DPマッチングによる単語音声の認識

麻生英寿
21C1005

June 29, 2023

Contents

1 目的

DP マッチングのアルゴリズムを利用して、小語彙の音声認識の実験を行う。

2 実験方法

DP マッチングのアルゴリズムによって音声データの単語間距離を計算するプログラムを作成する。そのプログラムに、100 単語の音声データのテンプレートに対して、同じ発声内容の 100 単語を未知入力音声として、順に入力していく。入力された音声データの発声内容を判定し、その正答率を計算する。

2.1 使用したデータセットについて

この実験で使用した音声データセットには、2 人の話者がそれぞれ同じ 100 種類の単語を発話したものが含まれている。話者は 100 種類の単語の発話を 2 回行っているため、合計 400 個の音声データが含まれている。1 つの音声データにはファイル名、発声内容、フレーム数とフレーム数分の 15 次のメルケプストラム特徴量である。

2.2 単語間距離の計算

実験ではDPマッチングのアルゴリズムを用いて2つの音声データの単語間距離を計算する。単語間距離の計算は以下の手順で行う。

入力として与えられた2つの音声データのフレーム長をそれぞれIとJとする。$(i,j), 0 < i \leq I, 0 < j \leq J$で表せられる2つの音声データの各ノードのメルケプストラム特徴量の距離を計算し、(i,j)でのフレームの距離を局所距離$d(i,j)$と表す。音声データの最初のフレームから任意のフレームまでの局所距離の総和を累積距離$g(i,j)$とする。最終フレームでの累積距離$g(I,J)$が最小となるような経路を探索することで、単語間距離を計算する。

2.3 最終フレームでの累積距離の計算

はじめに、初期条件を以下のように設定する。

$$g(0,0) = d(0,0)$$

次に境界条件を以下のように設定する。

$$j > 0 \rightarrow g(0,j) = g(0,j-1) + d(0,j)$$

$$i > 0 \rightarrow g(i,0) = g(i-1,0) + d(i,0)$$

最後に、再帰的に以下の式を用いて、最終フレームでの累積距離$g(I,J)$を計算する。

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

3 実験結果

音声認識の正答率は次のようになった。

モデル/認識対象	city011	city012	city021	city022
city011		99%	90%	84%
city012	100%		92%	86%
city021	83%	91%		99%
city022	86%	94%	100%	

