機械学習入門

経済学部 BX584

ロジスティック回帰

線形関数とは（正確にはアフィン関数）

- d次元ベクトルを入力とし、スカラーを出力とする以下のような関数

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = b + w_1 x_1 + \cdots + w_D x_D$$

- $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$とも書ける。
  - ただし$\boldsymbol{w} = (b, w_1, \dots, w_D)$、および、$\boldsymbol{x} = (1, x_1, \dots, x_D)$とする。
- 訓練データを使って$(b, w_1, \dots, w_D)$を求めるのが機械学習
  - 損失関数が小さくなるように、これらのパラメータを計算機に動かさせる。

線形関数の使いみち

- 回帰 (regression)
  - 分析対象の特定の性質がひとつの数値で表される
  - $f(\boldsymbol{x})$がその数値を表すように線形関数を選ぶ
- 二値分類 (binary classification)
  - 分析対象が2つのグループに分かれている
  - $f(\boldsymbol{x})$の値によって2つのグループを区別できるように線型関数を選ぶ

線形関数を分類にどう使うか？

- 回帰では関数の出力値をそのまま使っていた
  - 出力値の範囲は$-\infty$から$+\infty$まで
- 出力値を2値分類に使うにはどうすればいいか？
  - <u>0から1の範囲に押し込める</u>
    - ロジスティック回帰
  - <u>符号を使う（正か負か）</u>
    - パーセプトロン、SVM

問題（出力が0か1かの二値）

- ある箱に、
  - 1という数値を入れたら0という数値が出てきてほしい。
  - 2という数値を入れたら0という数値が出てきてほしい。
  - 7という数値を入れたら1という数値が出てきてほしい。
  - 8という数値を入れたら1という数値が出てきてほしい。
- 箱の中でどういう計算をすればいいでしょうか？
  - こういうタイプの問題は、またいずれ。

線形回帰（単回帰の場合）

- モデル＝観測された現象を数式で表したもの。
- 関数のかたちを一次式に設定。一番簡単なので。

$$f(x) = ax + b$$

- そして「aとbをいくらにすればいいか？」という問題を解く。

回帰から分類へ

- 出力値は$-\infty$から$+\infty$まで

$$f(x) = ax + b$$

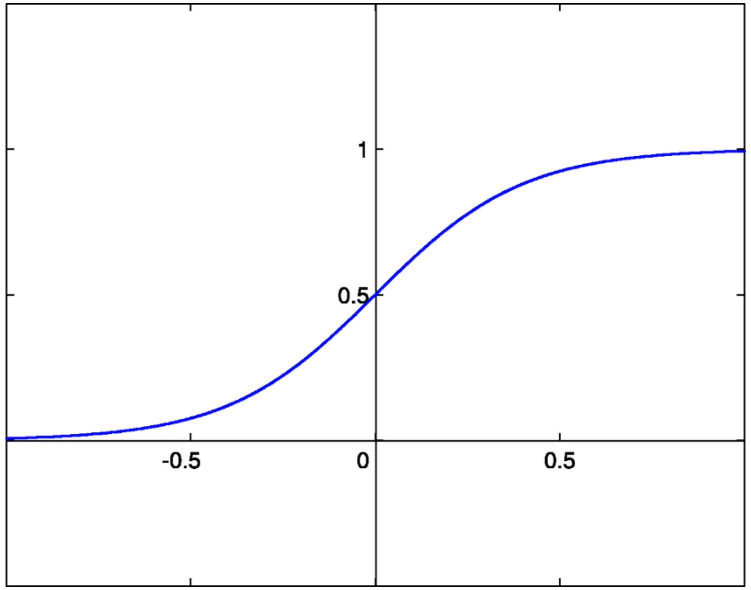
- 出力値を<u>0から1の範囲</u>に変えたい
  - 0.5より上か下かで1と0の2値に割り振る
  - <u>どうやって0から1の範囲に押し込める？</u>

