機械学習
scikit-learnによる機械学習入門

管理工学科
篠沢　佳久

機械学習の基礎①

- 機械学習で（主に）取り扱う問題
  - 分類問題
    - 与えられたデータがどのクラス（カテゴリー）に所属するかを予測する問題
  - 回帰問題
    - 与えられたデータから正解（時系列データの場合，将来の値）となる値を予測する問題

機械学習の基礎②

- scikit-learnによるプログラミングの基本
  - 分類問題：ロジスティック回帰*
  - 回帰問題：重回帰分析（線形回帰）
- データの取り扱い方
  - 学習データ，評価データ
  - ホールドアウト法
  - 交差検証（Cross Validation）

*回帰とありますが，回帰問題を扱っているわけではありません．が，しかし，行なっていることは回帰かもしれません（後述）

分類問題

ロジスティック回帰

二値分類問題①

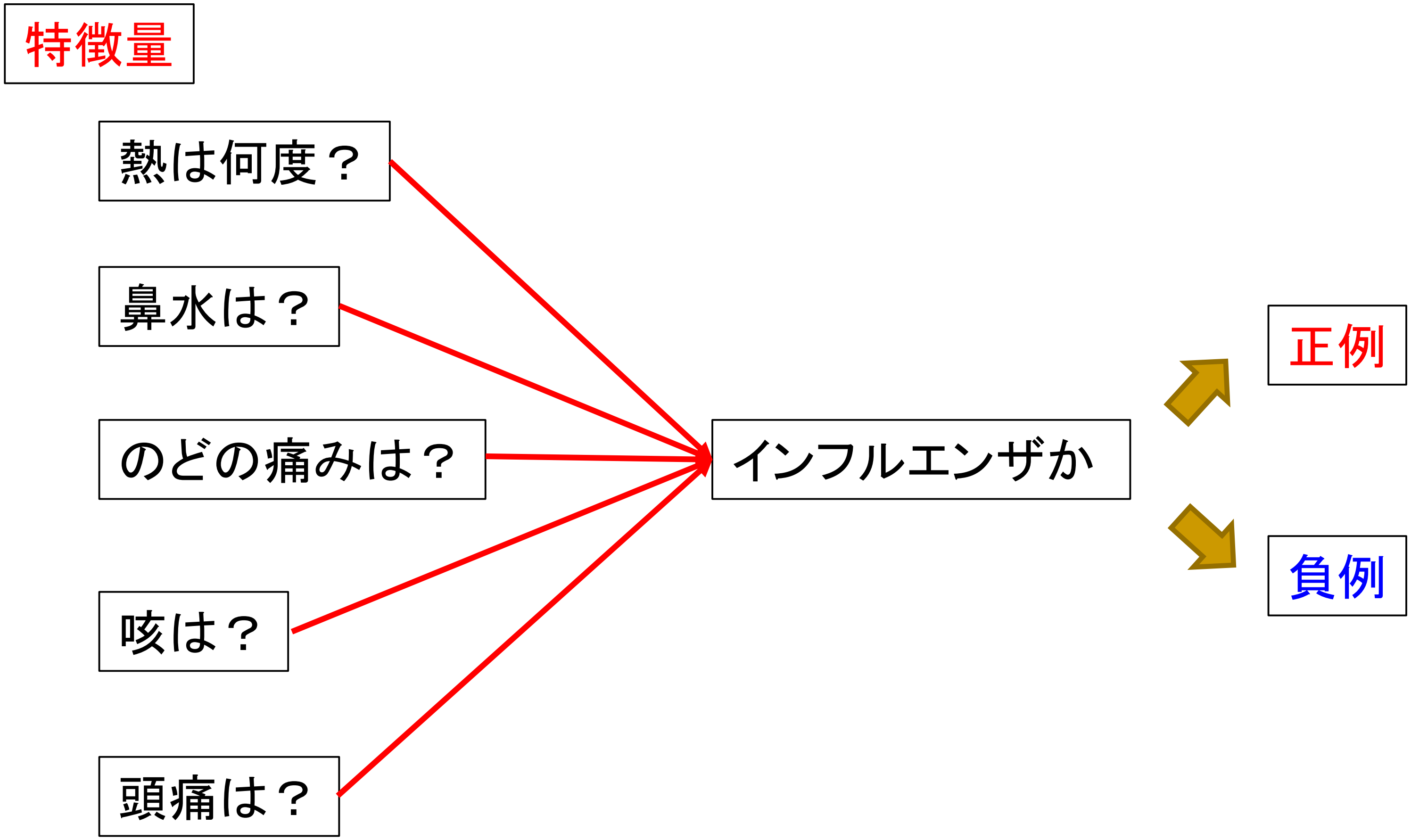
- 二クラスに分類する問題

- (例)インフルエンザに感染しているか
  - 感染している場合→ 正例, Positive, 1(数値)*
  - 感染していない場合 → 負例, Negative, 0(数値)

- (例)明日の日経平均株価は上昇するか
  - 上昇する場合→ 正例, Positive, 1(数値)
  - 上昇しない場合 → 負例, Negative, 0(数値)

*数値で表す場合, 他にも感染している場合は+1, 感染していない場合は-1 などいろいろと表現できます

二値分類問題②



二値分類問題③

- データ(特徴量):$\mathbf{x}_i^t = (x_{i1}, x_{i2}, \cdots, x_{iN})$ $(i=1,2,\cdots,P)$
  - データ数:P個, 特徴量の次元数:N
- 正解値:$t_1, t_2, \cdots, t_p$

- 予測値とは?
  - 二値分類では, 正例である確率(0~1の値)を予測*

- 正解値とは?
  - 正例である場合は1
  - 負例である場合は0

正解ラベル(正例か負例か)を直接予測していないことに注意

*負例である確率を予測してもかまいません

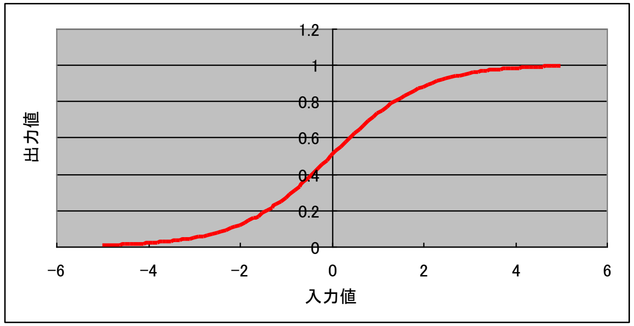
二値分類問題の解法

- 重回帰分析（線形回帰）

予測値 $$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN}$$

⇩ 0～1の範囲におさまるか

シグモイド関数*



- ロジスティック回帰

予測値 $$y_i = \frac{1}{1+\exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN}))}$$

$$\boxed{y = \frac{1}{1+\exp(-x)}}$$

$y_i > 0.5 \rightarrow$ 正例

$y_i < 0.5 \rightarrow$ 負例

*シグモイド関数を用いると，学習の更新則が簡単になるため使われています（後述）

ロジスティック回帰

$$y_i = \frac{1}{1+\exp(-(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN}))}$$

正例である確率

$$1-y_i = \frac{\exp(-(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN}))}{1+\exp(-(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN}))}$$

負例である確率

$$\frac{y_i}{1-y_i} = \frac{1}{\exp(-(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN}))}$$
$$= \exp(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN})$$

$$\log\left(\frac{y_i}{1-y_i}\right) = \beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN}$$

正例である確率

$$\frac{\boxed{y_i}}{\boxed{1-y_i}}$$

オッズ比

負例である確率

オッズ比の対数を目的変数とした重回帰分析

教師あり学習(Supervised learning)

i番目のデータ

$x_{i1}, x_{i2}, \cdots, x_{iN}$ ⇨

予測値

$$y_i = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN}))}$$

⇕

正解値，教師信号
（正解ラベル，教師ラベル）

t_i

正例の場合，確率は1
負例の場合，確率は0

- 正解値(教師信号)を与える**教師あり学習**
- 予測値と正解値の差を**誤差関数***として定義
- 誤差関数が最小となる係数 β を求める

*損失関数，目的関数，コスト関数とも呼ばれます

誤差関数①

- 誤差二乗和
  - 重回帰分析で利用

$$E=\sum_{i=1}^{P}(y_i-t_i)^2$$

- y_iとt_iが近い値の場合は小さくなり，y_iとt_iが離れている場合は大きくなる指標は？

- クロスエントロピー（交差エントロピー）*

$$E=-\sum_{i=1}^{P}(t_i\log y_i+(1-t_i)\log(1-y_i))$$

*シグモイド関数のスライドで補足しましたが，正確にはシグモイド関数とクロスエントロピーを組み合わせると，学習の更新則が簡単になります（後述）

クロスエントロピー①

- クロスエントロピー
  - 二つの確率分布P(x), Q(x)の類似度を測る指標

$$\boxed{H(P,Q) = -\sum_x P(x)\log Q(x)}$$

  - xが二値(x_1＝正例, x_2＝負例)の場合

$$p = P(x_1), P(x_2) = 1 - p$$

$$q = Q(x_1), Q(x_2) = 1 - q$$

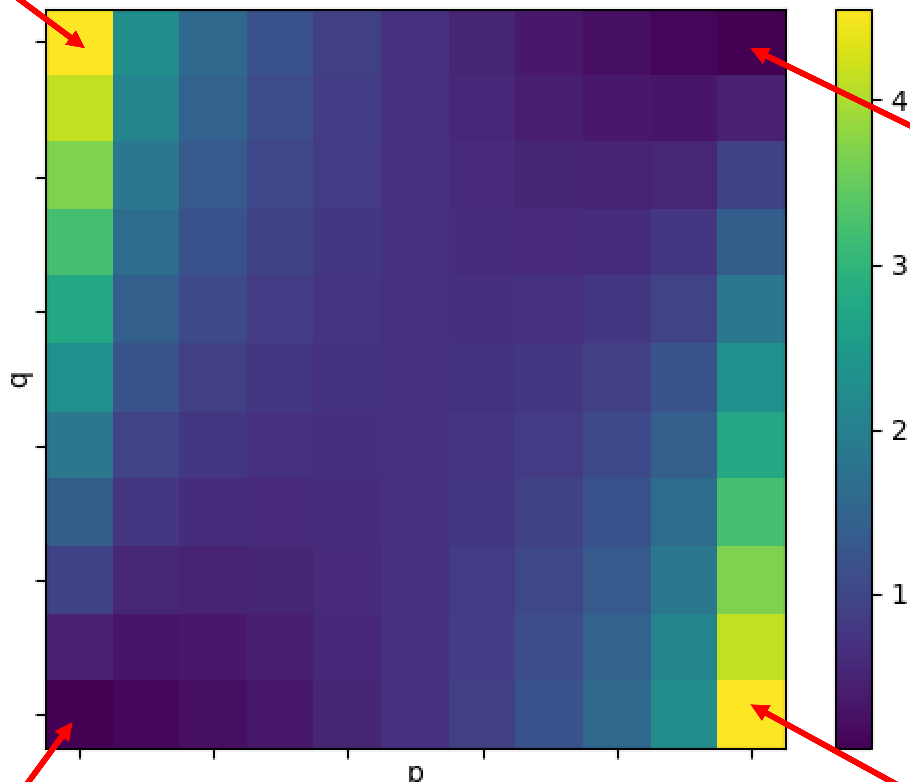
Pは正解の確率分布
Qは予測の確率分布

$$\begin{aligned} H(P,Q) &= -\sum_x P(x)\log Q(x) \\ &= -P(x_1)\log Q(x_1) - P(x_2)\log Q(x_2) \\ &= -p\log q - (1-p)\log(1-q) \end{aligned}$$

クロスエントロピー②

$$H(P,Q) = -p\log q - (1-p)\log(1-q)$$

p,qが離れた値である程，値は大きい



1

0

0

1

p,q共に1に近い程，値は小さい

p,qが離れた値である程，値は大きい

p,q共に0に近い程，値は小さい

誤差関数②

誤差関数（クロスエントロピー）

$$E(\boldsymbol{\beta}) = -\sum_{i=1}^{P}(t_i \log y_i + (1-t_i)\log(1-y_i))$$

最小

$$o_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN}$$

$$y_i = \frac{1}{1+\exp(-o_i)}$$

$$E_i(\boldsymbol{\beta}) = -(t_i \log y_i + (1-t_i)\log(1-y_i))$$

データ1個あたりの誤差関数

$$E(\boldsymbol{\beta}) = \sum_{i=1}^{P} E_i(\boldsymbol{\beta})$$

ロジスティック回帰の解法①

i番目のデータ

$x_{i1}, x_{i2}, \cdots, x_{iN}$

予測値

$$y_i = \frac{1}{1+\exp(-(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN}))}$$

