

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 1 に示す。

表 1: 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

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from scipy.io import loadmat
6
7
8  def load_data(path):
9      data = loadmat(path)
10     #print(data.keys())
11     x = data['x']
12     l = data['l']
13     n = data['n'][0, 0]
14     d = data['d'][0, 0]
15     return x, l, n, d
16
17
18 def plot(x, l, aw, neg):
19     plt.clf()
20     plt.xlim([-1, 1])
```

```

21     plt.ylim([-1, 1])
22     plt.plot(
23         x[0, np.where(np.logical_and(l==1, ~neg))],
24         x[1, np.where(np.logical_and(l==1, ~neg))],
25         'bo',
26     )
27     plt.plot(
28         x[0, np.where(np.logical_and(l==-1, ~neg))],
29         x[1, np.where(np.logical_and(l==-1, ~neg))],
30         'bx',
31     )
32     plt.plot(
33         x[0, np.where(np.logical_and(l==1, neg))],
34         x[1, np.where(np.logical_and(l==1, neg))],
35         'ro',
36     )
37     plt.plot(
38         x[0, np.where(np.logical_and(l==-1, neg))],
39         x[1, np.where(np.logical_and(l==-1, neg))],
40         'rx',
41     )
42
43     if abs(aw[1]) > abs(aw[2]):
44         plt.plot(
45             [-1, 1],
46             [-(aw[0]-aw[1])/aw[2], -(aw[0]+aw[1])/aw[2]]
47         )
48     else:
49         plt.plot(
50             [-(aw[0]-aw[2])/aw[1], -(aw[0]+aw[2])/aw[1]],
51             [-1, 1]
52         )
53     plt.waitforbuttonpress()
54
55
56 def perceptron(x, l, n, d, max_iter=100, fig_path=None):
57     # hyperparameter
58     rho = 0.1
59
60     # augmented vectors
61     ax = np.concatenate((np.ones((1, n)), x))
62     aw = (2*np.random.rand(d+1) - np.array([1, 1, 1]))[:, np.newaxis]
63
64     # normalize
65     ax[:, np.where(l == -1)] = -ax[:, np.where(l == -1)]
66
67     # solve

```

```

68     neg = ((ax.T.dot(aw)).T < 0)[-1]
69     plt.figure()
70     plt.ion()
71     for n_iter in range(1, 1+int(max_iter)):
72         # update
73         aw += rho * ax[:, neg].sum(axis=1)[:, np.newaxis]
74
75         # result
76         neg = ((ax.T.dot(aw)).T < 0)[-1]
77         n_left_neg = len(np.where(neg)[-1])
78
79         print(f'#Iter: {n_iter}\t#Left Neg: {n_left_neg}')
80         print('aw: ', aw.reshape(d+1))
81         print()
82
83         #plot(x, l, aw, neg) # use when want to look move of boundary while
learning
84
85         # convergence condition
86         if n_left_neg == 0:
87             break
88
89         # plot
90         plot(x, l, aw, neg)
91         if fig_path:
92             plt.savefig(fig_path)
93         plt.show()
94
95     return aw
96
97
98 def mse(x, l, n, d, use_lms=False, max_iter=100, eps=1e-4, fig_path=None):
99     # hyperparameter
100     #rho = 0.015
101     rho = 0.001
102
103     # augmented vectors
104     ax = np.concatenate((np.ones((1, n)), x))
105     aw = (2*np.random.rand(d+1) - np.array([1, 1, 1]))[:, np.newaxis]
106
107     plt.figure()
108     plt.ion()
109
110     # solve
111     if not use_lms:
112         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 != 1)
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
134         while learning
135
136             # convergence condition
137             if np.linalg.norm(aw - aw_last) < eps:
138                 break
139             aw_last = aw.copy()
140
141     # plot
142     g = (ax.T.dot(aw)).T
143     g[g > 0] = 1
144     g[g < 0] = -1
145     neg = (g != 1)
146     plot(x, l, aw, neg)
147     if fig_path:
148         plt.savefig(fig_path)
149     plt.show()
150
151     return aw
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()

```