認識工学　大レポート 1
DP マッチングによる単語音声認識

未来ロボティクス学科 学部 3 年
18C1057 下鳥晴己

2020 年 7 月 14 日

目次

1 目的 2

2 理論 2

2.1 局所距離 2

2.2 累積距離 2

2.3 DP マッチング 3

3 実験方法 3

4 環境・使用機器 4

5 実験結果 4

5.1 同一の話者 4

5.2 異なる話者 8

6 考察 13

7 まとめ 13

付録 A ソースコード 14

1　目的

DP マッチングを行うプログラムを作成し、小語彙の単語音声認識実験を行う。実験した結果から、作成したプログラムの性能を評価する。

2　理論

2.1　局所距離

テンプレート A と未知入力 B の各分析フレーム相互間の距離を局所距離という。それぞれのデータが n 次のメルケプストラム特徴量で表現されており、n 次番目のデータをそれぞれ a_n、b_n とすると、1 フレーム間の局所距離 d は式 1 で表される。

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2} \tag{1}$$

また、全フレームの局所距離は、テンプレートのフレーム数が I、未知入力のフレーム数が J のとき、$I \times J$ の二次元配列 $d(i,j)[0 \leqq i \leqq I-1, 0 \leqq j \leqq J-1]$ のように表すとする。$d(i,j)$ の各ノードは、式 2 のように局所距離を計算する。

$$d(i,j) = \sqrt{(a_{i,1} - b_{j,1})^2 + (a_{i,2} - b_{j,2})^2 + \cdots + (a_{i,n} - b_{j,n})^2} \tag{2}$$

2.2　累積距離

累積距離は、二次元配列 $g(i,j)[0 \leqq i \leqq I-1, 0 \leqq j \leqq J-1]$ で表すとする。初期条件は $g(0,0) = d(0,0)$ であり、境界条件は $i > 0$ においては式 3、$i > 0$ においては式 4 のような漸化式で表される。その他のノードについては式 5 のような漸化式で表現される。

$$g(i,0) = g(i-1,0) + d(i,0) \tag{3}$$

$$g(0,j) = g(0,j-1) + d(0,j) \tag{4}$$

$$g(i,j) = \min \begin{bmatrix} g(i,j-1) & + & d(i,j) \\ g(i-1,j-1) & + & 2d(i,j) \\ g(i-1,j) & + & d(i,j) \end{bmatrix} \tag{5}$$

2.3 DP マッチング

DP マッチングは動的計画法のことで、全体の問題の最適解は部分問題の最適解に一致するという、最適性の原理を使った手法である。具体的には、テンプレートと未知入力の局所距離の累積距離が最短になる経路を求めることで計算できる。

例えば、発声のたびに長さが非線形に伸縮してしまうような音声信号に対して DP マッチングを行うと、時間軸方向の正規化を図ることができ、小語彙の単語音声認識に応用できる。

3 実験方法

2 人の話者が 100 個の単語を、それぞれ 2 回ずつ発声した 4 つのデータベースを用いて、作成した DP マッチングのアルゴリズムの性能を評価する。評価方法は、任意の話者の 100 単語のテンプレートに対して、同じ発生内容の 100 単語を未知入力として DP マッチングを行い、最小の累積距離が得られた単語がテンプレートと一致しているかを調べ、その正答率を計算して評価する。組み合わせは、同一話者間の組み合わせ 4 通りと、別話者間の組み合わせ 4 通りの合計 8 通りを比較した。

使用する音声データは、先頭 3 行にヘッダ情報があり、4 行名以降に 15 次のメルケプストラム特徴量のデータがあるテキストファイルである。

プログラムは、テンプレートの話者と発声回数をそれぞれ、$talker1$、$talknum1$、未知入力の話者と発声回数を、$talker2$、$talknum2$ の変数で定義している。それぞれの変数には 1 か 2 を代入し、例えばテンプレートに話者 1 の 1 回目の発声、未知入力に話者 2 の 2 回目の発声を用いたいのであれば、以下の表 1 のように定義する。なお、作成したソースコードは付録 A に記載している。

表 1

$talker1$	1
$talknum1$	1
$talker2$	2
$talknum2$	2

4　環境・使用機器

以下の環境で実験を行う。

- OS: Windows 10 pro
- PC: Thinkpad P53
- CPU: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59GHz
- RAM: 32.0GB
- 開発環境: Jupyter Notebook
- 使用言語: Pyrhon3.7.6

