1　実験目的・課題

- 単純な二次元データの分類
- 動物園の動物データ (多次元データ) の分類
- 食べ物の画像データの分類

を行う。

2　実装方法

2.1　クラスタリングの手法について

データのクラスタリングをする手法には以下のようなものがある。

- 最短距離法
- 最長距離法
- 重心法
- 群平均法
- K-means 法

最短距離法とは 2 つのクラスター同士について、最も距離が小さいものから順にグループにしていくものである。計算量が少ないが、ある 1 つのクラスタにひとつづつ吸収されてしまう鎖効果が起こってしまうことがある。

最長距離法はクラスター間の距離をそのクラスター間の中で最も遠いデータとし、その距離の小さいものからクラスターにしていく手法である。分類感度は高いがクラスター同士が離れてしまう拡散現象が起きることがある。

重心法はクラスターに属するデータの重心を決定し、重心間の距離の小さい順にクラスターを形成していく手法である。計算量は多いが分類感度が良い。

群平均法は 2 つのクラスターに含まれるすべてのデータ間の距離を計算し、その平均の値をクラスター間の距離としてクラスタリングをする手法である。鎖効果や拡散現象をおこしにくい。

K-means 法とはデータ数を n、クラスタ数を k として、以下のような手順で表されるアルゴリズムである。

1. 各データ $d_i(i = 1, \ldots, n)$ に対してランダムにクラスタを割り振る
2. 各クラスタの中心 $m_j(j = 1, \ldots, k)$ を求める
3. 各 d_i と各 m_i との距離を求め d_i を最も近いクラスタに割り当てなおす
4. クラスタの割り当てが変化しなかった場合、終了する。
   そうでない場合手順 2 に戻る

結果は、最初に割り振られたクラスタに依存し、1 回の結果で最良のものが得られるとは限らない。

2.2 二次元データの分類

二次元データ間の距離をユークリッド距離 $d = \sqrt{x^2 + y^2}$ で定義する。これを最短距離法で分類することを考える。最短距離法では単に短い距離のものから順にクラスタにまとめていけばよい。これはクラスカル法で最小全域木を構築するのと同様の手順で行うことができる。

クラスカル法とはグラフ理論において重み付き連結グラフの最小全域木を求めることが出来る以下のようなアルゴリズムのことである。

1. $cost(e_0) \leq cost(e_1) \leq \ldots \leq cost(e_m)$ が成り立つように辺を並び替える
2. $T := (V(G), \emptyset)$ とする
3. i=1..m について、$E(T) \cup \{e_i\}$ が閉路を含まないならば $E(T) := E(T) \cup \{e_i\}$ とする

二次元データを頂点、各点間の距離を重みとした辺からなるグラフを考える。最初は N 個の木 (森) が存在し、このアルゴリズムが終了したときは 1 つの木だけが残っている。この手順の途中で木の数が k になったときにアルゴリズムを終了すると、k 個のクラスタに分類することが出来ている。これを実装したものをソースコード 1 に示す。

ソースコード 1　Rust による最短距離法の実装

```
fn main() {
  let mut data: Vec<Point> = Vec::new();
  if let Err(err) = read(&mut data) {
    println!("error running example: {}", err);
    process::exit(1);
  }
  let n: usize = data.len();
  let mut edges: Vec<(f64, usize, usize)> = Vec::new();
  for i in 0..n {
    for j in i..n {
      edges.push((calc_dist(&data[i], &data[j]), i, j));
    }
  }
  edges.sort_by(|a, b| a.partial_cmp(b).unwrap());
  let m = 2;
  let mut uni = Unionfind::new(n);
  for (_, u, v) in edges {
    if uni.tree_count == m {
      break;
    }
    uni.unite(u, v);
  }
  let r_st = uni.root_set();
  let mut clusters: Vec<Vec<usize>> = Vec::new();
  for r in r_st {
    clusters.push(uni.group(r));
```

```
    }
    let colors = ["red", "blue", "green", "black"];
    let mut fg = Figure::new();
    {
      let x = fg.axes2d();
      for i in 0..m {
        let mut dat_x: Vec<f64> = Vec::new();
        let mut dat_y: Vec<f64> = Vec::new();
        for &j in &clusters[i] {
          dat_x.push(data[j].x);
          dat_y.push(data[j].y);
        }
        x.points(&dat_x, &dat_y, &[Color(colors[i % 4])]);
      }
    }
    fg.set_terminal("png", "plot.png");
    let _ = fg.show();
  }
```

2.3 多次元データの分類

二次元データの分類の時と同様にクラスカル法を実行することで最小距離法での分類を行う。このとき n 次元データ a と b の間の距離は $d_{ab} = \sqrt{\sum_{i=1}^{n} w_i(a_i - b_i)^2}$ として計算した。ここで w_i は各系列に対する重みである。

2.4 画像データの分類

画像データを次元の小さい多次元データに圧縮することで、前節と同様に分類をすることができるようになる。

画像のデータは Python の OpenCV では RGB を表す 3 要素からなる配列の二次元配列である。この RGB の値は $256^3 = 16777216$ 通り存在する。各色についてその色が画像にいくつ含まれているかをカウントした 16777216 次元データを比べることで画像間の距離とすることでクラスタリングに必要な情報が得られる。しかし、16777216 個のデータを比較するのはデータの量が多すぎるため困難である。ここで、各画素の値を 64 で割り、0 から 255 の値を 0 から 3 までの値に変換する。こうすることで各画素を $4^3 = 64$ 通りの色に圧縮することが出来る。

画像を 64 次元データに圧縮することができたので最短距離法によって画像の分類を実行する。

画像の分類結果を確認するために、Python の subprocess パッケージの run 関数を使い、分類のたびに新たにディレクトリを作成し、そこへ元画像のファイルをコピーした。

3 結果と考察

3.1 二次元データの分類

3.2 多次元データの分類

3.3 画像データの分類

参考文献

[1] クラスター分析の手法（階層クラスター分析）
https://www.albert2005.co.jp/knowledge/data_mining/cluster/hierarchical_clustering

[2] k-means 法を理解する
https://qiita.com/g-k/items/0d5d22a12a4507ecbf11