K-means アルゴリズムの細見

司馬博文

2024-02-16

目次

課題

データ `mixture1.dat` と `mixture2.dat` に対して，次を実行して結果をまとめる：

1. Hard K-means と Soft K-means を $\beta \geq 0$ を変えて実装し，結果を比較する．
2. 正解率を計算する．
3. 最適と思われる初期値と K の値を議論する．

1　用いたデータとコード

1.1　用いたデータのプロット

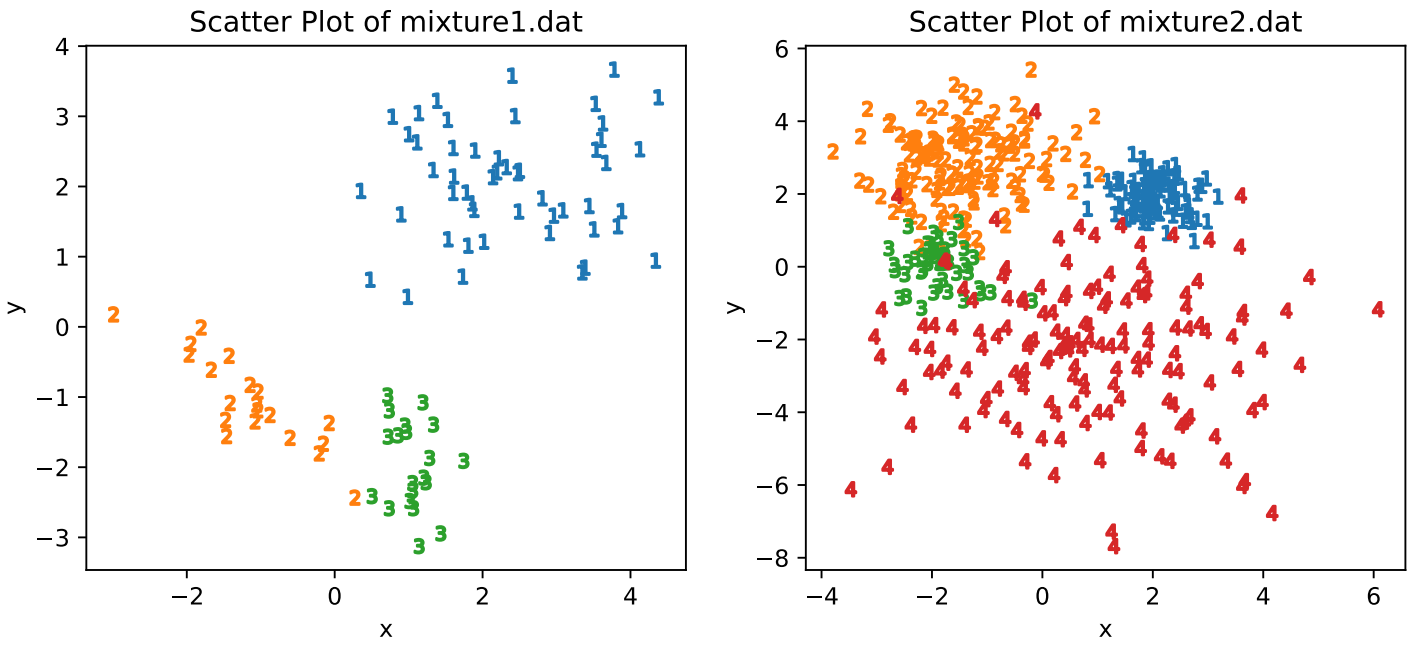


図 1: mixture1.dat（左）と mixture2.dat（右）のプロット

1.2　K-means アルゴリズムの実装

機能は次の通りである：

- クラスター平均の変化を追うために，Python の **クラスとして実装**した．
- K-means アルゴリズムは assignment と update との 2 つのステップからなり，これを 2 つのメソッド `soft_assignment` / `hard_assignment` と `update` として実装した．
- ソフトとハードの場合のアルゴリズムの違いは，（β を用いるかどうかと）負担率の計算のみであることを強調した実装になっており，実際，`run_soft()` と `run_hard()` ではその内容も `hard` と `soft` の 1 単語しか変わらない．
- `fetch_cluster` でアルゴリズムの最終的な回答を出力するが，`.r` フィールドから具体的な負担率の値を取得することができ，特にソフト K-means の場合に有用になりえる．
- `fetch_history` で **クラスター中心の履歴を取得できる**．

```
class kmeans_2d:
    """
    2次元データに対するソフト K-平均法の実装．

    Usage:
        kmeans = kmeans_2d(data, K, init, beta)
        kmeans.run()

    Parameters:
    - data: (N,2)-numpy.ndarray
    - K: int クラスター数
```

```
    - init: (2,K)-numpy.ndarray 初期値
    - beta: float 硬度パラメータ
    """

    def __init__(self, data, K, init, beta, max_iter=100):
        self.data = np.array(data, dtype=float)
        self.K = K
        self.init = np.array(init, dtype=float)
        self.beta = float(beta)
        self.max_iter = max_iter
        self.N = data.shape[0]  # データ数
        self.I = data.shape[1]  # 次元数 今回は２
        self.m = init  # クラスター中心の初期化．2×K行列．
        self.r = np.zeros((K, self.N), dtype=float)  # 負担率．K×N行列．
        self.history = [init.copy()] # クラスター中心の履歴．2×K行列．

    def soft_assigment(self):
        """soft K-means の場合の負担率の更新"""
        for i in range(self.N):
            distances = np.array([d(self.data[i], self.m[:,j]) for j in range(self.K)])
            denominator_ = np.sum(np.exp(-self.beta * distances))  # 分母
            self.r[:,i] = np.exp(- self.beta * distances) / denominator_

    def hard_assigment(self):
        """hard K-means の場合の負担率の更新"""
        for i in range(self.N):
            distances = np.array([d(self.data[i], self.m[:,j]) for j in range(self.K)])
            k_hat = np.argmin(distances)  # 最小距離のクラスター番号
            self.r[:,i] = 0  # 前のループの結果をリセット
            self.r[k_hat,i] = 1

    def update(self):
        """クラスター中心の更新"""
        new_m = np.zeros_like(self.m, dtype=float) # ここで float にしないと，クラスター中心が整数に限られてしま
        numerator = np.dot(self.r, self.data)  # (K,2)-numpy.ndarray
        denominator = np.sum(self.r, axis=1)  # 各クラスターの負担率の和
        for k in range(self.K):
            if denominator[k] > 0:
                new_m[:,k] = numerator[k] / denominator[k]
            else:
                new_m[:,k] = self.m[:,k]
        self.m = new_m

    def fetch_cluster(self):
        """最終的なクラスター番号を格納した (N,)-array を返す"""
        return np.argmax(self.r, axis=0)
```

```
    def fetch_history(self):
        """クラスター中心の履歴を格納したリストを，３次元の np.array に変換して返す"""
        return np.stack(self.history, axis=0)

    def run_soft(self):
        """soft K-means アルゴリズムの実行"""
        for _ in range(self.max_iter):
            self.soft_assigment()
            self.update()
            self.history.append(self.m.copy())
            if np.allclose(self.history[-1], self.history[-2]):
                break

    def run_hard(self):
        """hard K-means アルゴリズムの実行"""
        for _ in range(self.max_iter):
            self.hard_assigment()
            self.update()
            self.history.append(self.m.copy())
            if np.allclose(self.history[-1], self.history[-2]):
                break
```