正解値

t_i

正例の場合，確率は1
負例の場合，確率は0

誤差関数

$$E_i(\boldsymbol{\beta}) = -(t_i \log y_i + (1-t_i)\log(1-y_i))$$

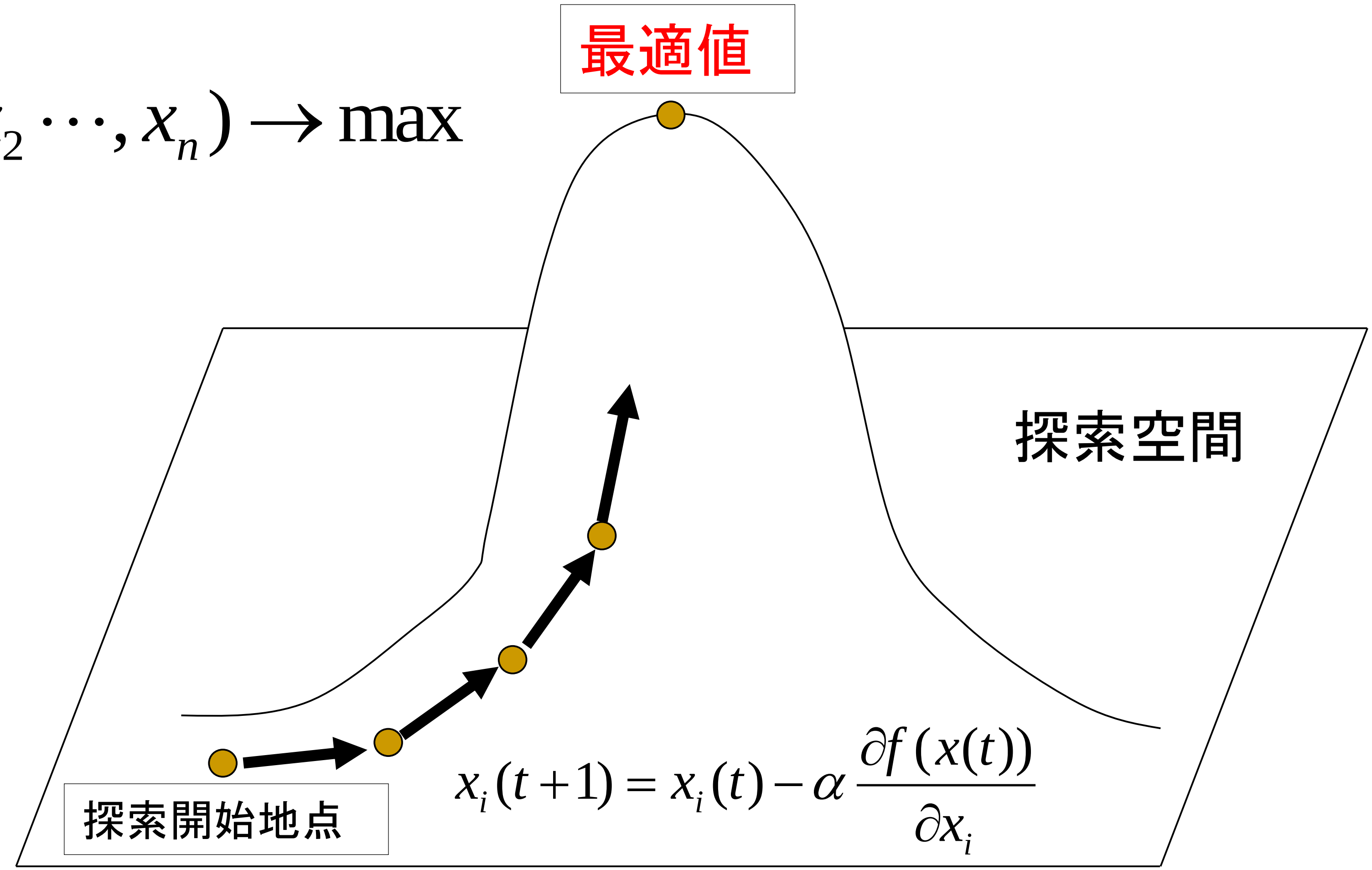
最急降下法

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \alpha \frac{\partial E_i}{\partial \boldsymbol{\beta}}$$

α :学習係数

最急降下法

$$f(x_1, x_2 \cdots, x_n) \to \max$$



α :学習係数

ロジスティック回帰の解法②

$$E_i(\boldsymbol{\beta}) = -t_i \log y_i - (1-t_i)\log(1-y_i)$$

$$o_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN}$$

$$y_i = \frac{1}{1+\exp(-o_i)}$$

シグモイド関数の性質

$$f(x) = \frac{1}{1+e^{-x}}$$

$$f'(x) = f(x)(1-f(x))$$

$$\frac{\partial E_i}{\partial \beta_j} = \boxed{\frac{\partial E_i}{\partial y_i}}\boxed{\frac{\partial y_i}{\partial o_i}}\boxed{\frac{\partial o_i}{\partial \beta_j}} = \left(-\frac{t_t}{y_i} + \frac{1-t_i}{1-y_i}\right) y_i(1-y_i)x_{ij} = (y_i - t_i)x_{ij}$$

$$\boxed{\frac{\partial E_i}{\partial y_i}} = -\frac{t_t}{y_i} + \frac{1-t_i}{1-y_i} \qquad \boxed{\frac{\partial y_i}{\partial o_i}} = y_i(1-y_i) \qquad \boxed{\frac{\partial o_i}{\partial \beta_j}} = x_{ij}$$

ロジスティック回帰の解法③

i番目のデータに対する修正*
（オンライン学習）

$$\frac{\partial E_i}{\partial \beta_j} = \underbrace{(y_i - t_i)}_{\text{予測値と正解値の差}} \underbrace{x_{ij}}_{\text{i番目のデータ}}$$

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial E_i}{\partial \beta_j}$$

全てのデータに対する修正
（バッチ学習）

$$\frac{\partial E}{\partial \beta_j} = \sum_{i=1}^{P} (y_i - t_i) x_{ij}$$

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial E}{\partial \beta_j}$$

*オンライン学習の最急降下法を確率的勾配降下法（stochastic gradient descent）
とも呼びます

確率的勾配降下法（オンライン学習）

while True :

    差の合計値 = 0

    for i in range(データ数) :

        予測値y_iを計算 $y_i = \dfrac{1}{1+\exp(-(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_N x_{iN}))}$

        予測値y_iと正解値t_iのクロスエントロピーを計算

        パラメータβを修正

        差の合計値 += クロスエントロピー

    if 差の合計値 < ε :

        break

実際にはランダムにデータを選びます（後述）

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial E_i}{\partial \beta_j}$$

$$\frac{\partial E_i}{\partial \beta_j} = (y_i - t_i)x_{ij}$$

微小値

確率的勾配降下法(オンライン学習)

$\mathbf{x}_i (i = 1, 2, \cdots, P)$

入力値

シグモイド関数

$$y_i = \frac{1}{1 + \exp(-\sum_{j=0}^{N} \beta_j x_{ij})}$$

予測値

正解値

t_i

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial E_i}{\partial \beta_j}$$

$$\frac{\partial E_i}{\partial \beta_j} = (y_i - t_i) x_{ij}$$

クロスエントロピー

$$E_i(\boldsymbol{\beta}) = -(t_i \log y_i + (1 - t_i) \log(1 - y_i))$$

未知データの予測

未知データ

$x_1, x_2, \cdots, x_N$

予測値

$$y = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_N x_N))}$$

正解ラベルの予測

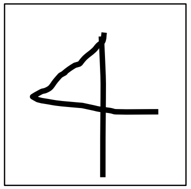
$y_i > 0.5 \rightarrow$ 正例

$y_i < 0.5 \rightarrow$ 負例

多クラス分類問題

- (例)じゃんけん，ぽん → 相手がパーを出すと予想
  - グー → 負例, Negative, 0(数値)
  - チョキ → 正例, Positive, 1(数値)
  - パー → 負例, Negative, 0(数値)

- (例)数字認識
  - 0 → 負例　5 → 負例
  - 1 → 負例　6 → 負例
  - 2 → 負例　7 → 負例
  - 3 → 負例　8 → 負例
  - 4 → 正例　9 → 負例



多クラスへの拡張①

- データ(特徴量):$\mathbf{x}_i^t = (x_{i1}, x_{i2}, \cdots, x_{iN})$ ($i=1,2,\cdots,P$)
  - データ数:P個, 特徴量の次元数:N
- クラス数はC個($k=1,2,\cdots,C$)

- どう予測するのか?
  - 正解がクラスkの場合

  ↓

  - クラス1は負例 → クラス1の確率は0
  - クラス2は負例 → クラス2の確率は0

    ⋮

  - クラスkは正例 → クラスkの確率は1

    ⋮

  - クラスCは負例 → クラスCの確率は0

クラスkが正例となる確率を予測

多クラスへの拡張②

- データ（特徴量）: $\mathbf{x}_i^t = (x_{i1}, x_{i2}, \cdots, x_{iN})$ （i=1,2,・・・,P）
  - データ数：P個，特徴量の次元数：N
- クラス数はC個（k=1,2,・・・,C）

クラスkが正例となる予測値（確率）

$$y_k = \beta_{k0} + \beta_{k1}x_1 + \beta_{k2}x_2 + \cdots + \beta_{kN}x_N$$

線形？

$$y_k = \frac{1}{1 + \exp(-(\beta_{k0} + \beta_{k1}x_1 + \beta_{k2}x_2 + \cdots + \beta_{kN}x_N))}$$

シグモイド関数？

⇩

条件

$$\sum_{k=1}^{C} y_k = 1$$

多クラスへの拡張③

- データ（特徴量）: $\mathbf{x}_i^t = (x_{i1}, x_{i2}, \cdots, x_{iN})$ （i=1,2,・・・,P）
  - データ数：P個，特徴量の次元数：N
- クラス数はC個（k=1,2,・・・,C）

クラスkが正例となる予測値（確率）

$$y_k = \frac{\exp(\beta_{k0} + \beta_{k1}x_1 + \beta_{k2}x_2 + \cdots + \beta_{kN}x_N)}{\sum_{j=1}^{C} \exp(\beta_{j0} + \beta_{j1}x_1 + \beta_{j2}x_2 + \cdots + \beta_{jN}x_N)}$$

$$y_k = \frac{\exp(a_k)}{\sum_{j=1}^{C} \exp(a_j)}$$

$$a_k = \mathbf{w}_k^t \mathbf{x}$$

$$\mathbf{w}_k^t = (\beta_{k0}, \beta_{k1}, \beta_{k2}, \cdots, \beta_{kN})$$

$$\mathbf{x}^t = (1, x_1, x_2 \cdots, x_N)$$

ソフトマックス関数

正解値の与え方

予測値

$$\mathbf{y}^t = (y_1, y_2, \cdots, y_C)$$

$$y_k = \frac{\exp(a_k)}{\sum_{j=1}^{C} \exp(a_j)}$$

$$a_k = \mathbf{w}_k^t \mathbf{x}$$

$$\mathbf{w}_k^t = (\beta_{k0}, \beta_{k1}, \beta_{k2}, \cdots, \beta_{kN})$$

$$\mathbf{x}^t = (1, x_1, x_2 \cdots, x_N)$$

正解値
クラスkが正例の場合
→ 正解クラスのみ1，他は0

$$\mathbf{t}^t = (0,0,\cdots,0,\underset{k\text{番目}}{1},0,\cdots 0)$$

one hot vector

誤差関数（クロスエントロピー）

t_{ik}：データ$\mathbf{x}_i$に対するクラスkの正解値
y_{ik}：データ$\mathbf{x}_i$に対するクラスkの予測値

$$E(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_C) = -\sum_{i=1}^{P} \sum_{k=1}^{C} t_{ik} \log y_{ik}$$

最急降下法による解法

データ$\mathbf{x}_i$に対する誤差関数（オンライン学習）

$$E_i = -\sum_{k=1}^{C} t_{ik} \log y_{ik}$$

全データに対する誤差関数（バッチ学習）

$$E = \sum_{i=1}^{P} E_i = -\sum_{i=1}^{P} \sum_{k=1}^{C} t_{ik} \log y_{ik}$$

最急降下法

$$\boxed{\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \frac{\partial E_i}{\partial \mathbf{w}_j}}$$

$$\boxed{\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \frac{\partial E}{\partial \mathbf{w}_j}}$$

α :学習係数

$$\frac{\partial E_i}{\partial \mathbf{w}_j} = -\sum_{k=1}^{C} \frac{\partial (t_{ik} \log y_{ik})}{\partial y_{ik}} \boxed{\frac{\partial y_{ik}}{\partial a_{ij}}} \boxed{\frac{\partial a_{ij}}{\partial \mathbf{w}_j}}$$

$$= -\sum_{k=1}^{C} \frac{t_{ik}}{y_{ik}} y_{ik} (I_{kj} - y_{ij}) \mathbf{x}_i$$

$$= -\left(\sum_{i=1}^{C} t_{ik} \underline{I_{kj}} - \sum_{i=1}^{C} \underline{t_{ik}} y_{ij} \right) \mathbf{x}_i$$

k=jの時は1, それ以外は0

正解の時は1, それ以外は0

$$= -(t_{ij} - y_{ij}) \mathbf{x}_i = (y_{ij} - t_{ij}) \mathbf{x}_i$$

$$E_i = -\sum_{k=1}^{C} t_{ik} \log y_{ik}$$

$$y_{ik} = \frac{\exp(a_{ik})}{\sum_{j=1}^{C} \exp(a_{ij})}$$

$$a_{ij} = \mathbf{w}_j^t \mathbf{x}_i$$

$$\mathbf{w}_j^t = (\beta_{j0}, \beta_{j1}, \beta_{j2}, \cdots, \beta_{jN})$$

$$\mathbf{x}_i = (1, x_{i1}, x_{i2} \cdots, x_{iN})$$

ソフトマックス関数の微分

$$\boxed{\frac{\partial y_{ik}}{\partial a_{ij}}} = y_{ik} (I_{kj} - y_{ij}) \qquad \boxed{\frac{\partial a_{ij}}{\partial \mathbf{w}_j}} = \mathbf{x}_i$$

