CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

scikit learn

Machine Learning

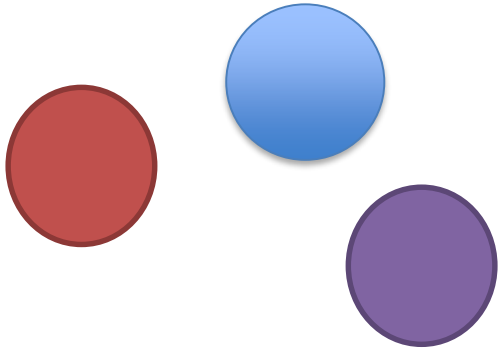
Machine Learning

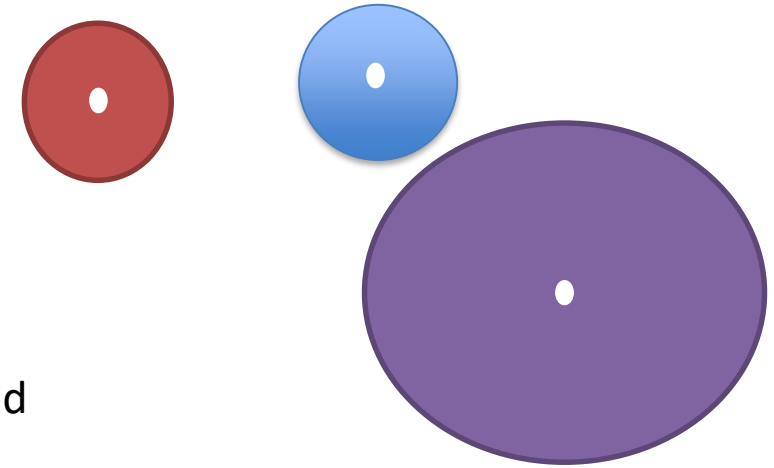
Lecture: Clustering

Ted Scully

Disadvantages of K-Means

- Converges to **suboptimal** solutions.
- Very sensitive to **initialization** strategy.
- By definition in KMeans, clusters are defined based on distance from a centroid. Therefore, this is equivalent to assuming **spherical** shaped clusters.
- Another issue with k-means is that it has real difficultly when faced with clusters of **different overall size**. For example the dispersion or spread of data points within one cluster might be much greater than the spread of data points in another cluster.
- For example, we might end up with a situation as shown on the right where we have three centroids (shown as white circles). Notice that many of the points in the purple cluster are closer to the centroid in the blue cluster and as such will be assigned to that cluster.





