Data Structure and Algorithms
16:332:573

Project Report: Analysis of Basic Clustering Algorithms

Song Yang
Xin Yang
Yi Wu
Zhuohang Li

May 2018

Electrical and Computer Engineering Department
Rutgers University, Piscataway, NJ 08854

Abstract

This project mainly researched the differences between four clustering algorithms. We chose the K-means, BIRCH, DBSCAN, Spectral Clustering as the representatives of Partition-based Clustering, Hierarchical clustering, Density-based clustering and Graph-based clustering algorithms to perform the analyses and comparisons.

1 Motivation

Clustering is the task of grouping a set of objects in such a way that objects in the same group. It is widely used in many fields, including pattern recognition, machine learning, and computer graphics, e.g., advertisement companies use clustering algorithms to find the preference of users and precisely push the advertisements, medical fields also implement it to help analyze medical images.

We choose four of the representative clustering algorithms, and analysis them in aspects of key technology, advantages, and disadvantages We use some typical and famous dataset to evaluate the algorithms in performance, efficiency and clustering results.

2 Algorithms Description

2.1 K-means

K-means clustering algorithm is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. It aims to divide a group of samples into K different categories, which is called unsupervised learning in machine learning. K is artificially given.

The basis of K-means is to calculate the minimum square error. Here is the cost function:

$$J(c,u) = \sum_{i=1}^{k} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

In the formula above, $\mu_{c^{(i)}}$ is the mean of points in cluster i. The more similar the samples in each category are, the smaller the J is. When the value of J is less than a certain number, we can say the cost function is convergence.

2.2 DBSCAN

DBSCAN which is short for Density-Based Spatial Clustering of Applications with Noise is a clustering algorithm based on density. Given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). [1] [2]

2.3 Spectral Clustering

The spectral clustering has little requirements for the datasets despite other algorithms like K-means in that it is based on the graph theories. The spectral clustering will consider the given dataset(containing N elements) as an N×N edge-weighted graph D, which is a diagonal matrix and the weight is the degree of the vertex.

Then we will have the affinity matrix W, which is the adjacency matrix and can be done in several ways. Here we will use KNN for the convenience.

Having these two matrices, we can calculate the graph Laplacian L, which can be computed by $L = D - W$, and the regularized Laplacian matrix $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$.

By implementing traditional clustering algorithms such as K-means on the normalized eigenmatrix formed by k eigenvectors corresponding to the first k eigenvalues, we can get the results of a basic spectral clustering.

The problem of clustering is turned into a graph dividing issue, in which we need to divide the graph into several subgraphs, and the accumulation inside a subgraph should be as high as possible, meanwhile, the weights between subgraphs should be low.

The spectral clustering algorithm is not sensitive to the shape of data, as well as the input order. Also the Laplacian matrix help to reduce the dimension and meanwhile keep the similarity relationships between vertexes.

Also, the convergence to the global optimum is guaranteed for its nature. The robustness plus the efficiency of implementation, the spectral clustering is a popular clustering method for scientific and engineering applications.

2.4 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a hierarchical clustering algorithm. This algorithm is based on a tree structure called CF (Clustering Feature) tree to implement multiphase clustering. Every node in CF tree consist of several Clustering Features. CF vector is given by

$$\text{CF=(N, LS, SS)}$$

where N is the number of data points, LS is the linear sum of the N points, SS is the square sum of the N points. The algorithm can be mainly described as two phases:

- Phase 1: Scan the dataset to build an initial CF tree which aims to preserve the inherent clustering structure of the data.
- Phase 2: Use an arbitrary clustering algorithm to cluster the leaf nodes of the CF tree.

The key idea of BIRCH is to use multi-level clustering to get better efficiency. Micro-clustering on lower level can reduce complexity and increase scalability, at the same time, also provide enough flexibility for high level clustering that operates on these lower level structures. [3]

3 Implementation

3.1 Dataset

For the low dimensional data, we use the most famous iris dataset [4]. For the 13 dimensional data, which is to show the performance on high dimensional conditions, we utilize the wine dataset [5].

3.2 Indicators

Since currently there is no effective way to evaluate the clustering algorithms, here we choose several indicators to reflect the performances in certain aspects.

Silhouette Coefficient

Silhouette Coefficient can be calculated by this formula:

$$s = \frac{b-a}{max(a,b)}$$

For each sample in a category, a is it's average distance to other samples, b is it's average distance to the samples in the nearest category. We calculate the Silhouette Coefficient of all sample to get the final result.

Adjusted Rand index, Adjusted Mutual Information based scores

The adjusted rand index and the adjusted mutual information based scores both range from -1 to 1, and the bigger value indicate the more consistent the predicted result is with the real value.

V-measure

The V-measure is the harmonic mean of the homogeneity and completeness. Homogeneity infers each cluster contains only one class of elements. Completeness reflects that all elements in a given classification are gathered into one cluster. The result of V-measure should be as high as possible and no larger than 1.

Calinski-Harabaz Index

Calinski-Harabaz Index can be calculated using the formula below:

$$s(k) = \frac{tr(B_k) * (m-k)}{tr(W_k) * (k-1)}$$

m is the number of data, B_k is the covariance matrix between categories, W_k is the covariance matrix between data in one category. Function tr is to calculate the trace of a matrix. The larger the Calinski-Harabaz Index is, the better performance the algorithm has.

3.3 K-means

The algorithm works in an iterative way described as follows:

1. Choose k arbitrary points as centroids.
2. For each sample i, assign it to it's nearest centroids.

$$c^{(i)} = \arg\min_{J} \|x^{(i)} - \mu_j\|^2$$

3. For each category j, recalculate the centroids.

$$\mu_j = \frac{\sum_{i=1}^{m}\{c^{(i)} = j\}x^{(i)}}{\sum_{i=1}^{m}\{c^{(i)} = j\}}$$

4. Reassign every data point to its nearest centroid.

5. Iterate from step 2 until no point is reassigned in step 4.

The pseudocode can be described as follows:

```
function K-Means(input data, K):
        get the dimension of the data(dim) and the size of it(N)
        randomly generate K Dim-dimensional points
        while(not convergence):
                for each point:find which category it belongs to
                for each centroid:
                        find all points which belongs to it's category
                        recalculate the centroid
                end
        print(result)
end
```

3.4 DBSCAN

According to what we described in 2.2 algorithm requires two parameters, Epsilon(eps) and Minimum Points(MinPts)

1. Start from a point. Search for the cluster though searching the eps neighborhood of each point. If the neighborhood of the point contains sufficiently many points more than MinPts, then create a cluster surrounding the point as the core.

