第5、6回
教師あり学習と
分類モデル

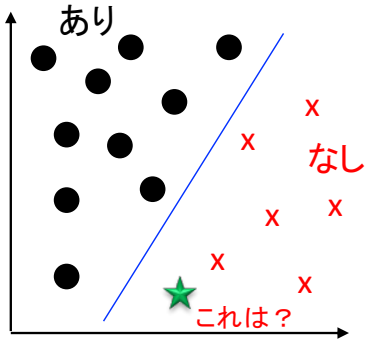
目次

- 分類とは
- ロジスティック回帰
- ロジスティック回帰の概念
- ロジスティック回帰を試す
- 訓練結果の可視化
- より実際的な事例
- 量的・質的変数とダミー変数
- 多項分類への応用
- サポートベクターマシン
- マージンの最大化
- サポートベクターマシンを試す
- 訓練結果の可視化

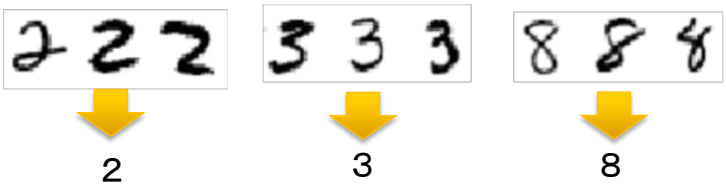
ロジスティック回帰

分類とは

- 分類は離散値（クラスラベル）を予測する教師あり学習
  - 過去の観測に基づき、新しいサンプルを対象としてクラスラベルを予測する
  - 二項分類
    - 2つのクラスを区別するためのルールを学習する。例：メールのスパムフィルタ
  - 多項分類
    - 複数あるクラスを区別するためのルールを学習する。例：手書き文字認識



二項分類



多項分類

ロジスティック回帰

- その名前とは裏腹に、回帰ではなく分類を行うためのモデル
- 説明変数を与えた場合に、2値（例えば0か1）をとる確率を出力する
- ロジスティック回帰モデルは、予測されるクラスラベルに関心があるだけでなく、クラスの所属確率を見積もりたい場合に有効
  - 例：天気予報において、雨が降るかだけでなく、降水確率も見積もりたい場合など

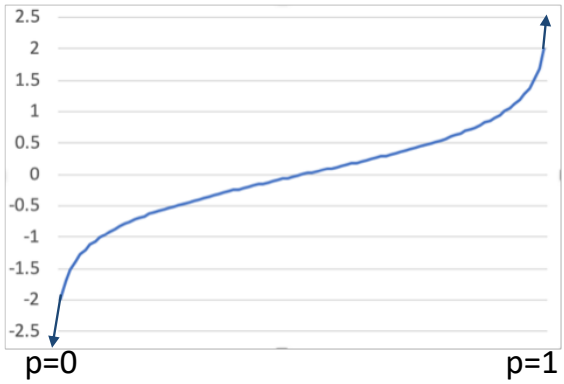
1か0かを予測するだけでなく、1になる確率は何パーセントかも含めて予測する

ロジスティック回帰の概念（1）

- 答えが1となる確率をpとすると、答えが0となる確率は1-pである。
- この比をオッズ比と呼ぶ $\frac{p}{1-p}$
- このオッズ比をもとに、ロジット関数を定義する

$$logit(p) = log\frac{p}{1-p}$$

- ロジット関数は、p=0のときマイナス無限大、p=1のときプラス無限大の値をとる



- このロジット関数を使って、説明変数からなる回帰直線の出力を0～1の確率pに変換する

ロジスティック回帰の概念(2)

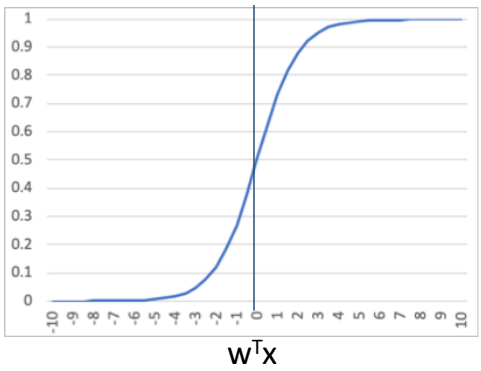
- ロジット関数の値と回帰直線の値を対応づける

$$logit(p) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \Sigma_{i=0}^{n}(w_ix_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

