# AIと機械学習

# AI（人工知能）

広義: 知的なコンピュータプログラムを作るための科学技術

狭義: 人間と区別が付かない人工的な知能

弱いAI: 特化型（特定のタスクのみを処理）

強いAI: 汎用型（汎用的なタスクに対し自律的に行動）

ルールベースAI

- Lv.1: 予め決められたルールに従うだけ
- Lv.2: 知識データベースを利用

機械学習

- Lv.3: ルールを自ら学習可能
- Lv.4: ディープラーニング

認識AI

生成AI

# AIブーム

第一次（1950~1960）　定義されたルールに基づく推論・探索

第二次（1980~1990）　知識データベースに基づく推論・探索
エキスパートシステム

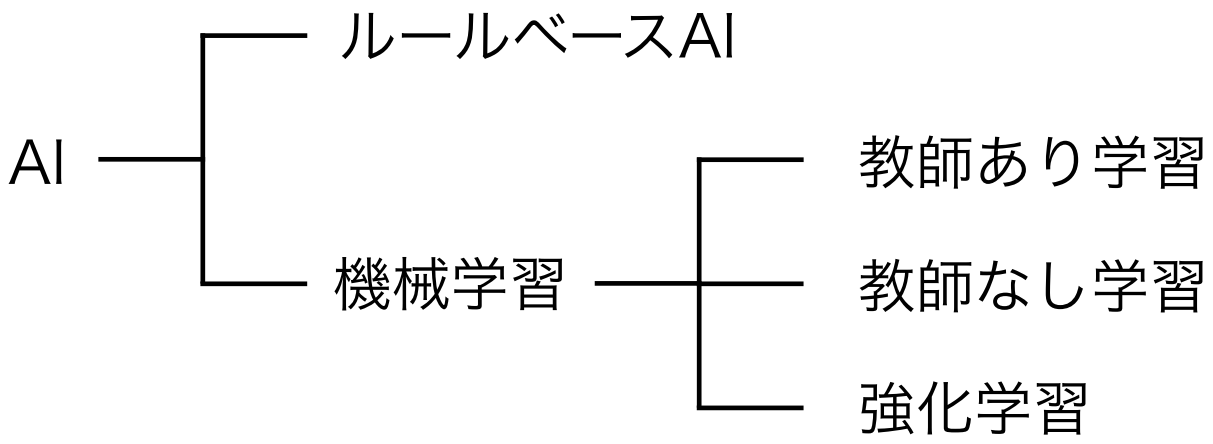
第三次（2010~）　ルールをデータから自動的に学習
機械学習

2013　Ponanza （山本、ロジスティック回帰）
佐藤天彦に勝利

2016　AlphaGo（Google DeepMind、ディープラーニング）
イ・セドルに勝利

# 機械学習

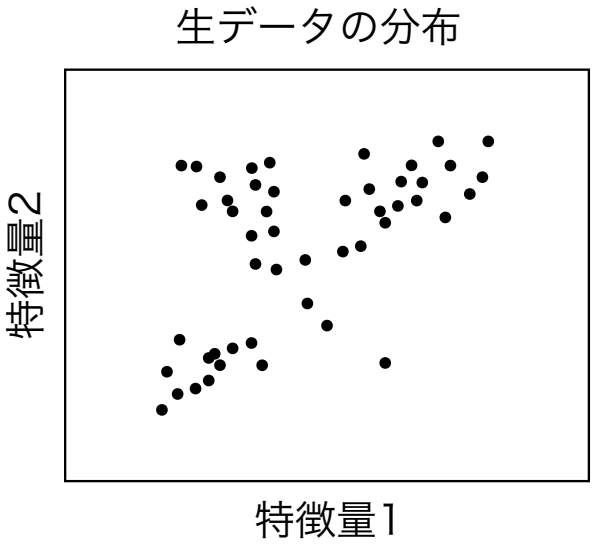
「明示的にプログラムしなくても学習する能力を
コンピュータに与える研究分野」（アーサー・サミュエル）

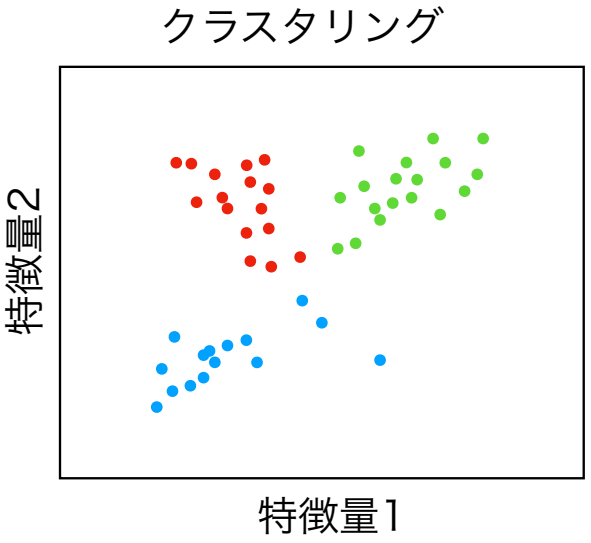


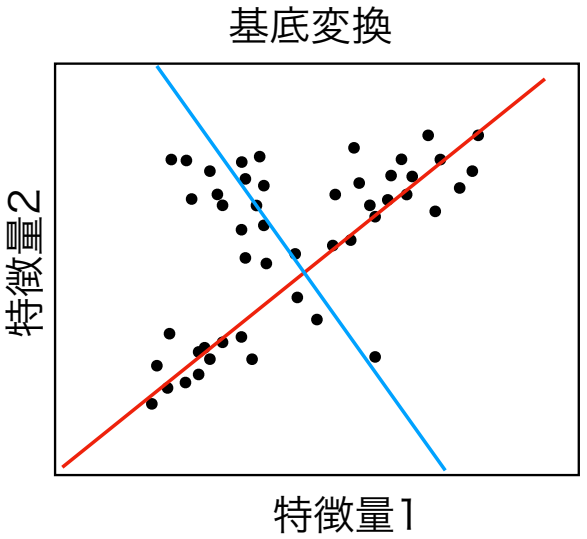
# 教師なし学習

「特徴量」のみを使って、データの分類方法や表現方法を学習

| | |
|---|---|
| クラスタリング | K-means法等 |
| 次元削減・基底変換 | 主成分分析等 |



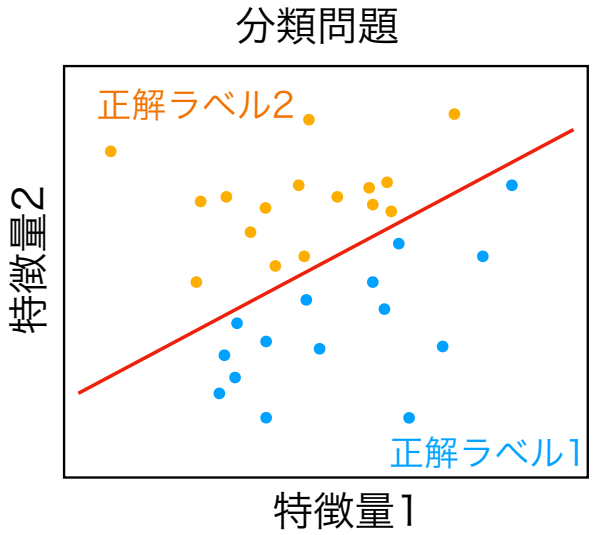
# 教師あり学習

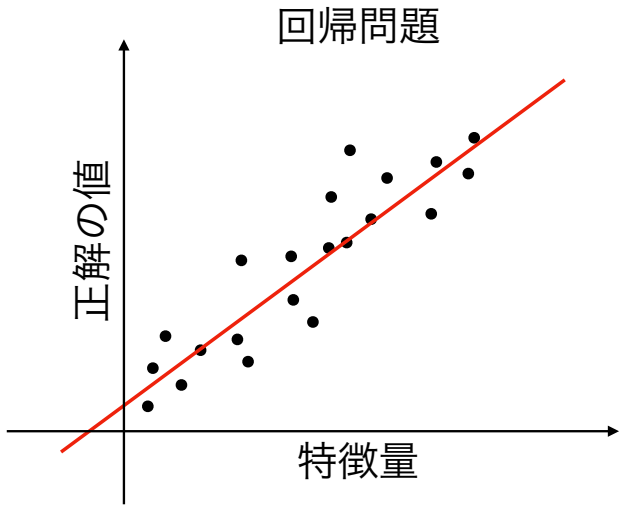
「特徴量と**正解の値**のセット」（**= 教師データ**）を使って予測モデルを学習

分類問題　　ロジスティック回帰等

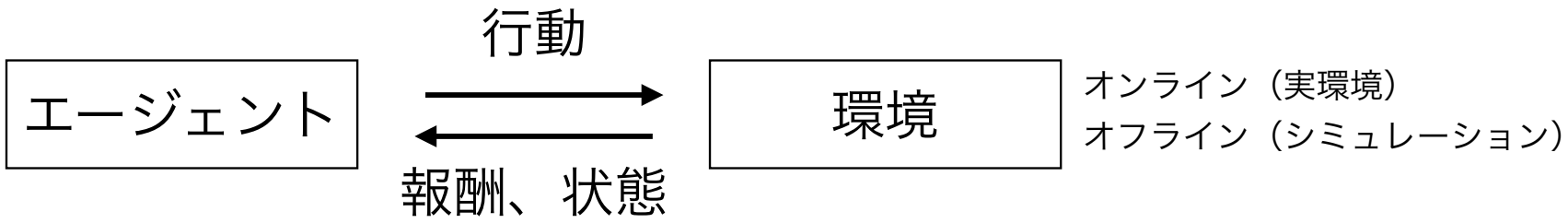
回帰問題　　線形回帰等

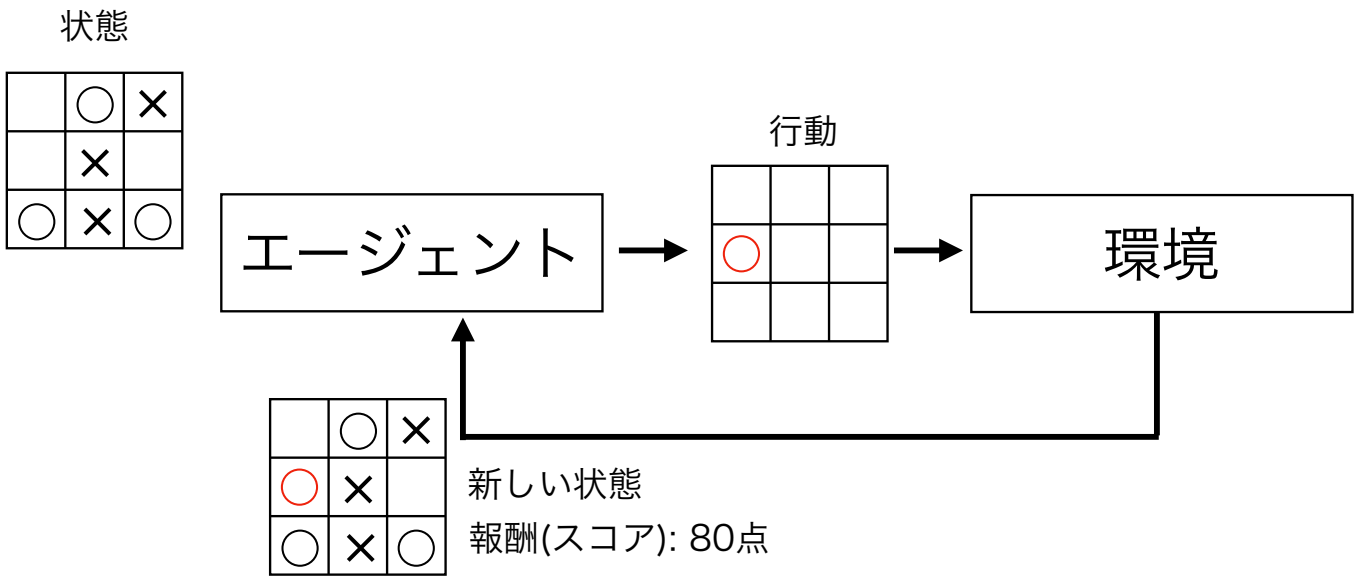
ディープラーニング
サポートベクターマシン
ランダムフォレスト
勾配ブースティング
etc.