ソフトマックス関数の微分

$$y_k = \frac{\exp(a_k)}{\sum_{j=1}^{C} \exp(a_j)}$$

$\boxed{k \neq j}$

$$\frac{\partial y_k}{\partial a_j} = \frac{-\exp(a_j)\exp(a_k)}{\left(\sum_{j=1}^{C} \exp(a_j)\right)^2}$$

$$= -y_k y_j$$

$\boxed{k = j}$

$$\frac{\partial y_k}{\partial a_j} = \frac{\exp(a_k)\sum_{j=1}^{C} \exp(a_j) - \exp(a_k)\exp(a_j)}{\left(\sum_{j=1}^{C} \exp(a_j)\right)^2}$$

$$= y_k(1 - y_j)$$

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$$

k=jの時は1,
それ以外は0

多クラスロジスティック回帰の解法

データ$\mathbf{x}_i$に対する修正
（オンライン学習）

$$\frac{\partial E_i}{\partial \mathbf{w}_j} = (y_{ij} - t_{ij})\mathbf{x}_i$$

予測値と正解値の差

i番目のデータ

$$\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \frac{\partial E_i}{\partial \mathbf{w}_j}$$

全てのデータに対する修正
（バッチ学習）

$$\frac{\partial E}{\partial \mathbf{w}_j} = \sum_{i=1}^{P} (y_{ij} - t_{ij})\mathbf{x}_i$$

$$\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \frac{\partial E}{\partial \mathbf{w}_j}$$

確率的勾配降下法（オンライン学習）

while True :

　　差の合計値 = 0

　　for i in range(データ数) :

$$y_i = \frac{\exp(\beta_{i0} + \beta_{i1}x_1 + \beta_{i2}x_2 + \cdots + \beta_{iN}x_N)}{\sum_{j=1}^{C}\exp(\beta_{j0} + \beta_{j1}x_1 + \beta_{j2}x_2 + \cdots + \beta_{jN}x_N)}$$

　　　　予測値$\mathbf{y_i}$を計算

　　　　予測値$\mathbf{y_i}$と正解値$\mathbf{t_i}$のクロスエントロピーを計算

　　　　パラメータβを修正

$$\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \frac{\partial E_i}{\partial \mathbf{w}_j}$$

$$\frac{\partial E_i}{\partial \mathbf{w}_j} = (y_{ij} - t_{ij})\mathbf{x}_i$$

　　　　差の合計値 += クロスエントロピー

　　if 差の合計値 < ε :

　　　　break

微小値

確率的勾配降下法（オンライン学習）

$\mathbf{x}_i (i = 1, 2, \cdots, P)$

入力値

ソフトマックス関数

$\mathbf{y}_i^t = (y_{1i}, y_{2i}, \cdots, y_{Ci})$

予測値

正解値

$\mathbf{t}_i^t = (0, 0, \cdots, 0, 1, 0, \cdots 0)$

$$\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \frac{\partial E_i}{\partial \mathbf{w}_j}$$

$$\frac{\partial E_i}{\partial \mathbf{w}_j} = (y_{ij} - t_{ij}) \mathbf{x}_i$$

クロスエントロピー

$$E_i(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_C) = -\sum_{k=1}^{C} t_{ik} \log y_{ik}$$

未知データの予測

未知データ

$x_1, x_2, \cdots, x_N$

予測値

$y_k (k=1,2,\cdots,C)$を予測

$$y_k = \frac{\exp(\beta_{k0} + \beta_{k1}x_1 + \beta_{k2}x_2 + \cdots + \beta_{kN}x_N)}{\sum_{j=1}^{C} \exp(\beta_{j0} + \beta_{j1}x_1 + \beta_{j2}x_2 + \cdots + \beta_{jN}x_N)}$$

正解ラベルの予測

最大値y_cの探索
→ クラスcを予測結果とする

scikit-learnによるロジスティック回帰

ロジスティック回帰で扱う問題（breast cancer）

- breast cancer dataset

用途	クラス分類
データ数	569
特徴量	30
目的変数	2

クラス名	データ数
malignant	212
benign	357

- 二値分類問題
  - 正例（ラベル名：malignant，数値：1）
  - 負例（ラベル名：benign，数値：0）

scikit-learn（Python）を用いて分類問題を解く際の注意

- 分類問題では正解値として正解ラベルを学習時に与える
  - 二値分類問題では，正例（1）もしくは負例（0）
  - 正例である確率ではないことに注意
  - 正解値として正例である確率を学習時に与えたい場合は，回帰問題として解く

- 未知データを予測する際は，正例である確率，正例でない確率を計算した上で，正解ラベルを予測
  - 正解ラベルの予測 → predict
  - 正例である確率，正例でない確率の予測 →predict_proba

Cancer_logistic.py

```
import sys
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix,
precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression

# データのロード
cancer = datasets.load_breast_cancer()

# 特徴量（30次元）
feature_names=cancer.feature_names
data = cancer.data

# 目的変数( malignant, benign )
name = cancer.target_names
label = cancer.target
```

パッケージのimport

breast cancerデータセットの読み込み

特徴量の名前

特徴量のデータ：（569,30）

目的変数の名前

目的変数の値（正解ラベル）：569

ホールドアウト法
train_test_split

```
# ホールドアウト法（学習データ，テストデータ）
train_data, test_data, train_label, test_label = train_test_split(data, label, test_size=0.5,
random_state=None)

model = LogisticRegression(C=1.0,penalty='l2',solver='lbfgs',max_iter=100)

# 学習
model.fit(train_data, train_label)

# 予測
predict = model.predict(test_data)

# 係数と切片
print( '¥n 係数ベクトル : ' , model.coef_ )
print( ' 切片 : ' , model.intercept_)

# 予測値，正解ラベル
print( '¥n [ 予測値  :  正解ラベル ]' )
predict_proba = model.predict_proba(test_data)
for i in range(len(test_label)):
    print( predict_proba[i] , ':' , test_label[i] )
```

ロジスティック回帰
LogisticRegression

学習
fit（学習データ，教師ラベル）

予測
predict（データ）
戻り値：ラベル（0もしくは1）

coef_：係数ベクトル
intercept_：切片

予測値の計算
predict_proba（データ）
戻り値：クラスごとの確率

```
# 予測結果の表示
print( "¥n [ 予測結果 ]" )
print( ' accuracy  : ' , accuracy_score(test_label, predict) )
print( ' precision : ' , precision_score(test_label, predict) )
print( ' recall    : ' , recall_score(test_label, predict) )
print( ' f1-score  : ' , f1_score(test_label, predict) )

print( "¥n [ 予測結果 ]" )
print( classification_report(test_label, predict) )

print( "¥n [ 混同行列 ]" )
print( confusion_matrix(test_label, predict) )
```

accuracyの計算
accuracy_score(正解, 予測)

precisionの計算
precision_score(正解, 予測)

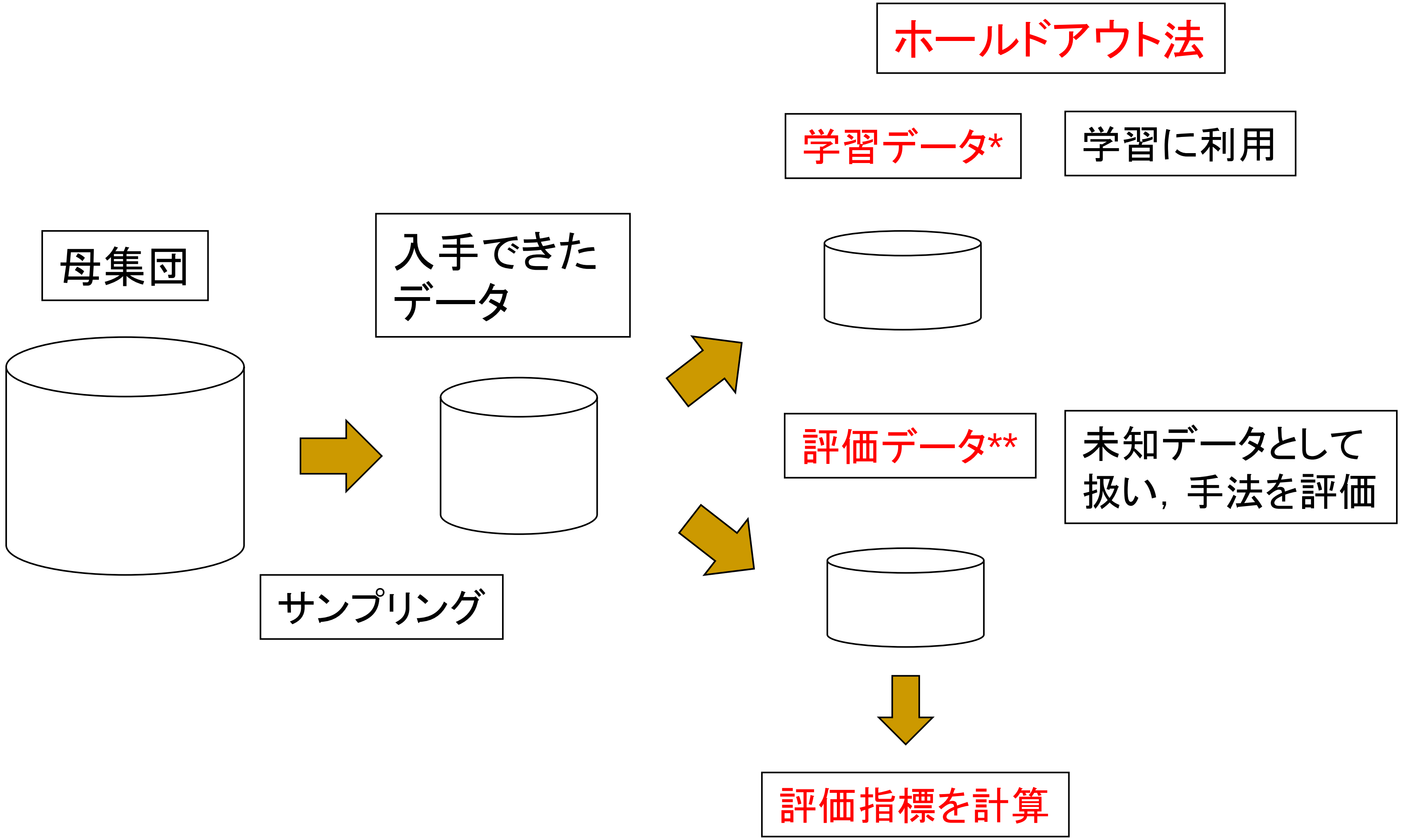
recallの計算
recall_score(正解, 予測)

f値の計算
f1_score(正解, 予測)

上記の評価指標の計算
classification_report(正解, 予測)

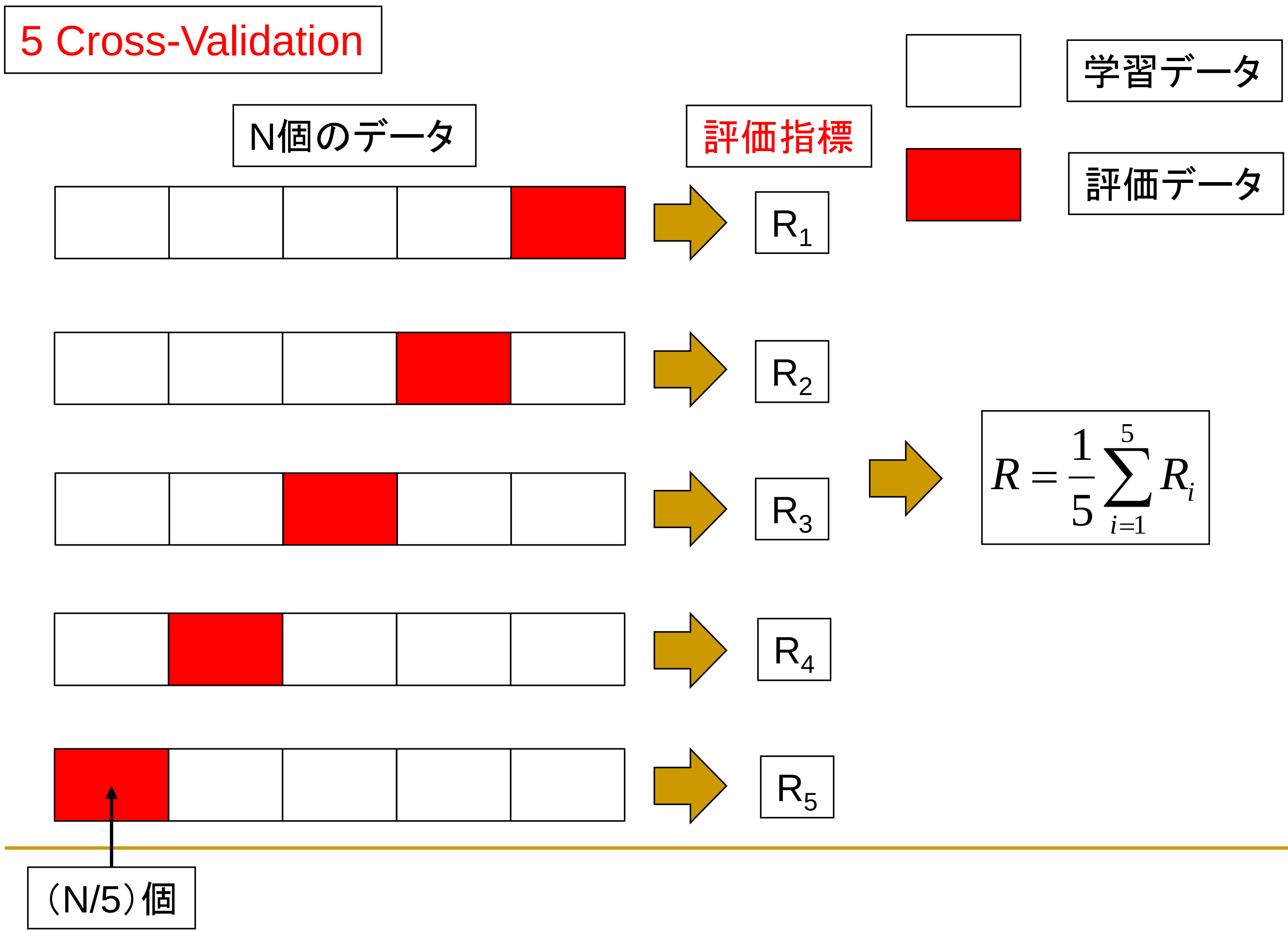
混同行列の計算
confusion_matrix(正解, 予測)

