最終更新日: 2024 年 1 月 22 日

「Python による数理・データサイエンス・AI」(初版) 正誤表

<table>
<tr><th></th><th>誤</th><th>正</th></tr>
<tr><td>p.iii</td><td>本書のキーワードはなるべくモデルカリキュラムのキーワードにあわせています．表 0.1 を見ると分かるように，本書は,「コア学修項目」と「基盤となる学修項目」はカバーしています.</td><td>本書のキーワードはなるべく「数理・データサイエンス・AI（応用基礎レベル）モデルカリキュラム～AI× データ活用の実践」 のキーワードにあわせています．表 0.1 を見ると分かるように，本書は,「コア学修項目」と「基盤となる学修項目」はカバーしています．そのため，本書は「数理・データサイエンス・AI 教育プログラム認定制度（応用基礎レベル）」に準拠しています.</td></tr>
<tr><td>p.iii, 表 0.1</td><td><table><tr><td>2-5. データ加工</td><td>集計処理，四則演算処理，ソート処理，サンプリング処理，データの標準化，ダミー変数，分散処理</td></tr></table></td><td><table><tr><td>2-5. データ加工</td><td>集計処理，四則演算処理，ソート処理，サンプリング処理，データの標準化，ダミー変数~~，分散処理~~</td></tr></table></td></tr>
<tr><td>p.v</td><td>そのような状況に遭遇した場合は，バージョンの更新を検討してください.</td><td>そのような状況に遭遇した場合は，バージョンの更新を検討してください．例えば，threadpoolctl を pip でアップグレードするには，プロンプトで下記の下線部を入力します.

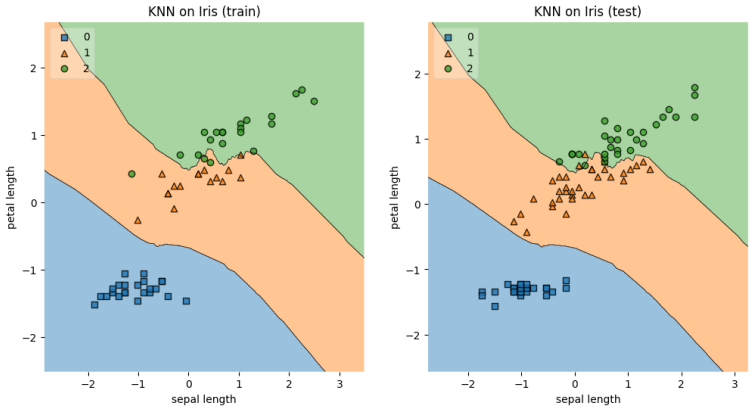
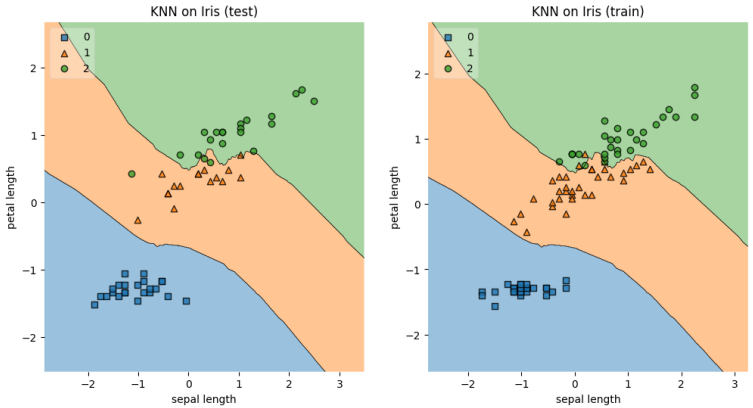
(base) pip install --upgrade threadpoolctl --user

また，バージョンを指定してインストールするには，次のように入力します.

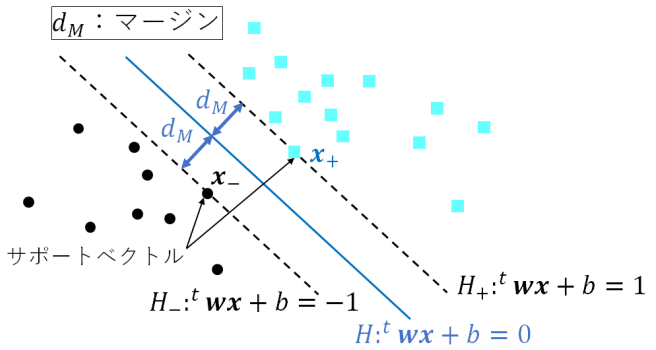
(base) pip install threadpoolctl==3.1.0</td></tr>
</table>

