3章: 分類問題 ー 機械学習ライブラリscikit-learnの活用

3.1(p.52) ~ 3.3(p.75)

3.1　分類アルゴリズムの選択

- あらゆるシナリオに万能な分類器というものは存在しない
  - 特徴量やサンプルの個数，データベースでのノイズの量，クラスの線形分離可能性によって分類性能が変わる

  →複数の学習アルゴリズムの性能を比較し，個々の問題に最適なモデルを選択することが常に推奨される

- 機械学習のアルゴリズムのトレーニング
  - 分類器の性能（予測力と計算能力）は学習に利用可能なデータに大きく依存する
  - 機械学習のアルゴリズムのトレーニング

    1. 特徴量の選択
    2. 性能指標の選択 （何を評価対象にするか，正解率など）
    3. 分類器と最適化アルゴリズムの選択
    4. モデルの性能評価
    5. アルゴリズムの調整

3.2　scikit-learn活用

- 非常に最適化されたさまざまな分類アルゴリズムの実装に対してユーザーフレンドリなインターフェイスを組み合わせたAPI
- scikit-learnライブラリは学習アルゴリズムだけでなくデータの前処理やモデルの調整や評価を行うための関数も取り揃えている
- パーセプトロンモデルのトレーニングをやってみる
  - Iris データセットを使用
    - アルゴリズムのテストや実験によく使用され，単純ながらも人気の高いデータセット
    - scikit-learnにすでに用意されている
  - 可視化のためにIrisデータセットの特微量を2つだけ使用する
    - 150個のサンプルの「花びらの長さ」と「花びらの幅」を特徴行列Xに代入し，対応する品種のクラスラベルをベクトル yに代入する

3.2　scikit-learn活用

- パーセプトロンモデルのトレーニングをやってみる
  - 可視化のためにIrisデータセットの特微量を2つだけ使用する
    - 150個のサンプルの「花びらの長さ」と「花びらの幅」を特徴行列xに代入し，対応する品種のクラスラベルをベクトル yに代入する

```
>>> from sklearn import datasets
>>> import numpy as np
>>> # Iris データセットをロード
>>> iris = datasets.load_iris()
>>> # 3、4 列目の特徴量を抽出
>>> X = iris.data[:, [2, 3]]
>>> # クラスラベルを取得
>>> y = iris.target
>>> # 一意なクラスラベルを出力
>>> print('Class labels:', np.unique(y))
Class labels: [0 1 2]
```

- np.unique(y)は，iris.targetに格納されている一意なクラスラベルを 3つ返している
- アヤメの花のクラス名である Iris-setosa, Iris-versicolor, Iris-virginicaがすでに整数(0,1,2)として格納されていることがわかる
  - 技術的なミスを回避し，メモリ消費を抑えて計算性能を向上させるために整数のラベルを使用することが推奨される
    （ほとんどの機械学習ライブラリではクラスラベルを整数として符号化するのが慣例となっている）

3.2　scikit-learn活用

- トレーニングしたモデルの性能を未知のデータで評価するためにデータセットをトレーニングデータセットとテストデータセットに分割する
- train_test_split関数（scikit-learnのmodel_selectionモジュール内）
  - x配列と y 配列を 30%のテストデータ(45個のサンプル)と70%のトレーニングデータ(105個のサンプル) にランダムに分割している
  - データセットを分割する前にトレーニングセットを内部でシャッフルする
  - そのままだとクラス0とクラス1のサンプルがトレーニングセットに追加され，テストセットがクラス2の45個のサンプルで構成されることになる

```
>>> from sklearn.model_selection import train_test_split
>>> # トレーニングデータとテストデータに分割
>>> # 全体の 30% をテストデータにする
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.3, random_state=1, stratify=y)
```

