パターン認識
2019-04-23 授業分 レポート

37-196360　森田涼介

2019 年 5 月 4 日

宿題 1

2 つの正規分布にそれぞれ従う 2 次元点群を 2 セット用意し，正規分布のパラメータを既知として識別関数を設計して，これらの点群を分類する。識別関数は，

- Case 1: $\boldsymbol{\Sigma}_i = \sigma^2 \boldsymbol{I}$
- Case 2: $\boldsymbol{\Sigma}_i = \boldsymbol{\sigma}$
- Case 3: $\boldsymbol{\Sigma}_i = \text{arbitrary}$

の 3 ケースについて考える。

理論

ガウスモデル

$$p(\boldsymbol{x}|y) = \frac{1}{(2\pi)^{d/2}\det(\boldsymbol{\Sigma})^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{1}$$

を考えると，ベイズの定理より，

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y)p(y)}{p(\boldsymbol{x})} \tag{2}$$

$$\log(p(y|\boldsymbol{x})) = \log(p(x|y)) + \log(p(y)) - \log(p(x)) \tag{3}$$

$$= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log(\det(\boldsymbol{\Sigma})) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) + \log(p(y)) - \log(p(\boldsymbol{x})) \tag{4}$$

$$= -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) + \log(p(y)) - \frac{1}{2}\log(\det(\boldsymbol{\Sigma})) + \log(p(y)) + C \tag{5}$$

$$C = -\frac{d}{2}\log(2\pi) - \log(p(\boldsymbol{x})) \tag{6}$$

となる。いま，二値分類を考えると，識別関数 g は，

$$g(\boldsymbol{x}) = \log(p(y=1|\boldsymbol{x})) - \log(p(y=2|\boldsymbol{x})) \tag{7}$$

$$= -\frac{1}{2}\left((\boldsymbol{x}-\boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) - (\boldsymbol{x}-\boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)\right) - \frac{1}{2}\log\left(\frac{\det(\boldsymbol{\Sigma}_1)}{\det(\boldsymbol{\Sigma}_2)}\right) + \log\left(\frac{p_1}{p_2}\right) \tag{8}$$

となる。この g が正であれば予測は 1，負なら 2 である。

表 1: 3 つの dataset の概要

dataset	1	2	3
総点数	1000	1000	1000
点群 1 に属する点の数	307	415	517
点群 2 に属する点の数	693	585	483
点群 1 の平均	$\begin{bmatrix}2\\0\end{bmatrix}$	$\begin{bmatrix}2\\2\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$
点群 2 の平均	$\begin{bmatrix}-2\\0\end{bmatrix}$	$\begin{bmatrix}-2\\-2\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$
点群 1 の共分散行列	$\begin{bmatrix}1&0\\0&9\end{bmatrix}$	$\begin{bmatrix}5&0\\0&6\end{bmatrix}$	$\begin{bmatrix}1&0\\0&2\end{bmatrix}$
点群 2 の共分散行列	$\begin{bmatrix}1&0\\0&2\end{bmatrix}$	$\begin{bmatrix}6&0\\0&4\end{bmatrix}$	$\begin{bmatrix}8&0\\0&5\end{bmatrix}$

Case 1 のとき，$\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \sigma^2 \boldsymbol{I}$ から，

$$g(\boldsymbol{x}) = \frac{1}{\sigma^2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{x} + \left(-\frac{1}{2}(||\boldsymbol{\mu}_1||^2 - ||\boldsymbol{\mu}_2||^2) + \log\left(\frac{p_1}{p_2}\right)\right) \tag{9}$$

Case 2 のとき，$\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \Sigma$ から，

$$g(\boldsymbol{x}) = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{x} + \left(-\frac{1}{2}(\boldsymbol{\mu}_1^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2) + \log\left(\frac{p_1}{p_2}\right)\right) \tag{10}$$

Case 3 のとき，

$$\begin{aligned} g(\boldsymbol{x}) = &-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{\Sigma}_1^{-1} - \boldsymbol{\Sigma}_2^{-1})\boldsymbol{x} + (\boldsymbol{\mu}_1^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1} - \boldsymbol{\mu}_2^{\mathrm{T}}\boldsymbol{\Sigma}_2^{-1})\boldsymbol{x} \\ &+ \left(\boldsymbol{\mu}_1^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_2 - \frac{1}{2}\log\left(\frac{\det(\boldsymbol{\Sigma}_1)}{\det(\boldsymbol{\Sigma}_2)}\right) + \log\left(\frac{p_1}{p_2}\right)\right) \end{aligned} \tag{11}$$

条件設定

2 つの正規分布にそれぞれ従う 2 次元点群（以下では dataset と呼ぶ）は 3 種類用意した。それぞれの概要は表 1 にまとめた。

プログラム

プログラムの本体は 22 ページの listing 1 に示す。以下に含まれる関数の簡単な説明を記載する。

- dataset1
  2 つの正規分布にそれぞれ従う 2 次元点群を生成する。点の総数，そのうちの点群 1 の割合，各正規分布の平均と共分散行列を設定し，必要な数だけサンプリングする。点群 1，点群 2 とそれぞれの平均・共分散行列を返す。dataset2，dataset3 についても同様。

- sampling_normal_dist
  正規分布の平均・共分散行列と生成したい数を入れると，そこから点群をサンプリングする関数。
- classifier_binary
  二値分類問題の識別関数。点群と係数行列を受け取り，識別関数の値を返す。
- classify_binary_1
  Case 1 について点群を二値分類する関数。識別結果の他，係数行列も返す。classify_binary_2，classify_binary_3 はそれぞれ Case 2，Case 3 用である。
- measure_accuracy
  識別結果と点群のラベル（1 か 2），および点の総数から正解数と精度を算出する。
- sampling_normal_dist_for_contour
  点群に対し正規分布の等高線を描くための値を計算する。
- plot_data
  点群・等高線・識別境界線をプロットする。
- main
  実行用の関数。case 変数で各 Case を入れ替えられる。また，dataset1() 関数を他のものに入れ替えることで，dataset も入れ替えられる。

結果

以下に，識別の結果として精度と散布図を示す。散布図には点群 1 と 2 がともにプロットされており，求めた識別境界線も描かれている。プロット中では，群 1 を丸，群 2 をバツ印で表し，群 1 のうち正しく群 1 に分類されたものをオレンジ（darkorange），誤って群 2 に分類されたものを赤（red），群 2 のうち正しく群 2 に分類されたものを薄い青（royalblue），誤って群 1 に分類されたものを青（blue）で表す。また，その識別境界線を緑（darkcyan）で表す。

考察

紙面の都合上，各 dataset についての結果を示す前に，簡単な考察を示す。2 分布間に被りの少ない，dataset 1 のような設定では，Case 1 のような単純なモデルで十分分類できた。しかし，2 分布が共に含む領域を持つような dataset 2 では，当然分類の精度は低くなり，Case 3 を用いても，改善はしたがあまり大きくはなかった。2 つの正規分布の平均が一致するような設定である dataset 3 については，Case 1，Case 2 では全ての点が点群 1 に分類される結果となった。これは，$\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 \equiv \boldsymbol{\mu}$ かつ $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 \equiv \boldsymbol{\Sigma}$ のとき，式 (8) は

$$g(\boldsymbol{x}) = \log\left(\frac{p_1}{p_2}\right) \tag{12}$$

となり，識別関数がどんな $\boldsymbol{x}$ に対しても事前分布の大きい方のみに分類するようなものとなってしまうからである。なお，Case 3 ではこの問題は解消されることが結果からもわかる。

dataset 1 の結果

表 2: dataset1 に対する識別の結果

	Case 1	Case 2	Case 3
点群 1 の識別精度	302	302	298
点群 1 の識別精度	0.984	0.984	0.971
点群 2 の識別精度	688	688	692
点群 2 の識別精度	0.993	0.993	0.999
全体の正解率	0.990	0.990	0.990

