

生物情報科学演習 課題2 クラスタリング

生物情報科学科 05-145508 谷川洋介 (yk.tanigawa@gmail.com)

2015 年 1 月 8 日

1 課題 1: 古典的な Lloyd の方法の実装

1.1 データ生成

データの生成のために、次のようなプログラムを書いた。このプログラムは n 個の点 $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ ($\forall i. \mathbf{x}_i \in [0, 1]^d$) を C++11 の乱数ライブラリのメルセヌスツイスタを用いて生成する。生成されたデータは、ファイルに書きだされる。

データの個数 n , 次元の大きさ d は引数として与える。また, reproducible research の観点から重要である, 乱数のシードも引数として与えることができる。プログラムがデータの生成の際に使用したシードの値は標準出力に表示される。

ソースコード 1 rand_data_generator.cpp (データ生成のためのプログラム)

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <fstream>
4 #include <random>
5
6 using namespace std;
7
8 /* define constant values */
9 static const double DATA_RANGE_MIN = 0.0;
10 static const double DATA_RANGE_MAX = 1.0;
11
12 #define DEBUG 0
13
14 int main(int argc, char *argv[]){
15     if(argc < 4){
16         cerr << "usage: " << argv[0]
17         << " <n: data num> <d: dimension> <file name> [seed]" << endl;
18         exit(1);
19     }else{
20         int n = atoi(argv[1]), d = atoi(argv[2]), seed;
21         char *file = argv[3];
22         {
23             std::random_device rd;
24             if(argc < 5){
25 #if DEBUG
26                 seed = 0;
27 #else
28                 seed = rd();
29 #endif
30             }else{
31                 seed = atoi(argv[4]);
32             }
33         }
34         cout << "seed = " << seed << endl;
35         {
```

```

36     ofstream fs(file);
37     if(fs.fail()){ exit(1); }
38     {
39     std::mt19937 mt(seed);
40     std::uniform_real_distribution<double> rand(DATA_RANGE_MIN, DATA_RANGE_MAX);
41
42     for(int i = 0; i < n; ++i){
43         for(int k = 0; k < d - 1; ++k){
44             fs << rand(mt) << " ";
45         }
46         fs << rand(mt) << endl;
47     }
48     }
49     fs.close();
50 }
51 return 0;
52 }
53 }

```

1.2 プログラムの実装

ベクトルの和などの基本的な演算は、ヘッダーファイル `my_vector.hpp` 内に実装した。

ソースコード 2 `my_vector.hpp` (ベクトルの基本演算)

```

1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <cmath>
5
6  using namespace std;
7
8  /* ベクトルに対する加算・減算・定数倍などを定義 */
9  template <class T>
10 std::vector<T>& operator+=(std::vector<T> &self,
11                          const std::vector<T> &other){
12     for(int i = 0; i < (int)self.size(); i++)
13         self[i] += other[i];
14     return self;
15 }
16
17 template <class T>
18 std::vector<T> operator+(const std::vector<T> &self,
19                        const std::vector<T> &other){
20     std::vector<T> result = self;
21     result += other;
22     return result;
23 }
24
25 template <class T>
26 std::vector<T>& operator-=(std::vector<T> &self,
27                          const std::vector<T> &other){
28     for(int i = 0; i < (int)self.size(); i++)
29         self[i] -= other[i];
30     return self;
31 }
32
33 template <class T>

```

```

34 std::vector<T> operator-(const std::vector<T> &self,
35                          const std::vector<T> &other){
36     std::vector<T> result = self;
37     result -= other;
38     return result;
39 }
40
41 template <class T>
42 std::vector<T>& operator*=(std::vector<T> &self, const T &mul){
43     for(int i = 0; i < (int)self.size(); i++)
44         self[i] *= mul;
45     return self;
46 }
47
48 template <class T>
49 std::vector<T> operator*(const std::vector<T> &self,
50                          const T &mul){
51     std::vector<T> result = self;
52     result *= mul;
53     return result;
54 }
55
56 template <class T>
57 std::vector<T> operator*(const T &mul, const std::vector<T> &self){
58     std::vector<T> result = self;
59     result *= mul;
60     return result;
61 }
62
63 template <class T>
64 ostream &operator<<(ostream &stream, vector<T> vector){
65     for(int i = 0; i < (int)vector.size() - 1; i++){
66         stream << vector.at(i) << ", ";
67     }
68     stream << vector.back() << endl;
69     return stream;
70 }
71
72 template <class T>
73 ostream &operator<<(ostream &stream, vector< vector<T> > matrix){
74     for(int i = 0; i < (int)matrix.size(); i++){ stream << matrix.at(i); }
75     return stream;
76 }
77
78 template <class T>
79 inline double Euclid_norm(vector<T> vec){
80     /* compute Euclid norm of a vector 'vec' */
81     double results = 0;
82     for(int i = 0; i < (int)vec.size(); ++i){
83         results += vec.at(i) * vec.at(i);
84     }
85     return sqrt(results);
86 }
87
88 template <class T>
89 inline double sum(vector<T> vec){
90     double results = 0;
91     for(int i = 0; i < (int)vec.size(); ++i){

```

