機械学習の整理帳

tomixy

2025 年 6 月 21 日

目次

第 1 章

機械学習とは何か

人工知能と機械学習

人工知能（AI: artificial intelligence）は包括的な用語

ref: なっとく！機械学習 p4〜6

> 🎓 人工知能　コンピュータが決定を下すことができるすべてのタスクを集めたもの

機械学習（machine learning）は人工知能の一部

> 🎓 機械学習　コンピュータが「データに基づいて」決定を下すことができるすべてのタスクを集めたもの

データとは、「経験」を表すコンピュータ用語

意思決定のプロセス

ref: なっとく！機械学習 p8〜9、p15

経験に基づいて意思決定を行うために人間が用いるプロセスは記憶・定式化・予測フレームワークと呼ばれ、次の 3 つのステップで構成されている

1. **記憶**：過去の同じような状況を思い出す
2. **定式化**：全般的なルールを定式化する
3. **予測**：このルールを使って将来起こるかもしれないことを予測する

コンピュータに「記憶・定式化・予測」フレームワークを使わせることで、コンピュータに私たちと同じように考えさせることができる

1. **記憶**：巨大なデータテーブルを調べる
2. **定式化**：さまざまなルールや式を調べてデータに最適なモデルを作成する
3. **予測**：モデルを使って未来（未知）のデータについて予測を行う

モデルとアルゴリズム

コンピュータはデータを使ってモデル（`model`）を構築するという方法で問題を解く

ref: なっとく！機械学習 p9、p15〜16

> 🎓 **モデル**　データを表すルールの集まりであり、予測を行うために使うことができる

モデルは、次のようなものと考えることができる

> 既存のデータをできる限り厳密に模倣する一連のルールを使って現実を表すもの

そして、最適なモデルとは、次のようなものである

新しいデータに最もうまく汎化するもの

最適なモデルを構築するためのさまざまなアルゴリズムがある

アルゴリズム（algorithm）は、モデルを構築するために使ったプロセスのこと

> 🎓 **アルゴリズム**　問題を解いたり計算を行ったりするために使われる手続き（一連のステップ）

データと特徴量

データがテーブルに含まれている場合、各行はデータ点である

たとえば、動物のデータセットがある場合、各行は異なる動物を表している

ref: なっとく！機械学習 p13、p19

このテーブル内の各動物は、その動物の特徴量（feature）によって説明される

> 🎓 **特徴量**　モデルが予測を行うために使うことができるデータの特性や属性

データがテーブルに含まれている場合、特徴量はテーブルの列であり、特徴量は各データを説明する

予測とラベル

特徴量の中には、ラベル（label）と呼ばれる特別なものがある

ref: なっとく！機械学習 p19〜20

一般に、特定の特徴量を他の特徴量に基づいて予測しようとしているなら、その特徴量はラベルである

機械学習モデルの目標は、

> データに含まれているラベルを推測すること

であり、モデルが行う推測を予測と呼ぶ

ラベル付きデータとラベルなしデータ

ref: なっとく！機械学習 p20～21

データには、大きく分けて、

- ラベル付きデータ：ラベルが付いているデータ
- ラベルなしデータ：ラベルが付いていないデータ

の 2 種類がある

予測したいと思うような列を持たないデータセットは、ラベルなしデータである

ラベル付きデータとラベルなしデータは、教師あり学習と教師なし学習という 2 種類の機械学習を生み出している

第 2 章

教師あり学習の概要

教師あり学習

教師あり学習（supervised learning）は、ラベル付きデータを扱う機械学習であり、その目標はラベルを予測すること
ラベルが付いていない新しいデータが渡された場合、教師あり学習モデルはそのデータ点のラベルを予測する

