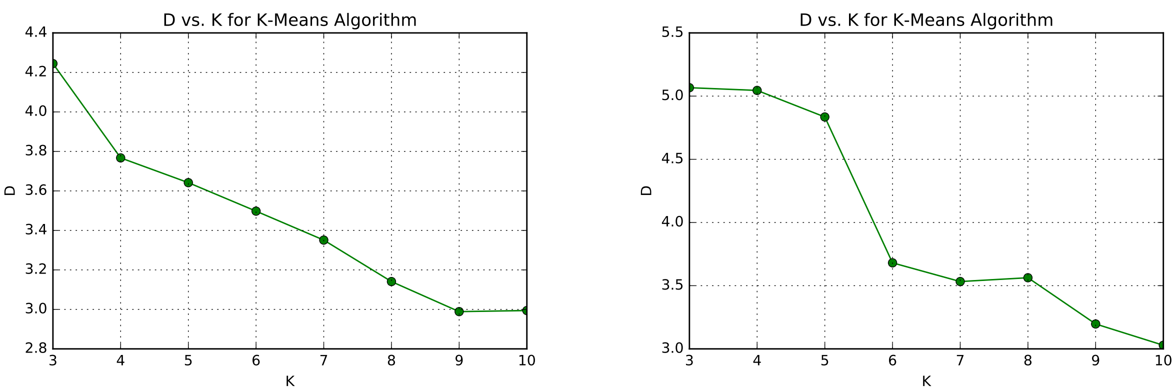
IE 529 Fall 2016 Computational Assignment 2

Jifu Zhao Date: 12/15/2016

1 Clustering Result Comparison

I. Lloyd's (K-means)Algorithm

1. In this part, with the Lloyd's algorithm for k-means clustering, we choose the best distortion D as the the objection function. The change of D versus cluster number K is shown in Fig. 1, where the result for clustering.txt is shown in Fig. 1a and the result for bigClusteringData.txt is show in Fig. 1b.



(a) clustering.txt (b) bigClusteringData.txt

Figure 1: Change of Distoration versus Cluster Number K for K-Means Algorithm

2. The scatter plot of the clustering result for clustering.txt is shown in Fig. 2 and the scatter plot of the clustering result for bigClusteringData.txt is shown in Fig. 3. The cluster centroids are clearly marked and different clusters are denoted by different colors.