2. Then, recursively gather these core points and points belong to its neighborhoods. (Operations like combining neighborhood points will be made in this step)

3. Stop clustering when there is no new point can add to any clusters.

The theory of DBSCAN is not that hard, but to optimize the algorithm, using some kind of advanced tree data structure

3.5 Spectral Clustering

The implementation of spectral clustering is based on the open-source library Scikit-Learn [6]. We first run our experiments on the Iris dataset [4] to compare the performances with different configurations, i.e., the way the affinity matrix is generated and the parameters that would affect the results. Here we choose three ways to generate the adjacency matrix, the KNN(Kth Nearest Neighbors) method, the polynomial method and the Radial Basis Function(RBF). Then we try to compare spectral clustering on different dimensions of data and with other algorithms using the wine dataset [5]. The default value of γ is 1 if not clarified, and the default degree in polynomial approach is 3.

3.6 BIRCH

The key process of BIRCH algorithm is to build the in-memory CF tree, which is to incrementally insert new points into the tree. A CF tree has similar structure as a B+ tree. It is a height balanced tree which stores the CF features. The non-leaf nodes in the tree store the sum of the CFs of their children. A CF tree has two parameters: branching factor which is maximum number of children and maximum diameter of sub-clusters stored at the leaf nodes. The insertion process can be described as follow:

```
FOR each point in the input:
        find its closest leaf entry
        add point to the leaf entry and update CF
        IF entry diameter > maximum diameter:
                split the leaf and possibly the parent
```

4 Results and Analysis

4.1 K-means

Results

Before clustering:

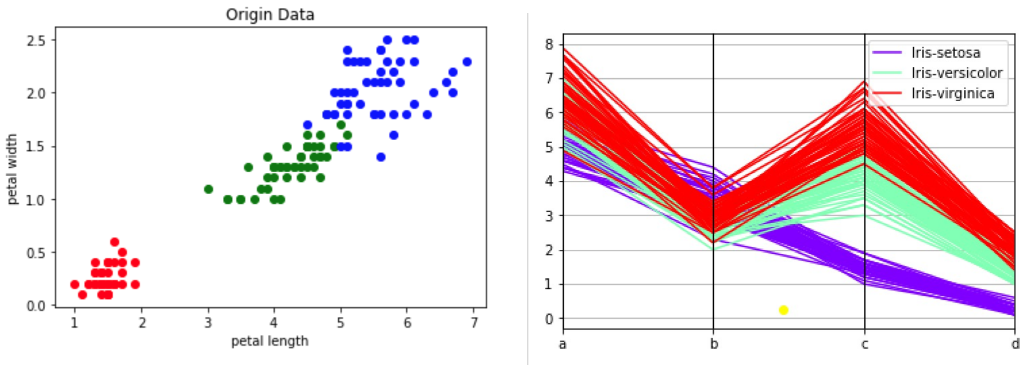


Figure 1: Origin Dataset

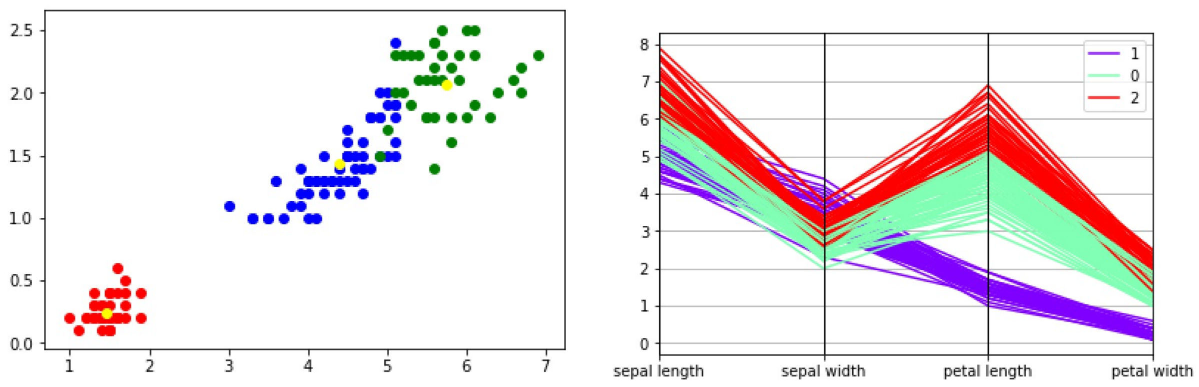


Figure 2: After clustering(k-means++):