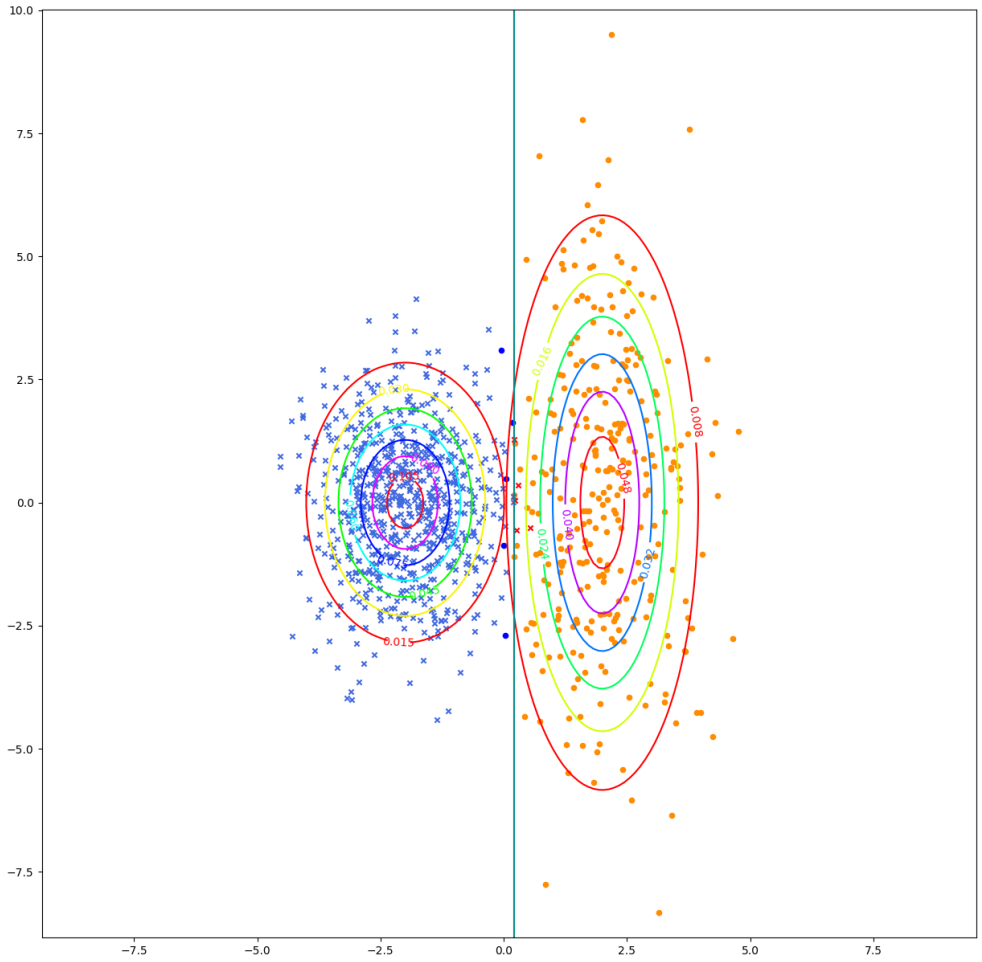


図 1: dataset 1 に対して Case 1 の識別関数を適用したときの散布図と識別境界線

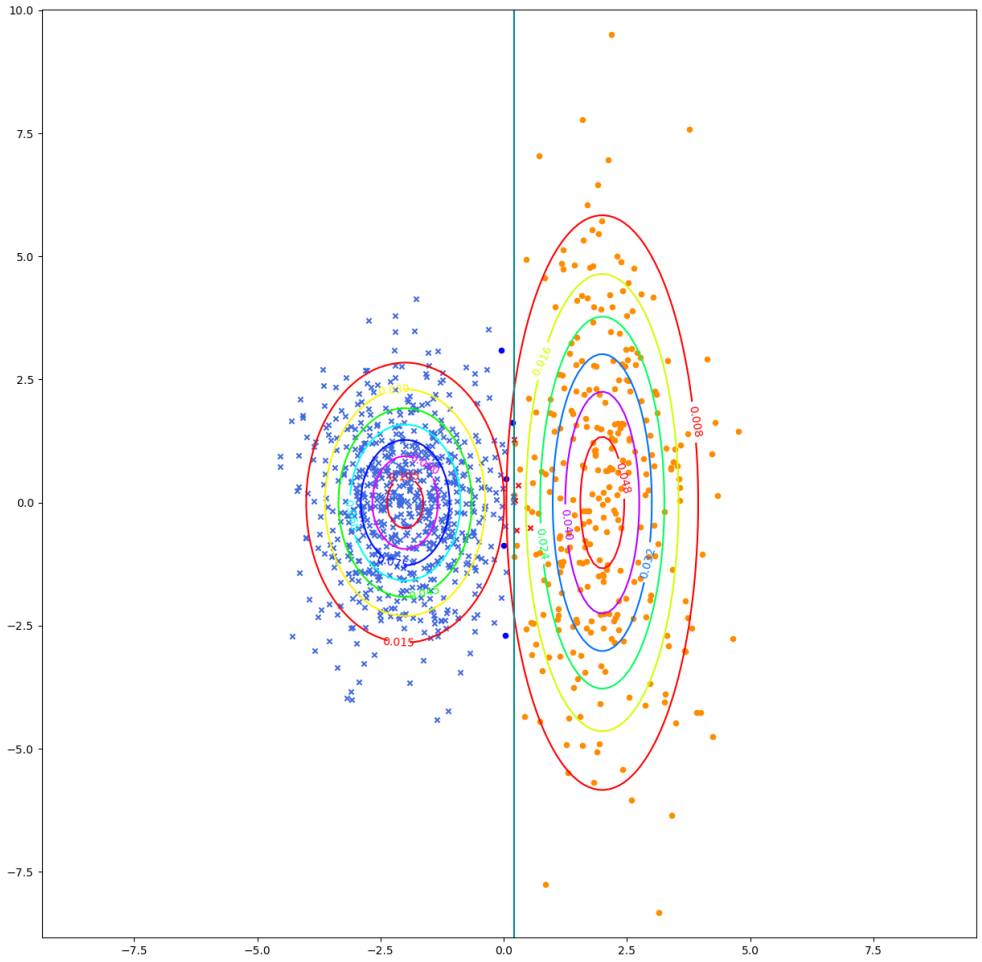


図 2: dataset 1 に対して Case 2 の識別関数を適用したときの散布図と識別境界線

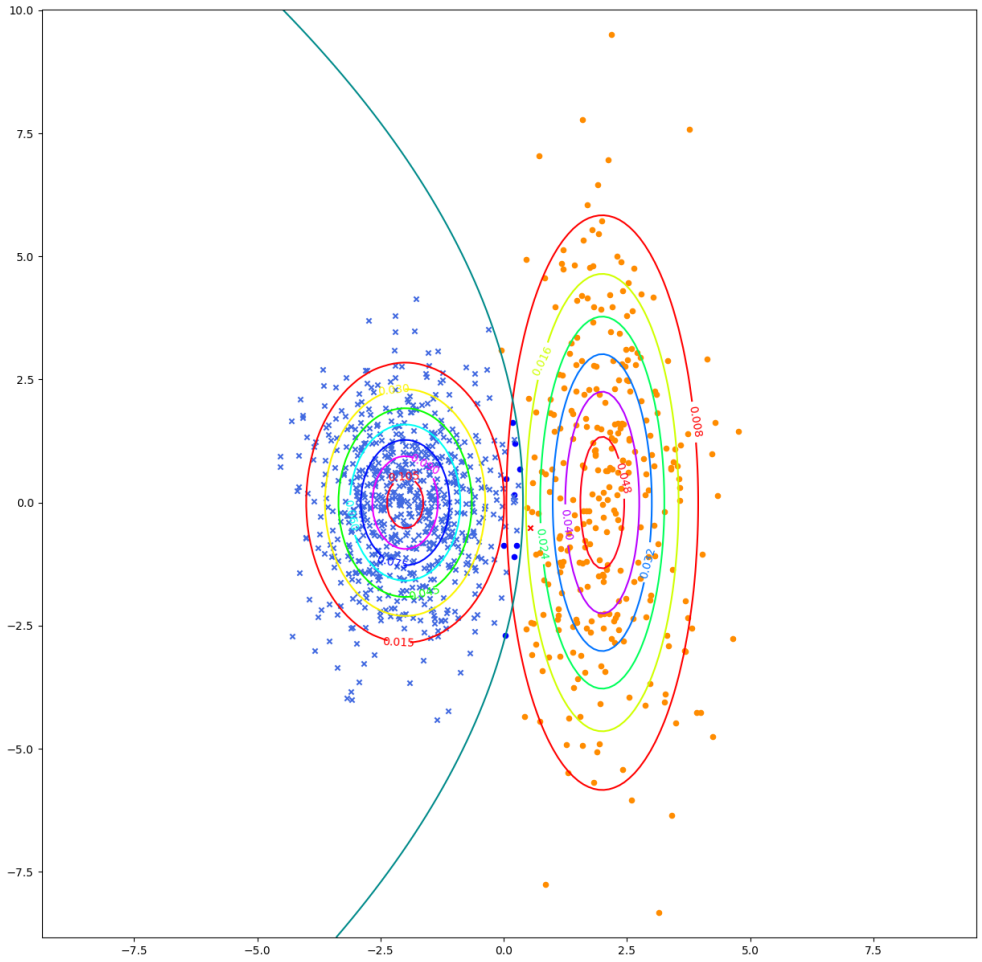


図 3: dataset 1 に対して Case 3 の識別関数を適用したときの散布図と識別境界線

dataset 2 の結果

表 3: dataset2 に対する識別の結果

	Case 1	Case 2	Case 3
点群 1 の識別精度	370	357	361
点群 1 の識別精度	0.892	0.860	0.870
点群 2 の識別精度	540	549	550
点群 2 の識別精度	0.923	0.938	0.940
全体の正解率	0.910	0.906	0.911