3.2　scikit-learn活用

- random_stateパラメータ
  - 内部の擬似乱数生成器に固定の乱数シード(random_ state=1)を指定している
  - データセットを分割する前のシャッフルに使用される
  - random_stateパラメータに固定の値を指定すると再現可能な結果が得られるようになる
- stratify=yを指定し，組み込み機能としてサポートされている層化サンプリングを利用している
  - この場合の層化サンプリングはtrain_test_split関数から返されるトレーニングサブセットとテストサブセットに含まれているクラスラベルの比率が入力データセットと同じであることを意味する
  - NumPyのbincount関数
    配列内の各値の出現回数を数える機能を提供する
    実際に上記を確認したい場合は，この関数を使用できる

```
>>> from sklearn.model_selection import train_test_split
>>> # トレーニングデータとテストデータに分割
>>> # 全体の30%をテストデータにする
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.3, random_state=1, stratify=y)
```

モデル評価の前処理

□特徴量のスケーリング

- Scikit-learnのpreprocessingモジュールの StandardScalerクラスを使って特徴量を標準化
- 特徴量の単位がそれぞれ異なる場合，値の大きな特徴量に引っ張られて値の小さな特徴量の影響が小さくなることがあるため

```
>>> from sklearn.preprocessing import StandardScaler
>>> sc = StandardScaler()
>>> # トレーニングデータの平均と標準偏差を計算
>>> sc.fit(X_train)
>>> # 平均と標準偏差を用いて標準化
>>> X_train_std = sc.transform(X_train)
>>> X_test_std = sc.transform(X_test)
```

- StandardScalerクラスをpreprocessingモジュールから読み込み新しいインスタンスを変数scに代入
- fitメソッドを呼び出し，トレーニングデータから特徴量ごとにパラメータμ(平均値)とσ(標準偏差)を推定
- transformメソッドを呼び出し，推定されたパラメータμとσを使ってトレーニングデータを標準化
- トレーニングデータセットとテストデータセットの値を相互に比較できるようにするためテストデータを標準化する時に同じスケーリングパラメータを使用

モデル評価の前処理

- linear_modelモジュールからPerceptronクラスを読み込んだ後Perceptronの新しいインスタンスを初期化し、fitメソッドを使ってモデルをトレーニングしている.
- この場合，パラメータ eta0はパーセプトロンの実装で使用した学習率 etaに相当するパラメータn_iterはエポック数(データセットのトレーニング回数)を定義する.

```
>>> from sklearn.preprocessing import StandardScaler
>>> sc = StandardScaler()
>>> # トレーニングデータの平均と標準偏差を計算
>>> sc.fit(X_train)
>>> # 平均と標準偏差を用いて標準化
>>> X_train_std = sc.transform(X_train)
>>> X_test_std = sc.transform(X_test)
```