```

92     results += vec.at(i);
93 }
94 return results;
95 }
96
97 template <class T>
98 double norm(vector<T> vec, int p = 2){
99     /* compute norm of a vector 'vec' */
100    if(p == 2){
101        return Euclid_norm(vec);
102    }else if(p == 1){
103        return sum(vec);
104    }else{
105        double results = 0;
106        for(int i = 0; i < (int)vec.size(); ++i){
107            results += pow(vec.at(i), p);
108        }
109        return pow(results, 1.0 / p);
110    }
111 }
112
113 template <class T>
114 double inner_product(vector<T> v1, vector<T> v2){
115     /* compute inner product of two vectors v1, v2 */
116     double results = 0;
117     int minsize = (((int)v1.size() < (int)v2.size())
118         ? (int)v1.size() : (int)v2.size());
119     for(int i = 0; i < minsize; ++i){
120         results += v1.at(i) * v2.at(i);
121     }
122     return results;
123 }

```

プログラムの実行時間を計測するためのサブルーチンは、ヘッダーファイル `time_bench.hpp` 内に実装した。

ソースコード 3 time_bench.hpp (実行時間計測のためのサブルーチン)

```

1  #include <time.h>
2  #include <sys/time.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5
6  long unsigned int diff_timeval(struct timeval start, struct timeval end){
7      /* struct timeval の差をmicro秒の単位で返す */
8      time_t diff_sec = end.tv_sec - start.tv_sec;
9      suseconds_t diff_usec = end.tv_usec - start.tv_usec;
10     return ((long unsigned int)diff_sec*1000000 + diff_usec);
11 }
12
13 #if 0 /* usage */
14 {
15     struct timeval t_start, t_end;
16     gettimeofday(&t_start, NULL);
17     {
18         /* 計測したい処理 */
19     }
20     gettimeofday(&t_end, NULL);
21     diff_timeval(t_start, t_end);
22 }

```

23 #endif

プログラムの本体は、Lloyd.cpp 内に実装した。

ソースコード 4 Lloyd.cpp (Lloyd の K-means アルゴリズム)

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <vector>
5  #include <cstdlib>
6  #include <time.h>
7  #include <sys/time.h>
8  #include <sys/types.h>
9  #include <unistd.h>
10
11 #include "my_vector.hpp"
12 #include "time_bench.hpp"
13
14 using namespace std;
15
16 void re_clustering(const int n, const int d, const int k,
17     vector<int> &cluster,
18     const vector<vector<double>> > representative,
19     const vector<vector<double>> > data){
20     for(int i = 0; i < n; ++i){
21         int closest = 0;
22         double dist = norm(data.at(i) - representative.at(closest));
23         double min_dist = dist;
24         for(int z = 1; z < k; ++z){
25             dist = norm(data.at(i) - representative.at(z));
26             if(dist < min_dist){
27                 min_dist = dist;
28                 closest = z;
29             }
30         }
31         cluster.at(i) = closest;
32     }
33     return;
34 }
35
36 void calc_representative(const int n, const int d, const int k,
37     const vector<int> cluster,
38     vector<vector<double>> &representative,
39     const vector<vector<double>> > data){
40     vector<int> count(k, 0); /* 各クラスタに何点あるか数える */
41     vector<vector<double>> sum(k); /* 各クラスタの座標の和 */
42     {
43         vector<double> zero(d, 0);
44         for(int z = 0; z < k; ++z){
45             sum.at(z) = zero;
46         }
47     }
48
49     /* それぞれのクラスタについて全データ点の座標の和を計算 */
50     for(int i = 0; i < n; ++i){
51         count.at(cluster.at(i))++;
52         sum.at(cluster.at(i)) += data.at(i);
53     }
```

```

54
55  /* 代表点を更新する */
56  for(int z = 0; z < k; ++z){
57      representative.at(z) = (1.0 / count.at(z)) * sum.at(z);
58  }
59
60  return;
61 }
62
63 double average_square_err(const int n, const int d, const int k,
64     const vector<int> cluster,
65     const vector<vector<double> > representative,
66     const vector<vector<double> > data){
67     double sum = 0;
68     for(int i = 0; i < n; ++i){
69         sum += norm(data.at(i) - representative.at(cluster.at(i))) * norm(data.at(i) -
70             representative.at(cluster.at(i)));
71     }
72     return (sum / n);
73 }
74
75 double Lloyd_repeat(const int n, const int d, const int k,
76     vector<int> &cluster,
77     vector<vector<double> > &representative,
78     const vector<vector<double> > data){
79     re_clustering(n, d, k, cluster, representative, data);
80     calc_representative(n, d, k, cluster, representative, data);
81
82     return average_square_err(n, d, k, cluster, representative, data);
83 }
84
85 void Lloyd(const int n, const int d, const int k,
86     const vector<vector<double> > data){
87     if(n < k){
88         cerr << "data size n is smaller than cluster size d" << endl;
89         exit(1);
90     }else{
91
92         struct timeval t_start, t_end;
93         gettimeofday(&t_start, NULL); /* 時間計測開始 */
94
95         vector<int> cluster(n);
96         /* 各点  $x_i$  がどのクラスに属しているか */
97         vector<vector<double> > representative(k);
98         /* 各クラスの代表点 */
99         for(int z = 0; z < k; ++z){ /* ランダムに初期化する */
100             representative.at(z) = data.at(z);
101         }
102
103         double sq_err, sq_err_before = Lloyd_repeat(n, d, k, cluster, representative, data);
104         int t = 1; /* 繰り返しを何ステップ行ったか */
105         while(1){
106             sq_err = Lloyd_repeat(n, d, k, cluster, representative, data);
107             if(sq_err == sq_err_before){ break; }
108             t++;
109             sq_err_before = sq_err;
110         }

