Large Scale Clustering

1. Introduction

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

The applications of clustering can be found in many different areas, such as human genetic clustering, grouping of shopping items, E-commerce, Social network analysis, Recommender systems, Crime analysis, image segmentation and so on.

Clustering seems easy when the data is small with only 2 dimensions. But in most cases, the applications involve high-dimensional space. Many popular clustering algorithms are inherently difficult to parallelize, and inefficient at large scales.

Before introducing different algorithms for clustering, we need to define the distance between the pair of data points. We can divide the data space into Euclidean space and Non-Euclidean space. There are different methods to measure the distance in different types of space. For Euclidean space, the measurement of distance can be Euclidean, Minkowski or Manhattan distance. Given two points $[x_1, x_2, \ldots, x_d]$ and $[y_1, y_2, \ldots, y_d]$ in d-dimension Euclidean Space, the Euclidean distance between them is $\sqrt{|x_1 - y_1|^2 + |x_2 - y_2|^2 + \ldots + |x_d - y_d|^2}$. The Minkowski distance between them is $\sqrt[q]{|x_1 - y_1|^q + |x_2 - y_2|^q + \ldots + |x_d - y_d|^q}$. And Manhattan distance between them is $|x_1 - y_1| + |x_2 - y_2| + \ldots + |x_d - y_d|$. Actually, Euclidean distance and Manhattan distance are respectively the special case when $q = 2$ *or* 1 for Minkowski distance. In the case of Non-Euclidean space, usually we use cosine distance to measure the similarity between vectors, use Jaccard distance

to measure the similarity between sets and use edit distance to measure similarity between strings.

2. Algorithm

2.1. Hierarchical Clustering

We will mainly discuss the Hierarchical Clustering in a Euclidean space. This algorithm can only be used for relatively small datasets. Initially, each point is a cluster by itself and is the centroid of that cluster, where the centroid of a cluster is defined as the average of its points. The key operation of hierarchical clustering is to repeatedly combine the two "nearest" clusters into one.

The distance between two clusters can be measured in different ways:

1) Distance between their centroids.
2) Minimum of between any two points, one from each cluster.
3) Average distance of all pairs of points, one from each cluster.
4) The radius of a cluster is the maximum distance between all the points and the centroid. Combine the two clusters whose resulting cluster has the lowest radius.
5) The diameter of a cluster is the maximum distance between any two points of the cluster. We may choose to merge those clusters whose resulting cluster has the smallest diameter.

Stopping rules of Hierarchical clustering:

1) Stop when we have k clusters for some predetermined k.
2) Continue clustering until there is only one cluster and return the tree representing the way in which all the points were combined.
3) If the diameter/density that results from the best merger exceeds/is below a threshold.

4) Stop when there is evidence that the next pair of clusters to be combined yields a bad cluster. For example, we could track the average diameter of all the current clusters. As long as we are combining points that truly belong in a cluster, this average will rise gradually. However, if we combine two clusters that really don't deserve to be combined, then the average diameter will take a sudden jump.

If the case is Non-Euclidean space, the only "location" are the points themselves. We need to use clutroid to represent a cluster of many points, where clutroid is defined as the points "closest" to other points. And "closest" can be:

1) Smallest maximum distance to other points
2) Smallest average distance to other points

2.2. K-means Clustering

K-means clustering is a well known algorithm for clustering, especially in point-assignment algorithms. We need to assume Euclidean space and known k for k-means algorithm. The steps of K-means clustering are as follows:

1) Picking k initial centroids of cluster
2) For each point place it in the cluster whose current centroid is the nearest
3) After all points are assigned, update the locations of centroid for each cluster
4) Reassign all points to their closest centroid form new clusters
5) Repeat 3 and 4 until convergence

In order to pick points that have a good chance of lying in different groups, there are two approaches.

1) Cluster a small sample of data (e.g. hierarchical clustering) to obtain k clusters. Pick a point from each cluster (e.g. the point closest to the centroid of the cluster)
2) Pick the "dispersed" set of points, which also means picking points that are as far away from one another as possible. The steps are as follows:

a. Pick the first point at random.
b. Pick the next point whose minimum distance from the selected points is as large as possible.
c. Repeat step (a) and (b) until we have k points.

Another question we want to discuss about k-means algorithm is how to determine the right k. There are also two methods.

1) The first method is to use the average distance. By trying different k, we can look at the change in the average distance to centroid as k increases. Average distance falls rapidly until convergence, so we can get the right k.
2) Another way to get right k is to use Silhouette value. The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

2.3. Map-Reduce of K-means Clustering

Given $x_1, x_2, \ldots, x_n \in R^d$ and k, we first initialize $\mu_1, \mu_2, \ldots, \mu_k$ and keep as a global variable. Each iteration of k-means is a process of Map-Reduce.