5　実験結果

すべての組み合わせの結果は以下のようになった。

5.1　同一の話者

5.1.1　話者 1 の 1 回目と話者 1 の 1 回目の発声

プログラムの変数と結果を表 2 に示す。同一の発声データなので正答率は 100[%] になった。

表 2

$talker1$	1
$talknum1$	1
$talker2$	1
$talknum2$	1
正答率 [%]	100

5.1.2　話者 1 の 1 回目と話者 1 の 2 回目の発声

プログラムの変数と結果を表 3 に示す。正答率は比較的高い 99[%] となった。これは同一話者であるため正答率が高くなったと考えられる。

誤答は、20 番目の単語「TOOKYOO」と 62 番目の単語「BOOSOO」が一致すると認識される誤答であった。これらは見たところ似たような単語であるため、誤答してしまったと考えられる。

正答と誤答の比較として、図 1 に DP マッチングによって求められた経路、図 2 に局所距離を色の濃淡で表したヒートマップを示す。これらを見てわかるように、非常に似た結果となっていることがわかる。

表 3

$talker1$	1
$talknum1$	1
$talker2$	1
$talknum2$	2
正答率 [%]	99

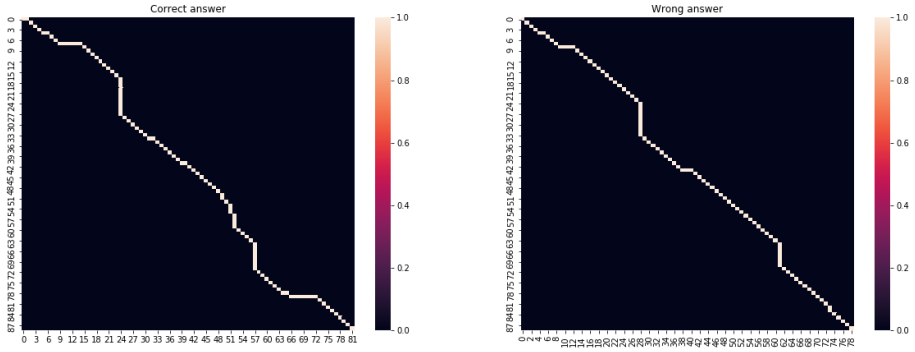

図 1

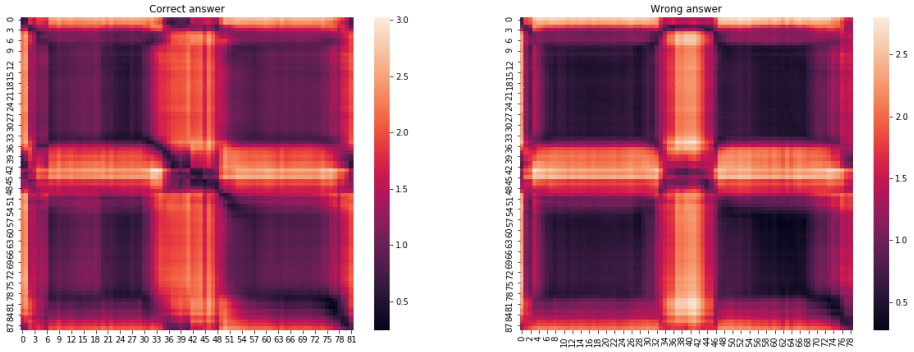

図 2

5.1.3　話者 2 の 1 回目と話者 2 の 1 回目の発声

プログラムの変数と結果を表 4 に示す。同一の発声データなので正答率は 100[%] になった。

表 4

$talker1$	2
$talknum1$	1
$talker2$	2
$talknum2$	1
正答率 [%]	100

5.1.4　話者 2 の 1 回目と話者 2 の 2 回目の発声

プログラムの変数と結果を表 5 に示す。同一話者のため、正答率比較的高いは 99[%] となった。

誤答は、15 番目の単語「SOSHIGAYA」と 54 番目の単語「ZOOSHIGAYA」が一致すると認識される誤答であった。これらは見たところ似たような単語であるため、誤答してしまったと考えられる。