	誤	正
p.17, 補題 2.2 の証明	$\dfrac{\partial E}{\partial w_n} = \displaystyle\sum_{i=1}^{N}\left(\sum_{j=0}^{p} w_j x_i^j - y_i\right) x_i, \quad \dfrac{\partial^2 E}{\partial w_m \partial w_n} = \dfrac{\partial}{\partial w_m}\left(\dfrac{\partial E}{\partial w_n}\right) = \displaystyle\sum_{i=1}^{N} x_i^m x_i^n$	$\dfrac{\partial E}{\partial w_n} = \displaystyle\sum_{i=1}^{N}\left(\sum_{j=0}^{p} w_j x_i^j - y_i\right) x_i^n, \quad \dfrac{\partial^2 E}{\partial w_m \partial w_n} = \dfrac{\partial}{\partial w_m}\left(\dfrac{\partial E}{\partial w_n}\right) = \displaystyle\sum_{i=1}^{N} x_i^m x_i^n$
p.48, ソースコード 4.5 の 2 行目, Series を削除	`import pandas as pd # Pandas` の読み込み `from pandas import Series,DataFrame # Pandas` から `Series, DataFrame` を読み込む	`import pandas as pd # Pandas` の読み込み `from pandas import DataFrame # Pandas` から `DataFrame` を読み込む
pp.88-89	単語の出現回数（TF: Term Frequency）に全文書数に対するその単語が出現する文書数の割合（DF: Document Frequency）の逆数を掛けることで，単語の重みを調整します． Inverse Document Frequency（文書頻度の逆数）は次のように定義されます． $$\mathrm{idf}(t) = \log \frac{n}{1+\mathrm{df}(t)}$$ ここで $\mathrm{df}(t)$ はその単語が出現する文書数を表し，n は全文書数です． 大まかに表現すると，IDF はその単語が出現する文書数の全体に対する割合の逆数と見ることができます． $$\mathrm{idf}(t) \approx \frac{\text{すべての文書数}}{\text{単語 } t \text{ が出現する文書数}}$$ TF-IDF はこれらを掛け合わせたもので，その単語の出現回数を全文書におけるその単語の出現する文書数の割合で割ることで，頻出単語の重みを小さくするように調整します． $$\begin{aligned}\mathrm{TF-IDF} &= \text{Term Frequency} \times \text{Inverse Document Frequency} \\ &= \frac{\text{単語 } t \text{ の出現回数}}{\text{単語 } t \text{ の出現する文書数の全体に対する割合}}\end{aligned}$$	1 つの文書中における単語の出現頻度（TF: Term Frequency）に全文書におけるその単語の出現頻度（DF: Document Frequency）の逆数を掛けることで，単語の重みを調整します．文書 d における単語 t の出現回数を $n_{d(t)}$ とし， 全単語数を T とすると，TF は $TF_{d(t)} = \frac{n_{d(t)}}{\sum_{t=1}^{T} n_{d(t)}}$ と表せます．一方，Inverse Document Frequency（IDF:文書頻度の逆数）は，単語 t が出現する文書が全文書中に占める割合の逆数であり，単語 t が一部の文書にどれだけ現れるかを示します．具体的には，$df(t)$ を単語 t が出現する文書数とし，N を全文書数としたとき単語 t に対する IDF を $$IDF(t) = \log\left(\frac{N}{df(t)+1}\right)$$ と定義します．$IDF(t)$ が大きいほど，単語 t が出現する文書数 $df(t)$ は少なくなります．全文書数 N が大きくなるほど，$\frac{N}{df(t)}$ の値が大きくなりやすいため，log をとるとともに，分母に 1 を加えることによってゼロ除算により計算できなくなることを回避しています． TF-IDF はこれらを掛け合わせたもので，文書 d における単語 t の $TF-IDF_{d(t)}$ は，次のようになります． $$TF-IDF_{d(t)} = TF_{d(t)} \times IDF(t) \approx \frac{\text{文書 } d \text{ における単語 } t \text{ の出現頻度}}{\text{全文書における単語 } t \text{ の出現頻度}}$$ 対象とする文書 d に単語 t が頻繁に登場し，しかもその単語が文書全体ではあまり現れない（つまり，一部の文書にしか登場しない）場合，$TF-IDF_{d(t)}$ の値は大きくなります．結局のところ，$TF-IDF_{d(t)}$ は，単語 t が文書 d にとってどれだけ重要かを示す指標になっています．