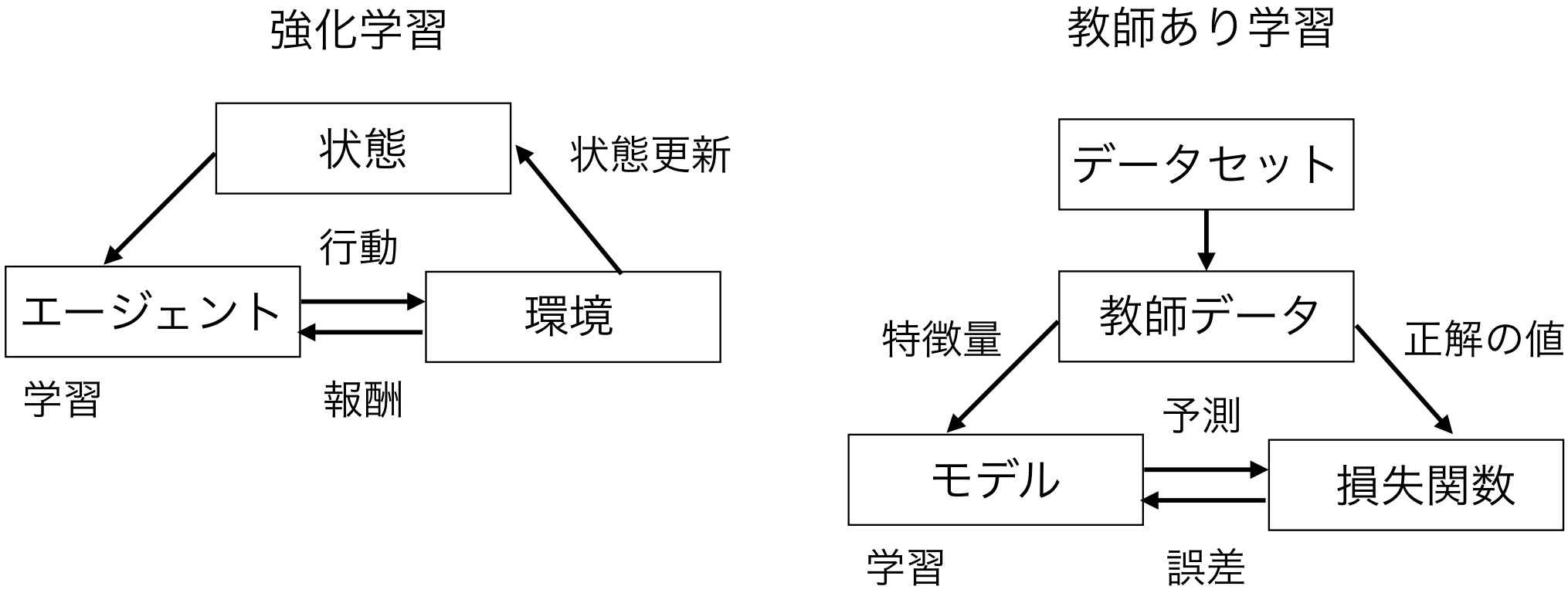



# 強化学習


行動
エージェント
環境
報酬、状態
オンライン（実環境）
オフライン（シミュレーション）


状態
行動
エージェント
環境
新しい状態
報酬(スコア): 80点

# 強化学習と教師あり学習の違い



# 教師あり学習

# 教師あり学習の例

ワインのデータ：
それぞれのワインの等級を3段階(0,1,2)で評価したもの
（各数値の単位は機械学習においてさほど重要ではないので省略。相対的な値の大きさが重要）

| | 等級 | アルコール度数 | リンゴ酸含有量 | マグネシウム含有量 |
|---|---|---|---|---|
| サンプル1 | 1 | 12.08 | 1.13 | 78 |
| サンプル2 | 2 | 13.27 | 4.28 | 120 |
| サンプル3 | 1 | 11.84 | 0.89 | 94 |
| サンプル4 | 0 | 14.10 | 2.16 | 105 |
| サンプル5 | 2 | 12.84 | 2.96 | 101 |
| … | … | … | … | … |

「等級」という量（目的変数）を、アルコール度数などの特徴量を使って予測できるような「関数（モデル）」を構築できないか？

サンプル 2 は等級が高く、アルコール度数も比較的高い。リンゴ酸、マグネシウムの値も大きい。
よって、全ての特徴量の値が大きいほど良いワインと言える？
サンプル 4 の等級は 0 なので、アルコールの値は大きすぎてもいけない？
サンプル 5 も等級 2 だが、マグネシウム含有量はそんなに大きくない。。。
…

# 教師あり学習の例

| | 等級 | アルコール<br>度数 | リンゴ酸<br>含有量 | マグネシウム<br>含有量 |
|---|---|---|---|---|
| サンプル1 | 1 | 12.08 | 1.13 | 78 |
| サンプル2 | 2 | 13.27 | 4.28 | 120 |
| サンプル3 | 1 | 11.84 | 0.89 | 94 |
| サンプル4 | 0 | 14.10 | 2.16 | 105 |
| サンプル5 | 2 | 12.84 | 2.96 | 101 |
| … | … | … | … | … |

等級を予測する「モデル」を作れるかもしれないが、複雑なものになりそう。
とても人間（ルールベースモデル: if ~ then）の手に負えない。

専門家の知識に頼ることなく、データから自動的に「モデル」を生成できないか？
→ 機械学習

# 教師データ

特徴量: （説明変数、記述子、$\boldsymbol{X}$）

サンプルから得られる何らかのデータ（アルコール度数、リンゴ酸含有量、etc.）。
そのサンプルを特徴付け、他のサンプルとの区別を可能とする数値。
**一般には、目的変数以外の特徴量のことを指す。**

目的変数: （従属変数、$\boldsymbol{t}$）

特徴量の中で、予測したい値（最も興味があるが、簡単には得られない値）。
学習する上で**正解となる値**のこと（ワインの等級）。

目的変数が**連続変数** → 回帰問題、マルチターゲット回帰問題（目的変数が複数の場合）
目的変数が**カテゴリ変数** → 分類問題、多クラス分類問題（カテゴリが3つ以上ある場合）

教師データ: （$D = \{\boldsymbol{t}, \boldsymbol{X}\}$）

目的変数と特徴量のセット。

# モデル

「特徴量 $\boldsymbol{X} = \left[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\right]$」という入力データを受け取り、

「目的変数 $\boldsymbol{t}$」の「**予測値 $\boldsymbol{y}$**」を出力する関数（予測器、識別器）。

当然、$\boldsymbol{t}$ と $\boldsymbol{y}$ の値が近いモデルが良いモデル。

モデルは大きく、「形」、「ウェイト（パラメータ）」、そして、「ハイパーパラメータ（後述）」の3つで構成されており、学習過程ではウェイトの値のみが更新される。

良いモデルを（データから自動的に）構築することが機械学習の目的。
（モデルを作らない手法もある。進化論的アルゴリズムなど。）

# モデルの「形」の例

線形回帰

$$y_i = \sum_j x_{ij} w_j$$ 線形モデル

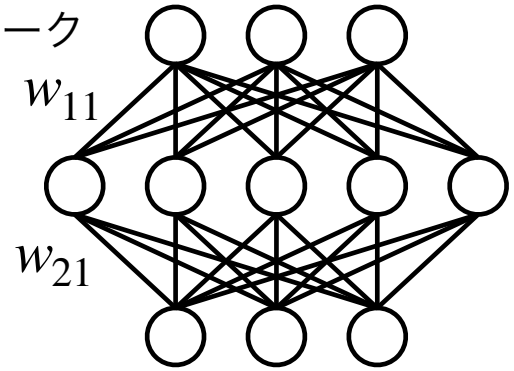
ロジスティック回帰

$$y_{ik} = \frac{e^{z_{ik}}}{\sum_l e^{z_{il}}}$$ Softmax関数

$$z_{ik} = \sum_j x_{ij} w_{jk}$$

ニューラルネットワーク

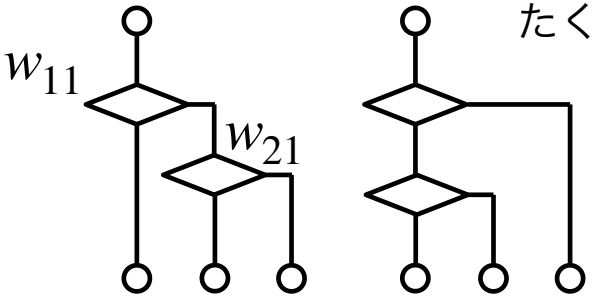
多層のネットワーク




ハイパーパラメータ:

層の数、層の種類、ノード数

ランダムフォレスト




たくさんの（決定）木

ハイパーパラメータ:

木の数、枝の数

# モデルパラメータ

ウェイト: $\boldsymbol{w}$

モデルに含まれるパラメータのうち、学習によって更新されるもの。

ハイパーパラメータ:

モデルに含まれるパラメータのうち、学習で更新されないもの。

- 学習の際に用いられる学習率等
- デープラーニングにおけるノードの数や層の深さ等
- 損失関数の中の罰則項の大きさ等

# 学習

教師データから何らかの方法で最適なウェイトを求めること。

多くの場合、適当な**損失関数（コスト関数）**$L(\boldsymbol{w})$ を定義して、
それが最小になるようなウェイトを求める。

$$\boldsymbol{w}_{\text{best}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} L(\boldsymbol{w})$$

