マルチメディア信号解析
レポート 2

濱崎 直紀

令和元年 7 月 24 日

SVM(Support Vector Machine)

SVM(Support Vector Machine) は機械学習の一つである．決定境界に最も近いデータであるサポートベクトルと決定境界との距離（マージン）が最大となるように識別境界を決める手法である．SVM は線形，非線形の両者が存在するが，今回実装したものは線形であるので，以下 SVM は線形 SVM のことを指すものとする．

概要

基本的に SVM は 2 クラス識別であるので，ここでは 2 クラス識別問題を例に挙げる．

各クラス A，B の存在する境界である超平面をそれぞれ $\boldsymbol{w}^T\boldsymbol{x}_A + b = 1$，$\boldsymbol{w}^T\boldsymbol{x}_B + b = -1$ とおくと，マージン d は

$$d = \frac{|\boldsymbol{w}^T\boldsymbol{x}_{A\mathrm{or}B} + b|}{\sqrt{||\boldsymbol{w}^2||}} \tag{1}$$

$$= \frac{1}{||\boldsymbol{w}||} \tag{2}$$

となる．これを最大化することは，$||\boldsymbol{w}||$ を最小化することと等価である．

また，すべてのデータは境界の超平面の片側に存在することから

$$y_i(\boldsymbol{w}^T\boldsymbol{x} + b_i) \geq 1 \tag{3}$$

$$y_i = \begin{cases} 1 & (x_i \in A) \\ -1 & (x_i \in B) \end{cases}$$

という制約条件が得られる．

よって SVM の目標は「制約条件 $y_i(\boldsymbol{w}^T\boldsymbol{x} + b_i) \geq 1$ のもとで $||\boldsymbol{w}||$ を最小化する」ことである．しかし，実際には計算の都合上 $\frac{1}{2}||\boldsymbol{w}||^2$ を最小化する問題へと変換する．

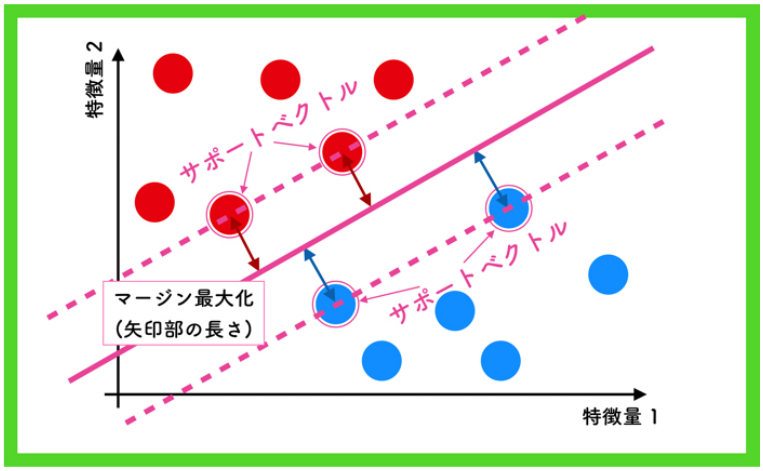

図 1　SVM のイメージ図

補足

SVM はデータの誤分類を許容しない「ハードマージン SVM」と許容する「ソフトマージン SVM」の 2 種類が存在する．ハードマージン SVM は，マージン内にデータが存在しないことを前提に考えられており，データがクラス間で重複している場合，識別境界を正確に求められないという問題が生じる可能性がある．それに対してソフトマージン SVM は，マージン内にデータが存在することやデータの誤分類をある程度許容する代わりに，そのデータにはペナルティを与えるという風に条件を少し緩めることで，多少の重複が存在しても識別できる可能性を上げることを可能としている．

では，実際にソフトマージン SVM において，どのようにしてペナルティを与えるのかということに関して簡単に述べる．まずスラック変数 ζ をデータごとに導入する．ζ はデータが正しく分類されてかつマージン内に存在しない場合は 0，正しく分類されているがマージン内に存在する場合は $0 < \zeta \leq 1$，誤って分類された場合は $\zeta > 1$ となる．そして，マージンの最大化に際して，このスラック変数 ζ をペナルティとして与える．式は以下のようになる．

$$\zeta_i = |y_i - (\boldsymbol{w}^T \boldsymbol{x} + b)| \tag{4}$$

$$C \sum_{i=1}^{N} \zeta_i + \frac{1}{2} ||\boldsymbol{w}||^2 \qquad (C > 0) \tag{5}$$

この式 (5) を最小化することが目標である．ここで C はペナルティの影響力を制御するパラメータであり，適当に設定する必要がある．誤分類が多くなると式 (5) の第 1 項が大きくなるので，最小化しようとするとなるべく誤分類が少なくなるようにパラメータ $\boldsymbol{w}$ を調整する．ちなみに，$C \to \infty$ とすると，ペナルティの影響力が ∞ になる，つまり誤分類が全く許容されなくなるので，ハードマージン SVM となる．

ただし，ソフトマージン SVM は与えるペナルティの大きさなどの調整を行う必要があり，少し複雑になることから，今回はハードマージン SVM の実装を行った．

アルゴリズム

実際に「制約条件 $y_i(\boldsymbol{w}^T \boldsymbol{x} + b_i) \geq 1$ のもとで $\frac{1}{2}||\boldsymbol{w}||^2$ を最小化する」という問題を解いていく．この問題は多変数関数の最適化問題なので，ラグランジュ乗数 α_i を導入することで，ラグランジュの未定乗数法に落とし込むことができる．