Density Based Clustering

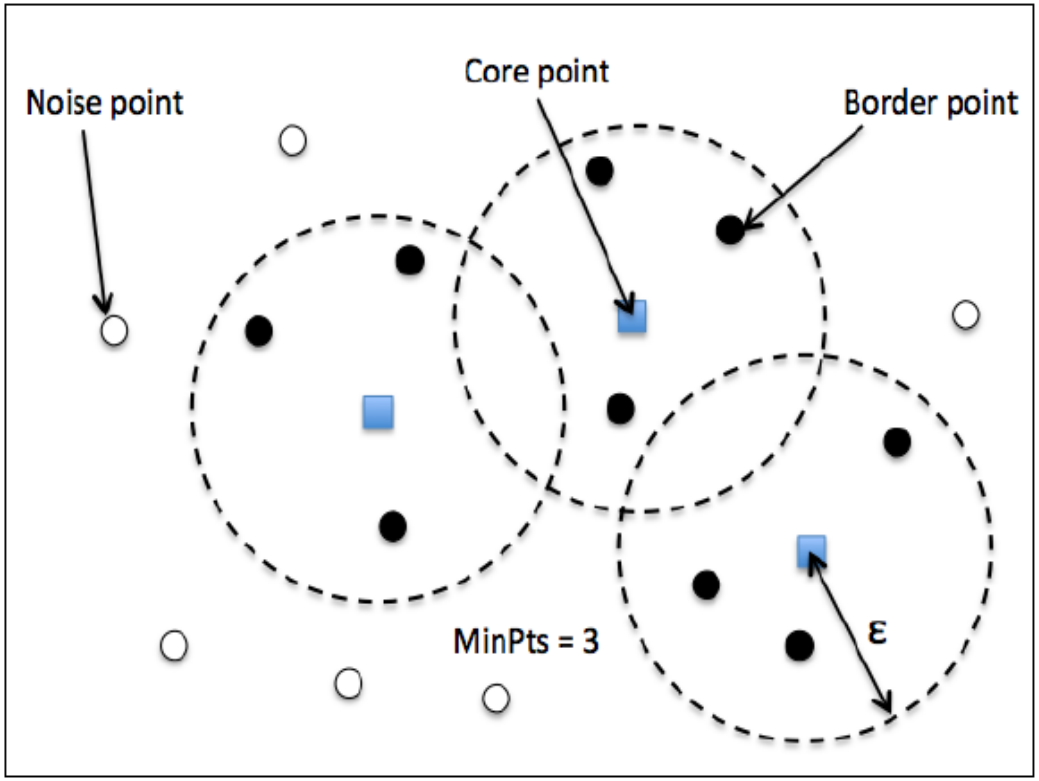
- This type of unsupervised learning focuses on **clusters as dense regions in the feature space**, separated by regions of lower density.
- Density based clustering algorithm attempt to search for areas in space that exhibit the appropriate level of density.
- Density-based spatial clustering of applications with noise (**DBSCAN**) is by far the most popular density based clustering algorithm.
- At a basic sense it groups together points that are in close proximity to each other (points with many nearby neighbours), marking as outliers points that lie alone in low-density regions
- DBScan is parameterized. MinPts and Epsilon (**ε**)

DBScan

- DBSCAN classified data points into one of three categories:
  - Core point. A point p is a core point if at least a specified number (**MinPts**) of neighbouring points fall within the specified radius **ε**.
  - Border point: A border point is a data point that has fewer neighbours than **MinPts** within **ε** but lies with the **ε** radius of a core data point.
  - All other points that are neither core nor border points are considered as noise points or outliers.

- A core data point will form a cluster with all other points that are within **ε** distance**.**
- Each **cluster** contains a minimum of **one core point**; non-core points can be part of a cluster, but they will form the periphery of the cluster as borderline points.

