x7fa77886a290>]
```



2. モデル評価

混同行列とクロスバリデーション

```
from sklearn.model_selection import train_test_split
```



```
traindata1, testdata1, trainlabel1, testlabel1 = train_test_split(data1, label1, test_size=0.2)
traindata1.shape
trainlabel1.shape
```

```
(712, 1)
```

```
traindata2, testdata2, trainlabel2, testlabel2 = train_test_split(data2, label2, test_size=0.2)
traindata2.shape
trainlabel2.shape
#本来は同じデータセットを分割しなければいけない。(簡易的に別々に分割している。)
```

```
(712, 1)
```

```
data = titanic_df.loc[:, ].values
label =  titanic_df.loc[:,["Survived"]].values
traindata, testdata, trainlabel, testlabel = train_test_split(data, label, test_size=0.2)
traindata.shape
trainlabel.shape
```

```
(712, 1)
```

```
eval_model1=LogisticRegression()
eval_model2=LogisticRegression()
#eval_model=LogisticRegression()
```

```
predictor_eval1=eval_model1.fit(traindata1, trainlabel1).predict(testdata1)
predictor_eval2=eval_model2.fit(traindata2, trainlabel2).predict(testdata2)
#predictor_eval=eval_model.fit(traindata, trainlabel).predict(testdata)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning
  y = column_or_1d(y, warn=True)
```

```
eval_model1.score(traindata1, trainlabel1)
```

```
0.6587078651685393
```

```
eval_model1.score(testdata1,testlabel1)
```

```
0.7374301675977654
```

```
eval_model2.score(traindata2, trainlabel2)
```

```
0.7794943820224719
```

```
eval_model2.score(testdata2,testlabel2)
```

```
0.7262569832402235
```

```
from sklearn import metrics
print(metrics.classification_report(testlabel1, predictor_eval1))
print(metrics.classification_report(testlabel2, predictor_eval2))
```

```
              precision    recall  f1-score   support

           0       0.76      0.90      0.82       122
           1       0.65      0.39      0.48        57

    accuracy                           0.74       179
   macro avg       0.70      0.64      0.65       179
weighted avg       0.72      0.74      0.72       179

              precision    recall  f1-score   support

           0       0.75      0.83      0.79       111
           1       0.67      0.56      0.61        68

    accuracy                           0.73       179
   macro avg       0.71      0.69      0.70       179
weighted avg       0.72      0.73      0.72       179
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix1=confusion_matrix(testlabel1, predictor_eval1)
confusion_matrix2=confusion_matrix(testlabel2, predictor_eval2)
```

```
confusion_matrix1
```

```
array([[110,  12],
       [ 35,  22]])
```

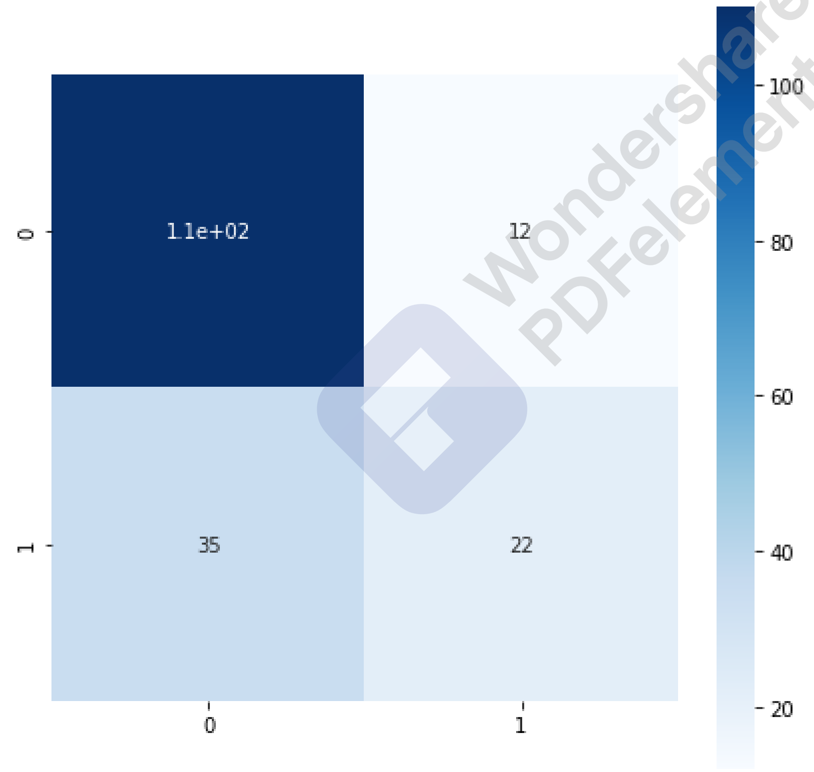
```
confusion_matrix2
```

```
array([[92, 19],
       [30, 38]])
```

```
fig = plt.figure(figsize = (7,7))
```

```
#plt.title(title)
sns.heatmap(
    confusion_matrix1,
    vmin=None,
    vmax=None,
    cmap="Blues",
    center=None,
    robust=False,
    annot=True, fmt='.2g',
    annot_kws=None,
    linewidths=0,
    linecolor='white',
    cbar=True,
    cbar_kws=None,
    cbar_ax=None,
    square=True, ax=None,
    #xticklabels=columns,
    #yticklabels=columns,
    mask=None)
```

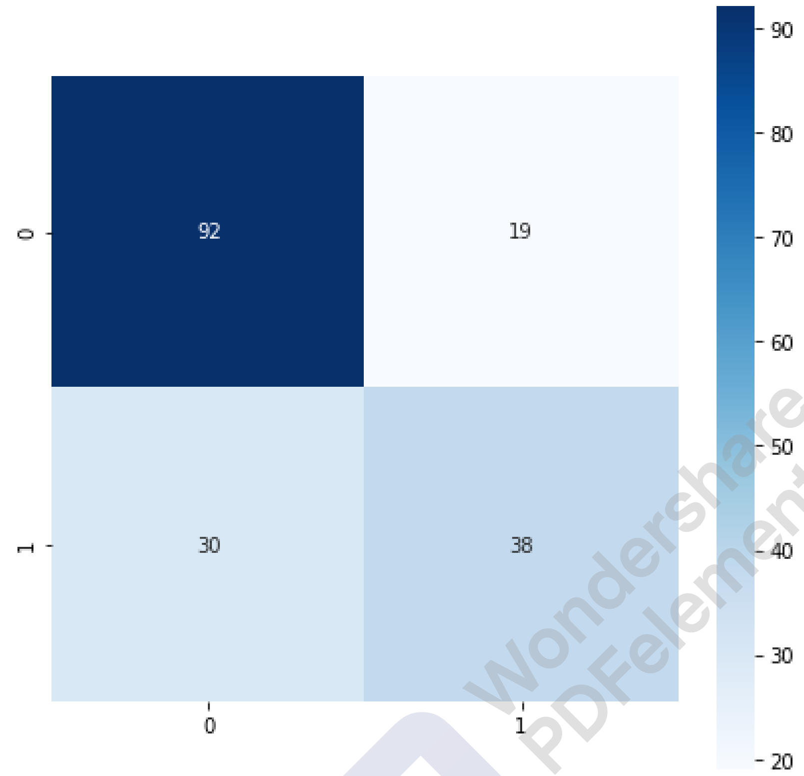
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa76ffb1d10>
```




```
fig = plt.figure(figsize = (7,7))
#plt.title(title)
sns.heatmap(
    confusion_matrix2,
    vmin=None,
    vmax=None,
    cmap="Blues",
    center=None,
    robust=False,
    annot=True, fmt='.2g',
    annot_kws=None,
    linewidths=0,
```

