

# 機械学習入門

経済学部 BX584

線形回帰

# 機械学習(=計算機に学習させる)とは・・・

- 予測モデル＝値を入力すると出力値が得られる関数
- 出力側には目標値(理想的な値)があらかじめ決められている
  - 損失関数＝目標値からのズレの量をあらわす関数
- 目標値に近い値を出す関数を計算機に選ばせるのが機械学習



# 1変数の線形回帰

- 予測モデル＝出力を入力関数で表したもの
- 線形回帰では関数のかたちを一次式（線形関数）に限定

$$f(x) = ax + b$$

- そして「 $a$ と $b$ をいくらにすればよいか？」という問題を解く

# 簡単な例題

- ある箱に、
  - 2.0を入れたら-4.0が出てきてほしい
  - 1.0を入れたら-2.0が出てきてほしい
  - -3.0を入れたら5.0が出てきてほしい
- パラメータ $a$ と $b$ をいくらにすればいいのでしょうか？

# 点の位置を描画してみる

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

```
x = [2, 1, -3]
```

```
y = [-4, -2, 5]
```

```
plt.plot(x, y, '.')
```

```
plt.show()
```

# パラメータを使って出力値を表す

- 左辺がパラメータ $a, b$ を使って表した出力値
- 右辺が目標値

$$2.0a + b \approx -4.0$$

$$1.0a + b \approx -2.0$$

$$-3.0a + b \approx 5.0$$

# 誤差の2乗

- $2.0a + b \approx -4.0$  誤差の2乗:  $\{-4.0 - (2.0a + b)\}^2$
- $1.0a + b \approx -2.0$  誤差の2乗:  $\{-2.0 - (1.0a + b)\}^2$
- $-3.0a + b \approx 5.0$  誤差の2乗:  $\{5.0 - (-3.0a + b)\}^2$

# 問題を解く方針

- 誤差の2乗の和(足したもの)を最小にすることで

$$f(x) = ax + b$$

を求める。つまり、次の式の値を最小にする $a$ と $b$ を求める。

$$\{-4 - (2a + b)\}^2 + \{-2 - (a + b)\}^2 + \{5 - (-3a + b)\}^2$$

- 誤差の2乗の和を最小にする方法を「**最小二乗法**」と呼ぶ。



# 手計算で解く方法

$$l(a, b) = \{-4 - (2a + b)\}^2 + \{-2 - (a + b)\}^2 + \{5 - (-3a + b)\}^2$$

- 上の関数 $l(a, b)$ を $a$ で偏微分したものを $= 0$ とおき、さらに
- 上の関数 $l(a, b)$ を $b$ で偏微分したものを $= 0$ とおくことで、
- 2本の方程式ができる。これを $a$ と $b$ について解けばよい。

$$\begin{aligned}\frac{\partial l(a, b)}{\partial a} &= -4\{-4 - (2a + b)\} - 2\{-2 - (a + b)\} + 6\{5 - (-3a + b)\} \\ &= 28a + 50 = 0\end{aligned}$$

$$\begin{aligned}\frac{\partial l(a, b)}{\partial b} &= -2\{-4 - (2a + b)\} - 2\{-2 - (a + b)\} - 2\{5 - (-3a + b)\} \\ &= 6b + 2 = 0\end{aligned}$$