```

```

111     gettimeofday(&t_end, NULL); /* 時間計測終了 */
112
113     cout << n << "\t"           /* サンプル数 */
114     << d << "\t"               /* 次元 */
115     << k << "\t"               /* クラスタ数 */
116     << diff_timeval(t_start, t_end) << "\t" /* 実行時間(us) */
117     << sq_err << "\t"          /* 平均二乗誤差 */
118     << t << endl;              /* 繰り返し回数 */
119     return;
120 }
121 }
122
123 int main(int argc, char *argv[]){
124     if(argc < 5){
125         cerr << "usage: " << argv[0]
126         << " <n: data num> <d: dimension> <k: cluster num> <data file>" << endl;
127         exit(1);
128     }else{
129         int n = atoi(argv[1]), d = atoi(argv[2]), k = atoi(argv[3]);
130         char *file = argv[4];
131         vector<vector <double> > data(n);
132         {
133             ifstream fs(file);
134             if(fs.fail()){ exit(1); }
135             for(int i = 0; i < n; ++i){
136                 data.at(i).resize(d, 0);
137                 for(int j = 0; j < d; ++j){
138                     fs >> data.at(i).at(j);
139                 }
140             }
141             fs.close();
142         }
143         Lloyd(n, d, k, data);
144         return 0;
145     }
146 }

```

1.3 プログラムの実行

下記のようなシェルスクリプトによりプログラムを実行した。レポートとして提出するファイルの中には、生成したデータファイルは含まれていない。

ソースコード 5 exec.sh (プログラムの実行のためのスクリプト)

```

1  #!/bin/sh
2
3  make
4
5  seed=0
6  out_file="./results.txt"
7
8  for n in 1000 10000 100000
9  do
10     for d in 2 10 100
11     do
12         for k in 10 100 1000
13         do
14             file="./data/n${n}_d${d}.dat"

```

```

15     ./rand_data_generator ${n} ${d} ${file} ${seed}
16     ./Lloyd ${n} ${d} ${k} ${file} >> ${out_file}
17     done
18     done
19 done
20
21 make clean

```

1.4 実行結果

プログラムの実行結果を次に示す。各列は順に, n (データサイズ), d (次元), k (クラス数), プログラム実行時間 (マイクロ秒単位), 平均二乗誤差, 収束に要した繰り返し回数を表している。ただし, $n = 100000, d = 100, k = 1000$ のケースについては, プログラムの実行時間がとても長くなったので途中で実行を中止したため, データがない。