Table 1: 音声認識の正答率

4 考察

実験結果より、同一話者よりも異なる話者の音声データの方が正答率が低いことが読み取れる。

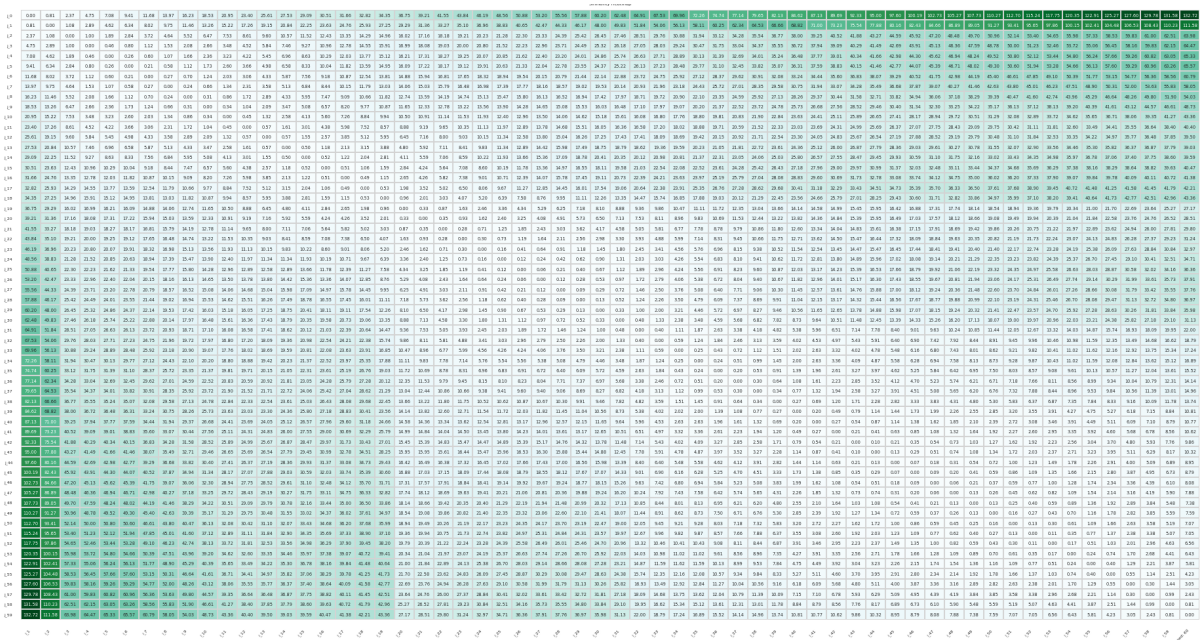

Figure 1: 同じ音声データ

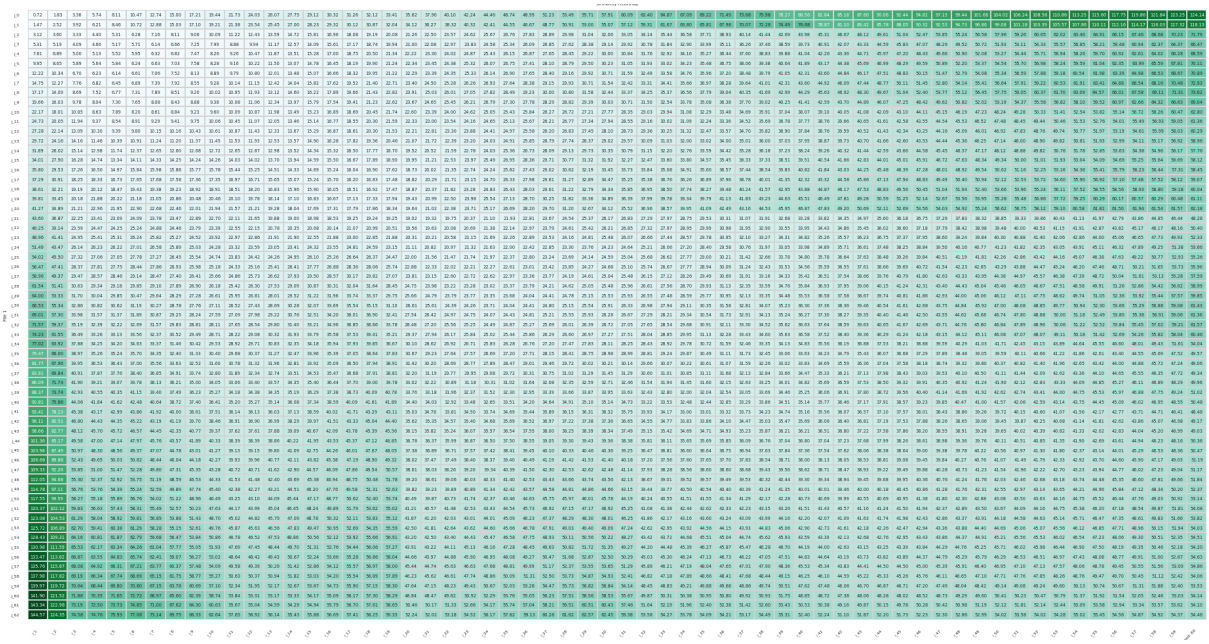

Figure 2: 同一話者

A 付録

この実験で使用したプログラムのソースコードを以下に示す。

A.1 付録 A: 単語間距離を求めるプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define STRING_SIZE 64
#define PRINT_CSV
#define EXPORT_G
struct FILEDATA{
    char filename[STRING_SIZE];
    char mean[STRING_SIZE];
    int frame_num;
    double *voice_data;
};

int ReadFile(char* filename, struct FILEDATA *filedata){
    FILE *fp;
    fp = fopen(filename, "r");
    if(fp == NULL) return -1;

    fgets(filedata->filename, STRING_SIZE, fp);
    fgets(filedata->mean, STRING_SIZE, fp);
    char _buf[STRING_SIZE];
    fgets(_buf, STRING_SIZE, fp);
    sscanf(_buf, "%d", &filedata->frame_num);