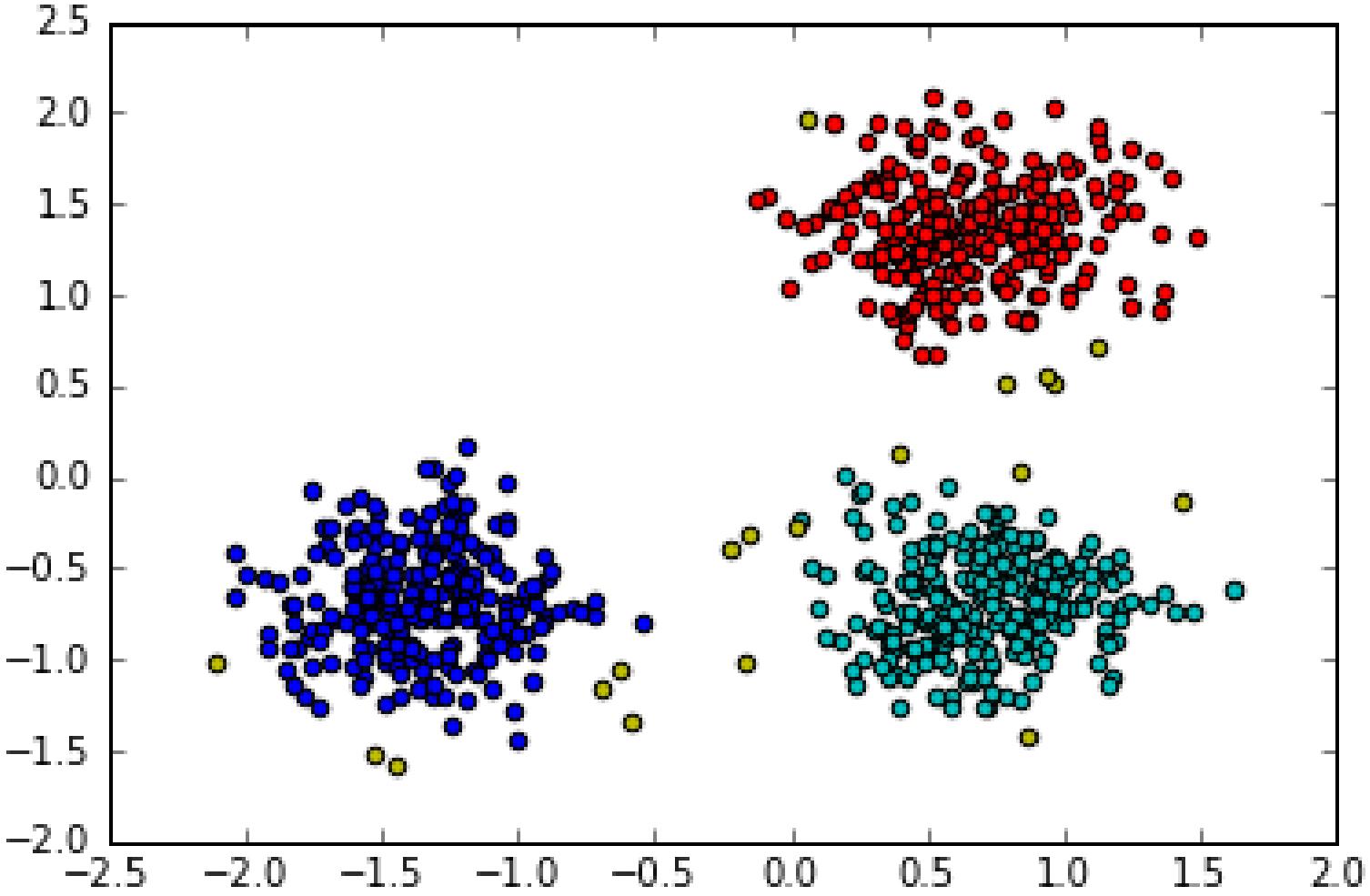
- In the example below **MinPts is set to 3**. Notice that each of the blue data points (which are core data points) have 3 neighbours within the distance defined by ε
- Notice the border points highlighted in black are within ε distance of a core point but are not classified as core points because they don't have a sufficient number of neighbours.



```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
from pylab import *

def main():
    # Generate sample data
    centers = [[1, 1], [-1, -1], [1, -1]]
    X, y = make_blobs(n_samples=750, centers=centers, cluster_std=0.3, random_state=0)
    X = StandardScaler().fit_transform(X)


    dbs = DBSCAN(eps=0.2, min_samples=5, metric="euclidean")
    dbsResults = dbs.fit_predict(X)
    plotCluster(X, dbsResults)


main()
```

```
def plotCluster(fullArray, results):

    allResults = np.unique(results)

    for currentResult in allResults:
        cluster = fullArray[results==currentResult]
        plt.scatter(cluster[:,0], cluster[:,1], c=np.random.rand(1,3))
    plt.show()
```

Applying DB Scan



Advantages and Disadvantages

- Advantages
  - Unlike K-Means DBSCAN does not require us to specify the number of clusters in advance.
  - DBScan can be highly effective in identifying **multi-variate outliers**.
  - DBSCAN can find **irregular shaped clusters** and does not assume spherical clusters. Interestingly, it can even find clusters that are completely surrounded by (but not connected to) a different cluster.

- Disadvantages
  - DBSCAN is **not deterministic**: border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data is processed.
  - Can struggle if the **dimensionality is quite large** (as it typically uses Euclidean distance it can be negatively impacted by the curse of dimensionality, which can make the data sparse).

Selecting Parameter Values

- The value for the parameter **minPts** should always be 3 or greater (typically greater than this). One rule of thumb that is often observed is that the minPts should always be greater than the number of dimensions in the data.

- Clearly the number of clusters will be very sensitive to the adopted value of $\boldsymbol{\varepsilon}$. One approach is to examine the quality of your clustering solution for various value of $\boldsymbol{\varepsilon}$.