ref: なっとく！機械学習 p21

意思決定を行うためのフレームワーク「記憶・定式化・予測」は、教師あり学習の仕組みそのもの

1. データセットを記憶する
2. 特徴と考えられるものをモデル（ルール）として定式化する
3. 新しいデータが与えられたときに、そのデータのラベルを予測する

回帰モデルと分類モデル

教師あり学習モデルでは、数値と状態の 2 種類のデータが使われる

ref: なっとく！機械学習 p22〜25

- 数値データ：数値を用いるあらゆる種類のデータ
- カテゴリ値データ：カテゴリ（状態）を用いるあらゆる種類のデータ

そして、この 2 種類のデータから、次の 2 種類の機械学習モデルが生まれた

- 回帰モデル：数値データを予測する機械学習モデル
- 分類モデル：カテゴリ値データを予測する機械学習モデル

回帰モデル（`regression model`）は数値のラベルを予測するモデルであり、この数値を特徴量に基づいて予測する

分類モデル（`classification model`）は状態の有限集合に含まれている状態を予測するモデルである（カテゴリ値データでは、各データ点に有限のカテゴリ集合が紐づけられる）

第 3 章

教師なし学習の概要

教師なし学習

教師なし学習（unsupervised learning）は、ラベルなしデータを扱う機械学習である
ラベル（予測の目的変数または正解値）がないデータから、できるだけ多くの情報を抽出することが目標となる

ref: なっとく！機械学習 p25〜26

たとえば、ラベルが付いていない動物の画像のデータセットからは、それぞれの画像が表している動物の種類はわからないため、新しい画像がどの動物なのかを予測することはできない
しかし、2 つの画像が似ているかどうかなど、他にできることがある

つまり、教師なし学習アルゴリズムは、類似性に基づいてデータを分類できるが、それぞれのグループが何を表すのかはわからない

教師なし学習によるデータの前処理

実際には、教師なし学習はラベルが付いている場合でも利用できる

ref: なっとく！機械学習 p26

教師なし学習を使ってデータの前処理を行うと、教師あり学習の手法の効果を高めることができる

教師なし学習の種類

教師なし学習には、大きく分けて 3 種類の学習法がある

ref: なっとく！機械学習 p26

- クラスタリング：データを類似性に基づいてクラスタに分類する
- 次元削減：データを単純化し、より少ない特徴量でデータを正確に説明する
- 生成学習：既存のデータに似ている新しいデータ点を生成する

クラスタリング

クラスタリング（clustering）は、データセット内の要素を類似性の高いデータ点ごとにクラスタ（グループ）に分割する

ref: なっとく！機械学習 p26〜30

特徴量が 3 つを超えると、その次元を可視化できなくなるため、人間がクラスタを目で確認するのは不可能になる
コンピュータを使うことで、巨大なデータセットに対してもクラスタリングを行うことができる

次元削減

[Todo 1:]

ref: なっとく！機械学習 p30〜32

行列分解と特異値分解

[Todo 2:]

ref: なっとく！機械学習 p32〜34

生成学習

[Todo 3:]

ref: なっとく！機械学習 p34

第 4 章

強化学習の概要

強化学習

[Todo 4:]

ref: なっとく！機械学習 p35～37

第 5 章

線形回帰

線形回帰

できる限り多くのデータ点の近くを通る直線を求めることで、その直線の式を使って新たなデータを大まかに予測することができる
このような手法を線形回帰という

ref: なっとく！機械学習 p40
ref: 線形代数の半歩先 p112、p121～122

線形回帰は、次のような手順で行われる

1. モデルとして直線や平面を仮定する
2. 誤差を測る指標（誤差関数）を設定する
3. 誤差が最小になるようにモデルのパラメータを調整する

モデルを表す式

線形回帰では、モデルとして一次式を使う

ref: 線形代数の半歩先 p122～123
ref: なっとく！機械学習 p42

$$f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d$$

この式では、D 個の特徴量 $x_1, \ldots, x_D$ を持つデータを考えている

モデルを表す式において、それぞれの特徴量にかける係数 $w_1, \ldots, w_D$ を**重み**（weight）と呼ぶ

また、モデルを表す式では、どの特徴量にも結びつかない定数 w_0 がある
この定数を**バイアス**（bias）と呼ぶ

誤差関数

ref: なっとく！機械学習 p65〜
ref: 線形代数の半歩先 p123

どんな直線が適合するといえるのか、「データに当てはまる基準」も自分で設定することになる
誤差が少ない、すなわち「当てはまりがよい」ほど小さい値をとるような関数を**誤差関数**（error function）と呼ぶ