# 問題の一般化(1変数の線形回帰＝単回帰)

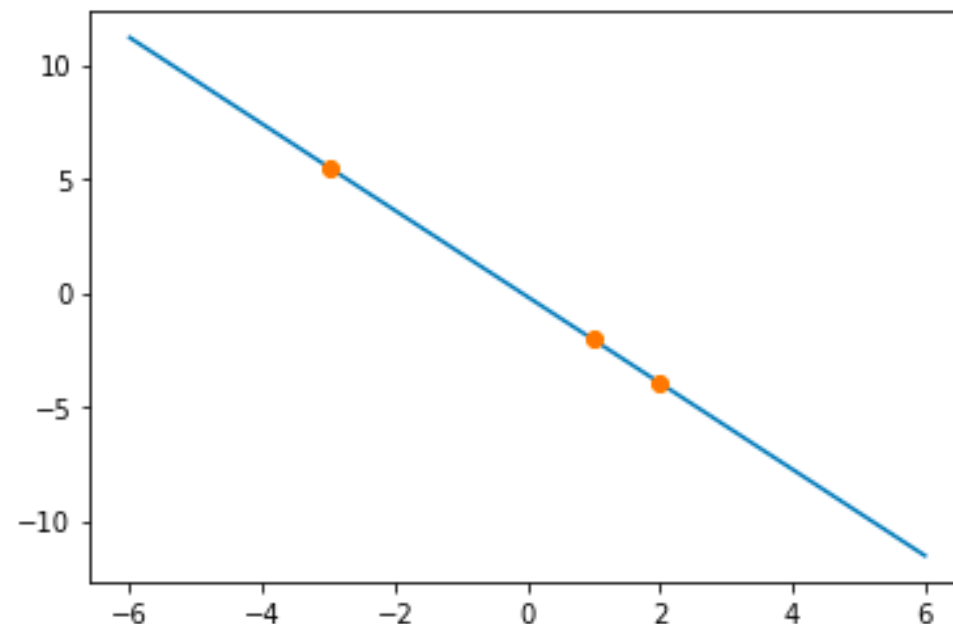
- 入力と、それに対する出力(＝目標値)がたくさん与えられている
  - 入力:  $\{x_1, x_2, \dots, x_N\}$
  - 出力:  $\{y_1, y_2, \dots, y_N\}$
- 入力と出力の間には、次の関係があると仮定する

$$f(x) = ax + b$$

- このとき、誤差の二乗和を最小にするような関数を求める

# 問題の解き方のイメージ

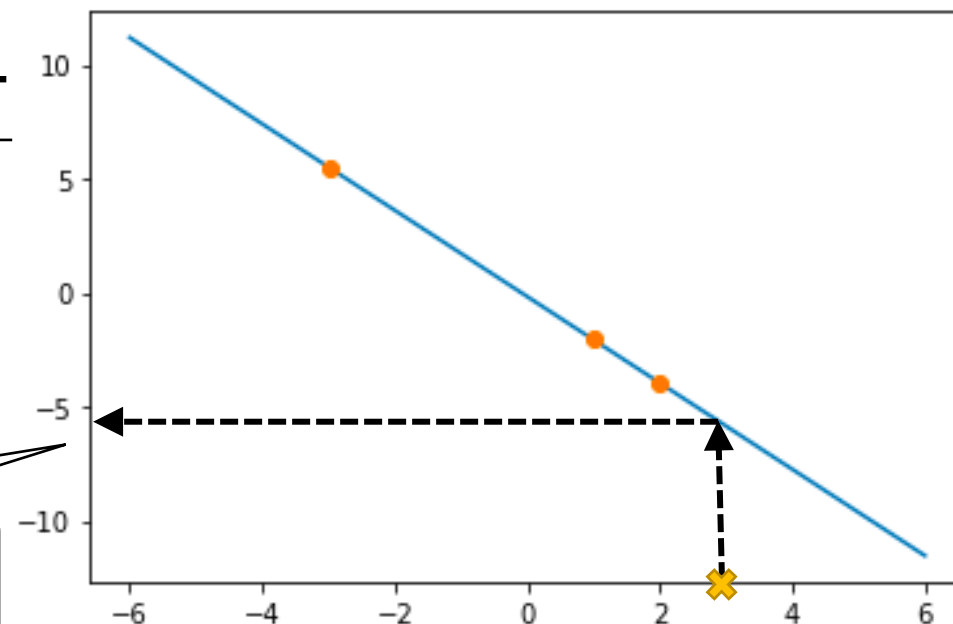
- 入力値と出力値のペアを表す点がたくさんある
  - 入力値が $x$ 座標、出力値が $y$ 座標。
- それらの点にぴったり合う直線を引く
- こういう問題を「線形回帰」という
  - 「線形」≡「まっすぐ」



# 線形回帰による予測

- 直線が求まれば・・・
  - つまり関数の一つ選べば・・・
- 任意の入力値について目標値を予測可
  - 見たことない入力値でも出力値を計算可
  - 出力値を正解(目標値)の予測として使う
  - 機械学習は予測に使える！