次のように用いた：

```
initial_points = np.array([[0,0],[2,4],[4,2]])

result = kmeans_2d(data1, K=3, initial_points.T, beta=1)
result.run_hard()

## クラスター番号を正解と比べる際は次を利用
pred = result.fetch_cluster()
ans = data1.iloc[:, 0]

## プロットする際は次を利用
history = result.fetch_history()
```

2　Hard K-means と正答率

2.1　用いたアルゴリズムの説明

前節 1.2 が実際に用いたアルゴリズムであるが，Python のクラスとして定義している関係上，実際のアルゴリズム部分の動きがわかりにくくなっているため，hard K-means の場合に限って１つの関数として完結した形で書き直し，詳細な説明を付した．適宜読み飛ばして欲しい．

soft K-means と同時に実装したため（第 3.1 節），負担率計算の部分を変えれば流用できるように配慮して設計してある．

```python
def hkmeans_2d(data, K, init, max_iter=100):
    """
    ２次元データに対するハード K-平均法の実装例．

    Parameters:
    - data: (N,2)-numpy.ndarray
    - K: int クラスター数
    - init: (2,K)-numpy.ndarray 初期値

    Returns:
    - clusters: (N,)-numpy.ndarray クラスター番号
    """

    N = data.shape[0]                                                    ①
    I = data.shape[1]                                                    ②
    m = init                                                             ③
    r = np.zeros((K, N), dtype=float)                                    ④

    for _ in range(max_iter):
        # Assignment Step
        for i in range(N):
            distances = np.array([d(data[i], m[:,k]) for k in range(K)])  ⑤
            k_hat = np.argmin(distances)                                 ⑥
            r[:,i] = 0                                                   ⑦
            r[k_hat,i] = 1

        # Update Step
        new_m = np.zeros_like(m, dtype=float)                            ⑧
        numerator = np.dot(r, data)  # (K,2)-numpy.ndarray               ⑨
        denominator = np.sum(r, axis=1)  # 各クラスターの負担率の和        ⑩
        for k in range(K):                                               ⑪
            if denominator[k] > 0:
                new_m[:,k] = numerator[k] / denominator[k]
            else:
                new_m[:,k] = m[:,k]

        if np.allclose(m, new_m):                                        ⑫
            break
        m = new_m

    return np.argmax(r, axis=0)                                          ⑬
```

① データ数を取得している．

② データの次元を取得している．今回はすべて２次元データを用いる．

③ クラスター中心に引数として受け取った初期値を代入．$2 \times K$-行列であることに注意．

④ 負担率を $K \times N$-行列として格納している．その理由は後ほど行列積を通じた計算を行うためである．`dtype=float` の理由は後述．

⑤ この `distances` 変数は `(K,)-numpy.ndarray` になる．すなわち，第 k 成分が，第 k クラスター中心との距離となっているようなベクトルである．ただし，`d` は Euclid 距離を計算する関数として定義済みとした．

⑥ 距離が最小となるクラスター番号 $\hat{k} := [\arg\min_{k\in[K]} d(m_k, x_i)]$ を，$i \in [N]$ 番目のデータについて求める．

⑦ $\hat{k}$ に基づいて負担率を更新するが，ループ内で前回の結果をリセットする必要があることに注意．

⑧ ここで `dtype=float` と指定しないと，初め引数 `init` が整数のみで構成されていた場合に，Python の自動型付機能が `int` 型だと判定し，クラスター中心 `m` の値が整数に限られてしまう．すると，アルゴリズムがすぐに手頃な格子点に収束してしまう．

⑨ `numpy` の行列積を計算する関数 `np.dot` を使用している．更新式

$$m^{(k)} \leftarrow \frac{\sum_{n=1}^{N} r_k^{(n)} x^{(n)}}{\sum_{n=1}^{N} r_k^{(n)}}$$

の分子を行列積と見たのである．

⑩ 分母 (denominator) は (K, N)-行列 `r` の行和として得られる．

⑪ ゼロによる除算が起こらないように場合わけをしている．

⑫ クラスター中心がもはや変わらない場合はアルゴリズムを終了する．

⑬ 負担率の最も大きいクラスター番号を返す．今回は `hat_k` の列をそのまま返せば良いが，soft K-means アルゴリズムにも通じる形で実装した．

2.2 hard K-means の初期値依存性の観察

`mixture1.dat` に次の 2 つの初期値を与えてみる．

$$m_1 := \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \quad m_2 := \begin{pmatrix} 1 \\ 4 \end{pmatrix}, \quad m_3 = \begin{pmatrix} -1 \\ 1 \end{pmatrix},$$