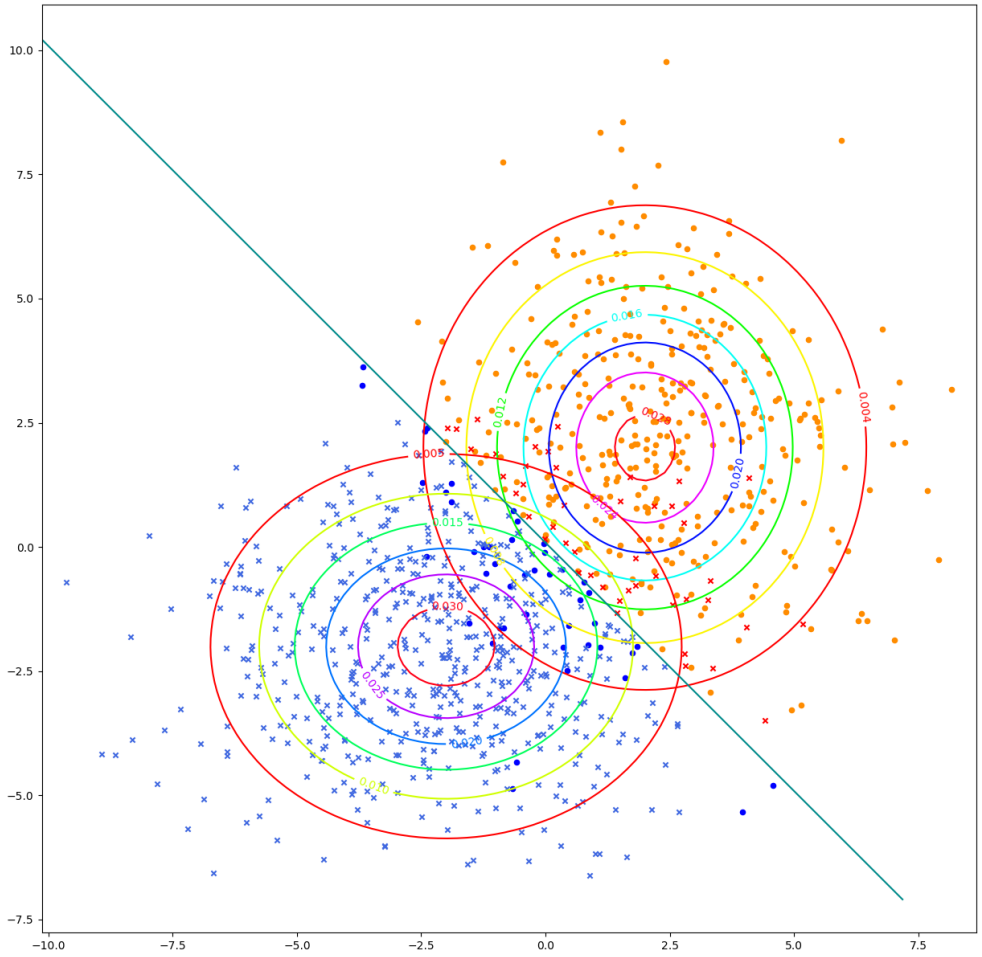


図 4: dataset 2 に対して Case 1 の識別関数を適用したときの散布図と識別境界線

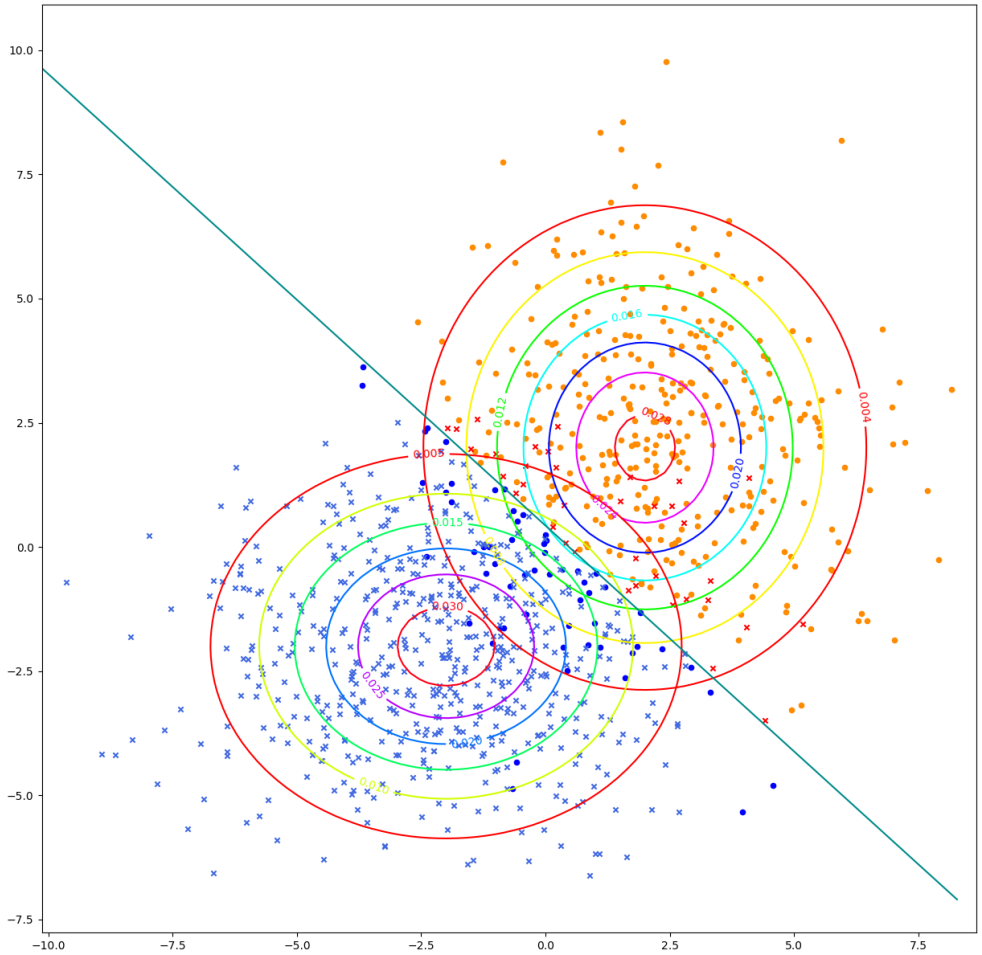


図 5: dataset 2 に対して Case 2 の識別関数を適用したときの散布図と識別境界線

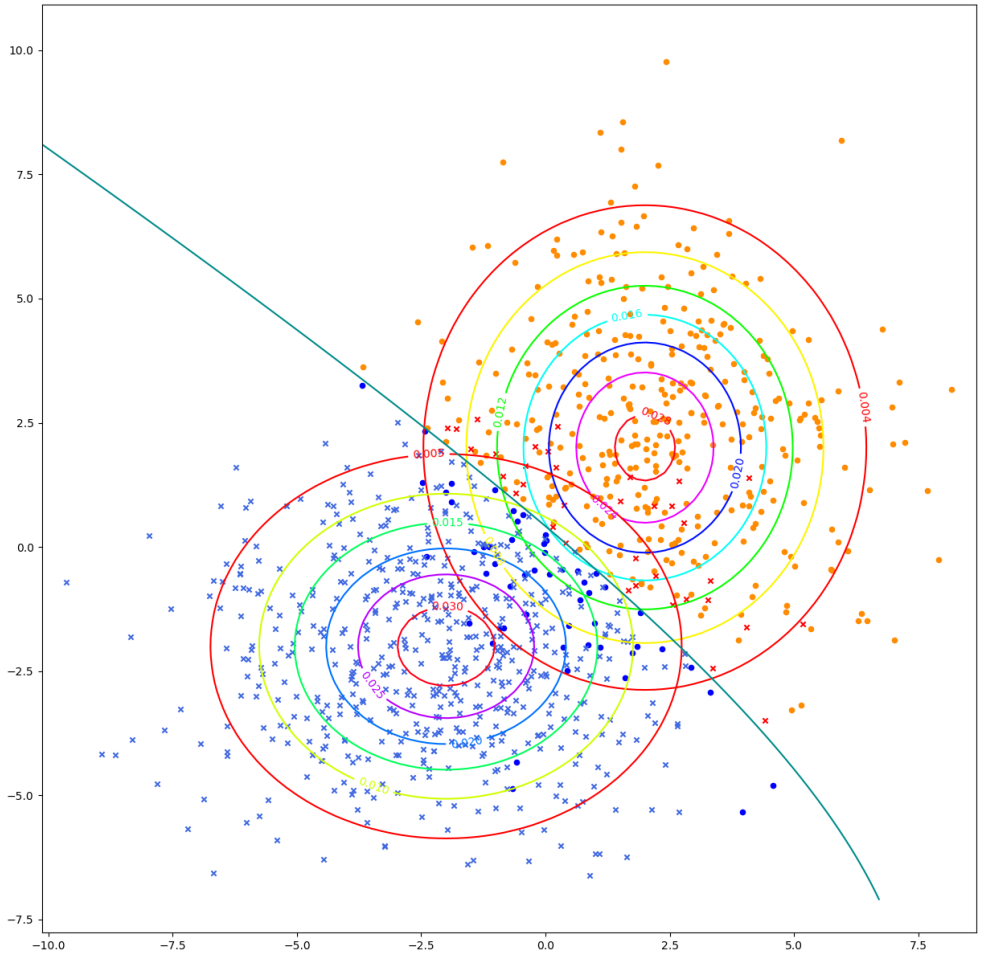


図 6: dataset 2 に対して Case 3 の識別関数を適用したときの散布図と識別境界線

dataset 3 の結果

表 4: dataset3 に対する識別の結果

	Case 1	Case 2	Case 3
点群 1 の識別精度	517	517	465
点群 1 の識別精度	1.000	1.000	0.899
点群 2 の識別精度	0	0	295
点群 2 の識別精度	0.000	0.000	0.611
全体の正解率	0.517	0.517	0.760