1) Map: find optimal assignments of x_i to μ_j by minimize $||x_i - y_j||_2^2$.

   The key value pair will be (j, x_i).

2) Reduce: Re-evaluate μ_j given $\mu_j = \frac{1}{n_j} \sum_{key=j} x_i$.

3. Implementation

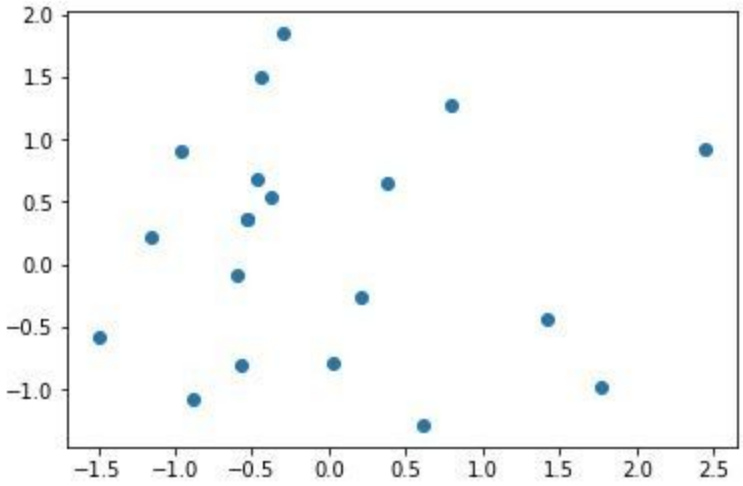
3.1. Implementation of Hierarchical Clustering

Since hierarchical clustering can only handle relatively small dataset, we tried several datasets generated by ourselves and a small mixed dataset in this section.

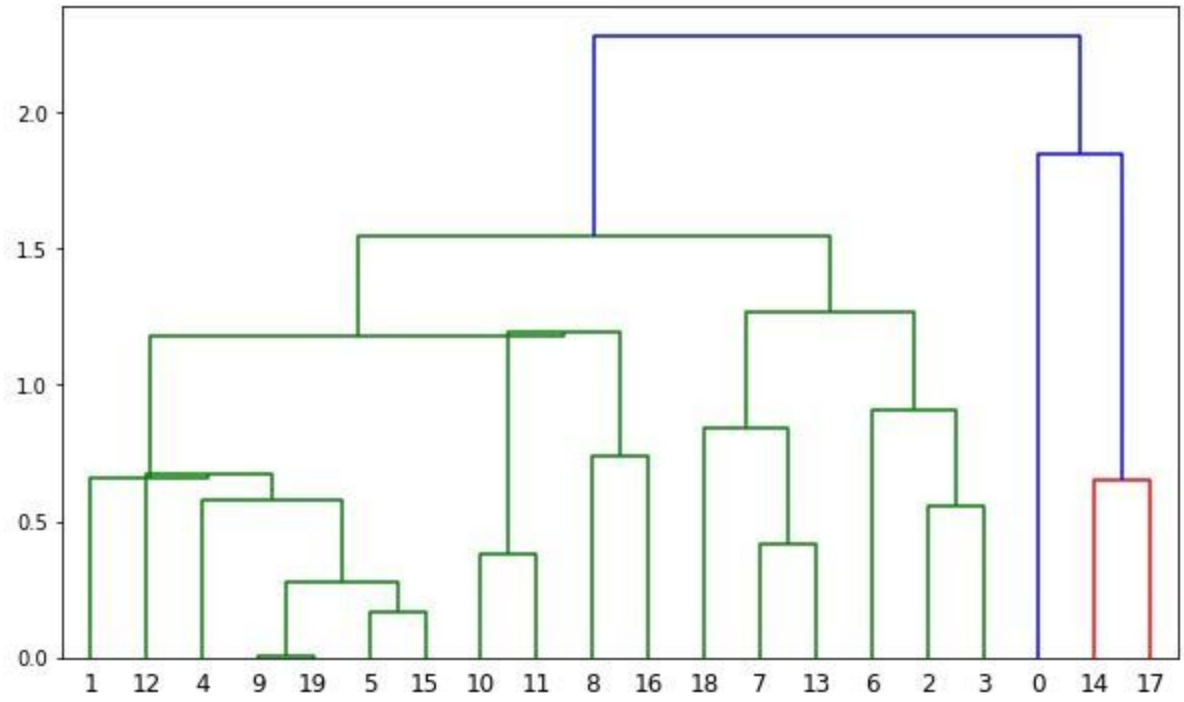
3.1.1. 2D case

The following is the location of 20 points and the diagram in 2D view:

```
[[ 2.44895463  0.92196277]
 [-0.96034651  0.9032957 ]
 [ 0.02901539 -0.78737855]
 [ 0.20649497 -0.26139178]
 [-0.5944483  -0.07775586]
 [-0.4647168   0.68516452]
 [ 0.61411929 -1.28645147]
 [-0.8829971  -1.08022213]
 [ 0.3793637   0.6532097 ]
 [-0.53376559  0.3554359 ]
 [-0.30089065  1.84956716]
 [-0.43878199  1.49272959]
 [-1.15690096  0.21399193]
 [-0.57729411 -0.8021647 ]
 [ 1.76912745 -0.98337359]
 [-0.37833888  0.54232537]
 [ 0.79654699  1.26724739]
 [ 1.41571574 -0.44007497]
 [-1.48971405 -0.57922219]
 [-0.53621585  0.35982533]]
```