	誤	正
p.90	**特異率 (Specificity)** 実際のクラスが Negative のうち，Negative だと予測できた割合 (SPE)．特異度は,「陰性」の予測における見落としをなるべく避けたい，別の言い方をすれば，偽陽性 (FP) を減らすことを重視する場合に使う．	**特異率 (Specificity)** 実際のクラスが Negative のうち，Negative だと予測できた割合 (SPE)．特異率は，特異度とも呼ばれる．特異度は,「陰性」の予測における見落としをなるべく避けたい，別の言い方をすれば，偽陽性 (FP) を減らすことを重視する場合に使う．
p.91	PRE と REC の最適化の長所と短所のバランスをとったものが F1 スコアです．適合率を最適化しようとすると，FP を下げることになりますが，これは FN の増加を招きます．一方，再現率を最適化しようとすると，FN を下げることになりますが，これは FP の増加を招きます．結局，FP と FN は同時に小さくすることはできず，トレードオフの関係にあります．	PRE と REC の~~最適化の長所と短所の~~バランスをとったものが F1 スコアです．適合率を上げようとすると，FP を下げることになります．FP が減るということは，Negative と判断される割合が増えるということですから，これは FN の増加を招きます．一方，再現率を上げようとすると，FN を下げることになりますが，これは FP の増加を招きます．結局，FP と FN は同時に小さくすることはできず，トレードオフの関係にあります．そのため，F 値のような指標が考えられました．
p.95	`#` 混同行列から特異性を求める関数	`#` 混同行列から特異率を求める関数
p.102	$w_{ik} = \begin{cases} 1 & (\boldsymbol{x}_i \text{が } k \text{ 番目のセントロイドのクラスターに属する}) \\ 0 & (\text{それ以外}) \end{cases}$	$w_{ik} = \begin{cases} 1 & (\boldsymbol{x}_i \text{がセントロイド } \boldsymbol{\mu}_k \text{のクラスターに属する}) \\ 0 & (\text{それ以外}) \end{cases}$
p.104	ここで，$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{in} \end{bmatrix}$，$\boldsymbol{\mu}_k = \begin{bmatrix} \mu_{k1} \\ \vdots \\ \mu_{kn} \end{bmatrix}$ とすれば，式 (8.1) は以下のように表せます．	ここで，$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{in} \end{bmatrix}$，$\boldsymbol{\mu}_k = \begin{bmatrix} \mu_{k1} \\ \vdots \\ \mu_{kn} \end{bmatrix}$，$\|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|^2 = \sum_{j=1}^{n} (x_{ij} - \mu_{kj})^2$ とすれば，式 (8.1) は以下のように表せます．

	誤	正
p.104, ややくどいので表現を変更.	また，この式は $\boldsymbol{\mu}_k$ は k 番目のクラスターに割り当てられたすべてのデータ点 $\boldsymbol{x}_n$ の平均値とおいているものと単純に解釈することができます．k-means(k 平均) アルゴリズムという名の由来はここにあります.	また，この式は $\boldsymbol{\mu}_k$ は k 番目のクラスターに割り当てられたすべてのデータ点~~$\boldsymbol{x}_n$~~の平均値であると単純に解釈することができます．k-means(k 平均) アルゴリズムという名の由来はここにあります.
p.109, ソースコード 8.4, 22 行目	`ax1.set_title('KNN on Iris (train)')`	`ax1.set_title('KNN on Iris (test)')`
p.110, 実行例，図の位置はそのままだが,「test」と「train」を入れ替える.		accracy_score による正解率 (train):0.9667 accracy_score による正解率 (test):0.9333 

