**Data Mining Portfolio Entry for Clustering (Frank Liu)**

**Data Acquisition**

A heads up

* There are 4 datasets that I used in this data portfolio, including 3 provided by professor Santago, and 1 is my own dataset

**Data provided by instructor:**

“separated\_2d.csv”

* This dataset contains 62 samples that have 3 features, including a class features and two numeric features which are used to do clustering

“separated\_2d\_mixed.csv”

* This dataset contains 62 samples that have 3 features, including a class features but more clearly labeled and two numeric features which are used to do clustering

“iris \_student.csv”

* This dataset contains 151 samples that have 5 features, including a class features and 4 numerical values specifying length and width of petal and sepal of iris.

**My own data**

“College.csv”

* This dataset is referenced from: <https://www.kaggle.com/karthickaravindan/k-means-clustering-project/notebook>
* The dataset is also uploaded to /data folder in the clustering analysis folder.
* This dataset has a similar format to all previous datasets, It contains 19 features of some attributes of a college, including the number of applicants, number of accepted, number of enrolled, number of undergraduates, etc. 17 of them are numeric.
* The purpose is to classify them into clusters of similar colleges. For such data, **the range of each feature is large, so normalizing them is a good idea.**
* This data has 778 sample points.

**Usage of dataset**

* I use ALL 4 datasets in my code development and package use section of the code in order to do a thorough analysis of ideas around association analysis.

**Code Development / Program Development**

This section of the DMP contains the analysis of the self-implemented **k-mean, k-means++, and DBSCAN algorithm, accompanied with cluster cohesion, cluster separation analysis, and different plots that evaluate the performance and some insights.** I will possibly include bisecting k-mean implementation after finishing this DMP later.

I will use Python in Google Colab to develop the code. Especially, I use two very helpful data structures: dictionary and set:

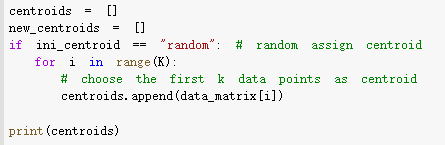
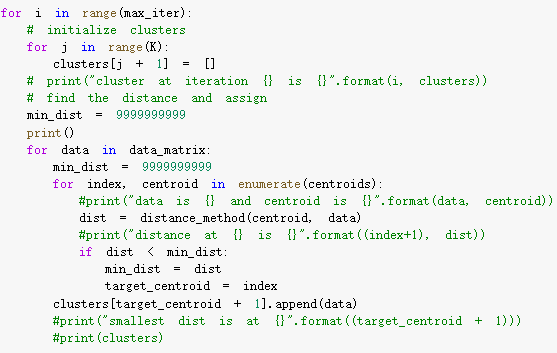
* Some data structure I used is list and dictionary
* I find using a **dictionary** is very efficient to organize my clusters since I have a unique key associated with each cluster, and the size of the dictionary can change according to hyperparameter K easily.

**K-means clustering:**

**Code link:**

<https://colab.research.google.com/drive/1XjEOWrD5dHC4AH9N4uAWRRuMJg1PHAvk#scrollTo=x3RFC5dm7Sp->

**Important points to note**

* Data input
  + Data is processed to remove the first column (class labels)
  + The decision of **scaling the data** or not will be analyzed later, but the example output console will use unscaled data of iris dataset
* Initial centroid assignment
  + I implemented random assignment (the first k points) and k-means++, I will compare the difference in a later section
  + 
* Assigning points to closest centroid
  + I have four different distance measures, and I will compare the difference in later section
  + 
* Hyperparameters
  + K = 3 # default
  + Ini\_centroid = “random” # can be k-means++
  + Max\_iter = 500 # the program will break if SSE, not change
  + Distance\_method = L2 # can change to other measures