正答と誤答の比較として、図 3 に DP マッチングによる経路、図 4 に局所距離を色の濃淡で表したヒートマップを示す。これらを見ると、正答の経路は一直線で最短のように見え、誤答の経路は曲がりくねっているため最短でないように見えるが、実際は誤答の経路

のほうが累積距離は短いということがわかる。

表 5

$talker1$	2
$talknum1$	1
$talker2$	2
$talknum2$	2
正答率 [%]	99

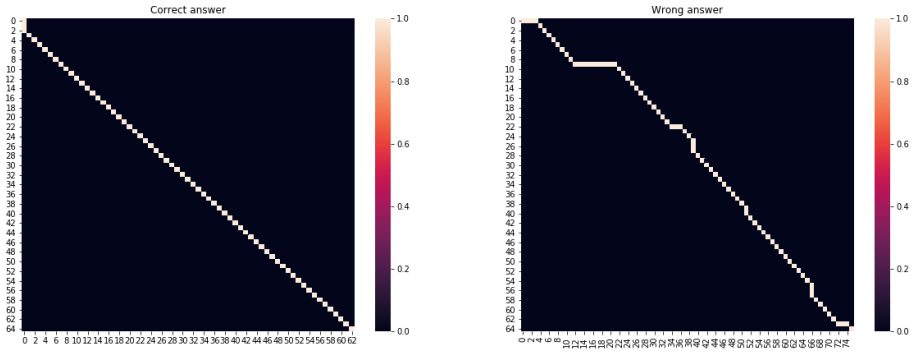

図 3

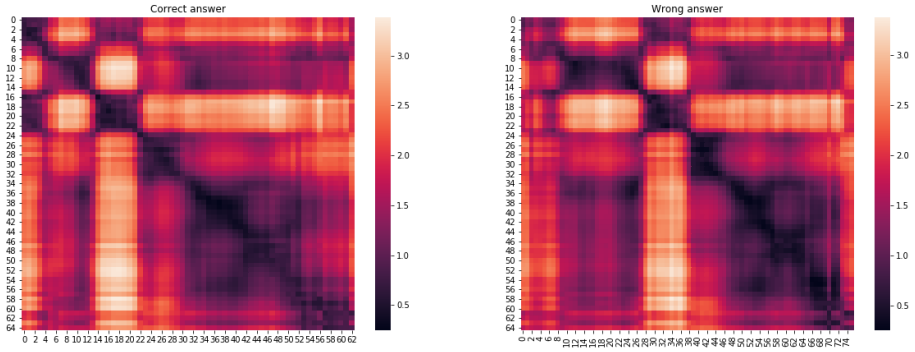

図 4

5.2　異なる話者

5.2.1　話者 1 の 1 回目と話者 2 の 1 回目の発声

プログラムの変数と結果を表 6 に示す。別話者のため、正答率は少し低い 90[%] となった。

誤答の 1 つは、13 番目の単語「SUGAMO」と 3 番目の単語「UENO」が一致すると認識される誤答であった。これらは見たところ似ていない単語だが、別話者であるため誤答してしまったと考えられる。

正答と誤答の比較の一例として、図 5 に DP マッチングによる経路、図 6 に局所距離を色の濃淡で表したヒートマップを示す。これらを見ると、経路の形が違っていることがわかる。