損失関数には、様々なものがあり、目的に応じて使い分ける。
また、学習方法にも多くの種類があり、学習効率が良いものを選ぶ。

# 損失関数（コスト関数）

コスト関数（スコア関数）：
ウェイトを最適化する際の目標となる値を得るための関数（モデルの良し悪しの評価基準）。
損失関数をそのまま使う場合もあれば、問題によって形を少し変えたり、学習を安定化させるために罰則項を導入するなど、何らかの手を加えることもある。

原理的には、良いウェイトのときに小さな値を返し、悪いウェイトの時に大きな値を返す関数であれば何でも良い。つまり、モデルで推定した値が正解に近ければ小さな値、正解から遠ければ大きな値を返すような関数であれば、学習は可能である（上手く収束するかどうかはわからない）。

**損失関数**（Loss関数、誤差関数）：
コスト関数とほぼ同じ意味（狭義には負の対数尤度のことを指すと思われる）。
こちらの方が良く使われる。

# 損失関数の例

$t_i$ ：正解の値、正解ラベル

$y_i$ ：予測値、モデルの出力

$$L = \frac{1}{2}\sum_{i}^{N}(t_i - y_i)^2$$

残差平方和　（回帰問題等で使用される）

$$L = -\sum_{i}^{N}\{t_i \ln y_i + (1 - t_i)\ln(1 - y_i)\}$$

バイナリ交差エントロピー誤差
（2値分類問題で使用される）

$$L = -\sum_{i,k}^{N,K} t_{ik} \ln y_{ik}$$

交差エントロピー誤差
（マルチクラス分類問題で使用される）

# 学習方法

線形回帰の場合は最適なウェイトを解析的に求めることができるが、ロジスティック回帰をはじめ多くの非線形モデルではそれは不可能。よって、何らかの方法で数値的に最適なウェイトを求める必要がある。最も基本的な学習方法に勾配（降下）法がある。

**勾配降下法**:（GD: Gradient Descent methodm、または、最急降下法）

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

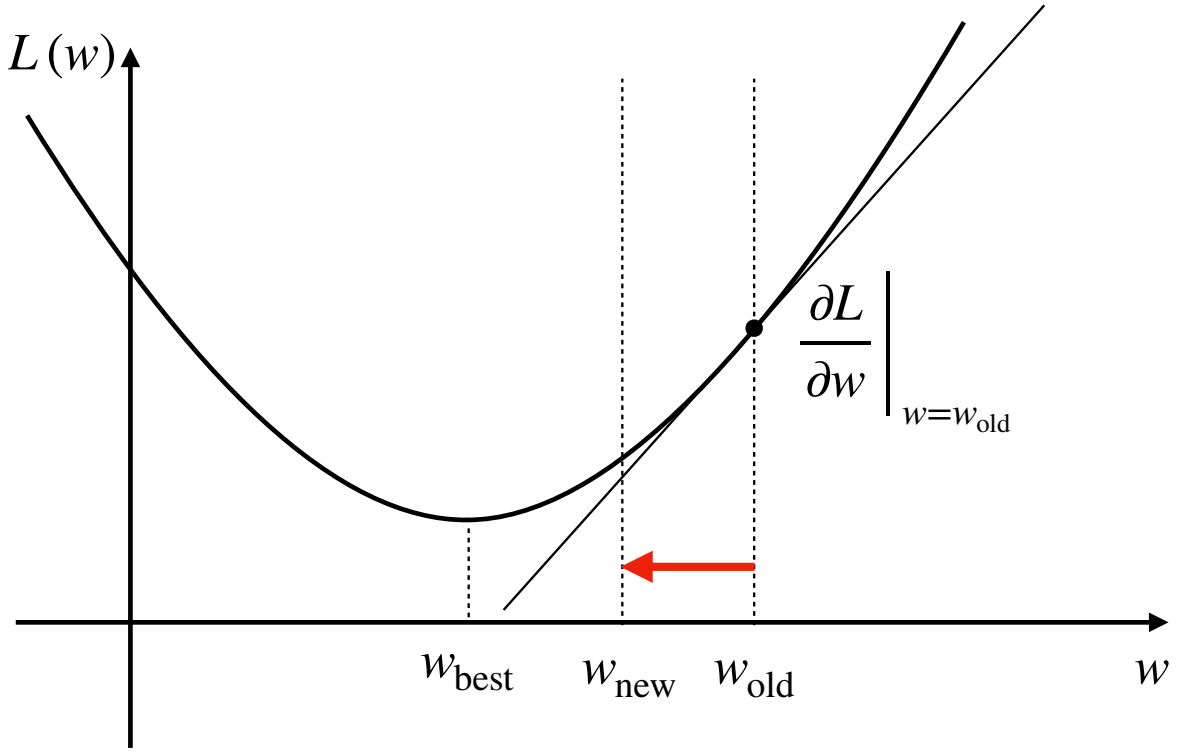
$\eta$ : 学習率、更新のステップサイズ

$$\left( w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w} \right)$$

これを指定回数、あるいは、ウェイトが変化しなくなる（収束する）まで繰り返す。
ウェイトは乱数で初期化。

$$\frac{\partial L}{\partial w} \sim 0$$

# 勾配降下法によるウェイトの最適化の様子

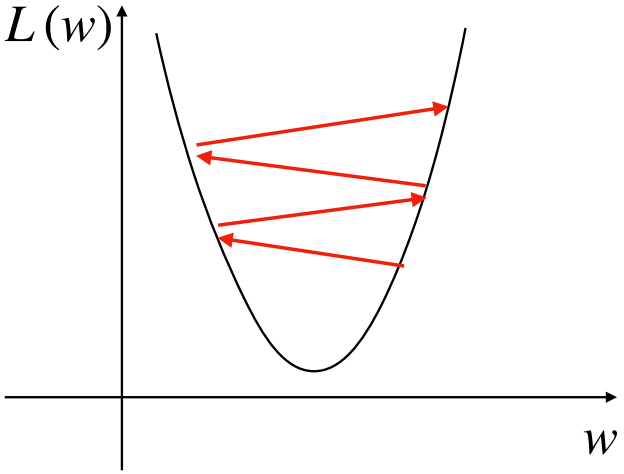



$$w_{\text{new}} = w_{\text{old}} - \eta \left.\frac{\partial L}{\partial w}\right|_{w=w_{\text{old}}}$$

# 勾配降下法の問題点1

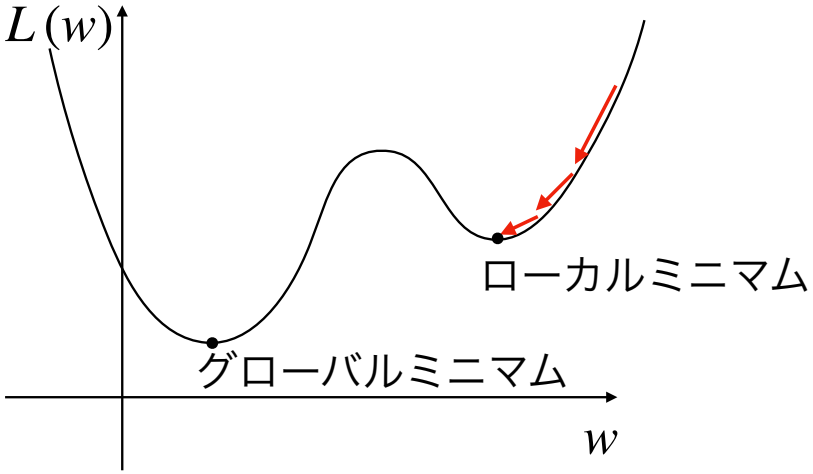
問題点:

適切な学習率を手で与えなければならない。




学習率が大きい場合

→ ウェイトの振動、発散




学習率が小さい場合

→ ローカルミニマムに引っかかる

解決策:

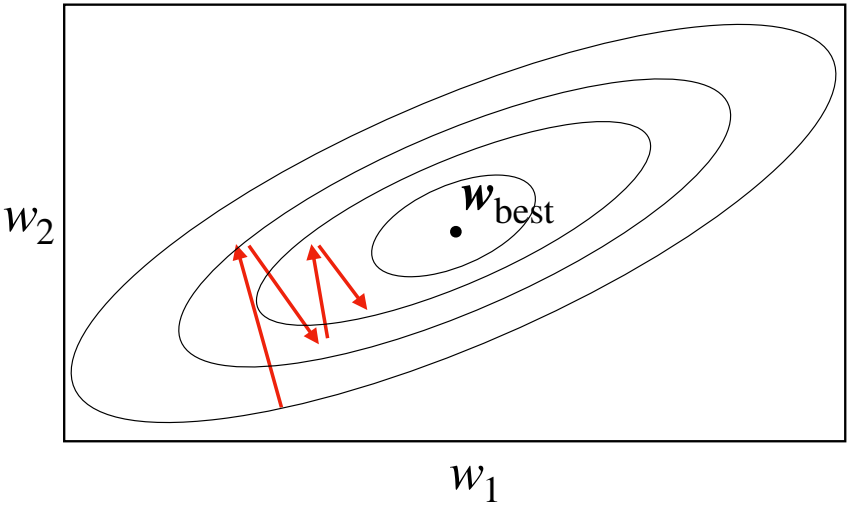
Learning rate decay: 学習の進み具合に応じて学習率を小さくしていく。

AdaGrad: 各ウェイトごとに学習率を設定し、よく変化するもの程、学習率を小さくする。

# 勾配降下法の問題点2

問題点:

無駄な学習ステップが生じ、学習に時間がかかる場合がある。




解決策: Momentum（勢い）

過去のウェイトの変化を覚えておき、それを慣性として利用し、
同じ方向を行ったり来たりしないようにする。

$$w \leftarrow w + v$$

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial w}$$

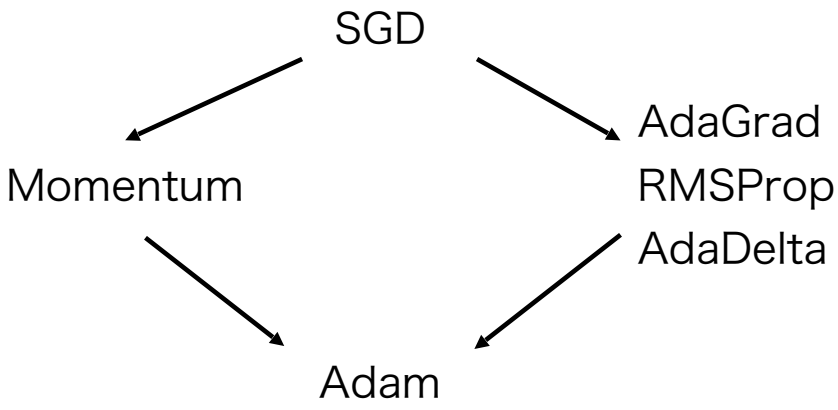
$\alpha$ : ダンピング(~ 0.9)

慣性

現在地での勾配

現在地での更新方向が前回のもの（慣性）と逆向きの場合は打ち消し合い、
同じ方向であれば足し合わされる。

# 様々な学習方法（Optimizer)




**SGD**（学習率固定）
Momentum（過去のウェイト更新情報を慣性として取り入れる）
AdaGrad（各ウェイトごとの学習率減衰）
RMSprop（AdaGradの拡張。指数移動平均。
学習率の過剰なdecayを緩和）
AdaDelta（RMSpropの拡張、単位を揃える）
**Adam**（RMSprop+Momentum）
Eve（鞍点からの脱出）

SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=**False**)
Adagrad(lr=0.01, epsilon=1e-08, decay=0.0)
RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
Adadelta(lr=1.0, rho=0.95, epsilon=1e-08, decay=0.0)
Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004)
Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

**lr**: float >= 0. Learning rate.
**momentum**: float >= 0. Parameter updates momentum.
**decay**: float >= 0. Learning rate decay over each update.
**nesterov**: boolean. Whether to apply Nesterov momentum.(NAG?)