	誤	正
p.110	ここでは，`init='k-means++'` を明示的に指定します．なお，セントロイドの位置は，`cluster_centers_`属性で取得できます．	ここでは，`init='k-means++'` を明示的に指定します．`n_init` は，異なる乱数のシードで初期セントロイドを選ぶ回数で，最終的には，最も良い結果が出力されます．デフォルト値は 10 ですが，ここではこれも明示しています．なお，セントロイドの位置は，`cluster_centers_`属性で取得できます．
p.111, ソースコード 8.5, 23 行目	`kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 0, n_init = 10) # 初期化`	`kmeans = KMeans(n_clusters = 5, init = 'k-means++', ~~random_state = 0,~~ n_init = 10) # 初期化`
p.112	1. 入力データからランダムに１つ選び，それを１つ目のセントロイドとする． 2. セントロイドが k 個選ばれるまで，以下の手順を繰り返す． (a) 各入力データと各セントロイドとの距離を求める． (b) 入力データ $\boldsymbol{x}_i$ ごとに，最も近いセントロイドとの距離 $d(\boldsymbol{x}_i)$ を求める． (c) 距離の２乗に比例する確率 $$p(\boldsymbol{x}_p) = \frac{d(\boldsymbol{x}_p)^2}{\sum_i d(\boldsymbol{x}_i)^2}$$ に従って，入力データの中からデータを１つ選択し，それをセントロイドとして採用する．手順 (c) のような選択方法はルーレット選択 (Roulette selection, Roulette wheel selection) と呼ばれる．	1. 入力データからランダムに１つ選び，それを１つ目のセントロイドとする． 2. セントロイドが k 個選ばれるまで，以下の手順を繰り返す． (a) 各入力データと各セントロイドとの距離を求める． (b) 入力データ $\boldsymbol{x}_i$ ごとに，最も近いセントロイドとの距離 $d(\boldsymbol{x}_i)$ を求める． (c) 入力データ $\boldsymbol{x}_p$ がセントロイドとして選ばれる確率（選択確率）を距離の２乗に比例する確率 $$p(\boldsymbol{x}_p) = \frac{d(\boldsymbol{x}_p)^2}{\sum_i d(\boldsymbol{x}_i)^2}$$ として求め，これに従って，入力データの中からデータを１つ選択し，それをセントロイドとして採用する．より具体的には，0 ～ 1 の乱数 r を生成し，各データの選択確率 $p(\boldsymbol{x}_i)$ を順に足していき，つまり，累積確率 $\sum_i p(\boldsymbol{x}_i)$ を考え，r がこの累積確率を超えた点をセントロイドとする． 手順 (c) のような選択方法はルーレット選択 (Roulette selection, Roulette wheel selection) と呼ばれる．データ $\boldsymbol{x}_p$ とセントロイドの距離が短いほど選択確率 $p(x_p)$ は低く，距離が長いほど高くなる．

	誤	正
p.128, ソース 9.7 の 86 行目	`print('累積寄与率(scikit-learn):'.format(【自分で補おう】)) )`	`print('累積寄与率(scikit-learn):'.format(【自分で補おう】))`
p.130, 図 10.1(a) を変更		
p.130, §10.1	p 次元ベクトル $\boldsymbol{x}={}^t[x_1,\ldots,x_p]$ と，同様に p 次元のパラメータベクトル $\boldsymbol{w}={}^t[w_1,\ldots,w_p]$，そしてスカラー b に対して，超平面は以下のように表現することができます． $${}^t\boldsymbol{w}\boldsymbol{x}+b=0 \quad (10.1)$$ ここでの目的は，マージン d_M を最大化する超平面を見つけ出すことです． マージン d_M は，サポートベクトルと決定境界（つまり超平面）との間の距離であるため，次の等式が成り立ちます．	p 次元ベクトル $\boldsymbol{x}={}^t[x_1,\ldots,x_p]$ と，同様に p 次元のパラメータベクトル $\boldsymbol{w}={}^t[w_1,\ldots,w_p]$，そしてスカラー b に対して，超平面 H は以下のように表現することができます． $${}^t\boldsymbol{w}\boldsymbol{x}+b=0 \quad (10.1)$$ ここでの目的は，マージン d_M を最大化する超平面を見つけ出すことです． マージン d_M は，サポートベクトル $\boldsymbol{x}_+, \boldsymbol{x}_-$ と決定境界（つまり超平面）との間の距離であるため，次の等式が成り立ちます．

	誤	正
p.132	式 (10.11), (10.12) を **KKT 条件** (KKT condition, arush-Kuhn-Tucker condition) と呼びます.	式 (10.11)∼(10.13) を **KKT 条件** (KKT condition, arush-Kuhn-Tucker condition) と呼びます.
p.132, (10.17), (10.18)	$\alpha_i \geq 0 \quad (i = 1, 2, \ldots, N)$ (10.17) $\alpha_i = 0 \quad$ または $\quad y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b) - 1 = 0$ (10.18)	$\alpha_n \geq 0 \quad (n = 1, 2, \ldots, N)$ (10.17) $\alpha_n = 0 \quad$ または $\quad y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b) - 1 = 0$ (10.18)
p.133 の式変形	$L(\boldsymbol{w}, b, \alpha_1, \ldots, \alpha_N) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \alpha_n \left\{1 - y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b)\right\}$ $\vdots$ $= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i\right)\left(\sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j\right)$	$L(\boldsymbol{w}, b, \alpha_1, \ldots, \alpha_N) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \alpha_n \left\{1 - y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b)\right\}$ $\vdots$ $= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}{}^t\left(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i\right)\left(\sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j\right)$
p.133, 論理にやや飛躍があるので説明を追加. 厳密には強双対定理が必要.	結局，以上の議論から，$\frac{1}{2}\|\boldsymbol{w}\|^2$ の最小化問題の代わりに，$\tilde{L}(\boldsymbol{\alpha})$ の最大化問題により最適解 $\hat{\boldsymbol{\alpha}}$ を求めれば，元の解 $\hat{\boldsymbol{w}}$ が得られることが分かります.	結局，以上の議論および $\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 = \min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha}\geq\boldsymbol{0}} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}\geq\boldsymbol{0}} \min_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}\geq\boldsymbol{0}} \tilde{L}(\boldsymbol{\alpha})$ から，$\frac{1}{2}\|\boldsymbol{w}\|^2$ の最小化問題の代わりに，$\boldsymbol{\alpha}$ に関する制約条件 $\alpha \geq 0$, $\sum_{n=1}^{N} \alpha_n y_n = 0$ の下で $\tilde{L}(\boldsymbol{\alpha})$ の最大化問題の最適解 $\hat{\boldsymbol{\alpha}}$ を求めれば，元の解 $\hat{\boldsymbol{w}}$,$\hat{b}$ が得られることが分かります.

