

5.6 Bayesian Statistics

ベイズ統計における確率

- 頻度論での確率は「データ x を無限に観測した際の発生頻度(割合)」というニュアンス
- ベイズでは、データ x の確率については頻度論と同様な解釈もできるが、パラメータ θ の確率は「確からしさや belief」という広いニュアンス

例えば [Bayesian probability](#) では

Bayesian probability is an interpretation of the concept of probability, in which, instead of frequency or propensity of some phenomenon, probability is interpreted as reasonable expectation representing a state of knowledge or as quantification of a personal belief.

と書かれており、分かりやすい。電通大の植野先生の[講義資料](#)にもまとまっている。

頻度論とベイズ

	頻度論	ベイズ
パラメータの真値	未知の定数として扱う	確率変数として扱い、取りうる値の確からしさを確率分布で表現
予測	パラメータを点推定した単一のモデルで予測	パラメータの確率で無数のモデルを加重平均(アンサンブル)して予測
イメージ	客観(データ、尤度)100%	主観(信念や知識、事前確率) + 客観
尤度の使い方	最尤推定での目的関数	事前分布を尤度で更新する

この2つの関係性として、例えば…

- $n \rightarrow \infty$ の時、ベイズ推定の結果はしばしば頻度論と一致する。詳細は Bishop (2006).
- 頻度論での荷重減衰を使った正則化推定は、しばしばベイズの MAP 推定としても解釈できる。詳細は後述。

ベイズモデリングのよくある流れ

1. データ x が発生する分布 $p(x|\theta)$ を設計する.
2. パラメータ θ の事前分布 $p(\theta)$ を設計する. あまり確信が無い場合はエントロピーの高い分布を選ぶ. 共役事前分布を使うと後々が楽.
3. 観測されたデータ $x^{(1)}, \dots, x^{(m)}$ が観測から、尤度 $p(x^{(1)}, \dots, x^{(m)}|\theta)$ を算出する. i.i.d. なら $p(x^{(1)}, \dots, x^{(m)}|\theta) = \prod_i^m p(x^{(i)}|\theta)$.
4. ベイズの定理を用いて、事前分布を尤度で更新し事後分布 $p(\theta|x^{(1)}, \dots, x^{(m)})$ を得る.

$$p(\theta|x^{(1)}, \dots, x^{(m)}) = \frac{p(x^{(1)}, \dots, x^{(m)}|\theta) p(\theta)}{p(x^{(1)}, \dots, x^{(m)})} \quad (5.67)$$

$$\propto p(x^{(1)}, \dots, x^{(m)}|\theta) p(\theta)$$

事後分布 \propto 尤度 \times 事前分布

\propto を使って θ に関係しない規格化定数の部分を無視している.

5. 将来観測されるデータ $x^{(m+1)}$ の予測分布 $p(x^{(m+1)}|x^{(1)}, \dots, x^{(m)})$ を得る.

$$p(x^{(m+1)}|x^{(1)}, \dots, x^{(m)}) = \int_{\Theta} p(x^{(m+1)}, \theta|x^{(1)}, \dots, x^{(m)}) d\theta$$

$$= \int_{\Theta} p(x^{(m+1)}|\theta) p(\theta|x^{(1)}, \dots, x^{(m)}) d\theta \quad (5.68)$$

比較：最尤推定の場合

1. データ x が発生する分布 $p(x;\theta)$ を設計する.
2. 観測されたデータから、尤度 $p(x^{(1)}, \dots, x^{(m)}|\theta)$ を算出する.
3. 最適化問題を解いて最尤推定量 $\hat{\theta}$ を得る.
4. 将来観測されるデータ $x^{(m+1)}$ を $p(x^{(m+1)};\hat{\theta})$ で予測する.

アンサンブル学習としての解釈

予測分布 (5.68) の右辺の積分を和に置き換えてみると、

$$\sum_{i=1}^{\infty} p(x^{(m+1)}|\theta_i) p(\theta_i|x^{(1)}, \dots, x^{(m)})$$

となる.

これは「無限個のモデル $p(x^{(m+1)}|\theta_i)$ をパラメータ θ_i の確からしさ $p(\theta_i|x^{(1)}, \dots, x^{(m)})$ で重み付け平均したもの」と解釈できる.

単一のモデル $p(x^{(m+1)}|\hat{\theta})$ を使う最尤アプローチと比べて,

- 訓練データが限られているときにも汎化しやすい
- が、計算コストは高い

参考：[上田 \(2005\)](#)

パラメータの不確かさの評価

推定したモデルを解釈する上で、パラメータの不確かさ（どのあたりの値が妥当か）を評価したくなる。

- 頻度論では最尤推定量の信頼区間から評価する。解釈が若干煩雑で、例えば「95 % 信頼区間は「サンプリングを繰り返したときにその区間にパラメータ真値が含まれる確率が 95 %」という意味。」
- ベイズでは $p(\theta|x^{(1)}, \dots, x^{(m)})$ の分布を直接見て評価でき、シンプル。

Example: Bayesian Linear Regression

設定

線形回帰モデルをベイズ的に捉える。簡単のため切片項は無視する。

- $y \in \mathbb{R}$: スカラーのアウトカム。
- $\mathbf{x} \in \mathbb{R}^n$: 入力ベクトル。
- $\mathbf{w} \in \mathbb{R}^n$: 回帰係数ベクトル。
- \mathbf{X} : m 個の訓練データの計画行列。
- \mathbf{y} : m 個の訓練データのアウトカムのベクトル。

1. アウトカムの分布（モデル）を設計

簡単のため、分散1のガウス誤差を仮定し、

$$p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y; \mathbf{w}^T \mathbf{x}, 1)$$

とする。

2. パラメータの事前分布を設計

パラメータに関する事前の信念や情報は少ないとし、エントロピーの高いガウス分布

$$\begin{aligned} p(\mathbf{w}) &= \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0) \\ &= \sqrt{\frac{1}{(2\pi)^n \det(\boldsymbol{\Lambda}_0)}} \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^T \boldsymbol{\Lambda}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right) \\ &\propto \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^T \boldsymbol{\Lambda}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right) \quad (5.73) \end{aligned}$$

で事前分布を表す。（後々、これが共役事前分布になっていることがわかる。）

$\boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0$ はハイパー-パラメータで、通常は相関構造を考えず $\boldsymbol{\Lambda}_0 = \text{diag}(\boldsymbol{\lambda}_0)$ とする。

3. 尤度を算出

i.i.d. を仮定すれば、

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \prod_{i=1}^m \mathcal{N}(y_i; \mathbf{w}^T \mathbf{x}_i, 1) \\ &= \mathcal{N}(\mathbf{y}; \mathbf{X}\mathbf{w}, \mathbf{I}) \\ &= \sqrt{\frac{1}{(2\pi)^m \det(\mathbf{I})}} \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{I}^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w})\right) \\ &\propto \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})\right) \quad (5.72) \end{aligned}$$

と算出できる。

4. 事前分布を尤度で更新して事後分布を得る

ベイズの定理を用いて、2で設定した事前分布 (5.73) を3で算出した尤度 (5.72) で更新することで、事後分布を計算していくと、

$$\begin{aligned}
p(\mathbf{w}|\mathbf{X}, \mathbf{y}) &= \frac{p(\mathbf{w}, \mathbf{y}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} \\
&= \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})} \\
&\propto p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w}) \\
&\propto \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})\right) \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^T\boldsymbol{\Lambda}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right) \\
&= \exp\left(-\frac{1}{2}(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}\right. \\
&\quad \left.+ \mathbf{w}^T\boldsymbol{\Lambda}_0^{-1}\mathbf{w} - \mathbf{w}^T\boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_0^T\boldsymbol{\Lambda}_0^{-1}\mathbf{w} + \boldsymbol{\mu}_0^T\boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)\right) \\
&\propto \exp\left(-\frac{1}{2}(-2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\boldsymbol{\Lambda}_0^{-1}\mathbf{w} - 2\boldsymbol{\mu}_0^T\boldsymbol{\Lambda}_0^{-1}\mathbf{w})\right) \quad [\text{板書で補足}]
\end{aligned}$$