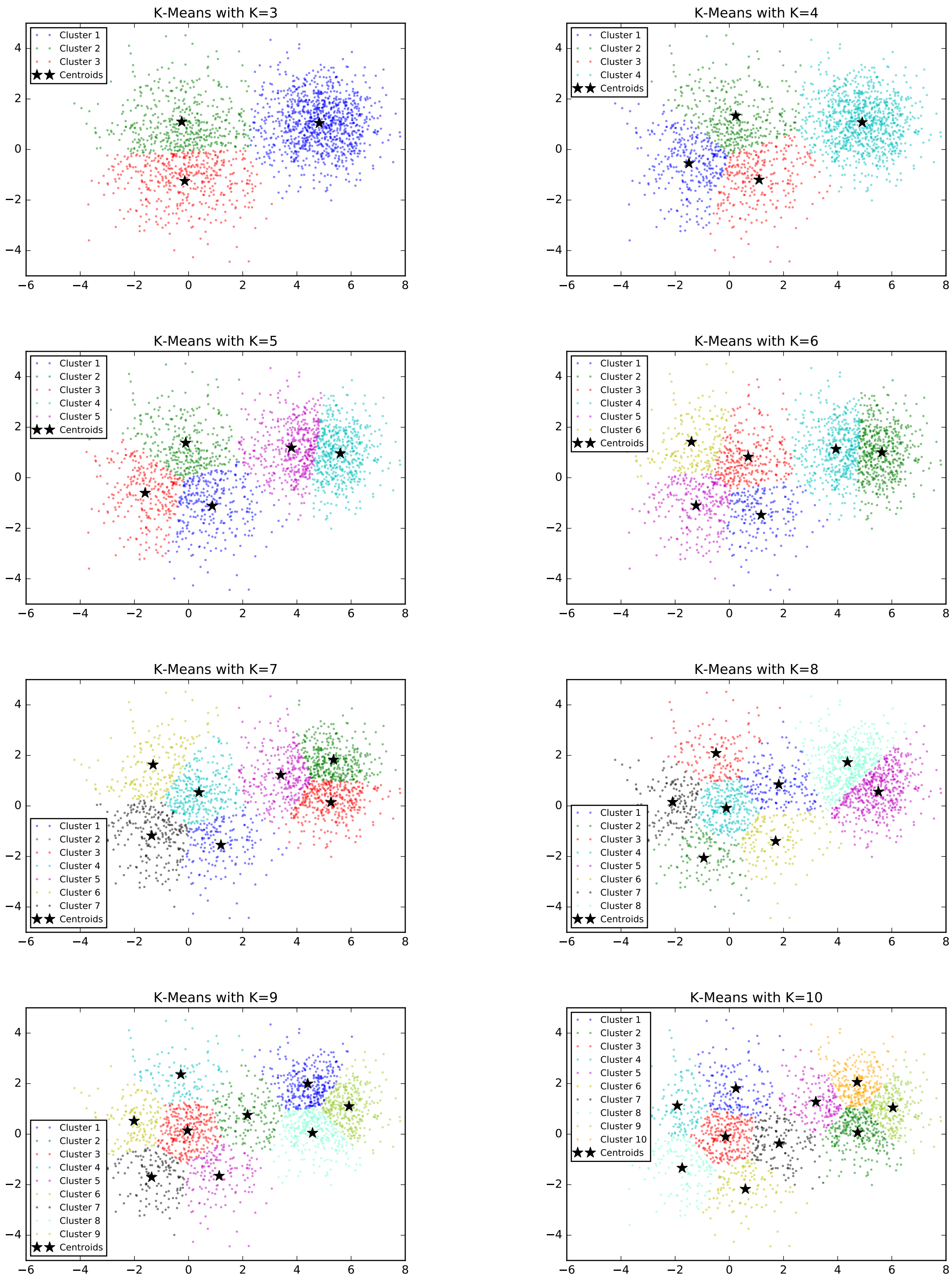


Figure 2: Clustering Result for clustering.txt with K-Means Algorithm

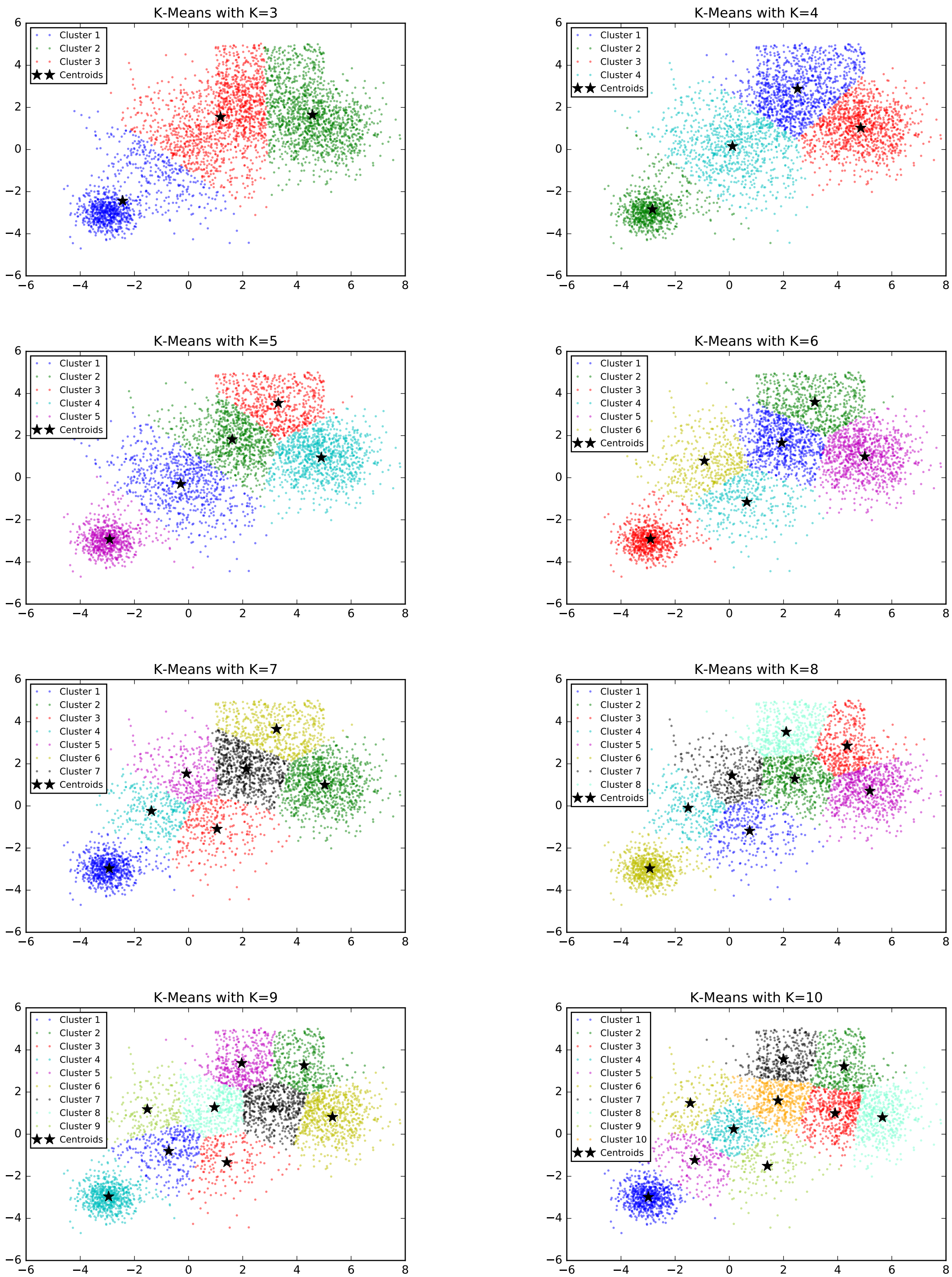


Figure 3: Clustering Result for bigClusteringData.txt with K-Means Algorithm

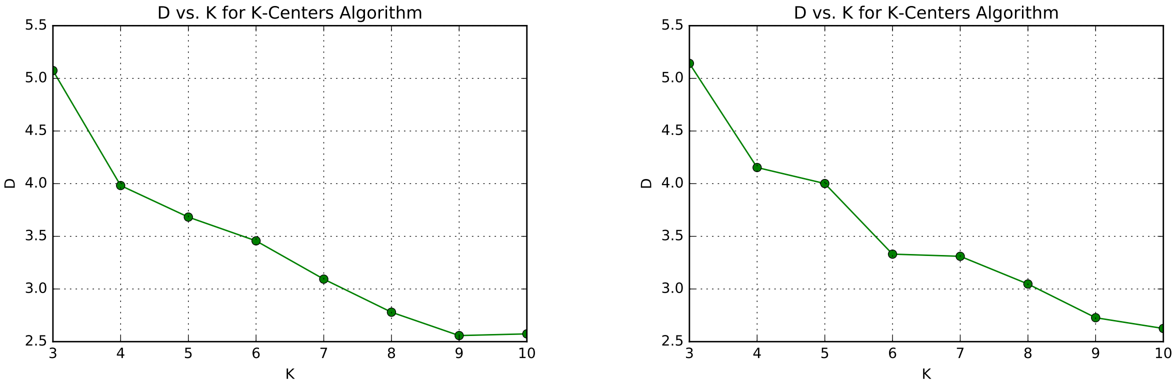
3. The Python code used for K-Means Algorithm is shown in Listing 1.

```python
import numpy as np
import time

def kMeans(X, K, tol=0.00001, random_state=None, verbose=True):
    """ function to implement the Lloyd's algorithm for k-means problem """
    np.random.seed(random_state)
    t0 = time.time()

    N, d = X.shape  # number of observations and dimensions
    index = np.random.choice(range(N), size=K, replace=False)
    Y = X[index, :]  # initial k centers
    C = np.zeros(N)
    D = 100
    count = 0
    diff = 100  # difference between D1 and D0

    while diff >= tol:
        D0 = D
        for i in range(N):
            # assign centers to ith data
            C[i] = np.argmin(np.sum((Y - X[i, :]) ** 2, axis=1))

        D = 0
        # re-compute the new centers
        for j in range(K):
            Y[j, :] = np.mean(X[C == j, :], axis=0)

        # compute the loss
        loss = np.zeros((N, K))
        for i in range(K):
            loss[:, i] = np.sqrt(np.sum((X - Y[i, :])**2, axis=1))
        D = np.max(np.min(loss, axis=1))
        diff = abs(D - D0)
        count += 1

    if verbose is True:
        t = np.round(time.time() - t0, 4)
        print('K-Means finished in ' + str(t) + 's, ' + str(count) + ' iters')

    return Y, C, D
```

Listing 1: K-Means Algorithm Python Code

II. Greedy K-centers Algorithm

1. In this part, with the Greedy K-Centers Algorithm, we choose the best distortion D as the the objection function. The change of D versus cluster number K is shown in Fig. 4, where the result for clustering.txt is shown in Fig. 4a and the result for bigClusteringData.txt is show in Fig. 4b.



(a) clustering.txt

(b) bigClusteringData.txt

Figure 4: Change of Distoration versus Cluster Number K for K-Center Algorithm

2. The scatter plot of the clustering result for clustering.txt is shown in Fig. 5 and the scatter plot of the clustering result for bigClusteringData.txt is shown in Fig. 6. The cluster centroids are clearly marked and different clusters are denoted by different colors.