# バッチ学習と逐次学習

・ （バッチ）勾配降下法:

全ての教師データ（バッチ） $D = \{t, X\}$ に対する勾配を求めてウェイトを更新する方法。局所解にハマりやすい。

一回のウェイト更新での計算量や必要なメモリが多くなる。

・ 逐次勾配降下法（確率的勾配降下法、SGD）:

一つの教師データ $t_i, x_i \in D$ から求めた勾配でウェイトを更新する方法。

局所解にハマりにくい。

メモリ消費が少ない。

オンライン学習（その場学習）が可能。

一回の計算（ウェイト更新）は高速。

全ての教師データをスイープし、さらにそれを収束するまで繰り返すのでやはり時間がかかる。

よって、通常はミニバッチと呼ばれる複数の教師データセットに対して更新を行う。

# ミニバッチ学習

教師データをある程度の大きさに分けて学習する方法。
最も効率の良い学習。

というのも、
python では逐次学習のように for ループを回す処理はとても遅いが、
行列演算は numpy があるのでとても速い。
（内部で高速な数値計算ライブラリを使っている）。

なので、教師データをある程度の大きさの行列にまとめて numpy で一気に演算する、ということを繰り返す方が、トータルでは高速になる。
また、局所解にハマりにくく収束が滑らかになり少ないループ数で収束させることができる。

# 勾配の計算方法

全ての Optimizer は勾配（損失関数の各ウェイトに関する微分）の情報を必要とするが、その計算方法にはいくつかの種類がある。

・手動微分
予め人の手で解析的に勾配の形を求めておき、それをコードに直書きする。
線形回帰やロジスティック回帰等の単純なモデルの場合は最も効率がよい。

・自動微分（手動微分を発展させたもの）
定義された関数を単純な演算に分解し、それぞれの演算において勾配も同時に計算させるようにしておき、連鎖律で元の関数の勾配を得る。

・数値微分（中心差分）　$\dfrac{\partial L}{\partial w} = \dfrac{L(w+h) - L(w-h)}{2h}$　　$h \sim 10^{-4}$

・記号微分（数式微分）
Mathematica 等の数式処理システム

# 微分の連鎖律

$$z = z(x),\quad y = y(z) \longrightarrow \frac{dy}{dx} = \frac{dz}{dx}\frac{dy}{dz}$$

合成微分

$$z_1 = z_1(x),\quad z_2 = z_2(x),\quad y = y(z_1, z_2) \longrightarrow \frac{dy}{dx} = \frac{dz_1}{dx}\frac{\partial y}{\partial z_1} + \frac{dz_2}{dx}\frac{\partial y}{\partial z_2}$$

$$z_1 = z_1(x_1, x_2),\quad z_2 = z_2(x_1, x_2),\quad y = y(z_1, z_2) \longrightarrow$$

$$\frac{\partial y}{\partial x_1} = \frac{\partial z_1}{\partial x_1}\frac{\partial y}{\partial z_1} + \frac{\partial z_2}{\partial x_1}\frac{\partial y}{\partial z_2}$$

$$\frac{\partial y}{\partial x_2} = \frac{\partial z_1}{\partial x_2}\frac{\partial y}{\partial z_1} + \frac{\partial z_2}{\partial x_2}\frac{\partial y}{\partial z_2}$$

# 連鎖律の一般形

$$
\begin{aligned}
z_1 &= z_1(x_1, x_2, \ldots, x_N) \\
z_2 &= z_2(x_1, x_2, \ldots, x_N) \\
&\vdots \\
z_Q &= z_Q(x_1, x_2, \ldots, x_N)
\end{aligned}
\qquad
y = y(z_1, z_2, \ldots, z_Q) \longrightarrow
$$

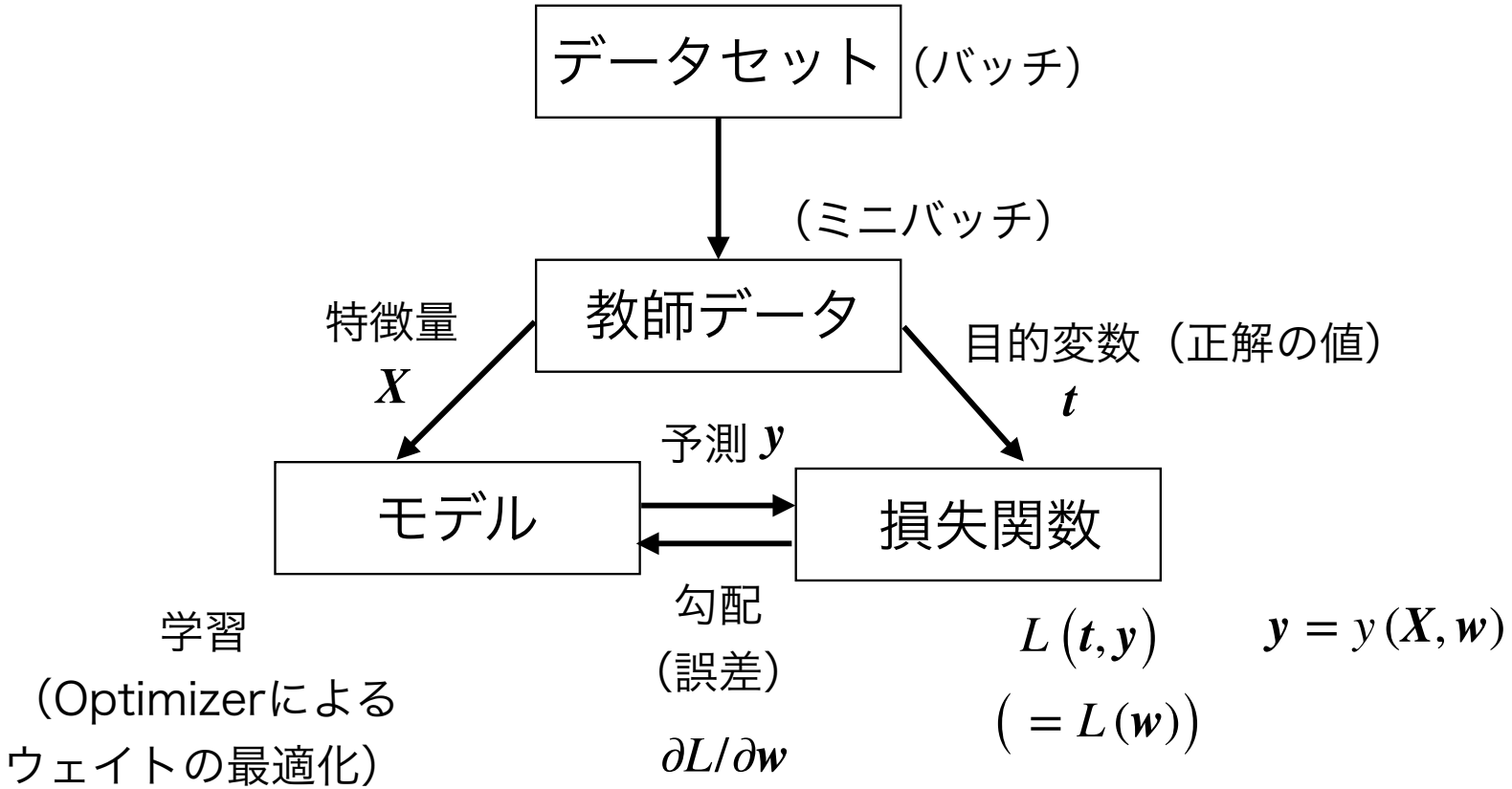
2重の場合

$$
\frac{\partial y}{\partial x_i} = \sum_{q}^{Q} \frac{\partial z_q}{\partial x_i} \frac{\partial y}{\partial z_q}
$$

*N* 重の場合

$$
\frac{\partial z_{q_N}^{(N)}}{\partial z_{q_0}^{(0)}} = \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \sum_{q_2}^{Q_2} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \sum_{q_{N-2}}^{Q_{N-2}} \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} \sum_{q_{N-1}}^{Q_{N-1}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}
$$

# 教師あり学習の流れ


データセット
(バッチ)
(ミニバッチ)
教師データ
特徴量
$X$
目的変数（正解の値）
$t$
予測 $y$
モデル
損失関数
勾配
(誤差)
$\partial L/\partial w$
学習
(Optimizerによる
ウェイトの最適化)
$L(t,y)$
$(=L(w))$
$y=y(X,w)$

# 色々な教師あり学習手法

・線形回帰、ロジスティック回帰

・ディープラーニング（次週詳しく解説）

・サポートベクターマシン

・アンサンブル学習（決定木ベース）

バギング形式(並列)　ランダムフォレスト

ブースティング形式(直列)　勾配ブースティング

- GBDT
- XGBoost
- AdaBoost
- LightGBM（マイクロソフト2016）

# 線形回帰とロジスティック回帰

# 線形回帰

モデル: 線形回帰モデル（最小二乗法、OLS）

$$y_i = \sum_{j=0}^{N} x_{ij} w_j \qquad w_j：ウェイト（回帰係数）$$

$x_{i0} \equiv 1$ 定数項

損失関数が最小となるウェイト $w_j$ を求める。

損失関数: 残差平方和

$$L = \frac{1}{2} \sum_{i=1}^{M} (t_i - y_i)^2$$

行列表現

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$$

$$L = \frac{1}{2} \|\boldsymbol{t} - \boldsymbol{y}\|^2$$

ワインの等級には順序があるので回帰問題として扱う事も出来る
（0: イヌ、1:ネコ、2: ウサギ、のような場合は順序に意味はない）

| | 等級 | アルコール度数 | リンゴ酸含有量 | マグネシウム含有量 |
|---|---|---|---|---|
| サンプル1 | 1 | 12.08 | 1.13 | 78 |
| サンプル2 | 2 | 13.27 | 4.28 | 120 |
| サンプル3 | 1 | 11.84 | 0.89 | 94 |
| サンプル4 | 0 | 14.10 | 2.16 | 105 |
| サンプル5 | 2 | 12.84 | 2.96 | 101 |
| … | … | … | … | … |

$$\boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_M \end{bmatrix}$$

目的変数

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1N} \\ 1 & x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{M1} & x_{M2} & \cdots & x_{MN} \end{bmatrix}$$