となり、さらに指数関数 $\exp(-\frac{1}{2}(\cdot))$ の中身を整理していくと、

$$\begin{aligned}
\text{指数関数の中身} &= \mathbf{w}^T(\mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0^{-1})\mathbf{w} - 2(\mathbf{y}^T\mathbf{X} + \boldsymbol{\mu}_0^T\boldsymbol{\Lambda}_0^{-1})\mathbf{w} \\
&= \mathbf{w}^T(\mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0^{-1})\mathbf{w} - 2(\mathbf{X}^T\mathbf{y} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)^T\mathbf{w} \quad [2\text{次形式の形}] \\
&= [\text{平方完成する}] \\
&= (\mathbf{w} - (\mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0^{-1})^{-1}(\mathbf{X}^T\mathbf{y} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0))^T(\mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0^{-1}) \\
&\quad (\mathbf{w} - (\mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0^{-1})^{-1}(\mathbf{X}^T\mathbf{y} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)) - \frac{1}{4}(\mathbf{w} \text{に関係しない部分}) \\
&= (\mathbf{w} - \boldsymbol{\mu}_m)^T\boldsymbol{\Lambda}_m^{-1}(\mathbf{w} - \boldsymbol{\mu}_m)^T - \frac{1}{4}(\mathbf{w} \text{に関係しない部分})
\end{aligned}$$

となる。ここで、 $\boldsymbol{\Lambda}_m^{-1} = (\mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0^{-1})$, $\boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m(\mathbf{X}^T\mathbf{y} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)$ と置いている。

以上より、事後分布について、

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \propto \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_m)^T\boldsymbol{\Lambda}_m^{-1}(\mathbf{w} - \boldsymbol{\mu}_m)^T\right) \quad (5.78)$$

が成り立つ。したがって結果をまとめると、事後分布はガウス分布

$$\begin{aligned}
p(\mathbf{w}|\mathbf{X}, \mathbf{y}) &= \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m) \\
\boldsymbol{\Lambda}_m^{-1} &= (\mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0^{-1}), \quad \boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m(\mathbf{X}^T\mathbf{y} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)
\end{aligned}$$

であることがわかった。

補足：2次形式の平方完成

2次形式 (i) を平方完成して (ii) の形に変形することを考える。 \mathbf{Q} を対称行列とするが、こうしても一般性は失われない。

$$\begin{aligned}
&\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{q}^T\mathbf{x} \quad (i) \\
&(\mathbf{x} + \mathbf{z})^T\mathbf{Q}(\mathbf{x} + \mathbf{z}) + c \quad (ii)
\end{aligned}$$

ここで、(ii) を変形すると、

$$\begin{aligned}
(\mathbf{x} + \mathbf{z})^T \mathbf{Q} (\mathbf{x} + \mathbf{z}) + c &= \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{x}^T \mathbf{Q} \mathbf{z} + \mathbf{z}^T \mathbf{Q} \mathbf{x} + \mathbf{z}^T \mathbf{Q} \mathbf{z} + c \\
&= \mathbf{x}^T \mathbf{Q} \mathbf{x} + (\mathbf{x}^T \mathbf{Q} \mathbf{z})^T + \mathbf{z}^T \mathbf{Q} \mathbf{x} + \mathbf{z}^T \mathbf{Q} \mathbf{z} + c \\
&= \mathbf{x}^T \mathbf{Q} \mathbf{x} + 2\mathbf{z}^T \mathbf{Q} \mathbf{x} + \mathbf{z}^T \mathbf{Q} \mathbf{z} + c \quad (\text{ii}'')
\end{aligned}$$

となり、(i) と (ii'') の係数を比較することで、

$$\begin{aligned}
&\left\{ \begin{array}{l} \mathbf{q}^T = 2\mathbf{z}^T \mathbf{Q} \\ 0 = \mathbf{z}^T \mathbf{Q} \mathbf{z} + c \end{array} \right. \\
\Leftrightarrow \left\{ \begin{array}{l} \mathbf{z} = \frac{1}{2}(\mathbf{q}^T \mathbf{Q}^{-1})^T = \frac{1}{2} \mathbf{Q}^{-1} \mathbf{q} \\ c = -\mathbf{z}^T \mathbf{Q} \mathbf{z} = -\frac{1}{4}(\mathbf{Q}^{-1} \mathbf{q})^T \mathbf{Q} (\mathbf{Q}^{-1} \mathbf{q}) = -\frac{1}{4} \mathbf{q}^T \mathbf{Q}^{-1} \mathbf{Q} \mathbf{Q}^{-1} \mathbf{q} = -\frac{1}{4} \mathbf{q}^T \mathbf{Q}^{-1} \mathbf{q} \end{array} \right.
\end{aligned}$$

となる。したがって (i) は次のように平方完成できる。

$$\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{q}^T \mathbf{x} = \left(\mathbf{x} + \frac{1}{2} \mathbf{Q}^{-1} \mathbf{q} \right)^T \mathbf{Q} \left(\mathbf{x} + \frac{1}{2} \mathbf{Q}^{-1} \mathbf{q} \right) - \frac{1}{4} \mathbf{q}^T \mathbf{Q}^{-1} \mathbf{q}$$

何が言えたか

ガウス分布で設計した事前分布を尤度で更新したら、事後分布もガウス分布となった。

このように「事前分布」と「それを尤度で更新して得られる事後分布」が同じ形状(族)になるとき、これを**共役事前分布**と呼ぶ。

何が嬉しいかと言うと、

- 事後分布を得るために複雑な計算は要らず、データをもとに事前分布のパラメータを少し調整するだけで良い。 $\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0) \rightarrow \mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m)$
- データが次々と系列的に観測されるとき、常に分布形は同じままで、逐次的に学習できる。例えば追加のデータ $(\mathbf{X}', \mathbf{y}')$ が得られたら、先ほどの事後分布 $\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m)$ を事前分布として使って、同じように

$$\begin{aligned}
\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m) &\rightarrow \mathcal{N}(\boldsymbol{\mu}_{m'}, \boldsymbol{\Lambda}_{m'}) \\
\boldsymbol{\Lambda}_{m'}^{-1} &= (\mathbf{X}'^T \mathbf{X}' + \boldsymbol{\Lambda}_m^{-1}), \quad \boldsymbol{\mu}_{m'} = \boldsymbol{\Lambda}_{m'} (\mathbf{X}'^T \mathbf{y}' + \boldsymbol{\Lambda}_m^{-1} \boldsymbol{\mu}_m)
\end{aligned}$$