と，m_2, m_3 は変えずに m_1 の y-座標を 1 だけ下げたもの

$$m_1' := \begin{pmatrix} 4 \\ -1 \end{pmatrix}$$

とを初期値として与えてみる．

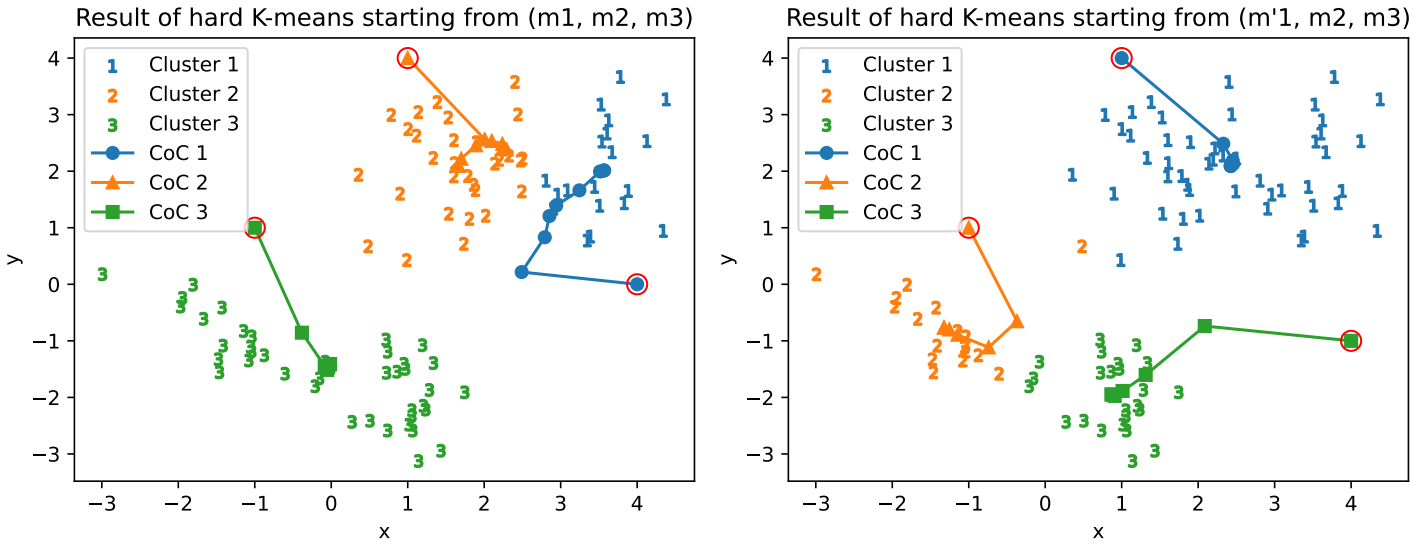


図 2: mixture1.dat のハード K-平均法によるクラスタリングの結果．赤丸で囲まれている点がクラスター中心 (CoC / Center of Cluster) の初期値で，その後の移動が図示されている．

正解数: 51 vs. 85　　**正解率**: 56.7 % vs. 94.4 %　　**反復数**: 9 回 vs. 7 回

結果が全く違い，(m_1', m_2, m_3) を与えた方が，大きく正解に近づいている．具体的には，右下の初期値 m_1 は右上の島に行くが，m_1' は左下の島に行ってくれる．

ハード K-平均アルゴリズムは初期値に敏感であることがよく分かる．

なお，(m_1', m_2, m_3) を与えた場合（上図の右側）を正解と並べてプロットしたものが次の図である．

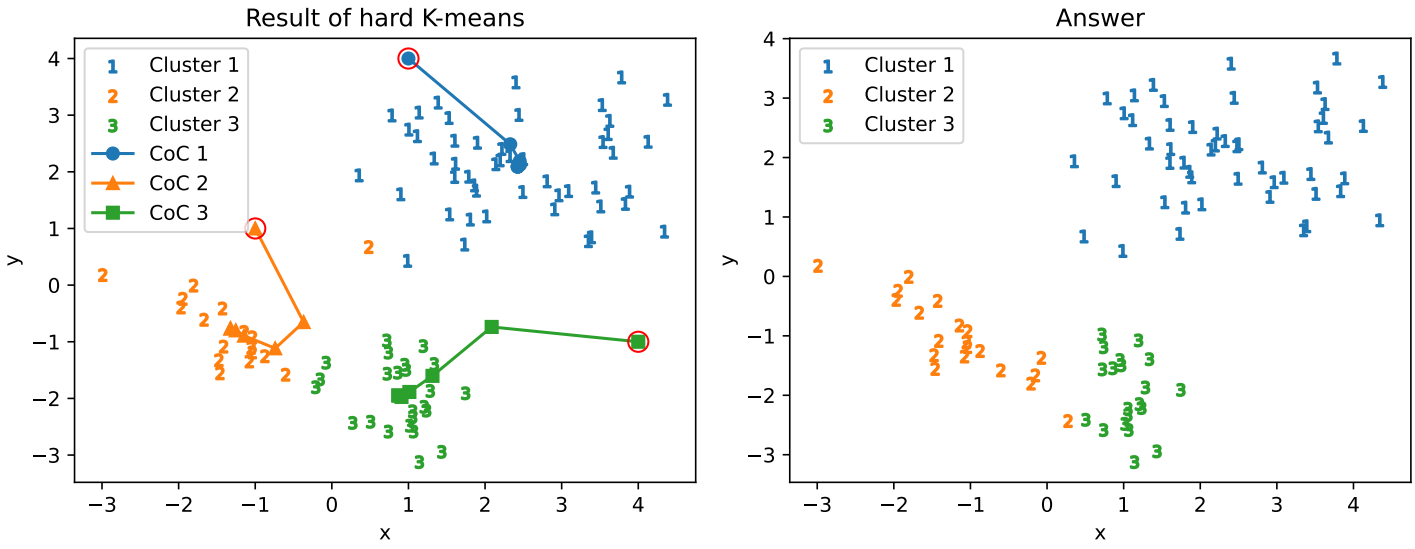


図 3: ハード K-平均法によるクラスタリングの結果（初期値は $(m_1', m_2, m_3) = \left(\begin{pmatrix}4\\-1\end{pmatrix}, \begin{pmatrix}1\\4\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}\right)$）．