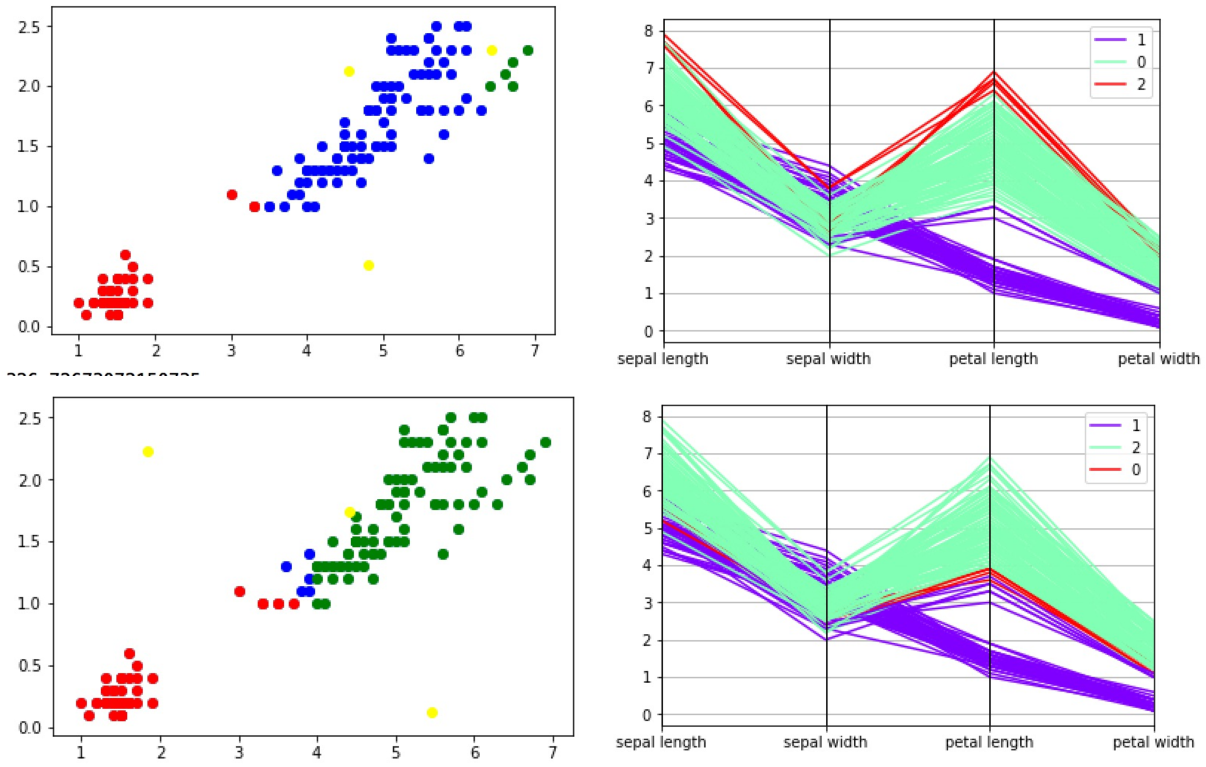


Figure 3: After clustering(original k-means):

We also use a 13-dimension data, and the result is following:

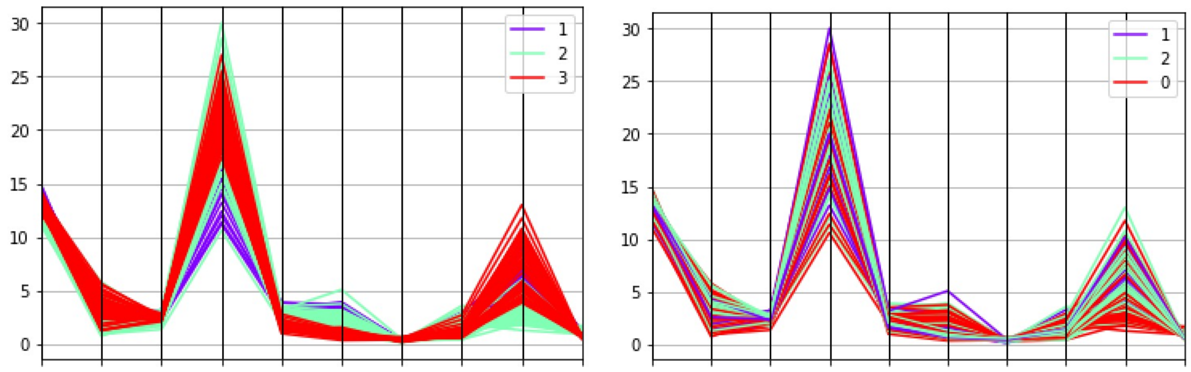


Figure 4: 13-dimension data

It's a pity that $K - means^{++}$ can not perform well on such a high-dimensional data.

Analysis

Time Complexity

The time complexity of k-means is $O(m * n * k * d)$. We can see that we need to calculate the distance of n points to k centroids, and we need to iterate m times. For a d-dimension data, the time needs to calculate the distance is proportional to d. So just simply multiply them together, we can get the time complexity.

Optimize

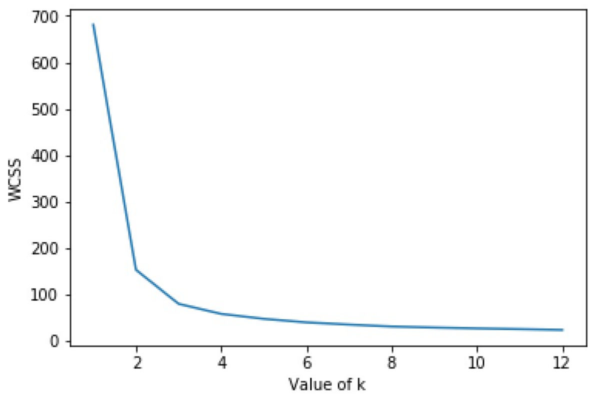
The original K-means algorithm has a critical disadvantage: the number of categories K must be decided by users, and the centroids are all randomly decided. As we can see from the above, the results of the original k-means have a lot difference. Centroids are different while categories are irregular. To prevent this happen, get a more accurate consequence, we can improve it by using $K - means^{++}$ algorithm.

The implementation of $K - means^{++}$ can be described as follows:

1. Choose one arbitrary point as centroids.
2. For each point x in the data set, calculate it's distance to the nearest centroid $D(x)$.
3. Choose one new point at random as a new centroid, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$.
4. Repeat Steps 2 and 3 until k centroids have been chosen.
5. Implement the original K-means algorithm.

Ensuring centroids will need k times' iteration, and for each time, we need to calculate all points' distance to the centroid. So the time complexity is $O(n * k * d)$, and will not influence the total complexity.

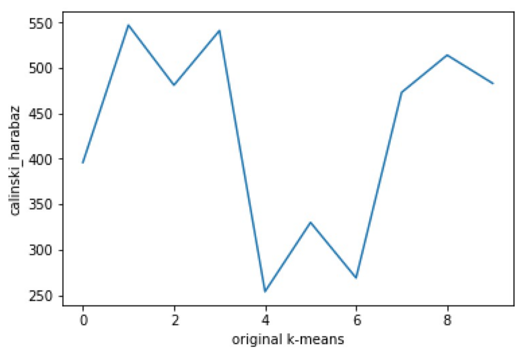
As for determining the number of k, we can use the following method: we may let k equals to $1, 2, 3 ... \sqrt[2]{n}$. For each k, we implement the $K - means^{++}$ on the dataset and calculate the average within-cluster sum of squares(WCSS) for each category. The result can be shown in the following figure:

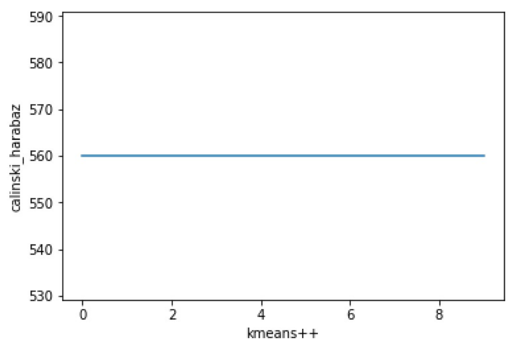


As we can see, after k is bigger than 3, the within-cluster sum of squares (WCSS) doesn't decrease significantly with every iteration. So we can ensure that in this case, k=3. Adding this step will cause the total time complexity increase to $O(m * n^2 * d)$, but it's worth because getting the correct value of k is very important.

