#4.
라이브러리를 이용한
클러스터링 알고리즘 비교분석

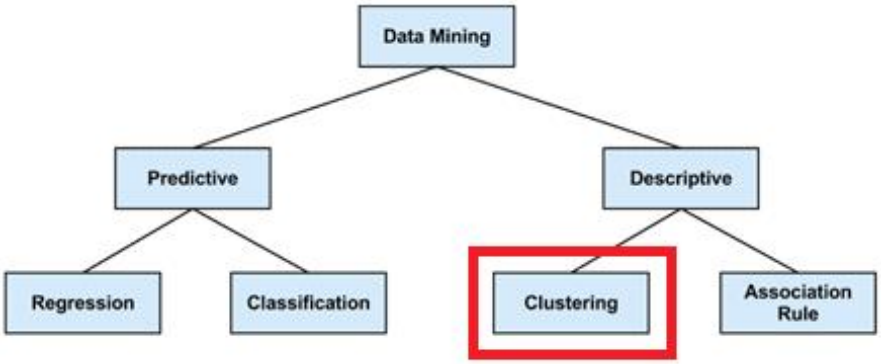
Monday, 11 June 2018

KeonHee Lee

목차

1. 개요

클러스터링(Clustering)이란 구분하려고 하는 각 class에 대한 아무런 지식이 없는 상태에서 분류(classify)하는 것이므로 자율학습(Unsupervised Learning)에 해당한다. 즉, sample들에 대한 지식없이 유사도(similarity)에 근거하여 cluster들을 구분한다. 패턴 공간에 주어진 유한 개의 패턴들이 서로 가깝게 모여서 무리를 이루고 있는 패턴 집합을 군집(cluster) 이라하고 무리지어 나가는 처리 과정을 클러스터링(clustering) 이라 한다. cluster간의 유사도를 평가하기 위해 여러 가지의 거리 측정 함수를 사용하는데 예를 들면 Euclidean distance, Mahalanobis distance, Lance-Williams distance, Hamming distance 등이 사용된다.



▲ 그림 1. 클러스터링이 속하는 범주

또한 클러스터링 분석 기법에서는 대표적으로 아래의 것들이 있다.

- K-means
- Mean-Shift Clustering
- Density-Based Speatial Clustering of Applications with Noise(DBSCAN)
- Expectation-Maximization (EM) Clustering
- Agglomerative Hierachical Clustering

▲ 표 1. 클러스터링 알고리즘의 일부

이 외에도 상당히 많은 클러스터링 알고리즘이 있지만, 나는 여기서 K-means, DBSCAN, Agglomerative Hierachical Clustering 3가지 알고리즘을 활용해는 쪽으로 구현해 볼 것이다.

그리고 이러한 알고리즘을 구현하기 위해서 2가지 언어를 사용할 것인데, Python과 Java를 활용해서 구현을 해 볼것이며, 직접적으로 구현하기보다는 라이브러리를 활용해서 구현을 해볼 것이다.

2. 클러스터링 관련 패키지/라이브러리

클러스터링 알고리즘 관련해서 사용할 수 있는 패키지의 종류는 많지만 흔히 활용되는 라이브러리는 Python의 scikit-learn 라이브러리와 Java의 weka 패키지가 있다.

▲ 그림 2,3. Python의 scikits-learn(왼쪽) / Java의 Weka(오른쪽)

먼저 scikit-learn 라이브러리에 대해서 설명하겠다. scikit-learn은 우선 Python 모듈로 설치하기 위해선 pip을 활용해서 설치를 해야한다. 또한 NumPy와 SciPy를 사용하며, 그래프로 그릴려면 matplotlib을 사용하면 된다. 설치법은 아래와 같다.

```
$ pip install numpy scipy matplotlib
```

▲ 표 2. scikits-learn 라이브러리 설치 관련 명령어

그리고 weka 패키지의 경우는 2가지가 제공되는데 API 패키지와 GUI 프로그램으로 활용할 수 있다. 여기서 2가지를 활용할 것이며, 아래의 링크에서 Weka 패키지 및 프로그램을 받을 수 있다.

```
https://www.cs.waikato.ac.nz/ml/weka/
```

▲ 표 3. weka 패키지 링크

특히 weka API 관련해서는 scikit-learn과 비슷하게 직접 데이터 셋을 활용해서 소스코드로 구현할 수 있게 되어있고, GUI의 경우는 데이터 셋(.arff 파일)을 링크 걸어주면 클러스터링 결과를 그래프로 보여준다. Python의 scikit-learn의 경우는 Numpy에 데이터 셋을 넣어서 링크를 걸어주면 되지만, Java의 weka의 경우는 arff파일의 작성 문법에 맞춰서 작성 해야된다. 이러한 점을 고려해서 각 라이브러리/패키지에 대해서 알아보자.

3. Python : Scikit-learn(scipy)

1) matplotlib, numpy

위에서 설명했듯이 데이터셋을 활용할려면 numpy를 활용해서 데이터셋을 넣어줘야하며, matplotlib를 이용해서 클러스터링 결과를 그래프로 보여줄 수 있다.

matplotlib의 사용법은 아래와 같다.

```
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color='black')

plt.show()
```

▲ 표 4. matplotlib 사용방법

우선 Python에선 **matplotlib.pyplot**을 import 걸어준다. 그 후에 scatter라는 멤버함수를 이용해서 그래프 내의 데이터 출력방법에 대해서 정할 수 있으며, 위의 경우는 kmeans.labels_라는 데이터를 raibow라는 색상으로 출력하며, kmeans.cluster_centers_라는 데이터를 black색상으로 출력한다는 것을 의미한다. 그리고 show()라는 함수를 사용함으로써, 그래프가 출력된다. 이것 외에도 상세한 사용방법들이 많으므로 부록의 링크를 확인하면 되겠다.

numpy의 사용법은 아래와 같다.

```
import numpy as np

X = np.array(data)


k_means(X=X)
```

▲ 표 5. numpy 사용방법

numpy를 우선 import 걸어서 라이브러리를 사용해야한다. 그 후에 만약 데이터셋을 리스트로 넣는다면 위와 같이 array라는 함수를 사용해서 매개변수로 리스트를 넣어주면 반환값으로 numpy의 데이터 셋을 넘겨준다. 이러한 반환 값으로 scikit-learn 관련 함수들에 넣어주면 된다. 이 또한 이것 외에도 더 많은 사용방법이 있기 때문에 부록의 링크를 확인하면 될 것이다.