特徴量（計画行列）

定数項

特徴量のベクトル表示

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1j} & \cdots & x_{1N} \\ 1 & x_{21} & x_{22} & \cdots & x_{2j} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & x_{i2} & \cdots & x_{ij} & \cdots & x_{iN} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{M1} & x_{M2} & \cdots & x_{Mj} & \cdots & x_{MN} \end{bmatrix}$$

行列表示や成分表示の場合は問題ないが、特徴量をベクトル表示している場合は注意が必要。行ベクトルと列ベクトルの2種類の流儀があるので、どちらの定義が使われているか常に確認する必要がある。

$$\boldsymbol{x}_i \equiv \left[1, x_{i1}, x_{i2}, x_{i3}, \cdots, x_{iN}\right]^T$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \boldsymbol{x}_3^T \\ \vdots \\ \boldsymbol{x}_M^T \end{bmatrix} = \left[\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \cdots, \boldsymbol{x}_M\right]^T$$

$$\boldsymbol{x}_j \equiv \begin{bmatrix} x_{1j} \\ x_{2j} \\ x_{3j} \\ \vdots \\ x_{Mj} \end{bmatrix}, \quad \boldsymbol{1} \equiv \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$\boldsymbol{X} = \left[\boldsymbol{1}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\right]$$

# 線形回帰

学習方法は何でも良いが、ウェイトを最適化するためには勾配の情報、
つまり、各ウェイトに対する損失関数の偏微分を計算する必要がある。

$$w_j \leftarrow w_j - \eta \frac{\partial L}{\partial w_j}$$

$$L = \frac{1}{2}\sum_{i=1}^{M}\left(t_i - y_i\right)^2 = \frac{1}{2}\sum_{i=1}^{M}\left(t_i - \sum_{j=0}^{N} x_{ij} w_j\right)^2$$

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{M} \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial w_j} = -\sum_{i=1}^{M}(t_i - y_i)x_{ij}$$

$$\frac{\partial L}{\partial y_i} = -\left(t_i - y_i\right)$$

誤差（残差）
（正解値と推定値との差）

$$\frac{\partial y_i}{\partial w_j} = \frac{\partial}{\partial w_j}\sum_{j'=0}^{N} x_{ij'} w_{j'} = x_{ij}$$

線形回帰の場合は最適なウェイトを解析的に求めることが可能！

# 線形回帰（厳密解）

$$\frac{\partial L}{\partial w_j} = -\sum_{i=1}^{M}(t_i - y_i)x_{ij} = 0$$

$$\rightarrow \sum_{i=1}^{M} t_i x_{ij} = \sum_{i=1}^{M} y_i x_{ij}$$

$$\rightarrow \sum_{i=1}^{M} t_i x_{ij} = \sum_{i=1}^{M}\sum_{j'=0}^{N} x_{ij'} w_{j'} x_{ij}$$

$$\rightarrow \sum_{i=1}^{M} \left[\boldsymbol{X}^T\right]_{ji} t_i = \sum_{j'=0}^{N}\sum_{i=1}^{M} \left[\boldsymbol{X}^T\right]_{ji} \boldsymbol{X}_{ij'} w_{j'}$$

$$\rightarrow \boldsymbol{X}^T\boldsymbol{t} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}$$

$$\rightarrow \boxed{\boldsymbol{w}_{\mathrm{OLS}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{t}}$$

解析解（厳密解）

但し、$\boldsymbol{X}^T\boldsymbol{X}$ は正則行列

# 線形回帰（勾配降下法によるウェイトの更新と収束解）

$$\frac{\partial L}{\partial w_j} = -\sum_{i=1}^{M} (t_i - y_i) x_{ij}$$

$$w_j \leftarrow w_j - \eta \frac{\partial L}{\partial w_j}$$

$$= w_j + \eta \sum_{i=1}^{M} \left(t_i - y_i\right) x_{ij}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{X}^T \left(\boldsymbol{t} - \boldsymbol{y}\right)$$

更新式

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}} + \eta \boldsymbol{X}^T \left(\boldsymbol{t} - \boldsymbol{y}\right)$$

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$$

$$\rightarrow \quad \boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}} + \eta \boldsymbol{X}^T \boldsymbol{t} - \eta \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{w}_{\text{old}}$$

$$\rightarrow \quad \boldsymbol{w}_{\text{new}} = \left(\boldsymbol{I} - \eta \boldsymbol{X}^T \boldsymbol{X}\right) \boldsymbol{w}_{\text{old}} + \eta \boldsymbol{X}^T \boldsymbol{t}$$

$$\boldsymbol{w}_{\text{conv}} \equiv \boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}}$$

$$\rightarrow \quad \boldsymbol{w}_{\text{conv}} = \left(\boldsymbol{I} - \eta \boldsymbol{X}^T \boldsymbol{X}\right) \boldsymbol{w}_{\text{conv}} + \eta \boldsymbol{X}^T \boldsymbol{t}$$

$$\rightarrow \quad \boldsymbol{w}_{\text{conv}} = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{t}$$

収束解（厳密解と一致）

# ロジスティック回帰（2値分類問題）

（回帰問題ではなく分類問題のための手法！）

0: 女性、1:男性

| | 性 | 身長 | 体重 | 体脂肪率 | … |
|---|---|---|---|---|---|
| サンプル | 1 | 182 | 78 | 12 | … |
| サンプル | 0 | 160 | 48 | 20 | … |
| サンプル | 1 | 176 | 65 | 15 | … |
| サンプル | 0 | 140 | 45 | 21 | … |
| サンプル | 0 | 163 | 50 | 23 | … |
| … | … | … | … | … | … |

モデル: シグモイド関数

$$y_i = \frac{e^{z_i}}{e^{z_i}+1} \qquad \left( = \frac{1}{1+e^{-z_i}} \right)$$

$$z_i = \sum_{j=0}^{N} x_{ij} w_j$$ ロジット

$$\boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_M \end{bmatrix}$$ 目的変数

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1N} \\ 1 & x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{M1} & x_{M2} & \cdots & x_{MN} \end{bmatrix}$$ 定数項 特徴量

損失関数: バイナリ交差エントロピー誤差

$$L = -\sum_{i=1}^{M} \left[ t_i \ln y_i + (1-t_i) \ln(1-y_i) \right]$$

$t_i = \{0,1\} = 0 \text{ or } 1$ : 正解ラベル

$y_i = [0,1] = 0 \sim 1$ : 予測値

# ロジスティック回帰

$$L = -\sum_{i=1}^{M} \left[ t_i \ln y_i + (1 - t_i) \ln(1 - y_i) \right]$$

$$y_i = \frac{e^{z_i}}{e^{z_i} + 1}$$

$$z_i = \sum_{j=0}^{N} x_{ij} w_j$$

$$\frac{\partial L}{\partial w_j} = \sum_{i}^{M} \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial w_j} = \sum_{i}^{M} \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial z_i} \frac{\partial z_i}{\partial w_j}$$

$$= -\sum_{i}^{M} \frac{t_i - y_i}{y_i(1 - y_i)} y_i(1 - y_i) x_{ij}$$

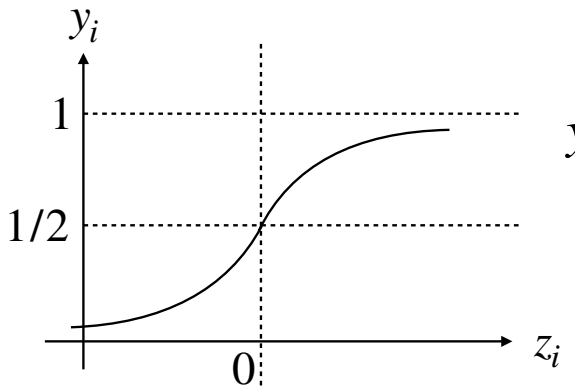
$$= \boxed{-\sum_{i}^{M} (t_i - y_i) x_{ij}}$$

線形回帰の場合と全く同じ形（になるような損失関数を選んだ）

$$\boxed{\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{X}^T (\boldsymbol{t} - \boldsymbol{y})}$$