と更新すれば良い。

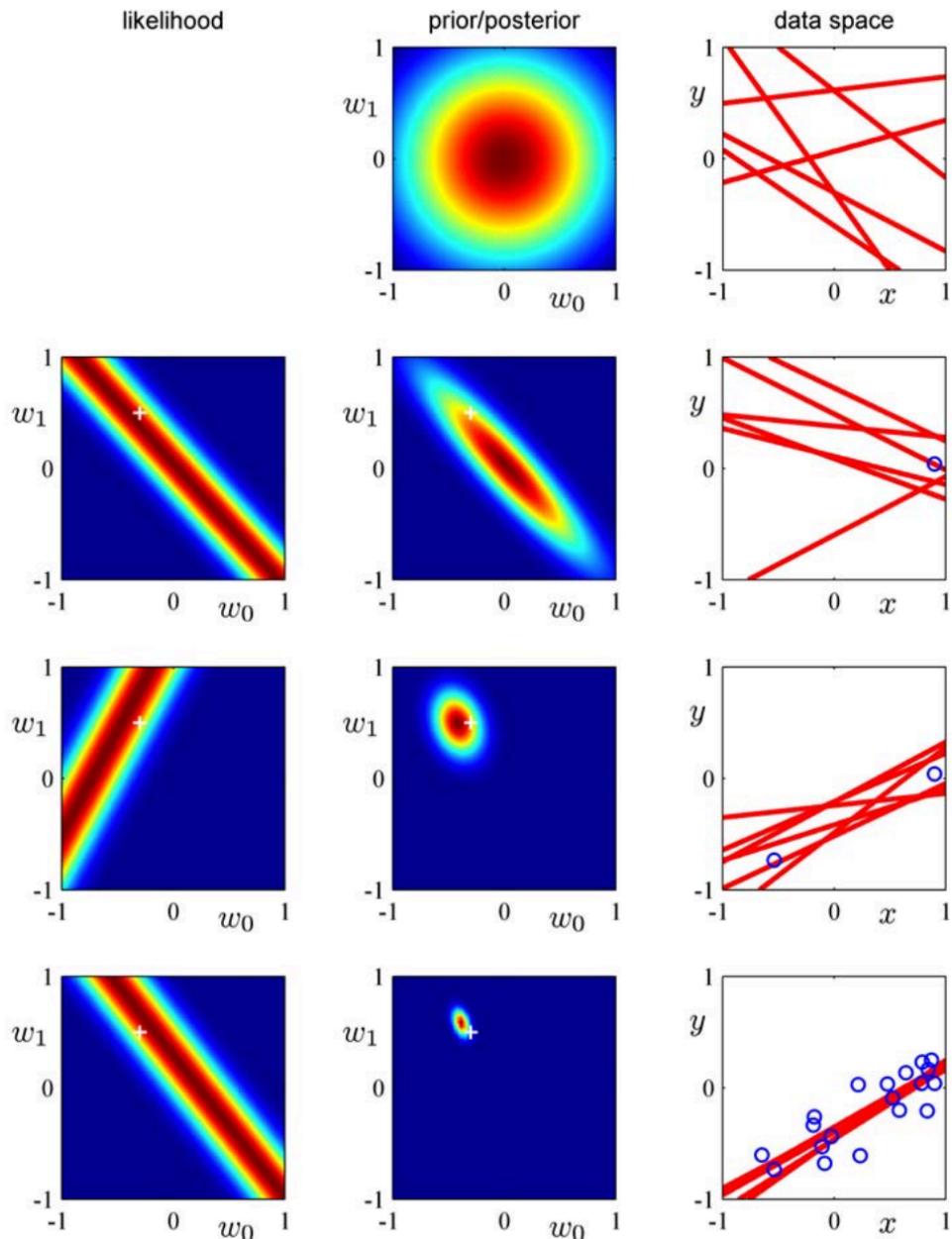


Figure 3.7 Illustration of sequential Bayesian learning for a simple linear model of the form $y(x, \mathbf{w}) = w_0 + w_1 x$. A detailed description of this figure is given in the text.

引用 : Bishop (2006), Pattern Recognition and Machine Learning

5.6.1 Maximum a Posteriori (MAP) Estimation

MAP 推定とは

ベイズではパラメータの事後分布 $p(\boldsymbol{\theta}|\mathbf{x})$ を得るが、点推定で单一の値を得たい時もある。

例えば、予測での積分 $p(x'|\mathbf{x}) = \int_{\Theta} p(x'|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x})d\boldsymbol{\theta}$ の計算が重いときなど。

自然に考えれば、事後確率が最大 (maximum a posteriori) になるパラメータを選ぶのが合理的で、それを **MAP 推定量** と呼ぶ。

$$\begin{aligned}\boldsymbol{\theta}_{\text{MAP}} &= \arg \max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathbf{x}) \\ &= \arg \max_{\boldsymbol{\theta}} p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \\ &= \arg \max_{\boldsymbol{\theta}} [\log p(\mathbf{x}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})]\end{aligned}$$

この $\arg \max_{\boldsymbol{\theta}} [\text{対数尤度} + \text{対数事前確率}]$ という形はベイズの考え方をよく表している。

主観を表している事前確率を抜いて $\arg \max_{\boldsymbol{\theta}} [\text{対数尤度}]$ とすると、これは最尤推定である。

ベイズ線形回帰で MAP 推定

先ほど扱ったベイズ線形回帰について、事前分布を

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \lambda^{-1} \mathbf{I})$$

のように設定し直す。これは「回帰係数は絶対値の大きい極端な値を取りづらい」というような事前の信念(知識、情報)を表していることになる。

ここで、先ほどの (5.72), (5.73) より、事後分布は

$$\begin{aligned}p(\mathbf{w}|\mathbf{X}, \mathbf{y}) &\propto \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})\right) \exp\left(-\frac{1}{2}\mathbf{w}^T(\lambda^{-1} \mathbf{I})^{-1} \mathbf{w}\right) \\ &= \exp\left(-\frac{1}{2}((\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w})\right)\end{aligned}$$

となるので、MAP 推定量は、

$$\mathbf{w}_{\text{MAP}} = \arg \min_{\mathbf{w}} [(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}]$$

で得ることができる。

これは、荷重減衰 $\lambda \mathbf{w}^T \mathbf{w}$ を使った正則化最尤推定 (Ridge 推定) と一致している。

すなわち、ベイズにおける「絶対値の大きい極端な値を取りづらい」という事前の信念(知識、情報)が正則化の効果をもたらしている。

5.7 Supervised Learning Algorithms

アプローチによる分類

教師あり分類アルゴリズムは、アプローチによって大きく3つに分けられる（回帰や教師なし分類でも同様）。

- 識別関数： $\hat{y} = f(\mathbf{x})$ のように入力 \mathbf{x} からクラス y を直接予測する関数 f を学習。
 - サポートベクターマシン (5.7.2)
- 識別モデル：各クラスに属する確率を分布 $p(y|\mathbf{x})$ でモデル化し、学習。
 - ロジスティック回帰 (5.7.1)
 - k 近傍分類、決定木 (5.7.3)
 - 最終レイヤーにソフトマックスを使ったニューラルネット
- 生成モデル：入力とクラス全体を同時分布 $p(\mathbf{x}, y)$ でモデル化し、学習。
 - ナイーブベイズ

ただ、同じアルゴリズムでも、捉え方や仮定によってどこに属するかは変わる。

5.7.1 Probabilistic Supervised Learning

ロジスティック回帰

線形回帰をロジスティックシグモイド関数に入れて、出力を \mathbb{R} から $(0, 1)$ に押し込む。

それを使って、クラス1に属する確率 $p(y = 1|\mathbf{x})$ をモデル化する。

$$\begin{aligned} p(y = 1|\mathbf{x}; \boldsymbol{\theta}) &= \sigma(\boldsymbol{\theta}^T \mathbf{x}) \\ &= \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})} \end{aligned}$$

パラメータ $\boldsymbol{\theta}$ の推定

2クラス分類なら、 y にベルヌーイ分布を仮定して最尤推定すればよい。

ただ、線形回帰とは違って、最尤推定量を解析的に求めることができない。

よって、一般的なニューラルネットと同様に勾配降下法でパラメータ推定を行う。

NN と違って勾配やヘッセ行列などの算出が比較的容易なので、SGD 等ではなく Newton 法系統のアルゴリズムを使うことが多い。

5.7.2 Support Vector Machines

線形 SVM

入力空間 \mathbb{R}^p を 2 分割する超平面 $\mathbf{w}^T \mathbf{x} + b = 0$ を学習し、2 クラス分類を実現したい。

$$\hat{y} = \begin{cases} \text{Class 1} & \mathbf{w}^T \mathbf{x} + b \geq 0 \\ \text{Class 2} & \mathbf{w}^T \mathbf{x} + b < 0 \end{cases}$$

マージン最大化の考え方で \mathbf{w}, b を推定すると、

$$\hat{\mathbf{w}}^T \mathbf{x} + \hat{b} = \hat{b} + \sum_{i=1}^m \alpha_i \mathbf{x}^T \mathbf{x}^{(i)} \quad (5.82)$$