	誤	正

p.139, ソースコード 10.2 の 16 行目を削除	`# ハードマージンのモデルを作成`
	`# model = LinearSVC(loss='hinge', C=10000.0, multi_class='ovr', penalty='l2', random_state=0)`
	`model = LinearSVC(loss='hinge', C=10000.0, multi_class='ovr', penalty='l2', dual='auto', random_state=0)`
		`# ハードマージンのモデルを作成`
		~~`# model = LinearSVC(loss='hinge', C=10000.0, multi_class='ovr', penalty='l2', random_state=0)`~~
		`model = LinearSVC(loss='hinge', C=10000.0, multi_class='ovr', penalty='l2', dual='auto', random_state=0)`

p.141	のような警告が出た場合は，`max_iter` を大きくする，`C` を小さくするなどの工夫が必要になります.	のような警告が出た場合は，`linearSVC` において `max_iter` を大きくする (デフォルトは 1000)，`C` を小さくするなどの工夫が必要になります.
p.142, (10.36)	$$y_n = \begin{cases} 1 & ({}^t\boldsymbol{w}\boldsymbol{x}_n > 0) \\ -1 & ({}^t\boldsymbol{w}\boldsymbol{x}_n < 0) \end{cases} \quad (10.36)$$	$$y_n = \begin{cases} 1 & ({}^t\boldsymbol{w}\boldsymbol{x}_n + b > 0) \\ -1 & ({}^t\boldsymbol{w}\boldsymbol{x}_n + b < 0) \end{cases} \quad (10.36)$$
p.146, 1 行目	究極的には一つ一つのデータをすべて別の次元，データが N 個あれば N 次元まで拡張すれば，必ず $N-1$ 次元の分離超平面で分離することができます.	究極的には一つ一つのデータをすべて別の次元，データが N 個のとき N 次元まで拡張すれば，必ず $N-1$ 次元の分離超平面で分離することができます.
pp.147-148, Y は太字にしない.	ここで，${}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}$ がスカラーであることに注意すれば，式 (11.2) および (11.3) より $$\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{y}_n){}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w} = \lambda\boldsymbol{w} \Longrightarrow \boldsymbol{w} = \frac{1}{\lambda N}\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{y}_n)\left({}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}\right)$$ $$= \frac{1}{\lambda N}\sum_{n=1}^{N}\left({}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}\right)\boldsymbol{\phi}(\boldsymbol{y}_n) = \frac{1}{N}\sum_{n=1}^{N}v_n\boldsymbol{\phi}(\boldsymbol{y}_n)$$ $$= \frac{1}{N}[\boldsymbol{\phi}(\boldsymbol{y}_1),\ldots,\boldsymbol{\phi}(\boldsymbol{y}_N)]\begin{bmatrix} v_1 \\ \vdots \\ v_N \end{bmatrix} = \frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{v}$$ を得ます．なお，この式変形において $v_n = \left({}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}\right)/\lambda$, $\boldsymbol{v} = {}^t[v_1,\ldots,v_N]$ としました．したがって，式 (11.3) は次のようになります. $$\frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{\phi}(\boldsymbol{Y})\frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{v} = \lambda\frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{v}$$ $$\Longrightarrow \frac{1}{N}\boldsymbol{\phi}(\boldsymbol{Y})^t\boldsymbol{\phi}(Y)\boldsymbol{\phi}(\boldsymbol{Y})^t\boldsymbol{\phi}(Y)\boldsymbol{v} = \boldsymbol{\phi}(\boldsymbol{Y})^t\boldsymbol{\phi}(Y)\boldsymbol{v}$$ $$\Longrightarrow \frac{1}{N}\boldsymbol{\phi}(\boldsymbol{Y})^t\boldsymbol{\phi}(Y)\boldsymbol{v} = \boldsymbol{\phi}(\boldsymbol{Y})\boldsymbol{v}$$ $$\Longrightarrow K_N\boldsymbol{v} = \lambda\boldsymbol{v}$$	~~ここで，${}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}$ がスカラーであることに注意すれば，~~式 (11.2) および (11.3) より $$V\boldsymbol{w} = \lambda\boldsymbol{w} \Longrightarrow \frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{\phi}(Y)\boldsymbol{w} = \lambda\boldsymbol{w}$$ $$\Longrightarrow \boldsymbol{w} = \frac{1}{N}{}^t\boldsymbol{\phi}(Y)\left(\boldsymbol{\phi}(Y)\frac{\boldsymbol{w}}{\lambda}\right) = \frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{v}$$ を得ます．なお，この式変形において $\boldsymbol{v} = \boldsymbol{\phi}(Y)\boldsymbol{w}/\lambda = {}^t[v_1,\ldots,v_N]$, $v_n = \left({}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}\right)/\lambda$ としました．したがって，$\mu = \lambda N$ とおけば，式 (11.3) は次のようになります. $$\frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{\phi}(Y)\frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{v} = \lambda\frac{1}{N}{}^t\boldsymbol{\phi}(Y)\boldsymbol{v}$$ $$\Longrightarrow \frac{1}{N}\boldsymbol{\phi}(Y)^t\boldsymbol{\phi}(Y)\boldsymbol{\phi}(Y)^t\boldsymbol{\phi}(Y)\boldsymbol{v} = \lambda\boldsymbol{\phi}(Y)^t\boldsymbol{\phi}(Y)\boldsymbol{v}$$ $$\Longrightarrow \frac{1}{N}\boldsymbol{\phi}(Y)^t\boldsymbol{\phi}(Y)\boldsymbol{v} = \lambda\boldsymbol{v}$$ $$\Longrightarrow \frac{1}{N}K_N\boldsymbol{v} = \lambda\boldsymbol{v} \Longrightarrow K_N\boldsymbol{v} = \mu\boldsymbol{v}$$