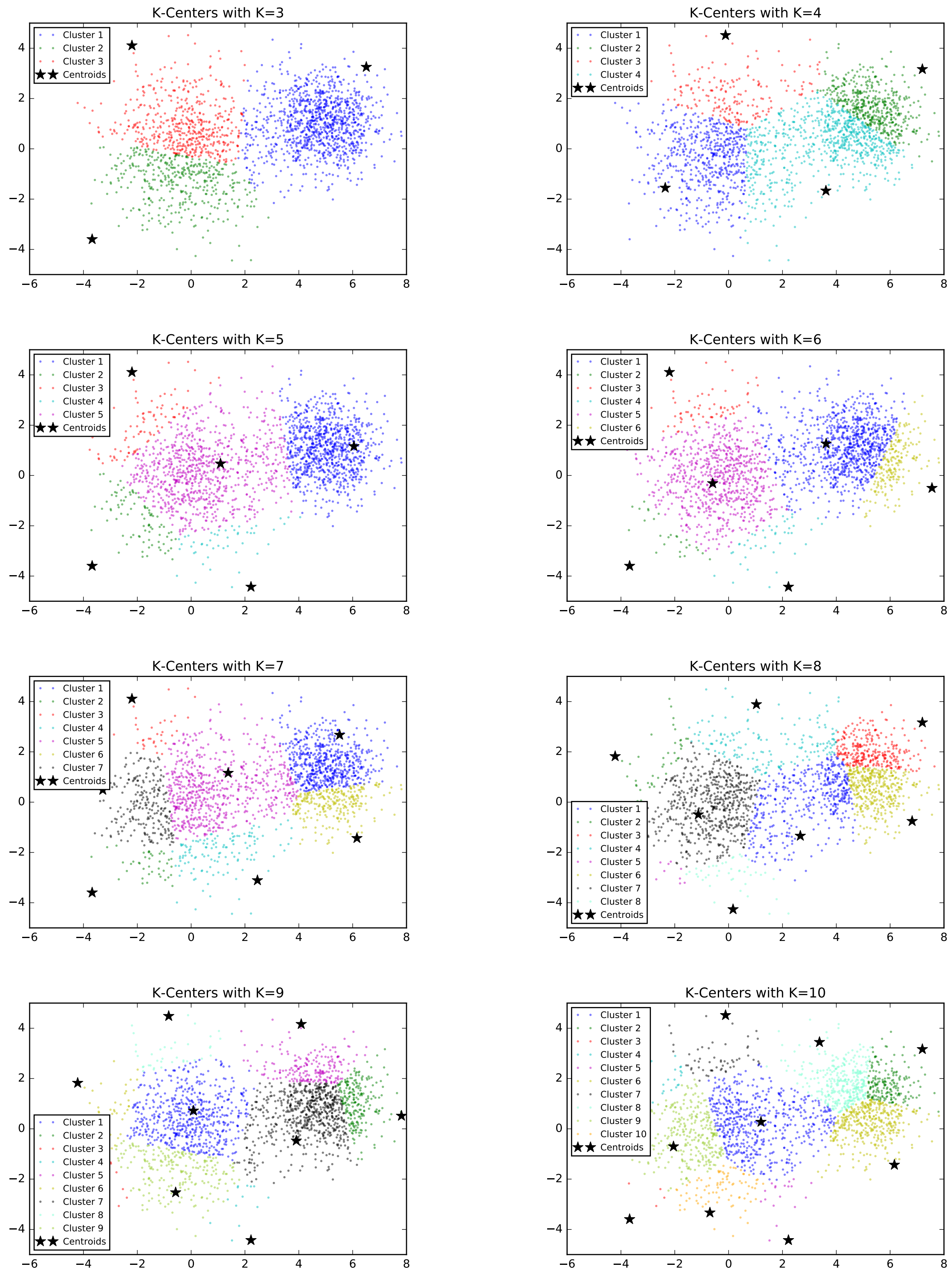


Figure 5: Clustering Result for clustering.txt with K-Center Algorithm

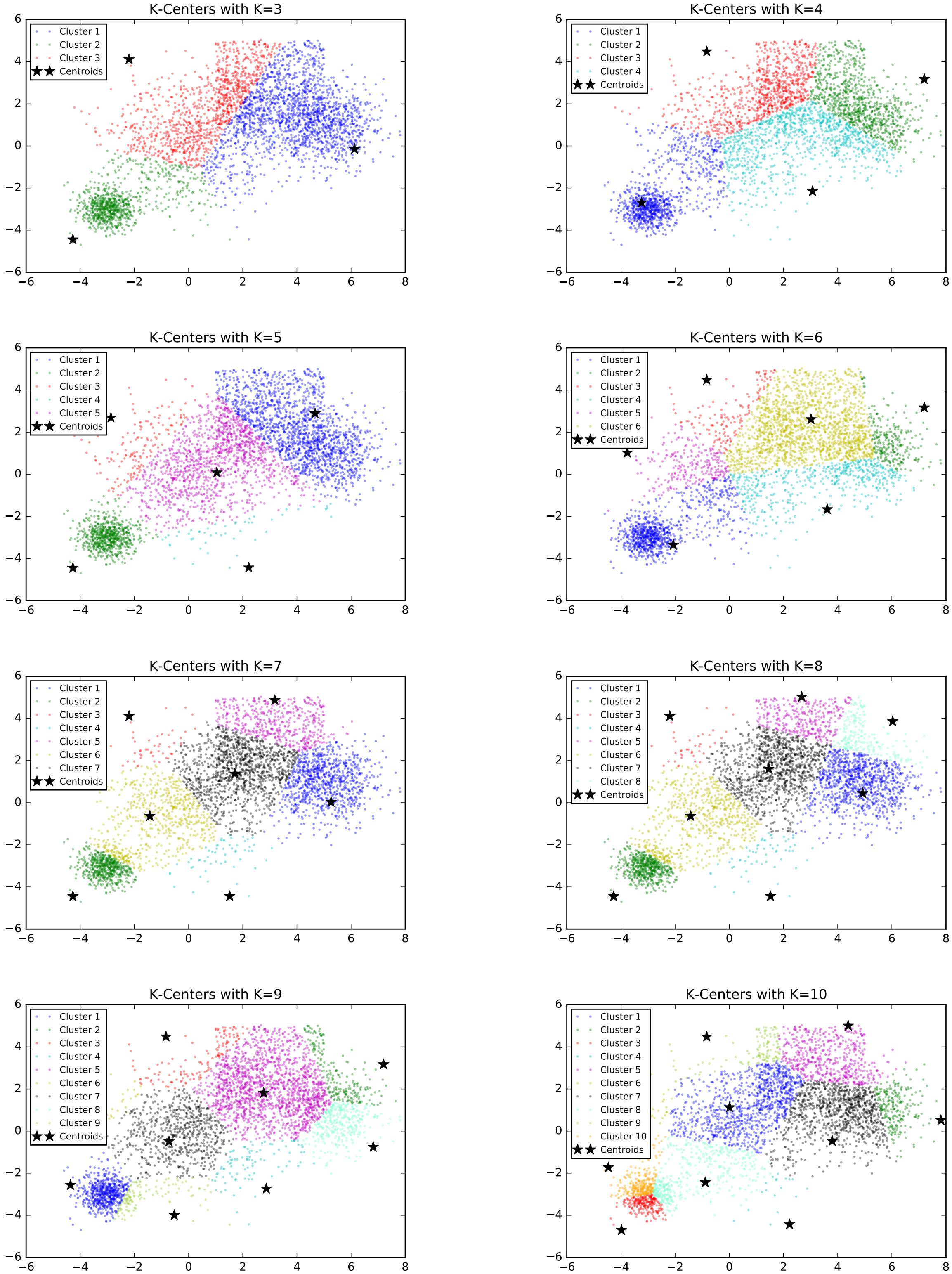


Figure 6: Clustering Result for bigClusteringData.txt with K-Center Algorithm

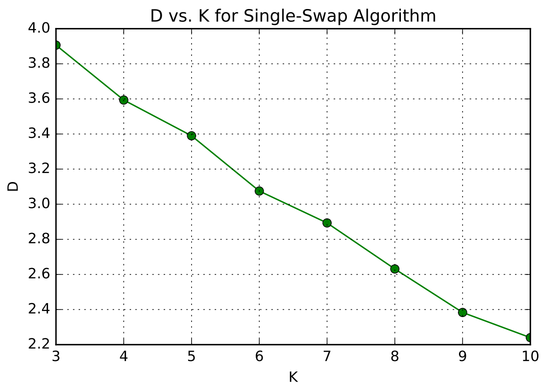
3. The Python code used for K-Centers Algorithm is shown in Listing 2.

```python
import numpy as np
import time

def kCenters(X, K, random_state=None, verbose=True):
    """ function to implement the greedy k-centers algorithm """
    np.random.seed(random_state)
    t0 = time.time()

    N, d = X.shape
    # find the initial center
    index = np.random.choice(range(N), size=1)
    Q = np.zeros((K, d))
    Q[0, :] = X[index, :]
    idx = [index]

    i = 1
    while i < K:
        distance = np.zeros((N, i))
        for j in range(i):
            distance[:, j] = np.sum((X - Q[j, :])**2, axis=1)
        min_distance = np.min(distance, axis=1)
        new_index = np.argmax(min_distance)
        idx.append(new_index)
        Q[i, :] = X[new_index, :]
        i += 1

    loss = np.zeros((N, K))
    for i in range(K):
        loss[:, i] = np.sqrt(np.sum((X - Q[i, :])**2, axis=1))
    D = np.max(np.min(loss, axis=1))
    C = np.argmin(loss, axis=1)

    if verbose is True:
        t = np.round(time.time() - t0, 4)
        print('K-Centers is finished in ' + str(t) + 's')

    return Q, C, D, idx
```

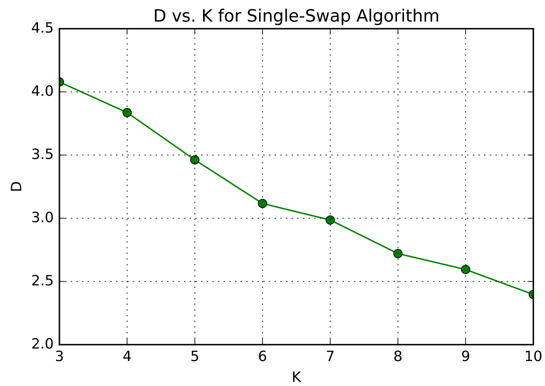
Listing 2: K-Centers Algorithm Python Code

III. Single-Swap Algorithm

1. In this part, with the Single-Swap Algorithm for K-Centers clustering, we choose the best distortion D as the the objection function. The change of D versus cluster number K is shown in Fig. 7, where the result for clustering.txt is shown in Fig. 7a and the result for bigClusteringData.txt is show in Fig. 7b.