実際にラグランジュ関数は以下のようになる．

$$L(\boldsymbol{w}, b, \alpha) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{N} \alpha_i \{t_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1\} \tag{6}$$

極値条件より，式 (6) を $\boldsymbol{w}$，b でそれぞれ偏微分して 0 とおくと

$$\left.\frac{\partial L}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} = \hat{\boldsymbol{w}} - \sum_{i=1}^{N} \alpha_i t_i \boldsymbol{x}_i = 0 \tag{7}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N} \alpha_i t_i = 0 \tag{8}$$

また，最適性条件より最適解 $\hat{\boldsymbol{w}}$ に対して以下の条件を満たす必要がある．

$$\nabla L(\hat{\boldsymbol{w}}, b, \alpha) = \nabla(\frac{1}{2}||\hat{\boldsymbol{w}}||^2) - \sum_{i=1}^{N} \alpha_i \nabla\{t_i(\hat{\boldsymbol{w}}^T \boldsymbol{x}_i + b) - 1\} = 0 \tag{9}$$

この式から以下に示す 3 つの制約条件が得られる．

$$t_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \geq 0 \tag{10}$$

$$\alpha_i \geq 0 \tag{11}$$

$$\alpha_i\{t_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1\} = 0 \tag{12}$$

式 (10)，(11)，(12) はまとめて KKT 条件と呼ばれる．

式 (7) より最適解 $\hat{\boldsymbol{w}}$ は

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{N} \alpha_i t_i \boldsymbol{x}_i \tag{13}$$

式 (8)，(13) を式 (6) に代入すると

$$L(\hat{\boldsymbol{w}}, b, \alpha) = \frac{1}{2}||\hat{\boldsymbol{w}}||^2 - \hat{\boldsymbol{w}}^T \sum_{i=1}^{N} \alpha_i t_i \boldsymbol{x}_i - b \sum_{i=1}^{N} \alpha_i t_i + \sum_{i=1}^{N} \alpha_i \tag{14}$$

となり，これを α の関数と見ると

$$\tilde{L}(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}||\hat{\boldsymbol{w}}||^2 \tag{15}$$

$$= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j t_i t_j \boldsymbol{x}_i^T \boldsymbol{x}_j \tag{16}$$

となる．

$\boldsymbol{w}$ はすでに最適化されており，b は $\boldsymbol{w}$ と α から一意に定まるので，あとは α を最適化すればよい．主問題は $\frac{1}{2}||\boldsymbol{w}||^2$ の最小化より，式 (16) の最大化を行えばよい．α の制約条件としては式 (8)，(11) となる．

つまり解くべき問題は「制約条件 $\sum_{i=1}^{N} \alpha_i t_i = 0$，$\alpha_i \geq 0$ のもとで $\tilde{L}(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j t_i t_j \boldsymbol{x}_i^T \boldsymbol{x}_j$ を最大化する」ということになる．このような問題を双対問題と呼ぶ．

課題 1

多クラス（2 クラス以上）のサンプル（2 次元以上とする）を，各クラスに対して設定した正規分布などに基づく乱数の発生により生成した後，一部を学習サンプル，残りをテストサンプルとし，学習サンプルを用いて学習した識別器を用いてテストサンプルを分類するプログラムを作成せよ．

サンプル生成方法

本実験ではサンプルとして，2 クラスの 2 次元正規分布に基づくデータを作成した．手法としては 1 次元の正規分布データの作成を各基底において行うという非常にシンプルなものである．

設定条件

今回は 2 種類の設定によるデータセットで実験を行った．

- データセット A
  クラス 1 は平均 5，分散 2.5 の 2 次元正規分布，クラス 2 は平均-5，分散 2.5 の 2 次元正規分布に基づくデータの集合．
- データセット B
  クラス 1 は平均 3，分散 2.5 の 2 次元正規分布，クラス 2 は平均-3，分散 2.5 の 2 次元正規分布に基づくデータの集合．