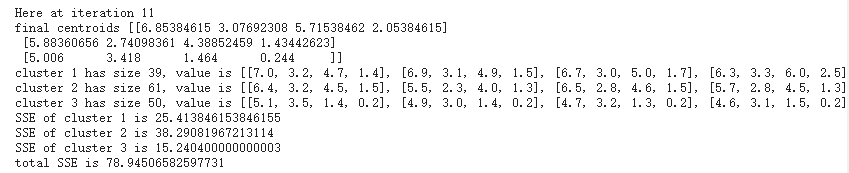
**Analysis of K-mean**

1. **Output console analysis**

All output follows the default hyperparameters listed above

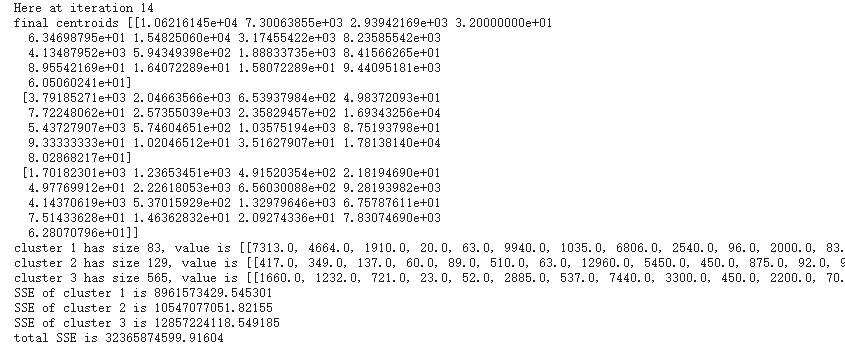
First, I will show the output console for four different datasets (the full list of clusters is not shown here)

* Iris data



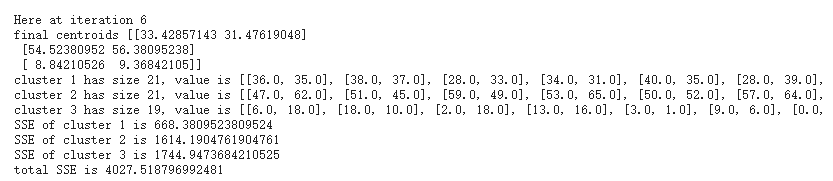
We can see the process finish at 11 iterations, with a total SSE of around 79. This is using

* College data



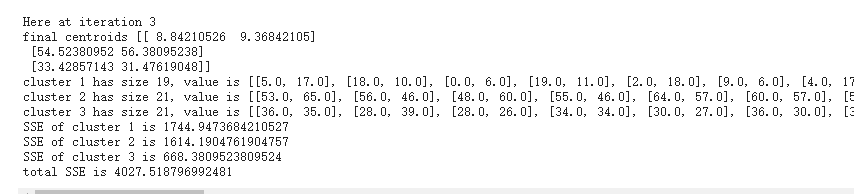
College data end at iteration 14, but the SSE is still very large, at around 30 billion. I think this might be **due to not scaling the data or a large sample size**. But I will analyze the change in SSE later.

* Separated\_2d



This process ends at iteration 6, with SSE of 4027

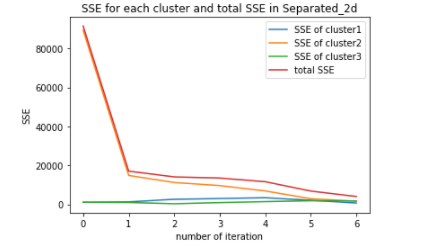
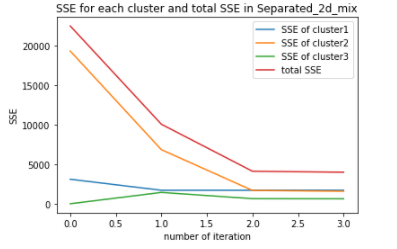
* Separated\_2d\_mix