(a) clustering.txt



(b) bigClusteringData.txt

Figure 7: Change of Distoration versus Cluster Number K for Single-Swap Algorithm

2. The scatter plot of the clustering result for clustering.txt is shown in Fig. 8 and the scatter plot of the clustering result for bigClusteringData.txt is shown in Fig. 9. The cluster centroids are clearly marked and different clusters are denoted by different colors.

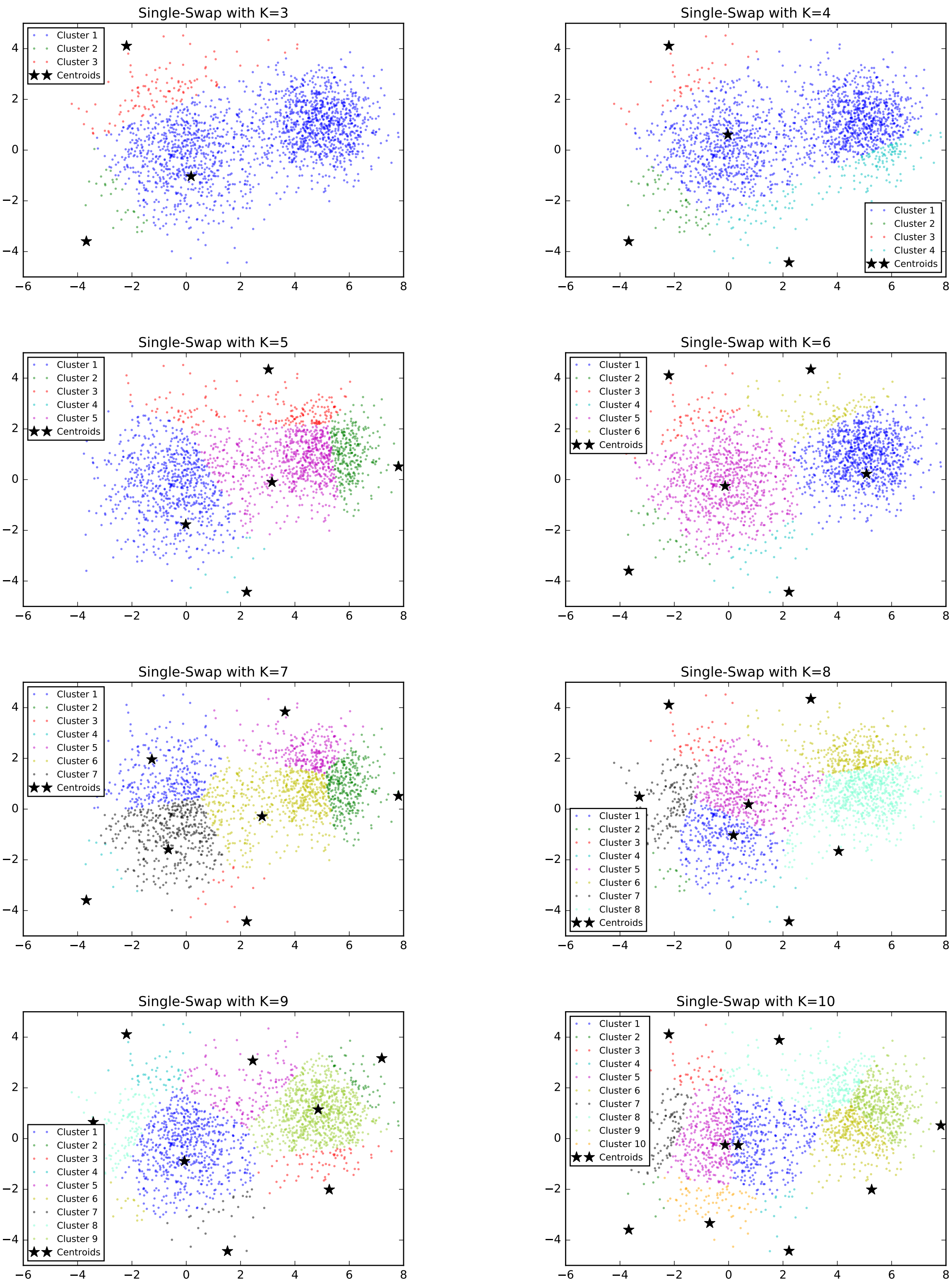


Figure 8: Clustering Result for clustering.txt with Single-Swap Algorithm

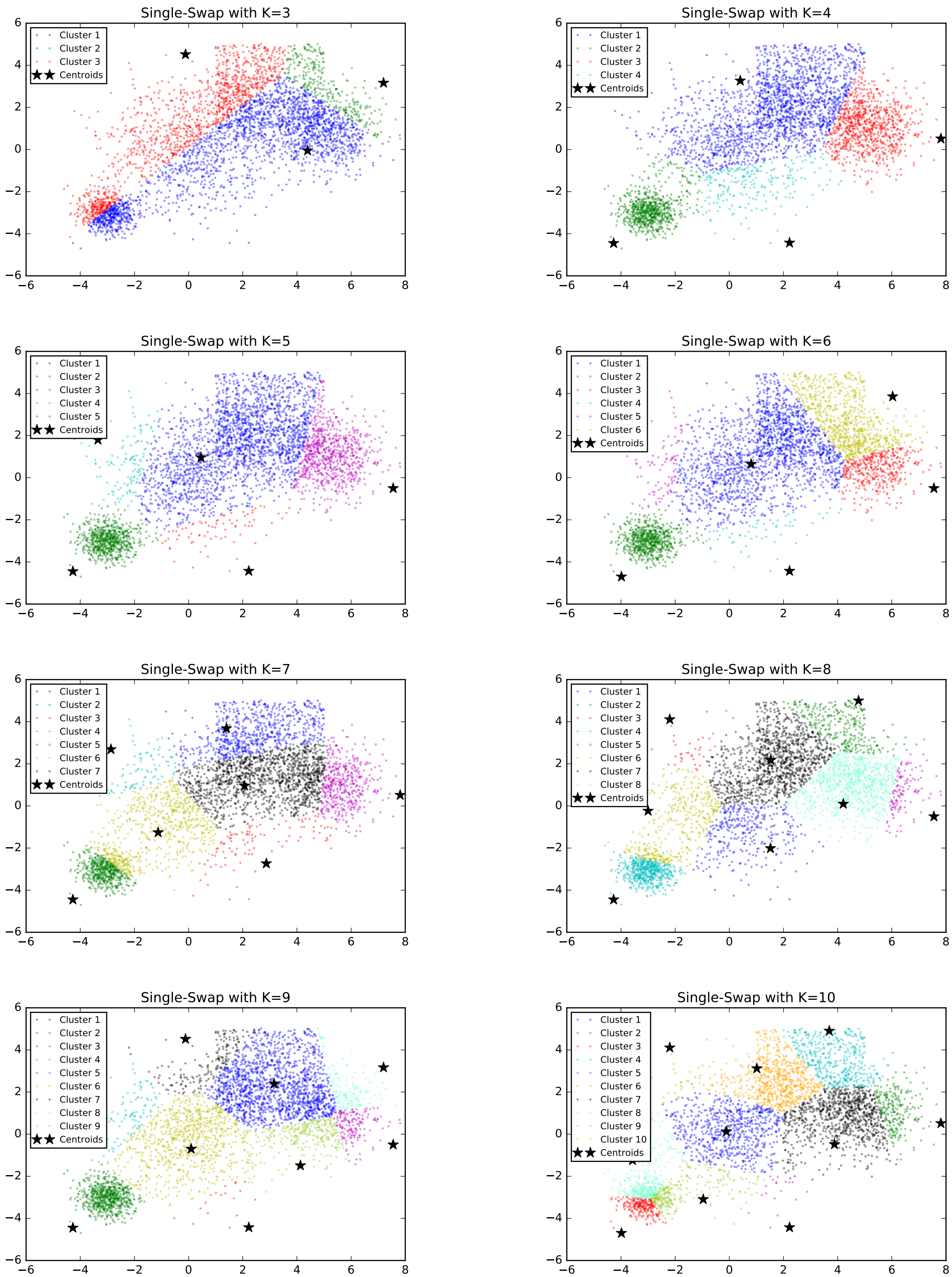


Figure 9: Clustering Result for bigClusteringData.txt with Single-Swap Algorithm

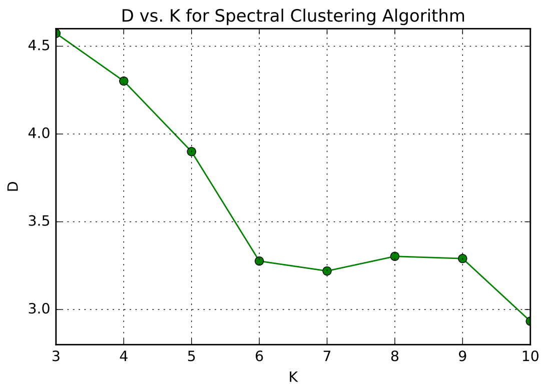
3. The Python code used for K-Centers Algorithm is shown in Listing 3.

```python
import numpy as np
import time
from k_centers import kCenters

def singleSwap(X, K, tau=0.05, random_state=None, verbose=True):
    """ function to implement the single-swap for k-centers algorithm """
    t0 = time.time()

    # calculate the initial centers
    Q, _, pre_cost, _ = kCenters(X, K, random_state=random_state,
                                 verbose=False)
    N, d = X.shape

    # compute the distance based on current centers
    distance = np.zeros((N, K))
    for idx in range(K):
        distance[:, idx] = np.sqrt(np.sum((X - Q[idx, :])**2, axis=1))
    cost = np.max(np.min(distance, axis=1))  # calculate cost

    i = 0
    while i < K:
        if i == 0:
            min_dist = np.min(distance[:, 0:], axis=1)
        elif i == (K - 1):
            min_dist = np.min(distance[:, :-1], axis=1)
        else:
            min_dist = np.minimum(np.min(distance[:, :i], axis=1),