> 🎓 **誤差関数**　モデルの性能がどれくらいかを明らかにする指標であり、性能が悪いモデルに大きな値を割り当て、性能がよいモデルに小さな値を割り当てる関数

誤差関数は、**損失関数**（loss function）や**コスト関数**（cost function）、最適化問題としての側面に注目した場合は**目的関数**と呼ばれることもある

誤差関数を定義する一般的な方法としては、次の 2 つがある

- **絶対誤差**（absolute error）：直線からデータ点までの垂直距離を合計したもの
- **二乗誤差**（square error）：直線からデータ点までの垂直距離の二乗を合計したもの

線形回帰では、誤差関数を最小にするモデル（直線）を探すことになる
誤差関数として**二乗誤差**を用いた場合、その最小化問題は**最小二乗法**と呼

ばれる

二乗誤差と最小二乗法

データの総数を N とすると、二乗誤差は、次のような数式で表される

ref: 線形代数の半歩先 p123〜127

$$J(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - f(\boldsymbol{x}_n))^2$$

n 番目の実際の出力 y_n と、n 番目の入力を使ったときのモデルの出力 $f(\boldsymbol{x}_n)$ との差を見ている
符号を正にするために二乗し、それをすべてのデータについて合計したものが二乗誤差である

この誤差関数 $J(\boldsymbol{w})$ を最小にするパラメータを探すことが目標となる

モデルの式を整理する

まずは、モデルの式を整理する

$$f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d$$

右辺はベクトルの内積で書けそうだが、w_0 が余分なので、$x_0 = 1$ と定義して、次のように書き換える

$$f(\boldsymbol{x}) = \sum_{d=0}^{D} w_d x_d$$

そして、次のようなベクトルを導入する

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{bmatrix}, \quad \boldsymbol{x}' = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}, \quad \boldsymbol{x}'_n = \begin{bmatrix} 1 \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{bmatrix}$$

すると、先ほどのモデルの式は、次のように $\boldsymbol{w}$ と $\boldsymbol{x}'$ の内積で表せる

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}'$$

N 個分のデータをまとめる

N 個分のデータをまとめた出力 $\boldsymbol{y}$ と入力 X を、それぞれ次のように書く

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{bmatrix} = \begin{bmatrix} (\boldsymbol{x}'_1)^\top \\ (\boldsymbol{x}'_2)^\top \\ \vdots \\ (\boldsymbol{x}'_N)^\top \end{bmatrix}$$

X は $N \times (D+1)$ 行列で、定数項の分だけ列が一つ増えている

この定数項の列を含まず、データだけを並べたものはデータ行列と呼ばれる

ただし、ここでは定数項の列を含めた X もデータ行列と呼ぶことにする

誤差関数をベクトルと行列で表す

ここまでの記号を使って、誤差関数 $J(\boldsymbol{w})$ を書き直す

まずは n 番目のデータにのみ注目すると、実際の値とモデルの差は、

$$\begin{aligned} y_n - f(\boldsymbol{x}_n) &= y_n - \boldsymbol{w}^\top \boldsymbol{x}'_n \\ &= y_n - (\boldsymbol{x}'_n)^\top \boldsymbol{w} \end{aligned}$$