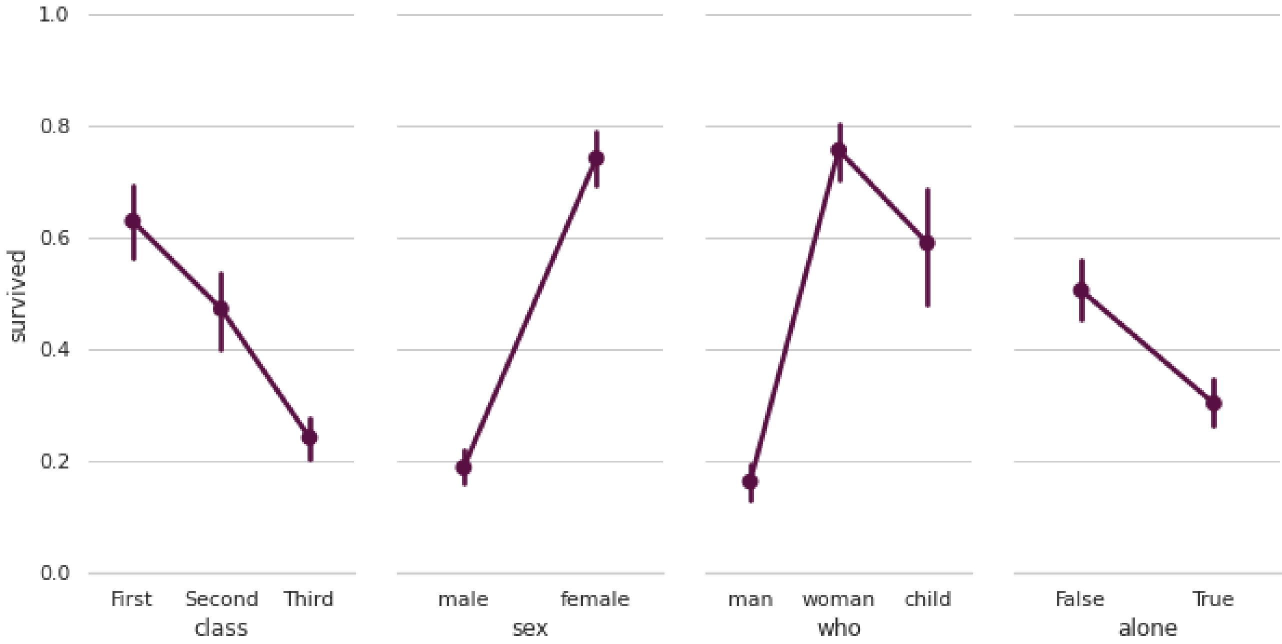
```
        linewidths=0,
        linecolor='white',
        cbar=True,
        cbar_kws=None,
        cbar_ax=None,
        square=True, ax=None,
        #xticklabels=columns,
        #yticklabels=columns,
        mask=None)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa76fe66c90>
```



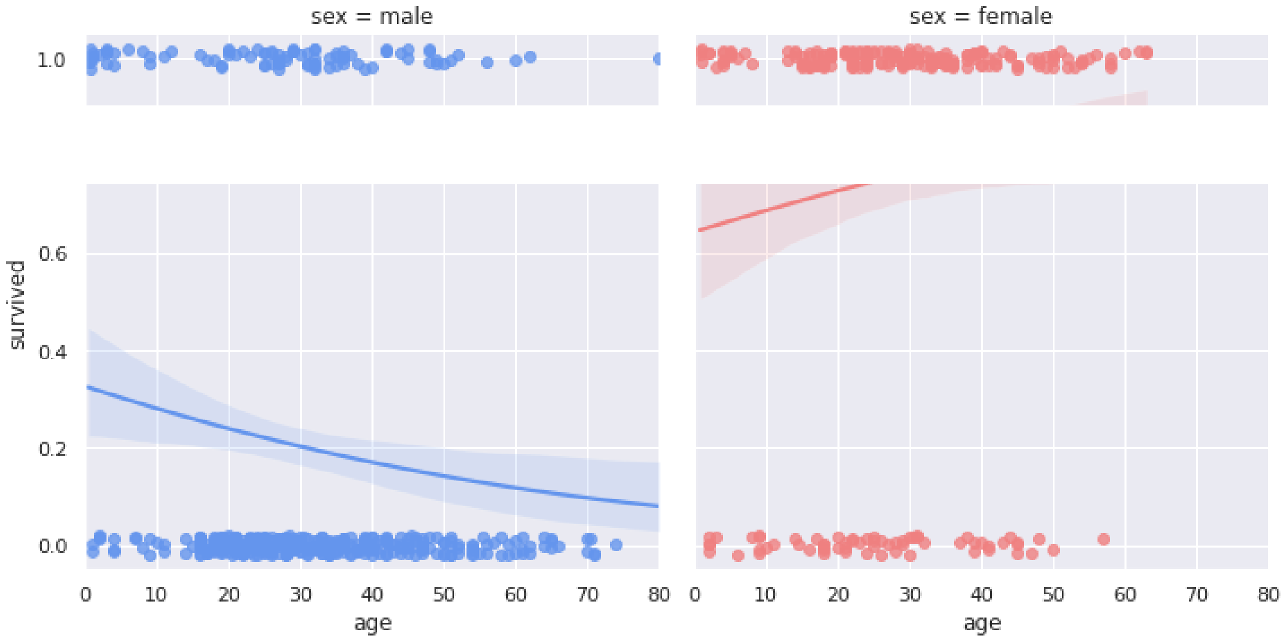

```
#Paired categorical plots

import seaborn as sns
sns.set(style="whitegrid")

# Load the example Titanic dataset
titanic = sns.load_dataset("titanic")

# Set up a grid to plot survival probability against several variables
g = sns.PairGrid(titanic, y_vars="survived",
                 x_vars=["class", "sex", "who", "alone"],
                 size=5, aspect=.5)

# Draw a seaborn pointplot onto each Axes
g.map(sns.pointplot, color=sns.xkcd_rgb["plum"])
g.set(ylim=(0, 1))
sns.despine(fig=g.fig, left=True)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:1152: UserWarning: The size para
  warnings.warn(UserWarning(msg))
```




```
#Faceted logistic regression

import seaborn as sns
sns.set(style="darkgrid")

# Load the example titanic dataset
df = sns.load_dataset("titanic")

# Make a custom palette with gendered colors
pal = dict(male="#6495ED", female="#F08080")

# Show the survival proability as a function of age and sex
g = sns.lmplot(x="age", y="survived", col="sex", hue="sex", data=df,
               palette=pal, y_jitter=.02, logistic=True)
g.set(xlim=(0, 80), ylim=(-.05, 1.05))
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: panda
  import pandas.util.testing as tm
```