Evaluation Indicators

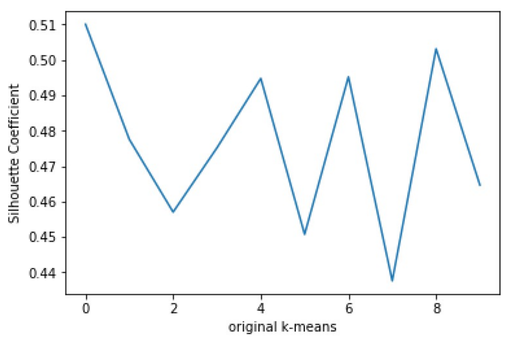
Implement both $k - means$ and $k - means^{++}$ ten times, we can get the following result of Calinski-Harabaz Index:

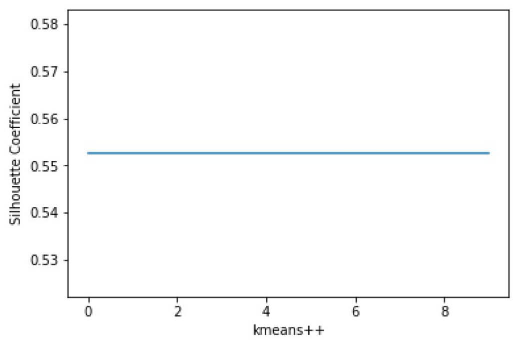




The original $k - means$ can only get a average of 429, while $k - means^{++}$ maintains a constant value of 560. Which algorithm performs better is obvious.

Also, we can get the result of Silhouette Coefficient:





Similar to Calinski-Harabaz, we can see that $k - means^{++}$ performs much better. Using other indicators, we can form a table:

Evaluation Indicators	**Original Kmeans**	**Kmeans++**
Adjusted Rand Index	0.62976264	0.73023827
Mutual Information scores	0.67618911	0.74837239
V-measure	0.72882873	0.75817568
Silhouette Coefficient	0.48473001	0.55259194
Calinski-Harabaz Index	429.04301650	560.39992424

Table 1: Evaluation Indicators Comparison of Kmeans and Kmeans++

Evaluation Indicators	**Kmeans++**
Adjusted Rand Index	0.11841356
Mutual Information scores	0.10019421
V-measure	0.11391062
Silhouette Coefficient	0.42739963
Calinski-Harabaz Index	241.13589012

Table 2: Evaluation Indicators of Kmeans++ on the 13-dim data

From the indicators, we can also see that even $k - means^{++}$ can not perform well on high-dimensional data.

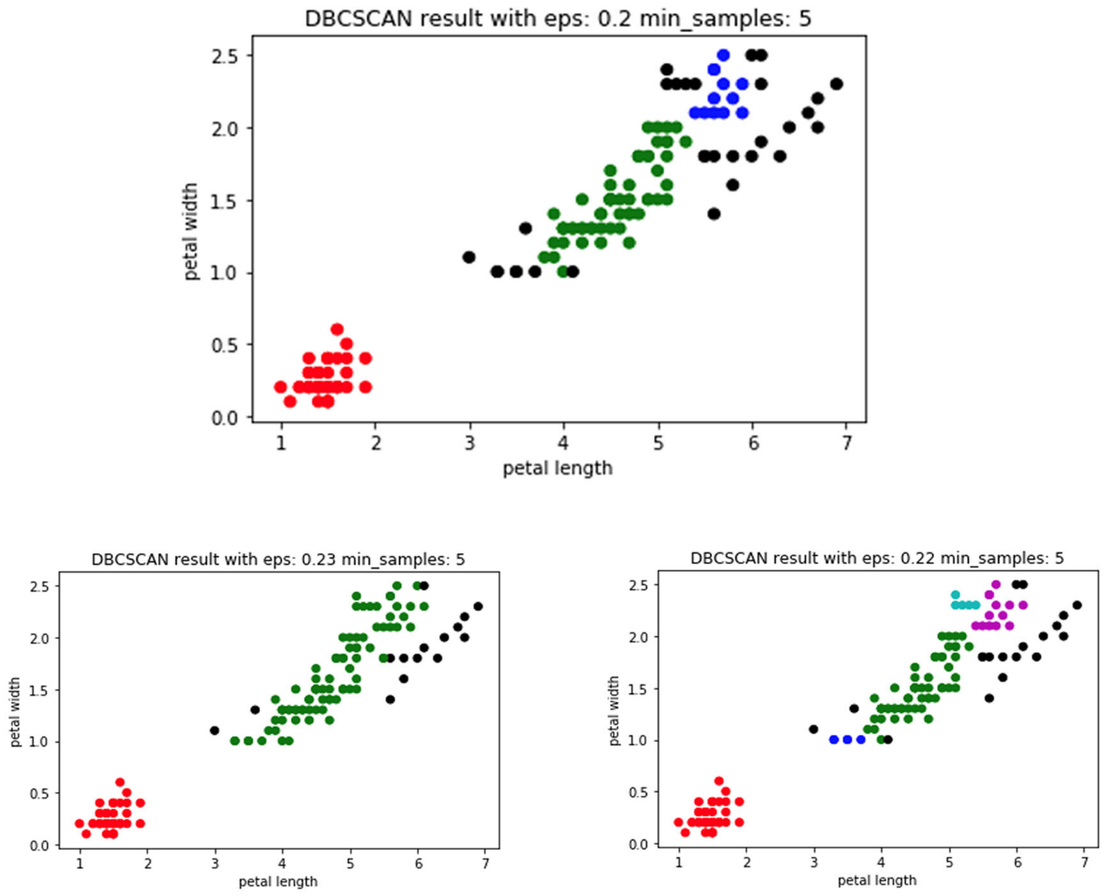


Figure 5: DBSCAN w/eps=0.2, 0.22, 0.23

4.2 DBSCAN

DBSCAN is a density-based algorithm, the result may differ when using different dataset. According to the theory and the procedure of DBSCAN, we have some conjectures and made implementations to verify them.