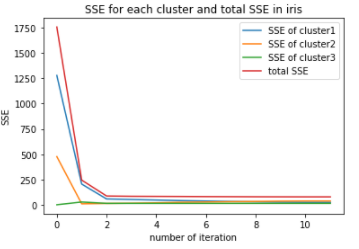
****

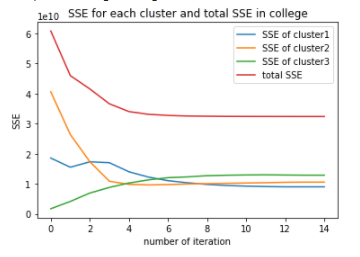
This process ends at iteration 3, with SSE of 4027

1. **SSE with num\_iterations**

To have an idea of if my implementation is working fine, I use K = 3 and plot the SSE change with the number of iterations





 (note the scale of y is 1e10)

I found out that some **individual clusters’ SSEs increase over time**. I was confused about this phenomenon initially, but I think this might be due to there being more points added into the clusters. As long as the sum of all individual SSE(total SSE) is decreasing, my algorithm will work.

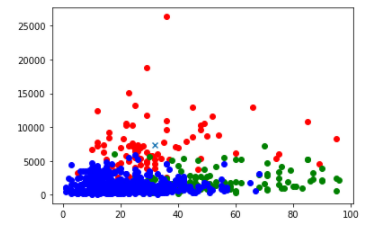
I observe an overall decreasing trend in the total SSE, meaning that clusters are all assigned to their corresponding closest clusters as much as possible.

We can see some SSE, after iteration stop, still very high. This might be solved **using normalization of data, or increasing the number of clusters** (here I only use k = 3). I will try this method.

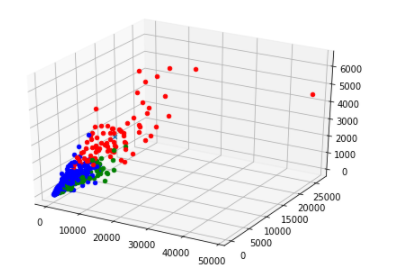
1. **2D / 3D plot analysis**

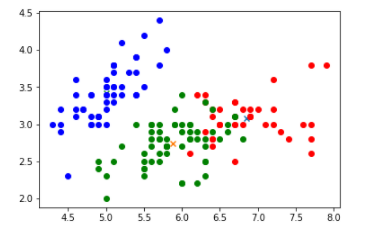
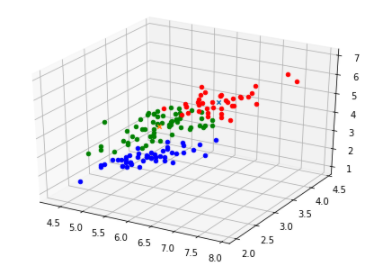
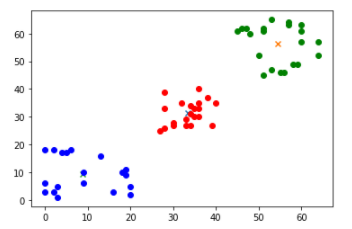
I plot 2D for datasets that only have 2 features, and both 2D and 3D for datasets that has more than 2 features. **Features are randomly chosen**. Since features are randomly chosen when generating the graph, this may not represent the best captures of clustering.

* College data



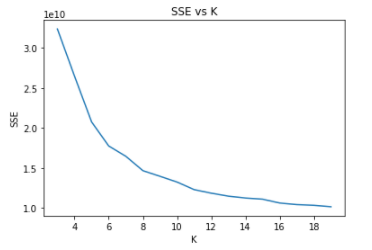
This seems like to have some outliers, but we cannot be sure in a 2D graph for data has 17 features



* Iris data
* 
* 
* Separate data (2 of them are the same)
* 
  + This looks pretty good

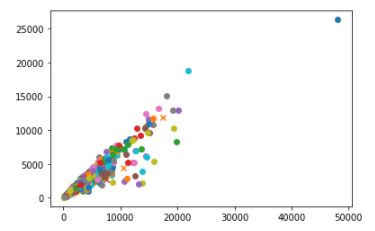
All following sections of analysis will **use my own data** that has more features so we can see a **significant difference** in analysis.