  - ただし、$x_0 = 1$
- 分類において求めたいのはp(答えが1となる確率)なので、上記方程式をp=の形に変換する

$$p = \phi(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}}$$

  - ロジット関数の逆関数でロジスティック関数(シグモイド関数)と呼ばれる
  - eは自然対数の底(≒2.718・・・)
- もしφ(x) が 0.5以上なら答えを1と予測し、φ(x) が0.5未満なら答えが0と予測
  - またφ(x)の値自体が答えが1となる確率を表している
  - w^Txの正・負は答えの予測1と0に対応



ロジスティック回帰を試す

- scikit-learnのdatasetsモジュールに含まれている"あやめ"の測定値(各部位の長さと幅)と品種のデータを使用する。
  - あやめの品種は3種類だが、今回はそのうちの1種のデータは捨てて学習させる。
    - 「花弁の長さ」と「花弁の幅」を説明変数として、品種0と品種1を学習させる。
- Spyderを起動し、[新規]-[新規ファイル]を作成
  - あやめデータを読み込み表示し、トレーニングデータとテストデータに分割

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np
import matplotlib.pyplot as plt

iris = datasets.load_iris()

X = iris.data[iris.target != 2][:, [2, 3]]
y = iris.target[iris.target != 2]
print(X, y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)
```

あやめデータの読み込み

品種2は捨てて、列2(花弁の長さ)と列3(花弁の幅)を説明変数として使用

学習と結果の表示

- ロジスティック回帰は、scikit-learnのlinear_modelパッケージのLogisticRegressionクラスに実装されている

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

～ 中略～

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

lr = LogisticRegression()
lr.fit(X_train, y_train)

print ("テストデータの予測", lr.predict(X_test))
print ("テストデータの正解", y_test)
print ("正解率", lr.score(X_test, y_test) * 100)
```

ロジスティック回帰モデルの生成

学習の実行

```
テストデータの予測 [0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1]
テストデータの正解 [0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1]
正解率 100.0
```


訓練結果の可視化

- 今回のロジスティック回帰の決定境界は

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

となる直線

  - 値はintercept_とcoef_ に入っているため、値を取得し境界をプロットする
  - また、直線の方程式は上記の式を変形し下記となる

$$x_2 = \frac{-w_1 x_1 - w_0}{w_2}$$

```
import matplotlib.pyplot as plt
```

～ 中略～

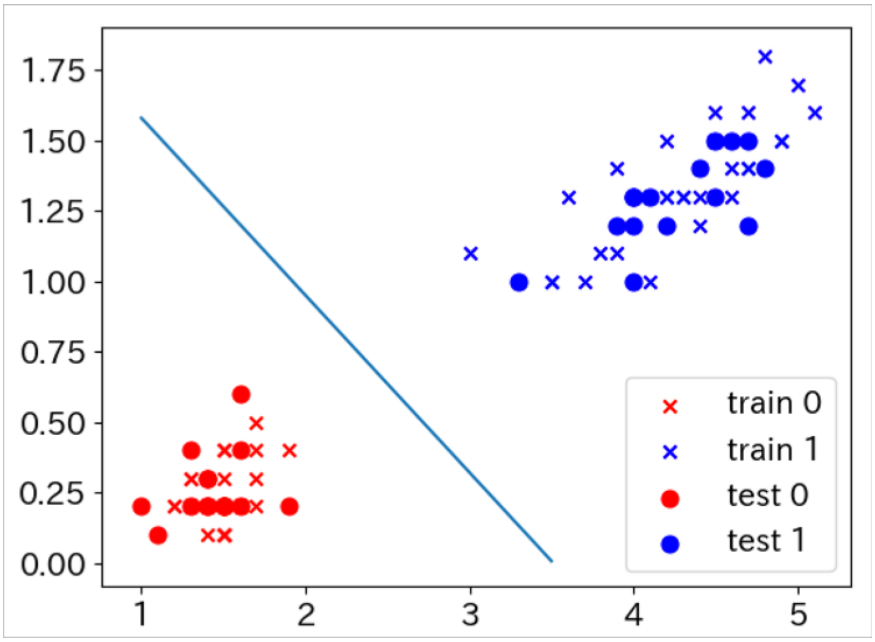
係数w_0～w_2の取り出し

```
w0 = lr.intercept_[0]; w1 = lr.coef_[0, 0]; w2 = lr.coef_[0, 1]
xmin = X_train[0].min(); xmax = X_train[0].max()

plt.plot([xmin, xmax], [(-w1 * xmin - w0) / w2, (-w1 * xmax - w0) / w2])
plt.scatter(X_train[y_train==0, 0], X_train[y_train==0, 1], c='red',
marker='x', label='train 0')
plt.scatter(X_train[y_train==1, 0], X_train[y_train==1, 1], c='blue',
marker='x', label='train 1')
plt.scatter(X_test[y_test==0, 0], X_test[y_test==0, 1], c='red', marker='o',
s=60, label='test 0')
plt.scatter(X_test[y_test==1, 0], X_test[y_test==1, 1], c='blue',
marker='o', s=60, label='test 1')
plt.legend(loc='lower right')
```