Distribution Conjecture

DBSCAN divide the data into clusters according to the point density. If the density of the data is uneven, what kind of result would DBSCAN returns? Our conjecture is that is should be hard to get a novel classification result when data are given in an uneven distribution. If the density of the data changes in a huge range, the DBSCAN is not that accurate to describe data with labels.

For the experimental setup, we use an uneven density dataset, whose distribution can be seen in the Figure 1. Red, green, blue points separately indicate Iris-setosa, Iris-versicolor, Iris-virginica. Intuitively, the red and green points have a similar density. As for blue points, the distribution is very scattered and have some intersection. It's should be hard to classify between the green and the blue according to the DBSCAN algorithm that it has to find spots next to each other.

The classification result shows as the Figure 5. The result of Iris-setosa is pretty well as expected. Although we tried to adjust the parameters, I think with the parameter of 0.2 and 5, it should be nearly the best result for the DBSCAN. When

Evaluation Indicators	DBSCAN w/2d	DBSCAN w/4d	DBSCAN w/13d
Adjusted Rand Index	0.58587262	0.67891690	0.29274711
Mutual Information scores	0.55726682	0.62396962	0.37305933
V-measure	0.64408265	0.69883015	0.39178824
Silhouette Coefficient	0.73086366	0.53018854	0.44951534
Calinski-Harabaz Index	538.90619072	204.26821329	239.25421075

Table 3: Evaluation Indicators Comparison

eps equal to 0.22, the originally blue parts are split into many parts, and Half of them are recognized as noise. When trying to add little to eps, the green and blue parts are totally mixed up which is overfitting. As what we get DBSCAN cannot classify the Iris-virginica with a clear boundary in this case.

Curse of Dimensionality Conjecture

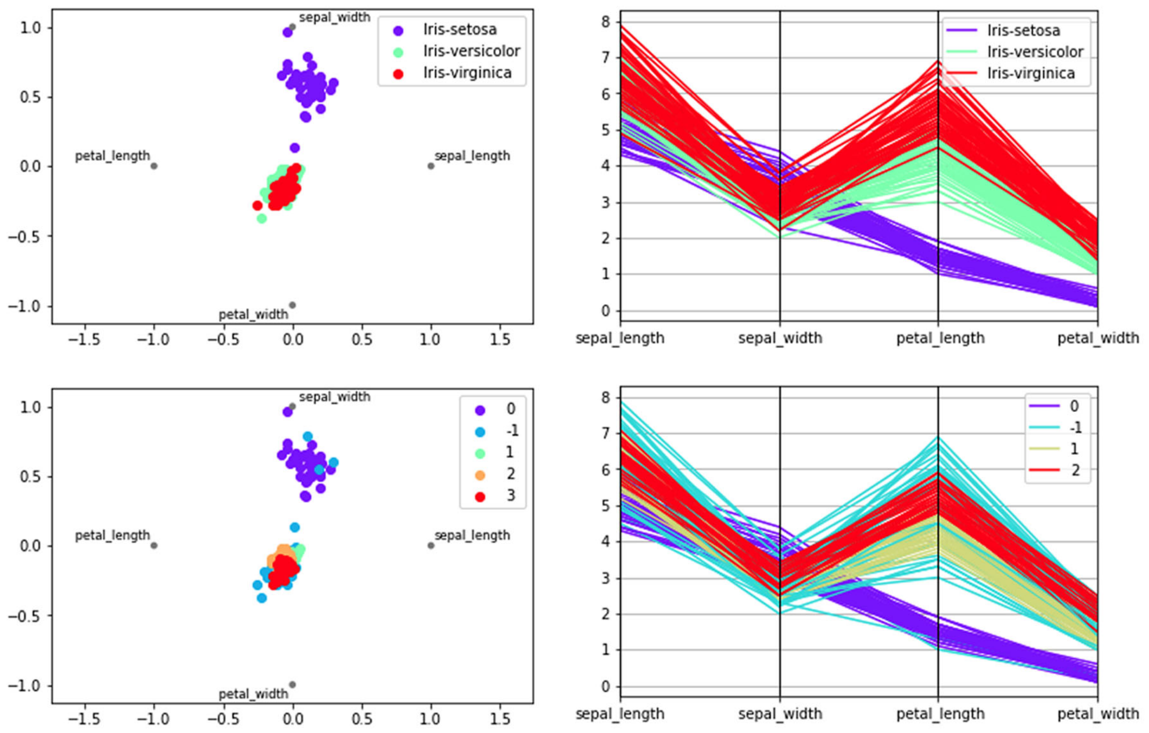
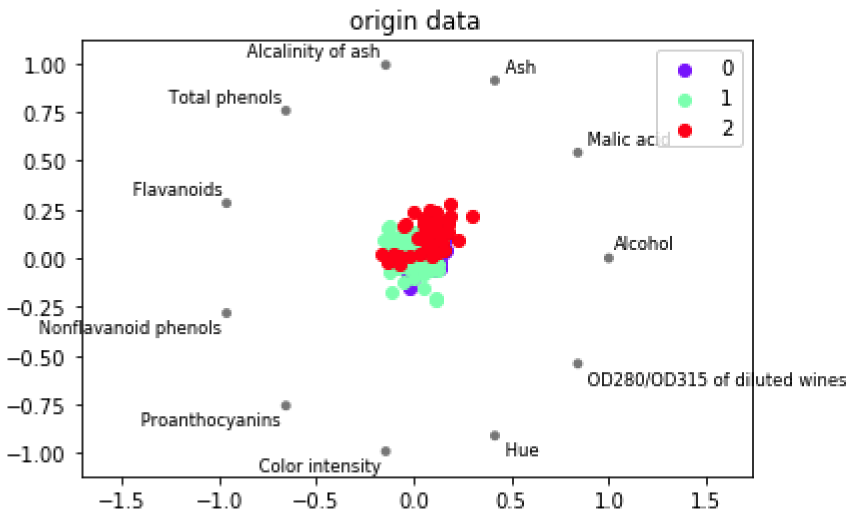


Figure 6: 4-Dimension Features DBSCAN w/eps=0.65

The increasing number of the dimension may take the Curse of Dimensionality, which results in the model becomes over-fitting. But we do not the how many dimensions of data DBSCAN can handle with. So in Iris dataset, we tried to use all four dimension data, we can see the result after setting parameters in Figure 6. The class that names '-1' in the result indicate the noise class. The DBSCAN works fine in data with four features, besides two of them have an uneven density distribution.