モデル評価

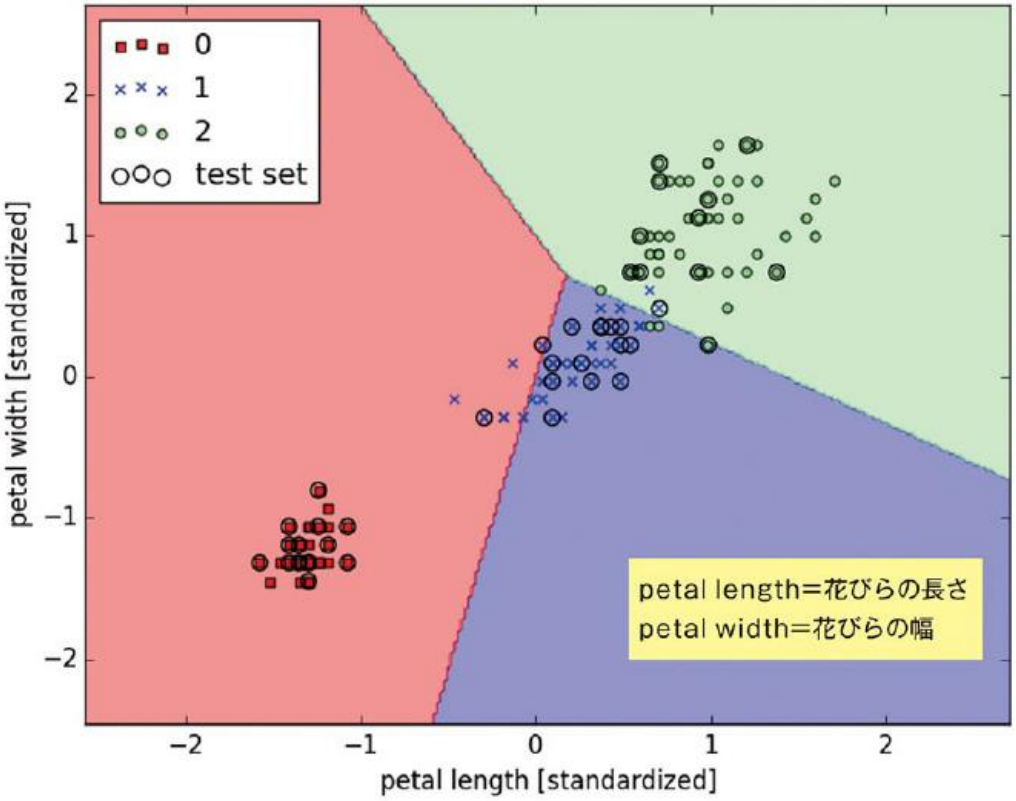
- 適切な学習率を割り出すには，ある程度の実験が必要
  - 学習率が高すぎる

    →アルゴリズムはコストの大局的な最小値を飛び越える
  - 学習率が低すぎる

    →アルゴリズムの収束に必要なエポックが増える
    大きなデータセットでは学習効率が低下することがある
- トレーニングデータをモデルに適合
  - トレーニングデータセットの並べ替え時にrandom_stateパラメータを設定
    - 何度実行しても同じように分離され，結果に再現性を持たせる
- 評価
  - テストデータの目的変数をy_test，predictメソッドを使い予測データをy_predとして比較，正解率を計算，評価

パーセプトロンモデルの識別を可視化

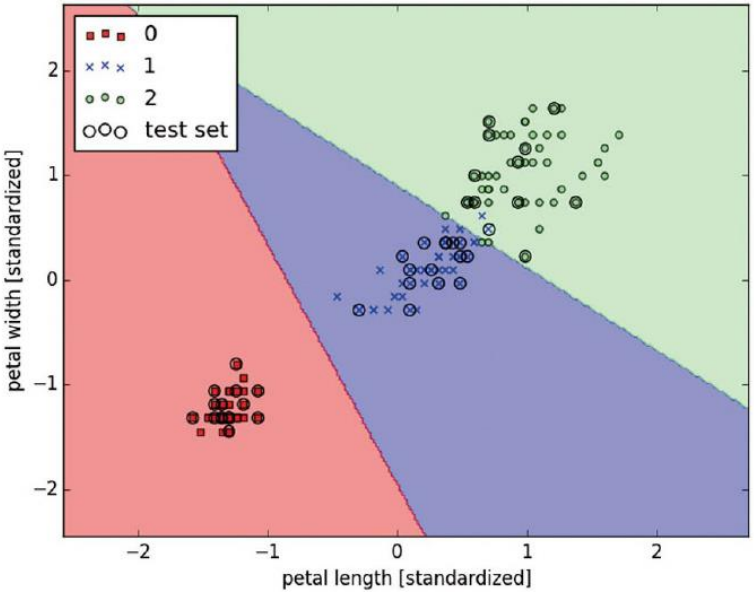
- さまざまな品種のサンプルをどの程度識別できるのかを可視化



- 完全な線形分離が不可能なデータセットでは，
  パーセプトロンアルゴリズムは決して収束しない

3.3 ロジスティック回帰

- 分類モデルの一つ（回帰ではない）
  - ある商品を買うか買わないか，ある会社が倒産するかしないか といった　カテゴリのデータを扱うアルゴリズム
  - 予測の信頼度を扱う（活性化関数を使う）
- 線形分離可能なクラスに対してのみ高い性能が発揮される
- 本章で取り上げるロジスティック回帰モデルは二値分類のための線形モデルでもあり，一対他（OvR）手法に基づいて多クラス分類モデルとして拡張できる



ロジスティック回帰

- 線形分離可能なクラスに対して非常に実装しやすく，高い性能が発揮される
- 産業界において最も広く使用されている分類のアルゴリズムの1つ

- オッズ比：正事象（予測したい事象）の起こりやすさを表すもの
  - 正事象をクラスラベル y=1 としロジット(logit)関数を定義
  - 0よりも大きく1よりも小さい範囲の入力値を受け取りそれらを実数の全範囲の値に変換する

$$\text{logit}(p) = \log \frac{p}{(1-p)}$$

  - 対数表記

$$\text{logit}\left(p\left(y=1 \mid \boldsymbol{x}\right)\right) = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

