第 1 章

機械学習の基礎

機械学習（machine learning）とは、明示的にプログラミングせずに、コンピュータが十分な量のデータからその特徴や傾向を発見し、未知の事例に対しても予測を行うための科学分野である。画像認識や音声認識、株価の予測、機械による多言語間の翻訳、ネット上でのおすすめの商品の提案など、現在その応用は多岐にわたっている。欧州原子核研究機構（CERN）における素粒子実験では、大量のデータを解析するために機械学習が用いられている。

機械学習には大きく分けて、**教師あり学習**（supervised learning）、**教師なし学習**（unsupervised learning）、**強化学習**（reinforcement learning）の 3 つの形態が存在する。教師あり学習では、アルゴリズムに与えるデータに正解がついていて、システムがその傾向を学習することにより、未知のデータに対しても予測ができるようにする。教師なし学習では正解が与えられず、システムが自力で傾向を学習する。強化学習では、エージェントとよばれる学習システムが環境を観察して行動し、できるだけ高い報酬を得るように学習をする。

本論文の目的である手書き数字の認識は教師あり学習に分類される。

1.1　教師あり学習

例えば、コンピュータに犬と猫の画像を入力したら、写っている動物がどちらなのかを判別するようにさせたい。適切なシステムをプログラミングしたのち、これに多数の画像を与えてそれが犬か猫なのかを教え、耳が立っているとか鼻が長いなどそのパターンを学習させる。それが済んだら新たな写真をシステムに与えてどちらかを予測させる。このように、システムに与えるデータに正解が付されている場合、その学習アルゴリズムは**教師あり学習**（supervised learning）に分類される。

教師あり学習において重要な用語がいくつかあるため、箇条書きして説明する。

- **モデル**（model）: ある入力を受け取って何らかの処理を行い出力するシステムのことを機械学習の分野ではモデルという。
- **訓練データ**（training data）: モデルに学習をさせるために用いられるデータのこと。訓練データの集合を**訓練集合**（training set）という。各訓練データは一般にベクトルの情報であり、数式の上では $\boldsymbol{x}$ などを用いる。
- **テストデータ**（test data）: モデルがどの程度学習できたのかを測るために与えられるデータのこと。訓練集合と同じように、テストデータの集合を**テスト集合**（test set）という。
- **ラベル**（label）: モデルに与える各データにタグづけされた、モデルにとっての正解。犬と猫の分