のような形となる。この導出や $\hat{\mathbf{w}}, \hat{b}, \alpha_i$ の中身は本書では省略されている。

非線形 SVM：高次元特徴空間への写像

線形 SVM では線形の決定領域しか作れないので、精度（表現力）があまり良くない。

これをどうにかするために、

1. 入力 $\mathbf{x} \in \mathbb{R}^p$ を非線型関数 $\phi : \mathbb{R}^p \rightarrow \mathbb{R}^r$ によって高次元の特徴空間に \mathbb{R}^r に写像する。
2. その特徴空間上で線形 SVM を実行し、分離超平面を構築する。
3. すると、もとの入力空間には非線形の決定領域が作られる。

という方針をとる。

入力を $\phi()$ で変換した特徴ベクトル $\phi(\mathbf{x}^{(i)})$ に対して線形 SVM を実行すれば良いので、(5.82) より、次の関数の正負を見てクラス分類を行えば良い。（ b, α の内容は省略）

$$f(\mathbf{x}) = b + \sum_{i=1}^m \alpha_i \phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)}) \quad (\text{i})$$

ただ、精度向上のために極めて高次元な特徴空間に写像した場合、巨大なベクトルの内積 $\phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)})$ の計算が困難になる。（ α の推定にも同じ困難が伴う）

非線形 SVM：カーネルトリック

問題の内積 $\phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)})$ は、カーネル関数 $k(\mathbf{x}, \mathbf{x}^{(i)})$ と呼ばれる $\mathbf{x}, \mathbf{x}^{(i)}$ の直接の関数で表現できることが知られている（Mercer の定理）。

$$\phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)}) = k(\phi(\mathbf{x}), \phi(\mathbf{x}^{(i)}))$$

(i) の内積をカーネルで置き換える(これをカーネルトリックと言う)と,

$$f(\mathbf{x}) = b + \sum_{i=1}^m \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)}) \quad (\text{ii})$$

となり、巨大な内積 $\phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)})$ が消えて計算が軽くなる。ここでは触れないが、 α の推定のための最適化もかなり効率が良くなる。

どのカーネルを使うか

まとめると、カーネルトリックをして線形 SVM を実行すれば、入力を高次元に写像した特徴空間上に分離超平面を作ったことになり、結果として \mathbf{x} の非線型モデルが得られる。

なので、使うカーネル関数の種類やそのパラメータは、ハイパーパラメータである。

代表的なものを挙げると、

- RBF カーネル (動経基底関数カーネル)

$$k(\mathbf{u}, \mathbf{v}) = \mathcal{N}(\mathbf{u} - \mathbf{v}; \mathbf{0}, \sigma^2 \mathbf{I})$$

-

$$k(\mathbf{u}, \mathbf{v}) = (\mathbf{u}^T \mathbf{v} + c)^d$$

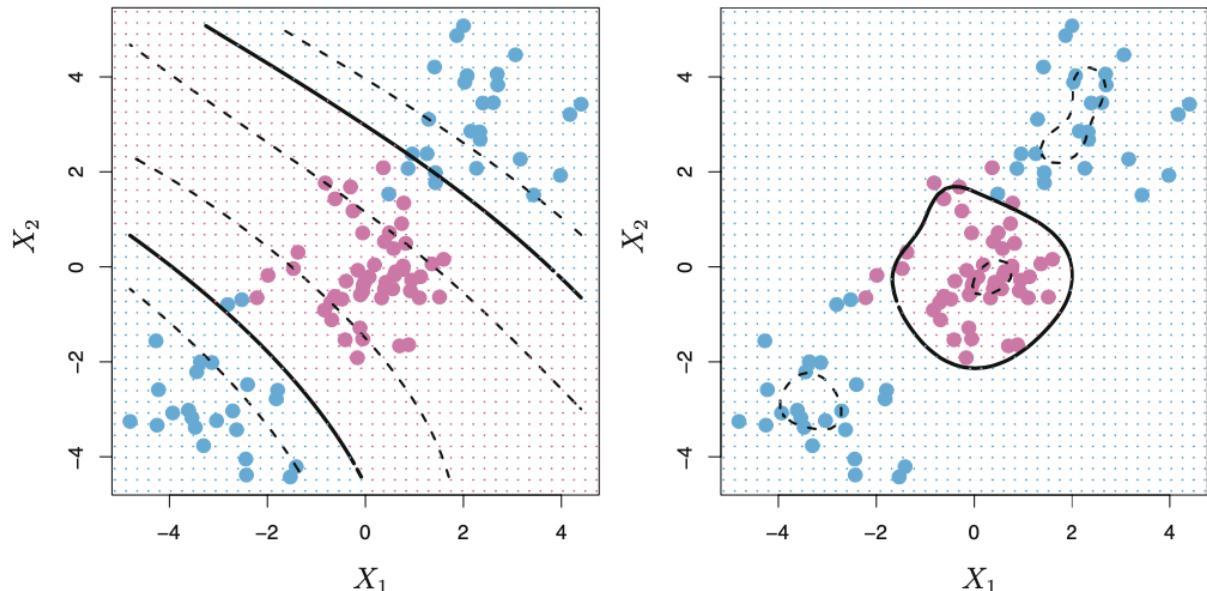


FIGURE 9.9. Left: An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule. Right: An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.

引用 : James et al. (2013), An Introduction to Statistical Learning

ビッグデータに対する計算コストの問題

- 予測式に $\sum_{i=1}^m \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)})$ が含まれているので、スコアの計算コストが訓練データのサイズ m に線形になってしまう。これについては、訓練データのうちある条件を満たす一部のケース（サポートベクトルと呼ぶ）についてのみ $\alpha_i k(\mathbf{x}, \mathbf{x}^{(i)})$ を計算すれば良いことが知られている。
- SVM をはじめとする多くのカーネル法は、学習の際に $m \times m$ 行列 $G = \{k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})\}$ が必要となり、これには $O(m^2)$ の計算量がかかってしまう。

SVM とディープラーニング

p 141 の本文

The current deep learning renaissance began when Hinton et al. (2006) demonstrated that a neural network could outperform the RBF kernel SVM on the MNIST benchmark.

について、[Hinton et al. \(2006\)](#) の論文で次のような実験結果が報告されていた。

(RBF kernel ではなく polynomial kernel ?)

Table 1: Error rates of Various Learning Algorithms on the MNIST Digit Recognition Task.

Version of MNIST Task	Learning Algorithm	Test Error %
Permutation invariant	Our generative model: 784 → 500 → 500 ↔ 2000 ↔ 10	1.25
Permutation invariant	Support vector machine: degree 9 polynomial kernel	1.4
Permutation invariant	Backprop: 784 → 500 → 300 → 10 cross-entropy and weight-decay	1.51
Permutation invariant	Backprop: 784 → 800 → 10 cross-entropy and early stopping	1.53
Permutation invariant	Backprop: 784 → 500 → 150 → 10 squared error and on-line updates	2.95
Permutation invariant	Nearest neighbor: all 60,000 examples and L3 norm	2.8
Permutation invariant	Nearest neighbor: all 60,000 examples and L2 norm	3.1
Permutation invariant	Nearest neighbor: 20,000 examples and L3 norm	4.0
Permutation invariant	Nearest neighbor: 20,000 examples and L2 norm	4.4
Unpermuted images; extra data from elastic deformations	Backprop: cross-entropy and early-stopping convolutional neural net	0.4
Unpermuted de-skewed images; extra data from 2 pixel translations	Virtual SVM: degree 9 polynomial kernel	0.56
Unpermuted images	Shape-context features: hand-coded matching	0.63
Unpermuted images; extra data from affine transformations	Backprop in LeNet5: convolutional neural net	0.8
Unpermuted images	Backprop in LeNet5: convolutional neural net	0.95

引用 : [Hinton et al. \(2006\)](#)

5.7.3 Other Simple Supervised Learning Algorithms

k 近傍法 (k - nearest neighbor : k - NN)

5.2 で扱った最近傍法を拡張した手法。新たな入力 \mathbf{x} に対応する \mathbf{y} を次の手順で予測する。

1. 訓練データから、 \mathbf{x} との距離が近い k 個のケースをピックアップする。
2. その k 個のアウトカム $\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(m)}$ の平均を取り、予測値として出力する。

分類の時は、 \mathbf{y} を所属クラスの one-hot encoding としてやれば良い。

ハイパーパラメータ k は決定境界の滑らかさ (モデルの複雑さ) をコントロールする.