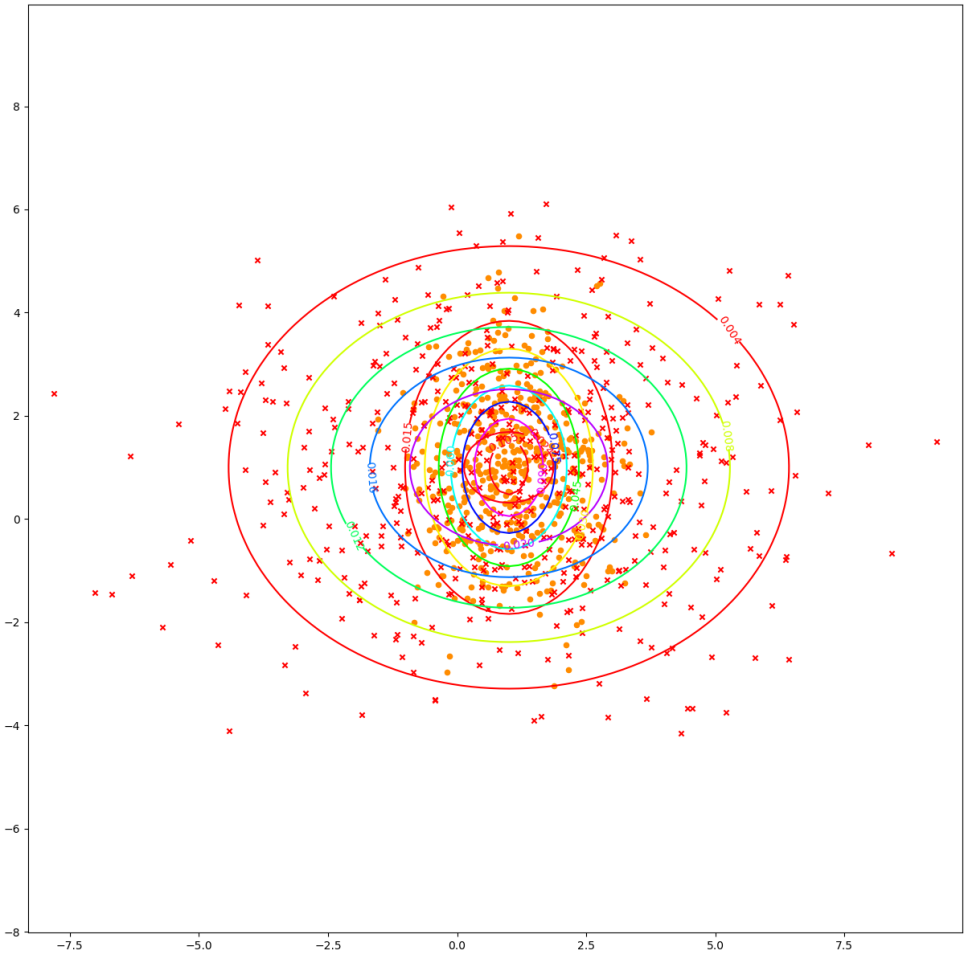


図 7: dataset 3 に対して Case 1 の識別関数を適用したときの散布図と識別境界線

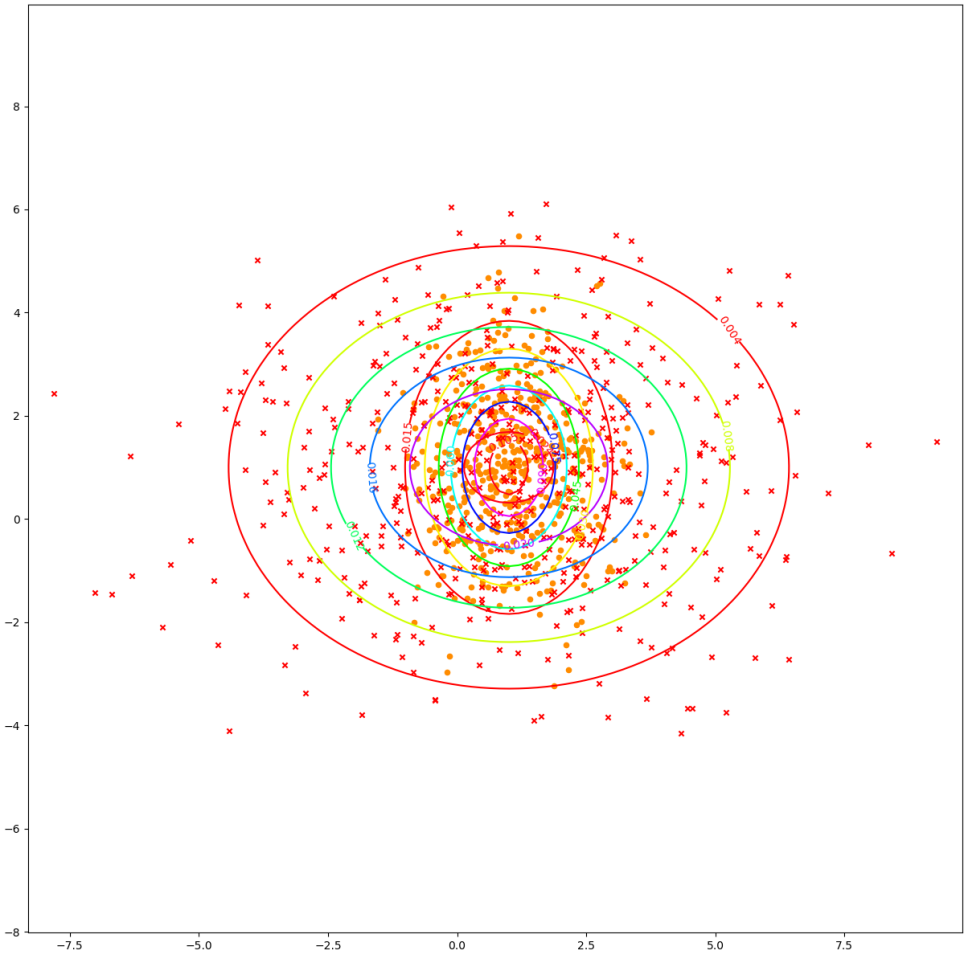


図 8: dataset 3 に対して Case 2 の識別関数を適用したときの散布図と識別境界線

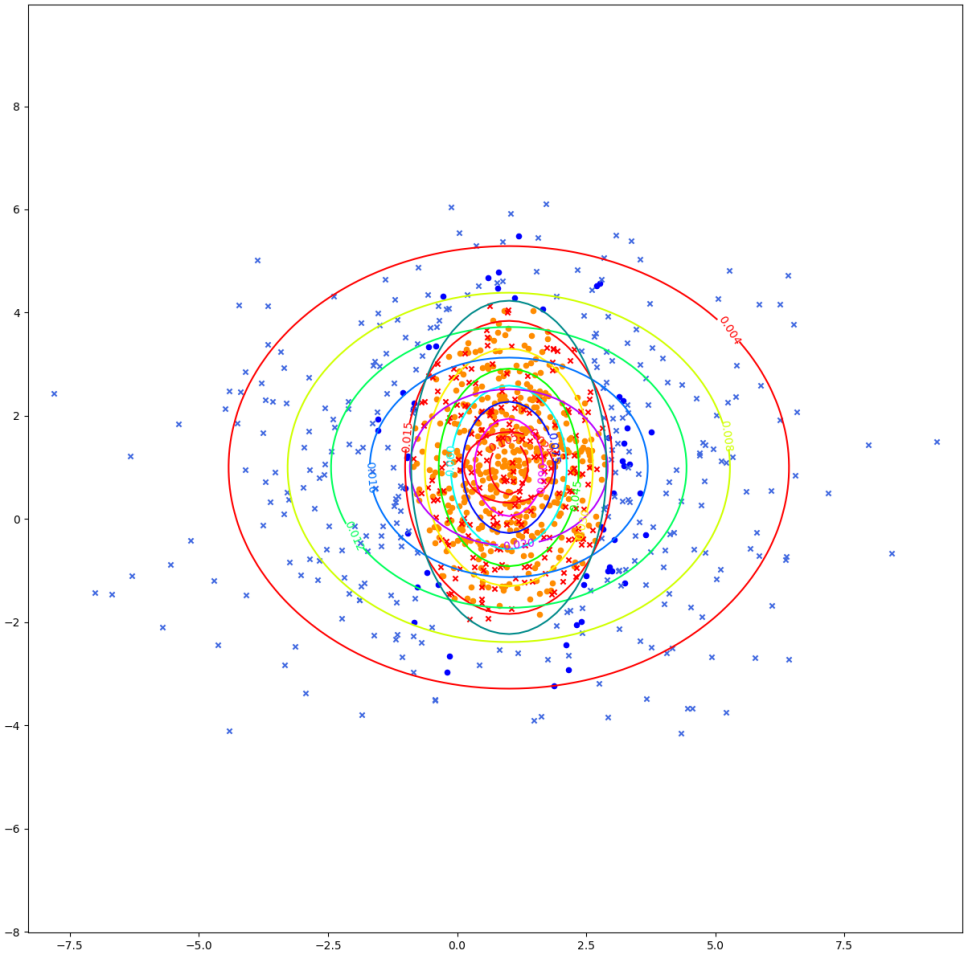


図 9: dataset 3 に対して Case 3 の識別関数を適用したときの散布図と識別境界線

宿題 2

MNIST を用いた手書き数字分類を行う。各画像は 784 ($= 28 \times 28$) 次元ベクトルとして扱い，各数字は正規分布に従うと仮定する。正規分布のパラメータを MNIST 学習データで推定し，そのパラメータにより識別関数を決定してテストデータを識別する。

識別関数は，

- Case 1: $\boldsymbol{\Sigma}_i = \sigma^2 \boldsymbol{I}$
- Case 2: $\boldsymbol{\Sigma}_i = \boldsymbol{\sigma}$
- Case 3: $\boldsymbol{\Sigma}_i =$ arbitrary

の 3 ケースについて考える。式 (5) を用いれば，ある $\boldsymbol{x}$ について，対数事後確率 $\log(p(y|\boldsymbol{x}))$ がそれぞれのラベル y について求まる。この値が最大となるものを予測値とすればよい。

プログラム

プログラムの本体は 29 ページの listing 2 に示す。共分散行列が正則でなかったため，その逆行列として，ムーア・ペンローズの疑似逆行列を用いた。また，Case 3 において，共分散行列の行列式が全てのラベルで 0 となったが，計算の都合上，行列式の関わる項（$\log(\det(\boldsymbol{\Sigma}))$）は無視した。

以下に含まれる関数の簡単な説明を記載する。

- load_data