類ならばそれらを 0 と 1 などで表し、手書きの数字画像を識別させるならば、その画像に書かれている数字がラベルとなる。数式の上では記号 y などを用いる。

- **クラス**（class）: 通常、ラベルは離散的である。そのラベルが意味する内容をクラスとよぶ。「犬」、「猫」、「0」、「1」、「2」…などがクラスである。犬と猫の例のように、2 つのクラスに分類する問題を**二値分類**（binary classification）、手書き数字のように分類すべきクラスが 3 つ以上あるときは**多クラス分類**（multiclass classification）という。
- **パラメータ**（parameter）: 最も単純なモデルの出力は、入力 $\boldsymbol{x}$ の何らかの関数である。その関数を $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ と表現したとき、この $\boldsymbol{w}$ と b をパラメータという。$\boldsymbol{w}$ を**重み**（weight）、b を**バイアス**（bias）という。重みとバイアスをまとめて、パラメータはギリシャ文字の $\boldsymbol{\theta}$ を用いて書かれる場合もある。モデルが行う学習とは、最適なパラメータの値を訓練データを用いて求めることである。
- **コスト関数**（cost function）: モデルの予測と実際のラベルとの差を表したもの。予測は一般にパラメータの関数であるから、コスト関数もパラメータの関数である。学習ではコスト関数が小さくなるように、パラメータを変化させていく。
- **ニューラルネットワーク**（neural network）: 教師あり学習をするモデルの一形態。神経細胞が連鎖したような構造をしており、**入力層**、**隠れ層**、**出力層**からなる。
- **ディープラーニング**（deep learning）: 多数の隠れ層の持つニューラルネットワークで学習をすること。**深層学習**ともいう。
- **汎化能力**（generalization ability）: 訓練時に与えられたデータだけでなく、新たな未知のデータに対しても正しい予測ができる能力のこと。
- **汎化誤差**（generalization error）: 学習を終えたモデルが新たな未知のデータについて予測を行ったとき、その予測と正しい答えとの誤差のこと。
- **計画行列**（design matrix）: 訓練データを 1 つ 1 つ入力するよりも、複数をまとめて入力する方が効率的な場合はしばしばある。このとき訓練データを用いて計画行列を作り、これを入力とする。具体的には m 個の訓練データ $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right),\ i = 1 \ldots, m$ が与えられたとし、スカラーの基底関数を $\phi_j(\boldsymbol{x}),\ j = 1, \ldots, k$ とすれば、計画行列 X の要素は $(X)_{ij} = \phi_j\left(\boldsymbol{x}^{(i)}\right)$ である。
- **過学習**（overlearning）: モデルの予測が訓練データに適合しすぎて、新たな未知のデータに対して誤った予測をしてしまうこと。**過剰適合**（overfitting）ともいう。原因としては訓練データが少なすぎることやパラメータの値が大きすぎる、パラメータが多すぎることなどが挙げられる。対応策としては訓練データを増やしたり、正則化を行うことが挙げられる。
- **正則化**（regularization）: 過学習を防ぐための手法の 1 つ。パラメータの値が大きくなりすぎないようにすること。具体的にはコスト関数に**正則化項**（regularization term）とよばれる $\lambda\|\boldsymbol{w}\|_1^1$ や $\lambda\|\boldsymbol{w}\|_2^2$ などの項を加えて学習をさせる。
- **ハイパーパラメータ**（hyperparameter）: モデルの学習では値を決めることのできないパラメータのこと。人間がこの値を調節する。

1.2　最適化

ある関数 $J(\boldsymbol{\theta})$ に対して、ある条件下で $J(\boldsymbol{\theta})$ が最小（あるいは最大）となる引数 $\boldsymbol{\theta}$ を求める問題を**最適化問題**（optimization problem）という。$J(\boldsymbol{\theta})$ の最小値、最大値を与える引数の集合をそれ

ぞれ $\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ J(\boldsymbol{\theta})$, $\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ J(\boldsymbol{\theta})$ と書き、要素が単一のときは要素そのものを返すとして取り扱う。$\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ J(\boldsymbol{\theta})$, $\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ J(\boldsymbol{\theta})$ の要素のことを、一般的には用いられないが本稿では**最小点**、**最大点**、**最適点**などとよぶことにし、その点を $\boldsymbol{\theta}^*$ のようにアスタリスク記号を付して表す。

紙とペンしかない状況では、$J(\boldsymbol{\theta})$ の導関数を求めてゼロとおいた方程式

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbf{0} \tag{1.1}$$

を解くことで最適点 $\boldsymbol{\theta}^*$ を求めることができる。条件付き最適化問題を解くための手法としては**ラグランジュの未定乗数法**（method of Lagrange multiplier）や、それを不等式制約に拡張した **KKT 条件**（Karush-Kuhn-Tucker conditions）を用いた解法などが知られている。しかし $J(\boldsymbol{\theta})$ によっては式 (1.1) の解析的な解が常に得られるとは限らないし、機械学習の分野では導関数の式の形ではなく、その値の方がより重要であることが多い。

最適化問題を解くための手法のことを**最適化アルゴリズム**（optimization algorithm）や**オプティマイザ**（optimizer）という。また関数の勾配を用いた最適化アルゴリズムの総称を**勾配法**（gradient method）という。ここでは機械学習における基本的なオプティマイザを 6 つ紹介する。

最急降下法（gradient descent, steepest descent）は勾配法のうち最も単純で基本的なものである。最急降下法は反復的なアルゴリズムで、k 回目の反復の引数を $\boldsymbol{\theta}^{(k)}$, 最小値を求める関数を $J(\boldsymbol{\theta})$ として、次のように計算を行う。