標準シグモイド関数

- 標準シグモイド関数は、以下のように定義される

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$

- 定義域は$-\infty$から$+\infty$まで
- 値域は<u>0から1まで</u>
  - 使える！



標準シグモイド関数の性質

標準シグモイド関数の定義

$$\sigma(s) = \frac{1}{1+e^{-s}}$$

$$\sigma(-s) = \frac{1}{1+e^{s}} = \frac{e^{-s}}{e^{-s}+1} = 1 - \frac{1}{1+e^{-s}} = 1 - \sigma(s)$$

$$\frac{\partial \sigma(s)}{\partial s} = (1-\sigma(s))\times\sigma(s)$$

関数の値を使って
傾きを表せる、
ということ。

ロジスティック回帰

- 関数のかたちを次の式に設定する

$$g(x) \ = \frac{1}{1 + e^{-(ax+b)}}$$

- そして「aとbをいくらにすればいいか？」という問題を解く
  - 最急降下法を利用

ロジスティック回帰（説明変数が複数ある場合）

- 関数のかたちを次の式に設定する

$$g(\boldsymbol{x}) = \frac{1}{1 + e^{-f(\boldsymbol{x})}} = \frac{1}{1 + e^{-(b + w_1 x_1 + \cdots + w_D x_D)}}$$

- そして「$b, w_1 \ldots, w_D$をいくらにすればいいか？」という問題を解く
    - 最急降下法を利用

問題(出力が0か1かの二値)

- ある箱に、
  - 1という数値を入れたら0という数値が出てきてほしい。
  - 2という数値を入れたら0という数値が出てきてほしい。
  - 7という数値を入れたら1という数値が出てきてほしい。
  - 8という数値を入れたら1という数値が出てきてほしい。
- 箱の中でどういう計算をすればいいでしょうか?
  - こういうタイプの問題は、またいずれ。

