パターン認識
2019-05-21 授業分 レポート

37-196360　森田涼介

2019 年 5 月 27 日

宿題 1

(1)

拡張特徴ベクトル・拡張重みベクトル・正規化を実装し，パーセプトロン (バッチもしくはオンライン) を実装する。また，Linearly separable (linear)・Linearly non-separable (nonlinear)・Skewed linearly separable (slinear) の 3 種類のデータについてパーセプトロンを実行し，その振る舞いを調べる。なお，括弧書きで略称を示した。以降は略称を用いる。

拡張重みの更新式は次のようになる。

$$w^{(t+1)} = w^{(t)} + \rho \sum_{x \in \tilde{X}} x \tag{1}$$

ここで，t は更新回数，ρ は学習率，$\tilde{X}$ は識別誤りとなっているすべてのサンプルである。これを実装すればよい。

データは全て 2 カテゴリからなる二次元点群で，slinear のみ 500 サンプル，他の 2 つは 100 サンプルある。本来，学習の終了条件は点が全て正しいカテゴリに分類されたときであるが，この設定では収束しない場合があるので。更新を 100 回行ったときも学習を終了させる。学習率 $\rho = 0.1$ としたときの結果を以下の表 1 に，各データの点群と境界のプロットを図 1–3 に示す。

結果から，線形分離可能なデータ（linear）や，完全に線形分離するのは不可能だが大半がうまく分けられるようなデータ（slinear) に対しては，パーセプトロンはうまく機能することがわかる。一方，線形分離不可能なデータ（nonlinear）に対しては当然全く機能しない。また，学習中の境界の挙動を観察した結果を以下に記す。linear については，図 1 にある最終的に得られる境界を中心に，それとほぼ平行に減衰振動するような形で境界が移動していき，最終的に収束した。nonlinear については，境界がプロットの範囲中に現れては消えることを繰り返し続け，最後まで振動し続けていた。slinear については，まず図 3 の右上と左上を線対称に置くような位置に境界が引かれ，そこから徐々に立ち上がっていき，最終的に図の中心付近の点を（ある程度）うまく分類できるようになった。linear を減衰振動，nonlinear を振動と表現したことに倣うと，slinear は過減衰のような挙動であった。