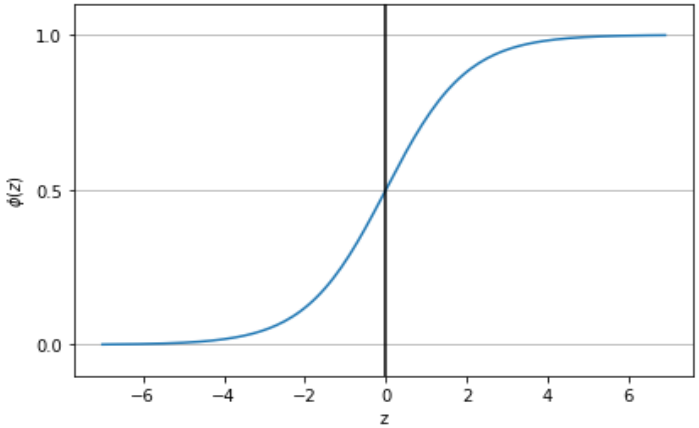
ロジスティック回帰

- 目的：サンプルが特定のクラスに属している確率を予測すること
  - ロジスティックシグモイド(logisticsigmoid)関数
    - ロジット関数の逆関数
    - シグモイド関数とも呼ばれる

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

- z は総入力（重みとサンプルの特徴量との線形結合）

$$z = \boldsymbol{w}^T \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m$$



シグモイド関数

e^{-z}はzの値が大きい場合は非常に小さいため
zが無限大に向かう場合(z→ ∞)は$\phi(z)$が1に近づくことがわかる
同様に$z \to -\infty$では分母が徐々に大きくなるため $\phi(z)$は0に向かう
よってこのシグモイド関数は入力として実数値を受け取り、$\phi(z) = 0.5$を切片としてそれらの入力値を$[0,1]$の範囲の値に変換する

ADALINEとロジスティック回帰の違い

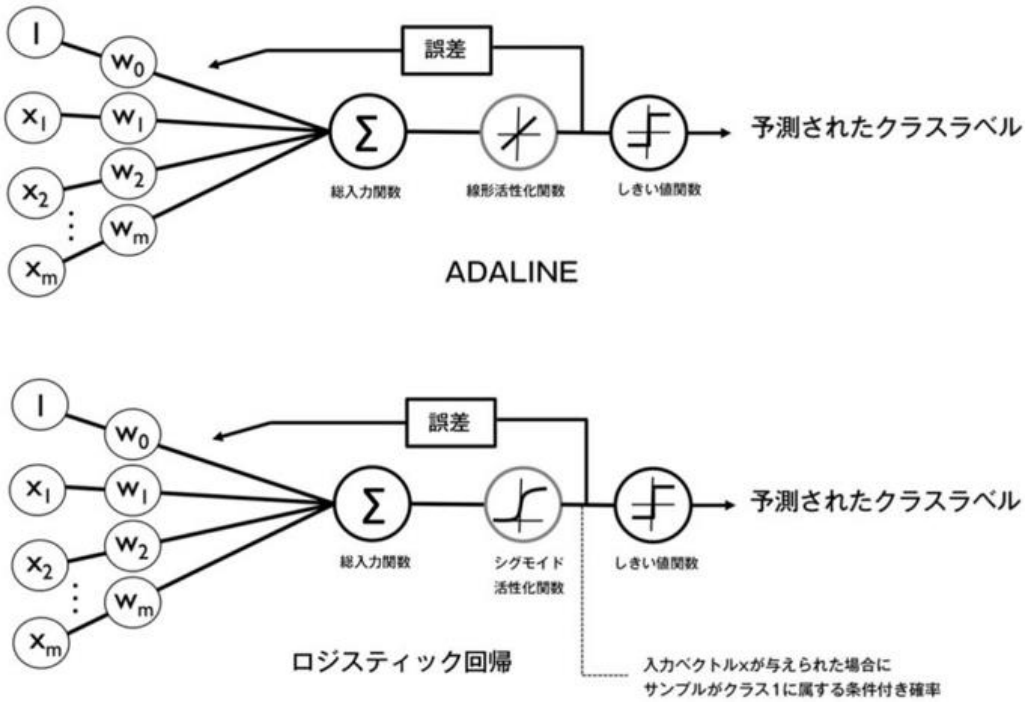
□活性化関数

- ADALINE:

$$\phi(z) = z$$

- ロジスティック回帰:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



ADALINEとロジスティック回帰の違い

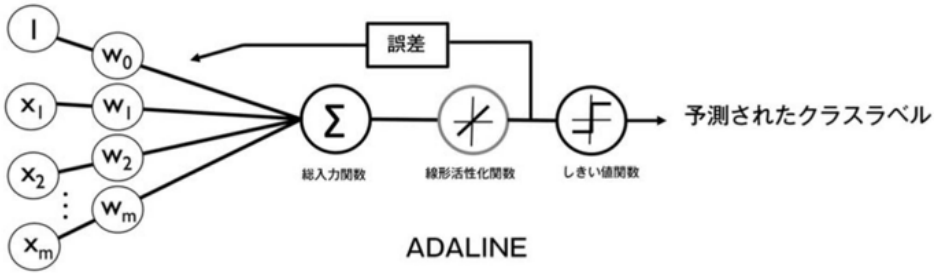
- 特徴量 x を重み wでパラメータ化
  サンプルがクラス1に属している確率は

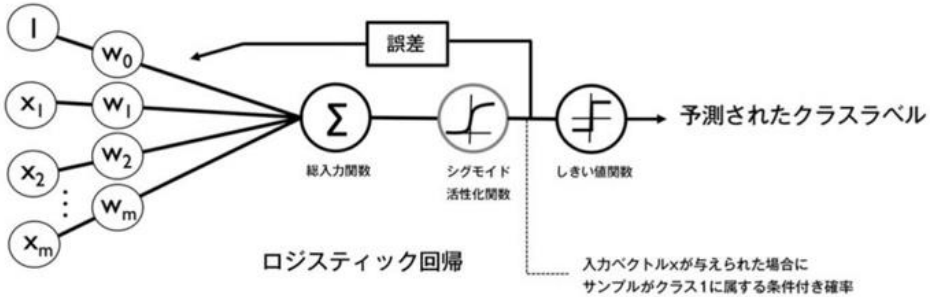
$$\phi(z) = P(y = 1|x; w)$$

  - サンプルに対して $\phi(z) = 0.8$が算出される場合
    このサンプルが品種 Iris-Versicolorである確率が 80%であること を意味する
    このサンプルが品種 Iris-Setosaである確率は

$$P(y = 0|x; w) = 1 - P(y = 0|x; w) = 0.2$$

    つまり20%として計算されるあとは，しきい値関数を使用することで予測された確率を二値の成果指標に変換すればよい





3.3.2 ロジスティック関数の重みの学習

- ロジスティック回帰モデルの構築時に最大化したい尤度（ゆうど: 結果から見た条件のもっともらしさ)を定義
  - 尤度$L(w)$:

$$L(\boldsymbol{w}) = P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{w}) = \prod_{i=1}^{n} P\left(y^{(i)} \mid x^{(i)}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \left(\phi\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \phi\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

  - 対数尤度$l(w)$:

$$l(\boldsymbol{w}) = \log L(\boldsymbol{w}) = \sum_{i=1}^{n} \left[y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right)\right]$$

- コスト関数$J(w)$として対数尤度を最小化できるように書き換え

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left[-y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right)\right]$$