訓練データが4つ（未知数は2つ）

$$\frac{1}{1+e^{-(a+b)}} \approx 0.0$$

$$\frac{1}{1+e^{-(2a+b)}} \approx 0.0$$

$$\frac{1}{1+e^{-(7a+b)}} \approx 1.0$$

$$\frac{1}{1+e^{-(8a+b)}} \approx 1.0$$

- 望ましい出力値（1または0）とのズレをどう測る？

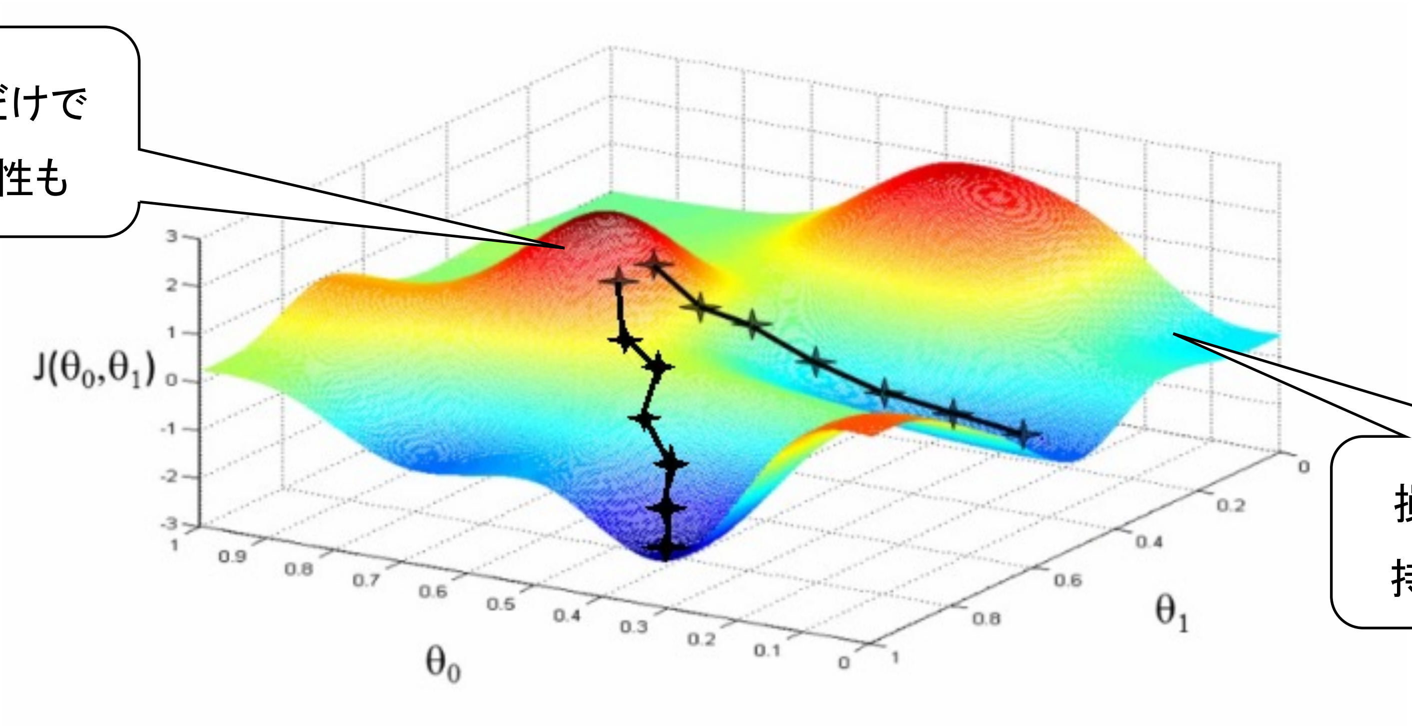
損失関数

- それを最小化すると良い線形関数が見つかるような関数
- 「損失」＝値が小さいほど良い
  - 無数にある$\boldsymbol{w}$の集合に、下るほど良いことがあるような地形を持ち込むのが、損失関数。
- 普通は勾配を利用して最小化する
  - 損失関数によって持ち込まれた地形の傾きを調べて（＝微分して）下っていく

損失関数は地形を持ち込む

- 坂を下る方向にパラメータを変化させる → 極小値に近づく

出発点が少し違うだけで
別の谷に行く可能性も



損失関数によって
持ち込まれた地形

https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

モデルと損失関数（ロジスティック回帰の場合）

- 線形関数＋ロジスティック関数をモデルとして採用
- そして以下の損失関数を最小化

$$L(\boldsymbol{w}) = \sum_{i=1}^{N} \{-t_i \log(g(\boldsymbol{x}_i)) - (1 - t_i)\log(1 - g(\boldsymbol{x}_i))\}$$

  - $\boldsymbol{x}_i = (\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,D})$は$i$番目の入力データ
  - t_iはi番目のデータの正解ラベル＝望ましい出力（0か1かのどちらか）

    例）スパムメールを$t = 0$、通常メールを$t = 1$で表すとする

- この損失関数は<u>クロスエントロピー</u>と呼ばれる

クロスエントロピー

- 個々の訓練データ$(\boldsymbol{x}_i, t_i)$に関するクロスエントロピー
  - t_iは正解ラベル（0または1）

$$-t_i \log\left(g(\boldsymbol{x}_i)\right) - (1 - t_i)\log(1 - g(\boldsymbol{x}_i))$$

  - $t_i = g(\boldsymbol{x}_i)$のとき（予測値が完全に当たっているとき）いくら？
  - $1 - t_i = g(\boldsymbol{x}_i)$のとき（予測値が間逆に外れているとき）いくら？

ロジスティック回帰の更新式（結果だけ示します）

ηは学習率

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} \leftarrow \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} - \eta\Big\{(1 - t_i)g(\boldsymbol{x}_i) - t_i\big(1 - g(\boldsymbol{x}_i)\big)\Big\}\begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{bmatrix}$$

- あれ？訓練データがひとつしか（$\boldsymbol{x}_i$しか）式に出てこないですけど・・・

確率的勾配降下法 (stochastic gradient descent, SGD)

- 損失関数が個々の訓練データに関する和として書ける時に使える
  - 線形回帰も、ロジスティック回帰も、そう。
- 最急降下法では、その和を偏微分して得られる勾配を使っていた
- 確率的勾配降下法では・・・
  1. パラメータを適当に初期化
  2. 訓練データをランダムにシャッフルする
  3. for $\boldsymbol{x}_i$ in シャッフルされた訓練データ:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla L_i$$

訓練データを
ひとつずつ取って来る

ミニバッチ学習

- バッチ式の勾配降下法と確率的勾配降下法(SGD)の中間
- 複数の訓練データを使ってパラメータを更新
  - 複数の訓練データの集まりをミニバッチと呼ぶ。
  - バッチは訓練データ全体。それよりかなり小さいので「ミニ」。
- ミニバッチに含まれる個々の訓練データに関する勾配の平均を使う
  - 学習率は適当に調整する。
- エポック＝全訓練データを一回使うこと

