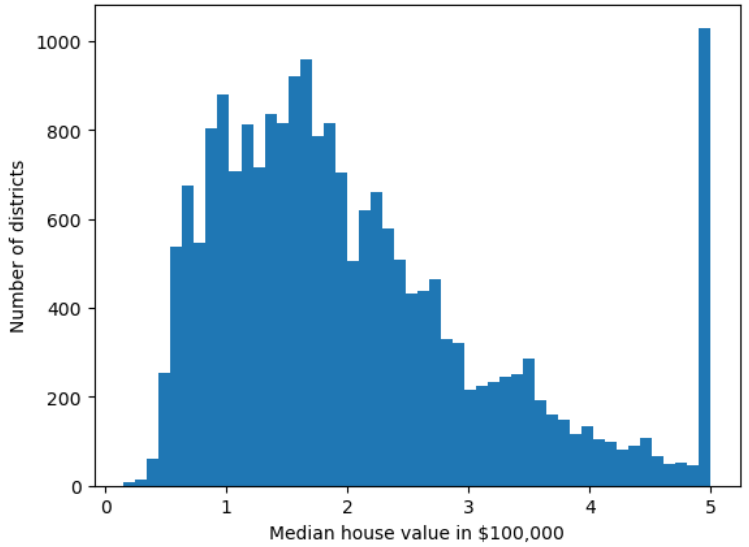
最終更新日: 2025 年 2 月 14 日

「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>全体的な修正，print の文字の色</td><td>print(....)</td><td>print(....)</td></tr>
<tr><td>全体的な修正，print および実行結果</td><td>accracy_score による</td><td>accuracy_score による</td></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」(2021 年 3 月 29 日制定) 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></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.13，ソースコード 1.6 の 6 行目	`random\_state:7`	`random_state:7`
p.15	ベクトル $\boldsymbol{x} = {}^t[x_1, x_2, \ldots, x_N]$ に対して，$\|\boldsymbol{x}\|_2 = \sum_{i=1}^N x_i^2$ です．また，^{t} はベクトルあるいは行列の転置を表します．	ベクトル $\boldsymbol{x} = {}^t[x_1, x_2, \ldots, x_N]$ に対して，$\|\boldsymbol{x}\|_2^2 = \sum_{i=1}^N x_i^2$ です．また，^{t} はベクトルあるいは行列の転置を表します．
p.16	と表され，この形は第 3.1 節の重回帰式 (3.1) と同じになります．	と表され，この形は第 3 章の重回帰式 (3.1) と同じになります．
p.17, 補題 2.2 の証明	$\dfrac{\partial E}{\partial w_n} = \sum_{i=1}^N \left(\sum_{j=0}^p w_j x_i^j - y_i \right) x_i, \quad \dfrac{\partial^2 E}{\partial w_m \partial w_n} = \dfrac{\partial}{\partial w_m} \left(\dfrac{\partial E}{\partial w_n} \right) = \sum_{i=1}^N x_i^m x_i^n$	$\dfrac{\partial E}{\partial w_n} = \sum_{i=1}^N \left(\sum_{j=0}^p w_j x_i^j - y_i \right) x_i^n, \quad \dfrac{\partial^2 E}{\partial w_m \partial w_n} = \dfrac{\partial}{\partial w_m} \left(\dfrac{\partial E}{\partial w_n} \right) = \sum_{i=1}^N x_i^m x_i^n$
p.20	これは，実際の値と予測値の絶対値の 2 乗を平均したものであり，値が 0 に近いほどいいモデルだといえます．	これは，実際の値と予測値の差の 2 乗を平均したものであり，値が 0 に近いほどいいモデルだといえます．
p.26, 課題 2.7	また，これらの結果に対する自分の考えや解釈を述べよ．	さらに，これらの結果に対する自分の考えや解釈を述べよ．
p.33, 課題 3.4	ソースコード 3.2〜3.4 の結果を確認せよ．また，ソースコード 3.2 の実行結果から，どのようなことが読み取れるか述べよ．	ソースコード 3.1〜3.4 の結果を確認せよ．また，ソースコード 3.4 の実行結果から，どのようなことが読み取れるか述べよ．
p.33，ソースコード 3.4 の 6 行目および実行例	`plt.xlabel('Median house value in 10,000$')`	`plt.xlabel('Median house value in $100,000')` 