未知の入力値について  
予測された出力値



# 最小二乗法＝誤差の二乗和の最小化

- 最小二乗法はデータ点の数がいくら増えても使える
  - 各データの誤差の二乗： $\{y_i - (ax_i + b)\}^2$
  - 誤差の二乗の和

$$\sum_{i=1}^N \{y_i - (ax_i + b)\}^2$$

# 今日の例題 (データ点をひとつ増やしただけです...)

- 関数  $y = ax + b$  によってモデル化される「箱」に
  - 2.0を入れたら-4.0が出てきてほしい
  - 1.0を入れたら-2.0が出てきてほしい
  - -3.0を入れたら5.0が出てきてほしい
  - -1.0を入れたら2.0が出てきてほしい
- パラメータ  $a$  と  $b$  をいくらにすればいいのでしょうか？

# 例題の点の位置を描画してみる

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

```
x = [2, 1, -3, -1]
```

```
y = [-4, -2, 5, 2]
```

```
plt.plot(x, y, '.')
```

```
plt.show()
```

# 最小二乗法による線形回帰(エラーが出る)

```
from sklearn import linear_model
```

```
x = [2, 1, -3, -1]
```

```
y = [-4, -2, 5, 2]
```

```
reg = linear_model.LinearRegression() #線形回帰を準備
```

```
reg.fit(x, y) #最小2乗法を実行
```

```
print(reg.coef_, reg.intercept_)
```



# 最小二乗法による線形回帰(エラーが出ない)

```
from sklearn import linear_model
```

```
x = [[2], [1], [-3], [-1]]
```

```
y = [-4, -2, 5, 2]
```

```
reg = linear_model.LinearRegression() #線形回帰を準備
```

```
reg.fit(x, y) #最小2乗法を実行
```

```
print(reg.coef_, reg.intercept_)
```

# なぜこうなっている？

- scikit-learnでは入力がベクトルであると想定
  - 入力が一個の数値であっても、一次元ベクトルに変換しないといけない



# 最小二乗法による線形回帰(こう書いてもよい)

```
import numpy as np
from sklearn import linear_model
```

```
x = [2, 1, -3, -1]
```

```
x = np.array(x).reshape(-1, 1)
```

```
y = [-4, -2, 5, 2]
```

```
reg = linear_model.LinearRegression() #線形回帰を準備
```

```
reg.fit(x, y) #最小2乗法を実行
```

```
print(reg.coef_, reg.intercept_)
```

# 重回帰

- 回帰のうち入力値が1つ＝単回帰
  - 直線をデータ点の集合にフィットさせる
- 入力値が2つ（説明変数が2つ）以上＝重回帰
  - 平面をデータ点の集合にフィットさせる
    - 「線形」＝「まったいら（曲がっていない）」