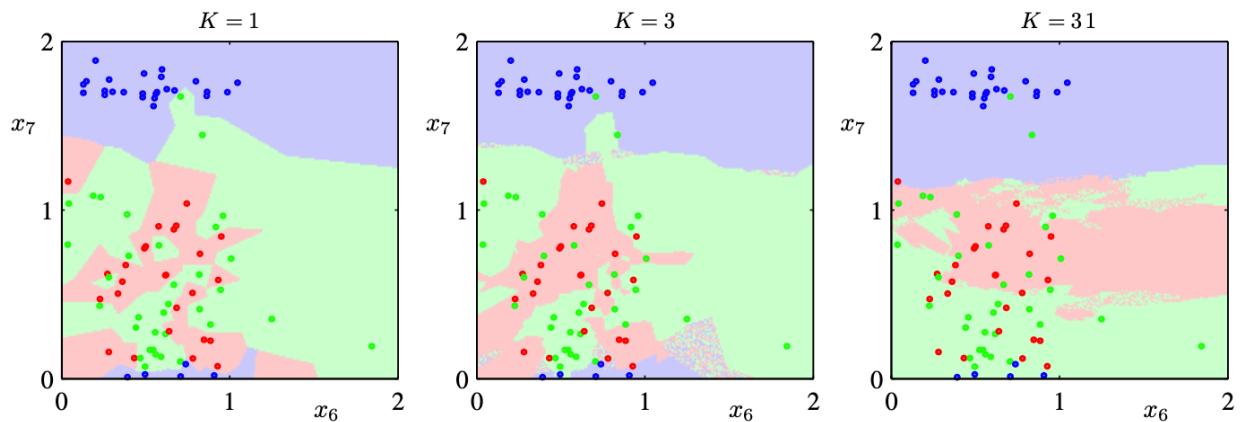


Figure 2.28 Plot of 200 data points from the oil data set showing values of x_6 plotted against x_7 , where the red, green, and blue points correspond to the ‘laminar’, ‘annular’, and ‘homogeneous’ classes, respectively. Also shown are the classifications of the input space given by the K -nearest-neighbour algorithm for various values of K .

引用 : Bishop (2006), Pattern Recognition and Machine Learning

k 近傍法の特徴

メリットは,

- capacity (表現力) が非常に高い.
- $n \rightarrow \infty$ のとき, 最近傍分類モデルの誤分類率はベイズ誤差 (真のクラス分布を用いた最適な分類モデルの誤分類率) のたかだか 2 倍にしかならない.

デメリットは,

- 訓練データ集合が大きくなるにつれ, 予測時の計算コストが高くなる.
- 訓練データ集合が小さいと, 汎化性能が悪くなる傾向がある.

決定木

入力空間を再帰的に分割していくことで, 決定境界を構築する.

1 つの分割領域を 1 つのノードに対応させ, ツリー構造でモデルを表現できる. 元の入力空間がルートに対応し, 最終的な決定領域がリーフに対応する.

分割方法 (どの変数でどの閾値で分割するか) を学習するアルゴリズムは色々あり, 本書では省略している. 基本的に impurity (クロスエントロピーやGini係数) の低下を目的として最適化を繰り返す.

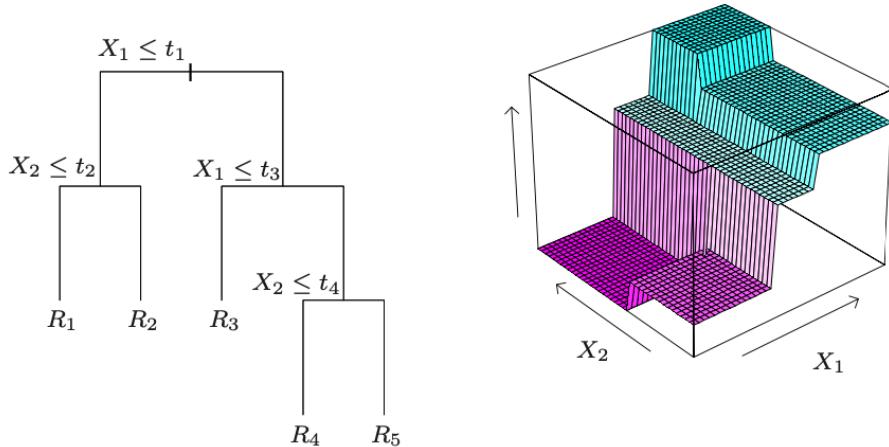
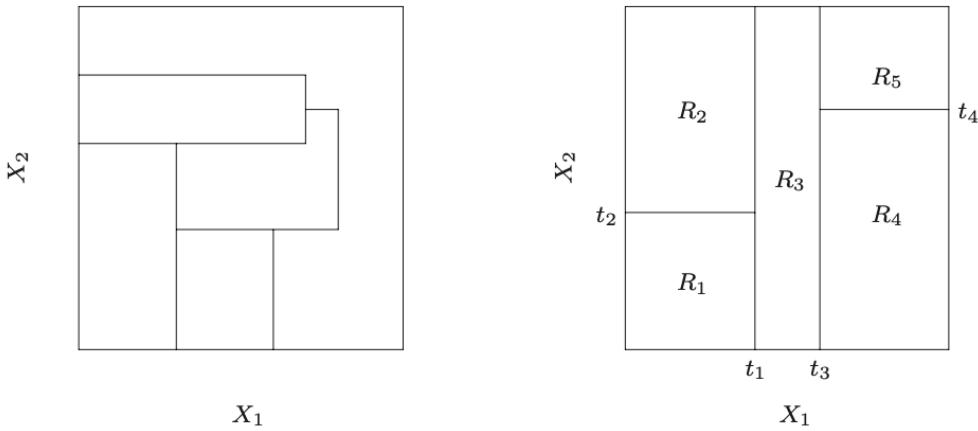


FIGURE 9.2. Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.

引用 : Hastie et al. (2009), The Elements of Statistical Learning

決定木の特徴

メリットは,

- GBDT (LightGBM や XGBoost) やランダムフォレストなどのアンサンブルモデルを構成する弱学習器として使える。
- モデルの解釈がしやすいので、意思決定の支援には使いやすい。

デメリットは,

- 単体での性能は悪い。軸に沿った分割しか行ないので、下図のような線形の決定境界を作るのに一苦労。ロジスティック等のシンプルなモデルでも簡単なのに。

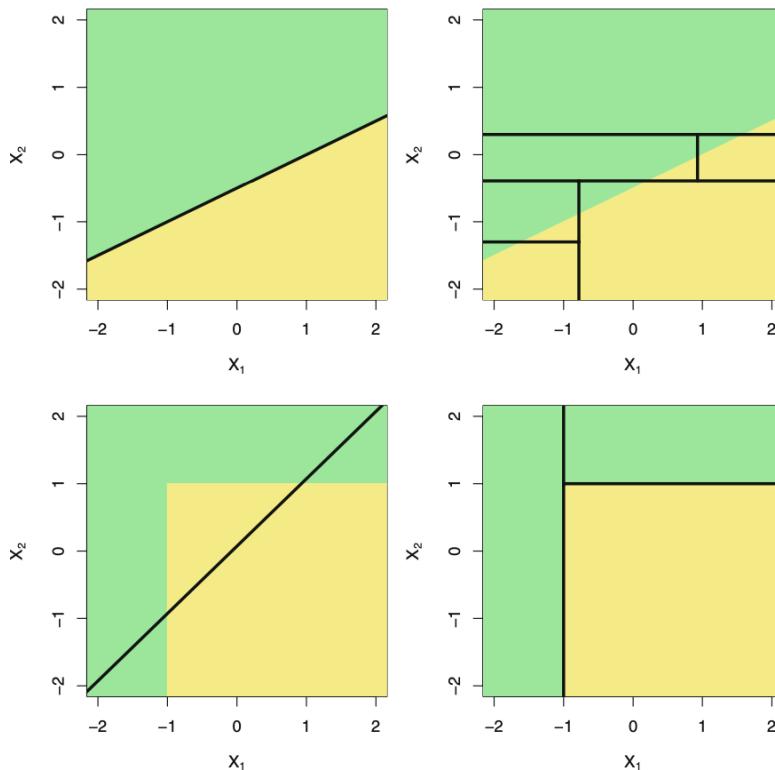


FIGURE 8.7. Top Row: A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right). Bottom Row: Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).