2) 사용 방법

sklearn을 이용한 K-means 구현을 예시로 사용방법에 대해서 알아보자.

```python
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

data = []

def set_data():
    filename = input("input file name: ")
    f = open(filename, "r")
    while True:
        line = f.readline()
        if not line: break
        point = line.split(",")
        data.append([float(point[0]), float(point[1])])

def k_means(X):
    kmeans = KMeans(n_clusters=5)
    kmeans.fit(X)

    print(kmeans.cluster_centers_)
    plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow')
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color='black')


if __name__ == "__main__":
    set_data()
    X = np.array(data)
    k_means(X=X)
    plt.show()
```

▲ 표 6. sklearn을 이용한 k-means 구현 예시

0. sklearn.cluster를 import 해준다. (from sklearn.cluster import KMeans)
1. 위에서 설명한 numpy를 이용해서 데이터셋을 먼저 넣어준다. (X = np.array(data))
2. KMeans의 생성자 선언 및 fit이라는 함수를 선언한다.

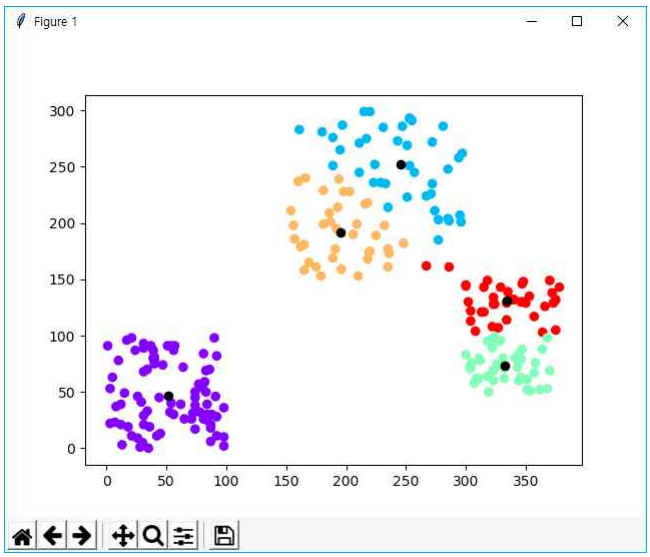
* 특히 2번의 경우는 fit_predict()라는 함수를 사용해도 된다.

3. 위에서 설명한 matplotlib를 이용해서 그래프로 출력하면 된다.

참고로 각 클러스터링마다 생성자나 함수가 다르긴 하지만 전반적으로 fit()이라는 함수를 통해서 클러스터링을 수행하게 된다. 필자의 경우는 K-means, DBSCAN, Agglomerative Hierachical Clustering 3가지를 활용할 것인데, k-means와 Agglomerative Hierachical Clustering는 클러스터의 개수를, DBSCAN은 eps와 최소 샘플의 개수를 생성자의 매개변수로 받는다. 위와 같은 과정을 거치면 k-means clustering 결과를 출력할 수 있다. 상세한 결과는 3번 항목에서 확인해보자.

3) 수행 결과

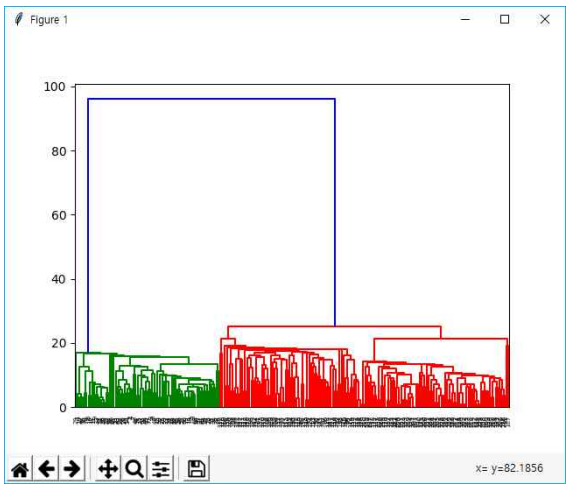
- K-means

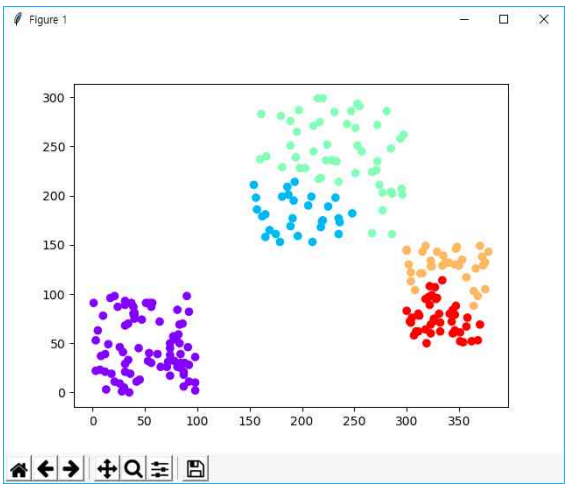


▲ 그림 4. k-means 출력 결과 (n_clusters=5)

참고로 검정색은 cluster의 중심점으로 선정하였으며, 주변 데이터들의 거리를 측정해서 군집화(Clustering)한 모습을 확인 할 수 있었다.