✓ 0 秒 完了時間: 22:40

4　主成分分析

4.1　要点のまとめ

・多変量データの持つ構造をより少数個の指標に圧縮。変量の個数を減らし、情報の損失はなるべく小さくする。
・主成分分析を行うことで、少数変数を利用した分析や可視化(2・3 次元の場合)が実現可能。
・情報の量を分散の大きさと捉える。
・係数ベクトルが変われば線形変換後の値が変化
・線形変換後の変数の分散が最大となる射影軸を探索
・ノルムが 1 となる制約を入れた最適化問題を解く。
・ラグランジュ関数を微分して最適解を求める。
・寄与率とは第 k 主成分の分散の全分散に対する割合(第 k 主成分が持つ情報量の割合)。

4.2　実装演習結果

次ページ以降に演習の結果を掲載

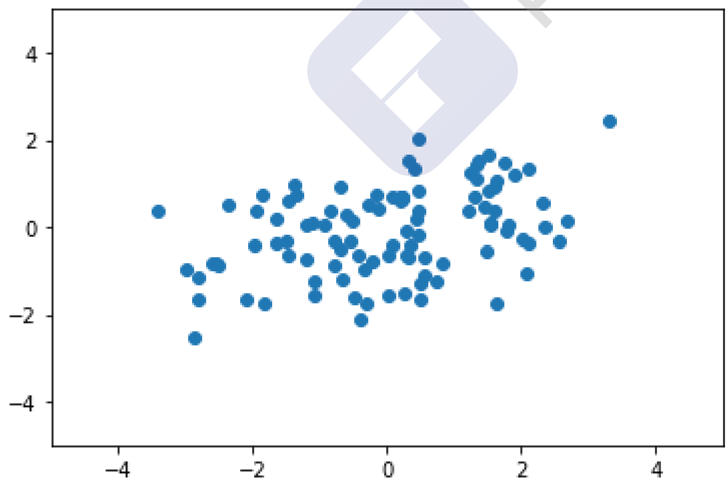
主成分分析

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

訓練データ生成

```
n_sample = 100

def gen_data(n_sample):
    mean = [0, 0]
    cov = [[2, 0.7], [0.7, 1]]
    return np.random.multivariate_normal(mean, cov, n_sample)

def plt_data(X):
    plt.scatter(X[:, 0], X[:, 1])
    plt.xlim(-5, 5)
    plt.ylim(-5, 5)
```

```
X = gen_data(n_sample)
plt_data(X)
```



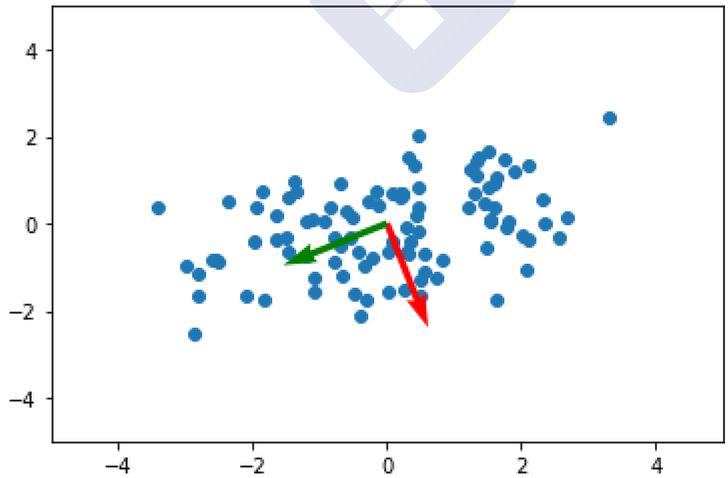
学習

訓練データ$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^{\mathrm{T}}$に対して$\mathbb{E}[\boldsymbol{x}] = \mathbf{0}$となるように変換する。

すると、不偏共分散行列は$Var[\boldsymbol{x}] = \frac{1}{n-1}X^{\mathrm{T}}X$と書ける。

$Var[\boldsymbol{x}]$を固有値分解し、固有値の大きい順に対応する固有ベクトルを第1主成分$(\boldsymbol{w}_1)$, 第2主成分$(\boldsymbol{w}_2)$, ...とよぶ。

```
n_components=2

def get_moments(X):
    mean = X.mean(axis=0)
    stan_cov = np.dot((X - mean).T, X - mean) / (len(X) - 1)
    return mean, stan_cov

def get_components(eigenvectors, n_components):
#     W = eigenvectors[:, -n_components:]
#     return W.T[::-1]
    W = eigenvectors[:, ::-1][:, :n_components]
    return W.T

def plt_result(X, first, second):
    plt.scatter(X[:, 0], X[:, 1])
    plt.xlim(-5, 5)
    plt.ylim(-5, 5)
    # 第1主成分
    plt.quiver(0, 0, first[0], first[1], width=0.01, scale=6, color='red')
    # 第2主成分
    plt.quiver(0, 0, second[0], second[1], width=0.01, scale=6, color='green')


#分散共分散行列を標準化
meean, stan_cov = get_moments(X)
#固有値と固有ベクトルを計算
eigenvalues, eigenvectors = np.linalg.eigh(stan_cov)
components = get_components(eigenvectors, n_components)

plt_result(X, eigenvectors[0, :], eigenvectors[1, :])
```



変換（射影）

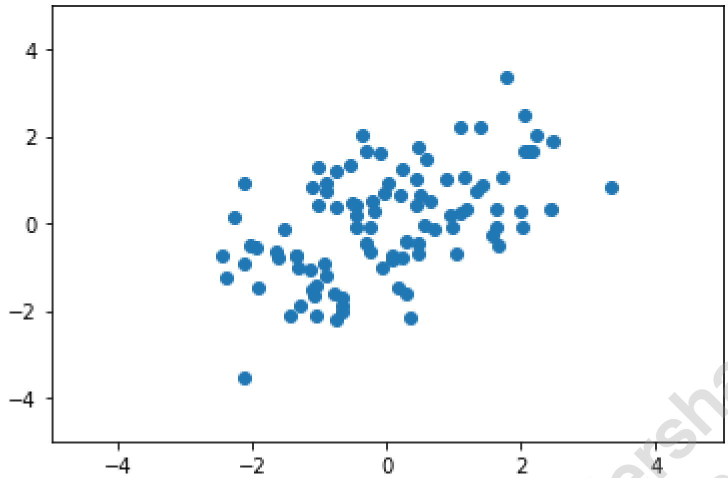
元のデータをm次元に変換(射影)するときは行列Wを$W = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_m]$とし、データ点$\boldsymbol{x}$を$\boldsymbol{z} = W^{\mathrm{T}}\boldsymbol{x}$によって変換(射影)する。