  MNIST のデータを読み込み，train 用の画像とラベル，test 用の画像とラベルの計 4 つの配列を返す関数。
- get_statistics
  画像データの統計量を取る。ここで統計量とは，あるラベルに属する画像データの数・その全体に対する割合（事前確率）・平均・共分散行列である。
- train
  各 Case で用いる共分散行列を求め，対数事後確率を求めるのに用いる関数（log_prob_n）を決める。
- classify
  得られた画像統計量を用いて分類を行う。
- evaluate
  予測ラベルと実ラベルを比較し，全体の正解率・ラベルごとの正解率・混同行列を求める。
- visualize
  予測が実ラベルと一致したもの・しなかったものの一部を出力する

表 5: テストデータにおける各ラベルの画像の数，及び各 Case における識別結果の正解数と正解率

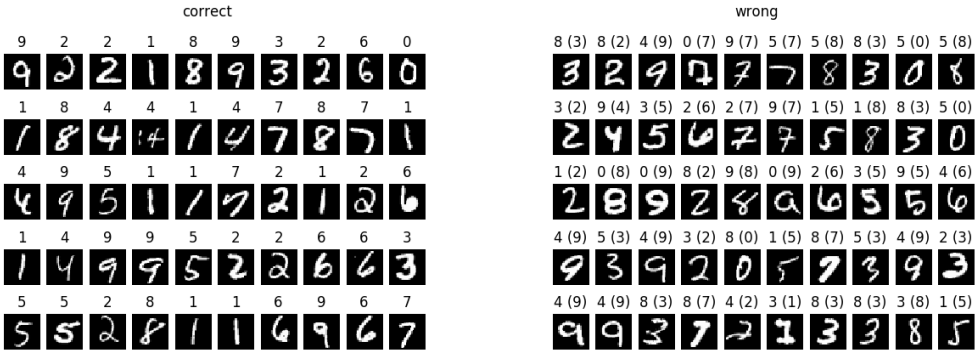
Label	#Data	Case 1		Case 2		Case 3	
		#Correct	Accuracy	#Correct	Accuracy	#Correct	Accuracy
0	980	878	0.896	940	0.959	916	0.935
1	1,135	1,092	0.962	1,092	0.962	765	0.674
2	1,032	781	0.757	815	0.790	966	0.936
3	1,010	815	0.807	881	0.872	884	0.875
4	982	811	0.826	890	0.906	893	0.909
5	892	611	0.685	736	0.825	713	0.799
6	958	827	0.863	857	0.895	853	0.890
7	1,028	856	0.833	860	0.837	888	0.864
8	974	718	0.737	795	0.816	865	0.888
9	1,009	814	0.807	859	0.851	829	0.822
All	10,000	8,203	0.820	8,725	0.873	8,572	0.857