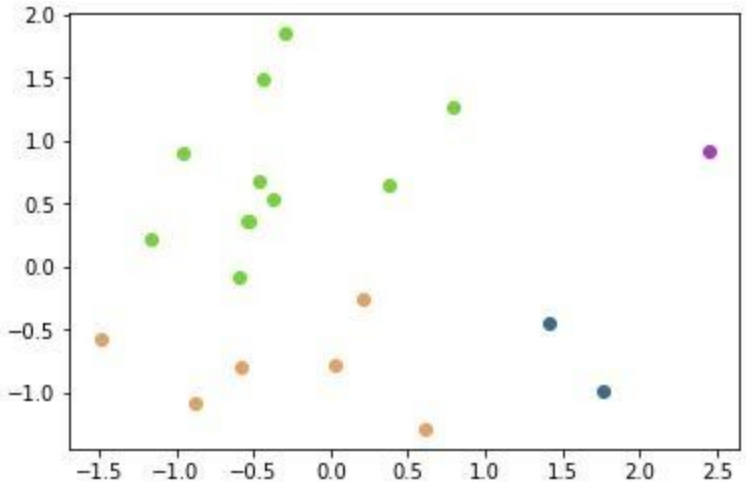


After hierarchical clustering, we can get a dendrogram as below:



The numbers on the horizontal axis correspond to the indexes of all the points, and the numbers on the vertical axis correspond to the distance between clusters during the combination.

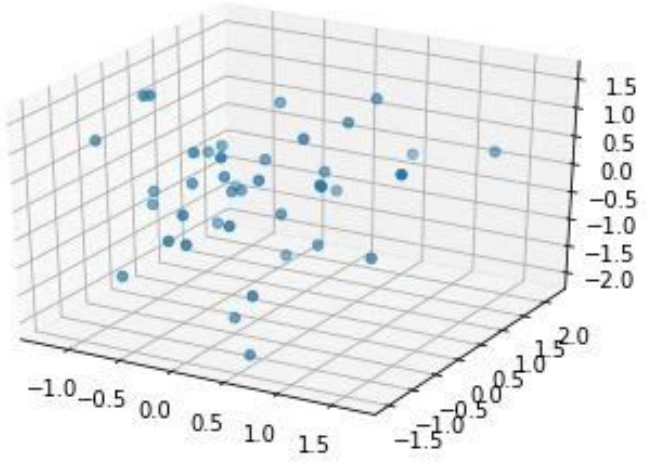
After hierarchical clustering, we can specify the number of clusters that we want. For example, let k = 4. Then the points in 2D view can be separated into 4 clusters:

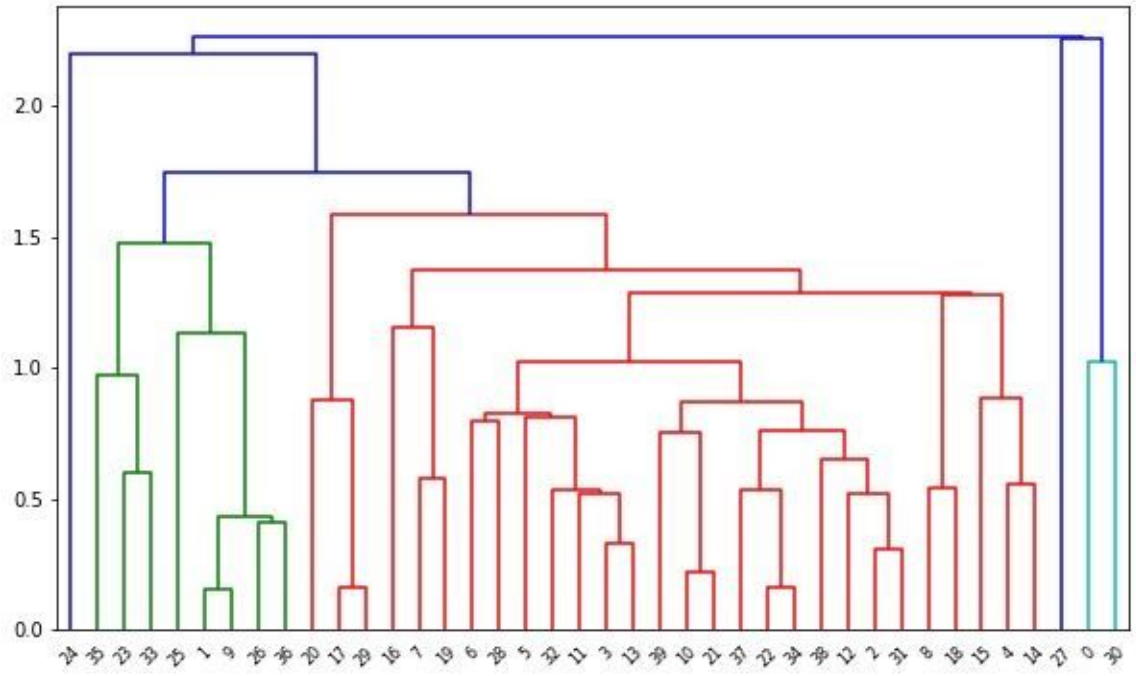


Different colors indicate different clusters.

3.1.2. 3D case

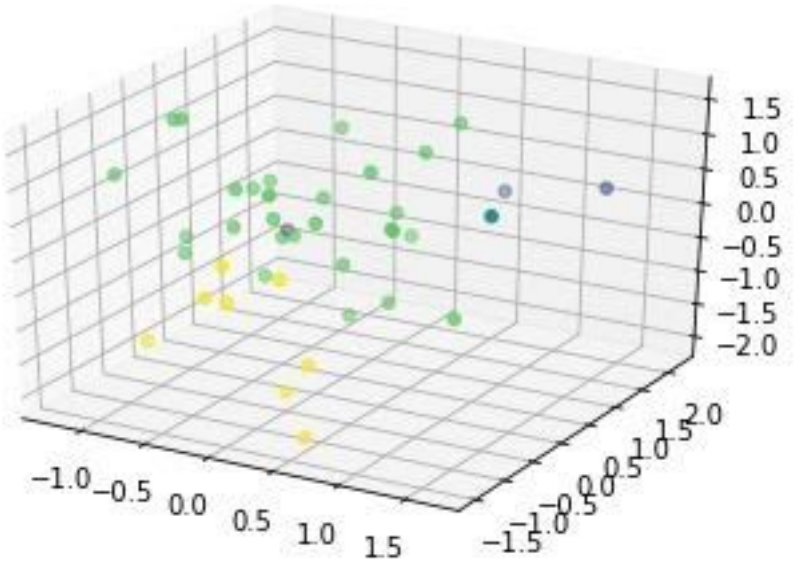
In the 3D case, we generated 40 points.





The hierarchical clustering result is as below:

If we stop at k = 5 clusters, we can get the result as below:



3.1.3. Categorical variables

	area	perimeter	compactness	length	width	asymmetry_coefficient	groove_length	grain_variety
0	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	Kama wheat
1	14.69	14.49	0.8799	5.563	3.259	3.586	5.219	Kama wheat
2	14.03	14.16	0.8796	5.438	3.201	1.717	5.001	Kama wheat
3	13.99	13.83	0.9183	5.119	3.383	5.234	4.781	Kama wheat
4	14.11	14.26	0.8722	5.520	3.168	2.688	5.219	Kama wheat

This dataset contains 8 variables, and the last variable is a categorical variable which indicates three varieties of grain. We first convert it into dummy variables:

(code in python)

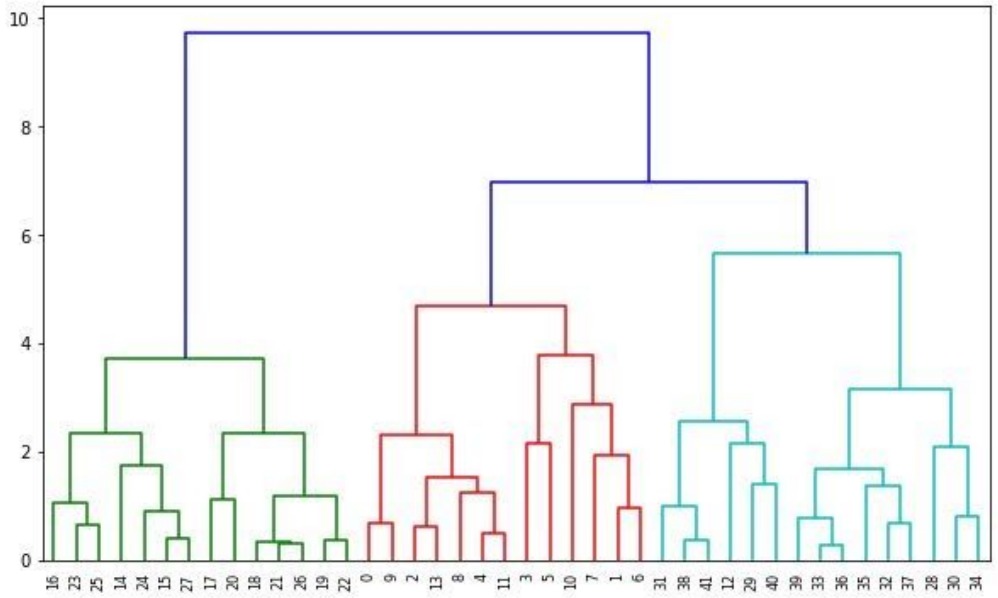
```
seeds_df = pandas.read_csv('seeds-less-rows.csv')
just_dummy = pandas.get_dummies(seeds_df.grain_variety)
seeds_dummy_df = seeds_df.join(just_dummy.astype(float))
seeds_dummy_df = seeds_dummy_df.drop(columns = 'grain_variety')
```