実験結果

各データセットに対する識別結果は以下のようになった．

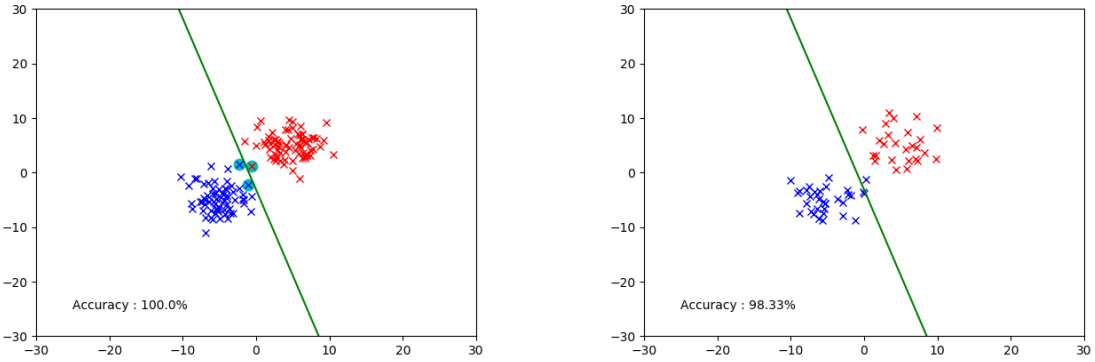

図 2　SVM によるデータセット A に対する識別結果（左：トレーニングデータ 140 個，右：テストデータ 60 個）

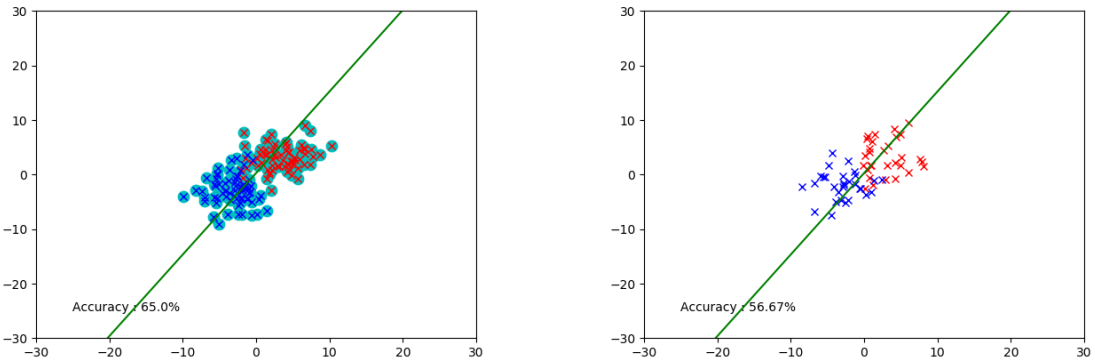

図 3　SVM によるデータセット B に対する識別結果（左：トレーニングデータ 140 個，右：テストデータ 60 個）

考察

実験設定から分かる通り，データセット B はデータセット A に比べてクラス間の距離がなく，重複している．前述した通り，今回実装した SVM はハードマージン SVM であるので，データセット B に対する識別は厳しいと予想した．

まずデータセット A に対する識別結果であるが，図 2 から分かるように，ある程度うまく識別されていると言える．

続いてデータセット B に対する識別結果であるが，図 3 から分かるように，識別精度は非常に低いものとなった．

結果としては予想通りなものとなり，ハードマージン SVM が重複のあるデータに非常に弱いことも見て取れる結果となった．

課題 2

課題 1 で作成したプログラムを応用し， 数字 0〜9 の学習用画像を用いて，手書き文字（モノクロ画像）を認識する識別器を作成し， 手書き文字を複数含む画像から各数字を検出するプログラムを作成せよ．

SVM における多クラス分類

SVM は基本的に 2 クラス分類の手法であるため，多クラス分類に用いるには工夫が必要となる．工夫の方法は幾つかあるが，その中でもよく用いられる”one-versus-the-rest”と”one-versus-one”について説明を行う．

- one-versus-the-rest

  ある特定のクラスに入るか，他の $k-1$ 個のクラスのどれかに入るかの 2 クラス識別器を作成して識別を行う．最終的な識別はそれらの識別結果による多数決で行う．識別器は k 個必要．

- one-versus-one

  ある特定のクラスに入るか，別の特定のクラスに入るかの 2 クラス識別器を作成して識別を行う．最終的な識別はそれらの識別結果による多数決で行う．識別器は $\frac{k(k-1)}{2}$ 個必要．

本実験では one-versus-one による識別を行った．

特徴量

特徴量として次元数削減のため，画像を縮小することを考えた．画像縮小を行わなかった場合と行った場合における MNIST のいづれか 2 つの数字の画像を SVM で識別した結果は以下の通りである．

Target	Accuracy	Target	Accuracy	Target	Accuracy	Target	Accuracy	Target	Accuracy
0-1	99.83%	1-2	98.88%	2-4	97.93%	3-7	97.87%	5-7	99.09%
0-2	98.70%	1-3	98.97%	2-5	96.89%	3-8	94.42%	5-8	94.37%
0-3	99.23%	1-4	99.40%	2-6	97.65%	3-9	97.64%	5-9	97.89%
0-4	99.55%	1-5	99.26%	2-7	97.87%	4-5	98.53%	6-7	99.86%
0-5	97.99%	1-6	99.82%	2-8	94.37%	4-6	98.68%	6-8	98.12%
0-6	98.40%	1-7	99.18%	2-9	98.31%	4-7	98.26%	6-9	99.73%
0-7	99.56%	1-8	97.54%	3-4	99.08%	4-8	98.92%	7-8	98.85%
0-8	98.61%	1-9	99.45%	3-5	93.30%	4-9	94.19%	7-9	92.50%
0-9	99.32%	2-3	95.65%	3-6	99.37%	5-6	96.96%	8-9	97.29%

Target	Accuracy	Target	Accuracy	Target	Accuracy	Target	Accuracy	Target	Accuracy
0-1	99.79%	1-2	98.17%	2-4	96.98%	3-7	97.61%	5-7	99.04%
0-2	98.12%	1-3	98.68%	2-5	96.69%	3-8	52.75%	5-8	53.09%
0-3	98.83%	1-4	99.32%	2-6	96.61%	3-9	97.27%	5-9	96.89%
0-4	99.41%	1-5	99.40%	2-7	97.61%	4-5	98.07%	6-7	99.73%
0-5	97.40%	1-6	99.64%	2-8	95.27%	4-6	98.84%	6-8	98.52%
0-6	98.45%	1-7	99.21%	2-9	97.85%	4-7	97.65%	6-9	99.54%
0-7	99.42%	1-8	96.23%	3-4	99.30%	4-8	98.48%	7-8	98.98%
0-8	97.95%	1-9	99.35%	3-5	64.79%	4-9	56.80%	7-9	51.31%
0-9	99.19%	2-3	55.05%	3-6	99.48%	5-6	97.22%	8-9	97.10%