	誤	正
p.35, 課題 3.6, 3.7	**課題 3.6** SciPy を使い，カリフォルニア住宅価格データセットに対し，SciPy を使って線形単回帰分析をせよ．ただし，説明変数は `MedInc` 目的変数は `MedHouseVal` とする． **課題 3.7** SciPy を使い，カリフォルニア住宅価格データセットに対し，NumPy を使って多項式回帰分析をせよ．ただし，説明変数は `MedInc` 目的変数は `MedHouseVal` とする．	**課題 3.6** ~~SciPy を使い，~~カリフォルニア住宅価格データセットに対し，SciPy を使って線形単回帰分析をせよ．ただし，説明変数は `MedInc` ，目的変数は `MedHouseVal` とする． **課題 3.7** ~~SciPy を使い，~~カリフォルニア住宅価格データセットに対し，NumPy を使って多項式回帰分析をせよ．ただし，説明変数は `MedInc` ，目的変数は `MedHouseVal` とする．
p.36, 課題 3.9	ソースコード 3.11〜3.12 の実行結果を確認せよ．	ソースコード~~3.11〜~~3.12 の実行結果を確認せよ．
p.42, ソースコード 3.21 実行例	`0.7253534565158777`	`0.5404128061709476`
p.42, 課題 3.17	ソースコード 3.20 の 18 行目で `reshape` しなかった場合はどのようになるかを確認せよ．	ソースコード 3.20 の実行結果を確認せよ．また，20 行目で `reshape` しなかった場合はどのようになるかを確認せよ．
p.43	${}^t\boldsymbol{wx} > 0$ であれば，分類モデルは $f({}^t\boldsymbol{wx}) \geq 0.5$ となり，クラス 1 と分類する	${}^t\boldsymbol{wx} \geq 0$ であれば，分類モデルは $f({}^t\boldsymbol{wx}) \geq 0.5$ となり，クラス 1 と分類する
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
p.52, 4.3.8	scikit-learn でロジスティック回帰を行うには，以下のように `LogisticRegression` クラスを使います．	scikit-learn でロジスティック回帰を行うには，以下のように `LogisticRegression` クラスを使います．なお，デフォルトでは L2 正則化が適用されます．

	誤	正
p.53, 課題 4.8	また，この結果やソースコード 4.9 の結果どのようなことがいえるかを述べよ.	また，この結果やソースコード 4.9 の結果からどのようなことがいえるかを述べよ.
p.53，ソースコード 4.11 の 3 行目．半角スペースを入れる.	`#プロット用です`	`# プロット用です`
p.55, (4.11)	$$\frac{\partial \hat{y}_n}{\partial w_m} = f'({}^t\boldsymbol{w}\boldsymbol{x}_n)\frac{\partial({}^t\boldsymbol{w}\boldsymbol{x}_n)}{\partial \boldsymbol{w}_m} = \hat{y}_n(1-\hat{y}_n)x_m^{(n)}$$	$$\frac{\partial \hat{y}_n}{\partial w_m} = f'({}^t\boldsymbol{w}\boldsymbol{x}_n)\frac{\partial({}^t\boldsymbol{w}\boldsymbol{x}_n)}{\partial w_m} = \hat{y}_n(1-\hat{y}_n)x_m^{(n)}$$
p.56	なお，反復終了判定は，$$\frac{\|\boldsymbol{w}_{new}-\boldsymbol{w}_{old}\|}{\|\boldsymbol{w}_{old}\|^2} < \varepsilon \quad (4.18)$$	なお，反復終了判定は，$$\frac{\|\boldsymbol{w}_{new}-\boldsymbol{w}_{old}\|}{\|\boldsymbol{w}_{old}\|} < \varepsilon \quad (4.18)$$
p.57，ソースコード 4.12 の 23 行目	`if 【自分で補おう】 < 0.001*np.dot(self.w, self.w)):`	`if 【自分で補おう】 < 0.001* np.sqrt(np.dot(self.w, self.w)):`
p.58，ソースコード 4.12 の 45 行目	`print("偏相関係数の表示：\n", myModel.w)`	`print("偏回帰係数の表示：\n", myModel.w [1:])`
p.58，課題 4.10 の前にソースコード 4.12 の実行結果を追加	**課題 4.10**	実行例 accuracy_score 正解率 (train)0.71693 accuracy_score 正解率 (test) 0.73044 偏回帰係数の表示： [-0.65518814 -0.06071187 0.09827254 0.0217363 -0.41395887 -0.00711961 0.8763661 1.0289339 0.80110543 1.51801194 1.7447284 0.12449028 0.27584449 0.10199607 0.09381185 0.04624111] **課題 4.10**