  - サンプルが1つのトレーニングデータに対して計算されるコスト

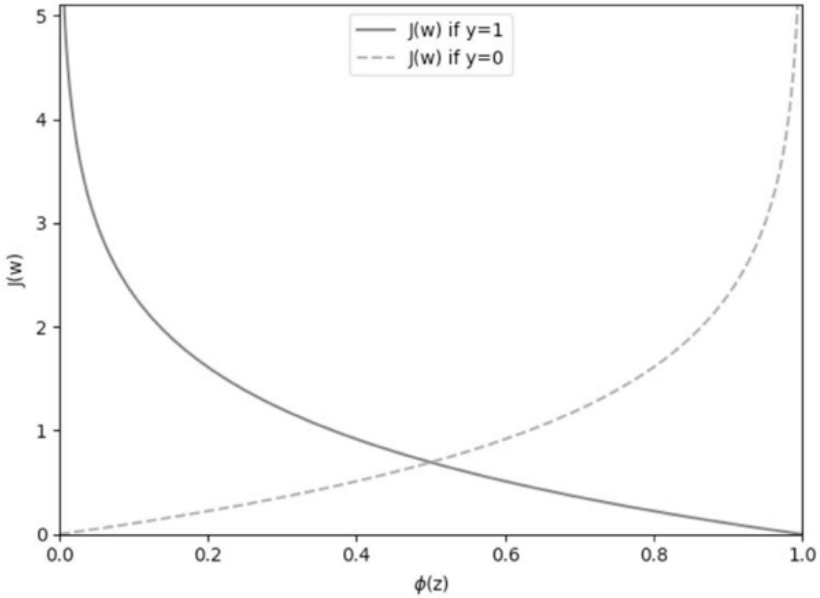
$$J(\phi(z), y; \boldsymbol{w}) = -y \log(\phi(z)) - (1 - y) \log(1 - \phi(z))$$

$$J(\phi(z), y; \boldsymbol{w}) = \begin{cases} -\log(\phi(z)) & (y = 1) \\ -\log(1 - \phi(z)) & (y = 0) \end{cases}$$

$y = 0$であれば1つ目の項が0になり
$y = 1$であれば2つ目の項が1になる

ロジスティック関数の重みの学習(p.65)

- x軸の範囲$[0,1]$ でシグモイド活性化関数を表しており
  y軸で関連するロジスティック回帰のコストを表している



- サンプルがクラス1に属していることを正しく予測した場合はコストが0に近づく
- $y = 0$である(サンプルがクラス0に属している)ことを正しく予測した場合はy軸のコストも0に近づく
- 予測が間違っていた場合コストは無限大に向かう

→ 予測を間違えたら徐々にコストを引き上げることでペナルティを科す

3.3.3 ADALINE実装をロジスティック回帰のアルゴリズムに変換する

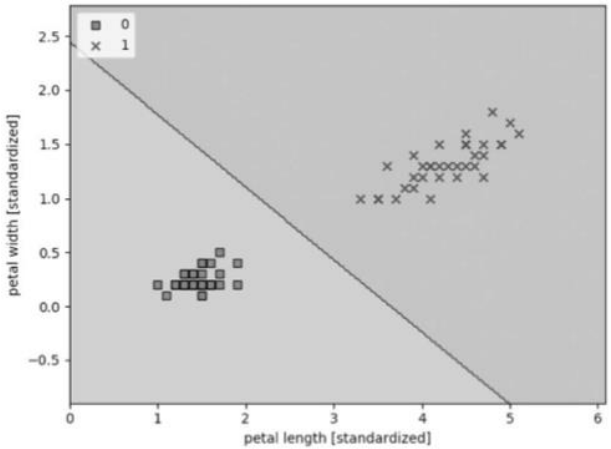
- ロジスティック回帰を独自に実装する場合は前章のADALINE実装のコスト関数$J(w)$を新しいコスト関数に置き換える

$$J(\boldsymbol{w}) = -\sum_i \left[y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right) \right]$$

トレーニングサンプルを分類するコストがエポックごとに計算される

- 線形活性化関数をシグモイド活性化関数に置き換える
- しきい値関数を書き換え，-1と 1の代わりにクラスラベル0と1を返すようにする

- ロジスティック回帰モデルを適合させるにあたり，そのモデルがうまくいくのは二値分類タスクに限られる
  - 今回は Iris-Setosa(クラス 0)と Iris-Versicolor(クラス 1)のみを考慮