表 1　各数字の画像におけるテストデータに対する識別精度（上：画像縮小なし，下：画像縮小あり）

実験には各数字画像の 20% をトレーニングデータに使用し，残りをテストデータとした．

この表から分かる通り，一部の数字画像において識別精度が著しく下がる結果となった．識別精度の下がった数字画像においては，画像縮小を行わなかった場合の識別精度においても低かったものが下がっており，比較的識別の難しい数字画像が縮小の影響を受けることで，識別することができなくなっていることが分かる．この結果を受けて，実験では画像縮小を行わずに進めることとした．

実験結果

最終的に識別結果の多数決で MNIST の数字画像を識別した結果は以下の通りである．

Target	Accuracy
0	99.0%
1	46.0%
2	76.0%
3	95.0%
4	92.0%
5	42.0%
6	95.0%
7	82.0%
8	96.0%
9	75.0%
All	79.8%

表 2　各数字画像における識別精度

実験では学習で使われていない各数字画像 100 枚ずつ，計 1000 枚を用いて識別精度を測った．

考察

表 2 から分かるように，最終的な識別精度としては 79.8% となったが，各数字ごとに着目すると，識別精度は数字ごとにばらつきがあることが分かる．特に”1”と”5”における識別精度が低くなっている．2 クラス分類の時点ではその傾向を読み取ることができなかったため，各 2 クラス識別器の識別精度とその識別結果の多数決による識別精度には必ずしも関係性があるわけではないと考えられる．

section*感想今回の実験ではハードマージン SVM の実装しか行えなかった．パラメータの調整さえできれば，ソフトマージン SVM の方が有用であるように見えるため，実装できていた場合の結果がどうなっていたのかが気になるところではある．また，SVM の多クラス分類に関しても，いくつかある手法のうちの 1 つしか実装できなかったので，他の手法を実装できなかったのは課題点として挙げられると思う．

付録 A　プログラム 1

ソースコード 1　課題 1 のデータ作成用プログラム

```
1    import random
2
3
4    def gauss(mu_x, sigma_x, mu_y, sigma_y, class_label, size):
5        data = []
6        label = []
7
8        for i in range(size):
9            elem = []
10           elem.append(random.gauss(mu_x, sigma_x))
11           elem.append(random.gauss(mu_y, sigma_y))
12           data.append(elem)
13           label.append(class_label)
14
15       return data, label
```