	誤	正
p.60, 中程	となります．ただし，$p(\boldsymbol{x}, C_k)$ は，入力 $\boldsymbol{x}$ とクラス C_k が同時に起こる確率を表します．	となります．ただし，$P(\boldsymbol{x}, C_k)$ は，入力 $\boldsymbol{x}$ とクラス C_k が同時に起こる確率を表します．
p.62	$$\begin{bmatrix} u_{n1} \\ u_{n2} \\ \vdots \\ u_{nK} \end{bmatrix} = \begin{bmatrix} w_{01} & w_{11} & \cdots & w_{p1} \\ w_{02} & w_{12} & \cdots & w_{p2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0K} & w_{1K} & \cdots & w_{pK} \end{bmatrix} \begin{bmatrix} x_{n0} \\ x_{n1} \\ \vdots \\ x_{np} \end{bmatrix} \Longleftrightarrow \boldsymbol{u}_n = W\boldsymbol{x}_n = [\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_p]\boldsymbol{x}_n \quad (5.6)$$	$$\begin{bmatrix} u_{n1} \\ u_{n2} \\ \vdots \\ u_{nK} \end{bmatrix} = \begin{bmatrix} w_{01} & w_{11} & \cdots & w_{p1} \\ w_{02} & w_{12} & \cdots & w_{p2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0K} & w_{1K} & \cdots & w_{pK} \end{bmatrix} \begin{bmatrix} x_{n0} \\ x_{n1} \\ \vdots \\ x_{np} \end{bmatrix} \Longleftrightarrow \boldsymbol{u}_n = W\boldsymbol{x}_n = \begin{bmatrix} {}^t\boldsymbol{w}_0 \\ {}^t\boldsymbol{w}_1 \\ \vdots \\ {}^t\boldsymbol{w}_p \end{bmatrix} \boldsymbol{x}_n \quad (5.6)$$
p.63	まず，初期値 $\boldsymbol{w}^{(1)}$ を適当に決め，式 (5.10) を $t = 1, 2, \ldots$ に対して逐次計算し，$\boldsymbol{w}^{(2)}$, $\boldsymbol{w}^{(3)}$, $\ldots$ を求めます．こうして計算される $\boldsymbol{w}^{(t)}$ は η が小さければ，t が増えるにつれ，$E(\boldsymbol{w}^{(t)})$ を確実に減少させます．	まず，初期値 $\boldsymbol{w}^{(0)}$ を適当に決め，式 (5.10) を $t = 0, 1, 2 \ldots$ に対して逐次計算し，$\boldsymbol{w}^{(1)}$, $\boldsymbol{w}^{(2)}$, $\ldots$ を求めます．こうして計算される $\boldsymbol{w}^{(t)}$ は η が小さければ，t が増えるにつれ，$E(\boldsymbol{w}^{(t)})$ を確実に減少させます．
p.63, 図 5.3		図 5.3 の w_{ij} を w_{ji} に変更．
p.70, 課題 5.4	ソースコード 5.4〜5.5 の実行結果を確認せよ．また，他の 3 つの説明変数についてもヒストグラムを作成し，ヒストグラムや散布図行列に対する自分の考えや解釈を述べよ．	ソースコード 5.4〜5.5 の実行結果を確認せよ．また，この結果に対する自分の考えや解釈を述べよ．

p.81，ソースコード 6.4 の 22〜30 行目を削除	`# ランダムフォレストのモデルを作成`
	`rfmodel = RandomForestClassifier(bootstrap=True, n_estimators=10,`
	`criterion='gini', max_depth=None, random_state=1)`
	
	`# モデルの訓練`
	`rfmodel.fit(X_train, y_train)`
	
	`# accuracy_score による正解率`
	`y_test_pred = rfmodel.predict(X_test)`
	`print('accracy_score による正解率 (test):{:.4f}'.format(accuracy_score(y_test, 【自分で補おう】)))`
	
	`# ランダムフォレストのモデルを作成`
	`rfmodel = RandomForestClassifier(bootstrap=True, n_estimators=10,`
	`criterion='gini', max_depth=None, random_state=1)`
	
	`# モデルの訓練`
	`rfmodel.fit(X_train, y_train)`
	
	`# accuracy_score による正解率`
	`y_test_pred = rfmodel.predict(X_test) # 予測`
	`print('accracy_score による正解率 (test):{:.4f}'.format(accuracy_score(y_test, y_test_pred)))`
		`# ランダムフォレストのモデルを作成`
		`rfmodel = RandomForestClassifier(bootstrap=True, n_estimators=10,`
		`criterion='gini', max_depth=None, random_state=1)`
		
		`# モデルの訓練`
		`rfmodel.fit(X_train, y_train)`
		
		`# accuracy_score による正解率`
		`y_test_pred = rfmodel.predict(X_test)`
		`print('accracy_score による正解率 (test):{:.4f}'.format(accuracy_score(y_test, 【自分で補おう】)))`

	誤	正
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, 適合率	Positive と分類したうちで，正解した割合 (PRE)．予測の誤判定を避けたい場合に利用する．予測の誤判定を避けたい場合に利用する．	Positive と分類したうちで，正解した割合 (PRE)．予測の誤判定を避けたい場合に利用する．例えば，$PER = 0.9$ は，Positive と判定したうち，90%が実際に Positive であることを意味する．予測の誤判定を避けたい場合に利用する．
p.90，再現率	実際のクラスが Positive のうち，Positive だと予測できた割合 (REC)．再現率は,「陽性」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，偽陰性 (FN) を減らすことを重視する場合に利用する．再現率は，**感度** (sensitivity) とも呼ばれる．	実際のクラスが Positive のうち，Positive だと予測できた割合 (REC)．例えば，$REC = 0.9$ は，データに含まれる Positive の 90%を検出できることを意味する．再現率は,「陽性」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，偽陰性 (FN) を減らすことを重視する場合に利用する．再現率は，**感度** (sensitivity) とも呼ばれる．
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．