	誤	正
p.148, (11.4)	$\bar{K}_N = K_n - \bar{E}_N K_N - K_N \bar{E}_N + \bar{E}_N K_N \bar{E}_N$	$\bar{K}_N = K_N - \bar{E}_N K_N - K_N \bar{E}_N + \bar{E}_N K_N \bar{E}_N$
p.149，表現を変更	カーネル関数は，元のデータが高次元空間に投影されたときにどのように見えるか，その「像」を提供します．実際にはデータを高次元空間に投影せずに，それらのデータが高次元空間でどのように見えるか（どのような関係性を持つか）を知ることができます．これは，元のデータ空間でのデータ点間の類似性を測定することによって達成されます． そのため，カーネル PCA における「固有ベクトル」は，この高次元空間における固有ベクトルの「像」を表しています．これらの固有ベクトルは元のデータ空間のデータ点に対応しており，各データ点が新しい（高次元の）特徴空間でどのように位置するかを示しています．それらは,「軸」ではなく,「射影されたデータ点」を表しています． 結局のところ，カーネル PCA の「固有ベクトル」は，この写像を通じてデータが高次元空間でどのように配置されるかを捉えています．したがって，これらの固有ベクトルは，元のデータ空間のデータに対応し，それぞれのデータ点が新しい特徴空間でどのように位置するかを示しています．	カーネル関数は，元のデータが高次元空間に投影された場合の関係性を捉えます．実際にはデータを高次元空間に投影せずに，元の空間におけるデータ点間の類似性を測定することによって，高次元空間でのデータの関係性を知ることができます． そのため，カーネル PCA における固有ベクトルは，主成分軸そのものではなく，データがこれらの軸に射影された結果を表しています．このことは，$v_n = {}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}/\lambda$ であり，${}^t\boldsymbol{\phi}(\boldsymbol{y}_n)\boldsymbol{w}$ が $\boldsymbol{\phi}(\boldsymbol{y}_n)$ を $\boldsymbol{w}$ へ射影した座標であることからも分かります．つまり，これらの固有ベクトルは高次元空間におけるデータ点の射影を表しており，元の空間におけるデータ点の関係性を捉えています． 結局のところ，カーネル PCA の固有ベクトルは，写像された高次元空間におけるデータの分散の方向を捉えています．したがって，これらの固有ベクトルは，元の空間のデータに対応しており，それぞれのデータ点が新しい特徴空間でどのように関連しているかを示しています．