1. 引数 $\boldsymbol{\theta}$ の初期値 $\boldsymbol{\theta}^{(0)}$ をランダムに選ぶ。
2. $\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right) = \mathbf{0}$ ならば終了する。
3. そうでなければ次の式にしたがって引数 $\boldsymbol{\theta}$ を更新する。

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right) \tag{1.2}$$

4. 上記 2 と 3 を繰り返す。

計算機上で実際には正確に勾配が 0 に等しくなることはないため、 $\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right)$ が十分小さな値になった時点で終了する。式 (1.2) の係数 η をステップ幅、あるいは機械学習の文脈では**学習率**（learning rate）という。これは更新の度合いを決めるハイパーパラメータであり、通常小さな正の値に設定される。

学習率の値は大きすぎず小さすぎないのがよい。もし学習率が大きすぎると、ステップごとに最小点を通り過ぎてしまい、収束が遅くなるか収束できなくなるからである。反対に小さすぎると 1 ステップが小さいため収束が遅くなる。

また初期値 $\boldsymbol{\theta}^{(0)}$ の値によっては、求めたい大域的最小値ではなく局所的最小値に収束してしまう可能性がある。関数のグラフが局所的に谷になっている部分を**局所的最小値**（local minimum）といい、関数の真の最小値を**大域的最小値**（global minimum）という（図 1.1）。

機械学習では一般に、関数 $J(\boldsymbol{\theta})$ は m 個の訓練データそれぞれから得られる関数 $J_i(\boldsymbol{\theta}),\ i = 1, \ldots, m$ の和

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{m} J_i(\boldsymbol{\theta})$$

であるが（後述、式 (**??**) や式 (**??**))、その場合最急降下法はコストが高い方法である。パラメータのそれぞれの成分についての微分を計算するので、 1 ステップあたり $m \times k$ 回の計算が必要である。総ステッ

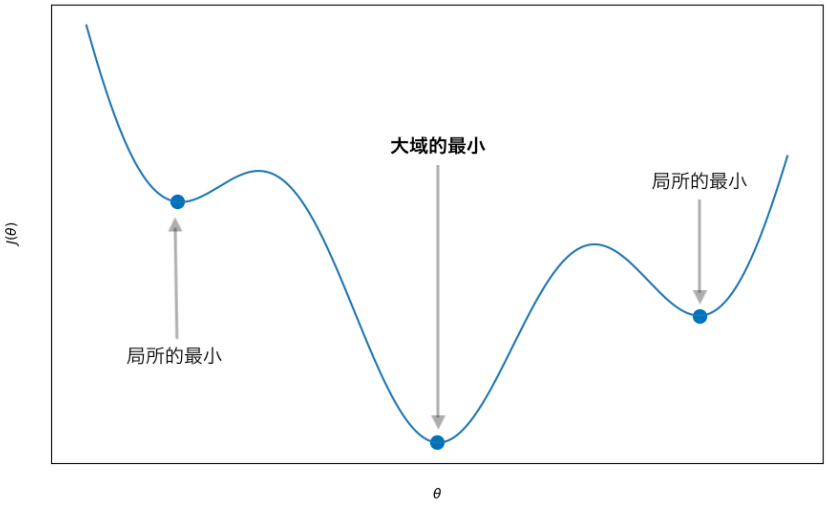

図 1.1　大域的最小と局所的最小

プは 1000 回とすると、例えば $m = 10000,\ k = 10$ であるとすれば全体で 10^8 回の計算が必要になってしまい、データ数が大きいと遅いアルゴリズムであることがわかる。

関数 $J(\boldsymbol{\theta})$ は m 個の訓練データに関する関数 $J_i(\boldsymbol{\theta})$ 全てを用いて計算されるが、このニュアンスを強調して最急降下法を**バッチ勾配降下法**（batch gradient descent）とよぶ。それに対して訓練データからランダムに 1 つ選んで、その関数のみについての勾配を計算し、それを用いて更新してゆく手法を**確率的勾配降下法**（stochastic gradient descent）という。その訓練データはステップごとに選び直す。