付録 B　プログラム 2

ソースコード 2　課題 1 における SVM 学習用プログラム

```
import cvxopt
import matplotlib.pyplot as plt
import numpy as np
import os
from sklearn.model_selection import train_test_split
import MakeData
os.chdir(os.path.dirname(os.path.abspath(__file__)))


# データを学習できる形に変換
def data_exchange(data1, data2):
    data = data1 + data2 # データを 1 つにまとめる
    label = label1 + label2 # ラベルを 1 つにまとめる
    data = np.array(data) # データ配列をnumpy 型に変換
    label = np.array(label) # ラベル配列をnumpy 型に変換
    train_data, test_data, train_label, test_label = train_test_split(data, label,
        test_size=0.3) # トレーニング用とテスト用に分ける

    return train_data, test_data, train_label, test_label


# SVM
class SVM:
    def __init__(self, data, label):
        self.data = data
        self.label = label


    # 線形カーネル
    def kernel(self, x, y):
        return np.dot(x, y)


    # ラグランジュ乗数を二次計画法で求める
    def Lagrange(self, n):
        K = np.zeros((n, n))
        for i in range(n):
            for j in range(n):
                K[i, j] = self.label[i] * self.label[j] * np.dot(self.data[i], self
                    .data[j])
```

```
39              Q = cvxopt.matrix(K)
40              p = cvxopt.matrix(-np.ones(n)) # -1がn 個の列ベクトル
41              G = cvxopt.matrix(np.diag([-1.0]*n)) # 対角成分が-1の (n × n)行列
42              h = cvxopt.matrix(np.zeros(n)) # 0がn 個の列ベクトル
43              A = cvxopt.matrix(self.label, (1,n)) # N 個の教師信号が要素の行ベクトル (1 × n)
44              b = cvxopt.matrix(0.0) # 定数 0.0
45              solution = cvxopt.solvers.qp(Q, p, G, h, A, b) # 二次計画法でラグランジュ乗
                    数alpha を求める
46
47              alpha = np.array(solution['x']).flatten()
48
49              return alpha
50
51
52          # サポートベクトルを抽出
53          def support_vector(self, alpha):
54              S = []
55              for i in range(len(alpha)):
56                  if alpha[i] >= 0.00001:
57                      S.append(i)
58
59              return S
60
61
62          # w を計算
63          def w_cal(self, S, alpha):
64              w = np.zeros(2)
65              for n in S:
66                  w += alpha[n] * self.label[n] * self.data[n]
67
68              return w
69
70
71          # b を計算
72          def b_cal(self, S, alpha):
73              _sum = 0
74              for n in S:
75                  tmp = 0
76                  for m in S:
77                      tmp += alpha[m] * self.label[m] * self.kernel(self.data[n], self.
                            data[m])
78                  _sum += (self.label[n] - tmp)
79              b = _sum / len(S)
80
81              return b
82
```

```
83
84          def main(self):
85              alpha = self.Lagrange(len(self.data)) # ラグランジュ乗数
86              S = self.support_vector(alpha) # サポートベクトル
87              w = self.w_cal(S, alpha) # w
88              b = self.b_cal(S, alpha) # b
89
90              return w, b, S
91
92
93      # ラベルごとにデータを分割
94      def data_split(data, label):
95          cls1 = []
96          cls2 = []
97          for i in range(len(data)):
98              if label[i] == 1:
99                  cls1.append(data[i])
100             elif label[i] == -1:
101                 cls2.append(data[i])
102
103         return cls1, cls2
104
105
106     def f(x, w, b):
107         return np.dot(w, x) + b
108
109
110     def f_plot(x1, w, b):
111         return -(w[0] / w[1]) * x1 - (b / w[1])
112
113
114     # 精度を計算
115     def accuracy(cls1, cls2, w, b):
116         num = len(cls1) + len(cls2)
117         c = 0
118         for i in cls1:
119             if f(i, w, b) >= 0:
120                 c += 1
121         for i in cls2:
122             if f(i, w, b) < 0:
123                 c += 1
124
125         acc = c / num
126
127         return acc
128
```

```
129
130     # 結果を描画
131     class Draw:
132         def __init__(self, data, cls1, cls2, x_min, x_max, y_min, y_max, w, b, S, acc,
                check):
133             self.data = data
134             self.cls1 = cls1
135             self.cls2 = cls2
136             self.x_min = x_min
137             self.x_max = x_max
138             self.y_min = y_min
139             self.y_max = y_max
140             self.w = w
141             self.b = b
142             self.S = S
143             self.acc = acc
144             self.check = check
145
146
147         # 結果を描画
148         def main(self):
149             # 訓練データを描画
150             x1, x2 = np.array(self.cls1).transpose()
151             plt.plot(x1, x2, 'rx')
152             x1, x2 = np.array(self.cls2).transpose()
153             plt.plot(x1, x2, 'bx')
154
155             # サポートベクトルを描画
156             if self.check == 'train':
157                 for n in self.S:
158                     plt.scatter(self.data[n,0], self.data[n,1], s=80, c='c', marker='o
                            ')
159
160             # 識別境界を描画
161             x1 = np.linspace(self.x_min, self.x_max, 1000)
162             x2 = [f_plot(x, self.w, self.b) for x in x1]
163             plt.plot(x1, x2, 'g-')
164
165             plt.xlim(self.x_min, self.x_max)
166             plt.ylim(self.y_min, self.y_max)
167             plt.text(-25, -25, 'Accuracy : {}%'.format(round(self.acc*100, 2)))
168             plt.show()
169
170
171     if __name__ == '__main__':
172         N1 = 100 # クラス 1 のデータ数
```

```
173         N2 = 100 # クラス２のデータ数
174
175         # データを作成
176         data1, label1 = MakeData.gauss(5, 2.5, 5, 2.5, 1.0, N1) # クラス１のデータとラ
                ベル
177         data2, label2 = MakeData.gauss(-5, 2.5, -5, 2.5, -1.0, N2) # クラス２のデータ
                とラベル
178         train_data, test_data, train_label, test_label = data_exchange(data1, data2)
179
180         # SVM
181         svm = SVM(train_data, train_label)
182         w, b, S = svm.main()
183
184         # トレーニングデータ､テストデータをラベル別に分割
185         cls1_train, cls2_train = data_split(train_data, train_label)
186         cls1_test, cls2_test = data_split(test_data, test_label)
187
188         # トレーニングデータ､テストデータのそれぞれに対して精度を計算
189         acc_train = accuracy(cls1_train, cls2_train, w, b)
190         acc_test = accuracy(cls1_test, cls2_test, w, b)
191         print('Accuracy for training data : {}'.format(acc_train))
192         print('Accuracy for test data : {}'.format(acc_test))
193
194         x_min = -30 # x の最小値（描画範囲）
195         x_max = 30 # x の最大値（描画範囲）
196         y_min = -30 # y の最小値（描画範囲）
197         y_max = 30 # y の最大値（描画範囲）
198
199         # トレーニングデータとテストデータに対する結果を描画
200         draw_train = Draw(train_data, cls1_train, cls2_train, x_min, x_max, y_min, y_max
                , w, b, S, acc_train, check='train')
201         draw_test = Draw(train_data, cls1_test, cls2_test, x_min, x_max, y_min, y_max, w
                , b, S, acc_test, check='test')
202         draw_train.main()
203         draw_test.main()
```

付録 C　プログラム 3

ソースコード 3　課題 2 の MNIST の数字画像識別における SVM 学習用プログラム

```
import cv2
import cvxopt
import numpy as np
import os
import pandas as pd
import time
from sklearn.model_selection import train_test_split
os.chdir(os.path.dirname(os.path.abspath(__file__)))


# データセットの作成
class DATASET:
    def __init__(self, num1, num2, num3):
        self.cls = num1
        self.num1 = num2
        self.num2 = num3


    def one_vs_the_rest(self):
        data = []
        label = []
        print('Loading data (positive:{})...'.format(self.cls), end='')
        for num in range(0, 10):
            for root, dirs, files in os.walk('train_img/{}'.format(num)):
                for f in files:
                    '''
                    # リサイズによる次元削減
                    img = cv2.imread('{}/{}'.format(root, f), 0)
                    dst = cv2.resize(img, dsize=None, fx=0.5, fy=0.5)
                    dst = dst.reshape(-1)
                    data.append(dst)
                    '''
                    data.append(cv2.imread('{}/{}'.format(root, f), 0).reshape(-1))
                         # 1次元配列に変換してdata に貯めていく
                    if num == self.cls:
                        label.append(1.0)
                    else:
                        label.append(-1.0)
        print('Finish!')