ソースコード 6 results.txt (プログラムの実行結果)

```

1 1000 2 10 145360 0.0178654 17
2 1000 2 100 684206 0.0015768 10
3 1000 2 1000 1150707 0 1
4 1000 10 10 393966 0.571266 30
5 1000 10 100 1081357 0.282467 10
6 1000 10 1000 1910273 0 1
7 1000 100 10 1471656 7.90357 23
8 1000 100 100 2963917 6.76475 5
9 1000 100 1000 9589658 0 1
10 10000 2 10 8903443 0.016817 108
11 10000 2 100 54266508 0.00164102 90
12 10000 2 1000 116081399 0.000137662 19
13 10000 10 10 19665332 0.595875 152
14 10000 10 100 76076117 0.342454 76
15 10000 10 1000 161746013 0.167454 16
16 10000 100 10 102368756 8.05287 160
17 10000 100 100 192431935 7.55779 38
18 10000 100 1000 765194613 6.41137 15
19 100000 2 10 62957070 0.0170205 75
20 100000 2 100 1678796753 0.00165097 281
21 100000 2 1000 4590171912 0.000159512 79
22 100000 10 10 326341487 0.601624 248
23 100000 10 100 2629831794 0.354804 266
24 100000 10 1000 7651529662 0.19913 79
25 100000 100 10 4464783923 8.09676 673
26 100000 100 100 12602891707 7.73588 252

```

1.5 データの解析

1.5.1 R によるデータのプロット

先の節で得た結果を解釈するために, R を用いてグラフを作成した。下記のプロットに用いたスクリプトを示す。

ソースコード 7 analysis.R (プロット用のスクリプト)

```

1 path = "~/moris/2_clustering/ytanigawa/"
2 fig_output_path = paste(path, "report/", sep="")
3
4 setwd(dir = path)
5 data <- read.table("results.txt")
6 name <- c("n", "d", "k", "time", "err", "rep")
7 names(data)=name

```



```

8 v_range<-rbind(c(1000,10000,100000), # 変数 n の値域
9              c(2,10,100),           # 変数 d の値域
10             c(10,100,1000))         # 変数 k の値域
11
12 for(variable in 1:3){                # グラフの横軸にする変数
13   for(observed in 4:6){              # グラフの縦軸にする変数
14     v2<-((variable %% 3) + 1)        # グラフの点の形を変化させる変数
15     v3<-(((variable + 1) %% 3) + 1)  # グラフの色を変化させる変数
16     v2_range<-v_range[v2,]          # 変数の値域をセット
17     v3_range<-v_range[v3,]          # 変数の値域をセット
18     png(paste(fig_output_path, name[variable], "_", name[observed], ".png", sep = ""),
19         pointsize = 30, width = 1200, height = 1200)
20                                     # 保存する png 画像の設定
21     par(new=F)                      # 前の画像に重ね書きしない
22     # par(xpd=T)                    # 凡例の枠外への描画を許可
23     x_lim<-c(min(data[variable]), max(data[variable]))
24     y_lim<-c(min(data[observed]), max(data[observed]))
25                                     # グラフの描画範囲の設定
26     plot(1,0, type="n", xlim=x_lim, ylim=y_lim, log="x",
27          main = paste(name[variable], " vs. ", name[observed], sep = ""),
28          xlab = name[variable], ylab = name[observed])
29                                     # 軸とタイトルのみからなるグラフを描画
30     cols<-rainbow(3)
31     for(v2_loop in 1:3){
32       for(v3_loop in 1:3){
33         par(new=T)
34         plot_data<-data[intersect(which(data[v2]==v2_range[v2_loop]),
35                                     which(data[v3]==v3_range[v3_loop])),]
36                                     # 描画するデータを抽出
37         plot(plot_data[,c(variable, observed)],
38              xlim=x_lim, ylim=y_lim, ann=F, axes=F, log="x",
39              pch=v2_loop, col=cols[v3_loop])
40         lines(plot_data[,c(variable, observed)],
41               xlim=x_lim, ylim=y_lim, ann=F,
42               col=cols[v3_loop])
43       }
44     }
45     legend("topleft", bty = "n", inset = c(0, 0),
46            legend = c(paste(name[v2], " = ", v2_range[1], sep=""),
47                       paste(name[v2], " = ", v2_range[2], sep=""),
48                       paste(name[v2], " = ", v2_range[3], sep="")),
49            pch = 1:3)
50     legend("topleft", bty = "n", inset = c(0, 0.15),
51            legend = c(paste(name[v3], " = ", v3_range[1], sep=""),
52                       paste(name[v3], " = ", v3_range[2], sep=""),
53                       paste(name[v3], " = ", v3_range[3], sep="")),
54            lty = 1, col = cols)
55     dev.off()
56   }
57 }