	誤	正
p.157, (11.12). ϕ を太字に.	$${}^t\hat{\boldsymbol{w}}\phi(\boldsymbol{x})+\hat{b}={}^t\left(\sum_{n=1}^{N}\hat{\alpha}_n y_n\boldsymbol{\phi}(\boldsymbol{x}_n)\right)\boldsymbol{\phi}(\boldsymbol{x})+\hat{b}=\sum_{n=1}^{N}\hat{\alpha}_n y_n K(\boldsymbol{x}_n,\boldsymbol{x})+\hat{b}=0$$	$${}^t\hat{\boldsymbol{w}}\boldsymbol{\phi}(\boldsymbol{x})+\hat{b}={}^t\left(\sum_{n=1}^{N}\hat{\alpha}_n y_n\boldsymbol{\phi}(\boldsymbol{x}_n)\right)\boldsymbol{\phi}(\boldsymbol{x})+\hat{b}=\sum_{n=1}^{N}\hat{\alpha}_n y_n K(\boldsymbol{x}_n,\boldsymbol{x})+\hat{b}=0$$
p.158, (11.15)	$$\sum_{n=1}^{N}\hat{\alpha}_n y_n K(\boldsymbol{x}_n,\boldsymbol{x})={}^t(\hat{\boldsymbol{\alpha}}\odot\boldsymbol{y})K(\tilde{X},X)$$	$$\sum_{n=1}^{N}\hat{\alpha}_n y_n K(\boldsymbol{x}_n,\boldsymbol{x})={}^t(\hat{\boldsymbol{\alpha}}\odot\boldsymbol{y})K(X,\boldsymbol{x})$$
p.166, (12.7), (12.8)	$$\begin{aligned}\boldsymbol{w}&\leftarrow w-\eta\frac{\partial E}{\partial\boldsymbol{w}}\\ \boldsymbol{b}&\leftarrow b-\eta\frac{\partial E}{\partial\boldsymbol{b}}\end{aligned}$$	$$\begin{aligned}\boldsymbol{w}&\leftarrow \boldsymbol{w}-\eta\frac{\partial E}{\partial\boldsymbol{w}}\\ \boldsymbol{b}&\leftarrow \boldsymbol{b}-\eta\frac{\partial E}{\partial\boldsymbol{b}}\end{aligned}$$
p.168, (12.22)	$$\frac{\partial\widehat{u}_j}{\partial\widehat{w}_{ij}}=\frac{\partial\left(\sum_{p=1}^{L}\widehat{w}_{pj}x_p+b_j\right)}{\partial\widehat{w}_{ij}}=x_i$$	$$\frac{\partial\hat{u}_j}{\partial\widehat{w}_{ij}}=\frac{\partial\left(\sum_{p=1}^{L}\widehat{w}_{pj}x_p+\widehat{b}_j\right)}{\partial\widehat{w}_{ij}}=x_i$$
p.182, 課題 12.3	... としてプログラムを実行せよ．これらの結果に対する自分の考えや解釈を述べよ.	... としてプログラムを実行せよ．さらに，PyTorch による結果も確認せよ．これらの結果に対する自分の考えや解釈を述べよ.
p.184	畳み込み処理は，**フィルタ** (filter) または**カーネル** (kernel) を用いて，画像から特徴を抽出する操作のことを指します.	畳み込み処理は，**フィルタ** (filter) または**カーネル** (kernel) と呼ばれるものを用いて，画像から特徴を抽出する操作のことを指します.
p.188	畳み込み層やプーリング層の出力を全結合層に入力する際には，出力された画像は平坦なベクトルに変換されます. ... 最終的に「猫らしさ」や「犬らしさ」などの数値を出力するためには，何らかの段階で特徴を 2 次元から 1 次元に変換する必要があります.	畳み込み層やプーリング層の出力を全結合層に入力する際には，出力された画像は~~平坦な~~ベクトル (1 次元配列) に変換されます. ... 最終的に「猫らしさ」や「犬らしさ」などの数値を出力するためには，何らかの段階で特徴を 2 次元配列 (画像あるいは行列) から 1 次元配列 (ベクトル) に変換する必要があります.