```

```
40              data = np.array(data)
41              label = np.array(label)
42
43              data = data / 255.0
44
45              train_data, test_data, train_label, test_label = train_test_split(data,
                    label, test_size=0.9)
46
47              return train_data, test_data, train_label, test_label
48
49
50          def one_vs_one(self):
51              data = []
52              label = []
53              print('Loading data (\"pos\"_\"neg\":{}_{})...'.format(self.num1, self.
                    num2), end='')
54              for root, dirs, files in os.walk('train_img/{}'.format(self.num1)):
55                  for f in files:
56                      '''
57                      # リサイズによる次元削減
58                      img = cv2.imread('{}/{}'.format(root, f), 0)
59                      dst = cv2.resize(img, dsize=None, fx=0.5, fy=0.5)
60                      dst = dst.reshape(-1)
61                      data.append(dst)
62                      '''
63                      data.append(cv2.imread('{}/{}'.format(root, f), 0).reshape(-1)) #
                            1次元配列に変換してdata に貯めていく
64                      label.append(1.0)
65              for root, dirs, files in os.walk('train_img/{}'.format(self.num2)):
66                  for f in files:
67                      '''
68                      # リサイズによる次元削減
69                      img = cv2.imread('{}/{}'.format(root, f), 0)
70                      dst = cv2.resize(img, dsize=None, fx=0.5, fy=0.5)
71                      dst = dst.reshape(-1)
72                      data.append(dst)
73                      '''
74                      data.append(cv2.imread('{}/{}'.format(root, f), 0).reshape(-1)) #
                            1次元配列に変換してdata に貯めていく
75                      label.append(-1.0)
76              print('Finish!')
77
78              data = np.array(data)
79              label = np.array(label)
80
81              data = data / 255.0
```

```
82
83              train_data, test_data, train_label, test_label = train_test_split(data,
                    label, test_size=0.8)
84
85              return train_data, test_data, train_label, test_label
86
87
88      # SVM
89      class SVM:
90          def __init__(self, data, label):
91              self.data = data
92              self.label = label
93
94
95          # 線形カーネル
96          def kernel(self, x, y):
97              return np.dot(x, y)
98
99
100         # ラグランジュ乗数を二次計画法で求める
101         def Lagrange(self, n):
102             K = np.zeros((n, n))
103             for i in range(n):
104                 for j in range(n):
105                     K[i, j] = self.label[i] * self.label[j] * np.dot(self.data[i], self
                            .data[j])
106             Q = cvxopt.matrix(K)
107             p = cvxopt.matrix(-np.ones(n)) # -1がn 個の列ベクトル
108             G = cvxopt.matrix(np.diag([-1.0]*n)) # 対角成分が-1の (n × n)行列
109             h = cvxopt.matrix(np.zeros(n)) # 0がn 個の列ベクトル
110             A = cvxopt.matrix(self.label, (1,n)) # N 個の教師信号が要素の行ベクトル (1 × n)
111             b = cvxopt.matrix(0.0) # 定数 0.0
112             solution = cvxopt.solvers.qp(Q, p, G, h, A, b) # 二次計画法でラグランジュ乗
                    数alpha を求める
113
114             alpha = np.array(solution['x']).flatten()
115
116             return alpha
117
118
119         # サポートベクトルを抽出
120         def support_vector(self, alpha):
121             S = []
122             for i in range(len(alpha)):
123                 if alpha[i] >= 0.00001:
124                     S.append(i)
```

```
125
126             return S
127
128
129         # w を計算
130         def w_cal(self, S, alpha):
131             w = np.zeros(784)
132             for n in S:
133                 w += alpha[n] * self.label[n] * self.data[n]
134
135             return w
136
137
138         # b を計算
139         def b_cal(self, S, alpha):
140             _sum = 0
141             for n in S:
142                 tmp = 0
143                 for m in S:
144                     tmp += alpha[m] * self.label[m] * self.kernel(self.data[n], self.
                        data[m])
145                 _sum += (self.label[n] - tmp)
146             b = _sum / len(S)
147
148             return b
149
150
151         def main(self):
152             alpha = self.Lagrange(len(self.data)) # ラグランジュ乗数
153             S = self.support_vector(alpha) # サポートベクトル
154             w = self.w_cal(S, alpha) # w
155             b = self.b_cal(S, alpha) # b
156
157             return w, b, S
158
159
160     # ラベルごとにデータを分割
161     def data_split(data, label):
162         cls1 = []
163         cls2 = []
164         for i in range(len(data)):
165             if label[i] == 1:
166                 cls1.append(data[i])
167             elif label[i] == -1:
168                 cls2.append(data[i])
169
```

```
170         return cls1, cls2
171
172
173     def f(x, w, b):
174         return np.dot(w, x) + b
175
176
177     # 精度を計算
178     def accuracy(cls1, cls2, w, b):
179         num = len(cls1) + len(cls2)
180         c1 = 0
181         c2 = 0
182         for i in cls1:
183             if f(i, w, b) >= 0:
184                 c1 += 1
185             elif f(i, w, b) < 0:
186                 c2 += 1
187         for i in cls2:
188             if f(i, w, b) < 0:
189                 c1 += 1
190             elif f(i, w, b) >= 0:
191                 c2 += 1
192         if c1 > c2:
193             acc = c1 / num
194         elif c1 < c2:
195             acc = c2 / num
196
197         return acc
198
199
200     # 保存
201     class SAVE:
202         def __init__(self, output_w, output_b, output_acc_train, output_acc_test,
                processing_time):
203             self.output_w = output_w
204             self.output_b = output_b
205             self.output_acc_train = output_acc_train
206             self.output_acc_test = output_acc_test
207             self.processing_time = processing_time
208
209
210         def one_vs_the_rest(self):
211             # 学習結果のパラメータを保存
212             df = pd.DataFrame(self.output_w, index=['0', '1', '2', '3', '4', '5',
                    '6', '7', '8', '9'])
213             df['b'] = self.output_b
```

```
214
215             dirname = 'classifier'
216             if not os.path.exists('{}'.format(dirname)):
217                 os.mkdir('{}'.format(dirname))
218
219             file_num = 1
220             while 1:
221                 if not os.path.exists('{}/one_versus_the_rest/SVM{}.csv'.format(dirname,
                        file_num)):
222                     df.to_csv('{}/one_versus_the_rest/SVM{}.csv'.format(dirname,
                            file_num))
223                     print('Save classifier as \"SVM{}.csv\"'.format(file_num))
224                     break
225                 else:
226                     file_num += 1
227
228             # 各学習の精度と学習時間を保存
229             df = pd.DataFrame({'acc_for_train':self.output_acc_train, 'acc_for_test':
                    self.output_acc_test, 'processing_time':self.processing_time}, index
                    =['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'])
230
231             dirname = 'result'
232             if not os.path.exists('{}'.format(dirname)):
233                 os.mkdir('{}'.format(dirname))
234
235             file_num = 1
236             while 1:
237                 if not os.path.exists('{}/one_versus_the_rest/result{}.csv'.format(
                        dirname, file_num)):
238                     df.to_csv('{}/one_versus_the_rest/result{}.csv'.format(dirname,
                            file_num))
239                     print('Save accuracy and processing time as \"result{}.csv\"'.format
                            (file_num))
240                     break
241                 else:
242                     file_num += 1
243
244
245         def one_vs_one(self):
246             # 学習結果のパラメータを保存
247             id = []
248             for i in range(0, 10):
249                 for j in range(i+1, 10):
250                     id.append('{}_{}'.format(i, j))
251
252             df = pd.DataFrame(self.output_w, index=id)
```

```
253             df['b'] = self.output_b
254 
255             dirname = 'classifier'
256             if not os.path.exists('{}'.format(dirname)):
257                 os.mkdir('{}'.format(dirname))
258 
259             file_num = 1
260             while 1:
261                 if not os.path.exists('{}/one_versus_one/SVM{}.csv'.format(dirname,
                        file_num)):
262                     df.to_csv('{}/one_versus_one/SVM{}.csv'.format(dirname, file_num))
263                     print('Save classifier as \"SVM{}.csv\"'.format(file_num))
264                     break
265                 else:
266                     file_num += 1
267 
268             # 各学習の精度と学習時間を保存
269             df = pd.DataFrame({'acc_for_train':self.output_acc_train, 'acc_for_test':
                    self.output_acc_test, 'processing_time':self.processing_time}, index=id)
270 
271             dirname = 'result'
272             if not os.path.exists('{}'.format(dirname)):
273                 os.mkdir('{}'.format(dirname))
274 
275             file_num = 1
276             while 1:
277                 if not os.path.exists('{}/one_versus_one/result{}.csv'.format(dirname,
                        file_num)):
278                     df.to_csv('{}/one_versus_one/result{}.csv'.format(dirname, file_num
                        ))
279                     print('Save accuracy and processing time as \"result{}.csv\"'.format
                        (file_num))
280                     break
281                 else:
282                     file_num += 1
283 
284 
285     if __name__ == '__main__':
286         while 1:
287             svm_type = int(input('\"one_versus_the_rest(0)\" or \"one_versus_one(1)\"?
                    : '))
288             if svm_type == 0 or svm_type == 1:
289                 break
290             else:
291                 print('Error. Please, input \"0\" or \"1\"')
292         output_w = []
```

```
        output_b = []
        output_acc_train = []
        output_acc_test = []
        processing_time = []

        if svm_type == 0:
            for num in range(0, 10):
                print('==============================')

                dataset_make = DATASET(num, None, None)
                train_data, test_data, train_label, test_label = dataset_make.
                    one_vs_the_rest()

                # SVM
                print('Start learning')
                svm = SVM(train_data, train_label)
                start = time.time()
                w, b, S = svm.main()
                elapsed_time = time.time() - start
                processing_time.append(elapsed_time)
                print('Finish!')

                output_w.append(w)
                output_b.append(b)

                # トレーニングデータ、テストデータをラベル別に分割
                cls1_train, cls2_train = data_split(train_data, train_label)
                cls1_test, cls2_test = data_split(test_data, test_label)

                # トレーニングデータ、テストデータのそれぞれに対して精度を計算
                acc_train = accuracy(cls1_train, cls2_train, w, b)
                output_acc_train.append(acc_train)
                acc_test = accuracy(cls1_test, cls2_test, w, b)
                output_acc_test.append(acc_test)
                print('Accuracy for training data (classify \"{}\") : {}'.format(num,
                    acc_train))
                print('Accuracy for test data (classify \"{}\") : {}'.format(num,
                    acc_test))

            print('==============================')

        elif svm_type == 1:
            for num1 in range(0, 10):
                for num2 in range(num1+1, 10):
                    print('==============================')