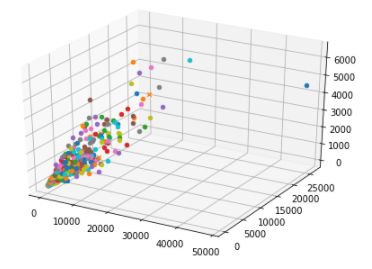
1. **Best K analysis (Final\_SSE with K)**

****

We can see our SSE decrease significantly with an increasing number of clusters, but the change in slope (derivative) is lower. Meaning as we increase the number of k, the improvement is smaller. According to this graph, I would say a good k will be 13 or 14.

The 2D / 3D plot of k = 14 is here.

****

****

Probability, it is not a good idea to visualize 17-dimensional features of 14 clusters in 2D and 3D plots, but anyway, our SSE is decreased by a lot

1. **Distance Measures analysis**

Based on previous analysis, I will use college data of k = 14 as a baseline to compare 4 different distance measures. The default way of doing it before is using the L2 norm

To keep the result comparable, I will compute the final SSE using the L2 norm so all data are on the same scale, but to assign data points to different clusters will use different distance measure

| **Distance Measure** | **SSE** |
| --- | --- |
| L2 norm | 11256448013.53 |
| L1 norm | 12666126304.75 |
| Cosine similarity | 177375182081.90 |
| Infinity Norm | 11770753989.64 |

We can see from the graph that the L2 norm performs the best, having the smallest SSE, followed by infinity norm, cosine similarity, and then L1 norm. To figure out why, Google tells me that **L2 tends to perform better when we have collinear/codependent features**, and in the college example, for sure that number of applicants is positively correlated to a number of enrollment, and a number of acceptance, etc. This explanation makes sense to me. However, the SSE number is so big that it is hard to analyze. We can **normalize the data** to make it in a smaller range without changing the relationships.

1. **Normalizing data analysis vs SSE**

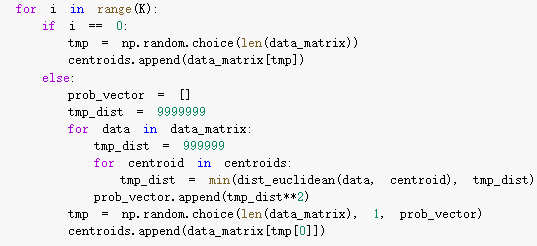
This section will use the College dataset with k = 14, and L2 norm distance measurement.

**The SSE result for not normalizing data is 11256448013.53, after we normalize the data, the SSE becomes 103.88**. I expect to see this decrease in SSE because the data itself is with a smaller value, so the distance will be smaller.

To see if normalization helps, one way is to **see the number of iterations to form a final cluste**r. Before normalization, it takes 38 iterations but it only takes 24 iterations to converge for data after normalization. To see if clusters are better, we should consider some cluster validation methods, which I will focus on in a later part.

1. **SSE with choice of initializing centroid**

This is my implementation of initializing centroid for k-means++

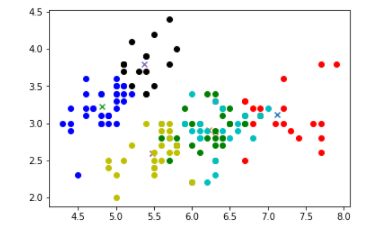


The first centroid is selected randomly, and **all other point has a probability of being picked as the new centroid that is proportional to the square of its distance to its closest centroid**

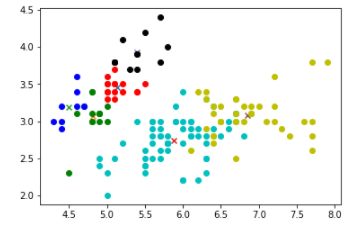
Compared to randomly initializing, the number of iterations reduces from 38 to 34, but the SSE turns out to increase. I think this might be because this **College dataset contains lots of outliers, and k-means++ are known to select outliers.**

I tried the method on the iris dataset, which is smaller compared to this dataset. With k = 6, random initialize has an SSE of 68.63856463550111 but k-means++ has an SSE of 47.82689415602404. This is a significant amount of reduction, and we can see with slightly higher k, and cleaner data, some strategy on choosing the initial centroid helps a lot.