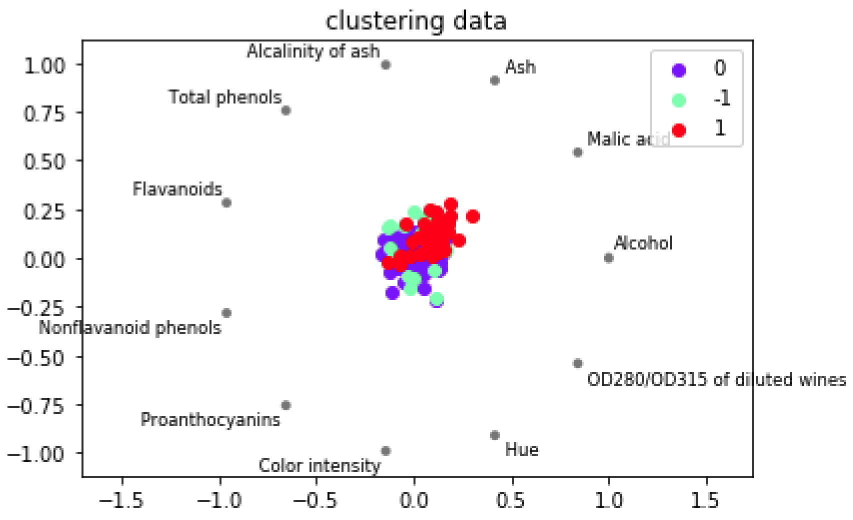


Figure 7: 4-Dimension Features DBSCAN w/eps=0.65

I also tried to cluster a dataset with 11 features, the result is pretty bad, showed in the Figure 7. As for this dataset, DBSCAN can only get one classification, the rest of two classes are totally mixed up since their densities are similar.

4.3 Spectral Clustering

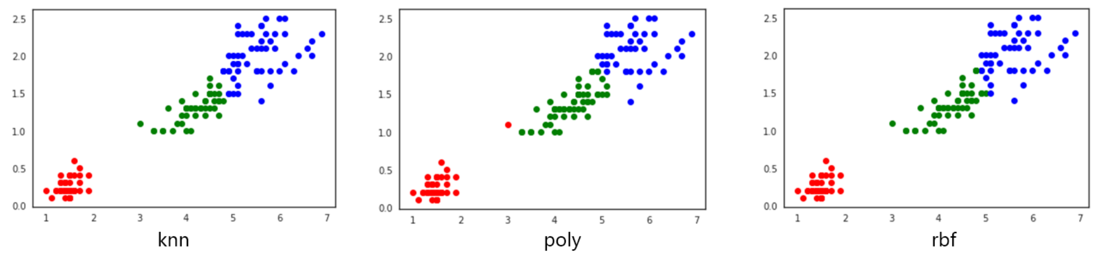
The principle of the spectral clustering algorithm is based on the graph theory and can be considered as an optimal graph cutting problem, thus the input order of the dataset will not change the result. Since the most time consuming operations are the calculation of eigenvectors [7], standard approach would cost $O(n^3)$ time complexity, but we can simplify this trouble by using a matrix-vector multiplication since the N cut theory only demands one eigenvector. The time complexity of spectral clustering is $O(N)$.

As we can preset the number of clusters, here we assume we already know there will be three clusters. As can be seen from the Table 4, which was generated by clustering the petal length and the petal width, the results of spectral clustering using KNN is slightly closer to the reality than the RBF approach, but from the Calinski-Harabaz Index, we can see that the distances between all vertexes in one cluster are shorter than the KNN ones, which means RBF can better classify than KNN by distance relationship. On two dimensional data, the KNN and RBF approach both have a relatively ideal result, while the polynomial kernel function approach has the lowest Calinski-Harabaz Index, which can be proved by the Figure4.3 that it can not distinguish the special noise points well.

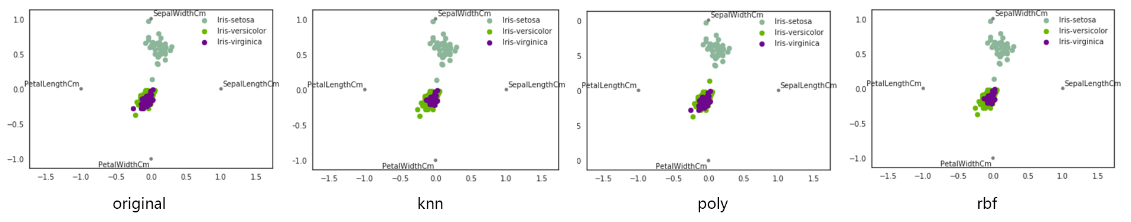
Due to the nature of radial basis function, the farther vertex will have smaller weight, which explains the vertexes at the edge are mistakenly clustered.

Indicators	**KNN 2D**	**POLY 2D** $\gamma = 0.18$	**RBF 2D**
Adjusted Rand Index	0.8857	0.8498	0.8856
Mutual Information Scores	0.8680	0.8401	0.8622
V-measure	0.8705	0.8443	0.8641
Calinski-Harabaz Index	1192.7951	1160.5767	1215.9292

Table 4: Evaluation Indicators of Spectral Clustering 2D



When we add up to four dimensions, we can see from Figure4.3 that the high dimension dataset has a negative influence on all three approaches, but the comparative connections remain except this time the polynomial method has the best results in that only the polynomial kernel function made the special discrete vertex into the correct cluster, while other kernel functions only focus on achieving the smallest graph cut with the tightest distribution.



Indicators	**KNN**	**POLY**	**RBF** $\gamma = 0.4$
Adjusted Rand Index	0.7591	0.8838	0.7302
Mutual Information Scores	0.7934	0.8484	0.7483
V-measure	0.8056	0.8503	0.7581
Calinski-Harabaz Index	555.6662	436.2717	560.3999

Table 5: Evaluation Indicators of Spectral Clustering 4D

The 13 dimension data6 is to test the performance on high dimensional data comparing with traditional methods. Though the overall performance is less than half of the four-dimensional results, this is still the best we can have comparing with the K-means and DBSCAN since spectral clustering has very little requirements to the data, and can reduce the dimension by utilizing Laplacian matrix.