```

1.5.2 プロットした結果

図??にプロットした結果を示す。プロットはすべて片対数軸で示した。なお、個々のプロットの画像のファイルは、レポートの tex ファイルと同じディレクトリ内にある。

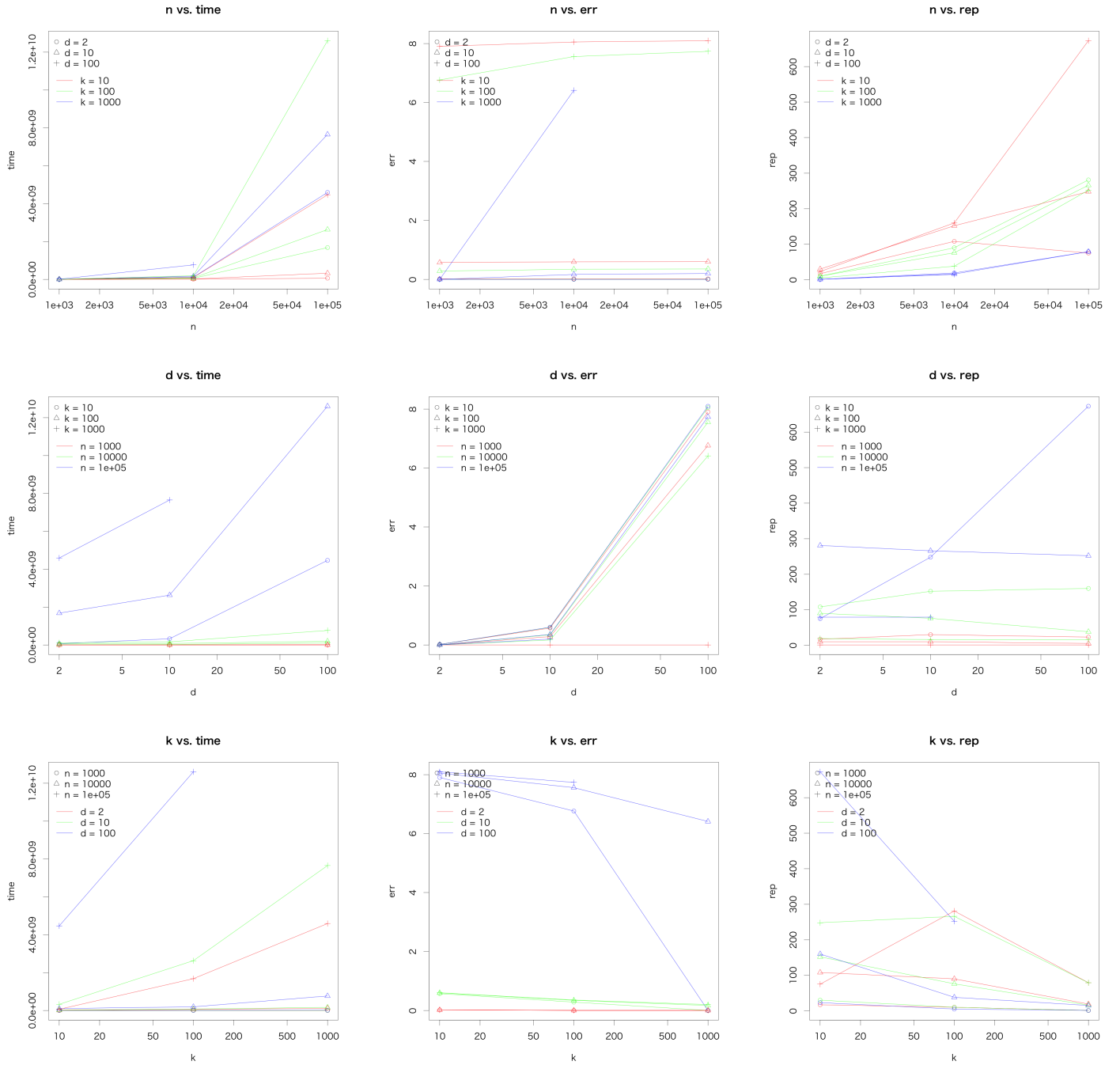


図1 変数 n , d , k を変化させたときのプログラム実行時間 (time), 平均二乗誤差 (err), 反復回数 (rep) の変化

1.6 考察

1.6.1 Lloyd の k-means 法の計算量

まず, Lloyd の k-means 法のアルゴリズムの繰り返しステップの計算量について考える。

再クラスタリングの手続きにおいては, 各データ点について, 最近接のクラスタを探すために $O(k)$ の時間がかかり, 全データ点について再クラスタリングを行うには $O(nk)$ 時間かかる。