# 2変数の線形回帰

- 予測モデル＝出力を入力関数で表したもの
- 線形回帰では関数のかたちを一次式（線形関数）に限定

$$f(x_1, x_2) = a_1x_1 + a_2x_2 + b$$

- そして「 $a_1$ と $a_2$ と $b$ をいくらにすればよいか？」という問題を解く

# 2変数の場合の誤差の二乗和

- 各データの誤差の二乗:  $\{y_i - (a_1x_{i,1} + a_2x_{i,2} + b)\}^2$
- 誤差の二乗の和

$$\sum_{i=1}^N \{y_i - (a_1x_{i,1} + a_2x_{i,2} + b)\}^2$$

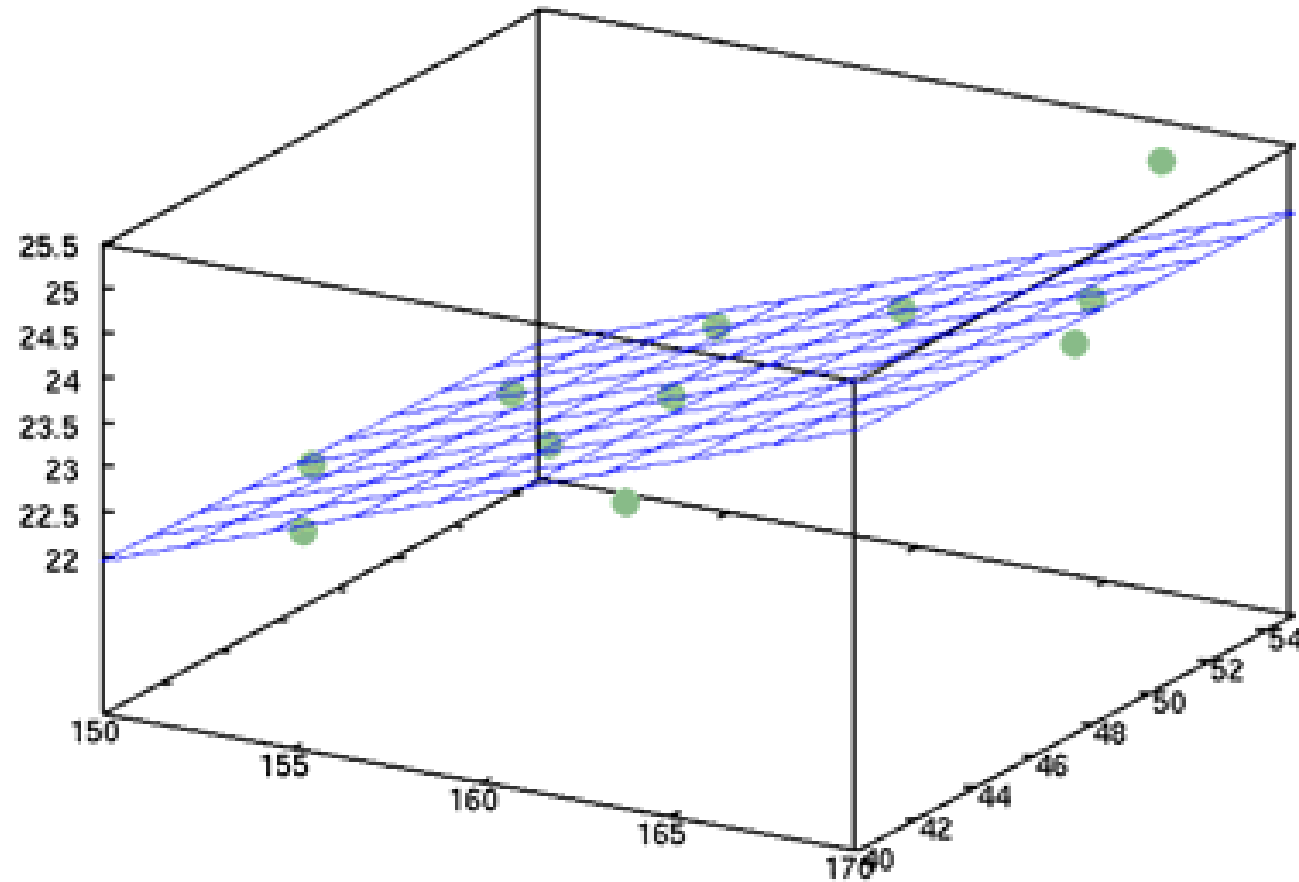
# 入力が2次元ベクトルの場合

```
import numpy as np
from sklearn import linear_model

x = [[2,2], [1,1], [-3,-3], [-1,-1]]
y = [-4, -2, 5, 2]

reg = linear_model.LinearRegression()
reg.fit(x, y)
print(reg.coef_, reg.intercept_)
```

# 入力が2次元ベクトルの場合のイメージ図



出典 <http://www.wakayama-u.ac.jp/~wuhhy/am8.pdf>



# 予測の仕方

- 例:  $[-1.5, -1.5]$ という入力に対応する目的変数の値を予測したい
- `predict()`を使う
  - 「`reg.predict([-1.5, -1.5])`」と書く
- 見たことがない入力ベクトルについても出力を予測できる

# 入力が2次元ベクトルの場合の予測

```
import numpy as np
from sklearn import linear_model

x = [[2,2], [1,1], [-3,-3], [-1,-1]]
y = [-4, -2, 5, 2]

reg = linear_model.LinearRegression()
reg.fit(x, y)

print(reg.predict([-1.5,-1.5]))
```