目次

1 目的 · · · 2
2 実験手順 · · · 2
2.1 識別部の実装 · · · 2
2.2 識別部の性能評価 · · · 2
3 結果 · · · 2
3.1 識別部の実装 · · · 2
3.2 識別部の性能評価 · · · 8
4 考察 · · · 8
4.1 最近傍決定則と統計的識別の識別精度の違い · · · 8
4.2 識別部の評価 · · · 8

1　目的

学習データをもとに最近傍決定則や統計的識別で識別部を実装し，入力データを識別する．また，識別部の性能評価を行い性能が信用できるかの考察を行う．

2　実験手順

2.1　識別部の実装

最近傍決定則

1. 学習データのそれぞれのクラスの平均ベクトルをプロトタイプ（お手本）として設定した
2. 入力と最も近いプロトタイプを探索することで識別を行った

統計的識別

1. 識別に必要な事前確率であるクラス ω_i の生起確率と計算に必要なクラスの平均ベクトルと分散ベクトルを求めた
2. ナイーブベイズ法を用いて，事前確率と各次元のクラス分布の値の積が最大となるクラスを識別結果とした

2.2　識別部の性能評価

3　結果

学習データは配布された数字のデータを用いており，識別に最も適した特徴として縦横の分散を採用した．

3.1　識別部の実装

最近傍決定則

最近傍決定則での識別部の実装をソースコード 1 に示す．以下のコードでは，識別が正しく行われなかった入力に対してのみ出力するようにしている．また，各クラスのプロトタイプとして，クラスの平均ベクトルを採用した．

```
1  from google.colab import drive
2  drive.mount('/content/drive')
```

```
3
4   import csv
5   import matplotlib.pyplot as plt
6   import numpy as np
7   import pprint
8   import random
9
10  # クラス数
11  NUM_CLASS = 10
12  # クラスごとのデータ数
13  NUM_CLASS_DATA = 10
14
15  # 数字ラベル
16  LABEL = 0
17  # 重心
18  MUX = 1
19  MUY = 2
20  # 分散
21  VX = 3
22  VY = 4
23  # ゆがみ
24  SX = 5
25  SY = 6
26  # 扁平度
27  FX = 7
28  FY = 8
29
30  # CSV形式のファイルを行列に読み込み
31  def read_csv(filename):
32      with open(filename) as f:
33          reader = csv.reader(f)
34          data = list(reader)
35      # 数値に変換
36      data = [[float(value) for value in row] for row in data]
37      return data
38
39  # データを読み込む
40  data = read_csv('/content/drive/MyDrive/pattern/week01/feature.csv')
41
42  # Numpy操作用に変換
43  data = np.array(data)
44
45  # 各クラスの平均ベクトルを求める関数
46  def get_class_mean(feature_list):
47    res = np.empty((0, 9))
48
49    for i in range(NUM_CLASS):
50      start = i * NUM_CLASS
51      mean_vec = np.mean(feature_list[start : start + NUM_CLASS_DATA], axis=0)
52      res = np.append(res, [mean_vec], axis=0)
53
54    return res
55
56  # 各クラスのプロトタイプ（今回は平均ベクトルをプロトタイプとした）
```

```
57  prototype_list = get_class_mean(data)
58  # print(prototype_list)
59
60  #=============================================================================
61  #                                  最近傍決定則
62  #=============================================================================
63
64  # 特徴ベクトルf1とf2の距離を求める関数
65  def get_distance(f1, f2, idx1, idx2):
66    p = np.array([f1[idx1], f1[idx2]])
67    q = np.array([f2[idx1], f2[idx2]])
68    return np.linalg.norm(p - q)
69
70  # 特徴の組み合わせ(idx1, idx2)に対して，識別を行う関数
71  def recognize_by_nn(feature, idx1, idx2):
72    l = []
73
74    for i in range(NUM_CLASS):
75      dist = get_distance(feature, prototype_list[i], idx1, idx2)
76      l.append(dist)
77
78    return l.index(min(l))
79
80  #=============================================================================
81  #                                  入力と実行
82  #=============================================================================
83
84  for i in range(NUM_CLASS):
85    for j in range(NUM_CLASS_DATA):
86      feature = data[i * NUM_CLASS + j]
87      result = recognize_by_nn(feature, VX, VY)
88      if i != result:
89        print('target number: ' + str(i))
90        print('case number: ' + str(j))
91        print('result: ' + str(result) + '(' + str(i == result) + ')')
92        print('')
```