よって、データXに対しては$Z = X^{\mathrm{T}}W$によって変換する。

```
def transform_by_pca(X, pca):
    mean = X.mean(axis=0)
    return np.dot(X-mean, components)
```

```
Z = transform_by_pca(X, components.T)
plt.scatter(Z[:, 0], Z[:, 1])
plt.xlim(-5, 5)
plt.ylim(-5, 5)
```

(-5.0, 5.0)



逆変換

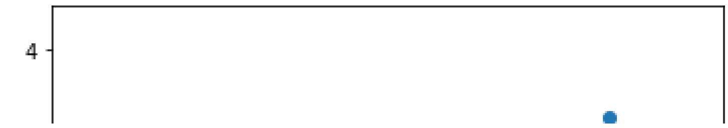
射影されたデータ点$\boldsymbol{z}$を元のデータ空間へ逆変換するときは$\bar{\boldsymbol{x}} = (W^{\mathrm{T}})^{-1}\boldsymbol{z} = W\boldsymbol{z}$によって変換する。

よって、射影されたデータZに対しては$\bar{X} = ZW^{\mathrm{T}}$によって変換する。

```
mean = X.mean(axis=0)
X_ = np.dot(Z, components.T) + mean
```

```
plt.scatter(X_[:, 0], X_[:, 1])
plt.xlim(-5, 5)
plt.ylim(-5, 5)
```

(-5.0, 5.0)



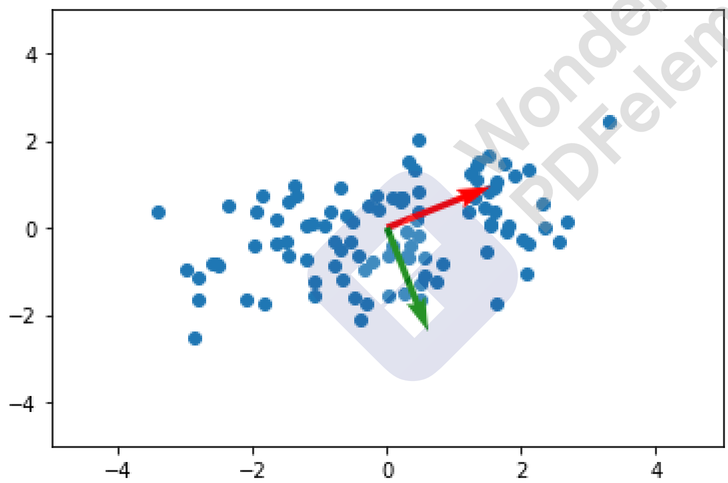

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
print('components: {}'.format(pca.components_))
print('mean: {}'.format(pca.mean_))
print('covariance: {}'.format(pca.get_covariance()))
```

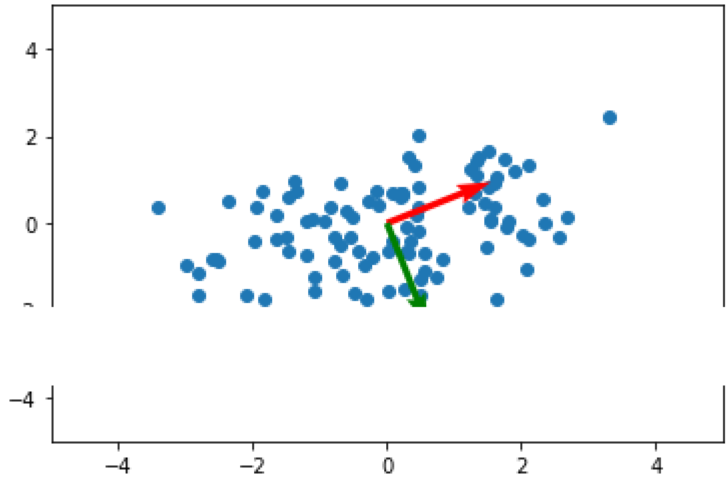
```
components: [[ 0.92941442  0.36903772]
 [ 0.36903772 -0.92941442]]
mean: [ 0.00713757 -0.09127978]
covariance: [[2.24383764 0.58343668]
 [0.58343668 1.0061257 ]]
```

```
plt_result(X, pca.components_[0, :], pca.components_[1, :])
```




```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
plt_result(X, pca.components_[0, :], pca.components_[1, :])

print('components: {}'.format(pca.components_))
print('mean: {}'.format(pca.mean_))
print('covariance: {}'.format(pca.get_covariance()))
```

```
components: [[ 0.92941442  0.36903772]
 [ 0.36903772 -0.92941442]]
mean: [ 0.00713757 -0.09127978]
covariance: [[2.24383764 0.58343668]
 [0.58343668 1.0061257 ]]
```



✓ 0 秒 完了時間: 23:56

機械学習レポート

5　アルゴリズム

5.1　要点のまとめ

5.1.1 k 近傍法

- 分類問題のための機械学習手法
- 最近傍のデータを k 個取り、それらがもっとも多く所属するクラスに識別する。
- k を変化させると結果も変わる。

5.1.2 k-平均法(k-means)

・教師なし学習。クラスタリング手法
・与えられたデータを k 個のクラスタに分類するクラスタリング
・特徴の似ているもの同士をグループ化 k-平均法(k-means)
・アルゴリズムの手順
① 各クラスタ中心の初期値を設定する
② 各データ点に対して、各クラスタ中心との距離を計算し、最も距離が近いクラスタを割り当てる
③ 各クラスタの平均ベクトル（中心）を計算する
④ 収束するまで 2, 3 の処理を繰り返す
・中心の初期値を変えるとクラスタリング結果も変わる
・k の値を変えるとクラスタリング結果も変わる

5.2　実装演習結果

次ページ以降に k 近傍法、k-平均法演習の結果を掲載

k近傍法

＋ コード ＋ テキスト

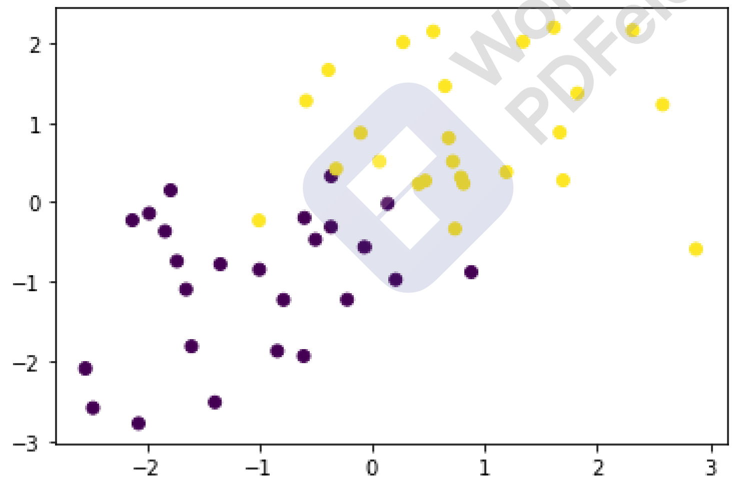
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

訓練データ生成