確率的な性質を持つため、最小点に向かってゆく過程は複雑になる。平均的には最小値に向かって緩やかに小さくなっていくのだが、最小値周りに達すると勾配は様々な方向を向いているので、1 箇所に落ち着くことがない。そのため確率的勾配降下法による最終的なパラメータは十分よいものだが最適ではない。しかし関数 $J(\boldsymbol{\theta})$ が複数の極小値を持つような複雑な関数の場合、確率的な性質のおかげで、関数の谷となっている部分から抜け出し、真の最小値にたどり着く可能性は大きくなる。

ミニバッチ勾配降下法（minibatch gradient descent）はバッチ勾配降下法と確率的勾配降下法の中間的なアルゴリズムである。ステップごとに訓練データから複数個のデータをランダムに抜き出して、それらの関数 $J(\boldsymbol{\theta})$ の勾配から最小値を探し出す。ランダムに選ばれた訓練データの集合を**ミニバッチ**（minibatch）とよぶ。N ステップ目の、$m'(< m)$ 個の訓練データからなるミニバッチを $\mathbb{B}_N$ と表せば、N ステップ目の関数は

$$J_N(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}^{(i)} \in \mathbb{B}_N} J_i(\boldsymbol{\theta})$$

とかける。N ステップ目ではこの勾配を用いて、それ自身の最小点を求める。

例えば $m = 60000$ 個の訓練データに対して、$m' = 1000$ のミニバッチ勾配降下法を行えば、60 回のパラメータ更新が行われるが、これを 1 エポックという。すなわち**エポック**（epoch）は訓練集合を 1 つ丸々使うのにかかる期間である。通常 1 エポックでは足りず、複数回行う必要がある。例でエポック数を 100 とすれば、パラメータ更新は $100 \times 60 = 6000$ 回行われる。

確率的勾配降下法とミニバッチ勾配降下法の違いは、ミニバッチに選ばれる訓練データが単一か複数かという点である。そのため両者の区別を気にせずにミニバッチ勾配降下法のことを確率的勾配降下法とよぶことがある。

確率的勾配降下法（ミニバッチ勾配降下法）は、その無作為性によって局所的な最小値から逃れる可能性はあがるが、最小値に落ち着かず収束が遅いという問題がある。Momentum SDG[?] はその影響を抑えることができる。そのアルゴリズムは式で

$$\begin{cases} \boldsymbol{v}^{(0)} & = \boldsymbol{0}, \\ \boldsymbol{v}^{(k+1)} & = \beta \boldsymbol{v}^{(k)} - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} J\big(\boldsymbol{\theta}^{(k)}\big), \\ \boldsymbol{\theta}^{(k+1)} & = \boldsymbol{\theta}^{(k)} + \boldsymbol{v}^{(k+1)} \end{cases} \tag{1.3}$$

と表せる。ただしここでのコスト関数 $J(\boldsymbol{\theta})$ はステップごとにランダムに選んだミニバッチを用いて計算されたものである。

これは物理的な解釈が可能である。質点 m の物体が抵抗係数 κ の空気中を運動することを考える。時刻を t として、連続的な位置ベクトル $\boldsymbol{\theta} = \boldsymbol{\theta}(t)$ と保存するポテンシャルエネルギー $J(\boldsymbol{\theta})$ を用いて、古典力学的な運動方程式は

$$m\frac{\mathrm{d}^2\boldsymbol{\theta}}{\mathrm{d}t^2} + \kappa\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = -\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

と与えられる。この運動方程式を離散化することで

$$m\frac{(\boldsymbol{\theta}(t+\Delta t) - \boldsymbol{\theta}(t)) - (\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-\Delta t))}{(\Delta t)^2} + \kappa\frac{\boldsymbol{\theta}(t+\Delta t) - \boldsymbol{\theta}(t)}{\Delta t} = -\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}(t))$$