- Agglomerative Hierachical Clustering

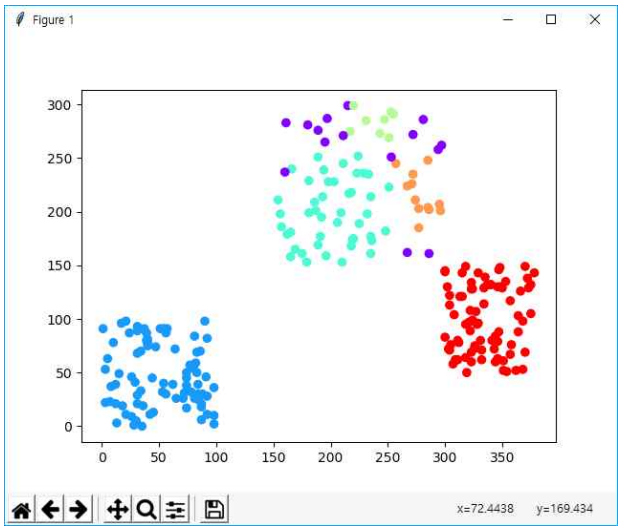




▲ 그림 5,6. Agglomerative Hierachical Clustering 출력 결과 (n_clusters=5)

이 경우는 계층적으로 주변의 데이터들과 병합 군집을 하기 때문에 dendrogram으로 출력한 것은 어떻게 계층적으로 이루어졌는지 확인을 할 수 있고, 오른쪽의 경우는 실제 데이터가 어떻게 군집화가 되었는지 확인을 할 수 있다. K-means와는 비슷한 것 같지만, 조금은 다른 결과가 나왔다.

- DBSCAN



▲ 그림 7. DBSCAN 출력 결과 (eps=20, min_sample=5)

DBSCAN의 경우는 k-means와 비교해선 다소 느린걸로 알려진 알고리즘이다. 하지만 밀집 지역을 찾아서 군집화를 시키 때문에 클러스터의 개수를 미리 지정할 필요가 없는 알고리즘이며, sklearn에서는 실제로 클러스터의 개수를 지정하라는 것이 아닌, eps와 min_sample을 지정하면 된다. eps의 거리 내에 있으면서 min_sample 개수보다 많으면 해당 데이터는 유효하다고 판단해서 묶어버리기 때문에 위의 그래프를 보면 가까운 데이터 위주로 묶어버리고, 나머지는 보라색으로 표기된 것을 확인 할 수 있다.

그리고 클러스터링을 수행하면서 걸린 시간은 아래와 같다.

K-means Clustering	C:₩Python35₩python.exe C:/Python/Algorithm/School/clustering/k-means.py input file name: data07.txt --- 0.03696894645690918 seconds ---
Agglomerative Hierachical Clustering	C:₩Python35₩python.exe C:/Python/Algorithm/School/clustering/agglomerative.py input file name: data07.txt 0.017988920211791992 seconds ---
DBSCAN	C:₩Python35₩python.exe C:/Python/Algorithm/School/clustering/dbscan.py - input file name: data07.txt - input eps :20 - input min_sample :5 0.002000570297241211 seconds ---

▲ 표 7. 걸린 시간 측정 결과

현재는 데이터셋이 그렇게 큰 편이 아니기 때문에 오히려, 병합 클러스터링 < DBSCAN < K-means 클러스터링 순으로 걸린시간이 오래걸렸다. 사실 K-means 알고리즘은 가장 기본적인 알고리즘이면서 클러스터의 평균을 구하고나서 내부의 거리를 모두 측정해야하기 때문에 오래 걸리기도 하고, 클러스터의 개수에 따라서도 상당히 변화가 많은 알고리즘이라 클러스터의 수에 따라 성능과 결과가 극명하게 달라질 수도 있다. 이러한 점을 본다면 무조건 위의 결과가 나온다고 단정할 수가 없다는 것이다.

4. Java : weka

1) 사용 방법

우선 위에서도 말했듯이 사용방법은 2가지가 된다. 직접 API를 이용한 구현방법이 있고, 다른 방법은 GUI로 결과를 직관적으로 확인하는 방법이다. 유연하게 활용하는 방법은 API를 이용하면 되는 것이고, 데이터셋이 어떤 추세로 수행되느냐는 GUI라는 말이다.

우선 weka API를 이용해서 소스코드로 작성하는 방법을 알아보자.

```
import weka.clusterers.ClusterEvaluation;
import weka.clusterers.SimpleKMeans;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class Clustering {
    private final static String dataset = "<test>.arff";

    public static void main(String args[]) throws Exception{
        DataSource source = new DataSource(dataset);
        Instances data = source.getDataSet();
        SimpleKMeans model = new SimpleKMeans();
        ClusterEvaluation clsEval = new ClusterEvaluation();

        model.setNumClusters(5);
        model.buildClusterer(data);
        System.out.println(model);

        clsEval.setClusterer(model);
        clsEval.evaluateClusterer(data);
        System.out.println("# of clusters: " + clsEval.getNumClusters());

    }
}
```