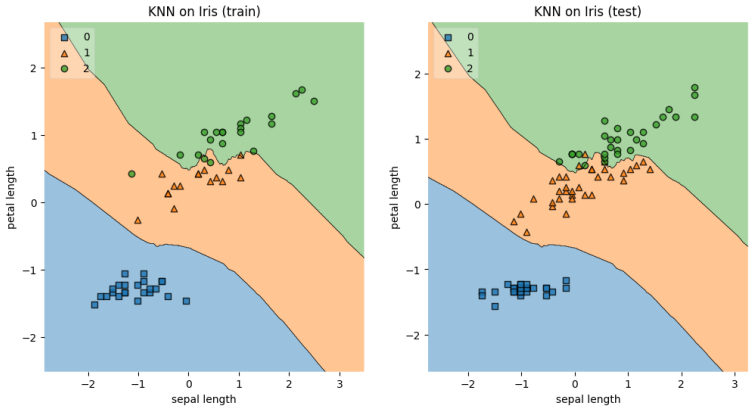
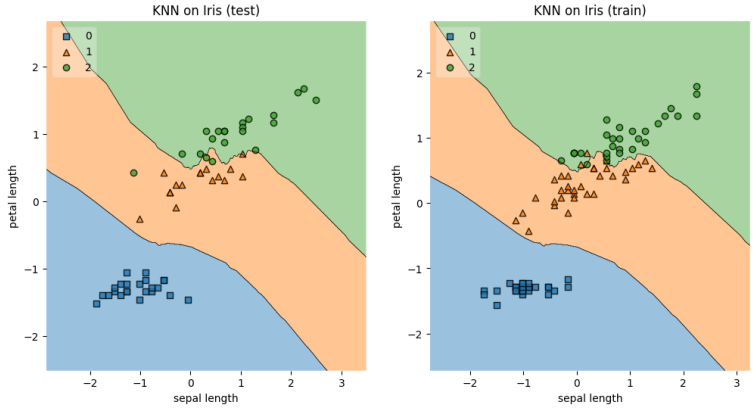
	誤	正

p.95	`# 混同行列から特異性を求める関数`
	`def specificity(y_true, y_pred):`
	`  tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()`
		`# 混同行列から特異率を求める関数`
		`def specificity(y_true, y_pred):`
		`  # 混同行列を 1 次元配列にして各要素を tn,fp,fn,tp に割り当てる`
		`  tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()`

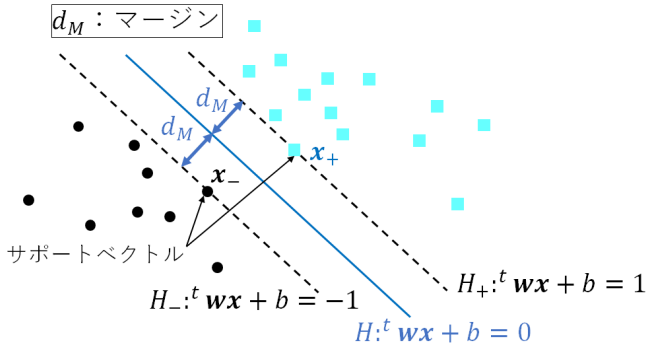
p.96, ソースコード 7.2 の 47, 48 行目	`# 陽性クラスの確率` `# 陽性クラスの確率`	`# 陽性クラスの確率（訓練データ）` `# 陽性クラスの確率（テストデータ）`
p.102	$w_{ik} = \begin{cases} 1 & (\boldsymbol{x}_i \text{が} k \text{番目のセントロイドのクラスターに属する}) \\ 0 & (\text{それ以外}) \end{cases}$	$w_{ik} = \begin{cases} 1 & (\boldsymbol{x}_i \text{がセントロイド} \boldsymbol{\mu}_k \text{のクラスターに属する}) \\ 0 & (\text{それ以外}) \end{cases}$
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.
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.108, ソースコード 8.2, 25 行目	`# テストデータの予測`
	`Y_pred = knn.predict(X_test)`
	`Y_pred_train = 【自分で補おう】`
		`# 訓練データ, テストデータの予測`
		`Y_pred = knn.predict(X_test) # テストデータの予測`
		`Y_pred_train = 【自分で補おう】 # 訓練データの予測`

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)')`

	誤	正
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.127	(1) p 次元の特徴量ベクトル $\boldsymbol{x}$ を平均偏差ベクトルにする． (2) 特徴量ベクトルに基づき，$p \times p$ の共分散行列を作成する． (3) 共分散行列の固有値と単位固有ベクトルを求める． (4) 固有値を大きい順に並べ替えて，単位固有ベクトルで $p \times p$ の射影行列を作成する． (5) $p \times p$ の射影行列の k 列より後の列を削除し，$p \times r$ の射影行列に変換する． (6) p 次元の特徴量ベクトル $\boldsymbol{x}$ に射影行列 W を掛けて，r 次元の主成分 $\boldsymbol{f}$ を作成する．	(1) p 次元の特徴量ベクトル $\boldsymbol{x}$ を平均偏差ベクトル $\boldsymbol{y}$ にする． (2) 特徴量ベクトルに基づき，$p \times p$ の共分散行列を作成する． (3) 共分散行列の固有値と単位固有ベクトルを求める． (4) 固有値を大きい順に並べ替えて，単位固有ベクトルで $p \times p$ の射影行列を作成する． (5) $p \times p$ の射影行列の k 列より後の列を削除し，$p \times r$ の射影行列に変換する． (6) p 次元の平均偏差ベクトル $\boldsymbol{y}$ に射影行列 W を掛けて，r 次元の主成分 $\boldsymbol{f} = {}^tW\boldsymbol{y}$ を作成する．
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.129, 課題 9.8	ソースコード 9.7 を実行せよ．また，この結果に対する自分の考えや解釈を述べよ．	ソースコード 9.7 の実行結果を確認せよ．また，この結果に対する自分の考えや解釈を述べよ．
p.130, 図 10.1(a) を変更		