正解数：85　　**正解率**：94.4 %　　**反復数**：7 **回**

２番のクラスターが３番のクラスターに侵食している４点と，１番のクラスターに侵食している１点で，計５点のミスがある．

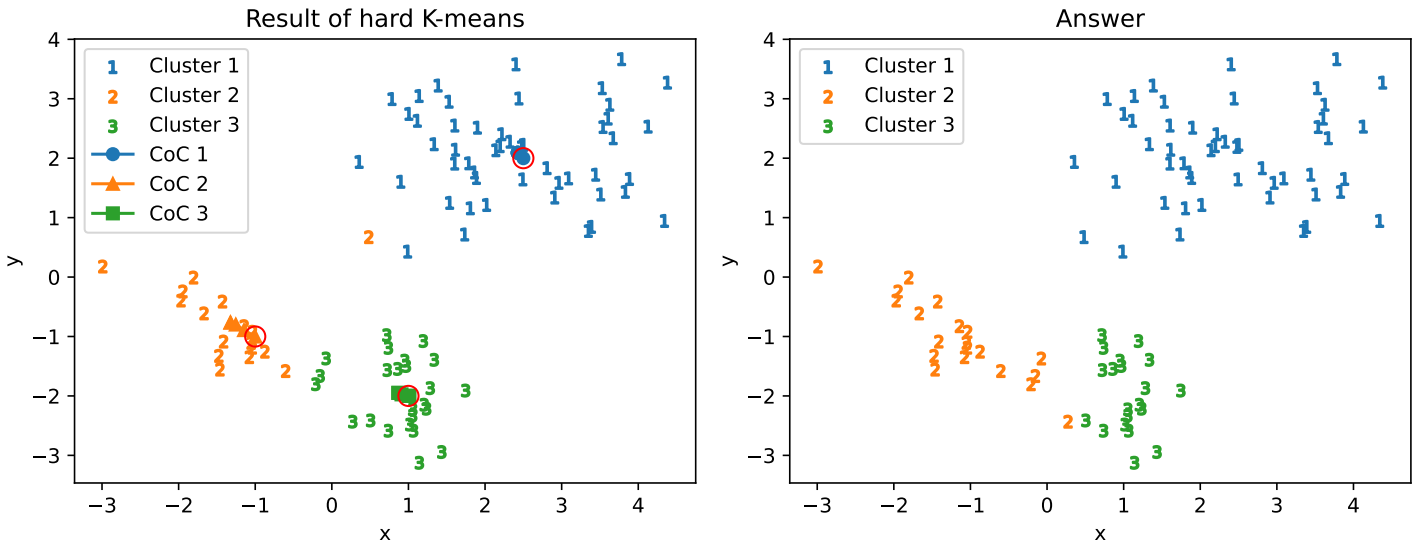
2.3　正解率を上げる試み

直前の結果ではクラスター２と３の境界線で４つのミスを犯しており，これを修正できないか試したい．

そこで，答えに近いように，

$$m_1 \leftarrow \begin{pmatrix}2.5\\2\end{pmatrix}, \quad m_2 \leftarrow \begin{pmatrix}-1\\-1\end{pmatrix}, \quad m_3 \leftarrow \begin{pmatrix}1\\-2\end{pmatrix},$$

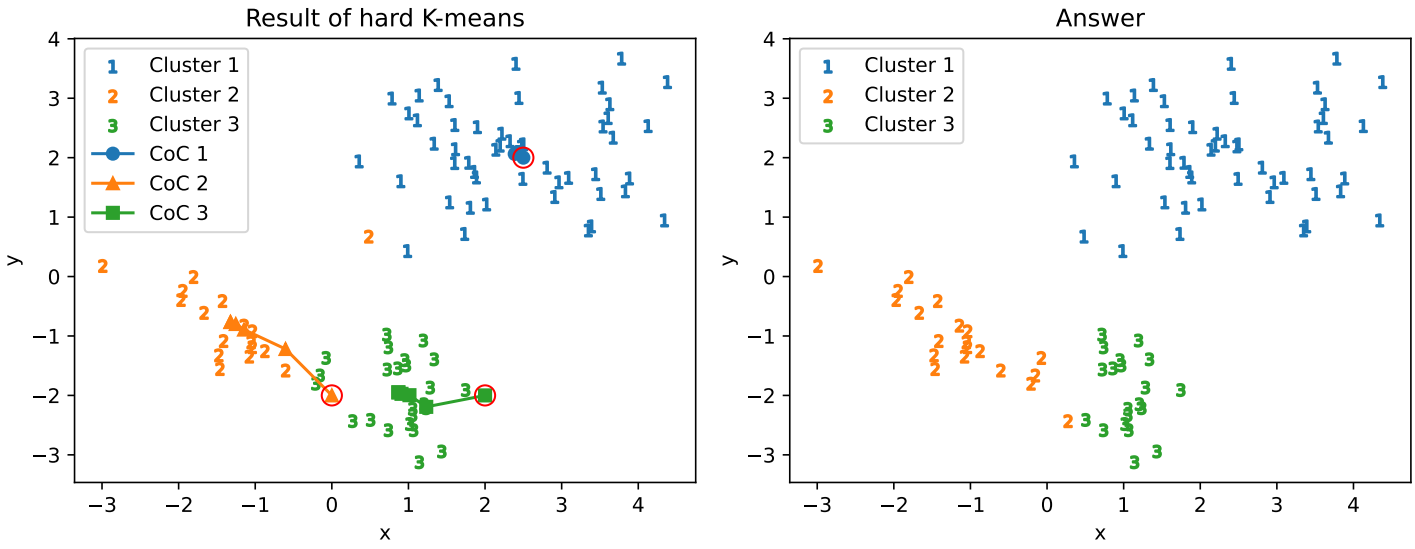
を初期値として与えてみて，正答率の変化を観察する．



正解数：85　　**正解率**：94.4 %　　**反復数**：5 回

もはや初期値から殆ど動いていないが，目標のクラスター３に分類された３つの点が，相変わらず３のままであり，加えてクラスター２の中心がこれらから逃げているようにも見えるので，クラスター２の初期値をよりクラスター３に近いように誘導し，クラスター３の中心をより右側から開始する：

$$m_2 : \begin{pmatrix} -1 \\ -1 \end{pmatrix} \mapsto \begin{pmatrix} 0 \\ -2 \end{pmatrix} \quad m_3 : \begin{pmatrix} 1 \\ -2 \end{pmatrix} \mapsto \begin{pmatrix} 2 \\ -2 \end{pmatrix}$$



正解数：85　　**正解率**：94.4 %　　**反復数**：6 回

こんなに誘導をしても，正しく分類してくれない．

実は，以上２つの初期値では，最終的に３つのクラスター中心は全く同じ値に収束している．

参考：最終的なクラスター中心の比較

今回の最終的なクラスター中心：初期値変更後

	Cluster1	Cluster2	Cluster3
x	2.426102	-1.323353	0.868333
y	2.091429	-0.765176	-1.948458

前回の最終的なクラスター中心：初期値変更前

	Cluster1	Cluster2	Cluster3
x	2.426102	-1.323353	0.868333
y	2.091429	-0.765176	-1.948458

よって，これ以上どのように初期値を変更しても，正答率は上がらないシナリオが考えられる．

以上の観察から，ハード K-平均法はある種の **局所解に収束する**ようなアルゴリズムであると考えられる．

3　Soft K-means と正答率

3.1　用いたアルゴリズムの説明

実際に用いたアルゴリズム（第 1.2 節）のうち，ソフト K-平均法に当たる部分のコードは，ハード K-平均法に該当する部分（第 2.1 節参照）と，次の部分のみ異なる：

```
for i in range(N):
        distances = np.array([d(data[i], m[:,k]) for k in range(K)])          ①
        denominator_ = np.sum(np.exp(-beta * distances))                      ②
        r[:,i] = np.exp(-beta * distances) / denominator_                     ③
```