内積の順番を変える

ベクトルと行列を使うと、N 個のデータに対しては次のように書ける

$$\boldsymbol{z} = \begin{bmatrix} y_1 - (\boldsymbol{x}'_1)^\top \boldsymbol{w} \\ y_2 - (\boldsymbol{x}'_2)^\top \boldsymbol{w} \\ \vdots \\ y_N - (\boldsymbol{x}'_N)^\top \boldsymbol{w} \end{bmatrix} = \boldsymbol{y} - X\boldsymbol{w}$$

この二乗をとった形は、$\boldsymbol{z}$ 自身との内積で書き表せる

$$J(\boldsymbol{w}) = \boldsymbol{z}^\top \boldsymbol{z} = (\boldsymbol{y} - X\boldsymbol{w})^\top (\boldsymbol{y} - X\boldsymbol{w})$$

ベクトルの微分で最小化問題を解く

誤差関数を最小にする $\boldsymbol{w}$ を求めるには、

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$$

を解けばよい

[Todo 5: ref: 線形代数の半歩先 p125〜127]

絶対誤差

「データとモデルの差」を測るなら、次のような絶対値を用いた絶対誤差を使ってもよさそうだと考えられる

ref: 線形代数の半歩先 p127〜128
ref: なっとく！機械学習 p66〜67

$$J^{abs}(\boldsymbol{w}) = \sum_{n=1}^{N} |y_n - f(\boldsymbol{x}_n)|$$

しかし、絶対値は微分不可能な部分を生むため、二乗誤差の方がよく使われる

誤差関数を比較する観点

目的によっては、二乗誤差以外の誤差関数を選ぶ必要もある

ref: 線形代数の半歩先 p128〜129

最小値からのズレを許しやすいか

原点付近の様子を見ると、二次関数は最小値からずれても関数の値が急激には増加しない
絶対値関数の方が、少しでもズレが生じると、関数の値が急に増える

つまり、二乗誤差より絶対誤差の方が、最小値からのズレを許しづらくなる

ズレの大きさを無視するか

二次関数は、大きくずれると二乗の大きさで効く

絶対値関数は、原点以外は直線（比例）であるため、大きなズレはどれも同じような扱いをする

平均を用いた誤差関数

ref: なっとく！機械学習 p67〜68

実際には、「合計」ではなく「平均」を求める、平均絶対誤差（MAE: mean absolute error）や平均二乗誤差（MSE: mean square error）がよく使われる

合計は総数に影響される

たとえば、次の 2 つのデータセットを使って、誤差またはモデルを比較しようとしているとする

- 10 個のデータ点が含まれるデータセット
- 100 万個のデータ点が含まれるデータセット

誤差関数がデータ点ごとに 1 つの数字を「合計」したものだとすれば、合計する数字の量が多い 100 万個のデータ点を含むデータセットの方が、誤差（合計値）がはるかに大きくなってしまう

これらのデータセットを正しく比較したい場合は、「平均値」を求める必要がある

さらに平方根を使う

よく使われている誤差関数として、二乗平均平方根誤差（RMSE: root mean square error）と呼ばれるものもある

RMSE は、MSE の平方根として定義されるもので、次のような目的で使わ

れる

- 問題の単位を合わせる目的
- モデルが予測を行うときにどれくらい誤差が生じるのかをよく理解する目的

たとえば、住宅の価格を予測しようとしていて、価格の正解値と予測値の単位がドルだとする
二乗平均の単位は二乗ドルになってしまうため、平方根をとることで、正しい単位が得られる
これにより、住宅 1 軒あたりのモデルの大まかな誤差をドル単位でより正確に知ることができる

多項式回帰

データがほぼ直線上に並んでいる場合、線形回帰はうまく機能する
しかし、データがより複雑に分布している場合、直線のモデルを当てはめるのは無理がある

ref: なっとく！機械学習 p76〜78
ref: 線形代数の半歩先 p129〜132

そこで、線形回帰の拡張として多項式回帰がある

[Todo 6: 数学的な詳細を書く]