```
def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 1
    x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
    x_train = np.concatenate([x0, x1])
    y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
    return x_train, y_train
```

```
X_train, ys_train = gen_data()
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
```

```
<matplotlib.collections.PathCollection at 0x7f883db9d5d0>
```



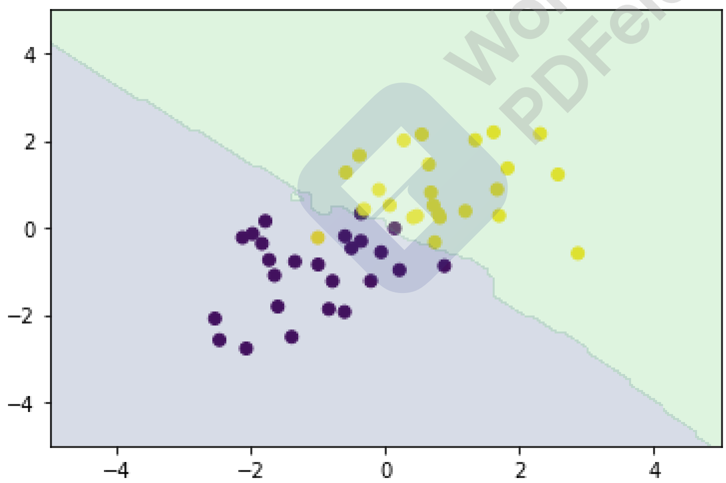
学習

陽に訓練ステップはない

予測

予測するデータ点との、距離が最も近いk個の、訓練データのラベルの最頻値を割り当てる

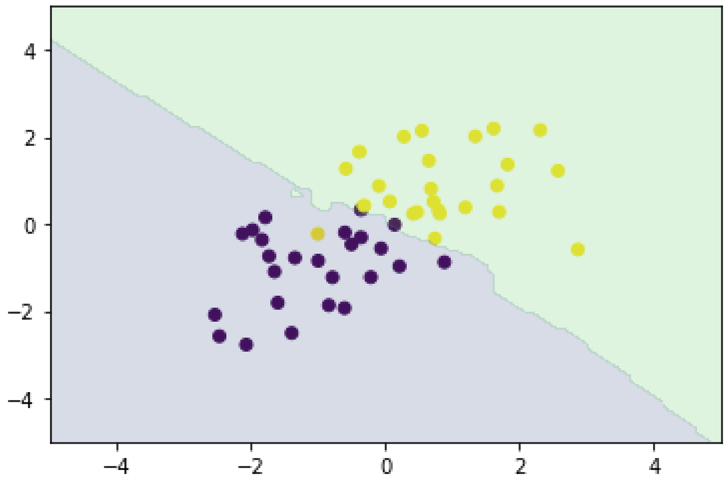
```
def distance(x1, x2):
    return np.sum((x1 - x2)**2, axis=1)

def knc_predict(n_neighbors, x_train, y_train, X_test):
    y_pred = np.empty(len(X_test), dtype=y_train.dtype)
    for i, x in enumerate(X_test):
        distances = distance(x, X_train)
        nearest_index = distances.argsort()[:n_neighbors]
        mode, _ = stats.mode(y_train[nearest_index])
        y_pred[i] = mode
    return y_pred

def plt_resut(x_train, y_train, y_pred):
    xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
    xx = np.array([xx0, xx1]).reshape(2, -1).T
    plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
    plt.contourf(xx0, xx1, y_pred.reshape(100, 100).astype(dtype=np.float), alpha=0.2, levels=np.li


n_neighbors = 3

xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
X_test = np.array([xx0, xx1]).reshape(2, -1).T

y_pred = knc_predict(n_neighbors, X_train, ys_train, X_test)
plt_resut(X_train, ys_train, y_pred)
```



numpy実装

```
xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T
```

```
from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X_train, ys_train)
plt_resut(X_train, ys_train, knc.predict(xx))
```



✓ 0 秒 完了時間: 0:00

k平均クラスタリング(k-means)

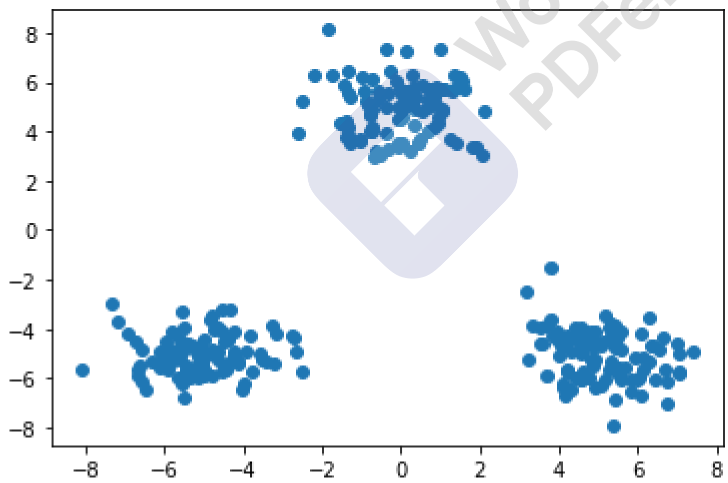
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

データ生成

```
def gen_data():
    x1 = np.random.normal(size=(100, 2)) + np.array([-5, -5])
    x2 = np.random.normal(size=(100, 2)) + np.array([5, -5])
    x3 = np.random.normal(size=(100, 2)) + np.array([0, 5])
    return np.vstack((x1, x2, x3))
```

```
#データ作成
X_train = gen_data()
#データ描画
plt.scatter(X_train[:, 0], X_train[:, 1])
```

```
<matplotlib.collections.PathCollection at 0x7f1df609d2d0>
```



学習

k-meansアルゴリズムは以下のとおりである

1) 各クラスタ中心の初期値を設定する

2) 各データ点に対して、各クラスタ中心との距離を計算し、最も距離が近いクラスタを割り当てる

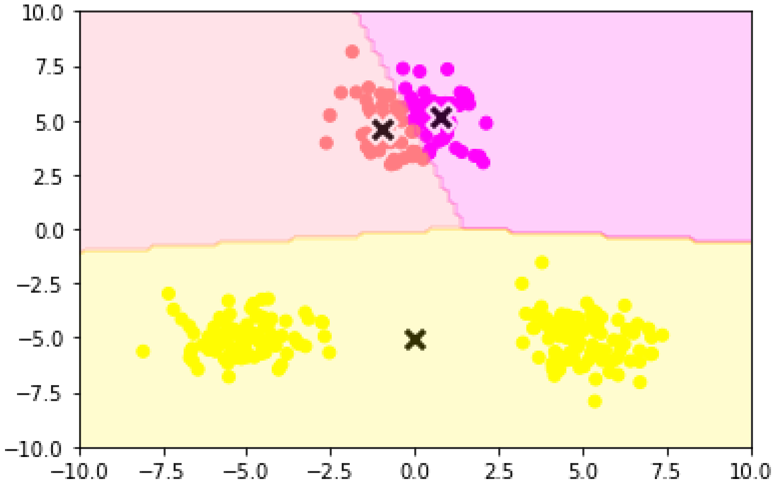
3) 各クラスタの平均ベクトル（中心）を計算する

4) 収束するまで2, 3の処理を繰り返す