Then we implement the hierarchical clustering:

(code in python)

```
seeds_linkage = hcluster.linkage(seeds_dummy_df, method='complete')
k_seeds = 4
f = hcluster.fcluster(seeds_linkage, k_seeds, 'maxclust')
fig = plt.figure(figsize=(10, 6))
dn = hcluster.dendrogram(seeds_linkage)
plt.show()
```

Finally, we can get the dendrogram:



3.2. Implementation of large-scale k-means Clustering

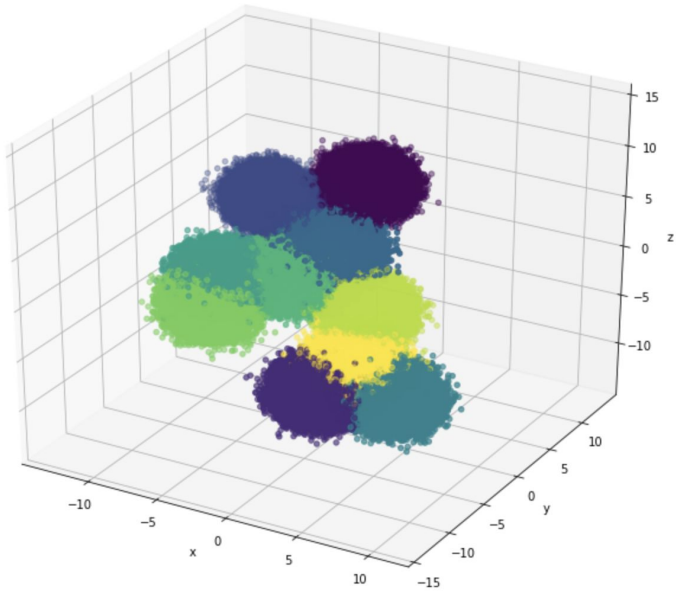
3.2.1. Dataset description

We used the make_blobs function to generate isotropic Gaussian blobs for clustering. The dimension of the dataset is 1000000x3. It has three features x, y, z and all the values are from -10 to 10

Then we get the dataframe like this:

	x	y	z	id
0	-0.971553	5.471752	-5.574415	row0
1	-0.704415	2.943728	-7.631530	row1
2	-7.502310	6.226396	2.620300	row2
3	1.194358	-6.347815	-7.652893	row3
4	5.477265	-5.332905	-6.690214	row4
...	...	...	...	...
999995	2.220502	-6.104834	-5.243481	row999995
999996	-2.681246	8.925995	4.416919	row999996

This is a visualization of the data we just generated, where the true type of each data point is represented by a unique color.



3.2.2. Data processing

Read data from a csv file into a Spark dataframe

```
from pyspark.sql import SparkSession
# spark = SparkSession.builder.getOrCreate()
spark = SparkSession.builder.appName("Python Spark Data Exploration")\
          .config("spark.some.config.option", "some-value").getOrCreate()
```

```
filename = "/Users/dengkun/Desktop/598/final project/input.csv"
df = spark.read.options(header=True, inferSchema=True).csv(filename)
```