学習データと評価データ



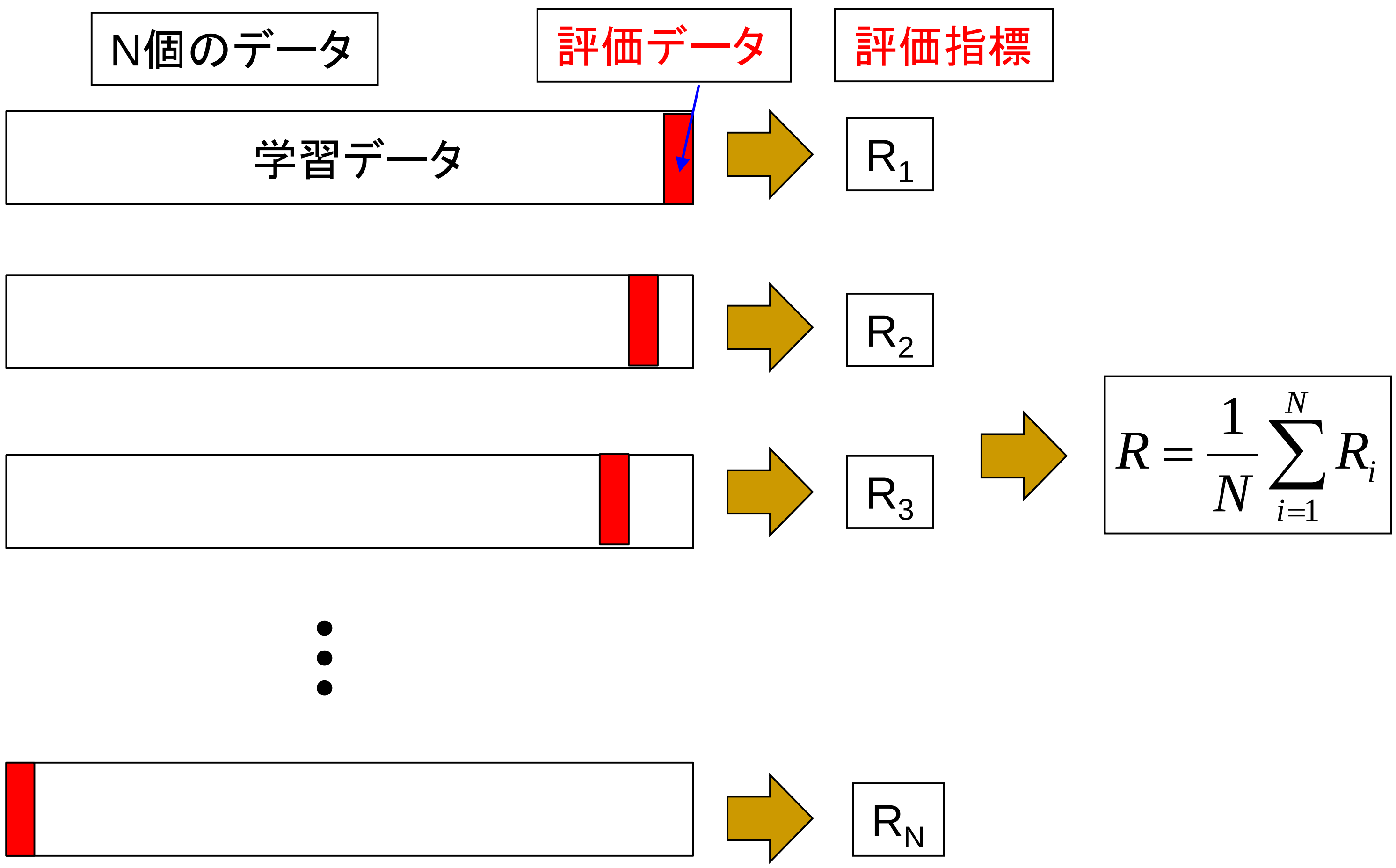
*訓練データとも呼ばれます
**テストデータとも呼ばれます

交差検証（Cross Validation）



$$R = \frac{1}{5}\sum_{i=1}^{5} R_i$$

一つ抜き法（Leave One Out）



ホールドアウト法①

from sklearn.model_selection import train_test_split

ホールドアウト
train_test_split（データ，正解ラベル，test_size=テストデータの割合）

返り値
学習データ，テストデータ，学習データのラベル，テストデータのラベル

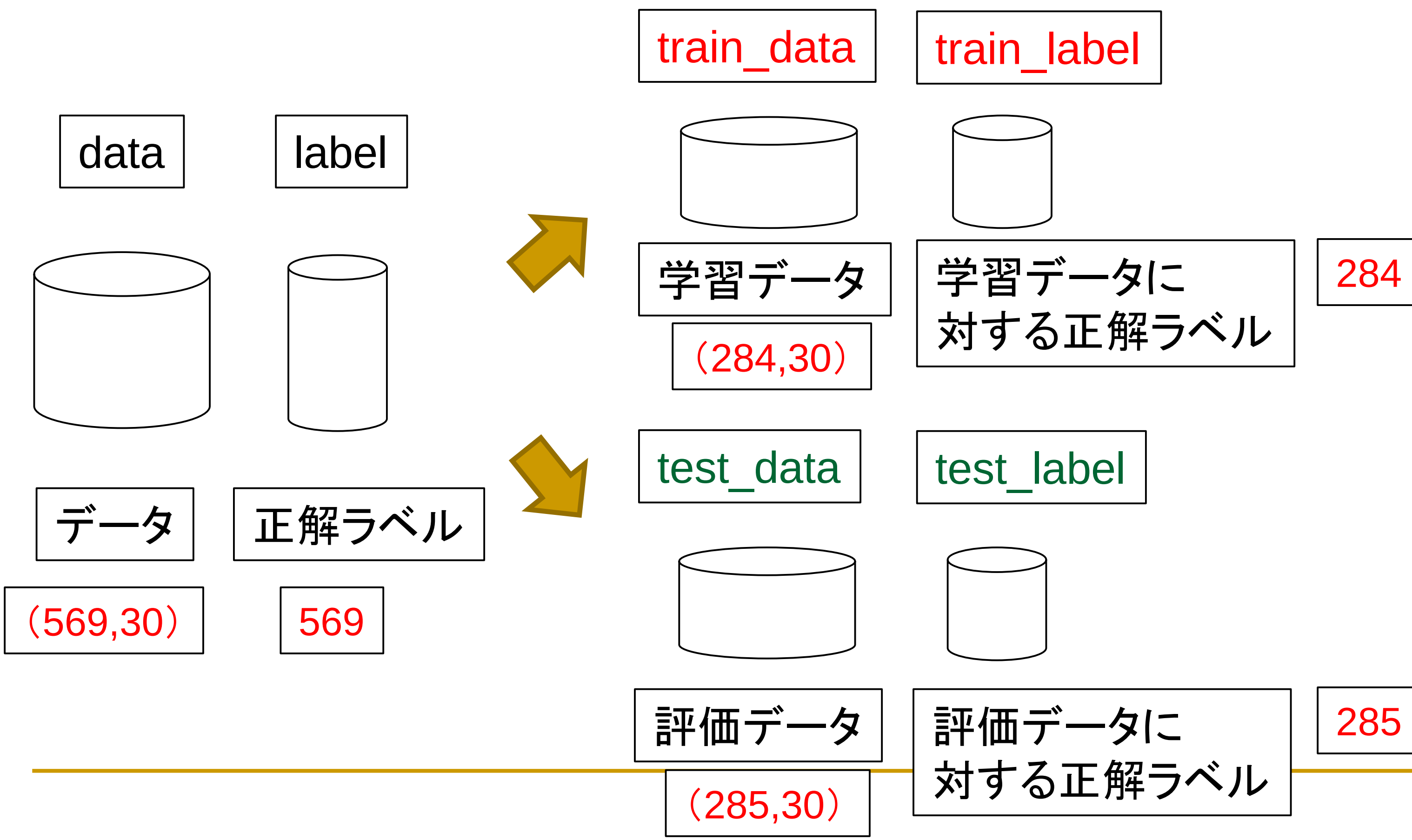
```
train_data, test_data, train_label, test_label =
train_test_split(data, label, test_size=0.5, random_state=None)
```

テストデータの割合は50%

乱数によって毎回分割方法を変える

ホールドアウト法②

```
train_data, test_data, train_label, test_label =
train_test_split(data, label, test_size=0.5, random_state=None)
```



LogisticRegression

```
from sklearn.linear_model import LogisticRegression
LogisticRegression(C=正則化のパラメータ, penalty=正則化項のノルム
,solver=最適化手法, max_iter=探索回数)
```

```
model =
LogisticRegression(C=1.0,penalty='l2',solver='lbfgs',max_iter=100)
```

変数model
に代入

正則化のパラメータ
C=1.0

L2ノルムの場合
penalty='l2'

L1ノルムの場合
penalty='l1'

solverの指定
‘newton-cg’
‘lbfgs’,
‘liblinear’
‘sag’,
‘saga’

探索回数の指定

デフォルトの場合

```
model = LogisticRegression()
```

正則化

誤差関数（クロスエントロピー）

$$E_i(\boldsymbol{\beta}) = -(t_i \log y_i + (1-t_i)\log(1-y_i))$$

$$E_i'(\boldsymbol{\beta}) = -(t_i \log y_i + (1-t_i)\log(1-y_i)) + \boxed{C\sum_{j=1}^{N}\beta_j^2}$$

正則化項

L1ノルム $\quad \|\boldsymbol{\beta}\|_1 = \sum_{i=1}^{N}|\beta_i|$

C：正則化パラメータ

L2ノルム $\quad \|\boldsymbol{\beta}\|_2 = \sqrt{\sum_{i=1}^{N}|\beta_i|^2}$

*L2ノルム正則化はweight decay（重み減衰）とも呼ばれます

学習と予測

学習

fit(学習データ, 学習データに対する正解ラベル)

予測

predict(予測したいデータ)

```
# 学習
model.fit(train_data, train_label)

# 予測（テストデータの場合）
predict = model.predict(test_data)

# 予測（学習データの場合）
predict = model.predict(train_data)
```

予測

predict(予測したいデータ)
ラベル（クラス番号）を予測

decision_function(予測したいデータ)
信頼度（confidence score）を計算
二値分類において，負の場合→クラス0，正の場合→クラス1

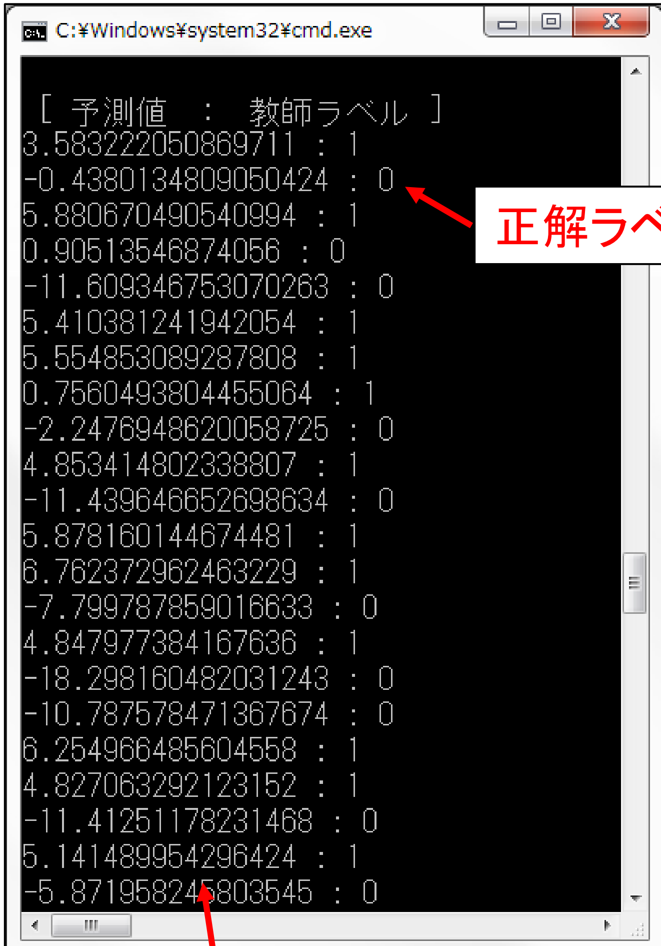
predict_proba(予測したいデータ)
クラスごとの所属確率を計算

```
# 信頼度の計算
df = model.decision_function(test_data)

# クラスごとの所属確率を計算
predict_proba = model.predict_proba(test_data)
```

```
# 信頼度の計算
df = model.decision_function(test_data)
```