```
def distance(x1, x2):
    return np.sum((x1 - x2)**2, axis=1)

n_clusters = 3
iter_max = 100

# 各クラスタ中心をランダムに初期化
centers = X_train[np.random.choice(len(X_train), n_clusters, replace=False)]

for _ in range(iter_max):
    prev_centers = np.copy(centers)
    D = np.zeros((len(X_train), n_clusters))
    # 各データ点に対して、各クラスタ中心との距離を計算
    for i, x in enumerate(X_train):
        D[i] = distance(x, centers)
    # 各データ点に、最も距離が近いクラスタを割り当
    cluster_index = np.argmin(D, axis=1)
    # 各クラスタの中心を計算
    for k in range(n_clusters):
        index_k = cluster_index == k
        centers[k] = np.mean(X_train[index_k], axis=0)
    # 収束判定
    if np.allclose(prev_centers, centers):
        break
```

クラスタリング結果

```
def plt_result(X_train, centers, xx):
    #  データを可視化
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_pred, cmap='spring')
    # 中心を可視化
    plt.scatter(centers[:, 0], centers[:, 1], s=200, marker='X', lw=2, c='black', edgecolor="white"
    # 領域の可視化
    pred = np.empty(len(xx), dtype=int)
    for i, x in enumerate(xx):
        d = distance(x, centers)
        pred[i] = np.argmin(d)
    plt.contourf(xx0, xx1, pred.reshape(100, 100), alpha=0.2, cmap='spring')


y_pred = np.empty(len(X_train), dtype=int)
for i, x in enumerate(X_train):
    d = distance(x, centers)
    y_pred[i] = np.argmin(d)


xx0, xx1 = np.meshgrid(np.linspace(-10, 10, 100), np.linspace(-10, 10, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

plt_result(X_train, centers, xx)
```



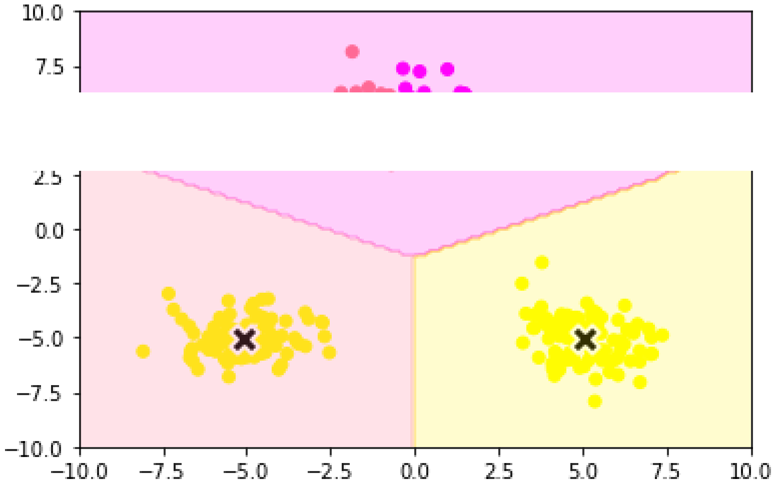
numpy実装

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X_train)
```

```
print("labels: {}".format(kmeans.labels_))
print("cluster_centers: {}".format(kmeans.cluster_centers_))
kmeans.cluster_centers_
```

```
labels: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0]
cluster_centers: [[ 0.01086921  4.95073575]
 [-5.07296767 -5.043214  ]
 [ 5.07255324 -5.01024348]]
array([[ 0.01086921,  4.95073575],
       [-5.07296767, -5.043214  ],
       [ 5.07255324, -5.01024348]])
```

```
plt_result(X_train, kmeans.cluster_centers_, xx)
```



✓ 0 秒 完了時間: 0:02

6　サポートベクタマシン(SVM)

6.1　要点のまとめ

・オリジナルの SVM が対象とする 2 クラス分類問題とは「与えられた入力データが 2 つのカテゴリーのどちらに属するかを識別する問題

・一般に 2 クラス分類問題では、特徴ベクトル x がどちらのクラスに属するか判定するため決定関数が使用される。

・分類境界を挟んで 2 つのクラスがどのくらい離れているかをマージンと呼ぶ。SVM ではこのマージンの最大化を図る。

・完全に分離できると仮定した SV 分類のことをハードマージンという。

・SV 分類を分離可能でないデータに適用できるように拡張したものをソフトマージンという。

・主問題と比べて双対問題の方が変数を少なくできるメリットがある。

6.2　要点のまとめ

実装演習結果

次ページ以降に演習の結果を掲載

サポートベクターマシン(SVM)

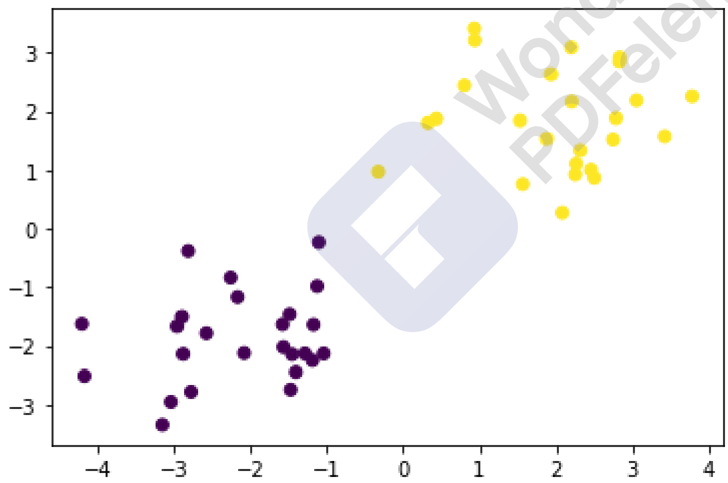
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

訓練データ生成①（線形分離可能）

```
def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 2.
    x1 = np.random.normal(size=50).reshape(-1, 2) + 2.
    X_train = np.concatenate([x0, x1])
    ys_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
    return X_train, ys_train
```

```
X_train, ys_train = gen_data()
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
```

```
<matplotlib.collections.PathCollection at 0x7f8a9aef61d0>
```



学習

特徴空間上で線形なモデル$y(\boldsymbol{x}) = \boldsymbol{w}\phi(\boldsymbol{x}) + b$を用い、その正負によって2値分類を行うことを考える。

サポートベクトターマシンではマージンの最大化を行うが、それは結局以下の最適化問題を解くことと同じである。

ただし、訓練データを$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^{\mathrm{T}}, \boldsymbol{t} = [t_1, t_2, \ldots, t_n]^{\mathrm{T}} (t_i = \{-1, +1\})$とする。