Convert all data columns to float type

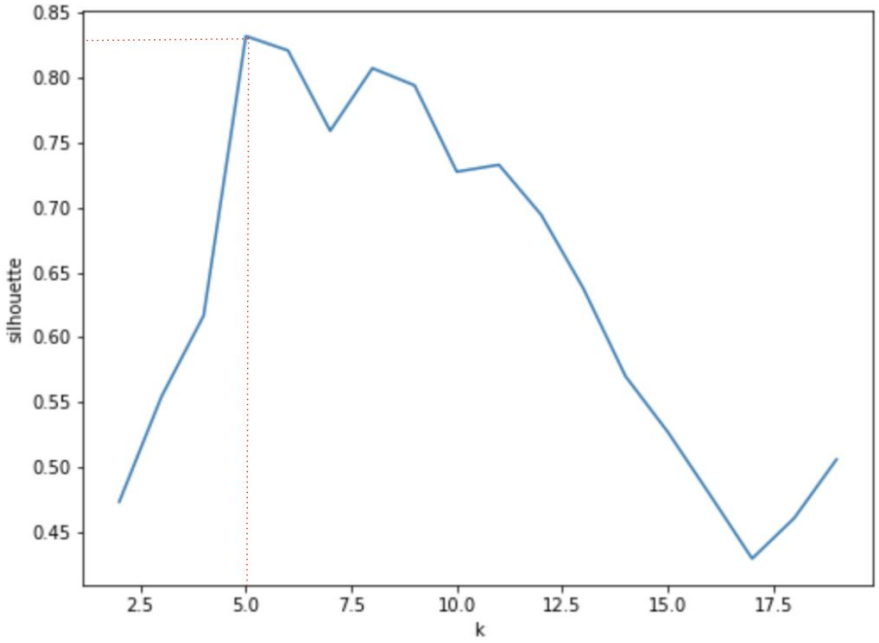
```
df_feat = df.select(*(df[c].cast("float").alias(c) for c in df.columns[1:]))
df_feat.show()
```

Create a features column to be used in the clustering

```
vecAssembler = VectorAssembler(inputCols=features, outputCol="features")
df_kmeans = vecAssembler.transform(df).select('id', 'features')
df_kmeans.show()
```

3.2.3. K-means implementation

One disadvantage of KMeans compared to more advanced clustering algorithms is that the algorithm must be told how many clusters k. So, we should try to find the best k for our dataset. To optimize k we cluster a fraction of the data for different choices of k and look for the highest value of silhouette for each k.



From the graph we could get the best k is k = 5.
Then we train the full dataset using k = 5.

```
kmeans = KMeans().setK(5).setSeed(1).setFeaturesCol("features")
model = kmeans.fit(df_kmeans.sample(False,0.1, seed=42))
predictions = model.transform(df_kmeans)
evaluator = ClusteringEvaluator()
silhouette[k] = evaluator.evaluate(predictions)
```

Compute centroids for the 5 clusters:

```
Cluster Centers:
[-5.66358195  8.17873893  3.33038004]
[-0.29157736  3.78033311 -7.52947683]
[ 4.14461596 -9.59779494  9.41049146]
[ 4.30005996 -6.31127465 -6.62276348]
[-4.97957351 -4.09357022  0.01071631]
```

Get the prediction value of each data point.

```
transformed = model.transform(df_kmeans).select('id', 'prediction')
rows = transformed.collect()
```

```
transformed.show(10)
```

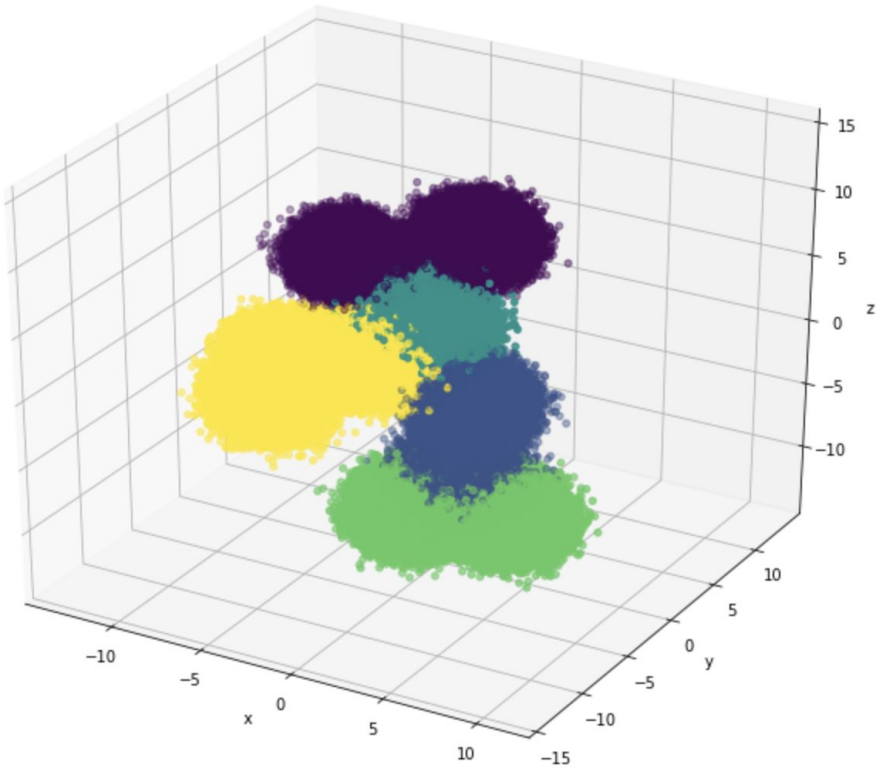
```
+----+----------+
|  id|prediction|
+----+----------+
|row0|         9|
|row1|         7|
|row2|         2|
|row3|         3|
|row4|         5|
|row5|         6|
|row6|         7|
|row7|         3|
|row8|         4|
|row9|         4|
+----+----------+
only showing top 10 rows
```

Join our prediction to the whole dataset.