引用 : James et al. (2013), An Introduction to Statistical Learning

5.8 Unsupervised Learning Algorithms

データの表現

教師なし学習のタスクは多岐にわたり厳密な定義も無いが、基本的かつ深層学習との関連が強いのは、データの最良の表現を探すこと。

「良い表現」は「多くの情報を保持したシンプル(解釈しやすい)な表現」と捉えられる。

シンプルさの基準と、それを達成する具体例を挙げると…

- 低次元での表現

- PCA, 因子分析, NMF

- 跡な表現
 - k-means クラスタリング, NMF
- 独立した表現
 - PCA

この3つは排反ではなく、かぶることも多い。例えばPCAは「低次元」と「独立」を満たすし、NMF(非負値行列因子分解)は「低次元」と「跡」を満たす。

5.8.1 Principal Components Analysis

主成分分析 PCA とは (2.12 節で既出)

PCAでは、データ \mathbf{x} を低次元の線形空間に射影して表現 \mathbf{z} を得る。

その低次元空間の軸(正規直交基底)である主成分ベクトルの学習は、2つの異なるアプローチから定式化できる。(結果は同じ)

- 射影されたデータの分散が最大化されるような軸を抽出し、それを第1主成分とする。次に、それと直交するという制約を課した上で射影後の分散が最大化される軸を抽出し、第2主成分を抽出する。これを繰り返す。参考:Bishop (2006)
- 元のデータ点 \mathbf{x} と射影後の点の距離(すなわち次元圧縮による情報損失)が小さくなるように、軸(主成分ベクトル)を抽出する。参考:2.12節

結果として、第 i 主成分ベクトルは、データの分散共分散行列の i 番目に大きい固有値に対応する固有ベクトルとして得られることが分かる。

特異値分解で考える

ここで \mathbf{X} を中心化済み計画行列とすると、主成分ベクトルは $\mathbf{X}^T \mathbf{X}$ の固有ベクトルだが、これは \mathbf{X} を $\mathbf{X} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{W}^T$ と特異値分解した時の右特異ベクトル $\mathbf{w}_{:,i}$ とも捉えられる。

よって、 $\mathbf{z} = \mathbf{x}^T \mathbf{W}$ で表現 \mathbf{z} を求めることができ、その不偏分散共分散行列は、

$$\begin{aligned}
(m-1) \operatorname{Var}[\mathbf{z}] &= \mathbf{Z}^T \mathbf{Z} \\
&= (\mathbf{XW})^T (\mathbf{XW}) \\
&= \mathbf{W}^T \mathbf{X}^T \mathbf{XW} \\
&= \mathbf{W}^T (\mathbf{U}\Sigma\mathbf{W}^T)^T (\mathbf{U}\Sigma\mathbf{W}^T)\mathbf{W} \\
&= \mathbf{W}^T \mathbf{W}\Sigma\mathbf{U}^T \mathbf{U}\Sigma\mathbf{W}^T \mathbf{W} \\
&= \mathbf{I}\Sigma\mathbf{I}\Sigma\mathbf{I} \\
&= \Sigma^2
\end{aligned}$$

となる。 Σ は特異値が並んだ対角行列なので、表現 \mathbf{z} が無相関であることを示している。

どのような「良い表現」を得られたか

- 表現 \mathbf{z} は \mathbf{x} を次元圧縮したもので、低次元。
- 表現 \mathbf{z} は（独立とまではいかないが）無相関。

fdsa

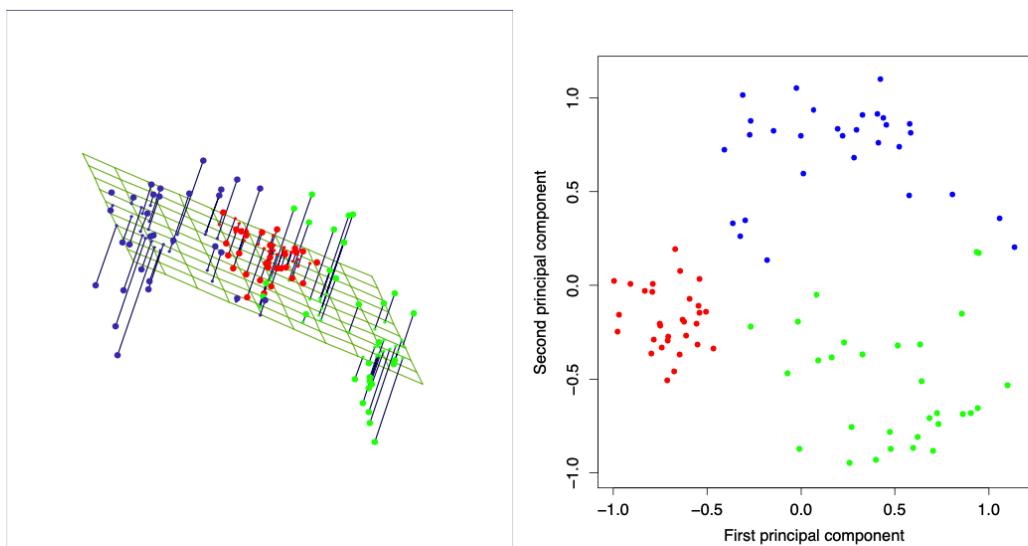


FIGURE 14.21. The best rank-two linear approximation to the half-sphere data. The right panel shows the projected points with coordinates given by $\mathbf{U}_2\mathbf{D}_2$, the first two principal components of the data.

引用 : Hastie et al. (2009), The Elements of Statistical Learning

5.8.2 k -means Clustering

k -means クラスタリングとは

代表的なクラスタリング手法。次のアルゴリズムで k 個のクラスタに分類する。

0. 各クラスタの重心 $\{\mu^{(1)}, \dots, \mu^{(k)}\}$ をランダムに初期化する。
1. 訓練データの各ケースを、重心との距離が最も近いクラスタに割り当てる。
2. 重心 $\{\mu^{(1)}, \dots, \mu^{(k)}\}$ を各クラスタに割り当てられたケース $x^{(i)}$ の平均に更新する。
3. 収束するまで 1,2 を繰り返す。

(このアルゴリズムは、潜在変数を導入して生成モデルとして解釈し、最尤推定を EM アルゴリズムでやろうとすることで導かれる。)