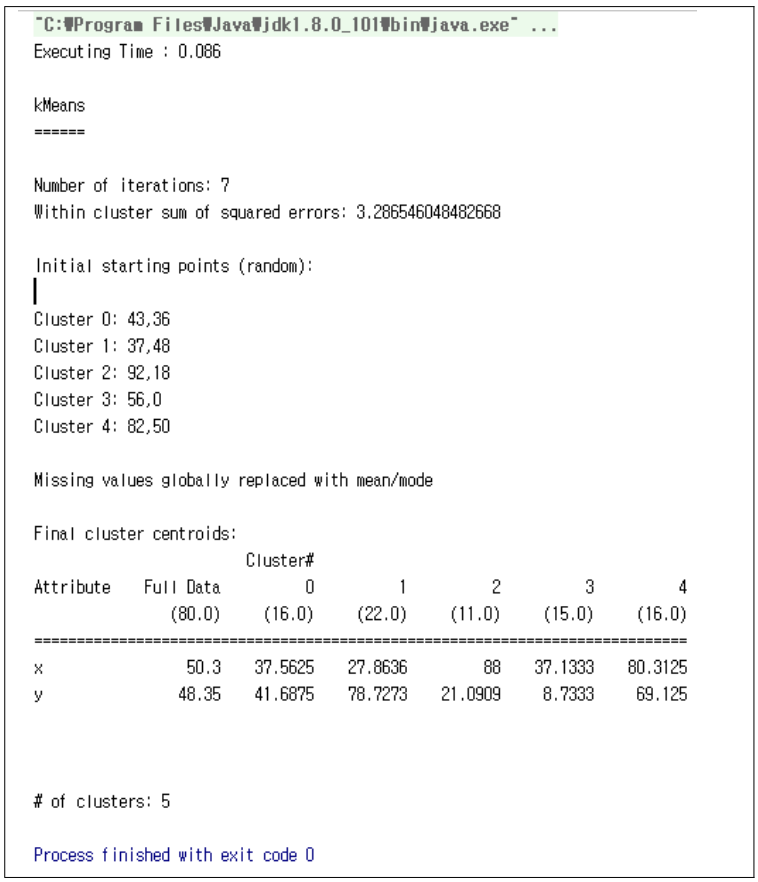
▲ 표 8. weka API 패키지로 구현한 K-means 클러스터링

1. 먼저 Datasource라는 클래스의 생성자로 arff확장자의 데이터를 넣어준다.
2. Instances로 데이터셋을 올린다.
3. SimpleKMeans 클래스를 생성한다. (K-means 클러스터 수행 클래스)
4. setNumClusters()라는 메서드를 이용해서 클러스터 개수를 지정
5. buildClusterer()라는 메서드를 통해 K-means 클러스터링 수행
6. ClusterEvaluation은 데이터를 출력하거나 평가할 때 사용하는 클래스(Python의 matplotlib과 비슷)

K-means 클러스터링은 위와 같은 방법으로 선언해주면 되고, 다른 알고리즘 또한 비슷한 과정을 거쳐서 사용하면 된다. 실제 구현 내용은 Python코드와 같이 올리도록 하겠다.

2) weka API를 이용한 수행 결과

- K-means

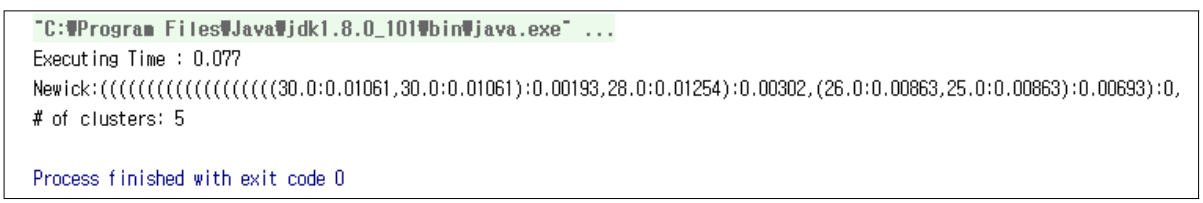


```
"C:₩Program Files₩Java₩jdk1.8.0_101₩bin₩java.exe" ...
Executing Time : 0.086

kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 3.286546048482668

Initial starting points (random):

Cluster 0: 43,36
Cluster 1: 37,48
Cluster 2: 92,18
Cluster 3: 56,0
Cluster 4: 82,50

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute    Full Data          0          1          2          3          4
                (80.0)     (16.0)     (22.0)     (11.0)     (15.0)     (16.0)
==============================================================================
x                50.3    37.5625    27.8636         88    37.1333    80.3125
y               48.35    41.6875    78.7273    21.0909     8.7333     69.125



# of clusters: 5

Process finished with exit code 0
```

▲ 그림 8. K-means 출력 결과 (n_clusters=5)

수행 시간은 0.086초가 나오며, GUI나 그래프를 이용해서 출력한 것이 아닌, Text 기반으로 출력을 했기 때문에 위와 같은 출력 결과가 나오게 되었다. Centroid의 좌표, 이터레이션 횟수 등의 수행 결과가 출력되는 형태다.

- Hierachical Clustering

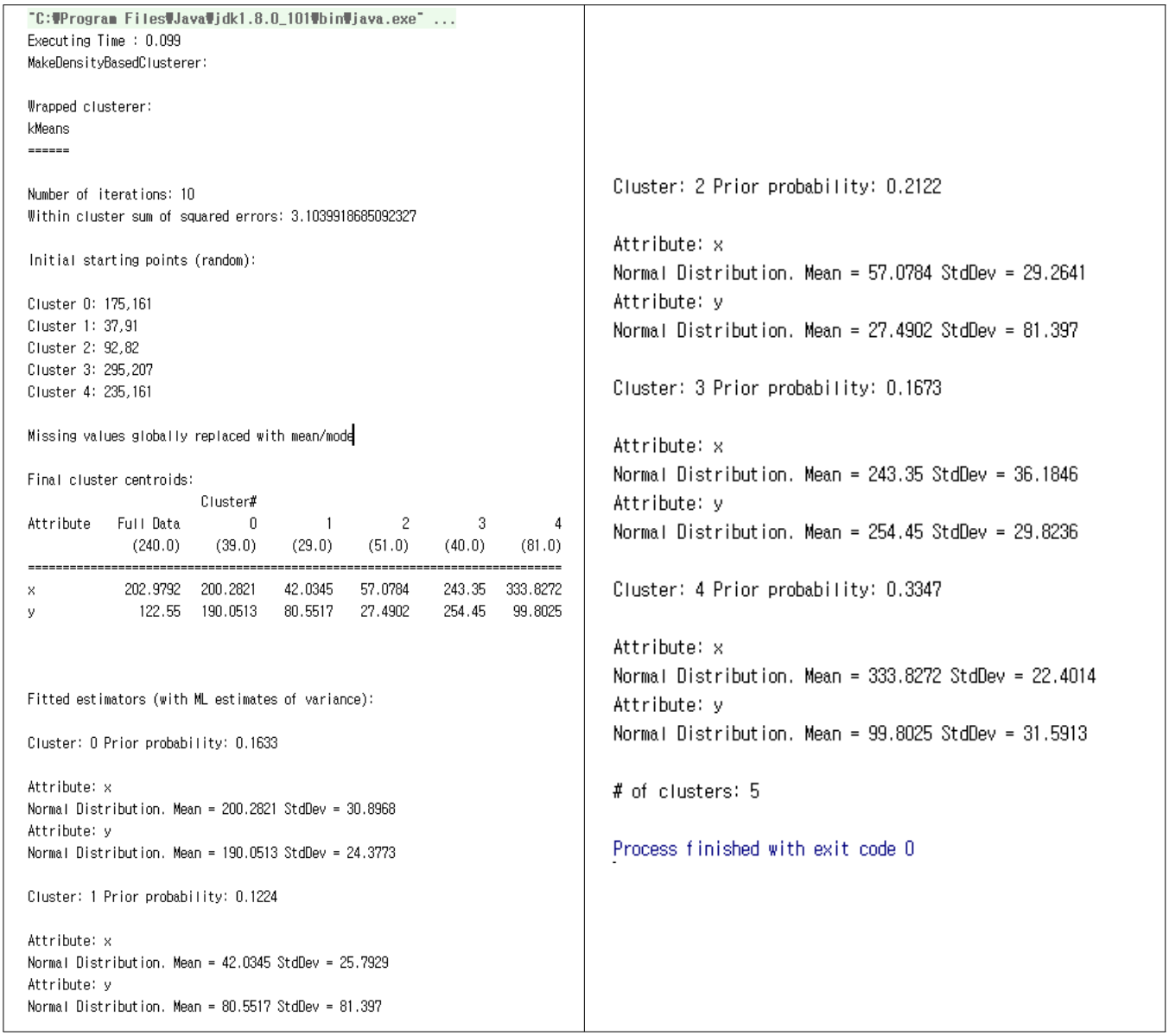


```
"C:₩Program Files₩Java₩jdk1.8.0_101₩bin₩java.exe" ...
Executing Time : 0.077
Newick:((((((((((((((((((((30.0:0.01061,30.0:0.01061):0.00193,28.0:0.01254):0.00302,(26.0:0.00863,25.0:0.00863):0.00693):0,
# of clusters: 5

Process finished with exit code 0
```