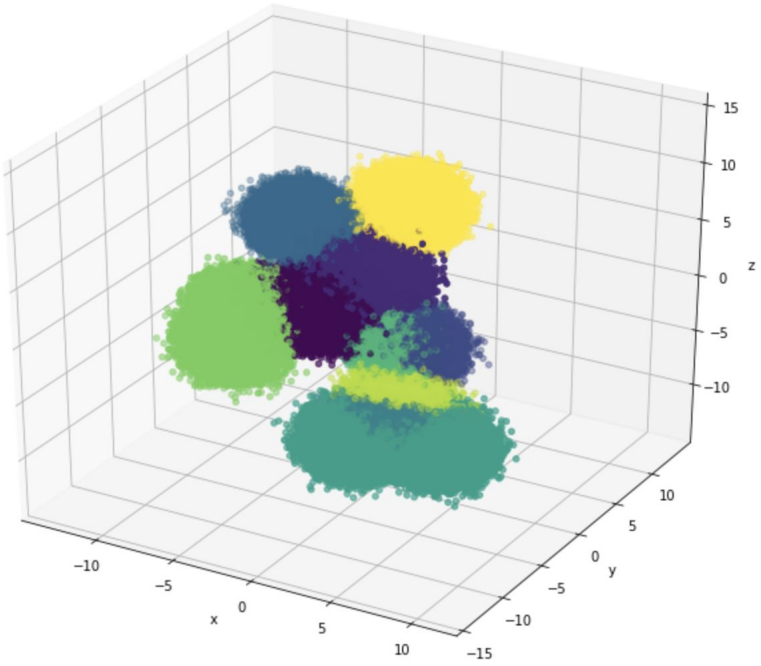
```
df_pred = transformed.join(df, 'id')
df_pred.show()
```

```
+---------+----------+-----------+----------+----------+
|       id|prediction|          x|         y|         z|
+---------+----------+-----------+----------+----------+
|row100093|         0| -1.1332687| -4.194967| 2.4416203|
|row100200|         1| -2.5354502|  8.941847| 4.6613994|
|row100356|         5|   6.814359| -5.945131|-5.7430987|
| row10044|         2|  -9.484745|  7.164065|  1.896777|
|row100578|         9|-0.43049732|  5.205617| -6.224876|
|row100888|         0| -1.1964002|-3.8600318| 1.9535162|
|row101217|         6| -7.9125886|-4.3994517|-2.0729105|
|row101689|         6|  -7.675096|-4.7346854|-3.4886727|
|row101831|         8|   -6.83433|-3.9891033| 0.2009155|
|row102220|         2|  -8.657342|  7.556673| 1.1812785|
| row10232|         0| -1.3604381|-4.4441013| 1.8473597|
```

The final step is to visually inspect the output to see if the KMeans model did a good job or not. Compared with the first figure it is clear that most clusters were indeed found, but it combines two nearby clusters to 1. From the first 10 clusters to 5 clusters.



If we use the same k (k=10) as we generate at first. We can get the visualization like this:



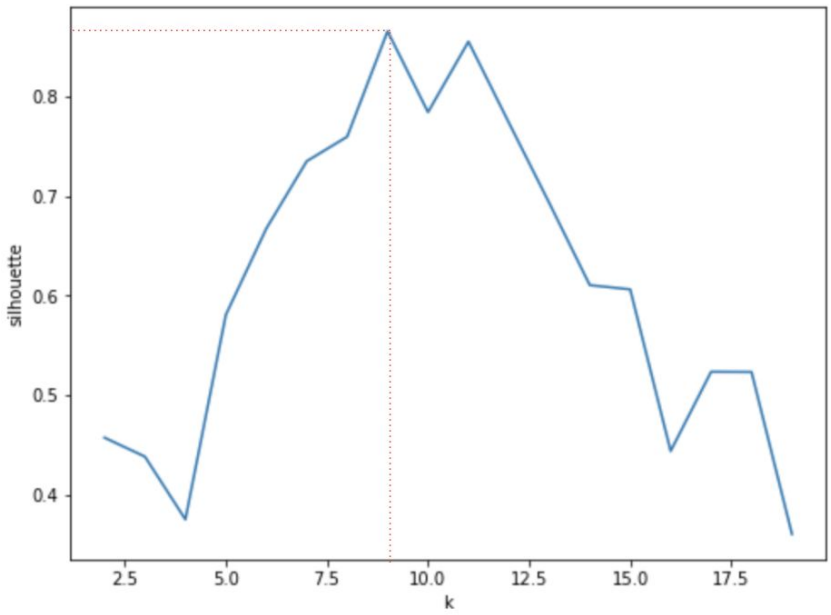
Obviously, it's not as good as k = 5. When k = 10 the below partitions are unclear classified.