                                  np.min(distance[:, (i + 1):], axis=1))
        swap = False  # keep recording whether or not swaped
        for j in range(N):
            tmp_dist = np.sqrt(np.sum((X - X[j, :])**2, axis=1))
            new_cost = np.max(np.minimum(min_dist, tmp_dist))
            if new_cost / cost < (1 - tau):
                Q[i, :] = X[j, :]
                distance[:, i] = tmp_dist
                swap = True
                cost = new_cost
        i += 1

        if swap is False:
            if i == K - 1:
                break
            else:
                i += 1
        elif (swap is True) and (i == K):
            i = 0

    C = np.argmin(distance, axis=1)

    if verbose is True:
        t = np.round(time.time() - t0, 4)
        print('Single-Swap is finished in ' + str(t) + 's')

    return Q, C, cost
```

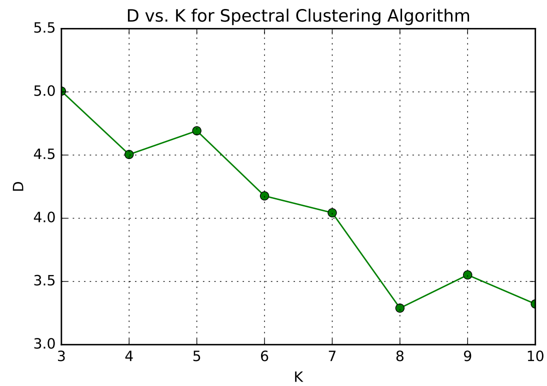
Listing 3: Single-Swap Algorithm Python Code

IV. Spectral Clustering Algorithm

1. In this part, with the Spectral Clustering Algorithm, we choose the best distortion D as the the objection function. The change of D versus cluster number K is shown in Fig. 10, where the result for clustering.txt is shown in Fig. 10a and the result for bigClusteringData.txt is show in Fig. 10b.



(a) clustering.txt



(b) bigClusteringData.txt

Figure 10: Change of Distoration versus Cluster Number K for Spectral Clustering

2. The scatter plot of the clustering result for clustering.txt is shown in Fig. 11 and the scatter plot of the clustering result for bigClusteringData.txt is shown in Fig. 12. The cluster centroids are clearly marked and different clusters are denoted by different colors.