```

```
336                     dataset_make = DATASET(None, num1, num2)
337                     train_data, test_data, train_label, test_label = dataset_make.
                            one_vs_one()
338
339                     # SVM
340                     print('Start learning')
341                     svm = SVM(train_data, train_label)
342                     start = time.time()
343                     w, b, S = svm.main()
344                     elapsed_time = time.time() - start
345                     processing_time.append(elapsed_time)
346                     print('Finish!')
347
348                     output_w.append(w)
349                     output_b.append(b)
350
351                     # トレーニングデータ、テストデータをラベル別に分割
352                     cls1_train, cls2_train = data_split(train_data, train_label)
353                     cls1_test, cls2_test = data_split(test_data, test_label)
354
355                     # トレーニングデータ、テストデータのそれぞれに対して精度を計算
356                     acc_train = accuracy(cls1_train, cls2_train, w, b)
357                     output_acc_train.append(acc_train)
358                     acc_test = accuracy(cls1_test, cls2_test, w, b)
359                     output_acc_test.append(acc_test)
360                     print('Accuracy for training data (classify \"{}_{}\") : {}'.format
                            (num1, num2, acc_train))
361                     print('Accuracy for test data (classify \"{}_{}\") : {}'.format(
                            num1, num2, acc_test))
362
363             print('==============================================')
364
365         # 学習結果のパラメータ、各学習の精度と学習時間を保存
366         save = SAVE(output_w, output_b, output_acc_train, output_acc_test,
                processing_time)
367         if svm_type == 0:
368             save.one_vs_the_rest()
369         elif svm_type == 1:
370             save.one_vs_one()
```