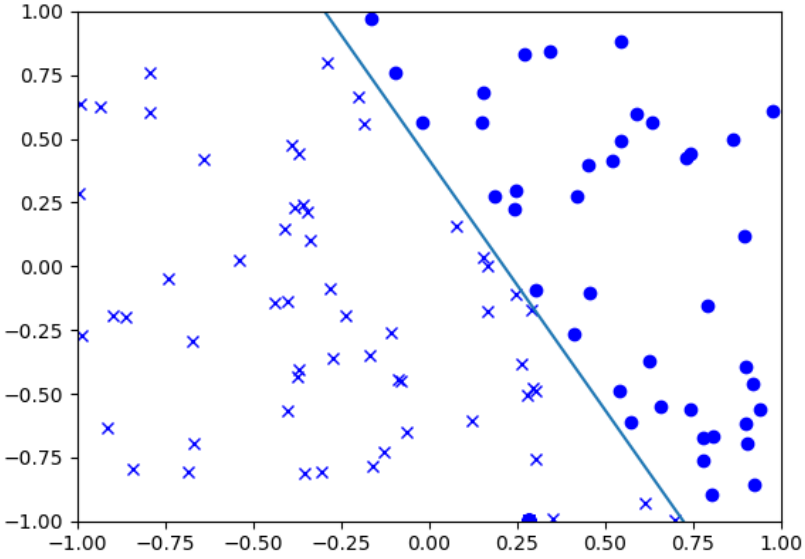

図 1: linear データに対してパーセプトロンを適用したときの点群と境界

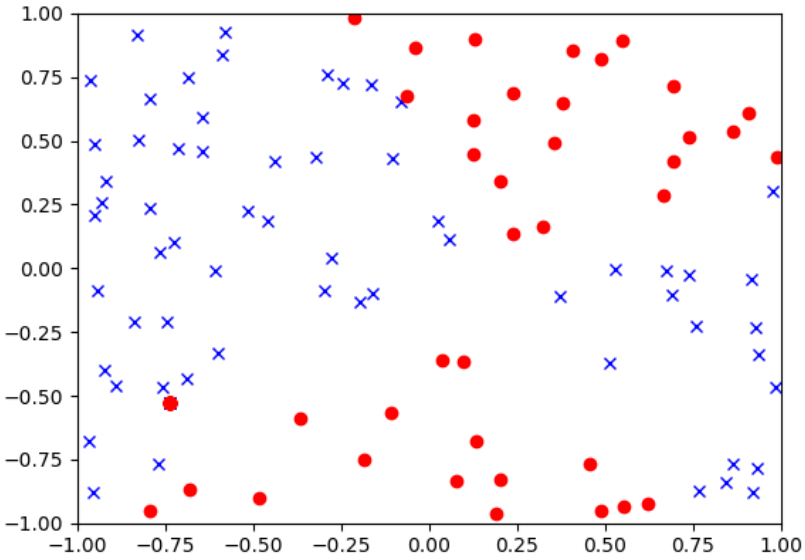

図 2: nonlinear データに対してパーセプトロンを適用したときの点群と境界

表 1: 結果

データの種類	w_0	w_1	w_2	識別誤り数	識別誤り率
linear	−1.002	4.727	2.401	0	0%
nonlinear	−4.002	1.870	−0.1073	38	38%
slinear	−0.2024	20.30	0.01964	2	0.4%

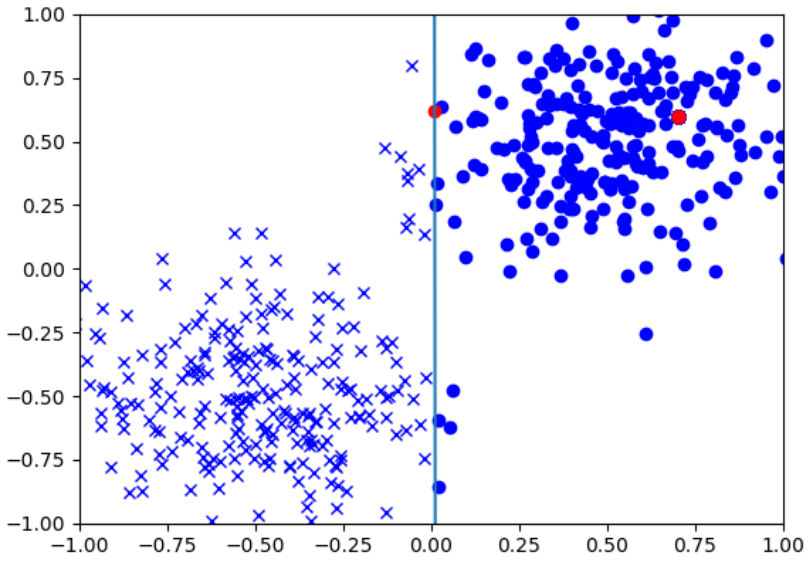

図 3: slinear データに対してパーセプトロンを適用したときの点群と境界

表 2: 結果

データの種類	学習率	w_0	w_1	w_2	識別誤り数	識別誤り率
linear	0.015	-0.2164	1.379	0.6243	6	6%
nonlinear	0.015	-0.2141	0.5494	0.03766	39	39%
slinear	0.001	-0.007554	1.035	0.7157	13	2.6%

(2)

MSE 法による識別器を実装し，3 種類のデータにより学習させる。また，その振る舞いについて調べる。

重みの解析解は次のようになる。

$$\boldsymbol{w} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}X^{\mathrm{T}}\boldsymbol{y} \tag{2}$$