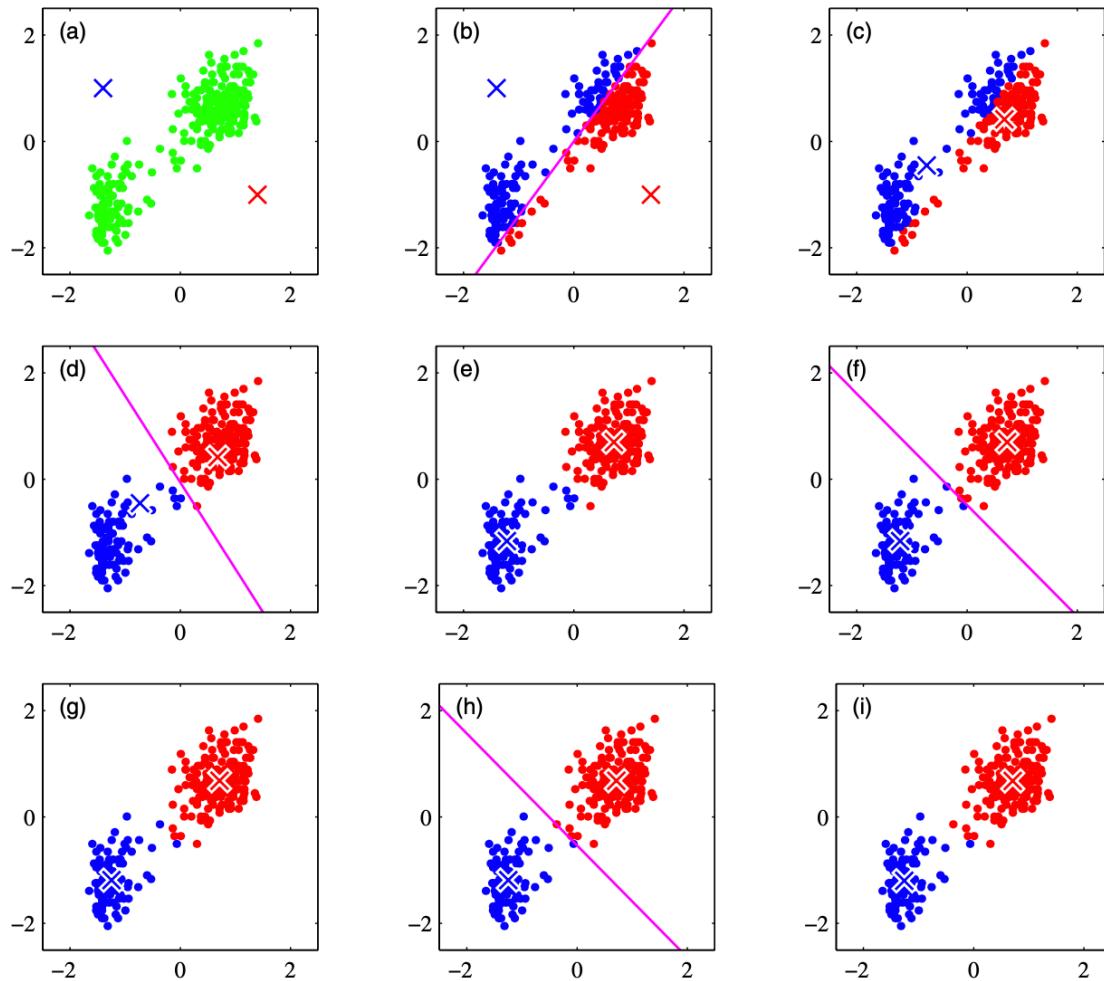


Figure 9.1 Illustration of the K -means algorithm using the re-scaled Old Faithful data set. (a) Green points denote the data set in a two-dimensional Euclidean space. The initial choices for centres μ_1 and μ_2 are shown by the red and blue crosses, respectively. (b) In the initial E step, each data point is assigned either to the red cluster or to the blue cluster, according to which cluster centre is nearer. This is equivalent to classifying the points according to which side of the perpendicular bisector of the two cluster centres, shown by the magenta line, they lie on. (c) In the subsequent M step, each cluster centre is re-computed to be the mean of the points assigned to the corresponding cluster. (d)–(i) show successive E and M steps through to final convergence of the algorithm.

引用 : Bishop (2006), Pattern Recognition and Machine Learning

どのような「良い表現」を得られたか

入力 \mathbf{x} に割り当てたクラスタを k 次元ベクトル \mathbf{h} に one-hot encoding で表せば,

- \mathbf{h} は \mathbf{x} の疎な表現となっている.
- 各クラスタ内のケースを同一視し \mathbf{x} の情報を捨てれば、低次元表現を得たことになる。

ただ、諸々の観点から one-hot 表現よりも **分散表現**の方が望ましく、詳細は 15.4 節。

(イメージとしては、 k^n 個の通りを表現するのに、one-hot 表現では k^n 次元が必要だが、分散表現ではそれぞれ k 通りの値をとりうる n 次元ベクトルで十分。)

5.9 Stochastic Gradient Descent

確率的勾配降下法 SGD とは (詳細は 8 章)

機械学習で最適化するコスト関数の多くは、訓練データ中の各ケースの和に分解できる。

$$Loss(\mathbf{X}, \mathbf{Y}, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \boldsymbol{\theta})$$

例えば最尤法では、目的関数を個々のケースの対数尤度の和に分解できる。

また、DNN では $L()$ に

- 二乗誤差 $(\hat{\mathbf{y}} - \mathbf{y})^T (\hat{\mathbf{y}} - \mathbf{y})$
- クロスエントロピー $-\sum_{k=1}^K y_k \log \hat{y}_k$

などを使う。

したがって、勾配降下法 (4.3 で既出) を実行する場合、毎回のステップで

$$\nabla_{\boldsymbol{\theta}} Loss(\mathbf{X}, \mathbf{Y}, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \boldsymbol{\theta})$$

を算出する必要があり、これには $O(m)$ の計算量がかかるので、ビッグデータでは辛い。

そこで、SGD では次のように最適化を進める。

1. 訓練データ集合からランダムに一部のケースをサンプリングする。これをミニバッチ $\mathbb{B} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \dots, (\mathbf{x}^{(m')}, \mathbf{y}^{(m')})\}$ と呼ぶ。
2. このミニバッチだけから勾配を算出する。

$$\mathbf{g} = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \boldsymbol{\theta})$$

3.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon \mathbf{g}$$

4. 設定した更新回数に達するか収束するまで、1～3を繰り返す。

SGD のメリット

- 訓練データのサイズ m がどんなに大きくなっても、ミニバッチのサイズ m' を固定しておけば、計算量は増加しない。かつ、十分に適合させることができる。
- 通常上の勾配降下法(最急降下法)と比べ、局所最小解にハマりづらい。
- Momentum, AdaGrad, 近年メジャーな Adam など、様々な optimizer の基礎となる。また、SGD 自体も最近でも結構使われる。

5.10 Building a Machine Learning Algorithm

機械学習アルゴリズムの構成

DNN を含む大半の機械学習アルゴリズムは、

- データの仕様
- モデル
- コスト関数
- 最適化手法

という4つの構成要素の組み合わせで捉えられる。

具体例

線形回帰モデルは,

- モデル : 入力の線形結合
- コスト関数 : 対数尤度
- 最適化手法 : 解析的に解ける

ロジスティック回帰は,

- モデル : 入力の線形結合をシグモイド関数に入れたもの
- コスト関数 : 対数尤度
- 最適化手法 : BFGS などニュートン法系統のもの

ディープラーニングなら, 例えば,

- モデル : ニューラルネット (Affine - ReLU - Affine - ReLU - ... - Affine - Softmax)
- コスト関数 : クロスエントロピー
- 最適化手法 : Adam

5.11 Challenges Motivating Deep Learning

深層学習のモチベーションとなつたいくつかの課題

ここまで挙げた従来の機械学習アルゴリズムには深刻な課題がいくつかあり, 5.11.1~5.11.3 でそれに触れる.

これらの課題は深層学習のモチベーションとなっている.

どのように克服を試みているかは, 6 章以降で詳しく説明される.