更新式も当然同じ

シグモイド関数の概形



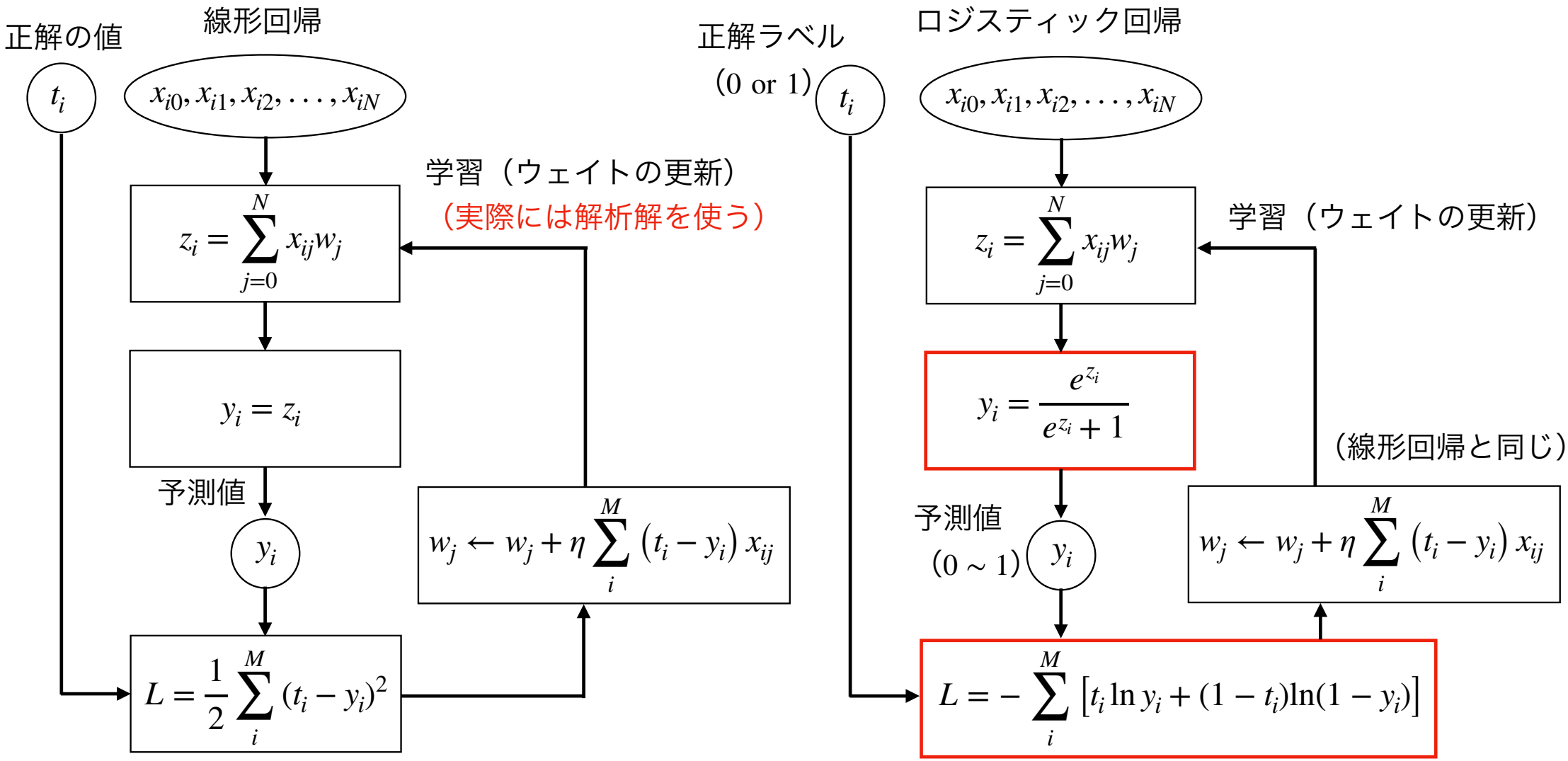

$y_i = 0 \sim 1$

$$\frac{\partial L}{\partial y_i} = -\frac{t_i}{y_i} + \frac{(1 - t_i)}{(1 - y_i)} = -\frac{t_i - y_i}{y_i(1 - y_i)}$$

$$\frac{\partial y_i}{\partial z_i} = \frac{e^{z_i}}{e^{z_i} + 1} - \frac{e^{2z_i}}{(e^{z_i} + 1)^2} = y_i(1 - y_i)$$

$$\frac{\partial z_i}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_{j'} x_{ij'} w_{j'} = x_{ij}$$

# 線形回帰とロジスティック回帰のまとめ



# マルチターゲット線形回帰

モデル: マルチターゲット線形回帰モデル

$$y_{ik} = \sum_{j=0}^{N} x_{ij} w_{jk} \qquad \boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W}$$

$x_{i0} \equiv 1$　定数項

損失関数: 残差平方和

$$L = \frac{1}{2} \sum_{i=1}^{M} \sum_{k=0}^{C-1} (t_{ik} - y_{ik})^2$$

$C$: 目的変数の種類の数

目的変数

$$\boldsymbol{T} = \begin{bmatrix} t_{10} & t_{11} & \cdots & t_{1,C-1} \\ t_{20} & t_{21} & \cdots & t_{2,C-1} \\ \vdots & \vdots & \ddots & \vdots \\ t_{M0} & t_{M1} & \cdots & t_{M,C-1} \end{bmatrix}$$

特徴量（計画行列）

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1N} \\ 1 & x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{M1} & x_{M2} & \cdots & x_{MN} \end{bmatrix}$$

定数項

# マルチターゲット線形回帰の更新式

$$L = \frac{1}{2} \sum_{i=1}^{M} \sum_{k=0}^{C-1} (t_{ik} - y_{ik})^2$$

$$\frac{\partial L}{\partial y_{ik'}} = -\left(t_{ik'} - y_{ik'}\right)$$

$$\frac{\partial L}{\partial w_{jk}} = \sum_{i,k'}^{M,C} \frac{\partial L}{\partial y_{ik'}} \frac{\partial y_{ik'}}{\partial w_{jk}}$$

$$\frac{\partial y_{ik'}}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_{j'}^{N} x_{ij'} w_{j'k'} = x_{ij} \delta_{kk'}$$

$$= -\sum_{i,k'}^{M,C} \left(t_{ik'} - y_{ik'}\right) x_{ij} \delta_{kk'}$$

$$= -\sum_{i}^{M} (t_{ik} - y_{ik}) x_{ij}$$

$$\nabla L \equiv \frac{\partial L}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial L}{\partial w_{00}} & \frac{\partial L}{\partial w_{01}} & \cdots & \frac{\partial L}{\partial w_{0,C-1}} \\ \frac{\partial L}{\partial w_{10}} & \frac{\partial L}{\partial w_{11}} & \cdots & \frac{\partial L}{\partial w_{1,C-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{N0}} & \frac{\partial L}{\partial w_{N1}} & \cdots & \frac{\partial L}{\partial w_{N,C-1}} \end{bmatrix} = -\boldsymbol{X}^T (\boldsymbol{T} - \boldsymbol{Y})$$

更新式

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \eta \boldsymbol{X}^T (\boldsymbol{T} - \boldsymbol{Y})$$

厳密解

$$\boldsymbol{W}_{\mathrm{OLS}} = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{T}$$

# 多クラスロジスティック回帰

今回はワインの等級データを分類問題として扱う
0, 1, 2 の3値分類問題

| | 等級 | アルコール度数 | リンゴ酸含有量 | マグネシウム含有量 |
|---|---|---|---|---|
| サンプル1 | 1 | 12.08 | 1.13 | 78 |
| サンプル2 | 2 | 13.27 | 4.28 | 120 |
| サンプル3 | 1 | 11.84 | 0.89 | 94 |
| サンプル4 | 0 | 14.10 | 2.16 | 105 |
| サンプル5 | 2 | 12.84 | 2.96 | 101 |
| … | … | … | … | … |

モデル: ソフトマックス関数

$$y_{ik} = \frac{e^{z_{ik}}}{\sum_l e^{z_{il}}}$$

$$z_{ik} = \sum_{j=0}^{N} x_{ij} w_{jk} \qquad \boldsymbol{Z} = \boldsymbol{XW}$$

損失関数: 交差エントロピー誤差

$$L = -\sum_{i=1}^{M} \sum_{k=0}^{C} t_{ik} \ln y_{ik}$$

$C$: クラスの数

$$\boldsymbol{T} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 1 \\ 0 \\ 2 \\ \end{matrix}$$

目的変数(ワンホット・エンコーディング)

$$\boldsymbol{Y} = \begin{bmatrix} 0.1 & 0.8 & 0.1 \\ 0.2 & 0.1 & 0.7 \\ 0.1 & 0.9 & 0.0 \\ 0.6 & 0.3 & 0.1 \\ 0.0 & 0.2 & 0.8 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

推定値の例

$t_{ik} = \{0,1\}$ : 正解ラベル

$y_{ik} = [0,1]$ : 推定値

# 多クラスロジスティック回帰

$$L = -\sum_{i,k}^{M,C} t_{ik} \ln y_{ik}$$

$$y_{ik} = \frac{e^{z_{ik}}}{\sum_l e^{z_{il}}}$$

$$z_{ik} = \sum_{j=0}^{N} x_{ij} w_{jk}$$

$$\frac{\partial L}{\partial w_{jk}} = \sum_{i,k'}^{M,C} \frac{\partial L}{\partial y_{ik'}} \frac{\partial y_{ik'}}{\partial w_{jk}} = \sum_{i,k'}^{M,C} \frac{\partial L}{\partial y_{ik'}} \sum_{k''}^{C} \frac{\partial y_{ik'}}{\partial z_{ik''}} \frac{\partial z_{ik''}}{\partial w_{jk}}$$

$$= -\sum_{i,k'}^{M,C} \sum_{k''}^{C} \frac{t_{ik'}}{y_{ik'}} y_{ik'} (\delta_{k'k''} - y_{ik''}) \delta_{kk''} x_{ij}$$

$$= -\sum_{i,k'}^{M,C} t_{ik'} (\delta_{k'k} - y_{ik}) x_{ij}$$

$$= -\sum_{i}^{M} (t_{ik} - y_{ik}) x_{ij} \qquad \left( \because \sum_{k}^{C} t_{ik} = 1 \right)$$

$$\frac{\partial L}{\partial y_{ik'}} = -\frac{t_{ik'}}{y_{ik'}}$$

$$\frac{\partial y_{ik'}}{\partial z_{ik''}} = \frac{e^{z_{ik'}}}{\sum_l e^{z_{il}}} \delta_{k'k''} - \frac{e^{z_{ik'}} e^{z_{ik''}}}{\left( \sum_l e^{z_{il}} \right)^2} = y_{ik'} (\delta_{k'k''} - y_{ik''})$$

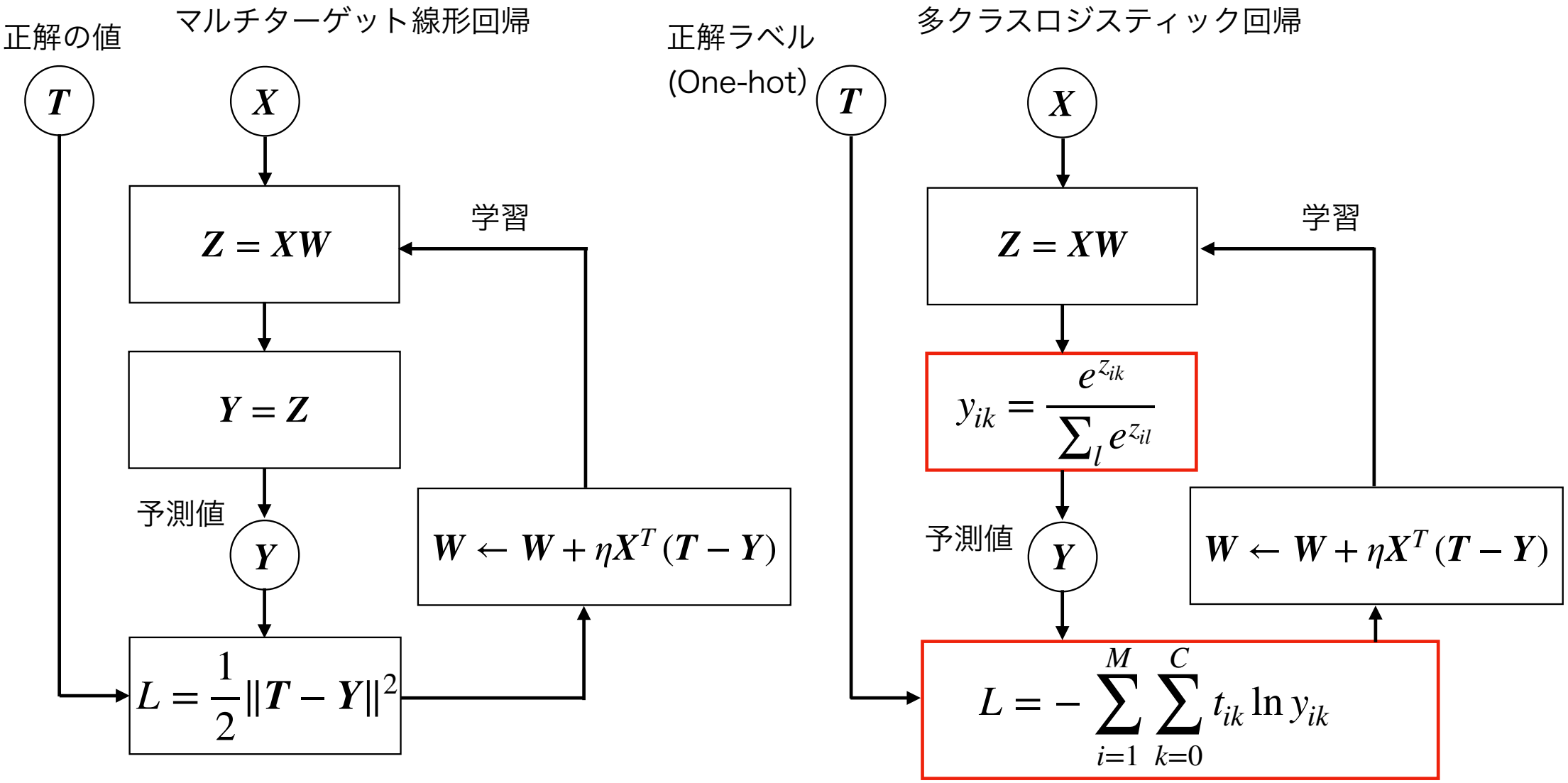
$$\frac{\partial z_{ik''}}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_{j'} x_{ij'} w_{j'k''} = \delta_{kk''} x_{ij}$$