また，LMS 法を用いるときの，拡張重みの更新式は次のようになる。

$$\hat{\boldsymbol{w}}^{(t+1)} = \hat{\boldsymbol{w}}^{(t)} - \rho\left(\hat{\boldsymbol{w}}^{\mathrm{T}}\hat{\boldsymbol{x}} - \boldsymbol{y}\right)\hat{\boldsymbol{x}} \tag{3}$$

学習の終了条件は重みの変化量のノルムが 1×10^{-5} 以下となったときとする。今回は学習率をデータごとに変えたため，それも結果にまとめて記すこととする。結果を以下の表 2 に，各データの点群と境界のプロットを図 4–6 に示す。

結果はパーセプトロンのときと同様，linear と slinear に対してはある程度対応でき，nonlinear に対してはうまくいかないというものになった。また，精度はパーセプトロンよりも劣る形となった。しかし，パーセプトロンと異なり，学習の終了条件が実現可能なものになっており，更新回数で切り上げないと収束せず無限ループに陥るということが起きないのは利点である。また，学習中の境界の挙動は，勾配降下法を用いているため，当然減衰振動のようなものとなった。

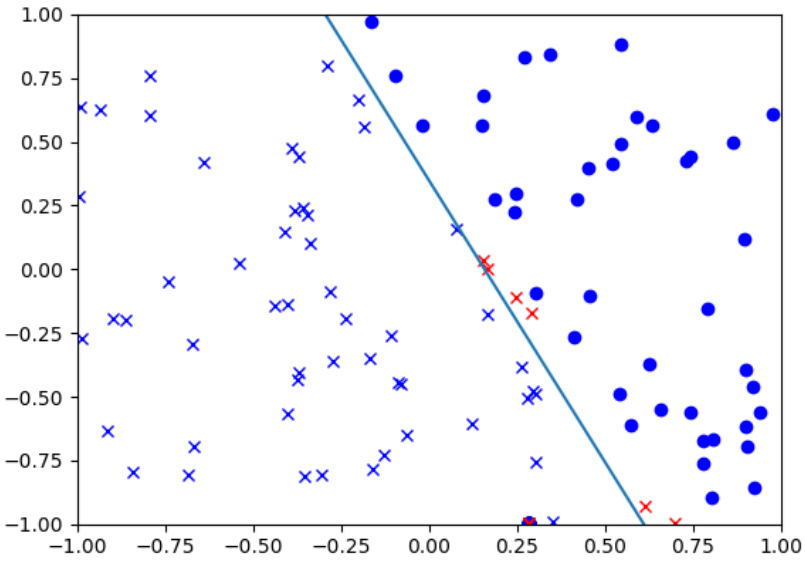

図 4: linear データに対して MSE 法を適用したときの点群と境界

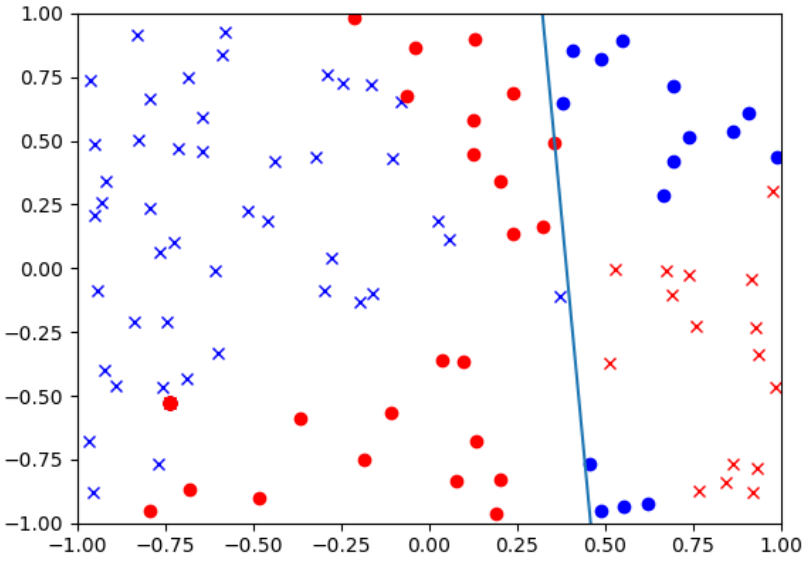

図 5: nonlinear データに対して MSE 法を適用したときの点群と境界

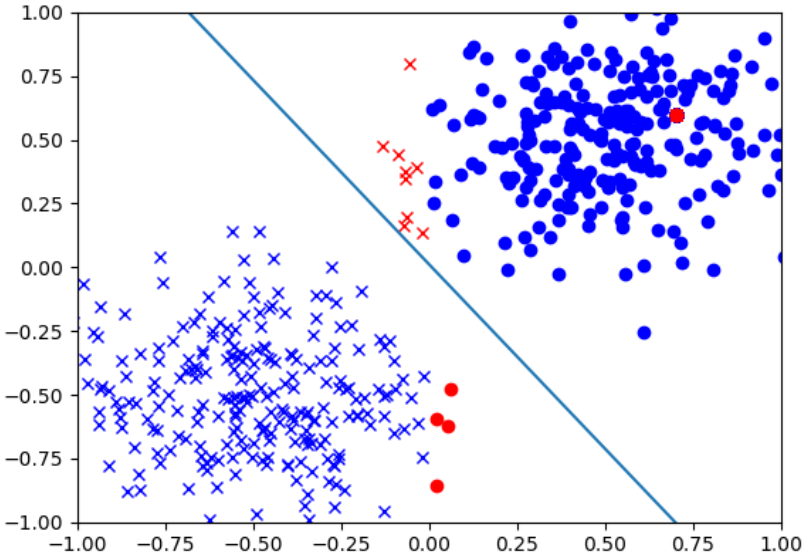

図 6: slinear データに対して MSE 法を適用したときの点群と境界

1　プログラム

実行環境と用いた言語・ライブラリを以下の表 3 に示す。

表 3: プログラムの実行環境

OS	:	Microsoft Windows 10 Pro (64bit)
CPU	:	Intel(R) Core(TM) i5-4300U
RAM	:	4.00 GB
使用言語	:	Python3.6
可視化	:	matplotlib ライブラリ

ソースコードは Listing 1 に示した。以下に簡単に各関数の説明を記す。

- load_data
  .mat ファイルからデータを取り出す。
- plot
  点群及び境界をプロットする。2 クラスを o と x で表し，分類結果が正しいものを青，誤っているものを赤で示す。
- perceptron
  パーセプトロンを用いて重みを求める関数。
- mse
  MSE 法を用いて重みを求める関数。LMS 法を用いる場合と解析的に求める場合とを使い分けられる。

Listings 1: assignment1.py

```
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat


def load_data(path):
    data = loadmat(path)