代表点の再計算については, 代表点の重心を考えるためには, 各クラスタに属する全ての点の座標を見る必要があるため, 全体で $O(n + k)$ 時間かかる。

これより, 繰り返しステップ全体では, $O(nk + n + k) = O(nk)$ 時間かかる と推論できる。

アルゴリズム全体では, この繰り返しステップが何回か繰り返される。何度の繰り返しが必要であるかは, データの性質に依存するため, 一般的な解析は難しいが, 仮に繰り返し回数を r とおくと, 全体で $O(rnk)$ となると推測できる。

1.6.2 データ点の数 n の影響

まず、データ点数 n に対して、平均二乗誤差 (err) はほぼ一定の値をとった。これは、平均をとる操作をしているため、 n の大きさに平均二乗誤差は影響を受けないからであると解釈できるだろう。 $d = 100, k = 1000$ の振る舞いが一見不思議に見えるかもしれないが、これは $n = 1000$ のとき $k = 1000$ と、データ数と等しい数のクラスタに分類すると、誤差が 0 になるということを示しているだけである。

次に、データ点数 n に対する繰り返し回数 (rep) の変化であるが、データによってばらつきがあるものの、 n の増加に対してほぼ指数の増加をしているとみなしてよいのではなかろうか。

データ点の数 n に対してプログラムの実行時間は single exponential より早い速度で増加するようだ。これは、先のアルゴリズムの計算量の考察で $O(rnk)$ と求めたが、 r 自体が n の指数で増加するため、 $O(rnk)$ 全体では n の指数よりも速い速度で増加すると推論でき、プログラムの実行結果と符合する。

1.6.3 次元 d の影響（「次元の呪い効果の考察」）

まず、次元の呪いの効果について。次元 d と平均二乗誤差 (err) の関係のグラフを見ると、 d の増加にしたがって、平均二乗誤差は急速に増加していることがわかる。増加の速度は一重指数よりも速く、 $(k = 1000, n = 1000)$ を除く全てのサンプルで同様の結果が得られている。これは、高次元になると、点同士の間隔が互いに粗になり、クラスタ内での分散が大きくなってしまうというふうに解釈できる。まさに「次元の呪い」効果の一例であろう。 $(k = 1000, n = 1000)$ のサンプルにて次元の呪い効果が見られなかった理由は、各点を個々のクラスタとみなした場合に平均二乗誤差が 0 となるからである。

次に、次元の大きさと繰り返し回数 (rep) について。これは、 $(k = 10, n = 100000)$ のサンプルを除いてほぼ一定とみてよいだろう。 $(k = 10, n = 100000)$ のケースだけ例外的であった理由は、次の通りである。このパラメタの組合せの場合、解としてありえるクラスタリングの数が他のパラメタよりも多くなる。このため、膨大な解空間の中で、現在の解の近傍を渡り歩きながら最適解を探索するために必要な繰り返し回数が多くなったと解釈できるだろう。

最後に次元の大きさとプログラムの実行時間について。次元の大きさによらずほぼ一定の結果と考えることができるが、 n が大きい場合には、プログラムの実行時間が長くなるという傾向が見られた。これについては、先に n の効果のところで見たとおりである。

1.6.4 クラスタ数 k の影響

クラスタ数 k を増やすことにより、平均二乗誤差は小さくなる。また、クラスタ数を大きくしたほうが、繰り返し回数が小さくなる傾向にあるのは、解空間が小さくなることによるものであろう。最後に、クラスタ数とプログラムの実行時間であるが、 k の増加にたいして指数的に実行時間が増加している。繰り返し回数 r の減少も加味して、プログラム実行時間の k に対する指数増加の説明を考えようとしたが、これについてはあまり良くわからなかった。