    double* data;
    data = (double*)malloc(sizeof(double)*filedata->frame_num*15);
```

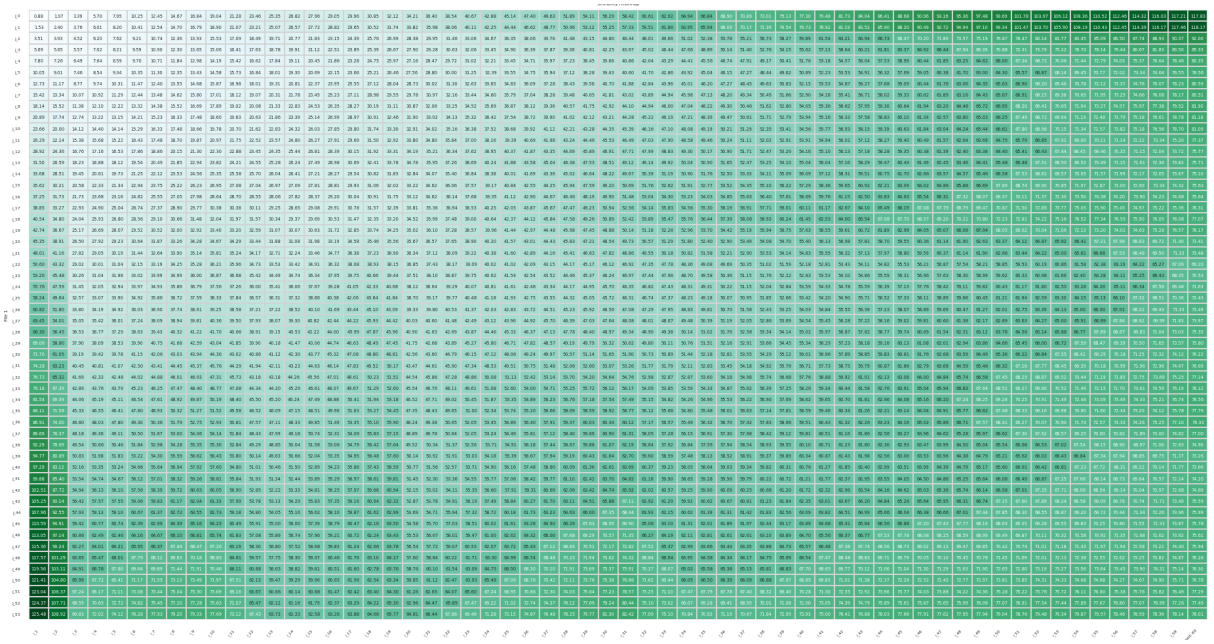

Figure 3: 異なる話者

```
    if(data == NULL) return -1;
    for(int i=0; i<filedata->frame_num; i++){
        for(int j=0; j<15; j++){
            fscanf(fp, "%lf", &data[i*15+j]);
        }
    }

    filedata->voice_data = data;

    fclose(fp);
    return 0;
}

void CalcLocalDistance(struct FILEDATA *answer, struct FILEDATA *
    question, double** d){
    for(int i=0; i<answer->frame_num; i++){
        for(int j=0; j<question->frame_num; j++){
            double sum = 0,a=0,q=0;
            for(int k=0; k<15; k++){
                a = answer->voice_data[i*15+k];
                q = question->voice_data[j*15+k];
                sum += (a-q)*(a-q);
            }
            d[i][j] = sqrt(sum);
        }
    }
}