を得るが、式変形をすれば

$$\boldsymbol{\theta}(t+\Delta t) - \boldsymbol{\theta}(t) = \frac{m}{m+\kappa\Delta t}(\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-\Delta t)) - \frac{(\Delta t)^2}{m+\kappa\Delta t}\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}(t))$$

となる。この式において

$$\boldsymbol{\theta}(t) \to \boldsymbol{\theta}^{(k)}, \quad \boldsymbol{\theta}(t+\Delta t) - \boldsymbol{\theta}(t) \to \boldsymbol{v}^{(k+1)},$$

$$\frac{m}{m+\kappa\Delta t} \to \beta, \quad \frac{(\Delta t)^2}{m+\kappa\Delta t} \to \eta$$

のように置き換えたのが式 (1.3) である。$\beta = 0$ とすれば通常の確率的勾配降下法に一致する。すなわち確率的勾配降下法は空気抵抗を無視して質点の運動を求めることに相当していたが、Momentum SDG は空気抵抗を考慮したものに相当する。また $\beta = m/(m+\kappa\Delta t)$ より $0 \leq \beta \leq 1$ であることがわかる。空気抵抗を考慮したので、質点がポテンシャルの最小点に近づいて速度を失ってゆくとステップの幅が小さくなってゆき、結果として確率的勾配降下法による振動の幅を小さくして、収束を早めることができる。

アルゴリズム AdaGrad[?] は学習率をステップごと、ベクトルの成分ごとに変えることで収束を早めることができる。

$$\begin{cases} \boldsymbol{v}^{(0)} = \boldsymbol{0}, \\ \boldsymbol{v}^{(k+1)} = \boldsymbol{v}^{(k)} + \boldsymbol{\nabla}_{\boldsymbol{\theta}} J\Big(\boldsymbol{\theta}^{(k)}\Big) \odot \boldsymbol{\nabla}_{\boldsymbol{\theta}} J\Big(\boldsymbol{\theta}^{(k)}\Big), \\ \boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \dfrac{\eta}{\sqrt{\boldsymbol{v}^{(k+1)}} + \epsilon} \odot \boldsymbol{\nabla}_{\boldsymbol{\theta}} J\Big(\boldsymbol{\theta}^{(k)}\Big) \end{cases} \tag{1.4}$$

と更新する。ただし式 (1.4) における $\eta/\left(\sqrt{\boldsymbol{v}^{(k+1)}}+\epsilon\right)$ は第 i 成分に $\eta/\left(\sqrt{v_i^{(k+1)}}+\epsilon\right)$ を持つようなベクトルで、$\epsilon>0$ は分母が 0 となることを防ぐ定数である。この $\boldsymbol{v}^{(k+1)}$ にはそれまでの各更新の勾配の 2 乗が加算されてゆき、各成分は更新ごとに大きくなるため、$\eta/\left(\sqrt{\boldsymbol{v}^{(k+1)}}+\epsilon\right)$ はだんだん小さくなってゆく。したがって勾配が小さくなると式 (1.4) で更新する量はさらに小さくなり、収束が早まることが期待される。しかし勾配がまだ大きくても更新が繰り返されると学習率の部分は小さくなってゆくので、更新が進まなくなる可能性もある。

Adam[**?**] は Momentum SGD と AdaGrad の両方を組み合わせたようなアルゴリズムで、以下のように更新する。

$$
\begin{cases}
\boldsymbol{m}^{(0)} = \boldsymbol{0}, \\
\boldsymbol{v}^{(0)} = \boldsymbol{0}, \\
\boldsymbol{m}^{(k+1)} = \beta_1 \boldsymbol{m}^{(k)} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right), & (1.5) \\
\boldsymbol{v}^{(k+1)} = \beta_2 \boldsymbol{v}^{(k)} + (1-\beta_2)\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right) \odot \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right), & (1.6) \\
\hat{\boldsymbol{m}}^{(k+1)} = \dfrac{\boldsymbol{m}^{(k+1)}}{1-\beta_1^k}, & (1.7) \\
\hat{\boldsymbol{v}}^{(k+1)} = \dfrac{\boldsymbol{v}^{(k+1)}}{1-\beta_2^k}, & (1.8) \\
\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \dfrac{\eta}{\sqrt{\hat{\boldsymbol{v}}^{(k+1)}}+\epsilon} \odot \hat{\boldsymbol{m}}^{(k+1)}.
\end{cases}
$$