▲ 그림 9. Agglomerative Hierachical Clustering 출력 결과 (n_clusters=5)

HierarchicalClusterer라는 클래스를 통해 지원하며, 출력결과는 위와 같았다. Python에선 Dendrogram를 출력하기 위해서 matplotlib를 이용하면 됐지만, java의 weka는 그냥 그대로 출력해버린다. 수행 시간은 0.077초가 나왔다.

- Density Based Cluster



```
"C:₩Program Files₩Java₩jdk1.8.0_101₩bin₩java.exe" ...
Executing Time : 0.099
MakeDensityBasedClusterer:

Wrapped clusterer:
kMeans
======

Number of iterations: 10
Within cluster sum of squared errors: 3.1039918685092327

Initial starting points (random):

Cluster 0: 175,161
Cluster 1: 37,91
Cluster 2: 92,82
Cluster 3: 295,207
Cluster 4: 235,161

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute    Full Data          0          1          2          3          4
              (240.0)     (39.0)     (29.0)     (51.0)     (40.0)     (81.0)
=============================================================================
x            202.9792   200.2821    42.0345    57.0784   243.35     333.8272
y            122.55     190.0513    80.5517    27.4902   254.45      99.8025



Fitted estimators (with ML estimates of variance):

Cluster: 0 Prior probability: 0.1633

Attribute: x
Normal Distribution. Mean = 200.2821 StdDev = 30.8968
Attribute: y
Normal Distribution. Mean = 190.0513 StdDev = 24.3773

Cluster: 1 Prior probability: 0.1224

Attribute: x
Normal Distribution. Mean = 42.0345 StdDev = 25.7929
Attribute: y
Normal Distribution. Mean = 80.5517 StdDev = 81.397

Cluster: 2 Prior probability: 0.2122

Attribute: x
Normal Distribution. Mean = 57.0784 StdDev = 29.2641
Attribute: y
Normal Distribution. Mean = 27.4902 StdDev = 81.397

Cluster: 3 Prior probability: 0.1673

Attribute: x
Normal Distribution. Mean = 243.35 StdDev = 36.1846
Attribute: y
Normal Distribution. Mean = 254.45 StdDev = 29.8236

Cluster: 4 Prior probability: 0.3347

Attribute: x
Normal Distribution. Mean = 333.8272 StdDev = 22.4014
Attribute: y
Normal Distribution. Mean = 99.8025 StdDev = 31.5913

# of clusters: 5

Process finished with exit code 0
```

▲ 그림 10. Density Based Cluster 출력 결과 (n_clusters=5)

밀도 위주의 클러스터링인데, DBSCAN과는 다르게 클러스터의 개수를 지정해야한다. 그리고 최소 밀도를 지정한 후 함수를 수행해서 출력하면 위와 같이 나온다. 수행시간은 0.099초다.

```
DataSource source = new DataSource(dataset);
Instances data = source.getDataSet();
MakeDensityBasedClusterer model = new MakeDensityBasedClusterer();
ClusterEvaluation clsEval = new ClusterEvaluation();

model.setMinStdDev(20);
model.setNumClusters(5);
long start = System.currentTimeMillis();
model.buildClusterer(data);
long end = System.currentTimeMillis();
System.out.println( "Executing Time : " + ( end - start )/1000.0 );
```

▲ 그림 11. Density Based Cluster 소스코드 일부

전체적인 수행시간 결과를 보면 아래와 같다.