	誤	正
p.189	これらを疑似的に再現することが，データ拡張における重要なポイントとなります.	~~これらを疑似的に再現することが，データ拡張における重要なポイントとなります.~~画像に含まれる被写体の意味を変えるような操作をしてはいけません. 例えば，親指を立てたサムズアップと下に向けたサムズダウンとでは意味が異なるので，これらの画像を 180 度回転させてデータ拡張することは適切ではありません.
p.190, モメンタム	ただし，確率的勾配降下法と比較すると，調整が必要な定数が η, α の 2 つなので，やや調整が難しくなります.	ただし，確率的勾配降下法と比較すると，調整が必要な定数が η, α の 2 つなので，やや調整が難しくなります．なお，本節で登場する式 (13.2)〜(13.12) は成分ごとに解釈します．例えば，(13.2) は $w_{jk} \leftarrow w_{jk} - \eta \frac{\partial E}{\partial w_{jk}} + \alpha \Delta w_{jk}$ と解釈します.
p.190, 最終行	以下においても，t が更新回数です.	以下において，右下の (t) が更新回数です.
p.191	$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial E}{\partial w_{t-1}} \quad (13.8)$$ $$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial E}{\partial w_{t-1}}\right)^2 \quad (13.9)$$ $$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t} \quad (13.10)$$ $$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t} \quad (13.11)$$ $$w_t \leftarrow w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} \quad (13.12)$$	$$m_{(0)} = v_{(0)} = 0$$ $$m_{(t)} = \beta_1 m_{(t)} + (1-\beta_1)\frac{\partial E}{\partial w_{(t-1)}} \quad (13.8)$$ $$v_{(t)} = \beta_2 v_{(t-1)} + (1-\beta_2)\left(\frac{\partial E}{\partial w_{(t-1)}}\right)^2 \quad (13.9)$$ $$\hat{m}_{(t)} = \frac{m_{(t)}}{1-\beta_1^t} \quad (13.10)$$ $$\hat{v}_{(t)} = \frac{v_{(t)}}{1-\beta_2^t} \quad (13.11)$$ $$w_{(t)} = w_{(t-1)} - \eta \frac{\hat{m}_{(t)}}{\sqrt{\hat{v}_{(t)}}+\epsilon} \quad (13.12)$$

	誤	正
p.196, 10 行目を 9 行目に移動	7 行目: `# 畳み込み層:(入力チャンネル数，フィルタ数、フィルタサイズ)` 8 行目: `self.conv1 = nn.Conv2d(3, 6, 5)` 9 行目：`# プーリング層:（領域のサイズ，ストライド）` 10 行目：`self.conv2 = nn.Conv2d(6, 16, 5)` 11 行目：`self.pool = nn.MaxPool2d(2, 2)`	7 行目: `# 畳み込み層:(入力チャンネル数，フィルタ数、フィルタサイズ)` 8 行目: `self.conv1 = nn.Conv2d(3, 6, 5)` 9 行目：`self.conv2 = nn.Conv2d(6, 16, 5)` 10 行目：`# プーリング層:（領域のサイズ，ストライド）` 11 行目：`self.pool = nn.MaxPool2d(2, 2)`

p.196, 上記変更に伴う出力結果変更．(conv2) と (pool) を入れ替え.	`Net(`
	`  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))`
	`  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)`
	`  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))`
	`  (fc1): Linear(in_features=400, out_features=256, bias=True)`
	`  (dropout): Dropout(p=0.5, inplace=False)`
	`  (fc2): Linear(in_features=256, out_features=10, bias=True)`
	`)`
		`Net(`
		`  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))`
		`  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))`
		`  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)`
		`  (fc1): Linear(in_features=400, out_features=256, bias=True)`
		`  (dropout): Dropout(p=0.5, inplace=False)`
		`  (fc2): Linear(in_features=256, out_features=10, bias=True)`
		`)`

p.197	**8 行目** ... **10 行目** ... **11 行目** ...	**8, 20 行目** ... **11, 20 行目** ... **9, 11, 21 行目** ...
p.198, 実行例の後に追記		ソースコード 13.5 の 32 行目にある eval メソッドは，モデルを評価モードに設定するために使用されます．この評価モードは，Dropout のような訓練と評価で挙動が異なるメソッドに影響します．訓練モードでは，これらのメソッドはランダムな挙動を示しますが，評価モードではランダム性が排除され，一貫した出力が得られます．そのため，テストデータセットでの評価や新しいデータに対する予測を行う際には，`network.eval()` のように指定して評価モードに設定することが重要です.

	誤	正
p.209, ソースコード 14.3	`# 最適化アルゴリズム SDG を指定`	`# 最適化アルゴリズム SGD を指定`

p.213, ソースコード 14.5	`# 各設定値`
	`n_steps = 10  # 時系列のステップ数`
	`n_input = 1  # 入力層のニューロン数`
	`n_hidden = 20  # 中間層のニューロン数`
	`n_output = 1  # 出力層のニューロン数`
		`# 各設定値`
		`n_steps = 10  # 時系列のステップ数．ソースコード 14.1 の n_time に相当`
		`n_input = 1  # 入力層のニューロン数．ソースコード 14.1 の n_in に相当`
		`n_hidden = 20  # 中間層のニューロン数`
		`n_output = 1  # 出力層のニューロン数．ソースコード 14.1 の n_out に相当`