2 高速化の実装

プログラムの本体は、Hamerly.cpp 内に実装した。

ソースコード 8 Hamerly.cpp (Hamerly の高速化 K-means アルゴリズム)

```
1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 #include <vector>
5 #include <cstdlib>
6 #include <time.h>
7 #include <sys/time.h>
8 #include <sys/types.h>
9 #include <unistd.h>
10
11 #include "my_vector.hpp"
12 #include "time_bench.hpp"
13
14 using namespace std;
```

```

15
16 template <class T>
17 inline double dist(vector<T> x, vector<T> y){
18     return norm(x - y); /* ベクトルのノルムから距離を定義 */
19 }
20
21 void Hamerly_update_ul(const int n, const int k,
22     const vector<vector<double>> data,
23     const vector<vector<double>> c, /* クラスタの重心 */
24     const vector<int> a, /* 各点  $x_i$  が属するクラスタ */
25     vector<double> &u, /* upper bound */
26     vector<double> &l /* lower bound */){
27     for(int i = 0; i < n; ++i){
28         u.at(i) = dist(data.at(i), c.at(a.at(i)));
29         double distance = dist(data.at(i), c.at(0));
30         double min = distance;
31         for(int j = 1; j < k; ++j){
32             distance = dist(data.at(i), c.at(j));
33             if(distance < min){ min = distance; }
34         }
35         l.at(i) = min;
36     }
37     return;
38 }
39
40 bool Hamerly_repeat(const int n, const int d, const int k,
41     const vector<vector<double>> data,
42     vector<int> &q, /* クラスタ中の点の数 */
43     vector<vector<double>> &c, /* クラスタの重心 */
44     vector<vector<double>> &c_sum, /* クラスタのベクトル和 */
45     vector<double> &s, /* 最近接クラスタの重心との距離 */
46     vector<int> &a, /* 各点  $x_i$  が属するクラスタ */
47     vector<double> &u, /* upper bound */
48     vector<double> &l /* lower bound */){
49     /* Hamerlyの繰り返しステップ クラスタ重心の更新の有無を booleanで返す */
50     bool updated = false;
51
52     /* まず  $s(j)$  を更新
53      *  $s[j] = \min_{\{jj \neq j\}} \text{dist}(c[jj], c[j])$  */
54     for(int j = 0; j < k; ++j){
55         double min = dist(c.at(j), c.at((j == 0) ? 1 : 0));
56         for(int jj = ((j == 0) ? 1 : 0) + 1; jj < k; ++jj){
57             if(jj != j){
58                 double distance = dist(c.at(j), c.at(jj));
59                 if(distance < min){ min = distance; }
60             }
61         }
62         s.at(j) = min;
63     }
64
65     for(int i = 0; i < n; ++i){
66         double m = max(s.at(a.at(i)) / 2.0, l.at(i));
67         if(u.at(i) > m){ /* Hamerlyの命題の条件を満たさない */
68             u.at(i) = dist(data.at(i), c.at(a.at(i))); /* upper boundを更新 */
69             if(u.at(i) > m){ /* Hamerlyの命題の条件を満たさない */
70                 int aa = a.at(i); /* 最近接クラスタが変化したか調べる */
71                 { /* 最近接クラスタを再計算 */
72                     int argmin_j = 0;

```

```

73     double min = dist(data.at(i), c.at(0));
74     for(int j = 1; j < k; ++j){
75         if(dist(data.at(i), c.at(j)) < min){ argmin_j = j; }
76     }
77     a.at(i) = argmin_j;
78 }
79 if(aa != a.at(i)){ /* 最近接クラスタの変化によるパラメタの更新 */
80     updated = true; q.at(aa)--; c_sum.at(aa) -= data.at(i);
81     q.at(a.at(i))++; c_sum.at(a.at(i)) += data.at(i);
82     Hamerly_update_ul(n, k, data, c, a, u, l);
83 }
84     }
85 }
86 }
87
88 if(updated){ /* いずれかのデータ点の最近接クラスタが変化した場合 */
89     /* 各クラスタに関するパラメタの更新 */
90     vector<double> p(k, 0); /* 各クラスタの重心の移動距離 */
91     double p_max = -1; /* 番兵で初期化 */
92     for(int j = 0; j < k; ++j){
93         vector<double> c_prev = c.at(j);
94         c.at(j) = (1.0 / q.at(j)) * c_sum.at(j);
95         p.at(j) = dist(c_prev, c.at(j));
96         if(p_max < p.at(j)){ p_max = p.at(j); }
97     }
98     for(int i = 0; i < n; ++i){
99         u.at(i) += p.at(a.at(i));
100         l.at(i) -= p_max;
101     }
102 }
103
104 return updated; /* データ点の最近接クラスタに変化があったかを返す */
105 }
106
107
108 void Hamerly_init(const int n, const int d, const int k,
109     const vector<vector<double>> &data,
110     vector<int> &q, /* クラスタ中の点の数 */
111     vector<vector<double>> &c, /* クラスタの重心 */
112     vector<vector<double>> &c_sum, /* クラスタのベクトル和 */
113     vector<int> &a, /* 各点  $x_i$  が属するクラスタ */
114     vector<double> &u, /* upper bound */
115     vector<double> &l /* lower bound */){
116     /* 各クラスタの代表点をランダムに選択 */
117     for(int j = 0; j < k; ++j){
118         vector<double> zero(d, 0);
119         q.at(j) = 0; c.at(j) = zero; c_sum.at(j) = zero;
120     }
121     for(int i = 0; i < n; ++i){
122         int argmin_j = 0;
123         { /*  $argmin_j \text{ dist}(x(i), c(j))$  の計算 */
124             double distance = dist(data.at(i), c.at(0));
125             double min = distance;
126             for(int j = 1; j < k; ++j){
127                 distance = dist(data.at(i), c.at(j));
128                 if(distance < min){
129                     min = distance; argmin_j = j;
130                 }