式 (1.5) は Momentum SGD を改良したもので、これまでの更新の蓄積 $\boldsymbol{m}^{(k)}$ と勾配 $\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right)$ のどちらを重要視するかという割合を β_1 で表している。式 (1.6) は同様に AdaGrad を改良したもので、これまでの更新の蓄積 $\boldsymbol{m}^{(k)}$ と勾配の成分ごとの 2 乗のベクトル $\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right) \odot \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(k)}\right)$ の重みの割合を β_2 で表している。更新量が小さくなってゆくのを防ぐために式 (1.7) で $1 < 1/\left(1-\beta_1^k\right)$ 倍しておく。$\boldsymbol{v}^{(k)}$ の方も同様である (1.8)。

1.3　ベイズ推定

ベイズの定理を用いて、観測されたデータからその原因を確率論的な意味で推論することを**ベイズ推定**（Bayesian inference）とよぶ。ここでは最尤推定を紹介する。

いま R という結果を手に入れている。この結果がなぜ R であったのか、原因を推定したい。様々な原因が考えられ、中には両立できないものも含まれるだろう。目下のタスクは、R という結果が得られたもとで原因が C であった確率を求めることである。

ベイズの定理の式 (**??**) によれば、その確率 $P(C \mid R)$ は次式で求められる。

$$
P(C \mid R) = \frac{P(R \mid C)P(C)}{P(R)} \propto P(R \mid C)P(C). \tag{1.9}
$$

この式の分母に現れる $P(R) = \sum_C P(R \mid C)P(C)$ は単に規格化定数であり、本質的ではない。

ここで式 (1.9) の各確率の名称とその意味を整理する。

- $P(C \mid R)$：**事後確率**（posterior probability）。R という結果が得られたもとで原因が C であった確率で、C の関数。

- $P(C)$：**事前確率**（prior probability）。**結果を得る前に**原因が C であると想定した確率で、C の関数。
- $P(R \mid C)$：**尤度**（likelihood）。原因が C と仮定した場合に結果として R が得られる確率で、もっともらしさを表している。R の関数。

式 (1.9) は、結果を得る前の確率（事前確率）に、新しい結果によって得られる尤度をかけることによって、よりもっともらしい確率（事後確率）に更新していることを意味している。

ここではニューラルネットワークに手書き数字の認識をさせることを目的として、m 個の訓練データの集合を $\mathbb{X} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$, その各訓練データ $\boldsymbol{x}^{(i)}$ に対応したラベル $y^{(i)}$ の集合を $\mathbb{Y} = \left\{y^{(1)}, \ldots, y^{(m)}\right\}$ とする。$\mathbf{x}$ と y を確率変数として、以下 3 つの確率分布を扱う。

- $p_{\text{data}}(\mathrm{y} \mid \mathbf{x})$：$\mathbf{x}$ と y を生成する、真であるが未知の分布。
- $p_{\text{model}}(\mathbb{Y} \mid \mathbb{X}; \boldsymbol{\theta})$：$\mathbb{X}$ が与えられたもとで $\mathbb{Y}$ が得られる確率。これは $\boldsymbol{\theta}$ をパラメータとして $p_{\text{data}}(\mathbb{Y} \mid \mathbb{X})$ を予測したもので、尤度に相当する。
- $\hat{p}_{\text{data}}(\mathrm{y} \mid \mathbf{x})$：訓練データ $\mathbb{X}$ と $\mathbb{Y}$ から得られた経験的な分布。得られたデータ $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right),\ i = 1, \ldots, m$ のみで確率密度を持つような分布で、数式ではディラックのデルタ関数を用いて

$$\hat{p}_{\text{data}}(\mathrm{y} = y \mid \mathbf{x} = \boldsymbol{x}) = \frac{1}{m} \sum_{i=1}^{m} \delta\left(\boldsymbol{x} - \boldsymbol{x}^{(i)}\right) \delta\left(y - y^{(i)}\right)$$