結果

テストデータにおける各ラベルの画像の数，及び各 Case における識別結果の正解数と正解率を以下の表 5 に示す。また，各 Case における識別結果の混同行列を表 6–8 に，各 Case において予測ラベルと実ラベルが一致したもの・しなかったものの一例を図 10–12 に示す。ここで，予測ラベルと実ラベルが一致しなかったものの各数字には，「予測値 (実値)」の形でラベルが付いている。

表 6: Case 1 の混同行列

	0	1	2	3	4	5	6	7	8	9
0	878	0	7	2	2	57	25	1	7	1
1	0	1,092	10	3	0	7	3	0	20	0
2	19	71	781	33	31	3	23	18	50	3
3	4	24	25	815	1	49	8	15	57	12
4	1	22	2	0	811	3	16	1	10	116
5	11	63	2	119	21	611	27	10	13	15
6	18	27	22	0	31	32	827	0	1	0
7	2	59	22	1	20	2	0	856	13	53
8	14	39	11	83	12	36	13	10	718	38
9	15	22	7	10	83	12	1	27	18	814

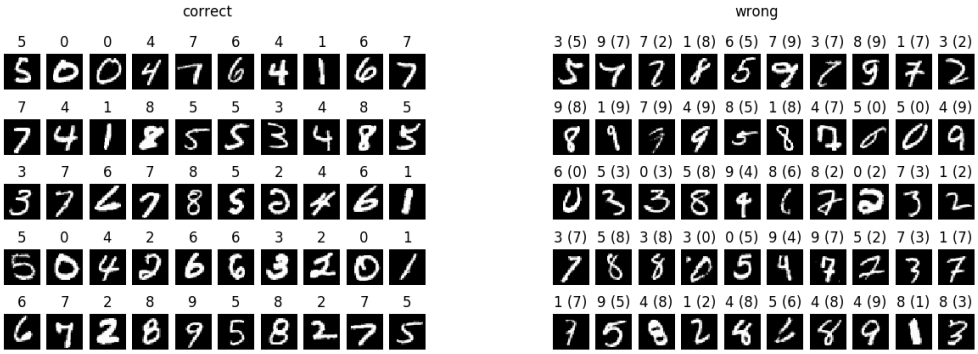


(a) 予測ラベルが実ラベルと一致したものの一例　　(b) 予測ラベルが実ラベルと一致しなかったものの一例

図 10: Case 1 のモデルの識別結果において予測ラベルと実ラベルが一致したもの・しなかったものの一例

表 7: Case 2 の混同行列

	0	1	2	3	4	5	6	7	8	9
0	940	0	0	4	2	13	9	1	10	1
1	0	1,092	4	4	2	3	3	0	27	0
2	15	32	815	35	21	5	37	9	57	6
3	5	6	25	881	4	27	3	15	28	16
4	0	12	6	0	890	2	7	1	11	53
5	8	8	4	45	12	736	15	10	36	18
6	12	8	11	0	25	29	857	0	16	0
7	2	31	15	9	22	2	0	860	4	83
8	7	27	7	26	20	53	9	5	795	25
9	9	7	1	13	62	6	0	40	12	859

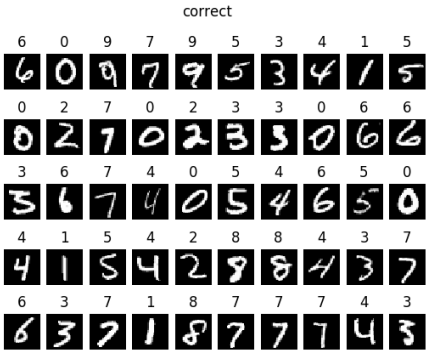


(a) 予測ラベルが実ラベルと一致したものの一例　　(b) 予測ラベルが実ラベルと一致しなかったものの一例

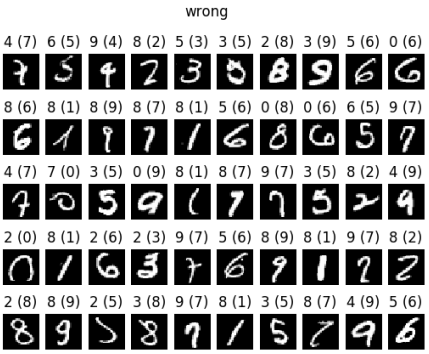
図 11: Case 2 のモデルの識別結果において予測ラベルと実ラベルが一致したもの・しなかったものの一例

表 8: Case 3 の混同行列

	0	1	2	3	4	5	6	7	8	9
0	916	0	20	7	1	8	4	1	23	0
1	0	765	44	8	7	3	8	0	300	0
2	8	0	966	18	4	0	2	3	31	0
3	4	0	46	884	2	12	0	6	52	4
4	2	0	38	3	893	3	2	7	27	7
5	10	0	16	56	5	713	6	2	80	4
6	15	0	29	1	5	24	853	0	31	0
7	0	0	16	16	30	2	0	888	30	46
8	8	0	33	34	5	19	1	6	865	3
9	8	0	9	12	61	1	0	35	54	829



(a) 予測ラベルが実ラベルと一致したものの一例



(b) 予測ラベルが実ラベルと一致しなかったものの一例

図 12: Case 3 のモデルの識別結果において予測ラベルと実ラベルが一致したもの・しなかったものの一例

考察

Case 2 では Case 1 より全てのラベルで精度が高くなった。これは，共分散行列のパラメータが 1 から 784×784 まで増えており，モデルの表現力が高くなったためであると考えられる。なお，Case 1 のように全てのピクセルについて同じ分散を仮定するのは，画像の端では値が 0 であることがほとんどであるのに対し画像の中心部ではそうでないことが多いことから考えても，モデルの表現力を制限しすぎていると考えられる。一方，Case 2 よりも $784 \times 784 \times 9$ だけパラメータが多くなっている Case 3 のモデルは，Case 2 よりも（全体での）精度が低くなってしまっている。表 5 を見ると，ラベル 2 と 8 の精度は大きく向上しているが，特にラベル 1 の精度が 30% 近く落ちてしまっている。この原因として考えられることとしては，train データの共分散行列が test データにおけるそれと大きくずれてしまっている可能性が挙げられる。

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 9 に示す。

表 9: 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

Listings 1: assignment1.py

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


import numpy as np
import matplotlib.pyplot as plt


def dataset1():
    n = 1000
    alpha = 0.3

    # prior probability and number of samples
    n1 = sum(np.random.rand(n) < alpha)
    n2 = n - n1

    mean1, mean2 = np.array([2, 0]), np.array([-2, 0])
    cov1 = np.array([[1, 0], [0, 9]])
    cov2 = np.array([[1, 0], [0, 2]])