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}||\boldsymbol{w}||^2$$
$$\text{subject to} \quad t_i(\boldsymbol{w}\phi(\boldsymbol{x}_i)+b) \geq 1 \quad (i=1,2,\cdots,n)$$

ラグランジュ乗数法を使うと、上の最適化問題はラグランジュ乗数$\boldsymbol{a}(\geq 0)$を用いて、以下の目的関数を最小化する問題となる。

$$L(\boldsymbol{w},b,\boldsymbol{a}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{n} a_i t_i(\boldsymbol{w}\phi(\boldsymbol{x}_i)+b-1) \qquad \cdots(1)$$

目的関数が最小となるのは、$\boldsymbol{w},b$に関して偏微分した値が0となるときなので、

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} a_i t_i \phi(\boldsymbol{x}_i) = \boldsymbol{0}$$
$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} a_i t_i = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{t} = 0$$

これを式(1)　に代入することで、最適化問題は結局以下の目的関数の最大化となる。

$$\begin{aligned}\tilde{L}(\boldsymbol{a}) &= \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j) \\ &= \boldsymbol{a}^{\mathrm{T}}\boldsymbol{1} - \frac{1}{2}\boldsymbol{a}^{\mathrm{T}}H\boldsymbol{a}\end{aligned}$$

ただし、行列Hのi行j列成分は$H_{ij} = t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j) = t_i t_j k(\boldsymbol{x}_i,\boldsymbol{x}_j)$である。また制約条件は、$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t} = 0(\frac{1}{2}||\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t}||^2 = 0)$である。

この最適化問題を最急降下法で解く。目的関数と制約条件を$\boldsymbol{a}$で微分すると、

$$\frac{d\tilde{L}}{d\boldsymbol{a}} = \boldsymbol{1} - H\boldsymbol{a}$$
$$\frac{d}{d\boldsymbol{a}}(\frac{1}{2}||\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t}||^2) = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t})\boldsymbol{t}$$

なので、$\boldsymbol{a}$を以下の二式で更新する。

$$\boldsymbol{a} \leftarrow \boldsymbol{a} + \eta_1(\boldsymbol{1} - H\boldsymbol{a})$$
$$\boldsymbol{a} \leftarrow \boldsymbol{a} - \eta_2(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t})\boldsymbol{t}$$

```
t = np.where(ys_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

eta1 = 0.01
eta2 = 0.001
n_iter = 500

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.where(a > 0, a, 0)
```

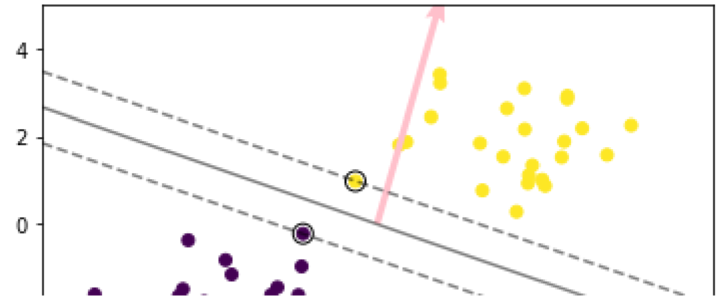
予測

新しいデータ点$\boldsymbol{x}$に対しては、$y(\boldsymbol{x}) = \boldsymbol{w}\phi(\boldsymbol{x}) + b = \sum_{i=1}^{n} a_i t_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b$の正負によって分類する。

ここで、最適化の結果得られた$a_i (i = 1, 2, \ldots, n)$の中で$a_i = 0$に対応するデータ点は予測に影響を与えないので、$a_i > 0$に対応するデータ点（サポートベクトル）のみ保持しておく。bはサポートベクトルのインデックスの集合をSとすると、
$b = \frac{1}{S} \sum_{s \in S} (t_s - \sum_{i=1}^{n} a_i t_i k(\boldsymbol{x}_i, \boldsymbol{x}_s))$によって求める。

```
index = a > 1e-6
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()


xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * sv.dot(X_test[i])
y_pred = np.sign(y_project)


# 訓練データを可視化
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
                    s=100, facecolors='none', edgecolors='k')
# 領域を可視化
#plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
                     levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])


# マージンと決定境界を可視化
plt.quiver(0, 0, 0.1, 0.35, width=0.01, scale=1, color='pink')
```

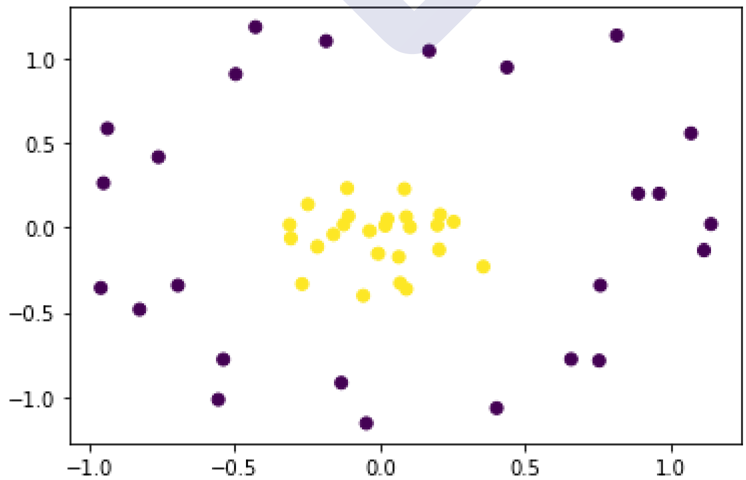
<matplotlib.quiver.Quiver at 0x7f8a8c8e74d0>



訓練データ生成②（線形分離不可能）

```
factor = .2
n_samples = 50
linspace = np.linspace(0, 2 * np.pi, n_samples // 2 + 1)[:-1]
outer_circ_x = np.cos(linspace)
outer_circ_y = np.sin(linspace)
inner_circ_x = outer_circ_x * factor
inner_circ_y = outer_circ_y * factor

X = np.vstack((np.append(outer_circ_x, inner_circ_x),
               np.append(outer_circ_y, inner_circ_y))).T
y = np.hstack([np.zeros(n_samples // 2, dtype=np.intp),
               np.ones(n_samples // 2, dtype=np.intp)])
X += np.random.normal(scale=0.15, size=X.shape)
x_train = X
y_train = y


plt.scatter(x_train[:,0], x_train[:,1], c=y_train)
```