3.2.4. K-means implementation to high dimensional data

It's easy for the K-means method to be used in a high dimension dataset. For example, we can also use the make_blobs method to generate a 10 dimension features dataset.

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	id
0	-3.166771	-6.656822	4.487451	-3.026211	-4.027539	2.232245	-7.045234	5.988627	-8.487368	9.334317	row0
1	-9.684210	9.229980	5.946197	-5.576625	-6.846506	-7.607507	-1.898138	0.385057	-1.131365	-5.017755	row1
2	-8.411245	-3.488244	-9.286693	7.545883	-5.168007	3.739833	-6.071231	2.385325	0.827271	-7.046513	row2
3	-3.426842	-4.028318	5.537911	-4.692147	-4.305796	-1.772540	-6.744186	3.940025	-9.321463	10.606296	row3
4	9.066450	6.851678	9.085654	5.846004	2.172495	9.023892	-7.224522	-8.324583	-7.246586	-3.314304	row4

Then, we can follow what we did above to get the best k for highest silhouette. (k=9)



Finally, get centroids for the 9 clusters.

```
Cluster Centers:
[ 2.14332851 -6.57765001 -8.69121321  8.98550475  9.3157142   6.16800035
 -3.92121984 -8.03985963  3.66850602 -1.19955819]
[-9.58623883  9.38841496  6.64098035 -5.76230542 -6.35230116 -6.34229102
 -3.90811076  0.50248413 -1.35570785 -4.17307294]
[ 9.3992149   5.50553021  8.79615082  7.88553735  1.95953629  8.44294603
 -8.2286448  -6.07735087 -9.09479548 -3.4991002 ]
[ 7.25092585  2.4654827  -3.37523309 -8.73394077 -3.78675174 -3.4923245
  4.59323184  2.74438472  7.75607733 -0.54831631]
[-2.49888046  9.01793405  4.64709909  1.97466447 -6.86596685 -6.87885853
 -8.83743109  7.3131444   2.02777823  4.16501508]
[-2.66616607 -3.65888614 -6.73525349  2.7749826  -2.84412544  4.47554106
 -4.89135097  0.34328021  1.38478608 -7.69425354]
[ 5.44643619 -6.02654085 -9.87863204  6.2918944   4.11262199  4.58712572
  5.4283074  -8.52177984 -2.81630374 -7.68735337]
[-7.61423113  4.25812005  5.23098448  1.23827501  5.4212614  -0.15410869
  0.43478931 -1.44870365 -9.49127797 -7.84771291]
[-2.2303524  -4.57572827  6.56487868 -2.86562952 -4.38127863  0.86453525
 -7.19516886  6.05691021 -8.51672449  9.74050447]
```

4. Summary and Discussion

4.1. Pros and Cons of Hierarchical Clustering

The advantages of hierarchical clustering is that no priori information about the number of clusters required. Also it's easy to implement and gives the best result in some cases. We can use hierarchical clustering for both Euclidean space and Non-Euclidean space.

The disadvantage of Hierarchical clustering is that it can't perform large-scale clustering. Because the basic algorithm for hierarchical clustering is not very efficient. At each step, compute pairwise distances between all pairs of clusters, then merge. Time complexity is $O(N^3)$. Using priority queue can reduce time complexity to $O(N^2 log\ N)$, which is still too expensive for really big datasets that do not fit in memory.

4.2. Pros and Cons of K-means Clustering

There are many advantages of k-means clustering. For example, k-means algorithm is relatively simple to implement and it can be used to large-scale data sets. Also, k-means clustering guarantees convergence and can warm-start the

positions of centroids. It can easily adapt to new examples. In addition, it generalizes to clusters of different shapes and sizes, such as elliptical clusters.
The disadvantages of k-means include that we need to choose k manually, the algorithm is dependent on initial values. It has trouble clustering data where clusters are of varying sizes and density. For k-means clustering, centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. So we need to consider removing or clipping outliers before clustering. Also, k-means clustering can only be used in Euclidean space and we cannot use it to cluster Non-Euclidean points.

4.3. Why the best k we got different with what we generate

1) We generate 1 million data points in range from -10 to 10. Too many data points in a narrow region may lead to unclear boundaries for clusters.
2) We use make_blobs function and set the standard deviation to be 1. The deviation is too large. So, the clusters which we generated will mix together.
3) Using a small deviation to generate the dataset could eliminate this problem.