表 6

$talker1$	1
$talknum1$	1
$talker2$	2
$talknum2$	1
正答率 [%]	90

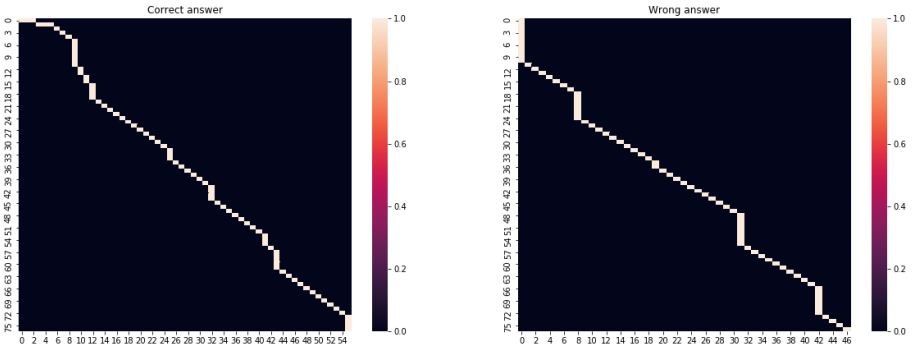

図 5

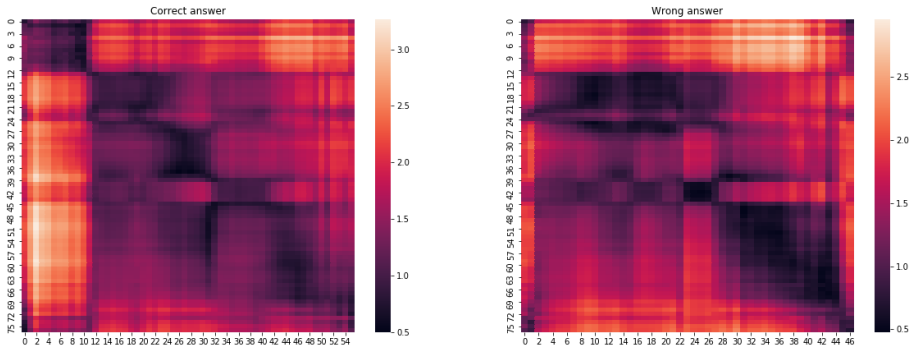

図 6

5.2.2　話者 1 の 1 回目と話者 2 の 2 回目の発声

プログラムの変数と結果を表 7 に示す。別話者のため、正答率は少し低い 84[%] となった。

誤答の 1 つは、93 番目の単語「JUUMONJI」と 7 番目の単語「KICHIJOOJI」が一致すると認識される誤答であった。これらは見たところ少しだけ似ている単語であるため、誤答してしまったと考えられる。

正答と誤答の比較の一例として、図 7 に DP マッチングによる経路、図 8 に局所距離を色の濃淡で表したヒートマップを示す。これらを見ると、経路の前半の形が大きく違っていることがわかる。

表 7

$talker1$	1
$talknum1$	1
$talker2$	2
$talknum2$	2
正答率 [%]	84

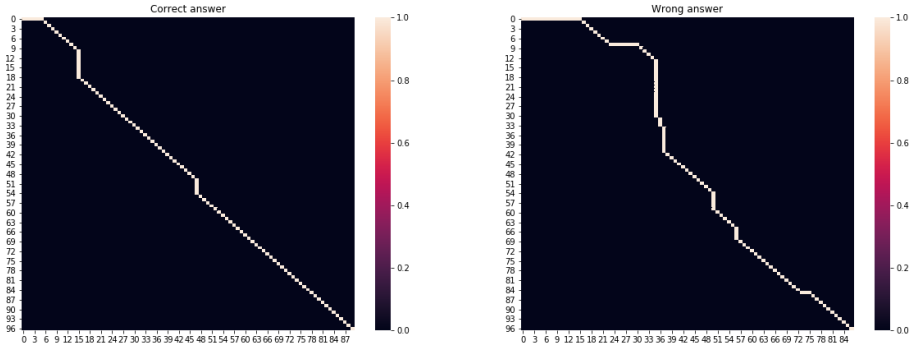

図 7

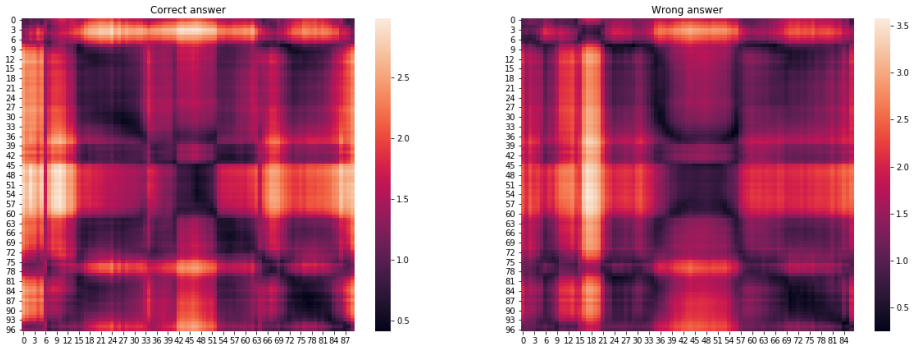

図 8

5.2.3　話者 1 の 2 回目と話者 2 の 1 回目の発声

プログラムの変数と結果を表 8 に示す。別話者のため、正答率は少し低い 92[%] となった。

誤答の 1 つは、42 番目の単語「REBUN」と 59 番目の単語「BIZEN」が一致すると認識される誤答であった。これらは見たところ似ていない単語だが、別話者であるため誤答してしまったと考えられる。