Indicators	**KNN**	**POLY** $d = 2\gamma = 0.0000955$	**RBF** $\gamma = 0.00005$
Adjusted Rand Index	0.3590	0.3747	0.3308
Mutual Information Scores	0.4132	0.4374	0.3730
V-measure	0.4199	0.4455	0.4029
Calinski-Harabaz Index	533.8577	529.7771	491.5899

Table 6: Evaluation Indicators of Spectral Clustering 13D

For the iris dataset, the performance of the spectral clustering has significant priorities beyond the traditional K-means and DBSCAN, especially under high dimensional conditions. The way of mapping the vertexes into a similarity coordinate system can better reflect the relationships between vertexes especially when the dataset is big. The overall result is not ideal enough for clustering high dimensional data or large dataset since they are still a common problem to be solved.

4.4 BIRCH

Based on previous discussion, there are several important parameters of BIRCH:

1. threshold: the threshold value of the radius of the leaf nodes, i.e. the maximum radius for all hypersphere formed by data points. Lower threshold value often results in bigger CF trees, which means more time and memory is needed to form the tree.

2. branching_factor: the maximum number of CFs in each node.

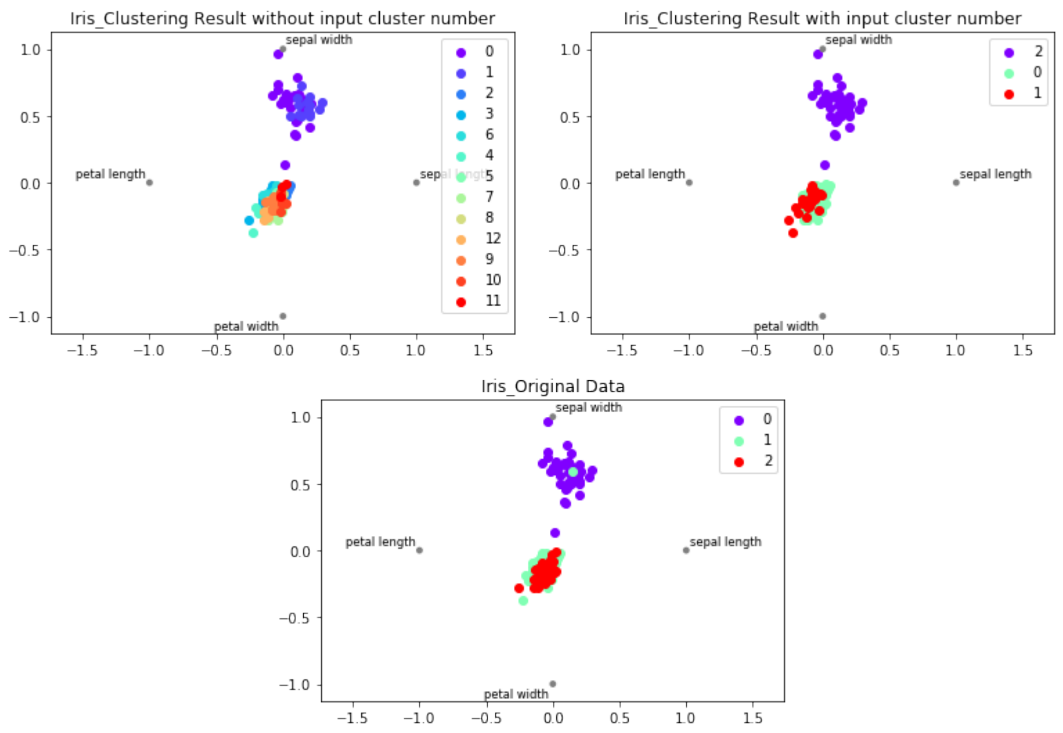


Figure 8: BIRCH result on Iris dataset

3. n_clusters: the number of clusters. Different from K-means, BIRCH does not necessarily need to take the number of clusters as input. If the input is None, the number of sub-clusters is the final result. If the number of clusters is provided, the algorithm will merge sub-clusters to fit the given number.

We first run experiment to see how these parameters influence the clustering result. As is shown in Figure 8, if we know the cluster number in the first place, the result is pretty much satisfying. From Table 7 and 8 we can see that there is no certain pattern to follow. We need to adjust these parameters to get the best performance.

To test the performance of BIRCH on high dimensional datasets, we extract 4,8,12 features from the wine dataset to form 3 sets of data with different dimensions. From results in Table 9 we can see that as the dimension of the dataset goes higher, the clustering performance is getting worse. Therefore we can conclude that BIRCH does not do well on high dimensional datasets.

threshold	**0.1**	**0.3**	**0.5**
Adjusted Rand Index	0.6517	0.5436	0.5823
Mutual Information Scores	0.6764	0.5984	0.6237
V-measure	0.6943	0.6698	0.6566
Calinski-Harabaz Index	503.1305	399.9507	457.5418

Table 7: BIRCH with different threshold values

branching factor	**10**	**30**	**50**
Adjusted Rand Index	0.7122	0.6517	0.6517
Mutual Information Scores	0.7470	0.6764	0.6764
V-measure	0.7606	0.6943	0.6943
Calinski-Harabaz Index	554.9067	503.1305	503.1305

Table 8: BIRCH with different threshold values

dimension	**4**	**8**	**12**
Adjusted Rand Index	0.1219	0.1189	0.1090
Mutual Information Scores	0.1429	0.1264	0.1090
V-measure	0.1572	0.1369	0.1238
Calinski-Harabaz Index	186.6881	244.9339	230.3928