① データ x_i とクラスター中心 $(m_k)_{k=1}^K$ との距離を計算し，ベクトル $(d(x_n, m_k))_{k=1}^K$ を `distances` に格納している．

② 負担率の計算

$$r_{ik} = \frac{\exp(-\beta d(m_k, x_i))}{\sum_{j=1}^K \exp(-\beta d(m_j, x_i))}$$

を 2 段階に分けて行なっており，分母を先に計算して変数 `denominator_` に格納している．

③ すでに計算してある分母 `denominator_` を用いてデータ x_i の負担率 $(r_{ki})_{k=1}^K$ を計算し，(K, N)-行列 `r` の各列に格納している．

3.2 正解率の観察

逆温度をひとまず $\beta = 1$ としてみる．図 2 の左と全く同様な初期値

$$m_1 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \quad m_2 = \begin{pmatrix} 1 \\ 4 \end{pmatrix}, \quad m_3 = \begin{pmatrix} -1 \\ 1 \end{pmatrix},$$

を与えてみると，次の通りの結果を得る：

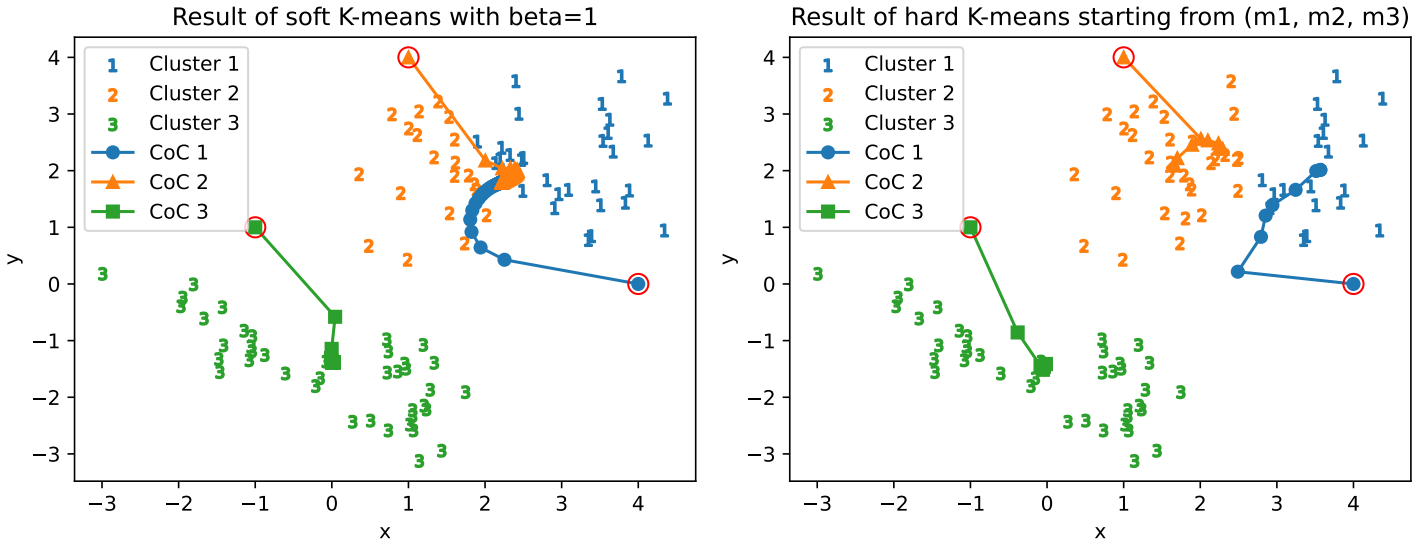


図 4: 左がソフト K-平均法（$\beta = 1$），右がハード K-平均法によるクラスタリングの結果（図 2 の左と全く同じもの）．赤丸で囲まれている点がクラスター中心 (CoC / Center of Cluster) の初期値で，その後の移動が図示されている．

正解数: 49 vs. 51　　**正解率**: 54.4 % vs. 56.7 %　　**反復数**: 65 回 vs. 9 回

正解率 43.3% であったところから少し改善している．さらに，反復数が 9 回であったところから，劇的に増えている（65 回）．また，左図中右上の 2 つのクラスター中心（青とオレンジ）の収束先は，微妙にずれているが**ほとんど一致している** 点も注目に値する．

参考：最終的なクラスター中心の座標

```
centers = history[-1, :, :]
df = pd.DataFrame(centers, columns=['Cluster1', 'Cluster2', 'Cluster3'])
print(df)
```

	Cluster1	Cluster2	Cluster3
x	2.217840	2.218129	0.015051
y	1.797156	1.797323	-1.385450