    # generate data
    x1 = sampling_normal_dist(mean1, cov1, n1)
    x2 = sampling_normal_dist(mean2, cov2, n2)

    return x1, x2, mean1, mean2, cov1, cov2


def dataset2():
    n = 1000
    alpha = 0.4

    # prior probability and number of samples
    n1 = sum(np.random.rand(n) < alpha)
    n2 = n - n1
```

```
34
35      mean1, mean2 = np.array([2, 2]), np.array([-2, -2])
36      cov1 = np.array([[5, 0], [0, 6]])
37      cov2 = np.array([[6, 0], [0, 4]])
38
39      # generate data
40      x1 = sampling_normal_dist(mean1, cov1, n1)
41      x2 = sampling_normal_dist(mean2, cov2, n2)
42
43      return x1, x2, mean1, mean2, cov1, cov2
44
45
46  def dataset3():
47      n = 1000
48      alpha = 0.5
49
50      # prior probability and number of samples
51      n1 = sum(np.random.rand(n) < alpha)
52      n2 = n - n1
53
54      mean1, mean2 = np.array([1, 1]), np.array([1, 1])
55      cov1 = np.array([[1, 0], [0, 2]])
56      cov2 = np.array([[8, 0], [0, 5]])
57
58      # generate data
59      x1 = sampling_normal_dist(mean1, cov1, n1)
60      x2 = sampling_normal_dist(mean2, cov2, n2)
61
62      return x1, x2, mean1, mean2, cov1, cov2
63
64
65  def sampling_normal_dist(mean, cov, n):
66      return np.random.multivariate_normal(mean, cov, n)
67
68
69  def log_probability_normal_dist(x, mean, cov, prior_prob,):
70      d = len(mean)
71      cov_inv = np.linalg.inv(cov)
72      logp = (
73          -(1/2) * ((x - mean).dot(cov_inv) * (x - mean)).sum(axis=1)
74          -(1/2) * np.log(np.linalg.det(cov))
75          -(d/2) * np.log(2 * np.pi)
76          )
77      return logp
78
79
80  def classifier_binary(x, a, b, c):
```

```
    result = (x.dot(a) * x).sum(axis=1)
    result += b.dot(x.T)
    result += c
    return result


def classify_binary_1(x, mean1, mean2, sigma, p1, p2):
    d = len(mean1)
    a = np.zeros((d, d))
    b = (1/sigma**2) * (mean1 - mean2)
    c = -(1/(2*(sigma)**2)) * (np.linalg.norm(mean1)**2 -
    np.linalg.norm(mean2)**2) + np.log(p1/p2)
    result = classifier_binary(x, a, b, c)
    return result, a, b, c


def classify_binary_2(x, mean1, mean2, cov, p1, p2):
    d = len(mean1)
    cov_inv = np.linalg.inv(cov)
    a = np.zeros((d, d))
    b = cov_inv.dot(mean1 - mean2)
    c = -(1/2) * (mean1.T.dot(cov_inv).dot(mean1) -
    mean2.T.dot(cov_inv).dot(mean2)) + np.log(p1/p2)
    result = classifier_binary(x, a, b, c)
    return result, a, b, c


def classify_binary_3(x, mean1, mean2, cov1, cov2, p1, p2):
    cov1_inv = np.linalg.inv(cov1)
    cov2_inv = np.linalg.inv(cov2)
    a = -(1/2) * (cov1_inv - cov2_inv)
    b = cov1_inv.dot(mean1)  - cov2_inv.dot(mean2)
    c = (
        -(1/2)*(mean1.T.dot(cov1_inv).dot(mean1) -
    mean2.T.dot(cov2_inv).dot(mean2))
        - (1/2)*np.log(np.linalg.det(cov1)/np.linalg.det(cov2))
        + np.log(p1/p2)
        )
    result = classifier_binary(x, a, b, c)
    return result, a, b, c


def measure_accuracy(result, label, n_data):
    if label == 1:
        is_correct = (result >= 0)
    elif label == 2:
        is_correct = (result < 0)
```

```
125     else:
126         raise ValueError("'label' must be 1 or 2.")
127 
128     n_correct = is_correct.sum()
129     acc = n_correct / n_data
130     print('#Data: {}\t#Correct: {}\tAcc: {:.3f}'.format(n_data, n_correct,
        acc))
131     return is_correct
132 
133 
134 def sampling_normal_dist_for_contour(x, y, mean, cov):
135     icov = np.linalg.inv(cov)
136     xt = x - mean[0]
137     yt = y - mean[1]
138     p = (
139         1./(2. * np.pi * np.sqrt(np.linalg.det(cov)))
140         * np.exp(
141             -(1./2.)*(
142                 icov[0, 0]*xt*xt
143                 + (icov[0, 1] + icov[1, 0])*xt*yt
144                 + icov[1, 1]*yt*yt
145                 )
146             )
147         )
148     return p
149 
150 
151 def plot_data(
152         x1_o, x1_x, x2_o, x2_x,
153         a, b, c,
154         mean1, mean2, cov1, cov2,
155         x_linespace=None, y_linespace=None,
156         show=True, save=False, path=None,
157         size=10,
158         ):
159     if x_linespace is None:
160         x_linespace = np.linspace(-10, 10, 100)
161     if y_linespace is None:
162         y_linespace = np.linspace(-10, 10, 100)
163     _x, _y = np.meshgrid(x_linespace, y_linespace)
164 
165     plt.figure(figsize=(15, 15))
166     plt.axis('equal')
167 
168     p1_contour = sampling_normal_dist_for_contour(_x, _y, mean1, cov1)
169     plt.scatter(x1_o[:, 0], x1_o[:, 1], s=size, marker='o',
        color='darkorange')
```

```
170     plt.scatter(x1_x[:, 0], x1_x[:, 1], s=size, marker='o', color='blue')
171     cs1 = plt.contour(_x, _y, p1_contour, cmap='hsv')
172     plt.clabel(cs1)
173 
174     p2_contour = sampling_normal_dist_for_contour(_x, _y, mean2, cov2)
175     plt.scatter(x2_o[:, 0], x2_o[:, 1], s=size, marker='x',
        color='royalblue')
176     plt.scatter(x2_x[:, 0], x2_x[:, 1], s=size, marker='x', color='red')
177     cs2 = plt.contour(_x, _y, p2_contour, cmap='hsv')
178     plt.clabel(cs2)
179 
180     # decision boundary
181     _xy = np.c_[np.reshape(_x, -1), np.reshape(_y, -1)]
182     pp = classifier_binary(_xy, a, b, c)
183     pp = np.reshape(pp, _x.shape)
184     cs = plt.contour(_x, _y, pp, levels=[0.0], colors=['darkcyan'])
185     # plt.clabel(cs)
186 
187     if save:
188         plt.savefig(path)
189     if show:
190         plt.show()
191     return
192 
193 
194 def main():
195     # settings
196     case = 3
197     dataset_id = 3
198     fig_path =
        '../figures/result_assignment1_dataset{}_case{}.png'.format(dataset_id,
        case)
199     offset = 0.5
200     n_linespace = 100
201     np.random.seed(0)
202 
203     print('Case {}'.format(case))
204     print('Dataset: {}'.format(dataset_id))
205     print()
206 
207 
208     # load data
209     x1, x2, mean1, mean2, cov1, cov2 = dataset3()
210     n1 = len(x1)
211     n2 = len(x2)
212     n = n1 + n2
213     p1 = n1 / n
```

```
    p2 = n2 / n


    # decide which model to use
    if case == 1:
        sigma = (np.diag(cov1).sum() + np.diag(cov2).sum())/4
        result_1, a, b, c = classify_binary_1(
            x1, mean1, mean2, sigma, p1, p2
            )
        result_2, a, b, c = classify_binary_1(
            x2, mean1, mean2, sigma, p1, p2
            )
    elif case == 2:
        cov = (cov1 + cov2)/2
        result_1, a, b, c = classify_binary_2(
            x1, mean1, mean2, cov, p1, p2
            )
        result_2, a, b, c = classify_binary_2(
            x2, mean1, mean2, cov, p1, p2
            )
    elif case == 3:
        result_1, a, b, c = classify_binary_3(
            x1, mean1, mean2, cov1, cov2, p1, p2
            )
        result_2, a, b, c = classify_binary_3(
            x2, mean1, mean2, cov1, cov2, p1, p2
            )
    else:
        raise ValueError("'case' must be 1, 2, or 3.")