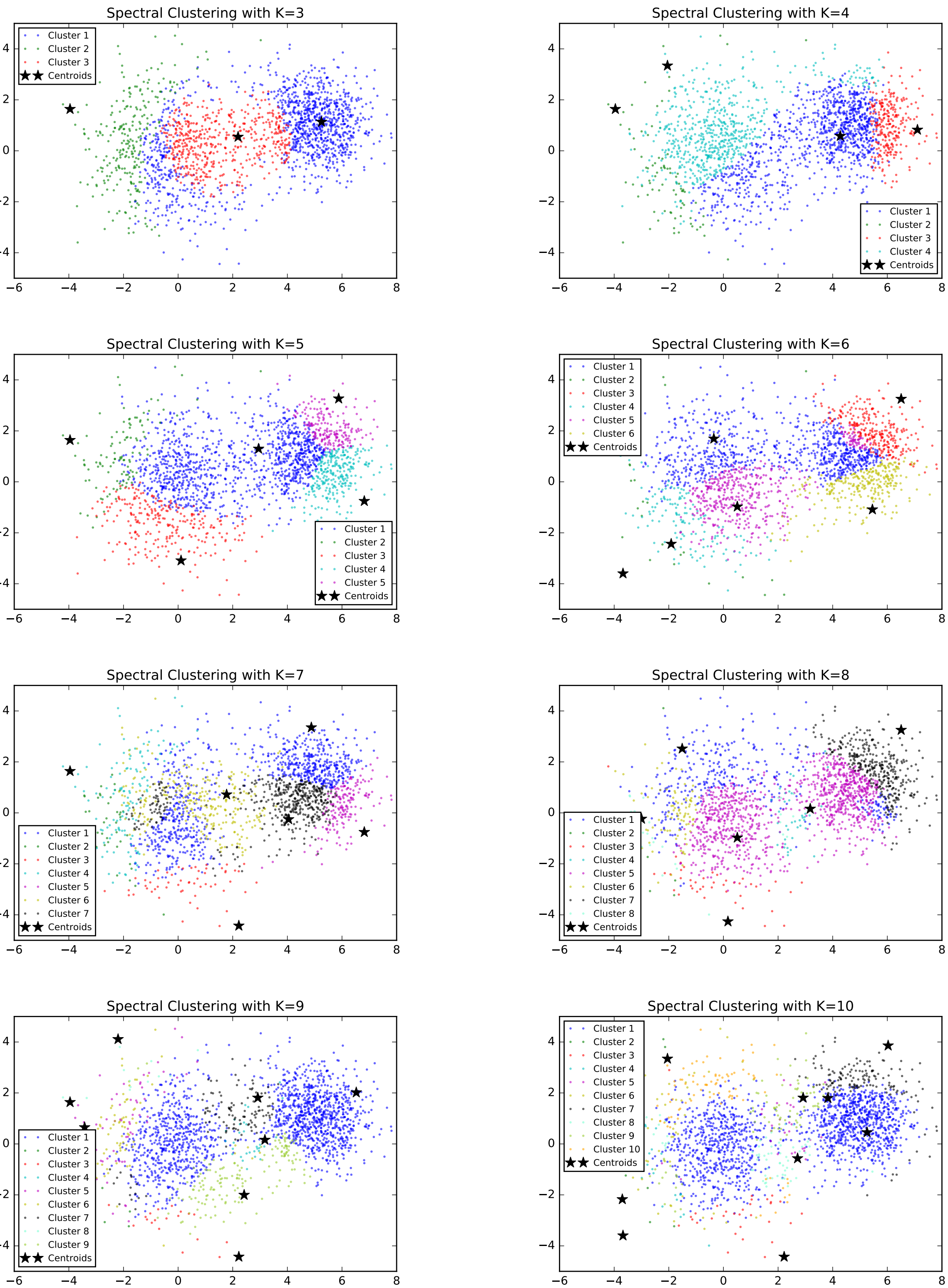


Figure 11: Clustering Result for clustering.txt with Spectral Clustering

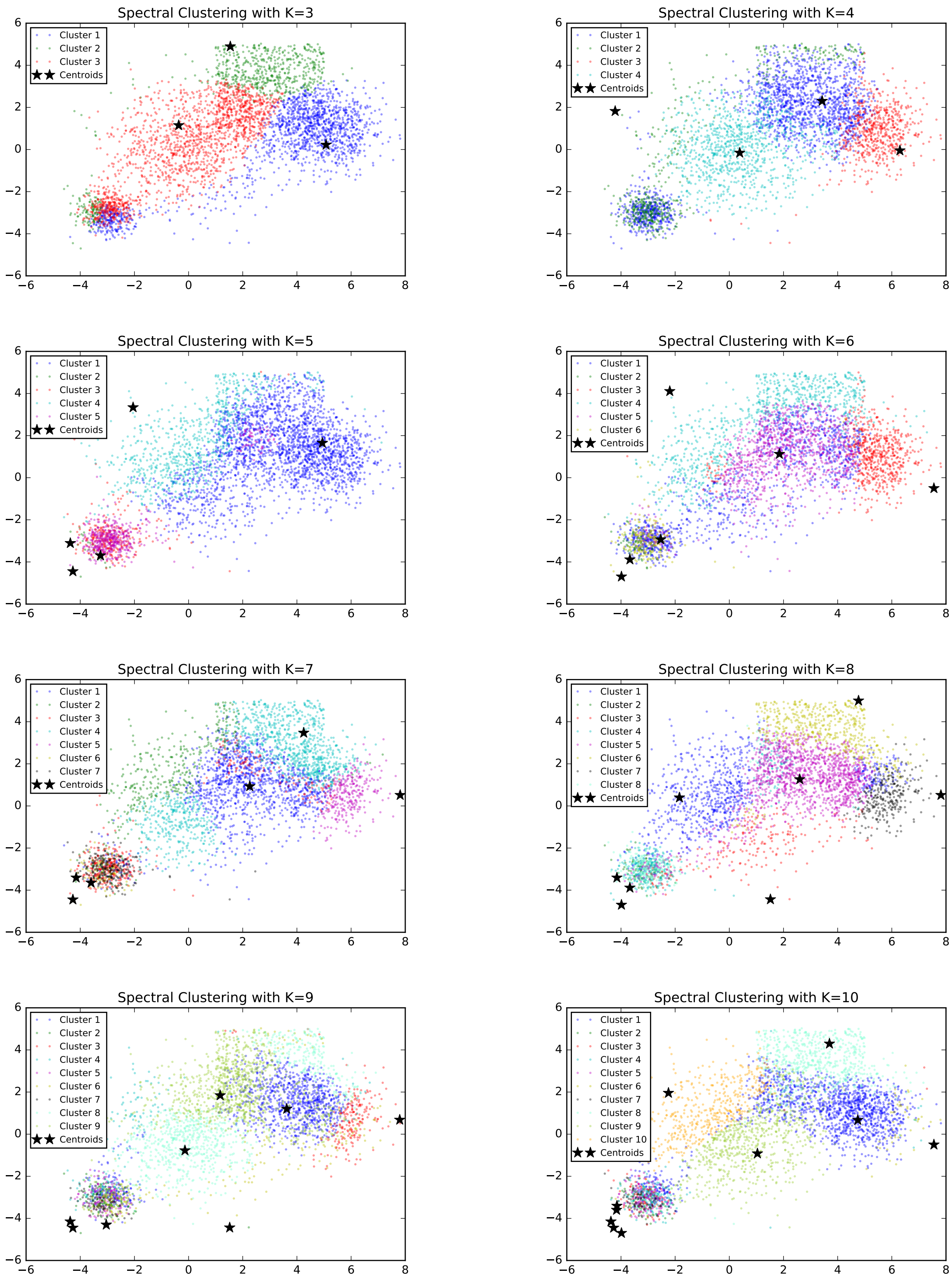


Figure 12: Clustering Result for bigClusteringData.txt with Spectral Clustering

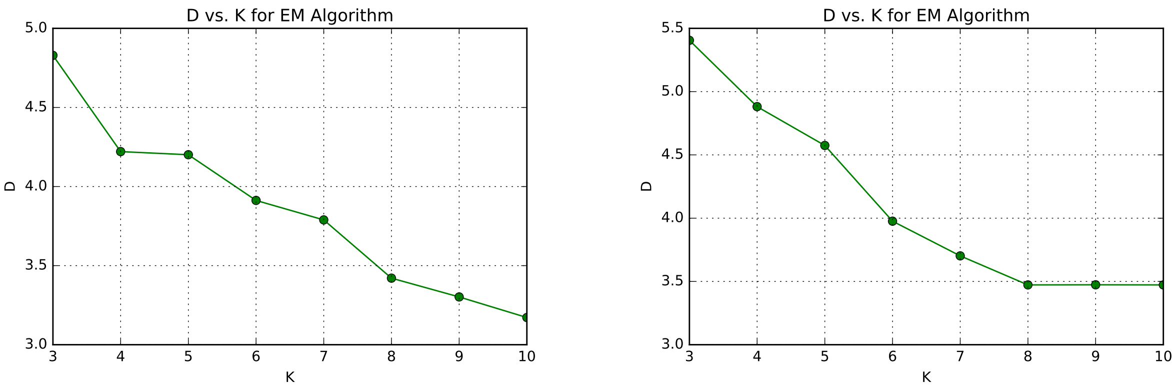
3. The Python code used for K-Centers Algorithm is shown in Listing 4.

```
import numpy as np
import time
from k_centers import kCenters

def spectralClustering(X, K, random_state=None, verbose=True):
    """ function to implement the spectral clustering algorithm """
    t0 = time.time()

    N, d = X.shape
    W = np.zeros((N, N))  # adjacency matrix W
    for i in range(N):
        distance = np.sqrt(np.sum((X - X[i, :])**2, axis=1))
        W[:, i] = distance

    diag = np.sum(W, axis=1)
    D = np.diag(diag)  # diagnoal matrix D
    L = D - W  # Laplacian matrix L
    L = np.identity(N) - np.dot(np.linalg.inv(D), W)
    eigvals, U = np.linalg.eigh(L)
    U = U[:, -K:]  # first K eigenvectors

    # call k-means for clustering
    _, C, _, idx = kCenters(U, K, random_state=random_state, verbose=False)

    Q = X[idx, :]
    loss = np.zeros((N, K))
    for i in range(K):
        loss[:, i] = np.sqrt(np.sum((X - Q[i, :])**2, axis=1))
    D = np.max(np.min(loss, axis=1))

    if verbose is True:
        t = np.round(time.time() - t0, 4)
        print('Spectral Clustering finished in ' + str(t) + 's')

    return W, U, Q, C, D
```

Listing 4: Spectral Clustering Algorithm Python Code

V. Expectation Maximization (EM) Algorithm

1. In this part, assuming we are dealing with the Gaussian Mixture Model (GMM), then with the EM algorithm, we choose the best distortion D as the the objection function. The change of D versus cluster number K is shown in Fig. 13, where the result for clustering.txt is shown in Fig. 13a and the result for bigClusteringData.txt is show in Fig. 13b.