K-means Clustering	0.086 sec
Hierachical Clustering	0.077 sec
Density Based Cluster	0.099 sec

▲ 표 8. 걸린 시간 측정 결과

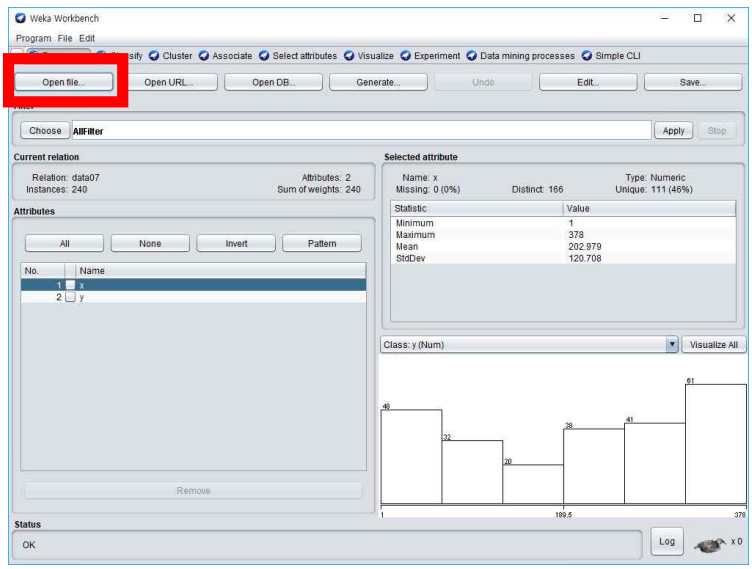
이것도 사실상 Python의 sklearn과 비슷한 결과가 나왔고, Density Based Cluster가 오히려 더 느리게 나왔다는 점이 차이가 있다.

3) GUI를 이용한 수행 결과

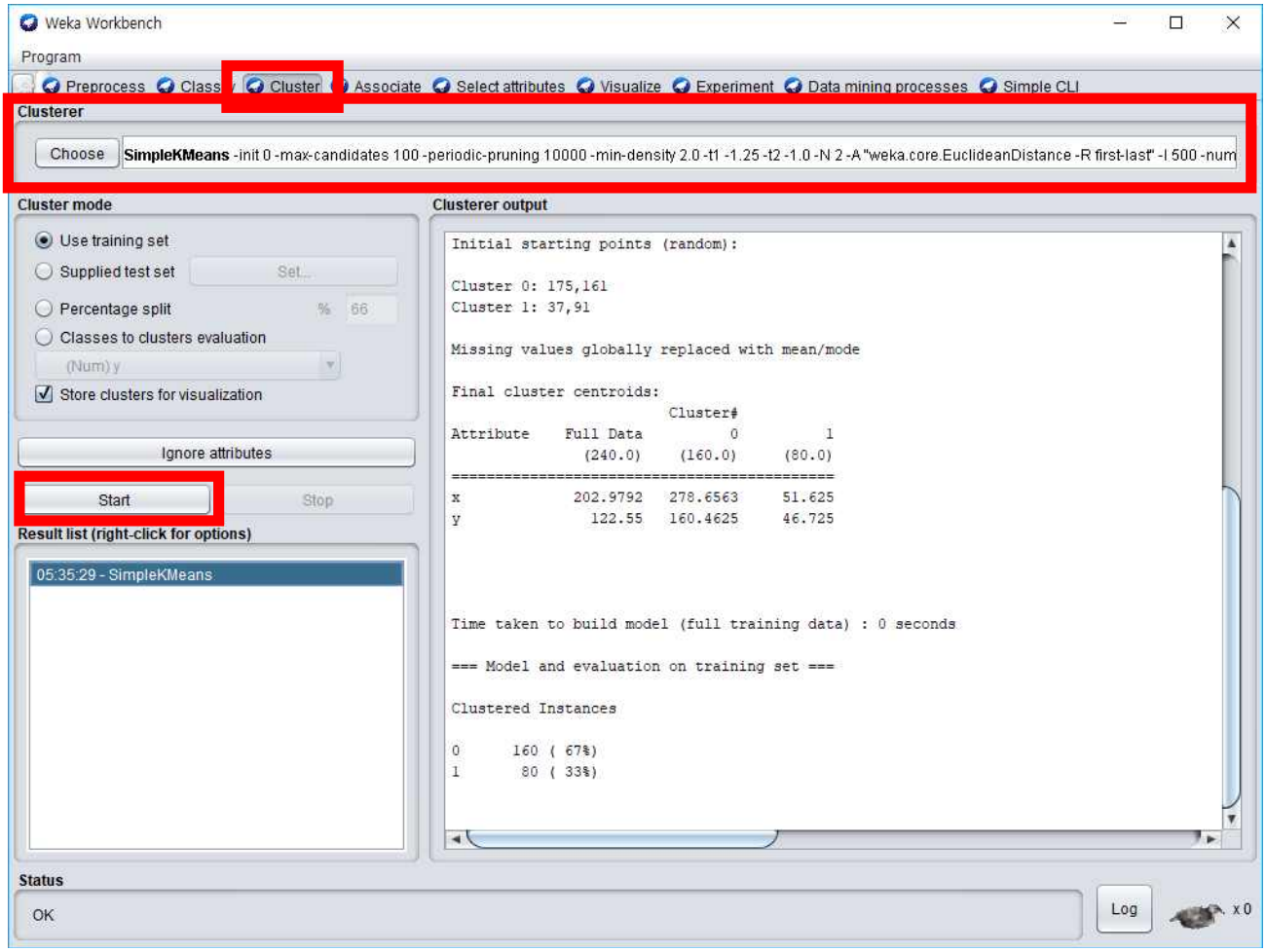
우선 사용방법에 대해서 알아보자.

1. Weka GUI Chooser에서 Workbench로 이동하자.

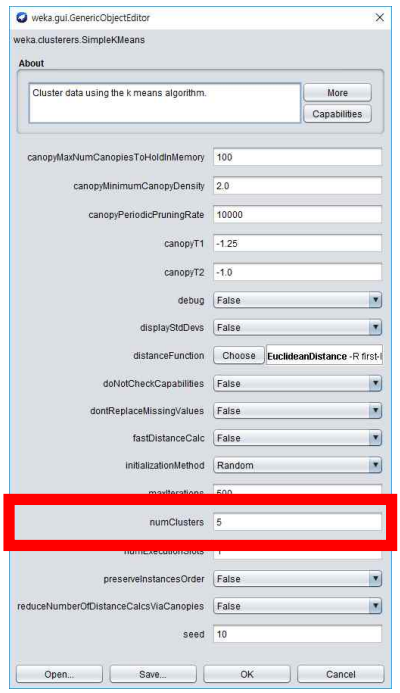
2. Workbench에서 데이터셋(.arff 확장자 파일)을 넣어주자.