- However, we cannot use an **elbow plot** for a clustering approach such as DB Scan.

Measuring Cluster Quality

- While we can use an **elbow plot** to estimate the cluster quality after k-means the same approach cannot be directly applied to other clustering methods (as there aren't any centroids).

- Instead a common technique used is called a **Silhouette plot**. The silhouette plot looks at the relationship between two different aspects of a clustering solution.

- More specifically, for each cluster it looks at:
  - The **density (cohesion)** of the cluster (the more dense the clusters the better)
  - The **cluster separation**, which looks at the distance between the current cluster and their nearest neighbouring cluster (again the greater the distance the better)

Building a Silhouette plot

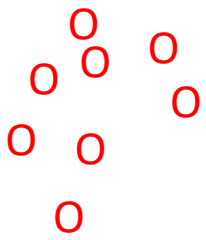
- To calculate the silhouette coefficient of a single sample in our dataset, we can apply the following three steps:

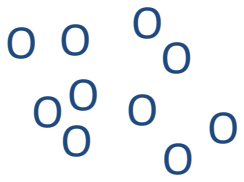
1. Calculate the **cluster cohesion $a^{(i)}$** as the **average distance** between a sample $x^{(i)}$ and all other points in the same cluster.

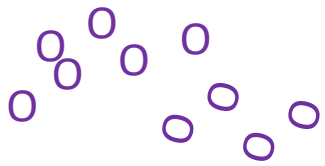
2. Calculate **the cluster separation $b^{(i)}$** from the next closest cluster as the average distance between the sample $x^{(i)}$ and all samples in the nearest cluster.

3. Calculate the **silhouette $s^{(i)}$** as the difference between cluster cohesion and separation divided by the greater of the two, as shown here:

$$s^{(i)} \begin{cases} 1 - a^{(i)} / b^{(i)}, & \text{if } a^{(i)} < b^{(i)} \\ 0 & \text{if } a^{(i)} = b^{(i)} \\ -1 + b^{(i)} / a^{(i)} & \text{if } a^{(i)} > b^{(i)} \end{cases}$$







Building a Silhouette plot

$1 - a^{(i)} / b^{(i)}$	if $a^{(i)} < b^{(i)}$
0	if $a^{(i)} = b^{(i)}$
$-1 + b^{(i)} / a^{(i)}$	if $a^{(i)} > b^{(i)}$

- We are ideally looking for clusters that have an $s^{(i)}$ value that is close to 1. Notice above that for $s^{(i)}$ to be close to 1 then $a^{(i)}$ must be small and $b^{(i)}$ must be large. That is, the cluster must be dense and must be quite a distance from the nearest neighbouring cluster.

- **If we obtain an $s^{(i)}$ value close to -1, then it means $b^{(i)}$ (the average distance to the nearest cluster) is greater than $a^{(i)}$** .(the average distance to other points within the same cluster). That is the point $x^{(i)}$ should really be clustered with the nearest neighbouring cluster.

- An $s^{(i)}$ value close to zero means that the sample data point is on the border of two clusters (as it's average distance to data points in the nearest cluster and within it's own is the same).

Creating a Silhouette Plot in Scikit Learning

- We can use sklearn.metrics.**silhouette_samples** to generate the silhouette coefficient for every data point.

- It takes as arguments:
  - The 2D NumPy array containing all the **feature data points** (standardized as usual)
  - A 1D array containing the **cluster labels** (this array indicate the clusters each data instance belongs to)
  - Specify a distance metric (typically Euclidean)
- It returns a single array with the silhouette coefficient value for each data point.

Creating a Silhouette Plot in Scikit Learning

- We can get the **mean silhouette (value)** coefficient for each cluster by averaging the silhouette coefficient of all data points assigned to each cluster.