```

```

131     }
132 }
133 a.at(i) = argmin_j;
134 q.at(argmin_j)++;
135 c_sum.at(argmin_j) += data.at(i);
136 }
137 /* upper bound, lower bound の更新 */
138 Hamerly_update_ul(n, k, data, c, a, u, l);
139 return;
140 }
141
142 double Hamerly_sqerr(const int n, const int k,
143     const vector<vector<double> > data,
144     vector<vector<double> > &c, /* クラスタの重心 */
145     vector<int> &a /* 各点  $x_i$  が属するクラスタ */){
146 /* クラスタリング結果に基づき二乗平均二乗誤差を計算して返す */
147 double sum = 0;
148 for(int i = 0; i < n; ++i){
149     sum += dist(data.at(i), c.at(a.at(i))) * dist(data.at(i), c.at(a.at(i)));
150 }
151 return (sum / n);
152 }
153
154
155 void Hamerly(const int n, const int d, const int k,
156     const vector<vector<double> > data){
157 if(n < k){
158     cerr << "data size n is smaller than cluster size d" << endl;
159     exit(1);
160 }else{
161     struct timeval t_start, t_end;
162     gettimeofday(&t_start, NULL); /* 時間計測開始 */
163
164     /* j番目のクラスタを主語とする記号 */
165     vector<int> q(k); /* クラスタ中の点の数 */
166     vector<vector<double> > c(k); /* クラスタの重心 */
167     vector<vector<double> > c_sum(k); /* クラスタ中のベクトルの総和 */
168     vector<double> s(k, 0); /* 最近接クラスタの重心との距離 */
169
170     /* i番目のデータ点を主語とする記号 */
171     vector<int> a(n); /* 各点  $x_i$  が属するクラスタ */
172     vector<double> u(n); /* upper bound */
173     vector<double> l(n); /* lower bound */
174
175     Hamerly_init(n, d, k, data, q, c, c_sum, a, u, l);
176
177     int t = 0;
178     while(Hamerly_repeat(n, d, k, data, q, c, c_sum, s, a, u, l)){ t++; }
179
180     //Hamerly_repeat(n, d, k, data, q, c, c_sum, s, a, u, l);
181     //Hamerly_repeat(n, d, k, data, q, c, c_sum, s, a, u, l);
182
183     gettimeofday(&t_end, NULL); /* 時間計測終了 */
184
185     cout << n << "\t" /* サンプル数 */
186     << d << "\t" /* 次元 */
187     << k << "\t" /* クラスタ数 */
188     << diff_timeval(t_start, t_end) << "\t" /* 実行時間(us) */

```

```

189     << Hamerly_sqerr(n, k, data, c, a) << "\t" /* 平均二乗誤差 */
190     << t << endl;                               /* 繰り返し回数 */
191     return;
192 }
193 }
194
195 int main(int argc, char *argv[]){
196     if(argc < 5){
197         cerr << "usage: " << argv[0]
198         << " <n: data num> <d: dimension> <k: cluster num> <data file>" << endl;
199         exit(1);
200     }else{
201         int n = atoi(argv[1]), d = atoi(argv[2]), k = atoi(argv[3]);
202         char *file = argv[4];
203         vector<vector <double> > data(n);
204         {
205             ifstream fs(file);
206             if(fs.fail()){ exit(1); }
207             for(int i = 0; i < n; ++i){
208                 data.at(i).resize(d, 0);
209                 for(int j = 0; j < d; ++j){
210                     fs >> data.at(i).at(j);
211                 }
212                 fs.close();
213             }
214             Hamerly(n, d, k, data);
215             return 0;
216         }
217     }
218 }

```

3 生物データでの比較

これらについては、まだ試していません。