3. 상단의 Cluster 카테고리로 이동 -> 원하는 클러스터 알고리즘 선택 -> start

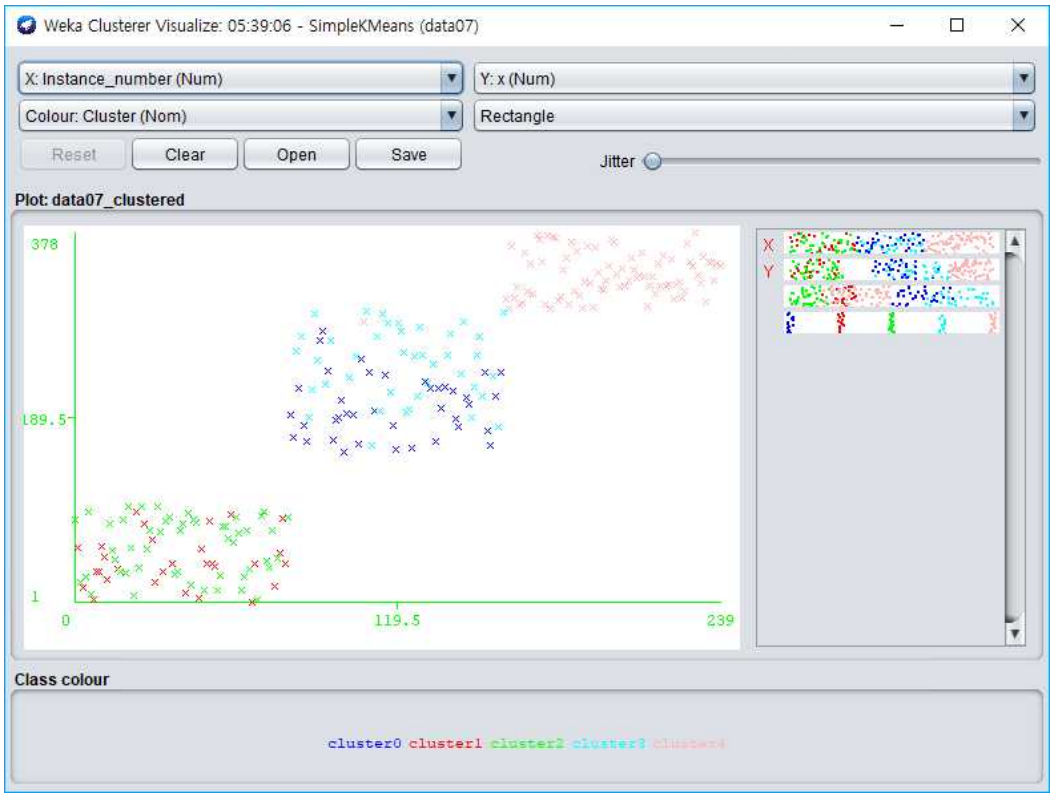


4. 만약 매개변수를 수정하고 싶다면, Choose 버튼 오른쪽의 클러스터 내용을 클릭하면 아래와 같이 뜬다. 매개변수를 자기가 원하는대로 선택하면 된다. (필자는 Cluster 개수를 5개로 변경하였다.)



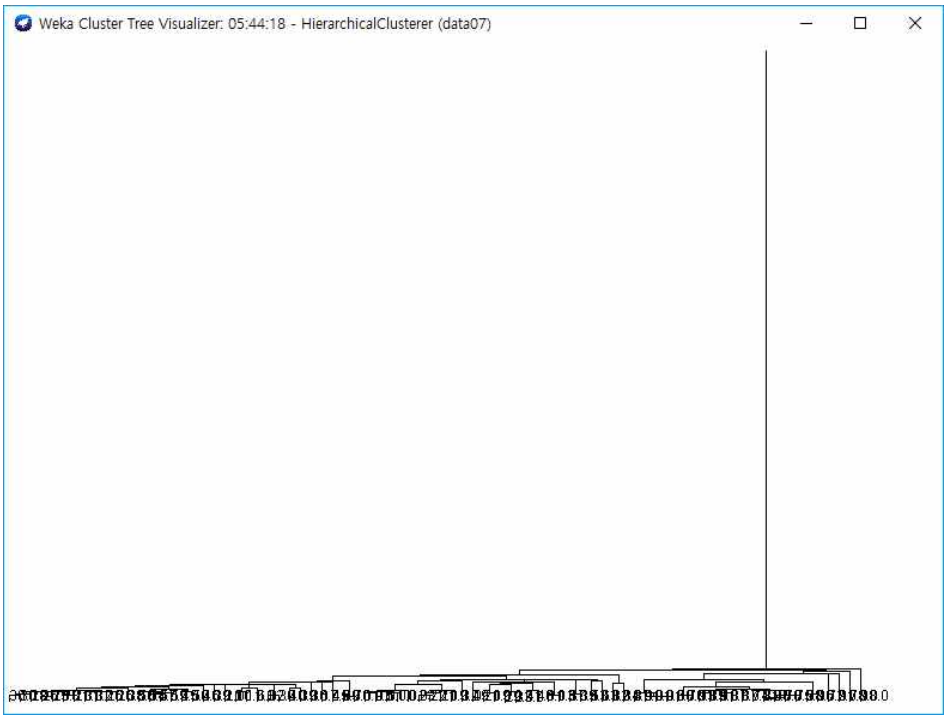
5. 좌측 하단의 Result list에서 아이템에 우클릭 -> Visualize Cluster assignments 클릭

이것을 기준으로 3가지 알고리즘의 결과를 출력해보자.