K-means++



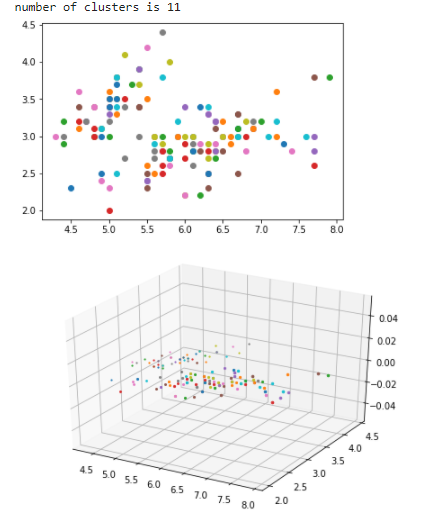
Random



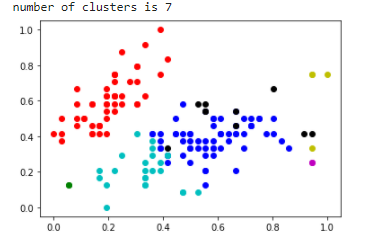
**DBSCAN**

I have also tried to implement the **density-based clustering algorithm --- DBSCAN**, which involves classifying each point into “core”, “border”, and “noise”, and then clustering “core” points within Eps into the same clusters.

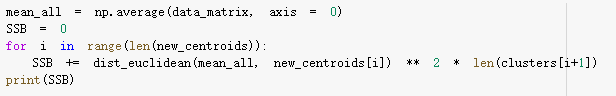
This takes me a very long time to come up with a way to tune the best hyperparameter (Eps and MinPts). The result of the cluster in the Iris dataset, with MinPts = 4 and Eps = 10 results in 11 clusters, with 2D and 3D graphs looking like this.



This obviously is **not ideal compared to K-mean**, and I am trying to improve the performance by tuning hyperparameters. I checked my implementation logic and think it is correct. But the points in the same clusters are looking different from each other. I guess it might be because the data is high-dimensional. But after I print out their distance, I find out that some distance is 2000, and even 5000. This reminds me to **normalize the data**. Later I also find out that to prevent using the same data, I use a list. remove with for data in list, which is a loop that will cause the index to get out of control somehow. Lots of data will not be processed. I change my implementation and keep track of whether the data has been classified by using a boolean array. This time it works fine with Eps = 0.2 and MinPts = 5. I was about to give up after hours of debugging, but I am glad I stick with it and finally made this algorithm work.



**Cluster Cohesion and Separation Implementation**

* I further implement the cluster cohesion and separation calculation so that I can do a more thorough analysis of the performance of the formed clusters
* Cohesion
  + Cohesion is measured by the within-cluster sum of squares, which is the SSE
  + SSE function is already implemented in the analysis of K-mean algorithm, which can be served as a measure of cohesion for clusters
* Separation
  + Separation is measured by the between-cluster sum of squares.
  + 
  + Here we need a value of centroid of all data points
  + 
* TSS = SSE + SSB?
  + Once I have the SSE, SSB, I want to test if SSE + SSB is a constant, per the book.
  + I tried different k values while keeping all other variables fixed, and see if SSE + SSB are constant