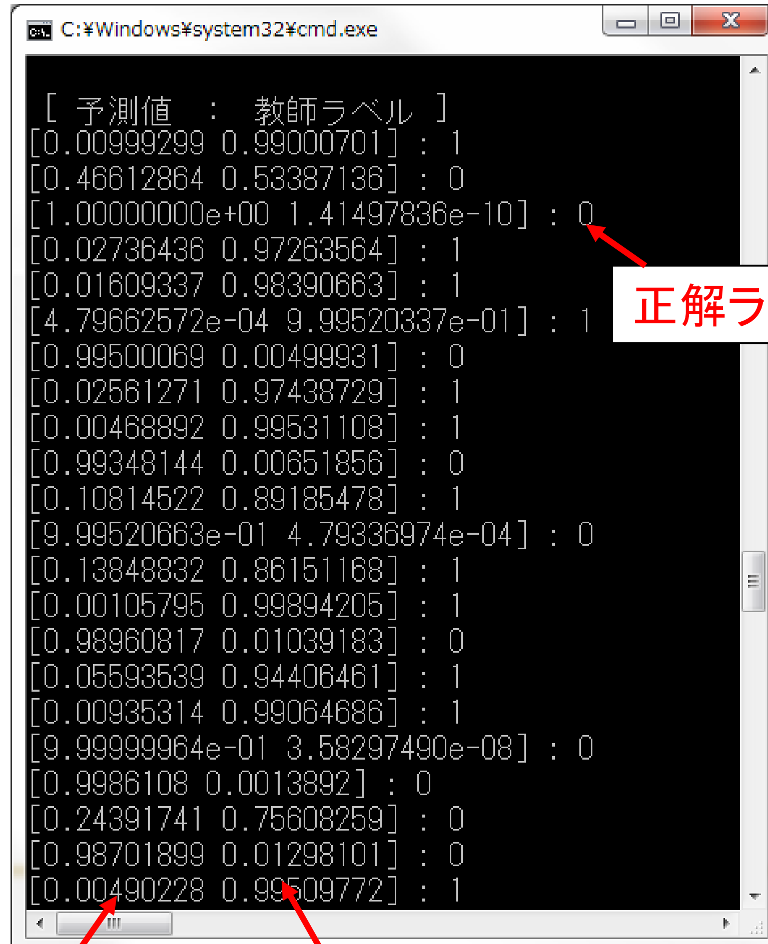
```
# クラスごとの所属確率を計算
predict_proba = model.predict_proba(train_data)
```



```
C:¥Windows¥system32¥cmd.exe
[ 予測値 : 教師ラベル ]
3.583222050869711 : 1
-0.4380134809050424 : 0
5.880670490540994 : 1
0.90513546874056 : 0
-11.609346753070263 : 0
5.410381241942054 : 1
5.554853089287808 : 1
0.7560493804455064 : 1
-2.2476948620058725 : 0
4.853414802338807 : 1
-11.439646652698634 : 0
5.878160144674481 : 1
6.762372962463229 : 1
-7.799787859016633 : 0
4.847977384167636 : 1
-18.298160482031243 : 0
-10.787578471367674 : 0
6.254966485604558 : 1
4.827063292123152 : 1
-11.41251178231468 : 0
5.141489954296424 : 1
-5.871958245803545 : 0
```

正解ラベル

正の場合→1
負の場合→0



```
C:¥Windows¥system32¥cmd.exe
[ 予測値 : 教師ラベル ]
[0.00999299 0.99000701] : 1
[0.46612864 0.53387136] : 0
[1.00000000e+00 1.41497836e-10] : 0
[0.02736436 0.97263564] : 1
[0.01609337 0.98390663] : 1
[4.79662572e-04 9.99520337e-01] : 1
[0.99500069 0.00499931] : 0
[0.02561271 0.97438729] : 1
[0.00468892 0.99531108] : 1
[0.99348144 0.00651856] : 0
[0.10814522 0.89185478] : 1
[9.99520663e-01 4.79336974e-04] : 0
[0.13848832 0.86151168] : 1
[0.00105795 0.99894205] : 1
[0.98960817 0.01039183] : 0
[0.05593539 0.94406461] : 1
[0.00935314 0.99064686] : 1
[9.99999964e-01 3.58297490e-08] : 0
[0.9986108 0.0013892] : 0
[0.24391741 0.75608259] : 0
[0.98701899 0.01298101] : 0
[0.00490228 0.99509772] : 1
```

正解ラベル

0の予測確率

1の予測確率

学習後のパラメータ

coef_ : 係数ベクトル
intercept_ : 切片

```
print( '¥n 係数ベクトル : ' , model.coef_ )
print( ' 切片 : ' , model.intercept_)
```

C:¥Windows¥system32¥cmd.exe

```
係数ベクトル :  [[ 1.60457236  0.06474629 -0.15965411  0.01346804 -0.06164696 -0.31707709
 -0.43906814 -0.18363863 -0.12257091 -0.01505985  0.06746939  0.8655651
  0.69294339 -0.09065138 -0.00531991 -0.06921985 -0.09643035 -0.02622336
 -0.0368389  -0.00527471  1.62767301 -0.25156929 -0.09968361 -0.03169438
 -0.11311723 -0.93848942 -1.16598071 -0.36693192 -0.34114202 -0.07335494]]
切片 :  [0.3492502]
```

係数ベクトル

切片

学習モデル

$$y = \frac{1}{1 + \exp(-(0.349 + 1.60x_1 + 0.064x_2 + \cdots - 0.073x_{30}))}$$

評価指標①

- 二値分類の場合
  - 正例（1, Positive），負例（0, Negative）

Confusin Matrix
混同行列

予測結果

データ	正解	予測
1	0	0
2	0	1
3	0	0
4	0	0
5	0	0
6	1	0
7	1	1
8	1	1
9	1	0
10	1	1

		予測	
	ラベル	0	1
正解	0	4	1
	1	2	3

正解が0のデータを1と予測した個数

正解が0のデータを0と予測した個数

正解が1のデータを0と予測した個数

正解が1のデータを1と予測した個数

評価指標②

混同行列

		予測	
		0	1
正解	0	TN	FP
	1	FN	TP

TN：True Negative
FP：False Positive
FN：False Negative
TP：True Positive

正解率（accuracy）

$$accuracy = \frac{TN + TP}{TN + FP + FN + TP}$$

全データ中，正解した割合

適合率（precision）

$$precision = \frac{TP}{FP + TP}$$

Positive（1）と予測されたデータの中で正解した割合

再現率（recall）

$$recall = \frac{TP}{FN + TP}$$

実際にPositive（1）のデータ中，正解した割合

F値（F-measure）

$$F - measure = \frac{2 \times precision \times recall}{precision + recall}$$

precisionとrecallの調和平均

評価指標③

混同行列

		予測	
	ラベル	0	1
正解	0	4	1
	1	2	3

		予測	
		0	1
正解	0	TN	FP
	1	FN	TP

正解率（accuracy）

$$accuracy = \frac{TN + TP}{TN + FP + FN + TP} = \frac{4+3}{4+1+2+3} = 0.7$$

適合率（precision）

$$precision = \frac{TP}{FP + TP} = \frac{3}{1+3} = 0.75$$

再現率（recall）

$$recall = \frac{TP}{FN + TP} = \frac{3}{2+3} = 0.6$$

F値（F-measure）

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.75 \times 0.6}{0.75 + 0.6} = 0.666$$

評価指標④

- 正例：800個，負例：200個のデータ

ほぼ全てPositiveと予測した場合の混同行列

		予測	
	ラベル	0	1
正解	0	1	199
	1	0	800

	評価指標
accuracy	0.801
precision	0.801
recall	1.0
F値	0.889

予測能力が高いと言えるのか

Negativeのデータから見ると予測できていない

評価指標を求める上での注意①

- 正例と負例を反転させると評価指標が異なる

混同行列

		予測	
	ラベル	0	1
正解	0	4	1
	1	2	3

		予測	
		0	1
正解	0	TN	FP
	1	FN	TP

正例と負例を反転

		予測	
	ラベル	1	0
正解	1	4	1
	0	2	3

		予測	
		1	0
正解	1	TP	FN
	0	FP	TN

評価指標を求める上での注意②

混同行列

		予測	
	ラベル	1	0
正解	1	4	1
	0	2	3

		予測	
		1	0
正解	1	TP	FN
	0	FP	TN

正解率(accuracy)

$$accuracy = \frac{TN + TP}{TN + FP + FN + TP} = \frac{4+3}{4+1+2+3} = 0.7$$

適合率(precision)

$$precision = \frac{TP}{FP + TP} = \frac{4}{2+4} = 0.666$$

再現率(recall)

$$recall = \frac{TP}{FN + TP} = \frac{4}{1+4} = 0.8$$

F値(F-measure)

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.666 \times 0.8}{0.666 + 0.8} = 0.727$$

評価指標を求める上での注意③

混同行列①

		予測	
	ラベル	0	1
正解	0	4	1
	1	2	3

混同行列②

		予測	
	ラベル	1	0
正解	1	4	1
	0	2	3

	混同行列①	混同行列②
accuracy	0.7	0.7
precision	0.75	0.666
recall	0.6	0.8
F値	0.666	0.727

どちらを正例（Positive）とするかで評価指標が異なる

評価指標を求める上での注意④

- 正例：800個，負例：200個のデータ

混同行列①

		予測	
	ラベル	0	1
正解	0	1	199
	1	0	800

混同行列②

		予測	
	ラベル	0	1
正解	0	1	199
	1	0	800

	混同行列①	混同行列②
accuracy	0.801	0.801
precision	0.801	1.0
recall	1.0	0.005
F値	0.889	0.01

予測できていない

正しい（？）評価指標の計算

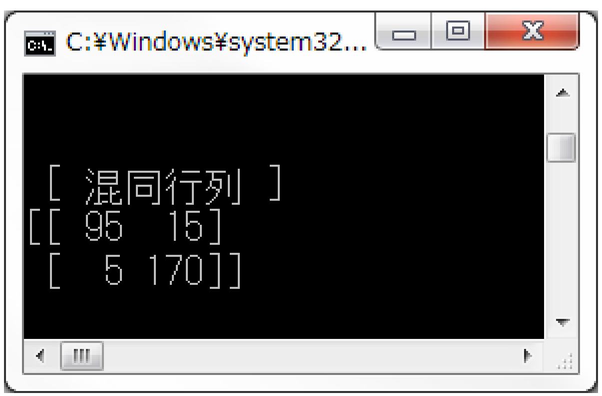
- マクロ平均
  - 正例と負例を反転させ，評価指標を求めた後，二つの評価指標の平均を求める
- マイクロ平均
  - 正例と負例を反転させ，二つの混合行列のTP，TN，FP，FNを合計した後，評価指標を計算
- 加重平均
  - 正例と負例を反転させ，評価指標を求めた後，データ数に応じて，加重平均をとる
  - データ数が同一の場合，マクロ平均と同じ

混同行列の計算

混同行列

```
from sklearn.metrics import confusion_matrix
confusion_matrix(正解ラベル, 予測結果)
```

```
print( "¥n [ 混同行列 ]" )
print( confusion_matrix(test_label, predict) )
```



		予測	
	ラベル	0	1
正解	0	95	15
	1	5	170

評価指標の計算①

accuracy

```
from sklearn.metrics import accuracy_score
accuracy_score（正解ラベル, 予測結果）
```

precision

```
from sklearn.metrics import precision_score
precision_score（正解ラベル, 予測結果, pos_label=1,
average=評価指標の求め方）
```

正例のラベル
デフォルトは1

'binary' : pos_labelで指定したラベル（デフォルト）
'micro' : マイクロ平均
'macro' : マクロ平均
'weighted' : 加重平均

評価指標の計算②

recall

```
from sklearn.metrics import recall_score
recall_score（正解ラベル, 予測結果, pos_label=1,
average=評価指標の求め方）
```

F値

```
from sklearn.metrics import f1_score
f1_score（正解ラベル, 予測結果, pos_label=1,
average=評価指標の求め方）
```

*引数はprecisionと同じです

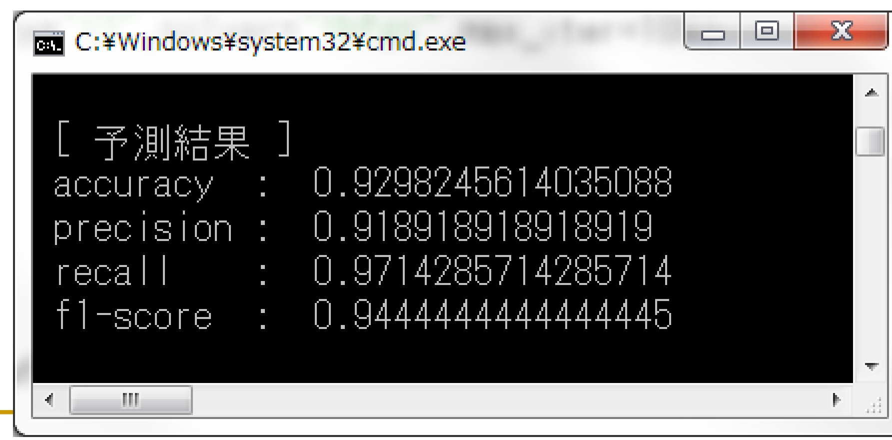
評価指標の計算③

```
print( "¥n [ 予測結果 ]" )
print( ' accuracy  : ' , accuracy_score(test_label, predict) )
print( ' precision : ' , precision_score(test_label, predict) )
print( ' recall    : ' , recall_score(test_label, predict) )
print( ' f1-score  : ' , f1_score(test_label, predict) )
```