- The following is a good rule of thumb for the average silhouette value.
  - **0.7 – 1.0** : A strong structure has been found
  - **0.51 – 0.7** : A reasonable structure has been found
  - **0.26 – 0.5** : A weak structure that could be artificial.
  - **<0.25** : No reasonable structure has been found in the data

Silhouette Plot Example

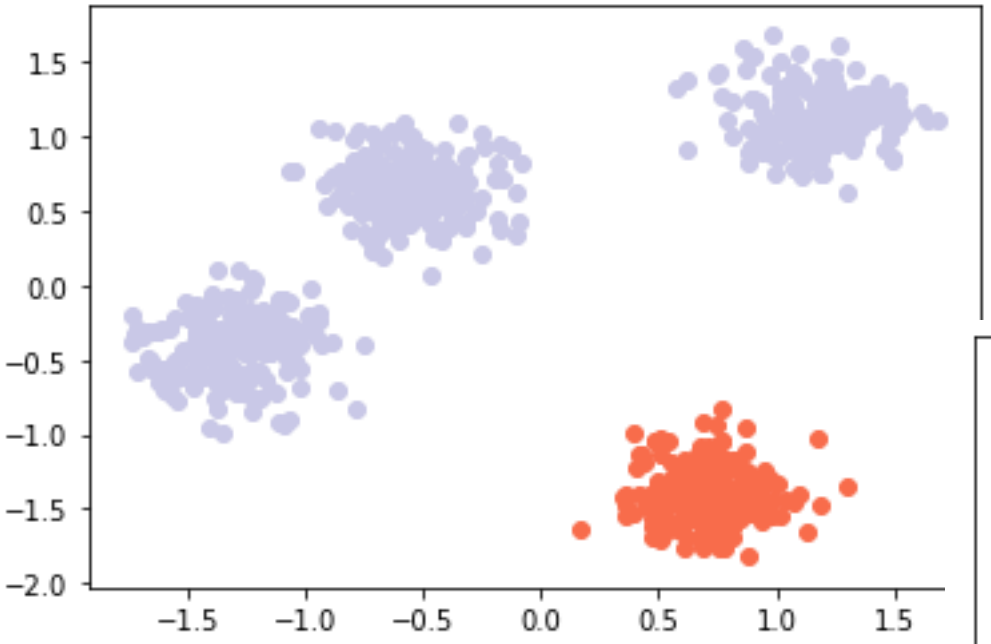
```
from sklearn.metrics import silhouette_samples

def main():
    # Generate sample data
    centers = [[0.5, 1], [-1, -1], [3, -3], [4, 2]]
    X, y = make_blobs(n_samples=750, centers=centers, cluster_std=0.4, random_state=0)
    X = StandardScaler().fit_transform(X)


    dbs = DBSCAN(eps=0.3, min_samples=5, metric="euclidean")
    dbsResults = dbs.fit_predict(X)
    plotCluster(X, dbsResults)

    # you must have more than one cluster in your results to run this code
    generateSilhouettePlot(X, dbsResults)


main()
```

Notice this time we have created 4 different clusters. The plotCluster functionality remains unchanged.