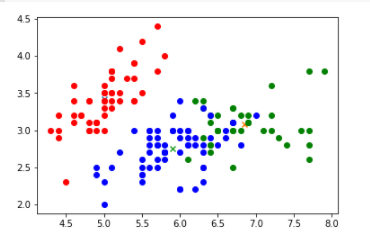
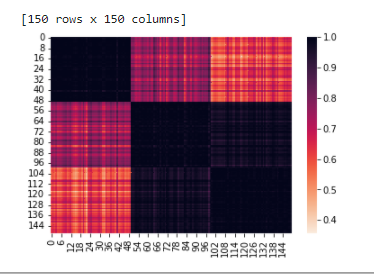
| **K** | **SSB** | **SSE** | **TSS** |
| --- | --- | --- | --- |
| **4** | 3914.95 | 71.34 | 3986.29 |
| **5** | 3972.50 | 49.74 | 4022.25 |
| **6** | 4009.67 | 47.91 | 4057.58 |
| **7** | 4017.34 | 47.05 | 4064.39 |
| **8** | 4110.86 | 30.53 | 4141.39 |

* As we can see, even if the TSS is not exactly the same, they do **fall into a small range of data**, approximately 4000, 4100] so it can have some indications that if the algorithm works perfectly, TSS = SSE + SSB. And there is an exercise in the book that requires us to prove part of it, and mathematically it should be equal to each other.
* With increases of k, **SSB is increasing, and SSE is decreasing,** which confirms the fact that if we see clustering as an optimization problem, minimizing SSE is equivalent to maximizing SSB.
* I further tried if normalizing the data will help get a more accurate result

| **K** | **SSB** | **SSE** | **TSS** |
| --- | --- | --- | --- |
| **4** | 12.21 | 5.53 | 17.75 |
| **5** | 12.43 | 5.20 | 17.63 |
| **6** | 13.07 | 4.56 | 17.52 |
| **7** | 13.49 | 3.86 | 17.36 |
| **8** | 14.22 | 3.34 | 17.56 |

* We see that SSB increasing, SSE decreasing as usual, but TSS are very stable this time, falls into a range of [17.36, 17.75], which can be seen as a strong indication of TSS = SSE + SSB

**Proximity Matrix Correlation**

* I will also use the iris data with k = 3 to plot the proximity matrix correlation to measure cluster validity
* I will first need to construct an ideal similarity matrix and proximity matrix, and then calculate the correlation coefficient between them
* This is the visualization of the plots
* 
* After calculate the correlation of the ideal similarity matrix and proximity matrix, the correlation is -0.6004
  + This makes sense to be negative since I use euclidean distance as proximity measure, and if two data points are in the same cluster, the ideal matrix will have 1, but the distance should be closer to 0. This is a negative correlation, which makes sense to have a negative number
* I will plot the correlation matrix here
* 

**Package Usage (scikit-learn)**

Code link: <https://colab.research.google.com/drive/15TCJAah-TNmNbqbj1NDN-x8JaSFfXNkr#scrollTo=wsYIcTwb-ZcL>

In the package use section, I will focus on my college dataset and iris dataset, but implement three different types of clustering using scikit-learn

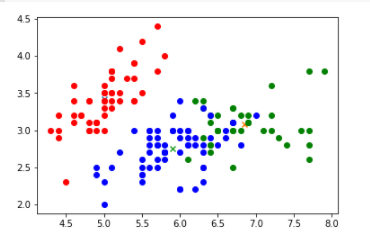
* Prototype-based --- K-mean, with plots
* Hierarchical --- Agglomerative, with plots
* Density-based --- DBSCAN

I will also compare the result of my own implementation and the package implementation of k-mean and DBSCAN. Lastly, I will do some cluster evaluation methods

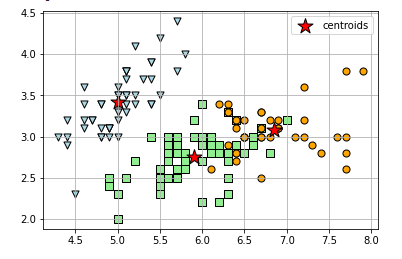
**K-mean**

Iris dataset:

My own implementation of K-mean with k = 3



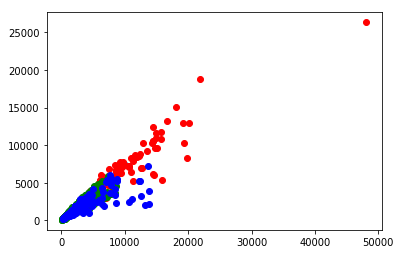
Sklearn implementation



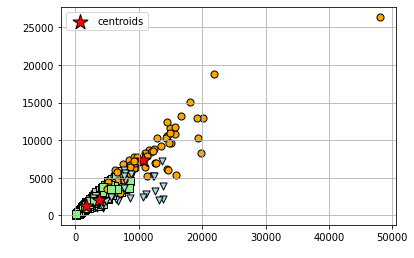
Based on this graph, I think I did a really good job in clustering the dataset, and we can see the clusters are almost exactly the same, so I believe my own implementation of K-mean is an accurate approximation of the original algorithm.

College dataset with k = 3

My own implementation

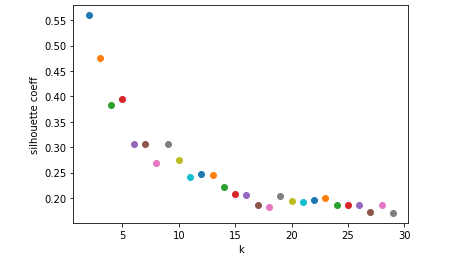


K-mean implementation

****

Given the high dimensional characteristics of the data, it is difficult to see which one is better, but clearly, it is a similar clustering result for my implementation compared to that of sklearn.

For college data, since it is not obvious to visualize, I use silhouette coefficient to choose the best number of clusters for College Dataset:



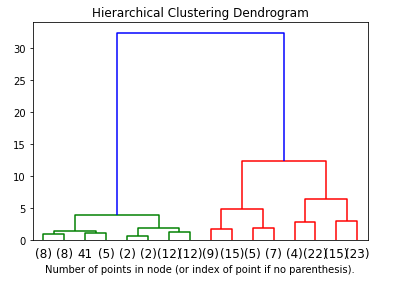
We choose the k where the silhouette coefficient stops dropping dramatically, which is typically around 10 - 15. This confirms with the previous SSE vs K analysis, where the best K is at 14.

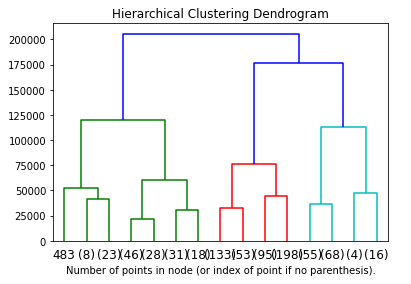
**Agglomerative**

Since I didn’t implement a hierarchical clustering algorithm on code development, I will show some results of package use of agglomerative algorithms in sklearn.

The good thing about this algorithm is that you do not need to specify the number of clusters, only setting distance\_threshold=0 ensures we compute the full tree.

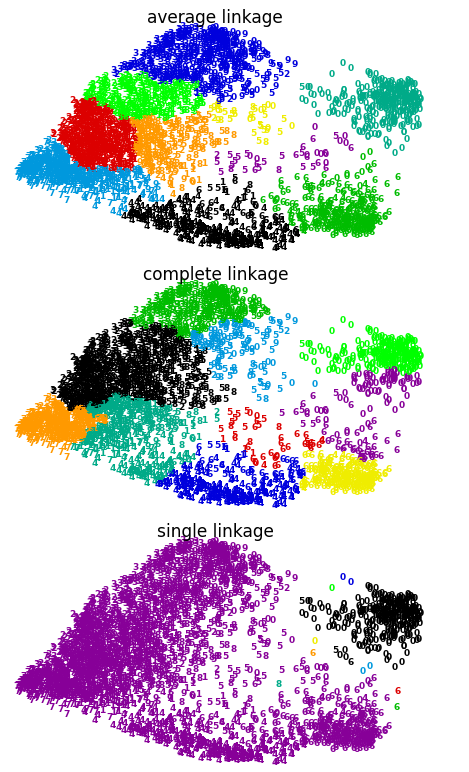
* Iris Data



* College Data
* 

Using a digit dataset imported from sklearn, I also show the v[arious agglomerative Clustering on a 2D embedding of digits](https://scikit-learn.org/stable/auto_examples/cluster/plot_digits_linkage.html#).