図 2 の右で与えた初期値 (m_1', m_2, m_3) も与えてみる．

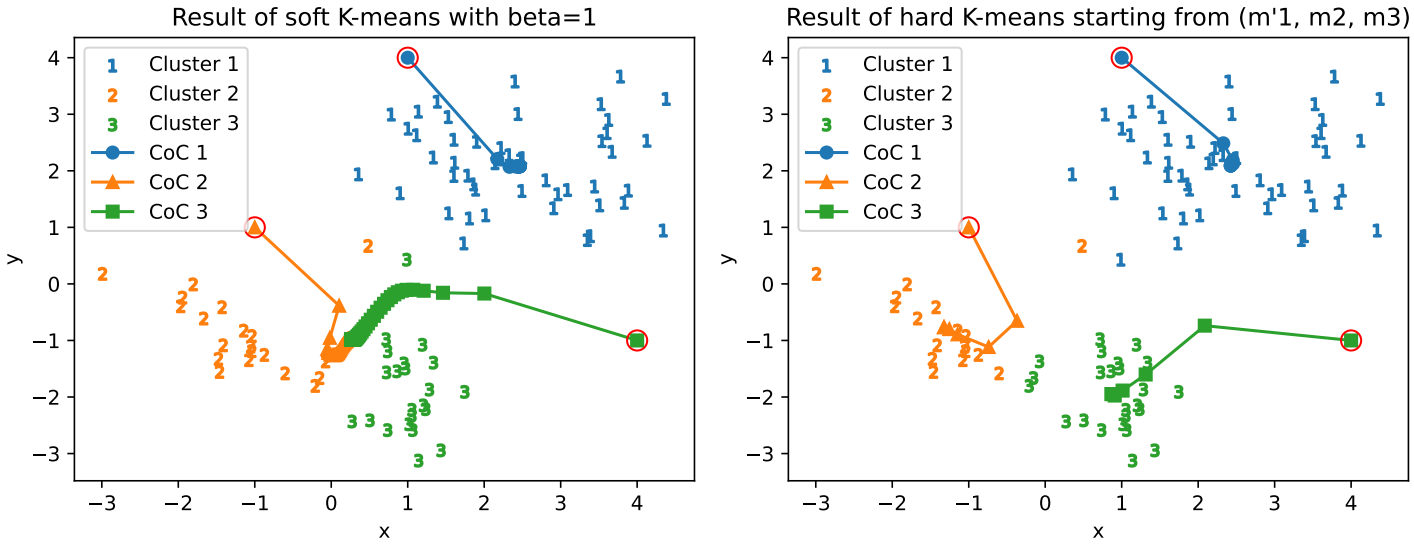


図 5: ソフト K-平均法（$\beta = 1$）によるクラスタリングの結果，右がハード K-平均法によるクラスタリングの結果（図 2 の右と全く同じもの）．

正解数: 87 vs. 85　　**正解率**: 96.7 % vs. 94.4 %　　**反復数**: 101 回 vs. 7 回

正答率は 94.4% からやはり少し改善しており，反復数が 7 回から大きく増えている．結果はやはり図 4 とは大きく異なっており，ハード K-平均法で観察された**初期値鋭敏性** が，変わらず残っている．加えてこの場合も図 4 のクラスター 1 と 2 と同様に，クラスター 2 と 3 の**中心がほぼ一致** している．

参考：最終的なクラスター中心の座標

	Cluster1	Cluster2	Cluster3
x	2.451958	0.257367	0.258702
y	2.080430	-0.984350	-0.984790

$\beta = 1$ の場合のソフト K-平均法は，この例では**クラスター中心が融合する傾向にある**ようである．一般に，β が小さく，温度が大きいほど，エネルギーランドスケープに極小点が少なくなり，クラスターは同じ場所へ収束しやすくなると予想される．

3.3　硬度パラメータに依る挙動の変化

初期値を直前で用いた

$$m_1 \leftarrow m_1' = \begin{pmatrix} 4 \\ -1 \end{pmatrix}, \quad m_2 \leftarrow \begin{pmatrix} 1 \\ 4 \end{pmatrix}, \quad m_3 \leftarrow \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \tag{1}$$

で固定とし，さらに温度を上げて，逆温度を $\beta = 0.1$ としてみる．

正解数: 68 vs. 87　　**正解率**: 75.6 % vs. 96.7 %　　**反復数**: 10 回 vs. 101 回

反復数は減少し，全てがほとんど同じクラスターに属する結果となってしまった．3 つのクラスター中心の座標が小数点以下 5 桁の精度で一致してしまっている．

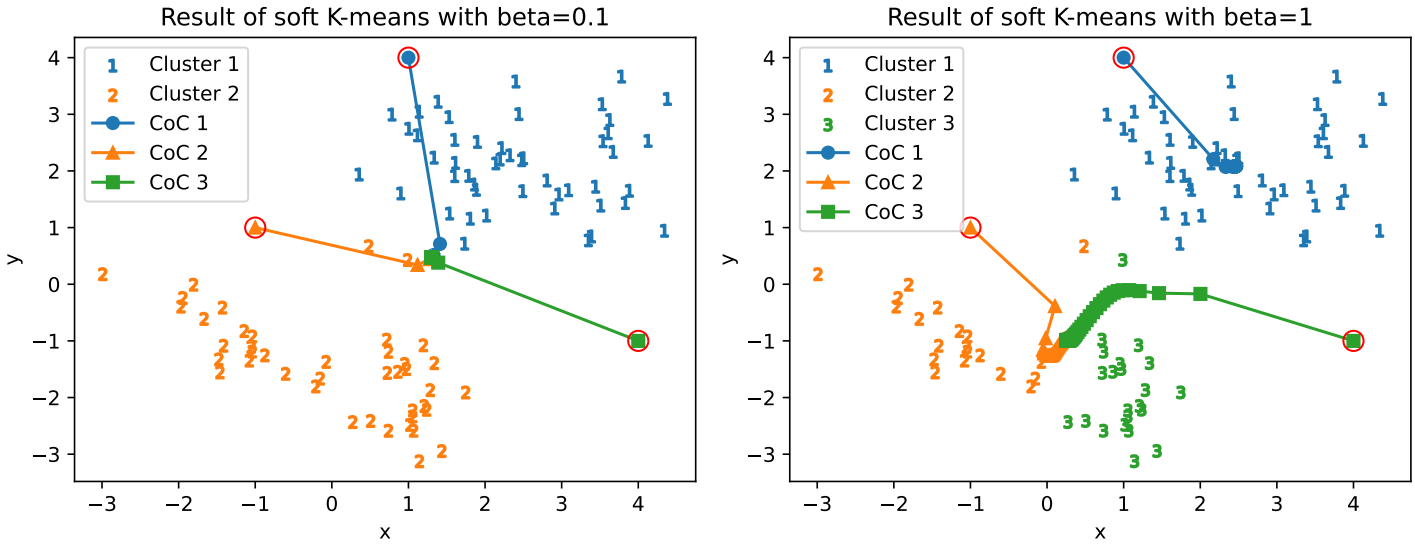


図 6: ソフト K-平均法（左 $\beta = 0.1$，右 $\beta = 1$）によるクラスタリングの結果.