(a) clustering.txt

(b) bigClusteringData.txt

Figure 13: Change of Distoration versus Cluster Number K for EM Algorithm

2. The scatter plot of the clustering result for clustering.txt is shown in Fig. 14 and the scatter plot of the clustering result for bigClusteringData.txt is shown in Fig. 15. The cluster centroids are clearly marked and different clusters are denoted by different colors.

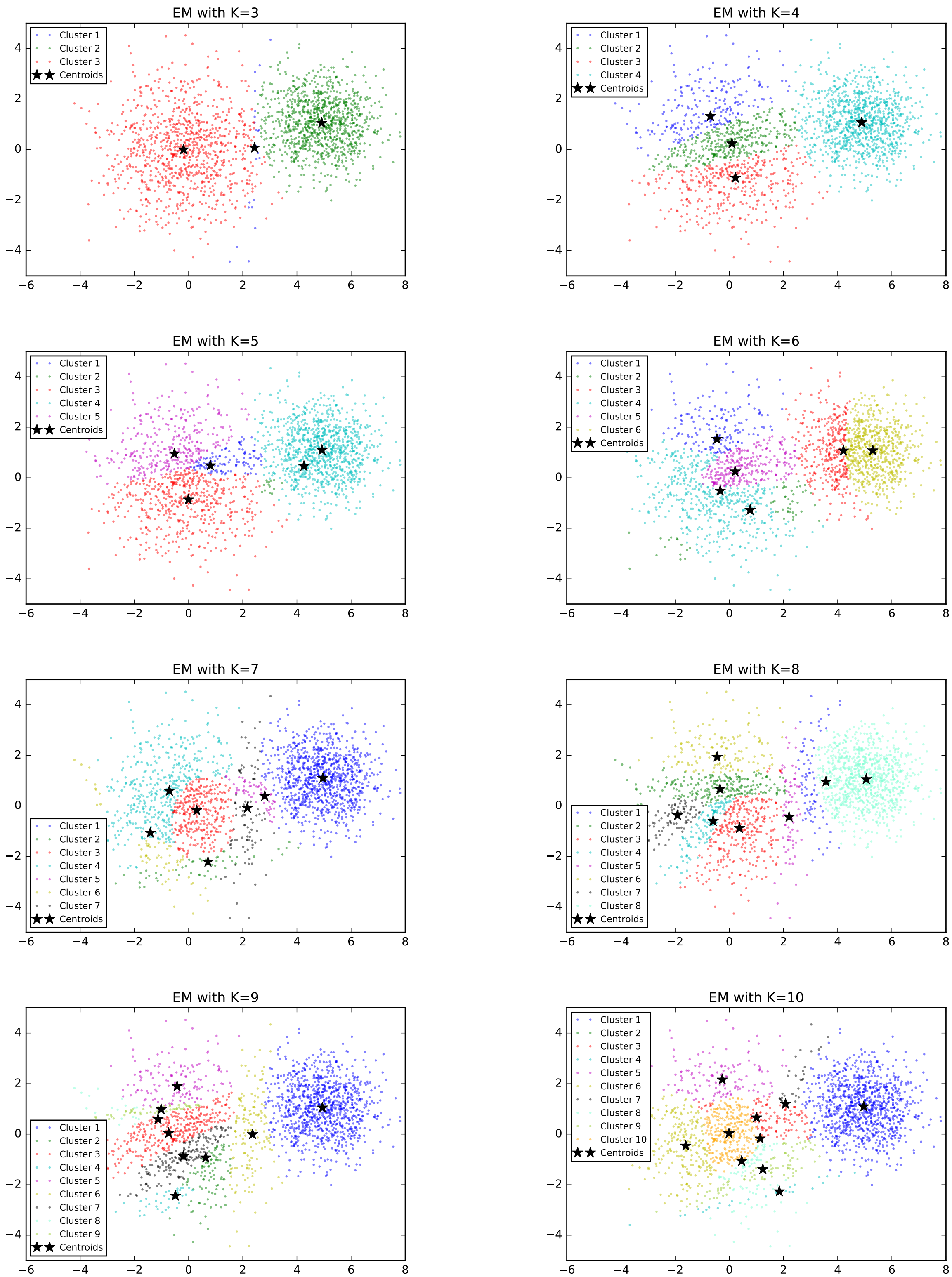


Figure 14: Clustering Result for clustering.txt with EM Algorithm

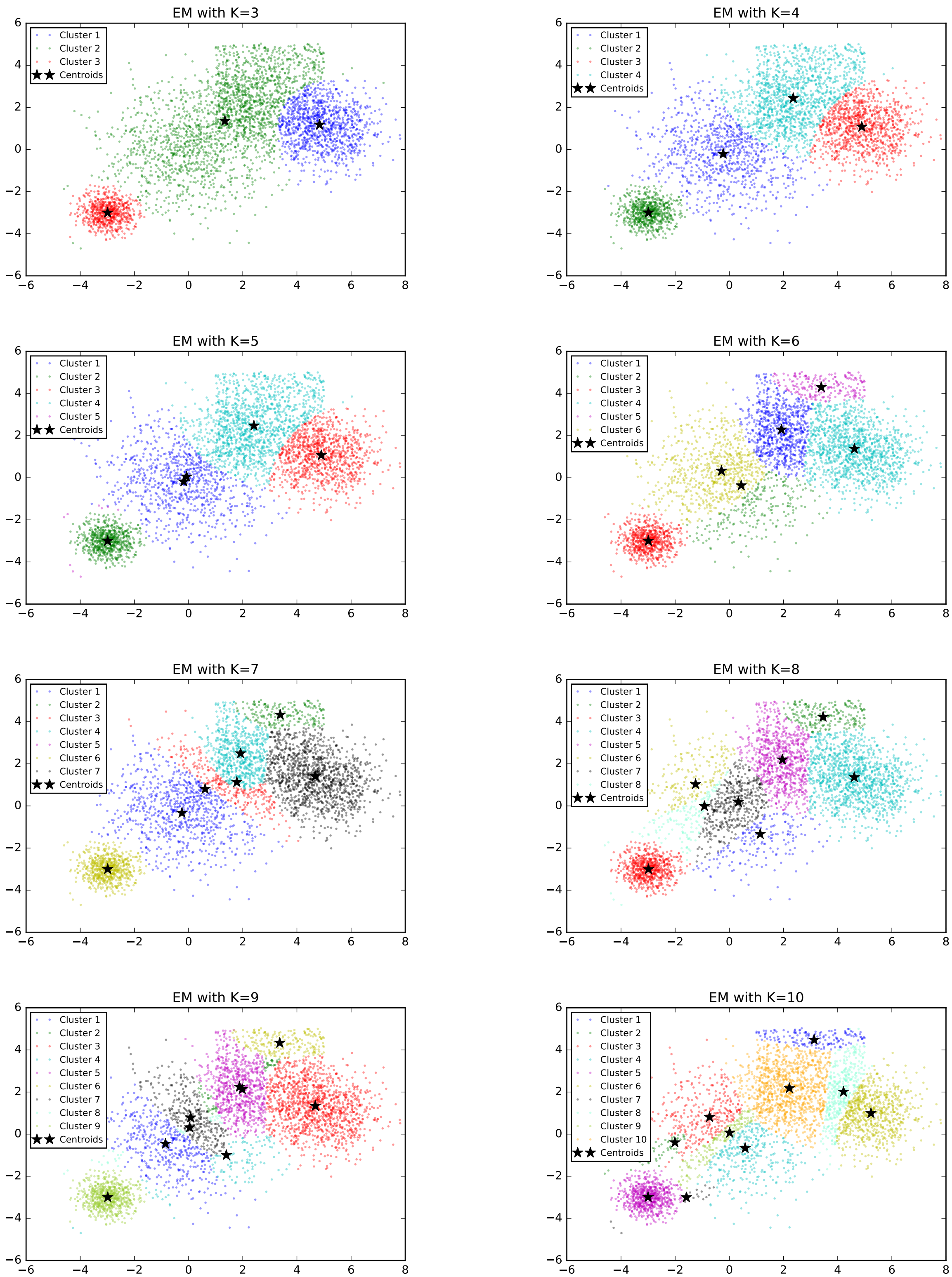


Figure 15: Clustering Result for bigClusteringData.txt with EM Algorithm

3. The Python code used for K-Centers Algorithm is shown in Listing 5.

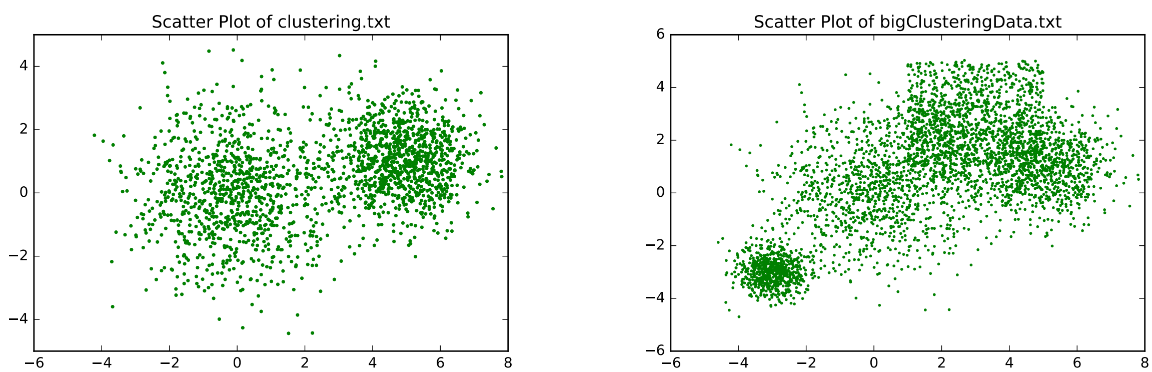
```python
import numpy as np
import time

class EM(object):
    """ self-defined calss for EM algorithm """
    def __init__(self, m, threshold=0.01, random_state=None, maxIter=500):
        """ initialize the EM algorithm """
        self.m = m
        self.threshold = threshold
        self.random_state = random_state
        self.maxIter = maxIter
        self.w = None
        self.gamma = None
        self.mu = None
        self.sigma = None
        self.gaussianProb = None
        self.logLikelihood = None
        self.distance = None
        self.D = None

    def train(self, x, verbose=False):
        """ function to perform EM algorithm on X """
        t0 = time.time()
        np.random.seed(self.random_state)

        # initialize the mean and covariance matrix
        self.initialize(x)

        # iterate through E and M steps
        for i in range(1, self.maxIter + 1):
            self.estep(x)
            self.mstep(x)
            if abs(self.logLikelihood[-1] - self.logLikelihood[-2]) \
               / abs(self.logLikelihood[-2]) < self.threshold:
                for i in range(self.m):
                    self.distance[:, i] = np.sqrt(np.sum((x - self.mu[i])**2,
                                                          axis=1))
                self.D = np.max(np.min(self.distance, axis=1))
                if verbose is True:
                    t = np.round(time.time() - t0, 4)
                    print('Reach threshold at', i,
                          'th iters in ' + str(t) + 's')
                return

        for i in range(self.m):
            self.distance[:, i] = np.sqrt(np.sum((x - self.mu[i])**2, axis=1))
        self.D = np.max(np.min(self.distance, axis=1))
        if verbose is True:
            t = np.round(time.time() - t0, 4)
            print('Stopped, reach the maximum iteration ' + str(t) + 's')

    def initialize(self, x):
        """ function to initialize the parameters """
        n, dim = x.shape  # find the dimensions
        self.distance = np.zeros((n, self.m))
```

```
        self.w = np.ones(self.m) * (1 / self.m)
        self.gamma = np.zeros((n, self.m))
        self.gaussianProb = np.zeros((n, self.m))
        self.mu = [None] * self.m
        self.sigma = [None] * self.m
        self.logLikelihood = []

        cov = np.cov(x.T)
        mean = np.mean(x, axis=0)
        for k in range(self.m):
            self.mu[k] = mean + np.random.uniform(-0.5, 0.5, dim)
            self.sigma[k] = cov

        # update gamma
        self.gamma = self.gammaprob(x, self.w, self.mu, self.sigma)

        # calculate the expectation of log-likelihood