5.11.1 The Curse of Dimensionality

次元の呪いとは

データの次元が高くなることによって、諸々の問題が生じる。

- 入力空間において、存在しうる構成(状態、各変数値の組み合わせ)の総数が、指数関数的に増加する。イメージとしては、入力空間に訓練データを散りばめたとき、次元が高くなるにつれてスカスカ(疎)な感じになる。なので、まともな予想をするために必要な訓練データの数も、爆発的に増加していく。
- しばしば、機械学習アルゴリズムの中の最適化問題の計算量が増加し、困難になる。

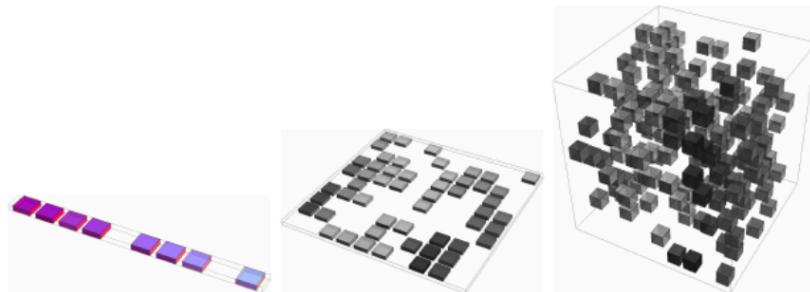


Figure 5.9: As the number of relevant dimensions of the data increases (from left to right), the number of configurations of interest may grow exponentially. (*Left*) In this one-dimensional example, we have one variable for which we only care to distinguish 10 regions of interest. With enough examples falling within each of these regions (each region corresponds to a cell in the illustration), learning algorithms can easily generalize correctly. A straightforward way to generalize is to estimate the value of the target function within each region (and possibly interpolate between neighboring regions). (*Center*) With two dimensions, it is more difficult to distinguish 10 different values of each variable. We need to keep track of up to $10 \times 10 = 100$ regions, and we need at least that many examples to cover all those regions. (*Right*) With three dimensions, this grows to $10^3 = 1,000$ regions and at least that many examples. For d dimensions and v values to be distinguished along each axis, we seem to need $O(v^d)$ regions and examples. This is an instance of the curse of dimensionality. Figure graciously provided by Nicolas Chapados.

引用(上) : Goodfellow et al. (2016), Deep Learning

5.11.2 Local Constancy and Smoothness Regularization

暗黙的な事前分布(事前の信念)

ベイズモデリングでは、明示的にパラメータの事前分布を用意し、ユーザーが事前に持っている信念(知識、情報)を機械学習アルゴリズムに反映させた。

これをしなくても、ある機械学習アルゴリズムを選択 (SVM じゃなくて DNN しよう等) している時点で、ある意味、事前の信念 (知識、情報) が暗黙的に表現されている。

碎いて言うと「ML を適用する際には何らかの仮定を暗黙的に置いている」ということで、本書はこれを「暗黙的な事前分布」と呼んでいる。

局所一様事前分布 (平滑化事前分布)

最も広く使われている暗黙的な事前分布 (ML 適用時に暗黙的に置いている仮定のうち一般的なもの) として、**局所一様事前分布 (平滑化事前分布)** がある。これは、

- 入力 \mathbf{x} のアウトカムが y であれば、おそらく \mathbf{x} の近傍のアウトカムも y であろう
- 学習する関数の出力は、局所的には (= 小さい領域の中では) あまり変化すべきではない

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \varepsilon)$$

といった信念 (知識、仮定) のことを指す。

具体例と課題

- 局所一様事前分布の仮定が極端に強く置かれている例は、 k 近傍法である。特に、 $k = 1$ とした最近傍法ではボロノイ図が作られることになるので、わかりやすい。
- 決定木にも、局所一様事前分布の仮定がわかりやすく表れている。

アルゴリズムから明らかなように、最近傍法と決定木には「識別可能な領域の数 (つまり決定領域の分割数) は訓練データのサイズを超えることができない」という限界がある。

その他の多くの機械学習アルゴリズムにも、多かれ少なかれ局所一様の仮定が置かれていて、同じような限界がある。イメージとしては、本文の checkerboard (チェスのボードみたいなもの) の話を考えると理解しやすい。

深層学習への動機付け

したがって「表現したい関数が複雑な (訓練データサイズと比較しても識別したい領域が多い) 時も、効率的に学習でき汎化もする」ような機械学習アルゴリズムが欲しくなる。

これが深層学習の 1 つのモチベーションになっていて、実際に大きな進展がなされている。(6.4.1, 15.4, 15.5 節で詳しく説明がされる。)

また、Bengio 先生のいくつかの論文で、このあたりの主要な結果が出ている。

5.11.3 Manifold Learning

多様体とは

数学的に厳密な定義があるが、機械学習では「高次元空間に埋め込まれている実質的に低次元の曲がった空間」という緩い意味で使われる。

2次元に埋め込まれている1次元多様体と、3次元に埋め込まれている2次元多様体の例。

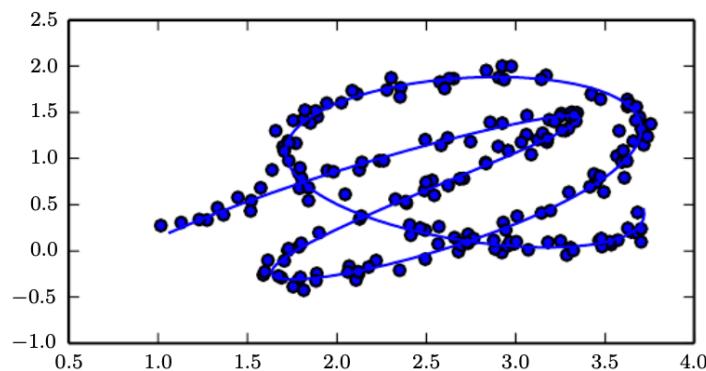


Figure 5.11: Data sampled from a distribution in a two-dimensional space that is actually concentrated near a one-dimensional manifold, like a twisted string. The solid line indicates the underlying manifold that the learner should infer.

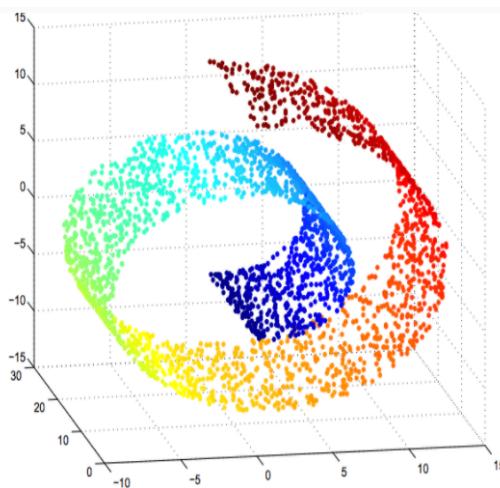


Figure 1: A curled plane: the swiss roll.

引用(上) : Goodfellow et al. (2016), Deep Learning

引用(下) : Verma, [Manifold Learning](#)

多様体仮説

現実世界で生じる画像・音声・自然言語などは、低次元の多様体に沿って存在していると考えられている。例えば下図の顔画像データセットは、

「画素数次元に埋め込まれた 2 次元 (水平方向と垂直方向の角度) の多様体上のデータ」

と捉えられる。追加で明るさが変化した場合は、3 次元多様体上に存在すると解釈できる。

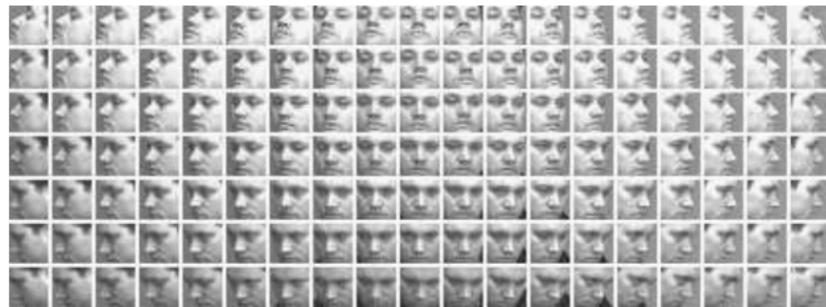


Figure 5.13: Training examples from the QMUL Multiview Face Dataset ([Gong et al., 2000](#)), for which the subjects were asked to move in such a way as to cover the two-dimensional manifold corresponding to two angles of rotation. We would like learning algorithms to be able to discover and disentangle such manifold coordinates. Figure 20.6 illustrates such a feat.

引用 : Goodfellow et al. (2016), Deep Learning

多様体表現を獲得する深層学習アルゴリズム

データから多様体構造を獲得できれば、大幅な次元削減によって次元の呪いを回避できる。

Goodfellow (2014) の GAN (generative adversarial networks) や Kingma (2013) の VAE (variational autoencoder) などの深層生成モデルは、データの低次元多様体をその潜在表現として獲得できることが分かっている。これらについては 20 章で詳しく扱われる。

参考 : [PFN の岡野原さんの記事](#)