決定境界の描画

可視化例

- 今回は分類が容易なデータであったことがわかる



所属確率

- ロジスティック回帰では、分類だけでなく所属確率も計算できる

```
print(lr.predict_proba(X_test))
```

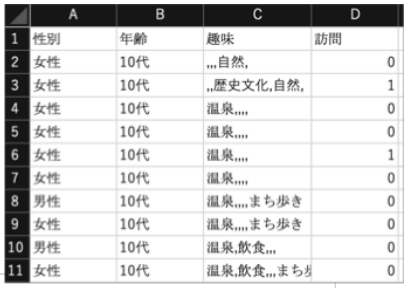
```
[[0.77509088 0.22490912]
 [0.03025837 0.96974163]
 [0.86257016 0.13742984]
 [0.04904237 0.95095763]
 [0.04644509 0.95355491]
 [0.20789651 0.79210349]
 [0.82177586 0.17822414]
 [0.04721441 0.95278559]
 [0.03323958 0.96676042]
 [0.07450556 0.92549444]]
```

品種0の確率

品種1の確率

より実際的な事例

- 実際のアンケートデータなどの場合は、データのクリーニングや加工が必要となる
- 問題設定
  - ある観光客が、観光施設Aを訪問するか否か
    - 上記を、観光客の性別、年齢、趣味から予測する



	A	B	C	D
1	性別	年齢	趣味	訪問
2	女性	10代	,,,自然,	0
3	女性	10代	,,歴史文化,自然,	1
4	女性	10代	温泉,,,,	0
5	女性	10代	温泉,,,,	0
6	女性	10代	温泉,,,,	1
7	女性	10代	温泉,,,,	0
8	男性	10代	温泉,,,,まち歩き	0
9	女性	10代	温泉,,,,まち歩き	0
10	男性	10代	温泉,飲食,,,	0
11	女性	10代	温泉,飲食,,,まち歩	0

- Spyderを起動し、[新規]-[新規ファイル]を作成

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

plt.rcParams["font.family"] = "IPAexGothic"
plt.rcParams["font.size"] = 15

file = pd.ExcelFile("http://pana4405.u-shizuoka-ken.ac.jp/wp-content/uploads/
2019/02/ch5.xlsx")
data = file.parse("アンケート")
print(data)
```


量的・質的変数とダミー変数

- 量的変数
  - 間隔尺度（間隔が等間隔で明確な0が決まってない：西暦など・・・西暦0年は人間が勝手に決めた0）
  - 比例尺度（間隔が等間隔で明確な0が決まっている：身長・体重など）
- 質的変数
  - 名義尺度（カテゴリを区別するための変数：性別、職業、居住地など）
  - 順序尺度（数値の大小には意味があるが、値の間隔には意味がない：満足、どちらでもない、不満足など）
- 質的変数をダミー変数に変換
  - 単一選択の場合
    - 選択肢が2種類の場合は1つのダミー変数の0か1に置き換える（例：性別0なら男性、1なら女性）
    - 選択肢が3種類以上の場合は、選択肢の数n-1のダミー変数の0か1に置き換える（例：会社員、公務員、その他＝会社員ダミー変数、公務員ダミー変数を用意し、どちらも0ならその他）
  - 複数選択の場合
    - 選択肢の数のダミー変数を用意し、各選択肢が含まれる場合は1、そうでない場合は0

ダミー変数への置き換え

- 新たに、gender、age、hot、food、art、nature、street変数をDataFrameに追加して値を設定する