We can compare different proximity measures using complete linkage, average linkage, and single linkage.

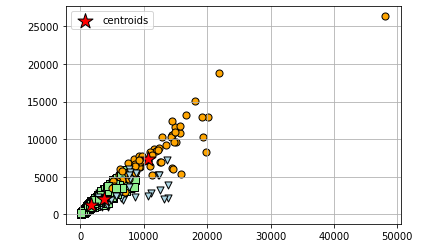


Obviously, the best performance is average linkage, and the worst performance is single linkage, which makes sense since average linkage computes the pairwise average between all clusters, which is a more comprehensive measure of proximity, so, during the update, the points can be assigned to the most approximate clusters.

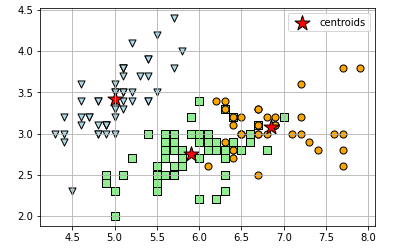
**DBSCAN**

I use sklearn for DBSCAN algorithm and plot the clustering for iris and college data, for K = 3

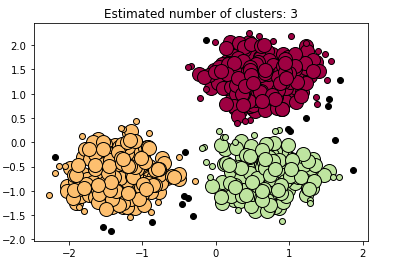
For college dataset, this has no visualization difference with that of k-mean



For Iris dataset, this also has no visualization difference with that of k-means



I think for low dimensional databases, two algorithms have a similar performance. DBSCAN has advantages over other algorithms in that it is resistant to outliers and can handle different shapes and sizes. Therefore I reference sklearn's library called “make\_blobs” and generate some Generate isotropic Gaussian blobs for clustering.



This can clearly visualize the noise point, core point, and border points.

**Chapter 7 Exercise Theory**

[Cluster Exercises](https://docs.google.com/document/d/1MdAJOgDg2QM35k9EZzuaCbHAWKNEq2WORGxFpYFm6t8/edit)

**Chapter Summary and Major Learning Point**

The attached link is the reading notes I take on ppt, reading, and in the lecture.

* [Lecture and Reading Note](https://docs.google.com/document/d/12-G0-Ljt1MGclVoozyHVOWtWeLP-R1rKDAYgJ7dSj2w/edit)

This is a bit wordy so I will include some of the major learning points that I have

* K-mean
* Different ways of initializing centroids
* Ways of improving K-mean
* Other extended K-mean algorithms
* Cluster visualization techniques
* Different proximity measures
* DBSCAN
* Density-based algorithm
* Time complexity of them
* SSE, SSB, TSS relationship
* Pros and Cons of each algorithm
* Different clustering validity measures
  + Correlation matrix
  + Silhouette coefficient
  + Hopkins statistics
  + Cohesion
  + Separation
  + etc.

**Learning Summary and Self-Assessment:**

Clustering is the DMP that I spent the most time on. There are so many places I can do analysis on and so many places that I am interested in discovering more. Given the unsupervised nature of clustering algorithms, there are no right or wrong answers to your clustering, it all depends on the way you evaluate it using different metrics. The freedom of exploration gives me more motivation and excitement.

Overall, I think I had a decent understanding of the material and gained lots of insights in doing the coding. Especially code the K-mean and its extended derivations are particularly challenging but interesting. But I am happy that I can implement it. I think I have an A to A- comfort level in clustering after finishing DMP.