    #print(data.keys())
    x = data['x']
    l = data['l']
    n = data['n'][0, 0]
    d = data['d'][0, 0]
    return x, l, n, d


def plot(x, l, aw, neg):
    plt.clf()
```

```
    plt.xlim([-1, 1])
    plt.ylim([-1, 1])
    plt.plot(
        x[0, np.where(np.logical_and(l==1, ~neg))],
        x[1, np.where(np.logical_and(l==1, ~neg))],
        'bo',
        )
    plt.plot(
        x[0, np.where(np.logical_and(l==-1, ~neg))],
        x[1, np.where(np.logical_and(l==-1, ~neg))],
        'bx',
        )
    plt.plot(
        x[0, np.where(np.logical_and(l==1, neg))],
        x[1, np.where(np.logical_and(l==1, neg))],
        'ro',
        )
    plt.plot(
        x[0, np.where(np.logical_and(l==-1, neg))],
        x[1, np.where(np.logical_and(l==-1, neg))],
        'rx',
        )

    if abs(aw[1]) > abs(aw[2]):
        plt.plot(
            [-1, 1],
            [-(aw[0]-aw[1])/aw[2], -(aw[0]+aw[1])/aw[2]]
            )
    else:
        plt.plot(
            [-(aw[0]-aw[2])/aw[1], -(aw[0]+aw[2])/aw[1]],
            [-1, 1]
            )
    plt.waitforbuttonpress()


def perceptron(x, l, n, d, max_iter=100, fig_path=None,):
    # hyperparameter
    rho = 0.1

    # augmented vectors
    ax = np.concatenate((np.ones((1, n)), x))
    aw = (2*np.random.rand(d+1) - np.array([1, 1, 1]))[:, np.newaxis]

    # normalize
    ax[:, np.where(l == -1)] = -ax[:, np.where(l == -1)]