多項式回帰モデルの訓練では、訓練プロセスの前に多項式の次数（degree）を決めなければならない

パラメータとハイパーパラメータ

回帰モデルは、重みとバイアスによって定義され、これらはモデルのパラメータ（parameter）である

ref: なっとく！機械学習 p78
ref: 線形代数の半歩先 p142

しかし、モデルを訓練する「前に」調整できるつまみは他にもいろいろある

- 学習率（learning rate）：モデルを訓練する前に選択する非常に小さな値で、訓練時にモデルを微小変化させるのに役立つ
- エポック数：ループの繰り返し回数
- 次数（多項式回帰の場合）

これらのつまみはハイパーパラメータ（hyperparameter）と呼ばれる

パラメータとハイパーパラメータは、大まかには次のように見分けられる

- ハイパーパラメータ：訓練プロセスの「前に」設定する数量
- パラメータ：訓練プロセスの「最中に」モデルが作成または変更する数量

第 6 章

訓練プロセスの最適化

学習不足と過学習

次数を d とする多項式関数による多項式回帰を考える

$d = 1$ の場合、これは線形回帰であり、関数のグラフは直線になる

$d > 1$ の場合、関数のグラフは、最大で $d - 1$ 回カーブする曲線になる

ref: なっとく！機械学習 p84〜88

ref: 線形代数の半歩先 p133〜134

学習不足（次数が小さすぎる場合）

たとえば、データ点がまったく直線状に並んでいるとはいえないデータセットに対して、うまく適合する直線を見つけることはできない

複雑なデータに $d = 1$ の線形回帰を適用しても、その結果得られた直線では、未知のデータを予測することはできない

これが学習不足（underfitting）の例であり、データセットが複雑であるにもかかわらず、単純なモデルしかないという状態に陥っている

> 🎓 **学習不足**　訓練データのパターンをうまく認識するにはモデルの複雑さが十分ではなく、未知のデータにうまく汎化（対応）で

きないこと

学習不足が起こるのは、「単純すぎる」モデルを訓練しようとしたときである

過学習（次数が大きすぎる場合）

単純なデータセットに対して、次数 d の大きい多項式回帰を適用した場合にも問題が生じる

グラフのカーブが多すぎると、どのデータ点にもうまく接しているように見えるが、データが存在しないところで余分なカーブが発生する
このような曲線はデータの本質を捉えておらず、未知のデータに対して的外れな予測値を返す可能性が高い

これが過学習（overfitting）の例であり、モデルが過度に柔軟（複雑）だと、データがないところでの振る舞いが不自然になってしまう

> 🎓 **過学習**　モデルは訓練データのパターンを認識しているが、モデルが複雑すぎるために、未知のデータにうまく汎化できないこと

過学習が起こるのは、「複雑すぎる」モデルを訓練しようとしたときである

正則化

ref: なっとく！機械学習 p93〜97、p99

複雑なモデルは多くのパラメータを持ち、柔軟に調整することができる
しかし、モデルを柔軟にしすぎると過学習を起こす可能性がある
そこで、パラメータに制限（条件）をつけることで過学習を防ぐ手法として、正則化（regularization）がある

> 🎓 **正則化**　モデルの複雑さにペナルティを科す（モデルに格納できる情報の量を調整するか、モデルに格納できる情報の種類を制限する）ことで、最終的に過学習を防ぐ手法

複数のモデルをテストし、性能と複雑さのバランスが最もよいものを選ぶことでモデルの最適化を図る方法もあるが、正則化を用いる場合は、モデルを何種類も訓練する必要はない

正則化では、モデルを訓練するのは 1 回だけだが、モデルを訓練しながら次の 2 つの最適化も試みる

- 性能を向上させる
- 複雑さを減らす

そのために、性能の指標と複雑度の指標を数値化し、それらを組み合わせた上で最適化問題を解くという方法をとる

- 回帰誤差（`regression error`）：モデルの性能（品質）の指標
- 正規化項（`regularization term`）：モデルの複雑度の指標