混同行列

		予測	
	ラベル	0	1
正解	0	95	15
	1	5	170

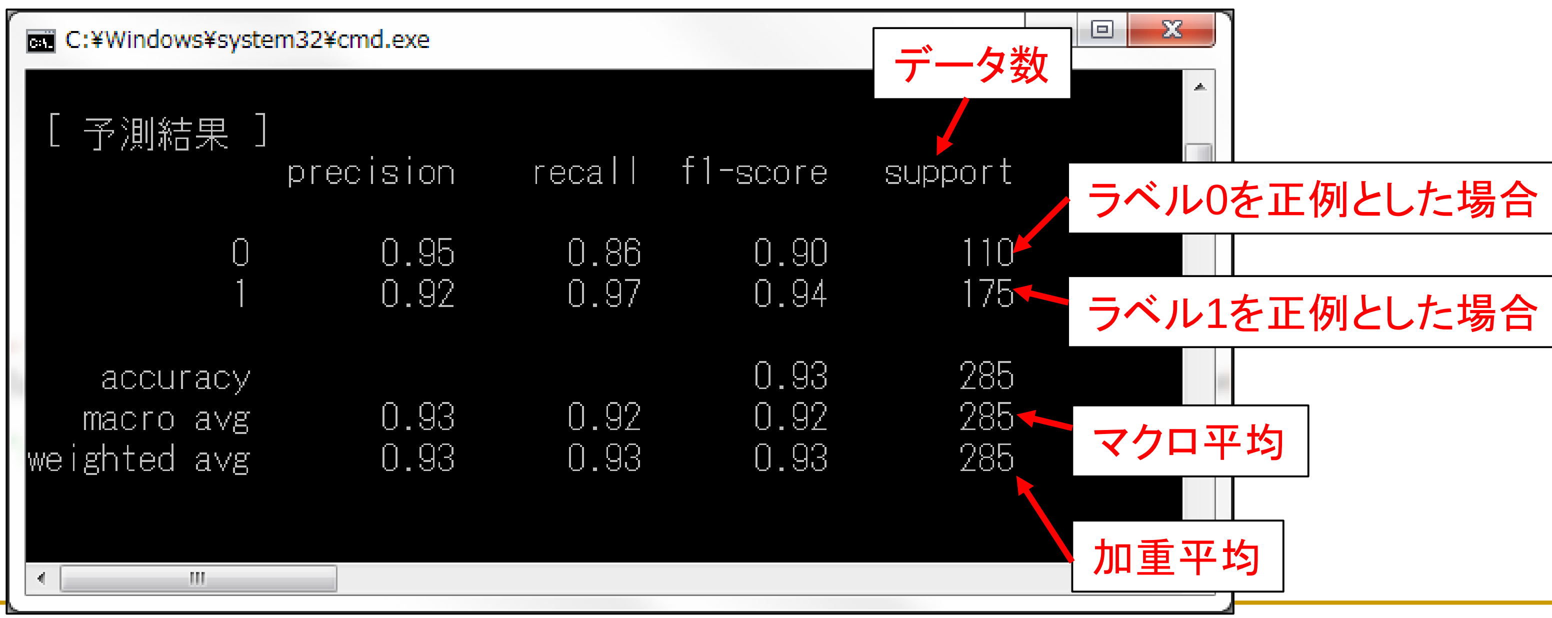
評価指標



まとめて表示する場合

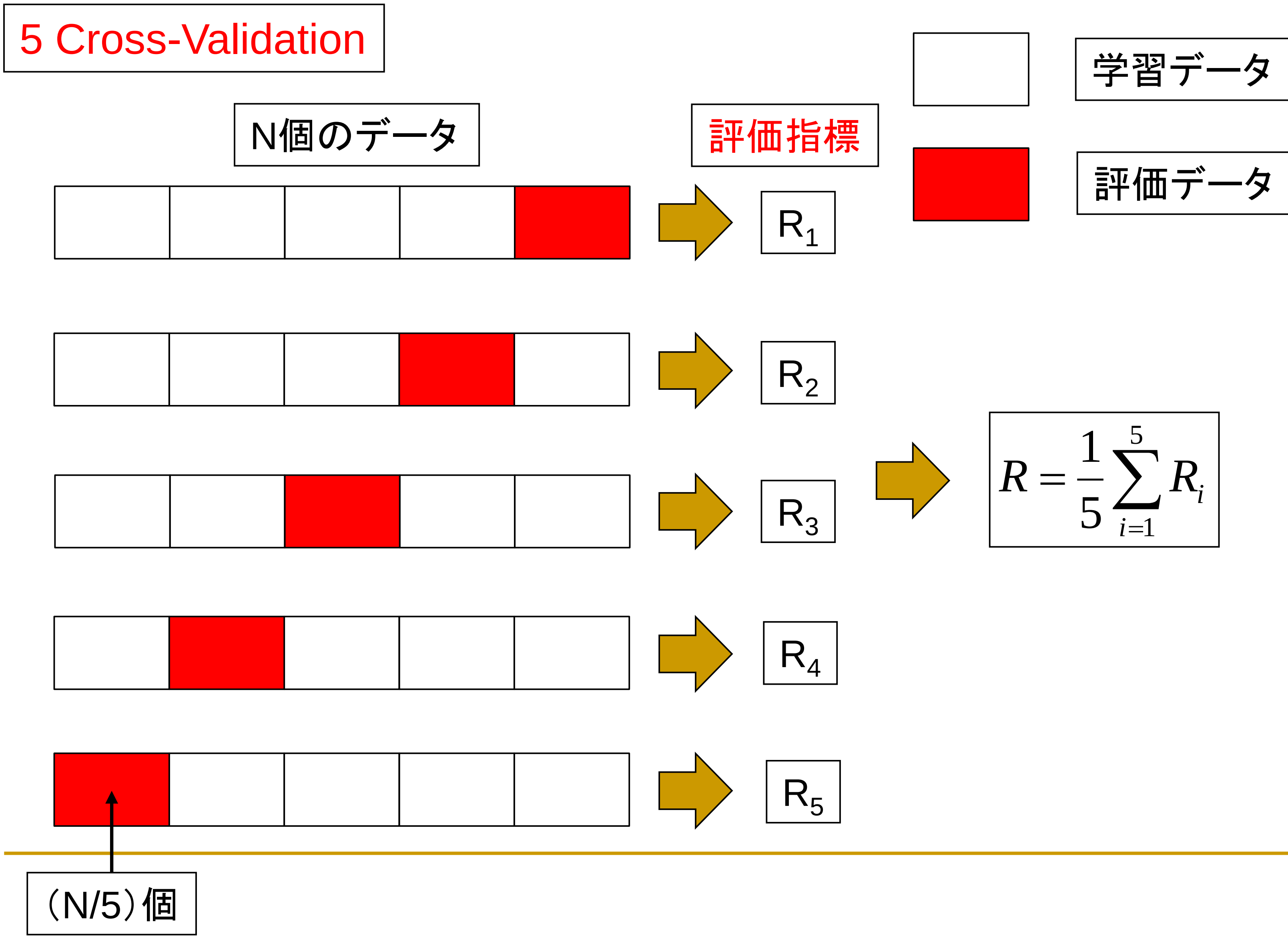
```
from sklearn.metrics import classification_report
classification_report(正解ラベル, 予測結果)
```

```
print( "¥n [ 予測結果 ]" )
print( classification_report(test_label, predict) )
```



交差検証

交差検証（Cross Validation）



交差検証(logistic_CV.py)

```
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate

# データのロード
cancer = datasets.load_breast_cancer()

# 特徴量(30次元)
feature_names=cancer.feature_names
data = cancer.data

# 目的変数( malignant, benign )
name = cancer.target_names
label = cancer.target
```

パッケージのimport

交叉検証にimportが必要

breast cancerデータセットの読み込み

特徴量の名前

特徴量のデータ:(569,30)

目的変数の名前

目的変数の値(正解ラベル):569

```
# ロジスティック回帰
model = LogisticRegression(C=1.0,penalty='l2',solver='lbfgs',max_iter=100)

# 交差検証
score = { "accuracy": "accuracy",
          "precision": "precision_macro",
          "recall": "recall_macro",
          "f":"f1_macro"
          }

result = cross_validate(model, data, label, cv=5, scoring=score)

# 結果の表示
for i , j in result.items():
    print( " {0:15s} : {1}".format( i , j ) )
```

ロジスティック回帰

求めたい評価指標
dict（辞書）形式で指定

score = { 'キーワード' : '求めたい評価指標' }

学習データ

正解データ

交差検証する回数

求めたい評価指標

```
from sklearn.model_selection import cross_validate
cross_validate( モデル, データ, 正解ラベル, cv=交差検証の回数,
scoring=評価指標 )
```

```
result = cross_validate(model, data, label, cv=5, scoring=score)

for i , j in result.items():
    print( " {0:15s} : {1}".format( i , j ) )
```

dict（辞書）形式

①　②　③　④　⑤

C:¥Windows¥system32¥cmd.exe

```
fit_time        : [0.0156002  0.0156002  0.0156002  0.03119993 0.03120041]
score_time      : [0.01559973 0.01559973 0.         0.         0.        ]
test_accuracy   : [0.94782609 0.93913043 0.96460177 0.92035398 0.95575221]
test_precision  : [0.95428475 0.93688845 0.96210597 0.91328904 0.95083056]
test_recall     : [0.93491602 0.93265504 0.96210597 0.91716968 0.95506372]
test_f          : [0.94314436 0.9346856  0.96210597 0.91514393 0.95285774]
```

学習時間

予測時間

score = { キーワード : 評価指標 }
で指定したキーワード

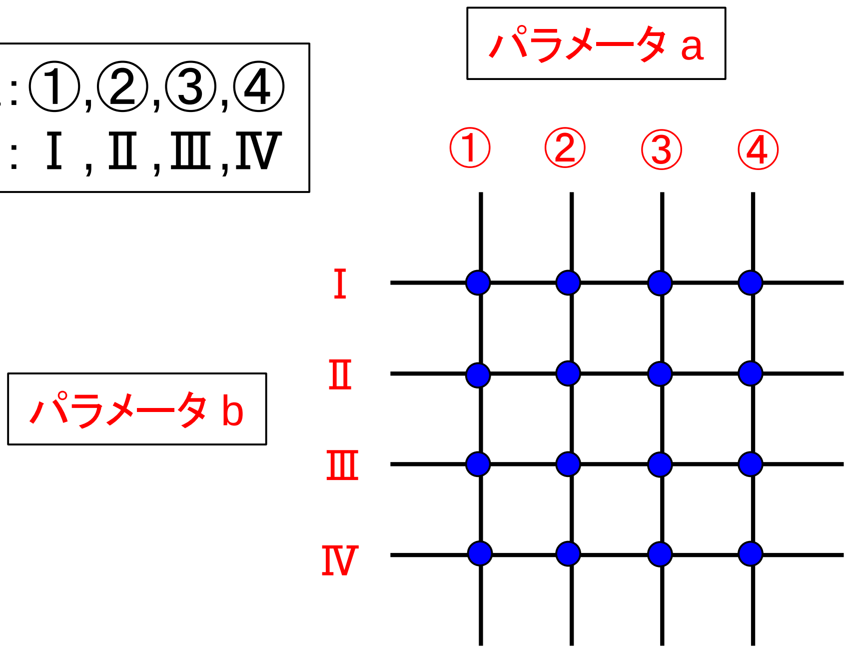
パラメータの決め方

グリッドサーチ

グリッドサーチ

- 全てのパラメータの値を組み合わせ，学習→予測
- 最良の予測結果からパラメータを決定

パラメータ a:①,②,③,④
パラメータ b: Ⅰ, Ⅱ, Ⅲ, Ⅳ



4×4通りのモデルを学習
→最良の予測結果の
パラメータを採用

*理工学ITCのPCではCancer_logistic-GS-ITC.pyを実行して下さい

グリッドサーチ（logistic_GS.py）

グリッドサーチのためにimportが必要

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression

# データのロード
cancer = datasets.load_breast_cancer()

# 特徴量（30次元）
feature_names=cancer.feature_names
data = cancer.data

# 目的変数( malignant, benign )
name = cancer.target_names
label = cancer.target
```

breast cancerデータセットの読み込み

特徴量の名前

特徴量のデータ：（569,30）

目的変数の名前

目的変数の値（正解ラベル）：569

ホールドアウト法

```
# ホールドアウト法（学習データ，テストデータ）
train_data, test_data, train_label, test_label = train_test_split(data, label,
test_size=0.5, random_state=None)

# グリッドサーチ
parameters = {'C': [ 0.2, 0.4, 0.6, 0.8, 1.0 ],
          'penalty' : [ 'l2' , 'none' ],
          'solver' : [ 'newton-cg', 'lbfgs' ],
          'max_iter' : [ 100 , 200 ]
          }

# ロジスティック回帰
model = GridSearchCV(LogisticRegression(), parameters, cv=5)

# 学習
model.fit(train_data, train_label)

# 最良モデル
best_model = model.best_estimator_
```

グリッドサーチで探索するパラメータ

dict（辞書型）
parameters = { 'パラメータ名': [値] }

パラメータ

交叉検証の数

ロジスティック回帰によるグリッドサーチ

学習

best_estimator_
最良モデル（modelも最良モデルが求まっている）

```
print( "¥n [ 最良なパラメータ ]" )
print( model.best_params_ )

# 予測
predict = best_model.predict(test_data)

# 係数と切片
print( '¥n 係数ベクトル : ' , best_model.coef_ )
print( ' 切片 : ' , best_model.intercept_)

# 予測値，教師ラベル
print( '¥n [ 予測値 ： 教師ラベル ]' )
predict_proba = best_model.predict_proba(test_data)
for i in range(len(test_label)):
    print( predict_proba[i] , ':' , test_label[i] )
```

best_params_
最良モデルのパラメータ

予測

coef_ : 係数ベクトル
intercept_ : 切片

クラスごとの
所属確率の予測