```
data["gender"] = 0
data["age"] = 0
data["hot"] = 0
data["food"] = 0
data["art"] = 0
data["nature"] = 0
data["street"] = 0
data.loc[data["性別"]=="女性","gender"] = 1
data.loc[data["年齢"]=="10代","age"] = 10
data.loc[data["年齢"]=="20代","age"] = 20
data.loc[data["年齢"]=="30代","age"] = 30
data.loc[data["年齢"]=="40代","age"] = 40
data.loc[data["年齢"]=="50代","age"] = 50
data.loc[data["年齢"]=="60代","age"] = 60
data.loc[data["年齢"]=="70代","age"] = 70
data.loc[data["趣味"].str.contains("温泉"),"hot"] = 1
data.loc[data["趣味"].str.contains("飲食"),"food"] = 1
data.loc[data["趣味"].str.contains("歴史文化"),"art"] = 1
data.loc[data["趣味"].str.contains("自然"),"nature"] = 1
data.loc[data["趣味"].str.contains("まち歩き"),"street"] = 1
```

性別は0か1のダミー変数へ

年齢は比例尺度へ

趣味の複数選択は5つのダミー変数へ

学習と予測

- gender、age、hot、food、art、nature、streetを説明変数とし、目的変数として観光施設Aを訪問したか（した場合には1、そうでないなら0）を学習

```
X = data.loc[:,["gender", "age", "hot", "food", "art",
"nature", "street"]]
y = data.loc[:,["訪問"]]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)

lr = LogisticRegression()
lr.fit(X_train, y_train.values.ravel())
print ("正解率", lr.score(X_test, y_test) * 100)
```

Y配列は1次元でないと警告が出るので、1次元に変換

```
正解率 82.06896551724138
```

訪問確率の計算

- ある性別、年齢（デモグラフィックデータ）と、趣味（サイコグラフィックデータ）を入力として、観光施設Aへの訪問確率を計算
  - scikit-learnのpredict_probaメソッドを使う場合
  - 係数wを用いてマニュアルで計算する場合（あとで別のアプリケーションに組み込んで使う場合を想定）

```
test = [[1, 40, 1, 0, 0, 0, 1]]
print(lr.predict_proba(test))

w0 = lr.intercept_[0]
w1 = lr.coef_[0][0]
w2 = lr.coef_[0][1]
w3 = lr.coef_[0][2]
w4 = lr.coef_[0][3]
w5 = lr.coef_[0][4]
w6 = lr.coef_[0][5]
w7 = lr.coef_[0][6]