  と表せる。

コンピュータは訓練集合 $\{\mathbb{X}, \mathbb{Y}\}$ からパラメータ $\boldsymbol{\theta}$ を変化させることで $p_{\text{data}}(\mathbb{Y} \mid \mathbb{X})$ を尤度 $p_{\text{model}}(\mathbb{Y} \mid \mathbb{X}; \boldsymbol{\theta})$ で近似する。この尤度に最大値を与えるパラメータの値 $\hat{\boldsymbol{\theta}}_{\text{MLE}}$ を最適値と考える推定方法を**最尤推定**（maximum likelihood estimation, MLE）といい、機械学習で用いられるベイズ推定の手法の一つである。 これを計算すると次のようになる。

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{\text{MLE}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{Y} \mid \mathbb{X}; \boldsymbol{\theta}) && (1.10)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) && (1.11)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) && (1.12)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \int \mathrm{d}\boldsymbol{x}\, \mathrm{d}y\, \frac{1}{m} \sum_{i=1}^{m} \delta\left(\boldsymbol{x} - \boldsymbol{x}^{(i)}\right) \delta\left(y - y^{(i)}\right) \log p_{\text{model}}(y \mid \boldsymbol{x}; \boldsymbol{\theta}) && (1.13)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x},\mathrm{y} \sim \hat{p}_{\text{data}}}[\log p_{\text{model}}(y \mid \boldsymbol{x}; \boldsymbol{\theta})] && (1.14)\\
&= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x},\mathrm{y} \sim \hat{p}_{\text{data}}}[-\log p_{\text{model}}(y \mid \boldsymbol{x}; \boldsymbol{\theta})]. && (1.15)
\end{aligned}$$

(1.10) は最尤推定による推定値の定義である。(1.10) から (1.11) への変形は、訓練データが**独立同分布**（independent and identically distributed, i.i.d.）に従う、すなわち訓練データの各点は同一の分布 $p_{\text{data}}(\mathrm{y} \mid \mathbf{x})$ から生成されるが、互いに独立であると仮定したため、$p_{\text{model}}(\mathbb{Y} \mid \mathbb{X}; \boldsymbol{\theta})$ は各訓練データの確率の総積で与えられる。(1.11) から (1.12) へは、対数関数が単調増加であるから最大点は同じであることを利用した。そのため総積の記号 $\prod$ は総和の記号 $\sum$ に置き換わっている。それをディラックのデルタ関数を用いて表現したのが (1.12) から (1.13) への変形である。ただし関数を定数倍

してもその最大点自体は変化しないため $1/m$ をかけておいた。これは $\mathbf{x}$ と y が経験的な確率分布 $\hat{p}_{\text{data}}$ に従うもとでの $\log p_{\text{model}}(y \mid \boldsymbol{x}; \boldsymbol{\theta})$ の期待値である (1.14)。最後に符号を反転させて $\underset{\boldsymbol{\theta}}{\text{argmax}}$ を $\underset{\boldsymbol{\theta}}{\text{argmin}}$ に置き換えた (1.15)。式 (**??**) を参照すればわかるように、最尤推定は交差エントロピー $H(\hat{p}_{\text{data}}, p_{\text{model}}) = \mathbb{E}_{\mathbf{x},\text{y}\sim\hat{p}_{\text{data}}}[-\log p_{\text{model}}(y \mid \boldsymbol{x}; \boldsymbol{\theta})]$ の最小化問題に帰着する。