付録 D　プログラム 4

ソースコード 4　課題 2 の MNIST の数字画像識別における識別器の多数決による識別用プログラム

```
1   import cv2
2   import numpy as np
3   import os
4   import pandas as pd
5   from collections import Counter
6   os.chdir(os.path.dirname(os.path.abspath(__file__)))
7
8
9   def dataset():
10      data = []
11      label = []
12      print('Loading data...', end='')
13      for num in range(0, 10):
14          for root, dirs, files in os.walk('mini_dataset/{}'.format(num)):
15              for f in files:
16                  data.append(cv2.imread('{}/{}'.format(root, f), 0).reshape(-1)) #
                        1次元配列に変換してdata に貯めていく
17                  label.append(num)
18      print('Finish!')
19
20      return data, label
21
22
23  def f(x, w, b):
24      return np.dot(w, x) + b
25
26
27  def distance(x, w, b):
28      return abs(np.dot(w, x) + b) / np.sqrt(np.dot(w, w))
29
30
31  if __name__ == '__main__':
32      while 1:
33          svm_type = int(input('\"one_versus_the_rest(0)\" or \"one_versus_one(1)\"?
                : '))
34          if svm_type == 0 or svm_type == 1:
35              break
36          else:
37              print('Error. Please, input \"0\" or \"1\"')
38
```

```
39          data, label = dataset()
40
41          if svm_type == 0:
42              par = pd.read_csv(filepath_or_buffer="classifier/one_versus_the_rest/SVM0_1.
                    csv", sep=",")
43              c = 0
44              _sum = 0
45              for img in data:
46                  judge = []
47                  for num in range(0, 10):
48                      w = par.iloc[num, 1:785].values
49                      b = par.iloc[num, 785]
50                      if f(img, w, b) >= 0:
51                          judge.append((num, distance(img, w, b)))
52                      elif f(img, w, b) < 0:
53                          judge.append((-1, distance(img, w, b)))
54
55                  pred = (-1, 1.7976931348623157e+308)
56                  for i in judge:
57                      if i[0] != -1 and i[1] < pred[1]:
58                          pred = i
59
60                  if pred[0] != -1:
61                      if pred[0] == label[_sum]:
62                          c += 1
63                  elif pred[0] == -1:
64                      _min = 1.7976931348623157e+308
65                      for i in judge:
66                          if i[1] < _min:
67                              n = i[0]
68                              _min = i[1]
69                      if n == label[_sum]:
70                          c += 1
71
72                  _sum += 1
73
74                  print('Finish No.{} image'.format(_sum))
75
76          elif svm_type == 1:
77              par = pd.read_csv(filepath_or_buffer="classifier/one_versus_one/SVM0_2.csv",
                     sep=",")
78              c = 0
79              _sum = 0
80              c_num = [0]*10
81              s_num = [0]*10
82              for img in data:
```

```
83                judge = []
84                index = 0
85                for i in range(0, 10):
86                    for j in range(i+1, 10):
87                        w = par.iloc[index, 1:785].values
88                        b = par.iloc[index, 785]
89                        if f(img, w, b) >= 0:
90                            judge.append(i)
91                        elif f(img, w, b) < 0:
92                            judge.append(j)
93                        index += 1
94
95                count = Counter(judge)
96                if count.most_common()[0][0] == label[_sum]:
97                    c += 1
98                    c_num[label[_sum]] += 1
99
100               s_num[label[_sum]] += 1
101               _sum += 1
102
103               print('Finish No.{} image'.format(_sum))
104
105       for i in range(0, 10):
106           accuracy = c_num[i] / s_num[i]
107           print('Accuracy for {} : {}'.format(i, accuracy))
108
109       accuracy = c / _sum
110       print('Accuracy : {}'.format(accuracy))
```