z = w0 + w1 * test[0][0] + w2 * test[0][1] + w3 * test[0][2] +
w4 * test[0][3] + w5 * test[0][4] + w6 * test[0][5] + w7 *
test[0][6]
proba = 1 / (1 + np.exp(-z))
print("訪問確率", proba)
```

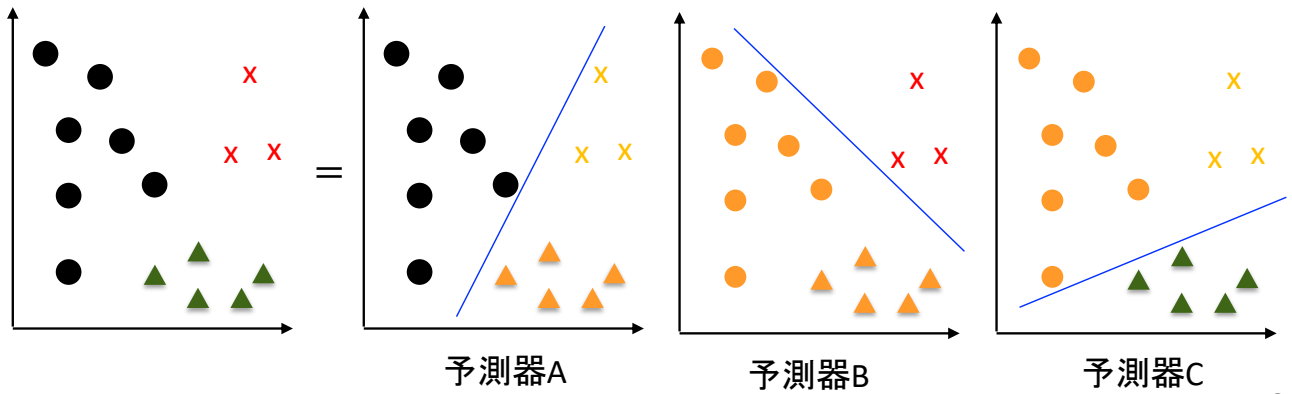
テストデータ

predict_probaメソッドで計算

ロジスティック関数の定義からマニュアルで計算

多項分類への応用

- One-vs-allアルゴリズム
  - クラスラベルが3（A, B, C）の場合
    - クラスがAか、それ以外かで学習・・・予測器A
    - クラスがBか、それ以外かで学習・・・予測器B
    - クラスがCか、それ以外かで学習・・・予測器C
  - 未知データを予測器A～Cに与え、最も確率が高いクラスに所属すると予測する



サポートベクターマシン

サポートベクターマシン

- SVM（Support Vector Machine）
  - 元の次元から一つ低い平面を超平面（＝決定境界）と呼ぶ
  - SVMでは、2つのクラスの間のマージンを最大化するような超平面を探す
    - マージンとは、超平面に最も近いトレーニングサンプルからの距離
    - 超平面に最も近いサンプルは、サポートベクトルと呼ばれる
  - 正と負の超平面
    - サポートベクトルと交わる超平面
    - 決定境界を下記とすれば

$$w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x} = 0$$

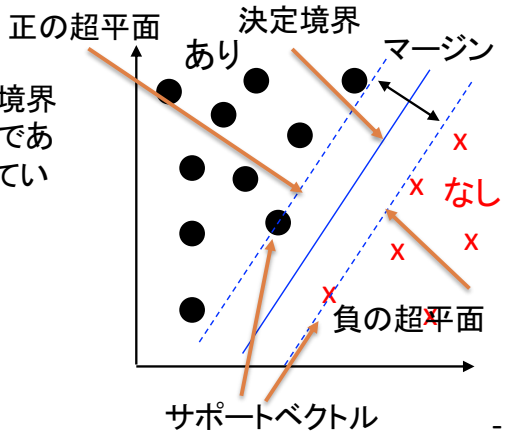
    - 正の超平面の式

$$\frac{w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x}_{\mathbf{pos}}}{\|w\|} = k$$

    - 負の超平面の式

$$\frac{w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x}_{\mathbf{neg}}}{\|w\|} = -k$$

点x_{pos}と決定境界との距離がkであることを表している



マージンの最大化

- 決定境界の式は右辺が0なので、左辺のwを定数倍しても超平面は変わらない

$$w_0 + \mathbf{w}^T\mathbf{x} = 0$$

- 正の超平面について、その分子の絶対値が1となるようにwを定数倍したとすると、決定境界との距離kは下記となる

$$k = \frac{1}{\|w\|}$$

- したがって、マージンを最大化するには、$\|w\|$ を最小化すればよい
- 制約条件
  - wを単純に最小化すると $\mathbf{w} = \mathbf{0}$ が解になってしまう
  - 今、学習サンプルyが-1または1で与えられているとして、正の超平面より遠いサンプルが1、負の超平面よりも遠いサンプルが-1とする
  - 下記の制約要件を満たす解を求めればよい

$$y_i(w_0 + \mathbf{w}^T\mathbf{x_i}) \geq 1$$

サポートベクターマシンを試す

- ロジスティック回帰の例題と同様に、scikit-learnのdatasetsモジュールに含まれている”あやめ”の測定値（各部位の長さ）と品種のデータを使用する。
- Spyderを起動し、[新規]-[新規ファイル]を作成
  - あやめのデータを読み込み表示し、トレーニングデータとテストデータに分割（ロジスティック回帰の例題をコピー貼り付けでOK）

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import numpy as np
import matplotlib.pyplot as plt

iris = datasets.load_iris()
print(iris)
X = iris.data[iris.target != 2][:, [2, 3]]
y = iris.target[iris.target != 2]
print(X, y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)
```