ソースコード 1: 最近傍決定則での識別部の実装

出力結果を図 1 に示す.

```
Mounted at /content/drive
target number: 8
case number: 6
result: 6(False)

target number: 8
case number: 7
result: 6(False)

target number: 9
case number: 1
result: 8(False)

target number: 9
case number: 3
result: 0(False)
```

図 1: 最近傍決定則での出力結果

最近傍決定則の精度は $1 - \frac{4}{100} = 0.96$，つまり 96% であった．

統計的識別

ナイーブベイズ法による識別部の実装をソースコードに示す．以下のコードにおいても，識別が正しく行われなかった入力に対してのみ出力するようにしている．

```python
from google.colab import drive
drive.mount('/content/drive')

import csv
import matplotlib.pyplot as plt
import numpy as np
import statistics
import math

# クラス数
NUM_CLASS = 10
# クラスごとのデータ数
NUM_CLASS_DATA = 10

# 数字ラベル
LABEL = 0
# 重心
MUX = 1
MUY = 2
# 分散
VX = 3
VY = 4
# ゆがみ
```

```
SX = 5
SY = 6
# 扁平度
FX = 7
FY = 8

# CSV形式のファイルを行列に読み込み
def read_csv(filename):
    with open(filename) as f:
        reader = csv.reader(f)
        data = list(reader)
    # 数値に変換
    data = [[float(value) for value in row] for row in data]
    return data

# データを読み込む
data = read_csv('/content/drive/MyDrive/pattern/week01/feature.csv')

# Numpy操作用に変換
data = np.array(data)

#================================================================================
#                                  ナイーブベイズ
#================================================================================

# 識別に適した特徴の組み合わせ
IDX1 = VX
IDX2 = VY

# 各クラスの事前確率P(ωi)
PRIOR_PROBABLITY = 0.1

# 各クラスの平均ベクトルを求める関数
def get_class_mean(feature_list):
  res = np.empty((0, 9))

  for i in range(NUM_CLASS):
    start = i * NUM_CLASS
    mean_vec = np.mean(feature_list[start : start + NUM_CLASS_DATA], axis=0)
    res = np.append(res, [mean_vec], axis=0)

  return res

# 各クラスの分散を求める
def get_class_variance(feature_list):
  res = np.empty((0, 9))

  for i in range(NUM_CLASS):
    start = i * NUM_CLASS
    var_vec = np.var(feature_list[start : start + NUM_CLASS_DATA], axis=0)
    res = np.append(res, [var_vec], axis=0)

  return res

```

```
78  # 各クラスの平均ベクトル
79  mean_list = get_class_mean(data)
80  # 各クラスの分散
81  variance_list = get_class_variance(data)
82
83  # 正規分布関数
84  def normal(x, u, v):
85    res = 1 / np.sqrt(2 * np.pi * v) * np.exp(-(x-u)**2/(2*v))
86    return res
87
88  # 尤度を求める関数
89  def get_likelihood(feature, mean, var):
90    p1 = normal(feature[IDX1], mean[IDX1], var[IDX1])
91    p2 = normal(feature[IDX2], mean[IDX2], var[IDX2])
92    return p1 * p2
93
94  def recognize_by_naive_bayes(vec_in):
95    l = []
96
97    for i in range(NUM_CLASS):
98      p = PRIOR_PROBABLITY * get_likelihood(vec_in, mean_list[i], variance_list[i])
99      l.append(p)
100
101   return l.index(max(l))
102
103 #=============================================================================
104 #                                    入力と実行
105 #=============================================================================
106
107 for i in range(NUM_CLASS):
108   for j in range(NUM_CLASS_DATA):
109     feature = data[i * NUM_CLASS + j]
110     result = recognize_by_naive_bayes(feature)
111     if i != result:
112       print('target number: ' + str(i))
113       print('case number: ' + str(j))
114       print('result: ' + str(result) + '(' + str(i == result) + ')')
115       print('')
```

ソースコード 2: ナイーブベイズ法による識別部の実装

出力結果を図 2 に示す.

```
target number: 8
case number: 6
result: 6(False)

target number: 9
case number: 1
result: 8(False)

target number: 9
case number: 3
result: 0(False)
```

図 2: ナイーブベイズ法での出力結果

ナイーブベイズ法の精度は $1 - \frac{3}{100} = 0.97$，つまり 97% であった．
以上の結果から，最近傍決定則よりもナイーブベイズ法の方が識別の精度が高くなった．

3.2 識別部の性能評価

4 考察

4.1 最近傍決定則と統計的識別の識別精度の違い

4.2 識別部の評価

参考文献

[1] 崔恩瀞．プロジェクト実習 II パターン認識 実験テキスト．京都工芸繊維大学，2024 年