    # classify x1
    print('x1')
    is_correct_1 = measure_accuracy(result_1, 1, len(x1))
    x1_o = x1[is_correct_1]
    x1_x = x1[~is_correct_1]
    print()

    # classify x2
    print('x2')
    is_correct_2 = measure_accuracy(result_2, 2, len(x2))
    x2_o = x2[is_correct_2]
    x2_x = x2[~is_correct_2]
    print()

    acc = (is_correct_1.sum() + is_correct_2.sum()) / (len(is_correct_1) +
    len(is_correct_2))
```

```
260     print('Accuracy: {:.3f}'.format(acc))
261     print()
262
263
264     # plot
265     _x = np.concatenate([x1, x2], axis=0)
266     x_linespace = np.linspace(
267         _x[:, 0].min()-offset, _x[:, 0].max()+offset, n_linespace
268         )
269     y_linespace = np.linspace(
270         _x[:, 1].min()-offset, _x[:, 1].max()+offset, n_linespace
271         )
272     plot_data(
273         x1_o, x1_x, x2_o, x2_x,
274         a, b, c,
275         mean1, mean2, cov1, cov2,
276         x_linespace, y_linespace,
277         save=True, path=fig_path,
278         size=20,
279         )
280
281
282 if __name__ == '__main__':
283     main()
```

Listings 2: `assignment2.py`

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


import pathlib
import numpy as np
import matplotlib.pyplot as plt

from pprint import pprint

import mnread


def load_data():
    x_train = mnread.readim(mnread.trdatafz)
    y_train = mnread.readlabel(mnread.trlabelfz)
    x_test = mnread.readim(mnread.tstdatafz)
    y_test = mnread.readlabel(mnread.tstlabelfz)
    return x_train, y_train, x_test, y_test


def get_statistics(data, labels):
    n_data = len(labels)
    all_labels = sorted(list(set(labels)))
    # n_label = len(all_labels)
    statistics = {}
    for label in all_labels:
        _data = data[np.where(labels == label), :]
        n = _data.shape[1]
        _data = np.reshape(_data, [n, -1])
        statistics[label] = {
            'n': n,
            'p': n/n_data,
            'mean': np.mean(_data, axis=0),
            'cov': np.cov(_data.T),
            }
    return statistics


def train(case, statistics):
    all_labels = list(statistics.keys())
    n_label = len(all_labels)
    if case == 1:
        sigma = 0.0
        for label in all_labels:
            cov = statistics[label]['cov']
            sigma += np.diag(cov).mean()
```

```
47          sigma /= n_label
48          for label in all_labels:
49              statistics[label]['cov_train'] = np.sqrt(sigma)
50          log_prob = log_prob_1
51      elif case == 2:
52          Sigma = np.zeros_like(statistics[all_labels[0]]['cov'])
53          for label in all_labels:
54              cov = statistics[label]['cov']
55              Sigma += cov
56          Sigma /= n_label
57          for label in all_labels:
58              statistics[label]['cov_train'] = Sigma
59          log_prob = log_prob_2
60      elif case == 3:
61          for label in all_labels:
62              statistics[label]['cov_train'] = statistics[label]['cov']
63          log_prob = log_prob_3
64      else:
65          raise ValueError("'case' must be 1, 2, or 3.")
66      return statistics, log_prob
67
68
69  def log_probability_normal_dist(x, mean, cov, prior_prob,):
70      d = len(mean)
71      cov_inv = np.linalg.pinv(cov)
72      logp = (
73          -(1/2) * ((x - mean).dot(cov_inv) * (x - mean)).sum(axis=1)
74          -(1/2) * np.log(np.linalg.det(cov))
75          -(d/2) * np.log(2 * np.pi)
76          + np.log(prior_prob)
77          )
78      return logp
79
80
81  def log_prob_1(x, mean, sigma, prior_prob, eps=1e-4):
82      logp = mean.T.dot(x.T) - (1/2)*mean.T.dot(mean) +
        (sigma**2)*np.log(prior_prob)
83      return logp
84
85
86  def log_prob_2(x, mean, cov, prior_prob, eps=1e-4):
87      #cov_new = cov + eps*np.eye(len(cov))
88      #cov_inv = np.linalg.inv(cov_new)
89      cov_inv = np.linalg.pinv(cov)
90      logp = mean.T.dot(cov_inv).dot(x.T) -
        (1/2)*mean.T.dot(cov_inv).dot(mean) + np.log(prior_prob)
91      return logp
```

```
92
93
94  def log_prob_3(x, mean, cov, prior_prob, eps=1e-4):
95      #cov_new = cov + eps*np.eye(len(cov))
96      #cov_inv = np.linalg.inv(cov_new)
97      #det = np.linalg.det(cov_new)
98      #print(det, np.linalg.det(cov))
99      cov_inv = np.linalg.pinv(cov)
100     logp = -(1/2) * ((x - mean).dot(cov_inv) * (x - mean)).sum(axis=1)
101     #logp += - (1/2)*np.log(det)
102     logp += np.log(prior_prob)
103     return logp
104
105
106 def classify(x, statistics, log_prob, eps=1e-4):
107     n_data = len(x)
108     x = np.reshape(x, [n_data, -1])
109     all_labels = list(statistics.keys())
110     y_pred = []
111     for label in all_labels:
112         mean = statistics[label]['mean']
113         cov = statistics[label]['cov_train']
114         prior_prob = statistics[label]['p']
115         logp = log_prob(x, mean, cov, prior_prob, eps)
116         y_pred.append(logp)
117     y_pred = np.array(y_pred).T
118     y_pred = np.argmax(y_pred, axis=1)
119     return y_pred
120
121
122 def evaluate(y_true, y_pred, labels):
123     n_data = len(y_true)
124     n_label = len(labels)
125
126     # accuracy
127     n_correct = (y_pred == y_true).sum()
128     acc = n_correct / n_data
129     print('All\t#Data: {}\t#Correct: {}\tAcc: {:.3f}'.format(n_data,
        n_correct, acc))
130
131     # acc per label
132     confusion_matrix = np.zeros((n_label, n_label), dtype=int)
133     for i, label_true in enumerate(labels):
134         idx_y_true = (y_true == label_true)
135         _y_pred = y_pred[idx_y_true]
136         _n_data = len(_y_pred)
137         for j, label_pred in enumerate(labels):
```

```
138             n = (_y_pred == label_pred).sum()
139             confusion_matrix[i, j] = n
140
141         n_correct = (_y_pred == label_true).sum()
142         acc = n_correct / _n_data
143         print('Label: {}\t#Data: {}\t#Correct: {}\tAcc:
        {:.3f}'.format(label_true, _n_data, n_correct, acc))
144     print()
145     print('Confusion Matrix\n', confusion_matrix)
146     print()
147     return confusion_matrix
148
149
150 def visualize(case, x, y_true, y_pred, image_dir, n=50):
151     image_dir = pathlib.Path(image_dir)
152
153     plt.figure()
154     plt.suptitle('correct')
155     indices_correct =
        np.random.permutation(np.where(y_pred==y_true)[-1])[range(n)]
156     for i, idx_correct in enumerate(indices_correct):
157         plt.subplot(5, 10, i+1)
158         plt.axis('off')
159         plt.imshow(x[idx_correct, :, :], cmap='gray')
160         plt.title(y_pred[idx_correct])
161     plt.savefig(str(image_dir /
        'result_assignment2_case{}_correct.png'.format(case)))
162
163     plt.figure()
164     plt.suptitle('wrong')
165     indices_wrong =
        np.random.permutation(np.where(~(y_pred==y_true))[-1])[range(n)]
166     for i, idx_wrong in enumerate(indices_wrong):
167         plt.subplot(5,10,i+1)
168         plt.axis('off')
169         plt.imshow(x[idx_wrong, :, :], cmap='gray')
170         plt.title('{} ({})'.format(y_pred[idx_wrong], y_true[idx_wrong]))
171     plt.savefig(str(image_dir /
        'result_assignment2_case{}_wrong.png'.format(case)))
172     plt.show()
173
174
175 def main():
176     # settings
177     case = 3
178     image_dir = pathlib.Path().cwd().parent / 'figures'
179     np.random.seed(0)
```

```
180     print('Case: {}'.format(case))
181
182     # load data
183     x_train, y_train, x_test, y_test = load_data()
184     #print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
185
186     all_labels = sorted(list(set(y_train)))
187     #n_data = len(y_train)
188     #n_label = len(all_labels)
189     #print(n_data, n_label, all_labels)
190
191     # train (get statistics and calc sigmas)
192     train_statistics = get_statistics(x_train, y_train)
193     train_statistics, log_prob = train(case, train_statistics)
194
195     # test
196     y_pred = classify(x_test, train_statistics, log_prob, eps=1e-2)
197     evaluate(y_test, y_pred, all_labels)
198     visualize(case, x_test, y_test, y_pred, image_dir, n=50)
199
200
201 if __name__ == '__main__':
202     main()
```