ロジスティック回帰の更新式

ηは学習率

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} \leftarrow \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} - \eta\Big\{(1-t_i)g(\boldsymbol{x}_i) - t_i\big(1-g(\boldsymbol{x}_i)\big)\Big\}\begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{bmatrix}$$

- この更新を、ランダムにシャッフルされた訓練データについて繰り返す

ロジスティック回帰の更新式（場合分け）

- $t_i = 1$の訓練データ

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} \leftarrow \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} + \eta\big(1 - g(\boldsymbol{x}_i)\big) \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{bmatrix}$$

- $t_i = 0$の訓練データ

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} \leftarrow \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} - \eta g(\boldsymbol{x}_i) \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{bmatrix}$$

ロジスティック回帰の更新式の導出

損失関数

$$L_i = -t_i \log\bigl(g(\boldsymbol{x}_i)\bigr) - (1 - t_i)\log(1 - g(\boldsymbol{x}_i))$$

$t_i = 0$の場合

$$\frac{\partial L_i}{\partial w_d} = \frac{\partial L_i}{\partial g(\boldsymbol{x}_i)} \frac{\partial g(\boldsymbol{x}_i)}{\partial w_d} = -\frac{\partial \log\bigl(1 - g(\boldsymbol{x}_i)\bigr)}{\partial g(\boldsymbol{x}_i)} \frac{\partial g(\boldsymbol{x}_i)}{\partial w_d} = \frac{1}{1 - g(\boldsymbol{x}_i)} \frac{\partial g(\boldsymbol{x}_i)}{\partial f(\boldsymbol{x}_i)} \frac{\partial f(\boldsymbol{x}_i)}{\partial w_d}$$

$$= \frac{1}{1 - g(\boldsymbol{x}_i)} \bigl(1 - g(\boldsymbol{x}_i)\bigr) g(\boldsymbol{x}_i) x_{i,d} = g(\boldsymbol{x}_i) x_{i,d}$$

$t_i = 1$の場合

$$\frac{\partial L_i}{\partial w_d} = \frac{\partial L_i}{\partial g(\boldsymbol{x}_i)} \frac{\partial g(\boldsymbol{x}_i)}{\partial w_d} = -\frac{\partial \log\bigl(g(\boldsymbol{x}_i)\bigr)}{\partial g(\boldsymbol{x}_i)} \frac{\partial g(\boldsymbol{x}_i)}{\partial w_d} = \frac{1}{g(\boldsymbol{x}_i)} \frac{\partial g(\boldsymbol{x}_i)}{\partial f(\boldsymbol{x}_i)} \frac{\partial f(\boldsymbol{x}_i)}{\partial w_d}$$

$$= \frac{1}{g(\boldsymbol{x}_i)} \bigl(1 - g(\boldsymbol{x}_i)\bigr) g(\boldsymbol{x}_i) x_{i,d} = (1 - g(\boldsymbol{x}_i)) x_{i,d}$$

scikit-learnのロジスティック回帰を使う

- scikit-learnのlinear_modelモジュールに含まれる
- 前回のLinearRegression()の代わりに

  LogisticRegression()を使えばいいだけ
  - Cというパラメータに注意
    - 正則化の弱さを表す（値が大きいほど正則化が弱くなる）
- 損失関数はバッチ最適化される
  - 確率的勾配降下法ではありません。
    - デフォルトではL-BFGSと呼ばれる手法で最適化の計算が行われるようです。

scikit-learnのロジスティック回帰における正則化

- 係数が大きくならないようにして過学習を防ぐ
- scikit-learnではオプションで指定
  - L1ノルムかL2ノルムかを引数penalityで指定
    - L1ノルム＝絶対値の和、L2ノルム＝２乗の和
  - 正則化の強さは引数Cで指定
    - 値が<u>小さい</u>ほど正則化が強くなる。
    - RidgeやLassoのalphaは逆で、値が<u>大きい</u>ほど正則化が強くなる。

- ここからはGoogle Colaboratoryのnotebookで説明します。