あやめデータの読み込み

品種2は捨てて、列2（花弁の長さ）と列3（花弁の幅）を説明変数として使用

学習と結果の表示

- SVMは、scikit-learnのsvmパッケージのSVMクラスに実装されている

```
from sklearn.svm import SVC
```

～ 中略～

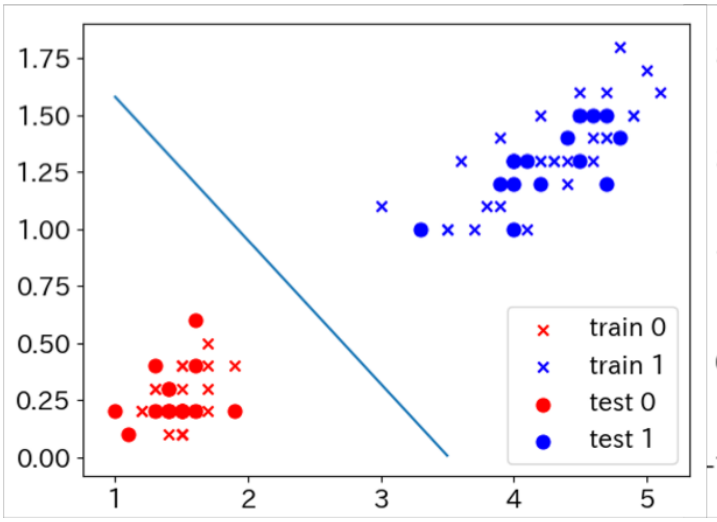
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)

svm = SVC(kernel="linear")
svm.fit(X_train, y_train)

print ("テストデータの予測", svm.predict(X_test))
print ("テストデータの正解", y_test)
print ("正解率", svm.score(X_test, y_test) * 100)
```

SVMモデルの生成

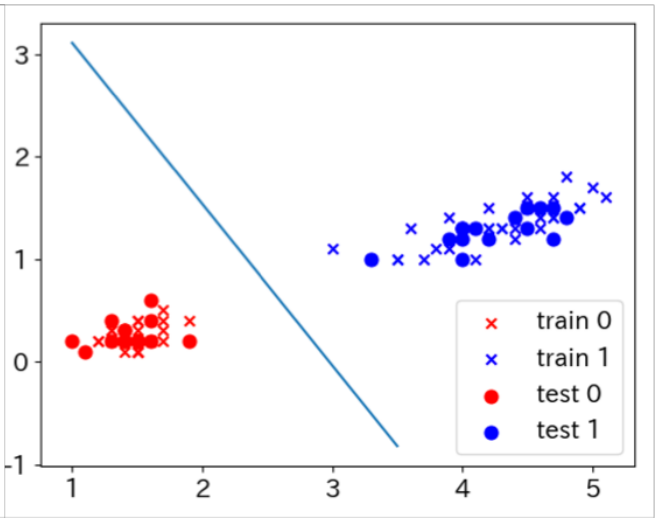
学習の実行

```
テストデータの予測 [0 1 0 1 1 1 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1]
テストデータの正解 [0 1 0 1 1 1 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1]
正解率 100.0
```


訓練結果の可視化



ロジスティック回帰



サポートベクターマシン

課題5

- 「あやめ」の他の特徴（別の列）を使って分類を試す。

列0：sepal length	ガクの長さ
列1：sepal width	ガクの幅
列2：petal length	花弁の長さ
列3：petal width	花弁の幅