▲ 그림 12. K-means 그래프 결과

▲ 그림 13. Hierachical Cluster 그래프 결과



▲ 그림 14. Hierachical Cluster 트리 결과

▲ 그림 15. Density Based Cluster 트리 결과

5. 결론

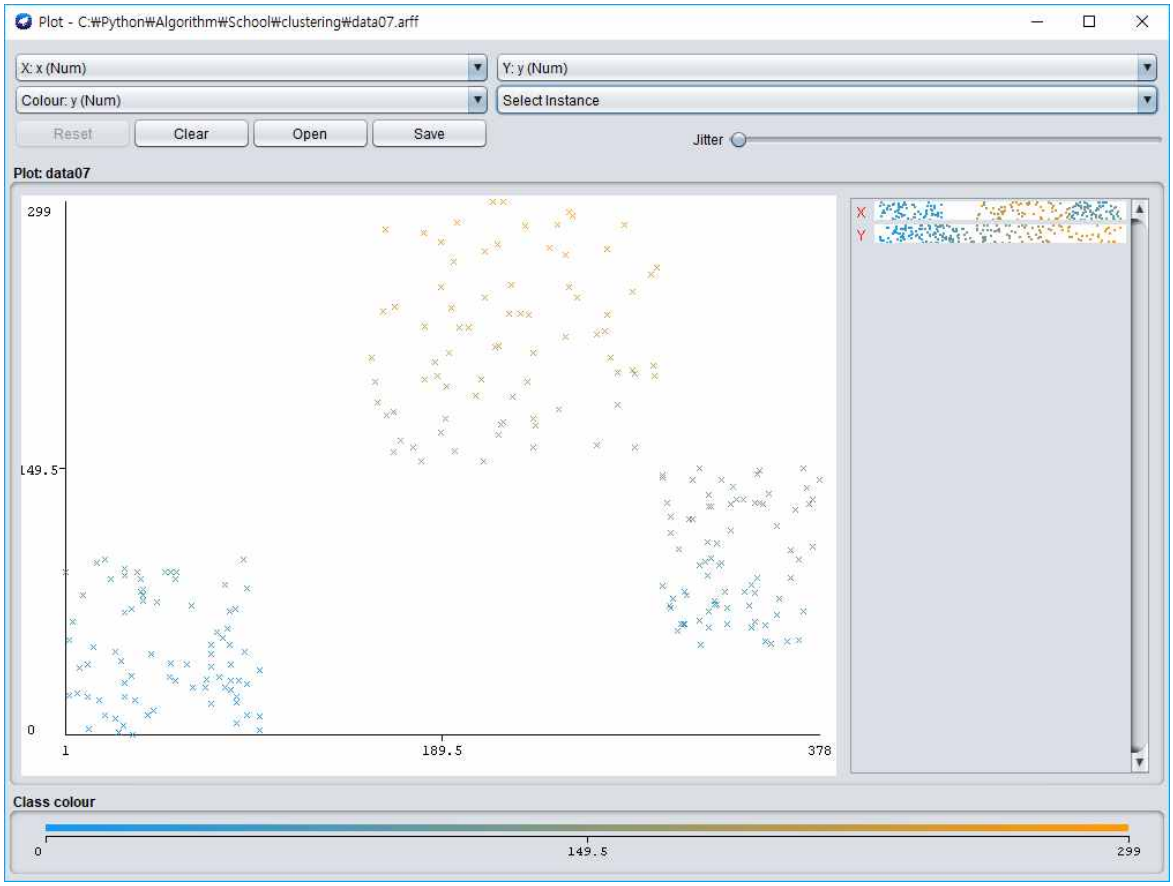
Python의 scikits-learn 라이브러리와 Java의 Weka 패키지를 활용해본 결과, 약간의 차이는 있는데, 아마도 Default 옵션의 차이가 다소 다르게 출력되게 된 원인이 아닐까 싶다. (실제로 Weka GUI를 봤을 때, 매개변수가 굉장히 많았다.) 하나의 데이터 셋으로 돌려가면서 썼기 때문에 완전히 이 결과가 정확하다라곤 할 수는 없을 것 같긴 하지만, 실제로 제공되고 많은 사람들이 사용하는 라이브러리이기 때문에 이를 직접 활용해보고, 원리를 파악하는 과정에 의의를 두는 것이 중요하다고 본다.

Python의 경우는 오히려 sklearn 관련해서 도와주는 라이브러리가 있기 때문에 더 사용하기가 편했고, java의 경우는 오히려 arff파일을 만들어서 데이터 셋으로 활용해야 했기 때문에 다소 불편하다고 느꼈다. 하지만, 둘 다 개별적으로 장점이 있을 것이고, 사람들이 많이 사용하는 데에 이유가 있다고 생각한다.

위의 라이브러리를 이용하면서 3가지 알고리즘을 사용해봤다. K-means의 경우는 가장 이해하기 쉬운 알고리즘이였는데, 그 이유는 실제로 centroid와의 거리를 위주로 계산을 하면서 centroid의 위치를 조정하는 방식이라서 그랬다. Hierachical Cluster는 근처에 있는 데이터들을 묶고 또 묶는 방식으로 쓰기 때문에 계층화가 이루어지는 방식이므로 Dendrogram이나 계층화 그래프를 같이 보는 것이 이해하기가 더 쉬웠다. DBSCAN의 경우는 오히려 k-means에서 클러스터의 개수에 영향을 받던 문제점을 없앴기 때문에 느리더라도 정확하게 묶어준다는 게 가장 큰 장점으로 이해했고, 실제로 centroid의 개수를 조정하는 것보다 더 편하고 정확하게 접근할 수 있다는 것이 장점이였다.

부록. 참고자료

A. 테스트에 사용한 데이터셋 시각화 (data07.txt)



▲ 그림 16. Weka를 이용한 데이터 셋의 시각화

B. Reference

[1] 안드레아스 뮐러, 세라 가이도, 「Introduction to Machine Learning with Python」, 한빛 미디어 p210 ~ 255 - 비지도 학습 : 군집

[2] Weka Package Reference - Package weka.clusterers
http://weka.sourceforge.net/doc.dev/weka/clusterers/package-summary.html

[3] Scikit-learn Document - sklearn.cluster
http://scikit-learn.org/stable/modules/classes.html

C. 본인의 Github (소스코드 원격저장소)

https://github.com/KeonHeeLee/clustering