```

```
    # solve
    neg = ((ax.T.dot(aw)).T < 0)[-1]
    plt.figure()
    plt.ion()
    for n_iter in range(1, 1+int(max_iter)):
        # update
        aw += rho * ax[:, neg].sum(axis=1)[:, np.newaxis]

        # result
        neg = ((ax.T.dot(aw)).T < 0)[-1]
        n_left_neg = len(np.where(neg)[-1])

        print(f'#Iter: {n_iter}\t#Left Neg: {n_left_neg}')
        print('aw: ', aw.reshape(d+1))
        print()

        #plot(x, l, aw, neg)  # use when want to look move of boundary while learning

        # convergence condition
        if n_left_neg == 0:
            break

    # plot
    plot(x, l, aw, neg)
    if fig_path:
        plt.savefig(fig_path)
    plt.show()

    return aw


def mse(x, l, n, d, use_lms=False, max_iter=100, eps=1e-4, fig_path=None):
    # hyperparameter
    #rho = 0.015
    rho = 0.001

    # augmented vectors
    ax = np.concatenate((np.ones((1, n)), x))
    aw = (2*np.random.rand(d+1) - np.array([1, 1, 1]))[:, np.newaxis]

    plt.figure()
    plt.ion()

    # solve
    if not use_lms:
        pseudo_inverse_matrix = np.linalg.inv(ax.dot(ax.T)).dot(ax)
```

```
113         aw = pseudo_inverse_matrix.dot(l.T)
114     else:
115         aw_last = aw.copy()
116         for n_iter in range(1, 1+int(max_iter)):
117             # predict
118             g = (ax.T.dot(aw)).T
119 
120             # update
121             aw -= rho * (g - l).dot(ax.T).T
122 
123             # result
124             g[g > 0] = 1
125             g[g < 0] = -1
126             neg = (g != l)
127             n_wrong = len(np.where(neg)[-1])
128 
129             print(f'#Iter: {n_iter}\t#Wrong: {n_wrong}')
130             print('aw: ', aw.reshape(d+1))
131             print()
132 
133             plot(x, l, aw, neg)  # use when want to look move of boundary
        while learning
134 
135             # convergence condition
136             if np.linalg.norm(aw - aw_last) < eps:
137                 break
138             aw_last = aw.copy()
139 
140     # plot
141     g = (ax.T.dot(aw)).T
142     g[g > 0] = 1
143     g[g < 0] = -1
144     neg = (g != l)
145     plot(x, l, aw, neg)
146     if fig_path:
147         plt.savefig(fig_path)
148     plt.show()
149 
150     return aw
151 
152 
153 def main():
154     # settings
155     data_type = 'linear'
156     #data_type = 'nonlinear'
157     #data_type = 'slinear'
158     data_path = f'../../code/{data_type}-data.mat'
```

```
159     np.random.seed(0)
160
161     # load data
162     x, l, n, d = load_data(data_path)
163     print(f'Data Type: {data_type}  #Sample: {n}  #Dim: {d}\n')
164
165     # perceptron
166     #fig_path = f'../figures/assignment1_1_{data_type}_result.png'
167     #aw = perceptron(x, l, n, d, max_iter=100, fig_path=fig_path)
168
169     # MSE
170     fig_path = f'../figures/assignment1_2_{data_type}_result.png'
171     aw = mse(
172         x, l, n, d,
173         use_lms=False, max_iter=100, eps=1e-4,
174         fig_path=fig_path
175         )
176
177
178 if __name__ == '__main__':
179     main()
```