正答と誤答の比較の一例として、図 9 に DP マッチングによる経路、図 10 に局所距離を色の濃淡で表したヒートマップを示す。これらを見ると、両者の経路は似ていないことがわかる。

表 8

$talker1$	1
$talknum1$	2
$talker2$	2
$talknum2$	1
正答率 [%]	92

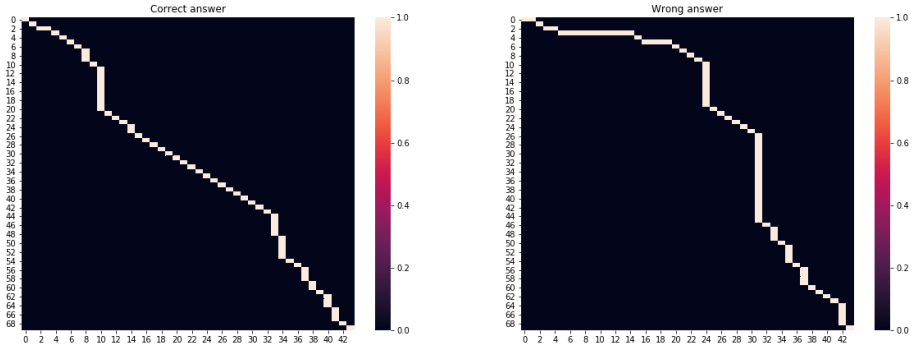

図 9

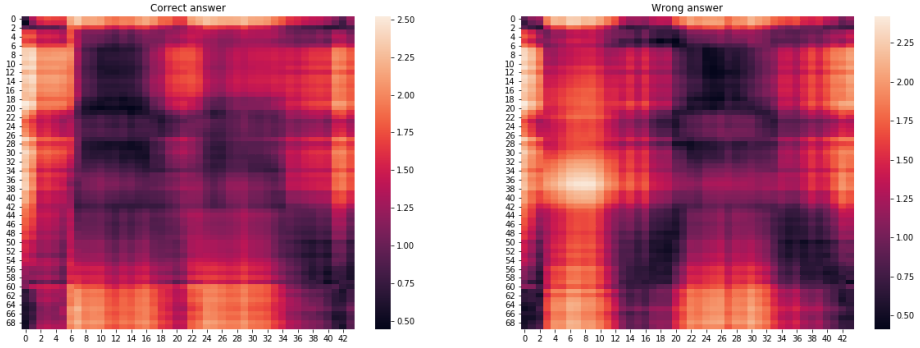

図 10

5.2.4　話者 1 の 2 回目と話者 2 の 2 回目の発声

プログラムの変数と結果を表 9 に示す。別話者のため、正答率は少し低い 86[%] となった。

誤答の 1 つは、19 番目の単語「TENRYUU」と 14 番目の単語「SENJU」が一致すると認識される誤答であった。これらは見たところ語感が似ている単語のため、誤答したと考えられる。

正答と誤答の比較の一例として、図 11 に DP マッチングによる経路、図 12 に局所距離を色の濃淡で表したヒートマップを示す。これらを見ると、両者はよく似た経路になっていることがわかる。