        self.logLikelihood.append(self.likelihood())

    def estep(self, x):
        """ function to conduct E-Step for EM algorithm """
        self.gamma = self.gammaprob(x, self.w, self.mu, self.sigma)

    def mstep(self, x):
        """ function to conduct M-Step for EM algorithm """
        n, dim = x.shape
        sumGamma = np.sum(self.gamma, axis=0)
        self.w = sumGamma / n

        for k in range(self.m):
            self.mu[k] = np.sum(x.T * self.gamma[:, k], axis=1) / sumGamma[k]
            diff = x - self.mu[k]
            weightedDiff = diff.T * self.gamma[:, k]
            self.sigma[k] = np.dot(weightedDiff, diff) / sumGamma[k]
            if np.linalg.matrix_rank(self.sigma[k]) != 3:
                randv = np.random.random(dim) / 10000
                self.sigma[k] = self.sigma[k] + np.diag(randv)

        # calculate the expectation of log-likelihood
        self.logLikelihood.append(self.likelihood())

    def gammaprob(self, x, w, mu, sigma):
        """ function to calculate the gamma probability """
        for k in range(self.m):
            self.gaussianProb[:, k] = self.gaussian(x, mu[k], sigma[k])

        weightedSum = np.sum(w * self.gaussianProb, axis=1)
        gamma = ((w * self.gaussianProb).T / weightedSum).T

        return gamma

    def gaussian(self, x, mu, sigma):
        """ function to calculate the multivariate gaussian probability """
        inversion = np.linalg.inv(sigma)
        part1 = (-0.5 * np.sum(np.dot(x - mu, inversion) * (x - mu), axis=1))
        part2 = 1 / ((2 * np.pi) ** (len(mu) / 2) *
                     (np.linalg.det(sigma) ** 0.5))
```

```python

            pdf = part2 * np.exp(part1)

            return pdf

        def likelihood(self):
            """ function to calculate the log likelihood """
            log = np.log(np.sum(self.w * self.gaussianProb, axis=1))
            logLikelihood = np.sum(log)

            return logLikelihood

        def get_label(self):
            """ function to predict the classes using calculated parameters """
            label = np.argmax(self.w * self.gaussianProb, axis=1)

            return label
```

Listing 5: EM Algorithm Python Code

2 Natural Clusters Discussion



(a) clustering.txt

(b) bigClusteringData.txt

Figure 16: Scatter Plot of Original Data

3 K-Means Convergence Discussion

4 Computational Effort Discussion

Table 1: Computation Time Comparison for clustering.txt

K	K-Means	K-Centers	Single-Swap	Spectral Clustering	EM
3	0.2673	0.0067	0.1582	1.5716	0.4412
4	0.2543	0.0010	0.1971	1.4587	0.5078
5	0.3624	0.0012	0.1895	1.4132	0.4881
6	0.4325	0.0013	0.2668	1.4840	0.7462
7	0.2837	0.0015	0.2693	1.4877	0.8227
8	0.1923	0.0019	0.3306	1.4708	0.9216
9	0.3481	0.0024	0.3012	1.4669	1.0106
10	0.1989	0.0026	0.3790	1.5053	1.1277

Note: all measured time is in unit of seconds (s)

Table 2: Computation Time Comparison for bigClusteringData.txt

K	K-Means	K-Centers	Single-Swap	Spectral Clustering	EM
3	0.4725	0.0020	0.4511	11.1860	0.3412
4	0.8326	0.0019	0.5692	10.2542	0.2553
5	0.4722	0.0021	0.6445	10.5166	0.5763
6	1.2580	0.0024	0.8809	10.2167	0.6196
7	0.9619	0.0025	0.7638	9.8986	1.1591
8	0.7767	0.0034	0.7815	9.4616	1.3177
9	0.7177	0.0037	0.9010	9.8733	1.4977
10	1.2369	0.0041	1.0744	9.8636	1.7560

Note: all measured time is in unit of seconds (s)

5 Single-Swap Algorithm Discussion