	誤	正
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．
p.131,(10.9)	$$\hat{\boldsymbol{w}}, \hat{b} = \underset{\boldsymbol{w},b}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{w}\|^2, \quad \text{subject to } y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b) \geq 1 \quad (n = 1, 2, \ldots, N) \quad (10.9)$$	$$\{\hat{\boldsymbol{w}}, \hat{b}\} = \underset{\boldsymbol{w},b}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{w}\|^2, \quad \text{subject to } y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b) \geq 1 \quad (n = 1, 2, \ldots, N) \quad (10.9)$$
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)	$$\alpha_i \geq 0 \quad (i = 1, 2, \ldots, N) \quad (10.17)$$ $$\alpha_i = 0 \quad \text{または} \quad y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b) - 1 = 0 \quad (10.18)$$	$$\alpha_n \geq 0 \quad (n = 1, 2, \ldots, N) \quad (10.17)$$ $$\alpha_n = 0 \quad \text{または} \quad y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b) - 1 = 0 \quad (10.18)$$
p.133 の式変形	$$\begin{aligned} L(\boldsymbol{w}, b, \alpha_1, \ldots, \alpha_N) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \alpha_n \left\{1 - y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b)\right\} \\ &\vdots \\ &= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i\right)\left(\sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j\right) \end{aligned}$$	$$\begin{aligned} L(\boldsymbol{w}, b, \alpha_1, \ldots, \alpha_N) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \alpha_n \left\{1 - y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b)\right\} \\ &\vdots \\ &= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}{}^t\left(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i\right)\left(\sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j\right) \end{aligned}$$

	誤	正
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．
p.134	この行列の成分は，$[H]_{ij} = y_i y_j {}^t\boldsymbol{x}_i\boldsymbol{x}_j$ であり，H は対称行列です． これらの記号を使うと，式 (10.21) は $\tilde{L}(\alpha_1,\ldots,\alpha_N) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j {}^t\boldsymbol{x}_i\boldsymbol{x}_j$ $= \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j (H)_{ij}$	この行列 H の (i,j) 成分は，$h_{ij} = y_i y_j {}^t\boldsymbol{x}_i\boldsymbol{x}_j$ であり，H は対称行列です． これらの記号を使うと，式 (10.21) は $\tilde{L}(\alpha_1,\ldots,\alpha_N) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j {}^t\boldsymbol{x}_i\boldsymbol{x}_j$ $= \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j h_{ij}$
p.134	したがって，勾配ベクトルは次のように計算できます． $\frac{\partial\tilde{L}(\alpha)}{\partial\alpha_j} = 1 - \frac{1}{2}\left\{\frac{\partial({}^t\boldsymbol{\alpha})}{\partial\alpha_j}(H\boldsymbol{\alpha}) + {}^t\boldsymbol{\alpha}H\frac{\partial\boldsymbol{\alpha}}{\partial\alpha_j}\right\} = 1 - H\boldsymbol{\alpha_j}$	$\frac{\partial\tilde{L}(\alpha)}{\partial\alpha_j} = 1 - \frac{1}{2}\left\{\frac{\partial({}^t\boldsymbol{\alpha})}{\partial\alpha_j}(H\boldsymbol{\alpha}) + {}^t\boldsymbol{\alpha}H\frac{\partial\boldsymbol{\alpha}}{\partial\alpha_j}\right\} = 1 - (H\boldsymbol{\alpha})_j$
p.135	具体的には，**スラック変数** (slack：ゆるい)ξ を導入することで，	具体的には，**スラック変数** (slack：ゆるい) ξ_n を導入することで，
p.136	$\arg\min_{\boldsymbol{w},\boldsymbol{\xi},b}\left\{\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\max\{0, 1 - y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b)\}\right\}$ (10.27)	$\operatorname{argmin}_{\boldsymbol{w},b}\left\{\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\max\{0, 1 - y_n({}^t\boldsymbol{w}\boldsymbol{x}_n + b)\}\right\}$ (10.27)

	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	この行列の成分は，$[H]_{ij} = y_i j_j {}^t\boldsymbol{x}_i \boldsymbol{x}_j$ であり，	この行列 H の (i,j) 成分は，$h_{ij} = y_i y_j {}^t\boldsymbol{x}_i \boldsymbol{x}_j$ であり，
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.144, ソースコード 10.5 の 83 行目	`# svm のパラメータを学習`	`# SVM のパラメータを学習`
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．

<table>
<tr><th></th><th>誤</th><th>正</th></tr>
<tr><td>p.156</td><td>に変更し，
<pre>
# SVC を利用　kernel='rbf', class_weight='balanced'
svc = SVC(kernel='rbf', class_weight='balanced')