```python
# 予測結果の表示
print( "¥n [ 予測結果 ]" )
print( ' accuracy  : ' , accuracy_score(test_label, predict) )
print( ' precision : ' , precision_score(test_label, predict) )
print( ' recall    : ' , recall_score(test_label, predict) )
print( ' f1-score  : ' , f1_score(test_label, predict) )

print( "¥n [ 予測結果 ]" )
print( classification_report(test_label, predict) )

print( "¥n [ 混同行列 ]" )
print( confusion_matrix(test_label, predict) )
```

評価指標の算出

混同行列

実行結果①

C:¥Windows¥system32¥cmd.exe

```
[ 最良なパラメータ ]
{'C': 0.2, 'max_iter': 100, 'penalty': 'none', 'solver': 'newton-cg'}

係数ベクトル :  [[ 6.80854124e+00 -6.01303337e-01 -4.56924252e-01 -6.27097213e-03
  -3.14085819e+01  2.54913556e+01 -1.92005948e+01 -7.66498303e+01
  -1.81636608e+01  3.29760616e+00 -2.85583974e+01 -2.01935718e+00
   4.40664902e+00 -4.73316860e-01 -4.24949008e+00  5.03952949e+01
   7.91776170e+01 -6.03601303e+00  8.69427803e+00  4.56738419e+00
   9.86857008e+00 -1.12834619e-01 -1.01904042e+00 -6.22796402e-02
  -6.69109678e+01  3.88651682e+01 -2.21534917e+01 -1.32788183e+02
   3.55860195e+00  1.80119193e+00]]
切片 :  [6.54299382]

[ 予測値  :  教師ラベル ]
[1.82071537e-07 9.99999818e-01] : 1
[9.99999999e-01 8.71150126e-10] : 0
[1.71009550e-07 9.99999829e-01] : 1
[0.00123286 0.99876714] : 1
[5.10933027e-08 9.99999949e-01] : 1
[9.99915309e-01 8.46910906e-05] : 0
[2.88657986e-15 1.00000000e+00] : 1
[6.55169232e-04 9.99344831e-01] : 1
[5.3135274e-13 1.0000000e+00] : 1
[2.44915199e-13 1.00000000e+00] : 1
```

グリッドサーチにより求められた最良のパラメータ

係数ベクトル

切片

正解ラベル

0の予測確率

1の予測確率

実行結果②

C:¥Windows¥system32¥cmd.exe

```
[ 予測結果 ]
accuracy  :  0.9403508771929825
precision :  0.9411764705882353
recall    :  0.967032967032967
f1-score  :  0.9539295392953929

[ 予測結果 ]
              precision    recall  f1-score   support

           0       0.94      0.89      0.92       103
           1       0.94      0.97      0.95       182

    accuracy                           0.94       285
   macro avg       0.94      0.93      0.93       285
weighted avg       0.94      0.94      0.94       285


[ 混同行列 ]
[[ 92  11]
 [  6 176]]
```

評価指標の表示

混同行列

交差検証の全結果
（コメントしています）

```
print( "¥n [ 交差検証の全結果 ]" )
for i ,  j in model.cv_results_.items():
    print( i , " : " , j )
```

cv_results_
交差検証の全ての結果
dict（辞書）形式

回帰問題

重回帰分析（線形回帰）

重回帰分析(線形回帰)

- データ(特徴量)*: $\mathbf{x}_i^t = (x_{i1}, x_{i2}, \cdots, x_{iN})$ (i=1,2,・・・,P)
  - データ数:P個, 特徴量の次元数:N
- 正解値*: $t_1, t_2, \cdots, t_p$

$$t_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN} + \varepsilon_i$$

誤差項

$$t_i = \sum_{j=0}^{N} \beta_j x_{ij} + \varepsilon_i \qquad x_{i0} = 1$$

- 最小二乗法による解法

$$E = \sum_{i=1}^{P} \varepsilon_i^2 = \sum_{i=1}^{P} (t_i - \sum_{j=0}^{N} \beta_j x_{ij})^2$$

誤差関数(誤差二乗和)が
最小となる係数を求める

*正しい統計用語(?)では説明変数, 被説明変数

最小二乗法(行列計算)

行列計算の場合

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_P \end{bmatrix} \qquad X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1N} \\ 1 & x_{21} & \cdots & x_{2N} \\ \vdots & \vdots & & \vdots \\ 1 & x_{P1} & \cdots & x_{PN} \end{bmatrix} \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix} \qquad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{bmatrix}$$

$$\mathbf{t} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

最小二乗法

$$\boxed{\boldsymbol{\beta} = (X^t X)^{-1} X^t \mathbf{t}}$$

最急降下法による解法
（詳細は識別モデルの講義で説明します）

誤差関数

$$E = \sum_{i=1}^{P} \varepsilon_i^2 = \sum_{i=1}^{P} (t_i - \sum_{j=0}^{N} \beta_i x_{ij})^2$$

データ1個あたりの誤差

$$E_i = (t_i - \sum_{j=0}^{N} \beta_i x_{ij})^2$$

$$\beta_j' = \beta_j - \alpha \frac{\partial E_i}{\partial \beta_j}$$

α：学習係数

$$\frac{\partial E_i}{\partial \beta_j} = -2(\underbrace{t_i}_{\text{正解値}} - \underbrace{\sum_{j=0}^{N} \beta_i x_{ij}}_{\text{予測値}}) \underbrace{x_{ij}}_{\text{データ}}$$

正解値

予測値

データ

正解値と予測値との誤差

デルタルール

- 確率的勾配降下法（オンライン学習）

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN}$$

$$\beta_j^{'} = \beta_j - \alpha \frac{\partial E_i}{\partial \beta_j}$$

$$\frac{\partial E_i}{\partial \beta_j} = \underline{(y_i - t_i)} x_{ij}$$

データ

正解値と予測値との誤差*

- ロジスティック回帰の解法と同じ
- ニューラルネットワークの学習の原理（デルタルール）

*デルタとも呼ばれています

確率的勾配降下法（デルタルール）

while True :

　差の合計値 = 0

　for i in range(データ数) :

　　予測値y_iを計算 $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_N x_{iN}$

　　予測値y_iと正解値t_iの誤差二乗和を計算

　　パラメータβを修正

　　差の合計値 += 誤差二乗和

$$\beta_j^{'} = \beta_j - \alpha \frac{\partial E_i}{\partial \beta_j}$$

$$\frac{\partial E_i}{\partial \beta_j} = (y_i - t_i) x_{ij}$$

　if 差の合計値 < ε :

　　break

微小値

確率的勾配降下法(オンライン学習)

$\mathbf{x}_i (i = 1, 2, \cdots, P)$

入力値

線形回帰

$$y_i = \sum_{j=0}^{N} \beta_j x_{ij}$$

予測値

正解値

t_i

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial E_i}{\partial \beta_j}$$

$$\frac{\partial E_i}{\partial \beta_j} = (y_i - t_i) x_{ij}$$

誤差二乗和

$$E_i = (t_i - \sum_{j=0}^{N} \beta_j x_{ij})^2$$

Pythonによる重回帰分析

重回帰分析で解く問題

- Boston datset
  - 説明変数:13個 → 住宅価格を予測
- Boston_Regression.py
  - 行列計算による解法
- Boston_Regression_SKL.py
  - scikit-learnを用いた解法

用途	回帰
データ数	506
特徴量	13
目的変数	1

重回帰分析(Boston_Regression.py)

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

# データのロード
boston = datasets.load_boston()

# 特徴量(13次元)
feature_name = boston.feature_names
data = boston.data
data_count , feature_count = data.shape

# 目的変数
label = boston.target

# 切片に対応する1の列を挿入
X = np.ones( (data_count,feature_count+1) )
X[:,1:] = data
```

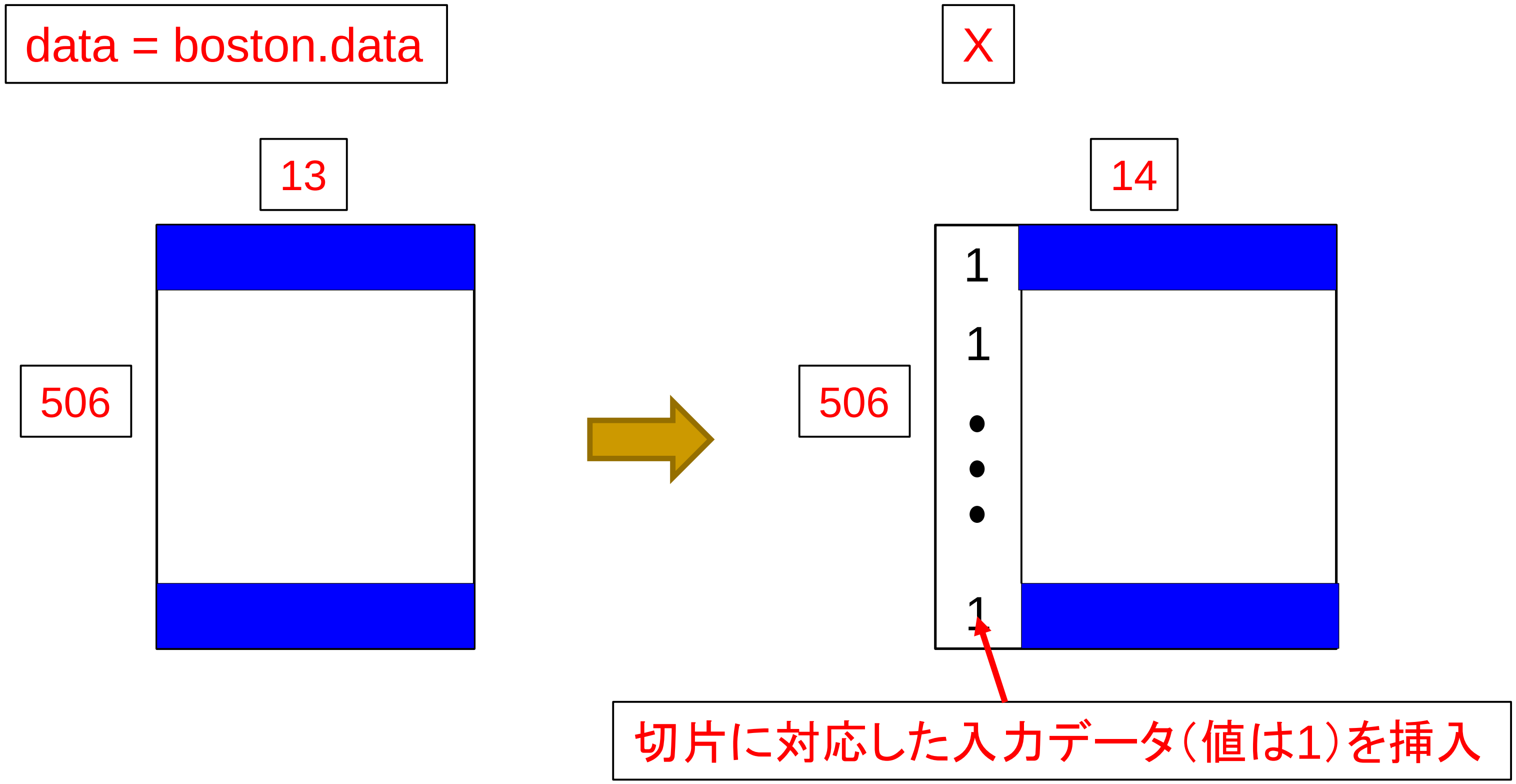
散布図の表示のために必要

Bostonデータセットの読み込み

特徴量の名前

特徴量のデータ:(506,13)

目的変数の値(正解ラベル):506




```
X = np.ones( (data_count,feature_count+1) )
X[:,1:] = data
```

```
# ホールドアウト法(学習データ, テストデータ)
train_X, test_X, train_y, test_y = train_test_split(X, label, test_size=0.5,
random_state=None)

# 係数の計算
work = np.dot( train_X.T , train_X )
work1 = np.linalg.inv( work )
work2 = np.dot( work1 , train_X.T )
w = np.dot( work2 , train_y )

print(  '¥n [ 係数 ]' )
print( w[1:] )
print(  ' [ 切片 ]' )
print( w[0] )

# 予測
predict = np.dot( test_X , w )
```

ホールドアウト法

$\boldsymbol{\beta} = (X'X)^{-1}X'\mathbf{y}$

$X'X$

$(X'X)^{-1}$

$(X'X)^{-1}X'$

$(X'X)^{-1}X'\mathbf{y}$

係数 : w[1]～1[13]

切片 : w[0]