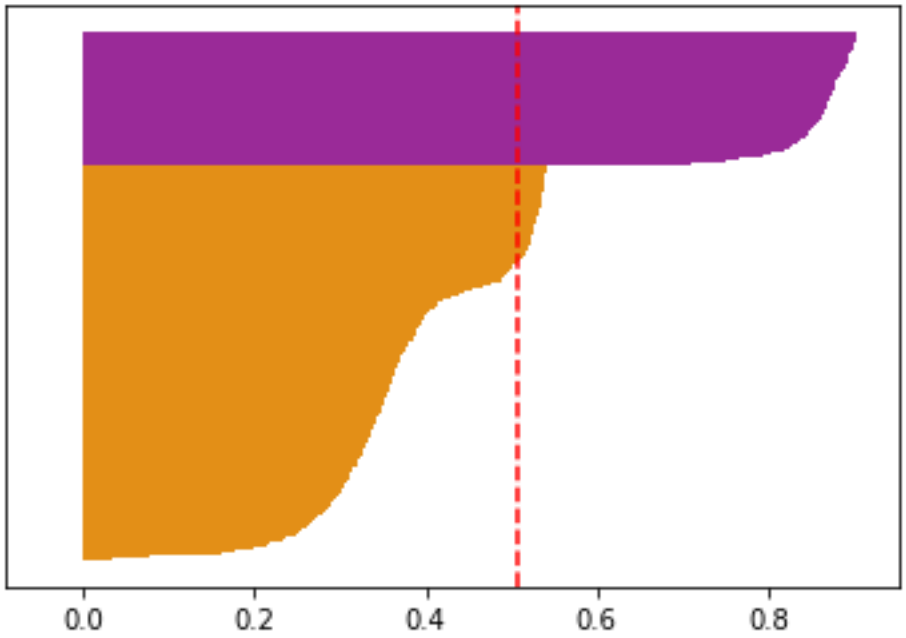
```
def generateSilhouettePlot(fullArray, results):

    silVals = silhouette_samples(fullArray, results, metric="euclidean")
    cluster_labels = np.unique(results)

    y_ax_lower, y_ax_upper = 0, 0
    yticks = []

    for c in cluster_labels:
        # extract silhouette coef's for current cluster c
        c_silhouette_vals = silVals[results == c]
        c_silhouette_vals.sort()
        y_ax_upper += len(c_silhouette_vals)
        color = np.random.rand(1,3)
        plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.0, edgecolor='none',
                        color=color)

        y_ax_lower += len(c_silhouette_vals)

    silhouette_avg = np.mean(silVals)
    print silhouette_avg
    plt.axvline(silhouette_avg, color="red", linestyle="--")
    plt.yticks(yticks, cluster_labels + 1)
```

Results for DBSCAN(eps=1.0, min_samples=5, metric="euclidean")

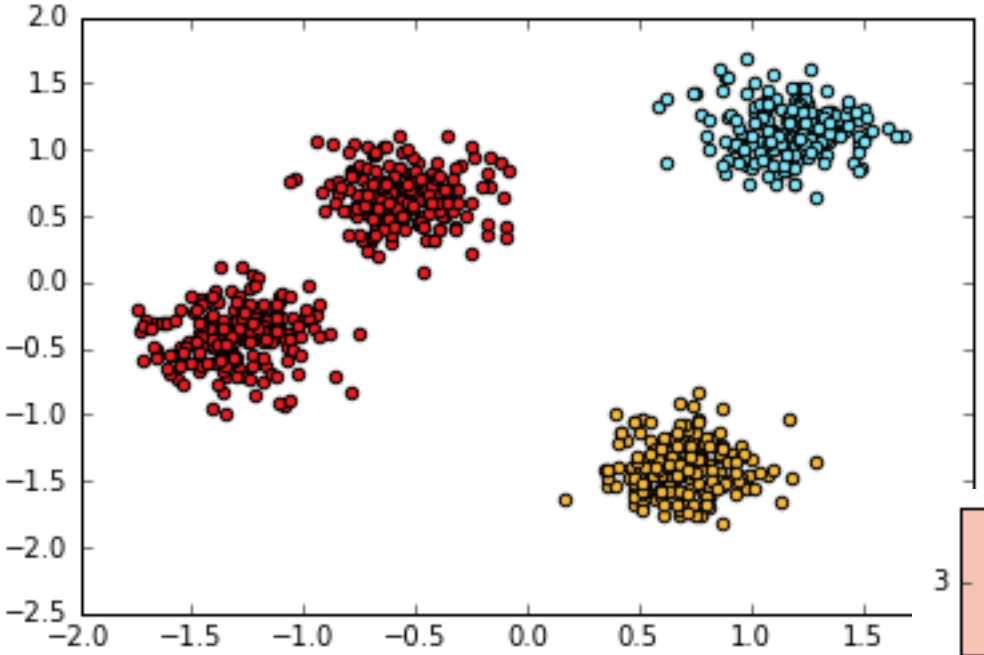
Results for DBSCAN(eps=1.0, min_samples=5, metric="euclidean")