# パイプラインで標準化，PCA，SVC をつなげる
model = make_pipeline(StandardScaler(), pca, svc)
</pre>を
<pre>
# KernelPCA で次元圧縮　80 次元，random_state=42
kpca = KernelPCA(n_components=80, random_state=42)

# SVC を利用　kernel='rbf', class_weight='balanced'
svc = SVC(kernel='rbf', class_weight='balanced')

# StandardScaler を用いてデータのスケーリングを行い，次に KernelPCA と SVC でパイプラインを作成
model = make_pipeline(StandardScaler(), kpca, svc)
</pre>に変更すればよい．</td><td>に変更し，ソースコード 11.2 の 21〜28 行目を
<pre>
# KernelPCA で次元圧縮　80 次元，random_state=42
kpca = KernelPCA(n_components=80, random_state=42)

# SVC を利用　kernel='rbf', class_weight='balanced'
svc = SVC(kernel='rbf', class_weight='balanced')

# StandardScaler を用いてデータのスケーリングを行い，次に KernelPCA と SVC でパイプラインを作成
model = make_pipeline(StandardScaler(), kpca, svc)
</pre>に変更すればよい．なお，KernelPCA では whiten オプションを指定することはできない．</td></tr>
</table>

	誤	正
p.157, (11.10)	$[H]_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$,	$H = [h_{ij}] = [y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)]$,
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, 1 行目	と表せば，以下のように表現できます.	と表せば，以下のように表現できます．ただし，X は「訓練データ」を成分とする行列，$\tilde{X}$ は「テストデータ（あるいは予測したいデータ）」を成分とする行列です.
p.159, ソースコード 11.5 の 88 行目	`# svm のパラメータを学習`	`# SVM のパラメータを学習`
p.161 の中程	最後に，この値 $u = w_1x_1 + w_2x_2 + w_3x_3$ を**活性化関数**	最後に，この値 $u = w_1x_1 + w_2x_2 + w_3x_3 + b$ を**活性化関数**
p.165	訓練データ全体を一度用いて行う学習を 1 **エポック** (epoch) と数えます．訓練データを複数のサンプルに分けたとき，そのサンプルのまとまりを**バッチ** (batch)，バッチに含まれるサンプル数を**バッチサイズ** (batch size) と称します.	訓練データ全体を一度用いて行う学習を 1 **エポック** (epoch) と数えます．訓練データを複数のサンプルに分けたとき，そのサンプルのまとまりを**バッチ** (batch)，バッチに含まれるサンプル数を**バッチサイズ** (batch size) と称します．バッチサイズのデータを一度用いて行う学習を 1 **イテレーション** (iteration) と数えます．例えば，訓練データセットが 1000 個のサンプルで構成されており，バッチサイズが 100 の場合，1 エポックは 10 イテレーションで構成されます.
p.166, (12.7), (12.8)	$$\begin{aligned} \boldsymbol{w} &\leftarrow w - \eta\frac{\partial E}{\partial \boldsymbol{w}} \\ \boldsymbol{b} &\leftarrow b - \eta\frac{\partial E}{\partial \boldsymbol{b}} \end{aligned}$$	$$\begin{aligned} \boldsymbol{w} &\leftarrow \boldsymbol{w} - \eta\frac{\partial E}{\partial \boldsymbol{w}} \\ \boldsymbol{b} &\leftarrow \boldsymbol{b} - \eta\frac{\partial E}{\partial \boldsymbol{b}} \end{aligned}$$

	誤	正
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.177, 課題 12.2	変更して，プログラムを実行せよ．これらの	変更して，ソースコード 12.3〜12.6 を実行せよ．これらの
p.168, (12.25) の前	バイアスも同様に求めることができます．	バイアスの勾配も同様に求めることができます．

p.182, ソースコード 12.8 の 21 行目	`for i, output in enumerate(output_layer.z):`
	`  class_index = np.argmax(【自分で補おう】)`
	`  # 各サンプルについて最も確率が高いクラスのインデックスを取得`
	`  # サンプルのクラス名とその確率を表示`
		`for i, output in enumerate(output_layer.z):`
		`  # 各サンプルについて最も確率が高いクラスのインデックスを取得`
		`  class_index = np.argmax(【自分で補おう】)`
		`  # サンプルのクラス名とその確率を表示`

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．	... としてソースコード 12.7〜12.8 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.191	式 (13.8) は式 (13.2) に，式 (13.9) は式 (13.6) に対応しており，式 (13.10) によりこれらを組み合わせた更新しています.	式 (13.8) は式 (13.2) に，式 (13.9) は式 (13.6) に対応しており，式 (13.12) によりこれらを組み合わせた更新しています.
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 19 行目と 32 行目では，それぞれ学習モード `network.train()` と評価モード `network.eval()` を指定しています．今回はドロップアウトを使用しているため，学習と評価で挙動が異なります．学習モードでは，ドロップアウトにより一定の確率でニューロンが無効化されるため，入力が同じでも出力がランダムに変化します．一方，評価モードではドロップアウトが無効化されるため，同じ入力に対して常に同じ出力を返し，一貫した推論結果が得られます．このように，学習時と推論時で挙動が変わるレイヤが含まれる場合，テスト時には必ず `model.eval()` を呼び出すのが PyTorch の推奨手法です.