参考：最終的なクラスター中心の座標

	Cluster1	Cluster2	Cluster3
x	1.302467	1.302466	1.302467
y	0.474545	0.474544	0.474544

温度が大変に高い状態では，全てが乱雑で，３つのクラスターが一様・公平に負担率を持つようになった．そのため，第一歩からほとんど全体の中心へと移動し，反復数が減る．加えて，関数 `argmax` が同じ値 k を返してしまっているのである．

次に，温度を少し下げて，逆温度を $\beta = 10$ としてみる．

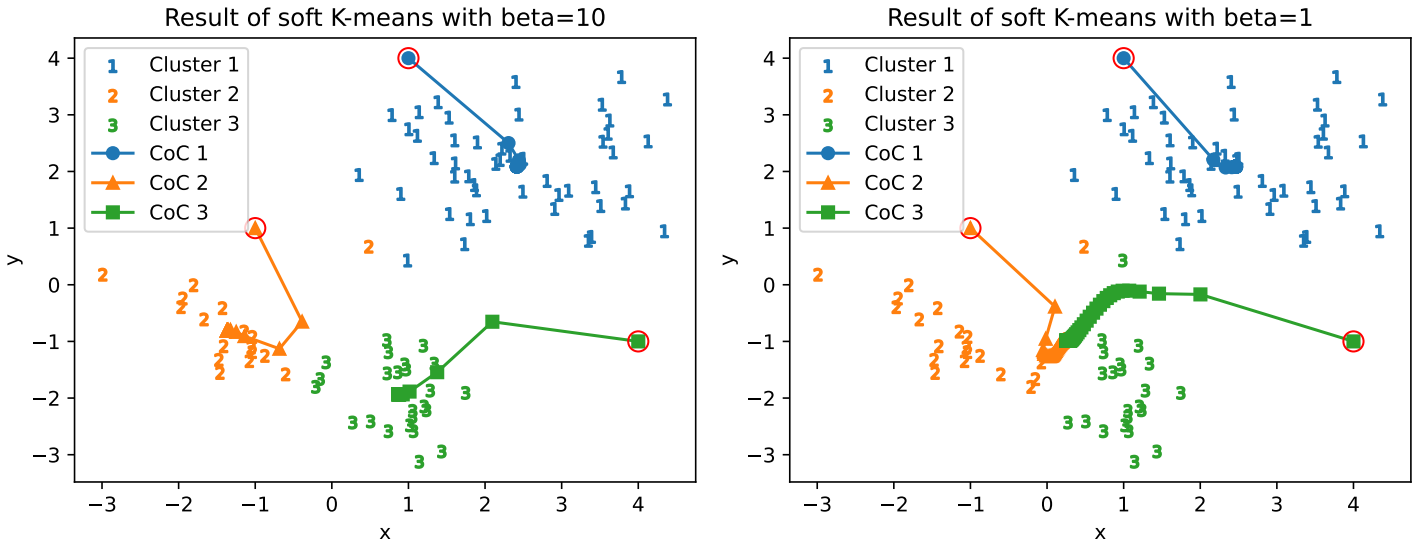


図 7: ソフト K-平均法（左 $\beta = 10$，右 $\beta = 1$）によるクラスタリングの結果.

正解数：85 vs. 87　　**正解率**：94.4 % vs. 96.7 %　　**反復数**：18 回 vs. 101 回

初めて soft K-means アルゴリズムを用いた場合で，３つのクラスター中心がはっきりと別れた．反復回数は，$\beta = 1$ の場合と比べればやはり落ち着いている．しかし，正解率はハード K-平均法の場合（図 2 など）と全く同じである．**実は，最終的なクラスター中心もハード K-平均法の場合 図 2 の最終的なクラスター中心とほとんど同じになっている．**

参考：最終的なクラスター中心の座標

今回のソフト K-平均法の最終的なクラスター中心

	Cluster1	Cluster2	Cluster3
x	2.413880	0.870308	-1.368158
y	2.084395	-1.931050	-0.806831

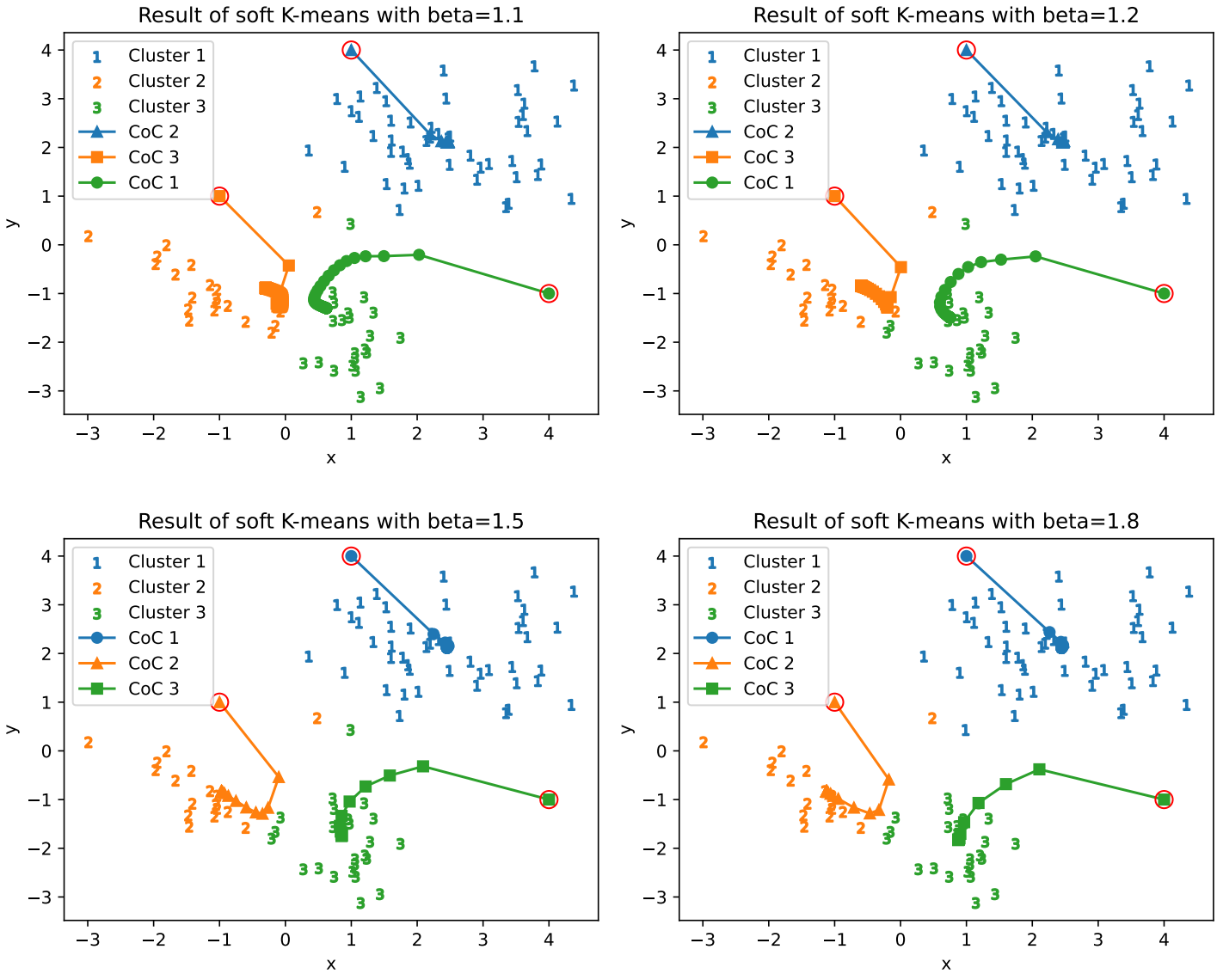
図 2 のハード K-平均法の最終的なクラスター中心

	Cluster1	Cluster2	Cluster3
x	2.426102	0.868333	-1.323353
y	2.091429	-1.948458	-0.765176