void CalcCumulativeDistance(struct FILEDATA *answer, struct FILEDATA *
    question, double** d,double** g){
    g[0][0] = d[0][0];
    for(int i=1;i<answer->frame_num;i++){
        g[i][0] = g[i-1][0] + d[i][0];
    }
    for(int j=1;j<question->frame_num;j++){
```

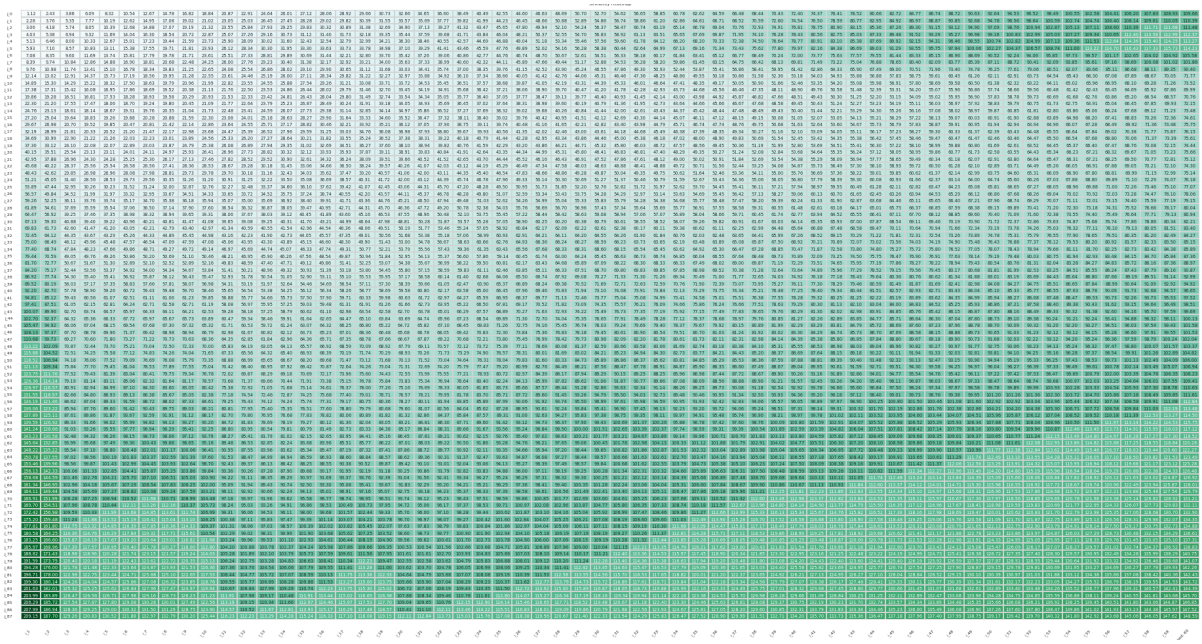

Figure 4: 異なる話者かつ異なる単語

```
61              g[0][j] = g[0][j-1] + d[0][j];
62          }
63          for(int i=1;i<answer->frame_num;i++){
64              for(int j=1;j<question->frame_num;j++){
65                  double a = g[i][j-1] + d[i][j];
66                  double b = g[i-1][j] + d[i][j];
67                  double c = g[i-1][j-1] + 2*d[i][j];
68                  double min = a;
69                  if(min > b) min = b;
70                  if(min > c) min = c;
71                  g[i][j] = min;
72              }
73          }
74      }
75
76      int main(int argc,char *argv[]){
77          if(argc <= 2){
78              printf("Enter␣the␣files␣name.\n");
79              return -1;
80          }
81
82          struct FILEDATA answer;
83          struct FILEDATA question;
84
85          if(ReadFile(argv[1], &answer) == -1){
86              printf("Answer␣File␣open␣error.\n");
87              return -1;
88          }
89          if(ReadFile(argv[2], &question) == -1){
90              printf("Question␣File␣open␣error.\n");
91              return -1;
92          }
93
94          double** d = (double**)malloc(sizeof(double*)*answer.frame_num);
95          if(d == NULL) return -1;
```