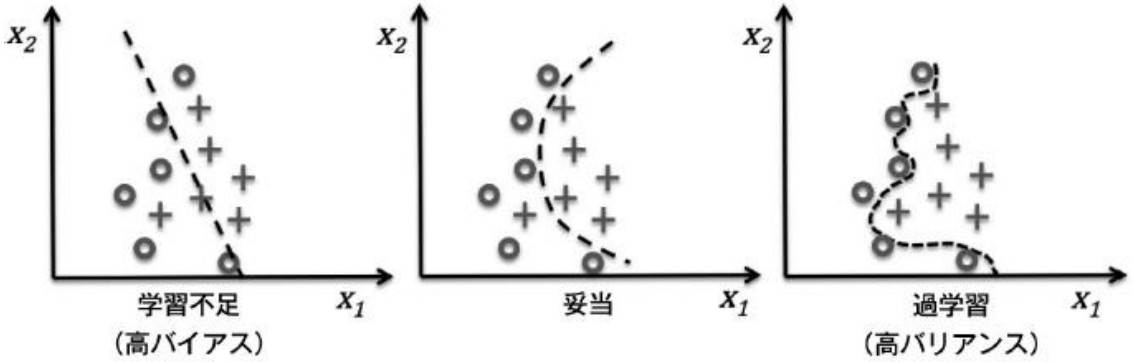
scikit-learnを使ったロジスティック回帰モデルのトレーニング

- 実際にコードを見ながら説明（p69~）

過学習への対処

- 過学習：
  トレーニングデータセットでは機能するモデルが
  未知のデータ（テストデータセット）ではうまく汎化されない問題

- 原因
  - パラメータの数が多すぎるために，データに対してモデルが
    複雑になりすぎる　→高バリアンス
  - データセットのパターンを捕捉するにはモデルが単純で，
    未知のデータに対する性能が低い　→高バイアス



- 正則化に基づいてモデルの複雑さを調整する
  - 正則化は共線性を処理する手法
  - 極端なパラメータの重みにペナルティを科すための追加情報(バイアス)を導入する

正則化

L2正則化

$$\frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

λ:正則化パラメータ

正則化の適用

$$J(\boldsymbol{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi\left(z^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)\right] + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

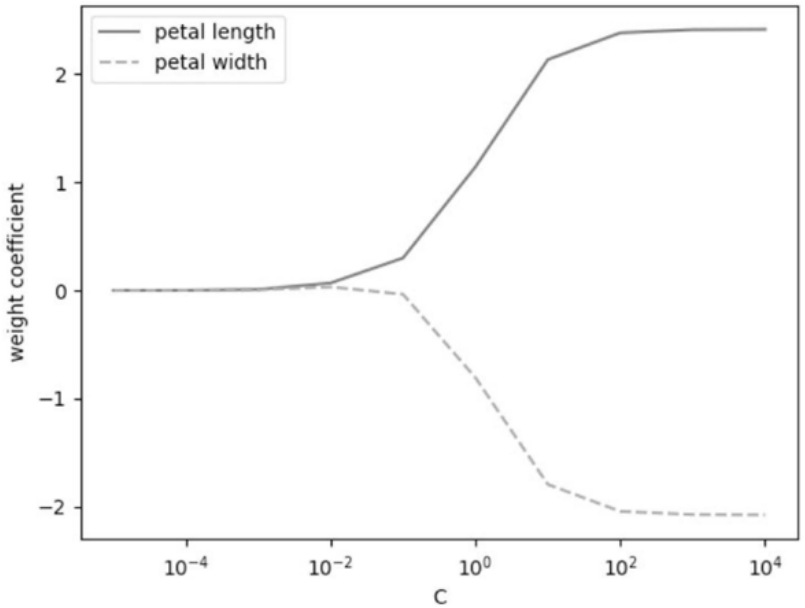
- ロジスティック回帰で定義したコスト関数に対して重みを小さくするための正則化の項を追加する
- Scikit-learnのLogisticRegressionクラスに実装されているパラメータCはλの逆数であるためロジスティック回帰の正則化されたコスト関数は

$$J(\boldsymbol{w}) = C\sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi\left(z^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)\right] + \frac{1}{2}\|\boldsymbol{w}\|^2$$

→逆正則化パラメータC正則化パラメータλの逆数)の値を減らすことは
正則化の強さを高めることを意味する

正則化

- 正則化の強さを可視化するため，2つの重み係数と逆正則化パラメー タとの関係をプロットする



- → パラメータCが減少し正則化の強さが増すと重み係数が0に近づいていく