更新式

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \eta \boldsymbol{X}^T (\boldsymbol{T} - \boldsymbol{Y})$$

（マルチターゲット線形回帰と全く同じ）

# マルチターゲット線形回帰と多クラスロジスティック回帰のまとめ



# 過学習
# (過剰適合)

# 教師あり学習のそもそもの目的

教師データ $D = \{\boldsymbol{t}, \boldsymbol{X}\}$

真の関係式

$$\boldsymbol{t} = f(\boldsymbol{X}) + \boldsymbol{\epsilon}$$

$\boldsymbol{\epsilon}$：ノイズ（確率変数）

$f(\boldsymbol{X})$ を知ることはできない。また、$\boldsymbol{\epsilon}$ の大きさを知る術もない

近似

$$f(\boldsymbol{X}) = y(\boldsymbol{X}) + \boldsymbol{d}$$

$$\boldsymbol{y} = y(\boldsymbol{X})$$

$\boldsymbol{d}$：近似誤差

仕方がないので、教師データ $D = \{\boldsymbol{t}, \boldsymbol{X}\}$ から $f(\boldsymbol{X})$ を推定する。つまり、$f(\boldsymbol{X})$ の近似モデル $y(\boldsymbol{X})$ を作ることが教師あり学習の目的。

$\rightarrow$ $\boldsymbol{t} = \boldsymbol{y} + \boldsymbol{e}$　　$\boldsymbol{e} \equiv \boldsymbol{\epsilon} + \boldsymbol{d}$：（推定）残差

# 損失関数の問題点

線形回帰の損失関数: 残差平方和

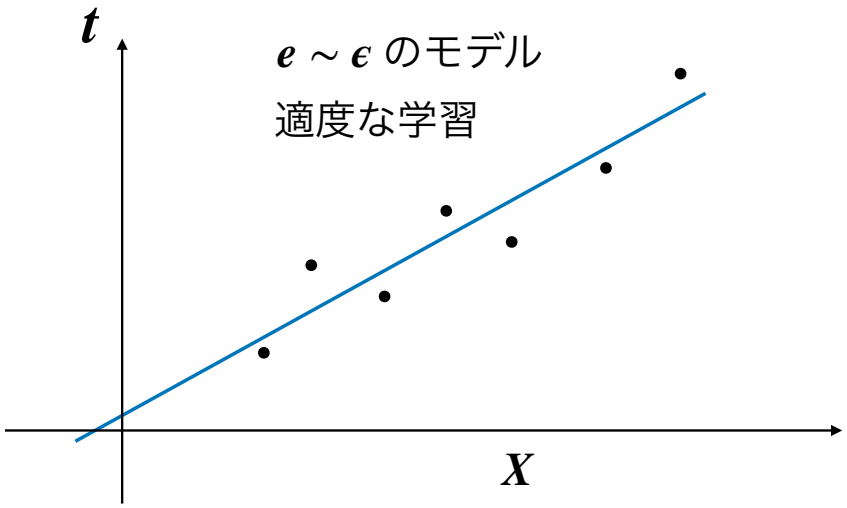
$$L = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{y}\|^2 = \frac{1}{2}\|\boldsymbol{e}\|^2$$

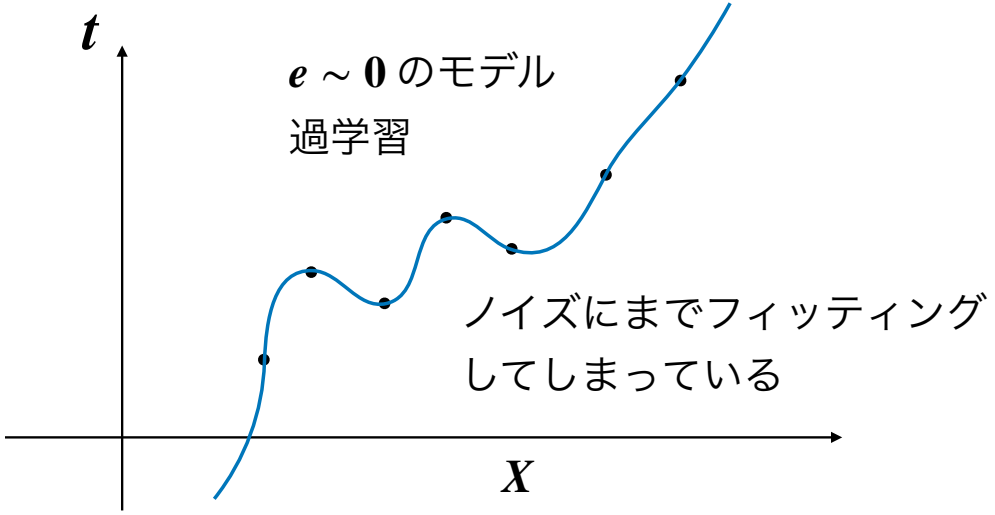
実はこの学習方法には問題がある。
本当に欲しいモデルは、

$$f(\boldsymbol{X}) \sim \boldsymbol{y} \qquad \boldsymbol{d} \sim \boldsymbol{0}$$

つまり、$\boldsymbol{e} \sim \boldsymbol{\epsilon}$ というモデルであって、上記の損失関数を用いた学習が目指しているような $\boldsymbol{e} \sim \boldsymbol{0}$ $(L \sim 0)$ というモデルではない。

残差は小さすぎてもいけないということ。



# 「推定」と「予測」の違い

（推定）残差

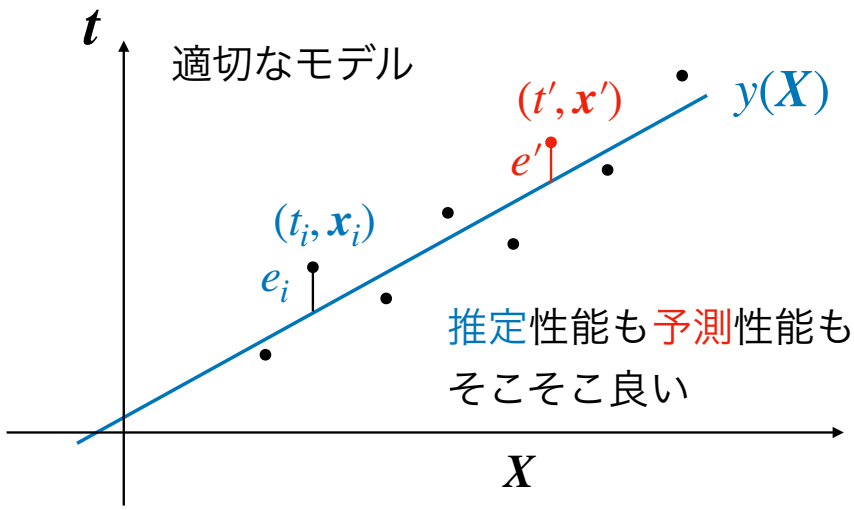
$$e_i = t_i - y(\boldsymbol{x}_i) \qquad t_i, \boldsymbol{x}_i \in D = \{\boldsymbol{t}, \boldsymbol{X}\}$$

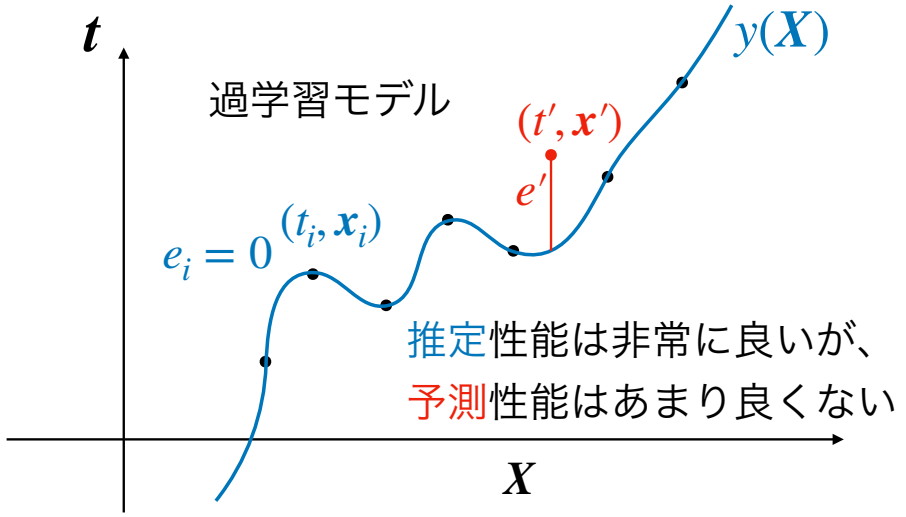
予測残差

$$e' = t' - y(\boldsymbol{x}') \qquad t', \boldsymbol{x}' \notin D = \{\boldsymbol{t}, \boldsymbol{X}\}$$

推定残差は（学習のために用いる）教師データの正解値とモデルの出力との差（フィッティングの誤差）で、学習済みモデルの場合、これが小さいのは当然（そうなるように学習を進める）。一方、予測残差は、新たに追加した（モデルの学習に使ってない）教師データ（の正解の値）とモデルの出力の差であり、これは、モデルの予測性能（汎化性能）の指標となる。