```
96      for(int i=0; i<answer.frame_num; i++){
97          d[i] = (double*)malloc(sizeof(double)*question.frame_num);
98          if(d[i] == NULL){
99              for(int k = 0;k<i;k++){
100                 free(d[k]);
101             }
102             free(d);
103             return -1;
104         }
105     }
106
107     double** g = (double**)malloc(sizeof(double*)*answer.frame_num);
108     if(g == NULL) return -1;
109     for(int i=0; i<answer.frame_num; i++){
110         g[i] = (double*)malloc(sizeof(double)*question.frame_num);
111         if(g[i] == NULL){
112             for(int k = 0;k<i;k++){
113                 free(g[k]);
114             }
115             free(g);
116             return -1;
117         }
118     }
119
120     CalcLocalDistance(&answer, &question, d);
121
122
123     CalcCumulativeDistance(&answer, &question, d,g);
124
125     int frame_i = answer.frame_num-1;
126     int frame_j = question.frame_num-1;
127     double distance = g[frame_i][frame_j];
128
129
130     #ifdef PRINT_CSV
131     printf("%f",distance/(double)(answer.frame_num+question.frame_num));
132     #else
133     #ifdef EXPORT_G
134     for(int i=0;i<frame_i;i++){
135         printf("i_%d,",i);
136     }
137     printf("\n");
138     for(int j=0;j<frame_j;j++){
139         printf("j_%d,",j);
140         for(int i=0;i<frame_i-1;i++){
141             printf("%f,",g[i][j]);
142         }
143         printf("%f\n",g[frame_i-1][j]);
144     }
145     printf("\n");
146     #else
147     printf("answer␣word␣is␣%s",answer.mean);
148     printf("question␣word␣is␣%s",question.mean);
149     printf("size␣=␣%dx%d\n", answer.frame_num,question.frame_num);
150     printf("distance␣=␣%f\n", distance);
151     printf("distance␣between␣answer␣and␣question␣=␣%f\n", distance/(
            answer.frame_num+question.frame_num));
152     #endif
153     #endif
```

```
154
155     free(answer.voice_data);
156     free(question.voice_data);
157     for(int i=0;i<answer.frame_num;i++){
158         free(d[i]);
159         free(g[i]);
160     }
161     free(d);
162     free(g);
163
164     return 0;
165 }
```

A.2 付録 B: 正答率を計算するプログラム

```
1  import os
2  import subprocess
3  import sys
4
5  def calculate_similarity(file1, file2):
6      arguments = [file1,file2]
7      #print(arguments)
8
9      process = subprocess.run([executable_path] + arguments,
           capture_output=True, text=True)
10     similarity = float(process.stdout)
11     return similarity
12
13 def correct_rate(dir1, dir2):
14     raw_paths1 = [os.path.join(dir1, f) for f in os.listdir(dir1) if os
           .path.isfile(os.path.join(dir1, f))]
15     raw_paths2 = [os.path.join(dir2, f) for f in os.listdir(dir2) if os
           .path.isfile(os.path.join(dir2, f))]
16     file_paths1 = sorted(raw_paths1)
17     file_paths2 = sorted(raw_paths2)
18
19     incorrect = 0
20     q = 1
21     a = 1
22
23     for path1 in file_paths1:
24         min = 10.
25         a = 1
26         for path2 in file_paths2:
27             similarity = calculate_similarity(path1, path2)
28             if q == a and similarity > min:
29                 print(f'{q}␣Incorrect')
30                 incorrect+=1
31                 break
32             if q<a and similarity < min:
33                 print(f'{q}␣Incorrect')
34                 incorrect+=1
35                 break
36             if similarity < min:
37                 min = similarity
38             a+=1
39         q+=1
```

```
        #print(f'\r{round(q/len(file_paths1)*100.0)}%', end='')

    print()
    return (a-incorrect)/q

if __name__ == "__main__":
    args = sys.argv

    executable_path = args[1]
    dir1_path = args[2]
    dir2_path = args[3]

    rate = correct_rate(dir1_path, dir2_path)
    print(f"CorrectRate: {round(rate*100.0)}%")
```

A.3 付録 C: 可視化するプログラム

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sys

# ファイルのパス
args = sys.argv
csv_file = args[1]

# ファイルを読み込む
df = pd.read_csv(csv_file, index_col=0)

# ヒートマップを描画
plt.figure(figsize=(20, 20))
sns.set(font_scale=0.4) # フォントサイズを調整
sns.heatmap(df, cmap='BuGn', annot=True, fmt=".2f", cbar=False, vmin=df
    .min().min(), vmax=df.max().max(), linewidths=0.1, linecolor='
    lightgray')
plt.xlabel('File 2')
plt.ylabel('File 1')
plt.title('Similarity Heatmap')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```