Table 9: BIRCH results on dataset with different dimensions

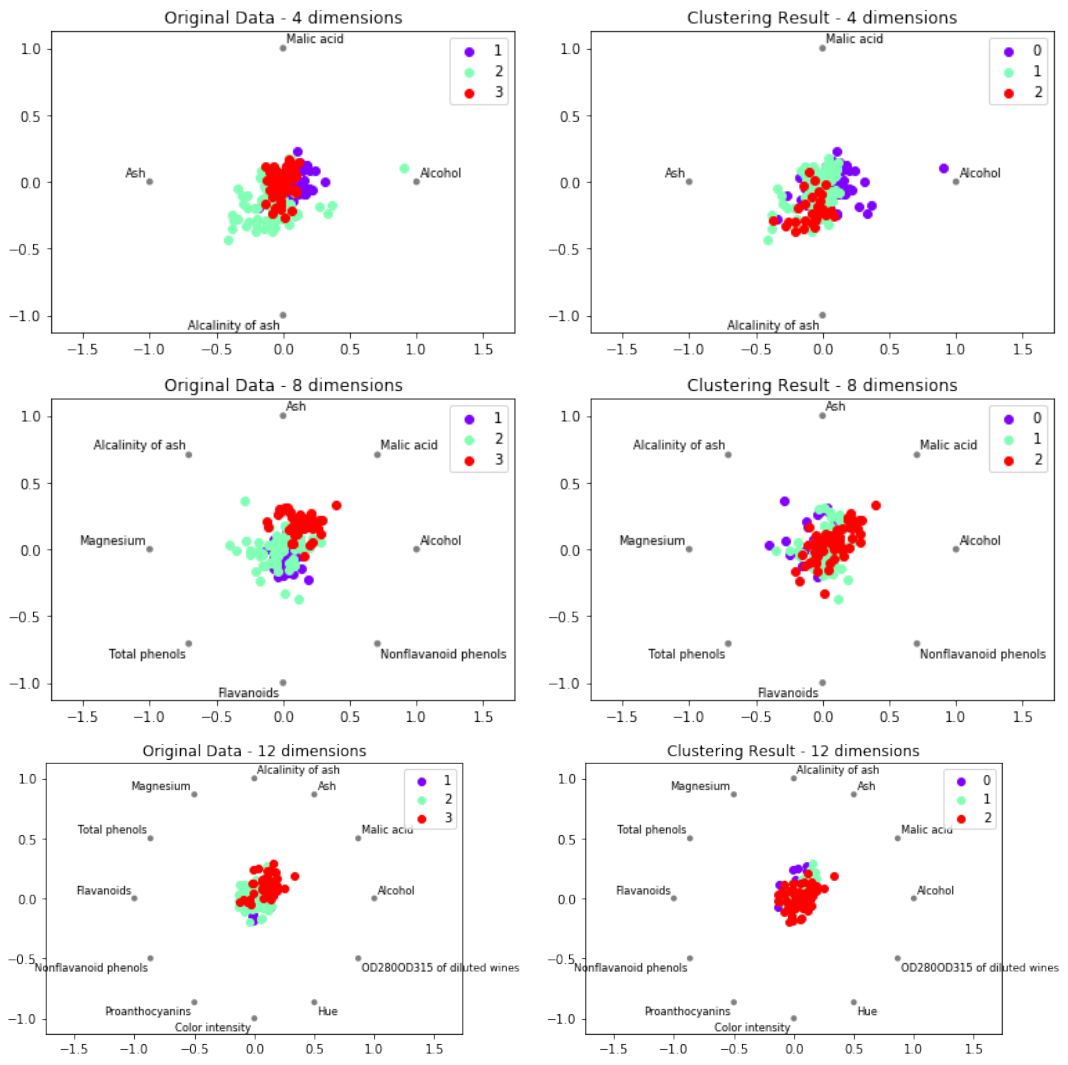


Figure 9: BIRCH results on wine dataset with different dimensions

5 Discussion

5.1 Features of K-means and DBSCAN

Generally, DBSCAN has some optimization comparing with K-means. A density-based algorithm can do clustering with any kinds of the dataset and is not sensitive to noise points. As long as the parameter of DBSCAN is fixed, for the same dataset, DBSCAN would always have the same result. Respectively, k-means can only treat with a convex set, and its initial value does have some influence on the clustering result. But DBSCAN has some requirement to data, for example, it requires an even dataset to get the best clustering result, and since it uses euclidean distance [8], it may not work correctly when numbers of feature go large.

5.2 Spectral Clustering

As a creative clustering algorithm, spectral clustering has little requirements for the input data and is highly scalable. But the spectral clustering is very sensitive to the parameters and the method that used to generate the affinity matrix, which means choosing a proper configuration can take a lot of time. There are several approaches which can improve the performance, which mainly focuses on the kernel function and the graph-cutting theory. More works can be carried out on the optimization of spectral clustering for high dimensional data and large-scale datasets for the robustness of spectral clustering.

Also since the spectral clustering is an unsupervised learning algorithm, we can add supervised elements, e.g., labels to help increase the accuracy.

5.3 BIRCH

Like DBSCAN, BIRCH does not require number of clusters as input, which allow us to still use this algorithm without prior knowledge of the cluster number. BIRCH is a very efficient algorithm: 1.it keeps the dataset on disk, and only CF tree is stored in memory 2. it needs only one scan to build the CF tree. Therefore BIRCH is good for large dataset with large number of clusters. On the other hand, due to the limitation of CF number in the CF tree nodes, the clustering result might be different from real classification. From our experiment, we can also see that BIRCH falls short of accuracy when dealing with high dimensional datasets. Last but not least, sometimes it is difficult to tune the parameters given the fact that different parameters finally result in different CF tree structures.

6 Contribution

Song Yang: Did the implementation and analysis of DBSCAN clustering.
Xin Yang: Xin did the implementation and analysis of the spectral clustering.
Yi Wu: Did the implementation and analysis of K-means clustering.
Zhuohang Li: Did the implementation and analysis of BIRCH clustering.

References

[1] T. I. Murphy, "Line spacing in latex documents," https://en.wikipedia.org/wiki/DBSCAN, accessed April 4, 2010.

[2] J. S. [1, J. L. [1, and L. Z. [1, "Clustering algorithms research," *Journal of Software*, vol. 19, no. 1, pp. 48–61, 2008.

[3] Tian ZhangRaghu RamakrishnanMiron Livny, "Birch: An efficient data clustering method for very large databases," *ACM*, 1996.

[4] R. Fisher, "The use of multiple measurements in taxonomic problems," *Annual Eugenics*, vol. 7, pp. 179–188, 1936.

[5] P. Forina, M. et al, "UCI wine data set," 1991. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/wine

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] Wikipedia, "Spectral clustering," https://en.wikipedia.org/wiki/Spectral_clustering, 2018.

[8] "Euclidean distance," https://en.wikipedia.org/wiki/Euclidean_distance.