	誤	正
p.201, 図 14.2, 14.3		図中の (t) を (T) にする.
p.202	**隠れ状態ベクトル** (hidden state layer) の次元を示します.	**隠れ状態ベクトル** (hidden state vector) の次元を示します.
p.205	入力に対する損失関数の勾配は次のようになります. $$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} {}^{t}X^{(t)}\Delta^{(t)} = \sum_{t=1}^{T} \begin{bmatrix} x_{11}^{(t)} & x_{21}^{(t)} & \cdots & x_{h1}^{(t)} \\ x_{12}^{(t)} & x_{22}^{(t)} & \cdots & x_{h2}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1m}^{(t)} & x_{2m}^{(t)} & \cdots & x_{hm}^{(t)} \end{bmatrix} \begin{bmatrix} \delta_{11}^{(t)} & \delta_{12}^{(t)} & \cdots & \delta_{1n}^{(t)} \\ \delta_{21}^{(t)} & \delta_{22}^{(t)} & \cdots & \delta_{2n}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{h1}^{(t)} & \delta_{h2}^{(t)} & \cdots & \delta_{hn}^{(t)} \end{bmatrix}$$ ここで，h はバッチサイズ，m は入力ベクトルの次元です．したがって，${}^{t}X^{(t)}$ および $\Delta^{(t)}$ はそれぞれ $h \times m$ および $h \times n$ の次元を持つ行列となります． 行列の各要素は，バッチ内のデータ点および時間ステップ全体での総和を取ったものになります．したがって，あるバッチ内での勾配は，各データ点における勾配の総和で求めることができます．これは，式 (14.7) によってバッチ処理に対応しています.	式 (14.7) より，入力に対する重み行列 W の勾配は次のようになります. $$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} {}^{t}X^{(t)}\Delta^{(t)} = \sum_{t=1}^{T} \begin{bmatrix} x_{11}^{(t)} & x_{21}^{(t)} & \cdots & x_{h1}^{(t)} \\ x_{12}^{(t)} & x_{22}^{(t)} & \cdots & x_{h2}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1m}^{(t)} & x_{2m}^{(t)} & \cdots & x_{hm}^{(t)} \end{bmatrix} \begin{bmatrix} \delta_{11}^{(t)} & \delta_{12}^{(t)} & \cdots & \delta_{1n}^{(t)} \\ \delta_{21}^{(t)} & \delta_{22}^{(t)} & \cdots & \delta_{2n}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{h1}^{(t)} & \delta_{h2}^{(t)} & \cdots & \delta_{hn}^{(t)} \end{bmatrix}$$ ここで，h はバッチサイズ，m は入力ベクトルの次元です．したがって，${}^{t}X^{(t)}$ および $\Delta^{(t)}$ はそれぞれ $m \times h$ および $h \times n$ の次元を持つ行列となります． 行列の各要素は，バッチ内のデータ点および時間ステップ全体での総和を取ったものになります．したがって，あるバッチ内での勾配は，各データ点における勾配の総和で求めることができます．~~これは，式 (14.7) によってバッチ処理に対応しています.~~

	誤	正
p.206	まず，次のように定義します． $$\delta_{out}^{(t)} = \frac{\partial E}{\partial \widetilde{u}^{(t)}} \quad (14.12)$$ これにより，式 (14.7) と同様の考え方から，次の式を得ることができます． $$\frac{\partial E}{\partial \widetilde{w}_i} = \sum_{i=1}^{T} \frac{\partial E}{\partial \widetilde{u}^{(t)}} \frac{\partial \widetilde{u}^{(t)}}{\partial \widetilde{w}_i} = \sum_{i=1}^{T} y_i^{(t)} \delta_{out}^{(t)} \quad (14.13)$$	まず，次のように定義します． $$\delta_{out,j}{}^{(t)} = \frac{\partial E}{\partial \widetilde{u}_j^{(t)}} \quad (14.12)$$ これにより，式 (14.7) と同様の考え方から，次の式を得ることができます． $$\frac{\partial E}{\partial \widetilde{w}_{ij}} = \sum_{i=1}^{T} \frac{\partial E}{\partial \widetilde{u}_j^{(t)}} \frac{\partial \widetilde{u}_j^{(t)}}{\partial \widetilde{w}_{ij}} = \sum_{i=1}^{T} y_i^{(t)} \delta_{out,j}^{(t)} \quad (14.13)$$
p.206, (14.14), (14.15)	$$\delta_j^{(t)} = \left(\sum_r \delta_r^{(t+1)} w_{jr} \right) \frac{\partial y_j}{\partial u_j} = \left(\sum_r \delta_r^{(t+1)} w_{jr} \right) f'(u_j^{(t)}) \quad (14.14)$$ $$\delta_j^{(t)} = \left(\sum_r \delta_{out,r}^{(t)} \widetilde{w}_{jr} + \sum_q \delta_r^{(t+1)} w_{jq} \right) f'(u_j^{(t)}) \quad (14.15)$$	$$\delta_j^{(t)} = \left(\sum_r \delta_r^{(t+1)} w_{jr} \right) \frac{\partial y_j^{(t)}}{\partial u_j^{(t)}} = \left(\sum_r \delta_r^{(t+1)} w_{jr} \right) f'(u_j^{(t)}) \quad (14.14)$$ $$\delta_j^{(t)} = \left(\sum_r \delta_{out,r}^{(t)} \widetilde{w}_{jr} + \sum_q \delta_r^{(t+1)} v_{jq} \right) f'(u_j^{(t)}) \quad (14.15)$$
p.206, (14.15) の後	なお，$\delta^{(T+1)}$ はまだ計算できないため，暫定的に $\delta_j^{(T+1)} = 0$ とします．これは，最後の時刻 $T+1$ における逆伝播の勾配が存在しないという意味です．	なお，$\delta_j{}^{(T+1)}$ はまだ計算できないため，暫定的に $\delta_j^{(T+1)} = 0$ とします．これは，~~最後の~~時刻 $T+1$ における逆伝播の勾配が存在しないという意味です．
p.209, ソースコード 14.2, 13 行目	`def forward(self, x):  # フォワードパスの定義`	`def forward(self, x):  # 順伝播の定義`
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`