表 9

$talker1$	1
$talknum1$	2
$talker2$	2
$talknum2$	2
正答率 [%]	86

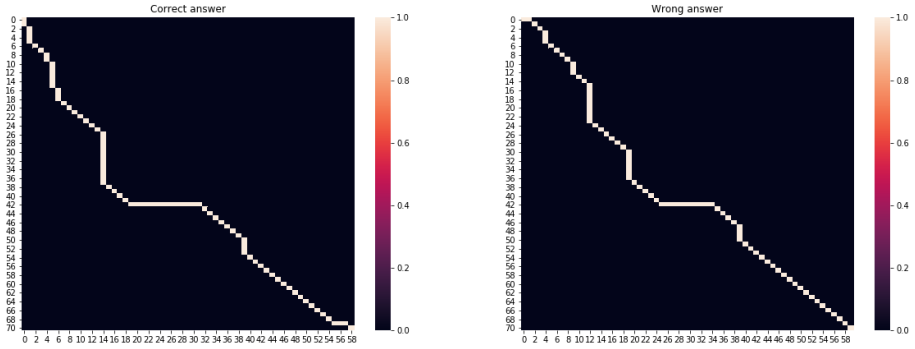

図 11

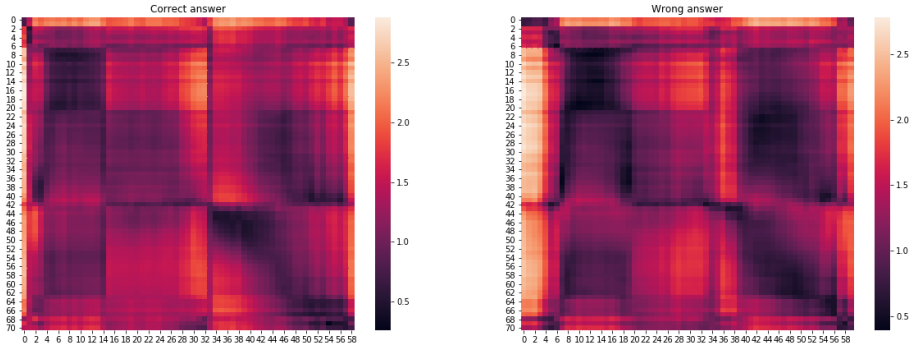

図 12

6　考察

実験から、同一話者の場合の正答率は 99[%] 以上となり、別話者の場合は 80[%] 以上となった。これは音声認識の精度としては十分実用的であると考えられる。

誤答してしまった結果を見てみると、「SOSHIGAYA」と「ZOOSHIGAYA」のように似ている単語が誤答されやすい傾向にあることがわかる。これは、1 つのテンプレートを用いるのではなく、いくつかのテンプレートを組み合わせて未知入力と DP マッチングを行うことで、正答率を上げることができると考えられる。また、誤答した単語の DP マッチングで求めた経路を比較してみると、非常に似ている経路もあれば、一見すると似ていないような経路もあることがわかる。つまり、経路の類似度が単語の類似度にはならないということがわかる。

7　まとめ

単語音声認識を DP マッチングで行うプログラムを作成し、その性能を調べた。結果は 8 割以上の正答率を得られることができた。実験結果から、似ている単語は誤答しやすいことがわかった。また、DP マッチングで得られた経路の類似度は単語の類似度と一致しないことが分かった。