一般に、良いモデルとは予測性能が良いモデルのことである。
過学習したモデルは推定性能は非常に良いが予測性能は悪い。



# 過学習の回避方法

過学習の原因:

教師データの数に対し、モデルの表現力が高過ぎる（ウェイトの数が多過ぎる）

回避方法:

・教師データの数や種類を増やす（問題を難しくする）

・モデルの表現力を下げる

ウェイトの数を減らす

損失関数に罰則項を導入

Ridge回帰

$$L = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{y}\|^2 + \frac{1}{2}\lambda\|\boldsymbol{w}\|_2^2$$

LASSO

$$L = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{w}\|_1$$

過学習しているモデルのウェイトは大きくなりがち。

（過学習すると必ずウェイトが大きくなる、というわけではないが、ウェイトの大きさを制限するとモデルが過学習する余裕がなくなるのは確かである。）

p-ノルムの定義

$$\|\boldsymbol{w}\|_p = \left( \sum_i^N |x_i|^p \right)^{1/p}$$

# Ridge回帰とLASSO

## Ridge回帰

$$L(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{X}\boldsymbol{w}\|^2 + \frac{1}{2}\lambda\|\boldsymbol{w}\|_2^2$$

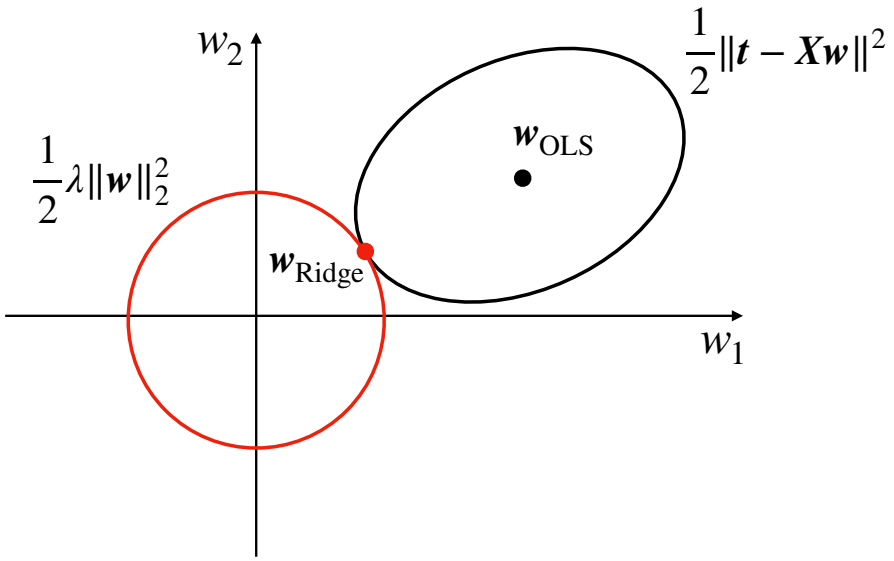
解析解

$$\boldsymbol{w}_{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda I\right)^{-1}\boldsymbol{X}^T\boldsymbol{t}$$

例え $\boldsymbol{X}^T\boldsymbol{X}$ が特異行列であったとしても、

$\boldsymbol{X}^T\boldsymbol{X} + \lambda I$ は正則行列となる。

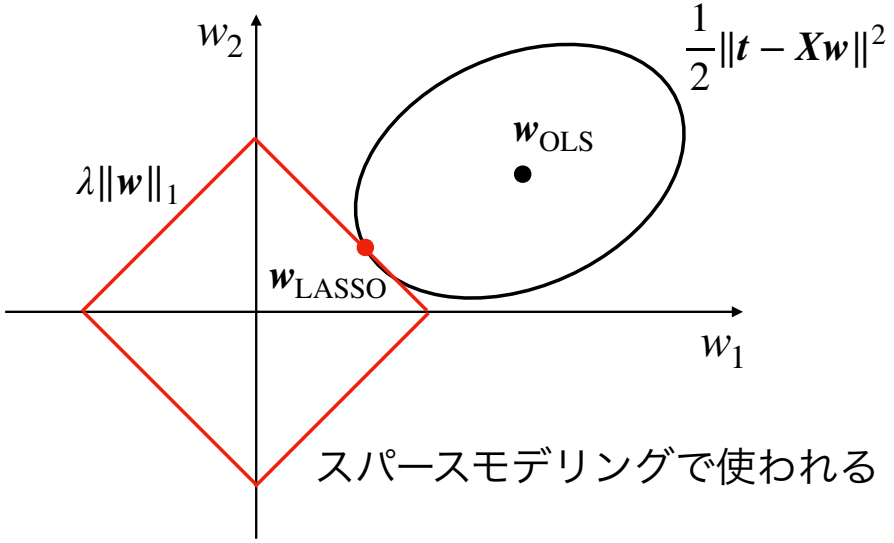
ゆえに罰則項 $\frac{1}{2}\lambda\|\boldsymbol{w}\|_2^2$ は**正則化**項とも呼ばれる。




## LASSO

$$L(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda\|\boldsymbol{w}\|_1$$

解析解なし（数値的に解く）




スパースモデリングで使われる

# ハイパーパラメータチューニング

$$D = \{t, X\}$$


モデル作成

$$w_{\text{Ridge}}(\lambda) = \left(X_{\text{train}}^T X_{\text{train}} + \lambda I\right)^{-1} X_{\text{train}}^T t_{\text{train}}$$

残差

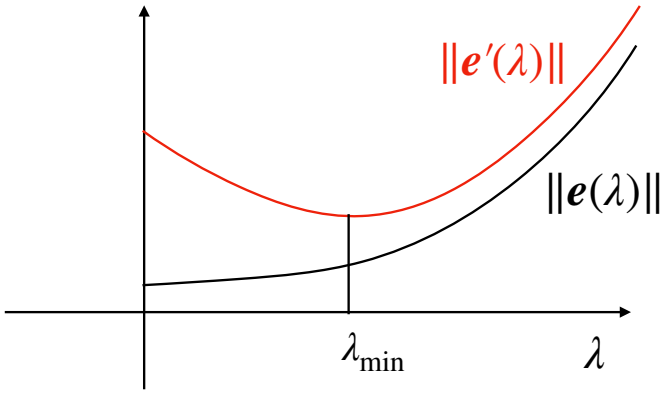
$$e(\lambda) = t_{\text{train}} - X_{\text{train}} w_{\text{Ridge}}(\lambda)$$

予測残差

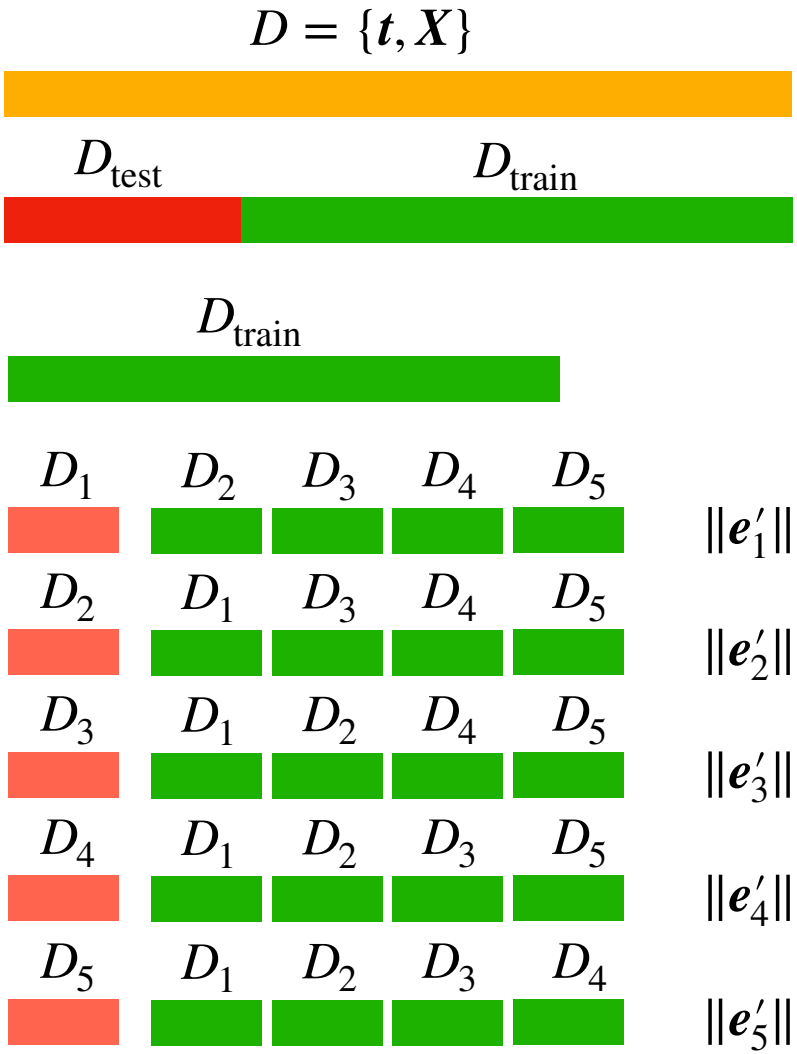
$$e'(\lambda) = t_{\text{test}} - X_{\text{test}} w_{\text{Ridge}}(\lambda)$$

一般的なハイパーパラメータチューニングの流れ

- 教師データ $D$ を検証データ $D_{\text{test}}$ と訓練データ $D_{\text{train}}$ に分割。
  （データ数に余裕がある場合、大体1:4に分割すると良い）
- ハイパーパラメータ $\lambda$ の値を決める。
- 訓練データ $D_{\text{train}}$ のみを使ってモデルを作る。
- 検証データ $D_{\text{test}}$ を使って予測残差 $e'(\lambda)$ を求める。
- ハイパーパラメータ $\lambda$ の値を変えて、同じことを繰り返す。
- 予測残差が最も小さくなる $\lambda_{\text{min}}$ を特定する。



# クロスバリデーションによるハイパーパラメータチューニング




K分割クロスバリデーション（K-fold CV）

・$D_{\text{train}}$ をさらにK分割し、それらのうちの一つを隠しておく。
・残りのデータでモデルを作る。
・隠しておいたデータに対する予測残差を求める。
・隠すデータを変えて（K回）繰り返す。
・K回の平均の予測残差を記録（分散も記録）。
・ハイパーパラメータ $\lambda$ の値を変えて、同じことを繰り返す。
・平均予測残差が最も小さくなる $\lambda_{\text{min}}$ を特定。

さらにダブルクロスバリデーションでは、$D_{\text{test}}$ での検証も行われる。

Kがサンプル数Mと同じ場合、つまり、M分割クロスバリデーションはLOOCV（leave-one-out CV: 一個抜き検証）と呼ばれる。