以上より，ソフト K-平均法は温度を上げるほどクラスター数が少なくなり，温度を下げるほどクラスター数は上がり，**十分に温度を下げるとハード K-平均法に挙動が似通う**．

3.4 最適な硬度の選択

$\beta = 1$ ではクラスターが 2 つに縮退し，$\beta = 10$ では hard K-means アルゴリズムの結果とほとんど変わらなくなった．そこで，この中間の値での挙動の変化を調べる．

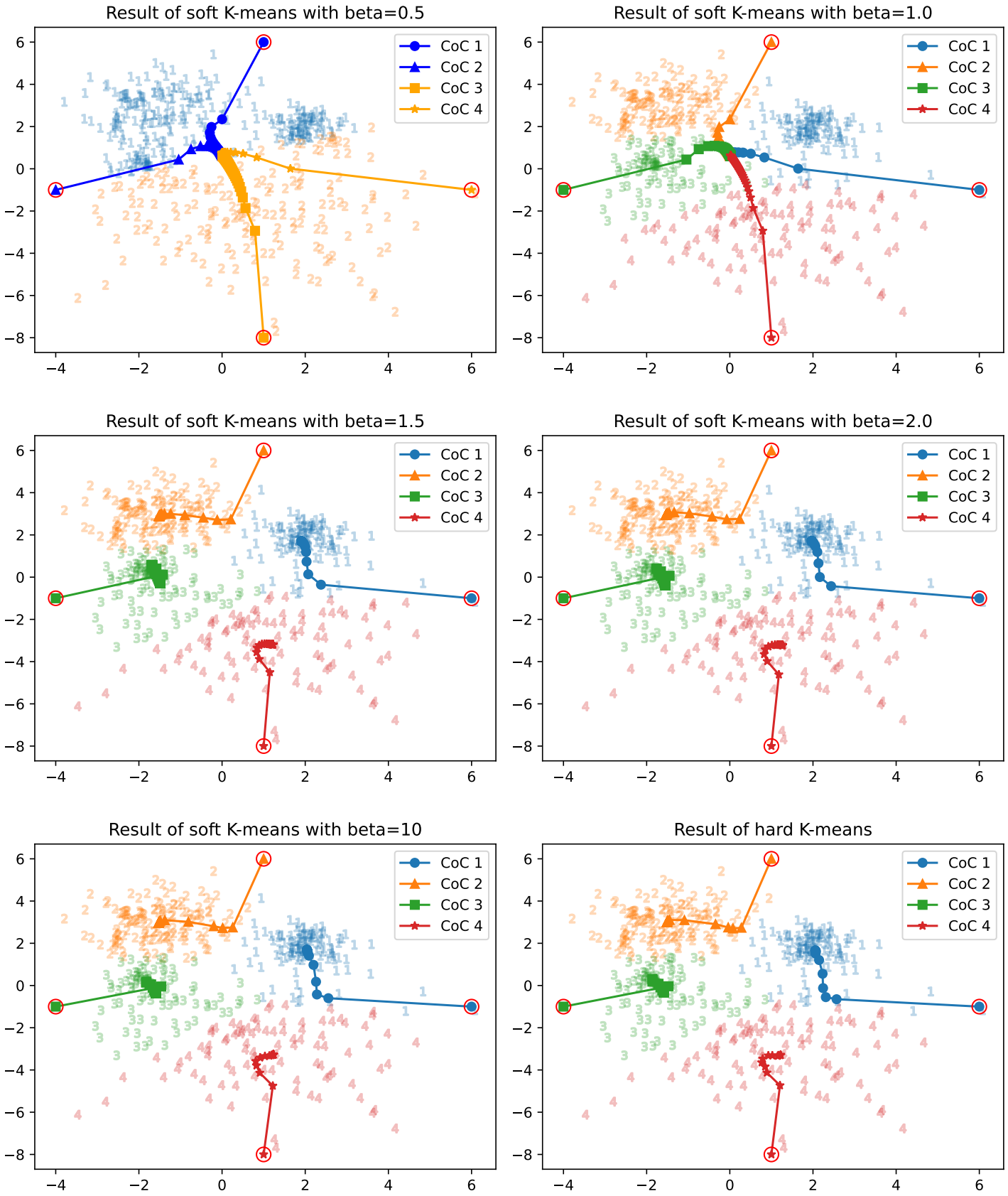


クラスター 2 と 3 の中心が，温度の低下と共に徐々に離れていくことが観察できる．やはり，温度が高い場合はクラスター中心が

合流・融合してしまいやすいが，冷却することでクラスター数は大きい状態で安定し，反復回数も減る．

正解数: 87 vs. 85　　**正解率**: 96.7 % vs. 94.4 %　　**反復数**: 82 回 vs. 38 回
正解数: 84 vs. 85　　**正解率**: 93.3 % vs. 94.4 %　　**反復数**: 19 回 vs. 17 回

4　mixture2.dat の場合での検証



正解数: 293 vs. 385　　**正解率**: 65.1 % vs. 85.6 %　　**反復数**: 101 回 vs. 59 回
正解数: 379 vs. 380　　**正解率**: 84.2 % vs. 84.4 %　　**反復数**: 47 回 vs. 40 回
正解数: 378 vs. 378　　**正解率**: 84.0 % vs. 84.0 %　　**反復数**: 29 回 vs. 14 回

5　まとめ

結論

- `mixture1.dat` に対して，（初期値 1 で）ソフト K-平均法を適用すると，
  - $\beta \geq 2$ の場合で結果はハード K-平均法と変わらなくなる．
  - $\beta = 1$ の場合で結果はクラスターがほとんど 2 つになり，$\beta \leq 0.5$ では計算機上では実際に 2 つになってしまう．
  - 正答率は $1 \leq \beta \leq 1.1$ で最大であった．
  - β を大きくするほど，反復回数は減少していった．
- `mixture2.dat` に対しても，以上の 4 点について同様の傾向が確認できた．