<matplotlib.collections.PathCollection at 0x7f8a8c63fc10>



学習

元のデータ空間では線形分離は出来ないが、特徴空間上で線形分離することを考える。

今回はカーネルとしてRBFカーネル（ガウシアンカーネル）を利用する。

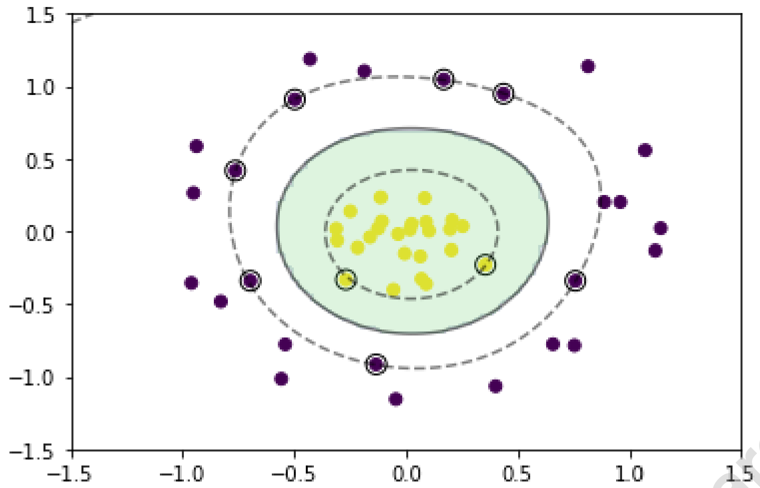
```
def rbf(u, v):
        sigma = 0.8
        return np.exp(-0.5 * ((u - v)**2).sum() / sigma**2)

X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# RBFカーネル
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        K[i, j] = rbf(X_train[i], X_train[j])

eta1 = 0.01
eta2 = 0.001
n_iter = 5000

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.where(a > 0, a, 0)
```


予測

```
index = a > 1e-6
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()


xx0, xx1 = np.meshgrid(np.linspace(-1.5, 1.5, 100), np.linspace(-1.5, 1.5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * rbf(X_test[i], sv)
y_pred = np.sign(y_project)


# 訓練データを可視化
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
# サポートベクトルを可視化
```

```
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
                    s=100, facecolors='none', edgecolors='k')
# 領域を可視化
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
                    levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

```
<matplotlib.contour.QuadContourSet at 0x7f8a8c65e210>
```



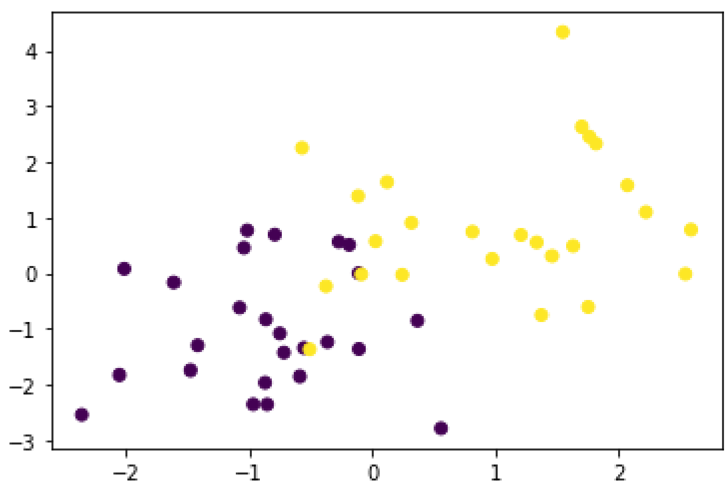
ソフトマージンSVM

訓練データ生成③（重なりあり）

```
x0 = np.random.normal(size=50).reshape(-1, 2) - 1.
x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
x_train = np.concatenate([x0, x1])
y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
```

```
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
```

```
<matplotlib.collections.PathCollection at 0x7f8a8c5aa4d0>
```



学習

分離不可能な場合は学習できないが、データ点がマージン内部に入ることや誤分類を許容することでその問題を回避する。

スラック変数$\xi_i \geq 0$を導入し、マージン内部に入ったり誤分類された点に対しては、$\xi_i = |1 - t_i y(\boldsymbol{x}_i)|$とし、これらを許容する代わりに対して、ペナルティを与えるように、最適化問題を以下のように修正する。

$$\begin{array}{ll} \min_{\boldsymbol{w},b} & \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^n \xi_i \\ \text{subject to} & t_i(\boldsymbol{w}\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \quad (i = 1, 2, \cdots, n) \end{array}$$

ただし、パラメータCはマージンの大きさと誤差の許容度のトレードオフを決めるパラメータである。この最適化問題をラグランジュ乗数法などを用いると、結局最大化する目的関数はハードマージンSVMと同じとなる。

$$\tilde{L}(\boldsymbol{a}) = \sum_{i=1}^n a_i - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n a_i a_j t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}} \phi(\boldsymbol{x}_j)$$

ただし、制約条件が$a_i \geq 0$の代わりに$0 \leq a_i \leq C (i = 1, 2, \ldots, n)$となる。（ハードマージンSVMと同じ$\sum_{i=1}^n a_i t_i = 0$も制約条件）

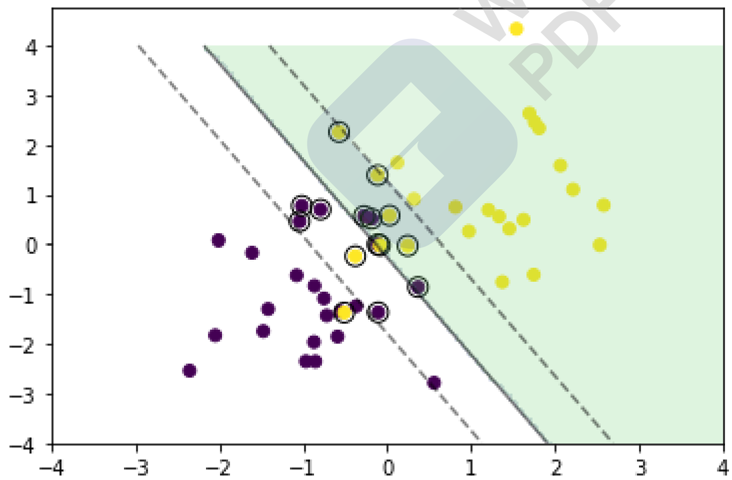
```
X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

C = 1
eta1 = 0.01
eta2 = 0.001
n_iter = 1000

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.clip(a, 0, C)
```

予測

```
index = a > 1e-8
support_vectors = X_train[index]
support_vector_t = t[index]
```

```
    support_vector_a = a[index]

    term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
    b = (support_vector_t - term2).mean()


    xx0, xx1 = np.meshgrid(np.linspace(-4, 4, 100), np.linspace(-4, 4, 100))
    xx = np.array([xx0, xx1]).reshape(2, -1).T

    X_test = xx
    y_project = np.ones(len(X_test)) * b
    for i in range(len(X_test)):
        for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
            y_project[i] += a * sv_t * sv.dot(X_test[i])
    y_pred = np.sign(y_project)


    # 訓練データを可視化
    plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
    # サポートベクトルを可視化
    plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
                        s=100, facecolors='none', edgecolors='k')
    # 領域を可視化
    plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
    # マージンと決定境界を可視化
    plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
                         levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

<matplotlib.contour.QuadContourSet at 0x7f8a8c51cad0>



0 秒 完了時間: 0:04