	誤	正
p.212 の式 (14.19)〜(14.21)	$$\frac{\partial E}{\partial W} = \sum_{t=1}^{\tau} {}^{t}X^{(t)}\Delta^{(t)} \quad (14.19)$$ $$\frac{\partial E}{\partial V} = \sum_{t=1}^{\tau} {}^{t}Y^{(t-1)}\Delta^{(t)} \quad (14.20)$$ $$\frac{\partial E}{\partial B} = \begin{bmatrix} \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{k1}^{(t)} & \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{k2}^{(t)} & \cdots & \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{kn}^{(t)} \\ \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{k1}^{(t)} & \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{k2}^{(t)} & \cdots & \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{kn}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{k1}^{(t)} & \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{k2}^{(t)} & \cdots & \sum_{t=1}^{\tau}\sum_{k=1}^{h}\delta_{kn}^{(t)} \end{bmatrix} \quad (14.21)$$	$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} {}^{t}X^{(t)}\Delta^{(t)} \quad (14.19)$$ $$\frac{\partial E}{\partial V} = \sum_{t=1}^{T} {}^{t}Y^{(t-1)}\Delta^{(t)} \quad (14.20)$$ $$\frac{\partial E}{\partial B} = \sum_{t=1}^{T} \begin{bmatrix} \sum_{k=1}^{h}\delta_{k1}^{(t)} & \sum_{k=1}^{h}\delta_{k2}^{(t)} & \cdots & \sum_{k=1}^{h}\delta_{kn}^{(t)} \\ \sum_{k=1}^{h}\delta_{k1}^{(t)} & \sum_{k=1}^{h}\delta_{k2}^{(t)} & \cdots & \sum_{k=1}^{h}\delta_{kn}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{h}\delta_{k1}^{(t)} & \sum_{k=1}^{h}\delta_{k2}^{(t)} & \cdots & \sum_{k=1}^{h}\delta_{kn}^{(t)} \end{bmatrix} \quad (14.21)$$
p.212, 14.9.2	出力層は全結合層となります．この出力層において，活性化関数を f ，重みを W ，バイアスを B ，入力を X ，そして出力を Z と表すこととします．すると，順伝播は以下の式で表現できます．	出力層は全結合層となります．この出力層において，活性化関数を f ，重みを W ，バイアスを B ，入力を X ，そして出力を Z と表すこととします．通常，f, W, B は RNN 層のそれとは違うものです．同じ記号を使っていますが，異なるものであることに注意してください．このように，RNN 層と同じ記号を使うことで，プログラムの実装時にコードの一貫性を保つとともに，記号の意味を解釈しやすくなります． すると，順伝播は以下の式で表現できます．

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`

	誤	正

p.213, ソースコード 14.5	`n_samples = 【ソースコード 14.1 と同じ】 # サンプル数`
	`input_data = 【ソースコード 14.1 と同じ】 # 入力`
	`correct_data = 【ソースコード 14.1 と同じ】 # 正解`
	`for i in range(0, n_samples):`
	`  input_data[i] = 【ソースコード 14.1 と同じ】`
	`  correct_data[i] = 【ソースコード 14.1 と同じ】 # 正解は入力よりも一つ後`
		`n_samples = 【ソースコード 14.1 と同様】 # サンプル数`
		`input_data = 【ソースコード 14.1 と同様】 # 入力`
		`correct_data = 【ソースコード 14.1 と同様】 # 正解`
		`for i in range(0, n_samples):`
		`  input_data[i] = 【ソースコード 14.1 と同様】`
		`  correct_data[i] = 【ソースコード 14.1 と同様】 # 正解は入力よりも一つ後`

p.214, ソースコード 14.5 の 71 行目	`delta = self.y - t # 出力の誤差 (14.25) の実装`	`delta = self.y - t # 出力の誤差 (14.29) の実装`