回帰誤差として使われるのは、絶対誤差や二乗誤差などである
正規化項として使われるのは、モデルの L1 ノルムや L2 ノルムである

これらを用いて、性能がよくあまり複雑ではないモデルを見つけ出すために、次のように変更された誤差関数を最小化する

$$\text{誤差} = \text{回帰誤差} + \text{正則化項}$$

正則化パラメータ

正則化を用いても、モデルの性能を向上させようとすると複雑さが増し、モデルを単純化しようとすると性能が低下するという綱引き状態になること

ref: なっとく！機械学習 p100

がある
その場合は、ハイパーパラメータを使って、性能と複雑度の間で調整を行う

ここで使われるハイパーパラメータは、正則化パラメータと呼ばれるもので、その目的はモデルの訓練プロセスにおいて、性能と単純さのどちらを重視すべきかを決めることにある

正則化項に正則化パラメータ λ をかけたものに、回帰誤差を足し、その結果を使ってモデルを訓練する

$$\text{誤差} = \text{回帰誤差} + \lambda \cdot \text{正則化項}$$

λ の値を大きくすると（おそらく次数の小さい）単純なモデルになり、データセットにあまりうまく適合しないことがある
そのため、検証データセットを使って、モデルの性能が最も良くなる λ の値を選択することが重要となる

L1 ノルムと L2 ノルム

ref: なっとく！機械学習 p97〜98

L1 ノルム（L1 norm）と L2 ノルム（L2 norm）は、モデルの複雑度を数値化するために用いられる

係数の数が多いモデルや、係数の値が大きいモデルは、複雑になる傾向がある
そのため、次の二つの数値が、モデルの複雑度を表す指標となる

- L1 ノルム：係数の絶対値の合計
- L2 ノルム：係数の二乗の合計

絶対値と二乗を使うのは、負の係数をなくすためである
そうしないと、大きな負の値によって大きな正の値が相殺され、非常に複雑なモデルでもこれらの値が小さくなってしまうリスクがある

リッジ回帰

正則化項として L2 ノルムを使ってモデルを訓練する場合、そのモデルをリッジ回帰（ridge regression）と呼ぶ
「ridge」は山の尾根などを表す英単語であり、誤差関数の形状から名付けられている

ref: なっとく！機械学習 p99
ref: 線形代数の半歩先 p139〜140

リッジ回帰では、「パラメータのノルムは小さいほどよい」という条件を課すことで、モデルの複雑さを抑える

具体的には、ノルムの二乗（L2 ノルム）すなわち、自分自身との内積

$$\boldsymbol{w}^\top \boldsymbol{w}$$

を正則化項として加える

すると、誤差関数は次のように修正される

$$J^{\text{Ridge}}(\boldsymbol{w}) = (\boldsymbol{y} - X\boldsymbol{w})^\top (\boldsymbol{y} - X\boldsymbol{w}) + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

ラッソ回帰

正則化項として L1 ノルムを使ってモデルを訓練する場合、そのモデルをラッソ回帰（lasso regression）と呼ぶ
「lasso」は、「least absolute shrinkage and selection operator」の略である

ref: なっとく！機械学習 p99
ref: 線形代数の半歩先 p141〜144

ラッソ回帰では、「ベクトルの各要素の絶対値の和が小さい」という条件を課すことで、モデルの複雑さを抑える

具体的には、L1 ノルム

$$\sum_{d=1}^{D} |w_d|$$

を正則化項として加える

すると、誤差関数は次のように修正される

$$J^{\mathrm{Lasso}}(\boldsymbol{w}) = (\boldsymbol{y} - X\boldsymbol{w})^{\top}(\boldsymbol{y} - X\boldsymbol{w}) + \lambda \sum_{d=1}^{D} |w_d|$$

. . . . . .

Zebra Notes

Type	Number
todo	6