テストデータの予測

$\mathbf{y} = X\boldsymbol{\beta}$

```
# 予測値, 正解値
print( '¥n [ 予測値 ： 正解値 ]' )
for i in range(len(test_y)):
    print( predict[i] , ':' , test_y[i] )

# R2を求める
r = np.corrcoef(test_y, predict)
print( '¥n [ R2 ]' )
print( ' テストデータ :' , r[0,1]*r[0,1] )

# 散布図の描画
fig = plt.figure()
plt.scatter( test_y , predict )
plt.xlabel("Correct")
plt.ylabel("Predict")
fig.savefig("result.png")
```

corrcoef（vec1, vec2）
vec1とvec2の相関行列を求める

r[0.0]	r[0.1]
r[1.0]	r[1.1]

x軸のデータ

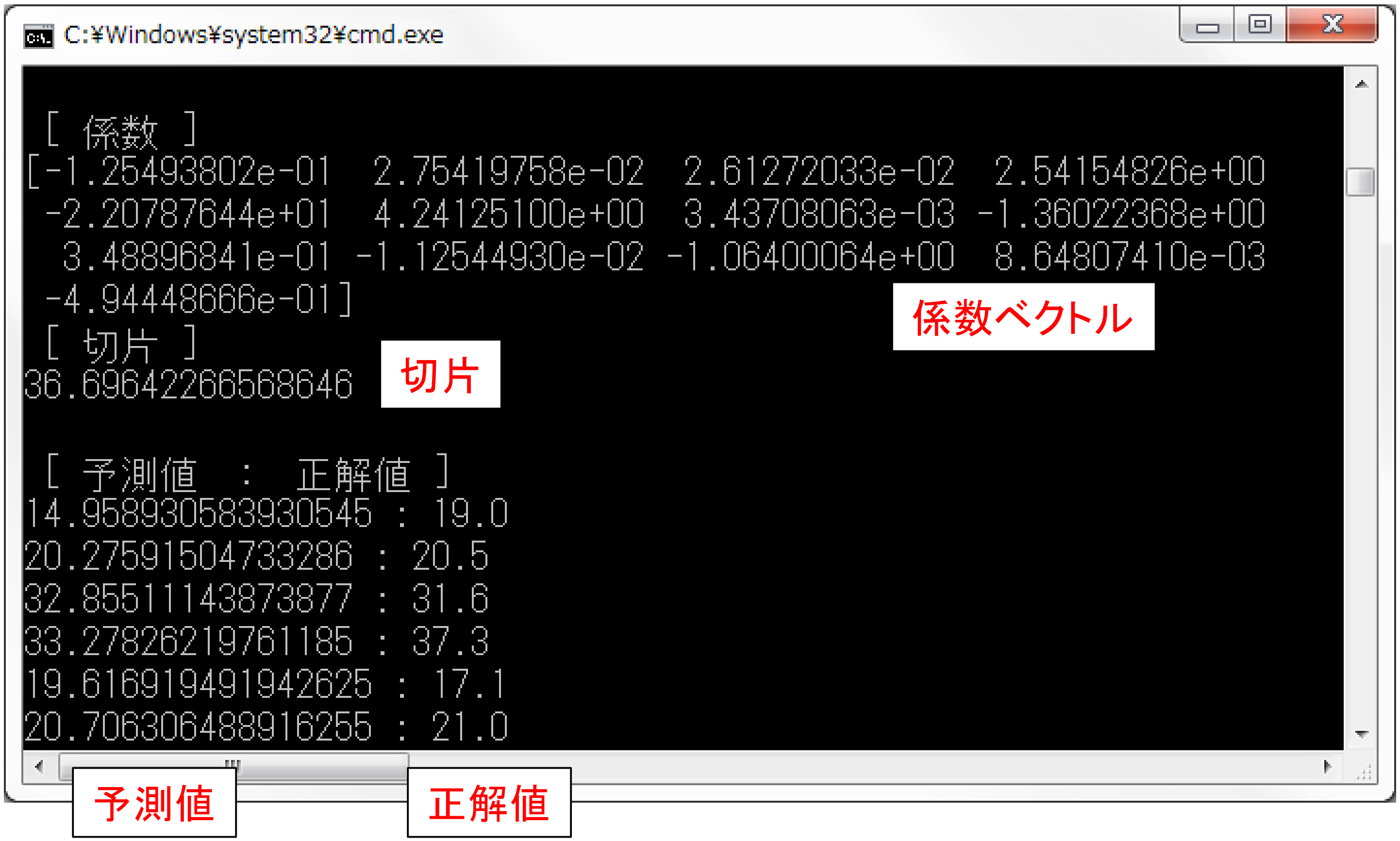
y軸のデータ

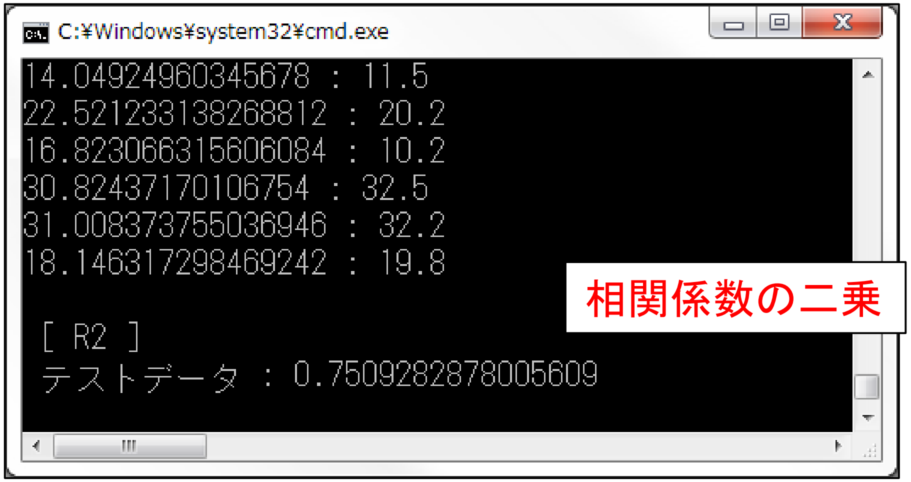
scatter（xの値, yの値）

xlabel（x軸の説明）
ylabel（y軸の説明）

savefig("画像ファイル名")
散布図を画像として保存

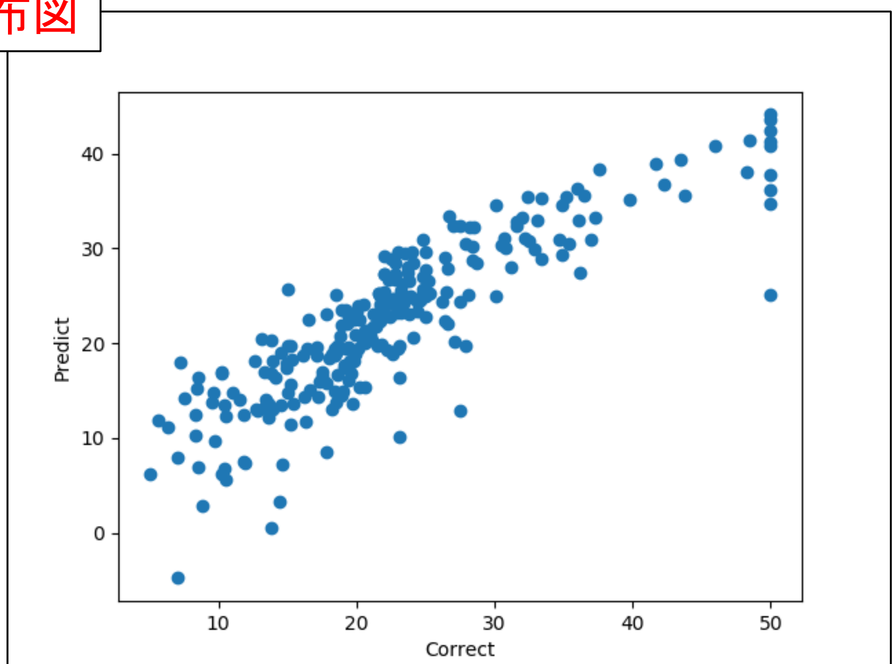
実行結果


C:¥Windows¥system32¥cmd.exe
[係数]
[-1.25493802e-01 2.75419758e-02 2.61272033e-02 2.54154826e+00
-2.20787644e+01 4.24125100e+00 3.43708063e-03 -1.36022368e+00
3.48896841e-01 -1.12544930e-02 -1.06400064e+00 8.64807410e-03
-4.94448666e-01]
[切片]
36.69642266568646
[予測値 : 正解値]
14.958930583930545 : 19.0
20.27591504733286 : 20.5
32.85511143873877 : 31.6
33.27826219761185 : 37.3
19.616919491942625 : 17.1
20.706306488916255 : 21.0
係数ベクトル
切片
予測値
正解値



相関係数の二乗

散布図



予測値

正解値

重回帰分析(Boston_Regression_SKL.py)

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

# データのロード
boston = datasets.load_boston()

# 特徴量(13次元)
feature_name = boston.feature_names
data = boston.data

# 目的変数
label = boston.target
```

重回帰分析のためにimportが必要

散布図の表示のために必要

Bostonデータセットの読み込み

特徴量の名前

特徴量のデータ:(506,13)

目的変数の値(正解ラベル):506

ホールドアウト法

```
# ホールドアウト法（学習データ，テストデータ）
train_data, test_data, train_label, test_label = train_test_split(data, label, test_size=0.5,
random_state=None)

model = linear_model.LinearRegression()

# 学習
model.fit( train_data, train_label )

# 係数と切片
print( '¥n 係数ベクトル : ' , model.coef_ )
print( ' 切片 : ' , model.intercept_)

# 予測
predict = model.predict(test_data)

# 予測値，正解値
print( '¥n [ 予測値 ： 正解値 ]' )
for i in range(len(test_label)):
    print( predict[i] , ':' , test_label[i] )
```

LinearRegression
重回帰

学習

coef_ : 係数ベクトル
intercept_ : 切片

予測

```
# R2を求める
train_score = model.score(train_data, train_label)
test_score = model.score(test_data, test_label)

print( "¥n [ R2 ]" )
print( " 学習データ　: {0:7.5f}".format( train_score ) )
print( " テストデータ: {0:7.5f}".format( test_score ) )

# 散布図の描画
fig = plt.figure()
plt.scatter( test_label , predict )
plt.xlabel("Correct")
plt.ylabel("Predict")
fig.savefig("result.png")
```

score（データ，正解値）
相関係数の二乗（R^2）を求める

x軸のデータ

y軸のデータ

scatter（xの値，yの値）

xlabel（x軸の説明）
ylabel（y軸の説明）

savefig("画像ファイル名")
散布図を画像として保存

実行結果

C:¥Windows¥system32¥cmd.exe

```
係数ベクトル :  [-8.79056606e-02  4.66954361e-02  9.87860757e-02  2.41107042e+00
-2.16952179e+01  3.88122771e+00  6.95693067e-03 -1.40399452e+00
 2.74985367e-01 -1.02636354e-02 -1.13498265e+00  1.06798599e-02
-4.78211833e-01]
切片 :  38.358119427566628

[ 予測値  :  正解値 ]
15.747662773287963 : 11.7
18.92936285079798 : 18.6
13.22868479941237 : 9.5
20.74745072608346 : 20.9
34.99064282823902 : 50.0
17.692941148633157 : 17.3
32.0748042038278 : 31.6
30.34864471830334 : 34.7
12.948867058957156 : 13.9
21.503152624539123 : 22.7
```

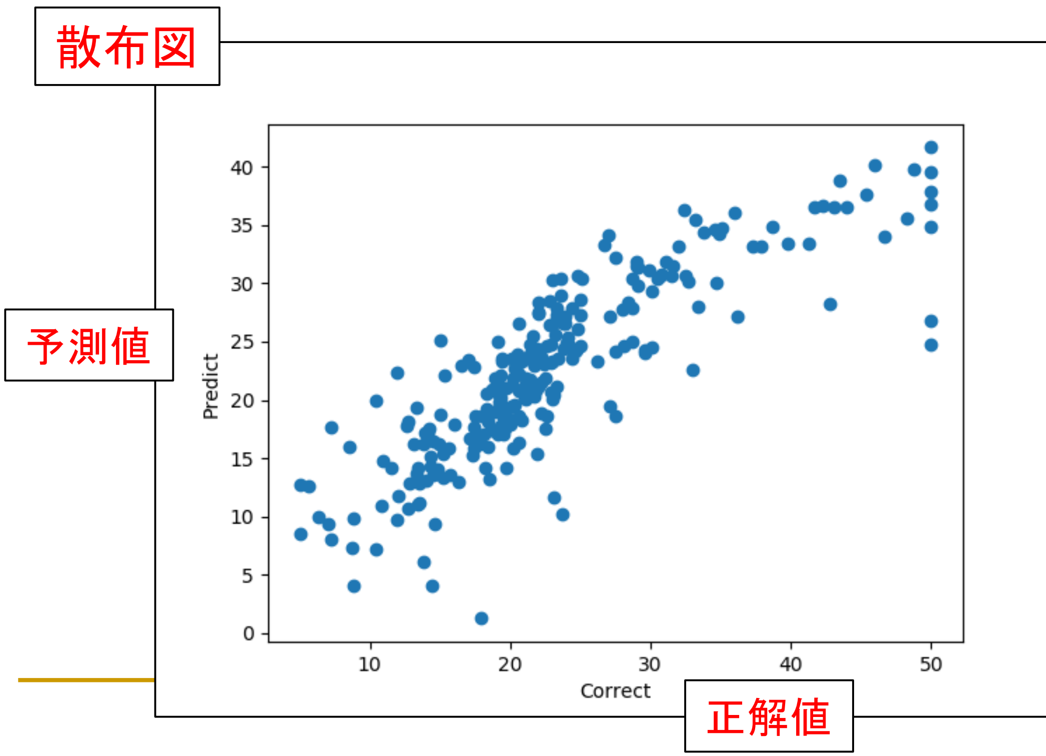
係数ベクトル

切片

予測値

正解値


C:¥Windows¥system32¥cmd.exe
14.188951708553809 : 18.2
17.026731345538444 : 18.1
7.287097515282902 : 8.7
33.438143160789996 : 39.8
[R2]
学習データ　: 0.75805
テストデータ: 0.70727
C:¥home¥shino¥ML-2019¥重回帰分析>
相関係数の二乗


散布図
予測値
正解値
Predict
Correct
0
5
10
15
20
25
30
35
40
10
20
30
40
50

参考文献

- LogisticRegression
  - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

- GridSearchCV
  - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

- LinearRegression
  - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html