付録 A　ソースコード

Listing 1　DP マッチングの Python コード

```
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns

class WordSpeechRecognition:
    def __init__(self):
        pass

    def dataSet(self, talker, talkNum):
        data = []
        info = []
        talker = str(talker)
        talkNum = str(talkNum)
        for wordNum in range(100):
            fileName = "city0" + talker + talkNum + "/city0" +
                talker + talkNum + "_" + "{:0=3}".format(wordNum +
                1) + ".txt"
            f = open(fileName)
            textList = f.read().split() # データ読み込み

            info.append(textList[0 : 3]) # 情報部分抽出
            frame = int(textList[2]) # フレーム数格納
            strData = textList[3 : len(textList)] # データ部分抽出
            numData = list(map(float, strData)) # に変換 float
            arrayData = np.array(numData) # 行列にする

            tempData = list(np.reshape(arrayData, (frame, 15))) #
                データをフレームごとに分
                ける
            data.append(tempData) # 個の行列のリストを作る 100

        f.close()
```

```
30          return data, info
31
32      def CalcDistance(self, wordNum1, wordNum2, data1, data2): #13:50
33          line = int(data1[1][wordNum1][2])
34          row = int(data2[1][wordNum2][2])
35          d = np.zeros((line, row))
36
37          for i in range(line):
38              for j in range(row):
39                  sum = 0
40                  for k in range(15):
41                      sum += (data1[0][wordNum1][i][k] - data2[0][
                            wordNum2][j][k]) ** 2
42
43                  d[i][j] = math.sqrt(sum)
44
45          return d
46
47      def DPmatching(self, d):
48          lens = d.shape
49          g = np.zeros(lens)
50          g[0][0] = d[0][0]
51
52          for i in range(1, lens[0]):
53              g[i][0] = g[i - 1][0] + d[i][0]
54          for j in range(1, lens[1]):
55              g[0][j] = g[0][j - 1] + d[0][j]
56
57          for m in range(1, lens[0]):
58              for n in range(1, lens[1]):
59                  g1 = g[m][n - 1] + d[m][n]
60                  g2 = g[m - 1][n - 1] + 2 * d[m][n]
61                  g3 = g[m - 1][n] + d[m][n]
62                  gs = [g1, g2, g3]
63
64                  g[m][n] = min(gs)
65
66          min_g = g[-1][-1] / sum(lens)
```

```
67          return [min_g, g]
68
69      def PathPlot(self, g):
70          lens = g.shape
71          RoutMap = np.zeros(lens)
72          RoutMap[-1][-1] = 1
73
74          m = lens[0] - 1
75          n = lens[1] - 1
76          while m != 0 or n != 0:
77
78              refList = [g[m - 1][n], g[m - 1][n - 1], g[m][n - 1]]
79
80              minNum = min(refList)
81              index = refList.index(minNum)
82
83              if index == 0:
84                  RoutMap[m - 1][n] = 1
85                  m = m - 1
86              elif index == 1:
87                  RoutMap[m - 1][n - 1] = 1
88                  m = m - 1
89                  n = n - 1
90              else:
91                  RoutMap[m][n - 1] = 1
92                  n = n - 1
93
94          sns.heatmap(RoutMap)
95
96  if __name__ == '__main__':
97      talker1 = 1
98      talknum1 = 1
99      talker2 = 2
100     talknum2 = 2
101
102     wsr = WordSpeechRecognition()
103     data1 = wsr.dataSet(talker1, talknum1)
104     data2 = wsr.dataSet(talker2, talknum2)
```

```
105
106         templeteNum = 100
107         inputNum = 100
108
109         # すべての局所距離を計算する（時間がかかる）
110         ds = [[0 for j in range(inputNum)] for i in range(templeteNum)]
111
112         for i in range(templeteNum):
113             for j in range(inputNum):
114                 ds[i][j] = wsr.CalcDistance(i, j, data1, data2) # (word1
                        , word2)
115
116             print(i)
117
118         # 100単語を総当たりでマッチング x100DP
119         store_gs_all = [[0 for j in range(inputNum)] for i in range(
                templeteNum)]
120
121         store_gs = [[0 for j in range(2)] for i in range(100)] # 列
                [()行 ()  ]を保存する変数 gs
122         muchWordNums = np.zeros([inputNum, 2])
123         muchWordNums[:,:] = np.nan
124         for i in range(templeteNum):
125             for j in range(inputNum):
126                 temp_gs = wsr.DPmatching(ds[i][j])
127                 store_gs_all[i][j] = temp_gs
128
129                 if j == 0:
130                     gs = temp_gs
131                     store_gs[i] = gs
132                     muchWordNums[i] = [i+1, j+1]
133                 if j != 0 and gs[0] > temp_gs[0]:
134                     gs = temp_gs
135                     store_gs[i] = gs
136                     muchWordNums[i] = [i+1, j+1]
137
138             print(muchWordNums[i])
139
```

```
140         # 正答率
141         cnt = 0
142         for i in range(100):
143             if muchWordNums[i][0] == muchWordNums[i][1]:
144                 cnt = cnt + 1
145 
146         print(cnt)
147 
148         # 経路表示
149         correct = 19 - 1
150         wrong = 14 - 1
151 
152         fig = plt.figure(figsize=(20,7))
153 
154         fig.add_subplot(1, 2, 1)
155         wsr.PathPlot(store_gs_all[correct][correct][1])
156         plt.title('Correct answer')
157 
158         fig.add_subplot(1, 2, 2)
159         wsr.PathPlot(store_gs_all[correct][wrong][1])
160         plt.title('Wrong answer')
161 
162         #fig.savefig("pics/1122_1919_1914_path.png")
163 
164 
165         print("Correct distance:", store_gs_all[correct][correct][0])
166         print("Wrong distance:", store_gs_all[correct][wrong][0])
167 
168         # ヒートマップ表示
169         fig = plt.figure(figsize=(20,7))
170 
171         fig.add_subplot(1, 2, 1)
172         sns.heatmap(ds[correct][correct])
173         plt.title('Correct answer')
174 
175         fig.add_subplot(1, 2, 2)
176         sns.heatmap(ds[correct][wrong])
177         plt.title('Wrong answer')
```

```
178
179         #fig.savefig("pics/1122_1919_1914_heat.png")
```