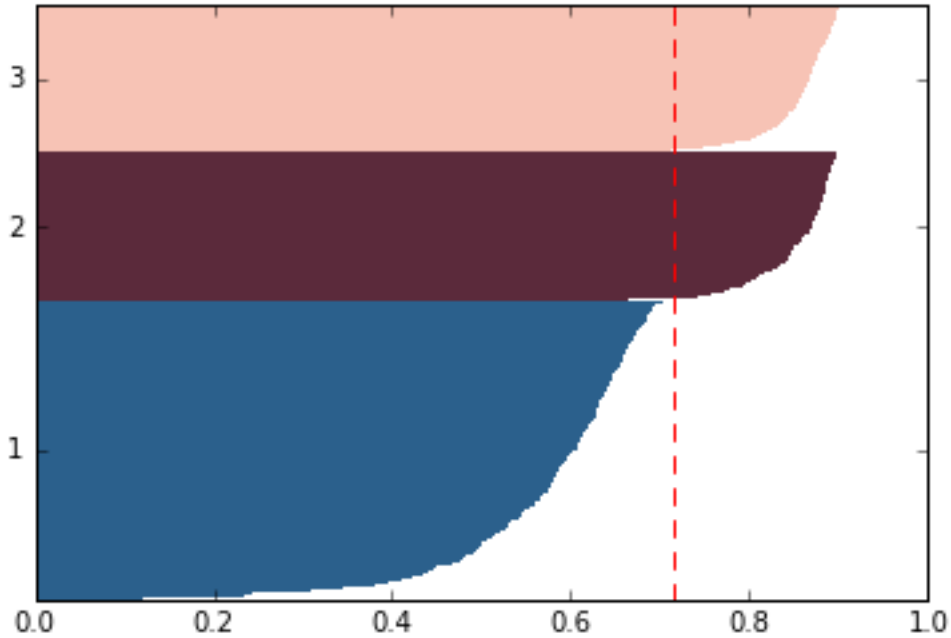
Average Silhouette Coefficient 0.507



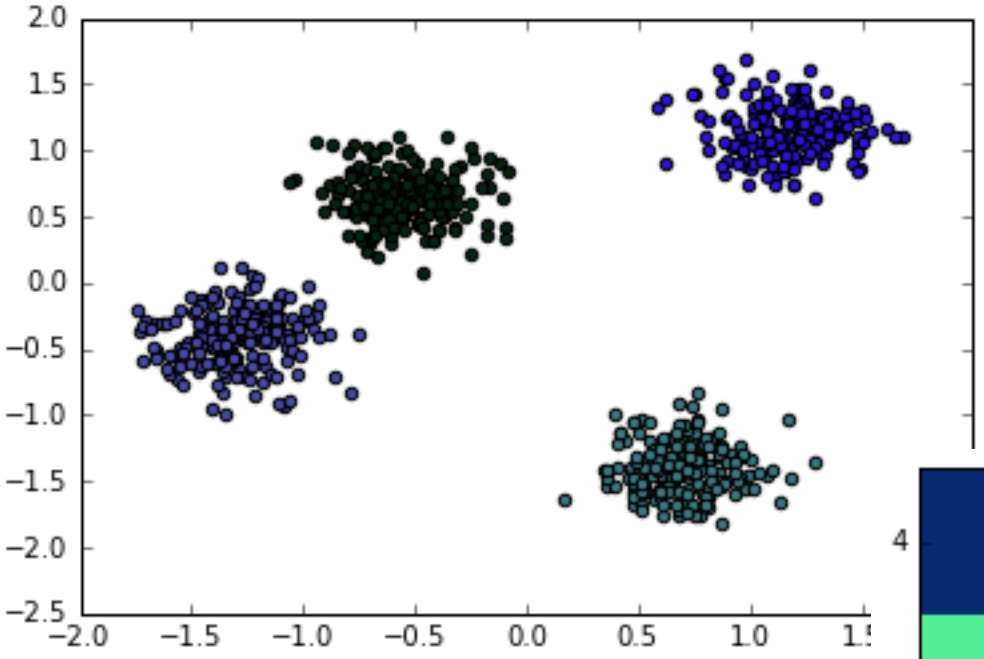
Results for DBSCAN(eps=0.7, min_samples=5, metric="euclidean")



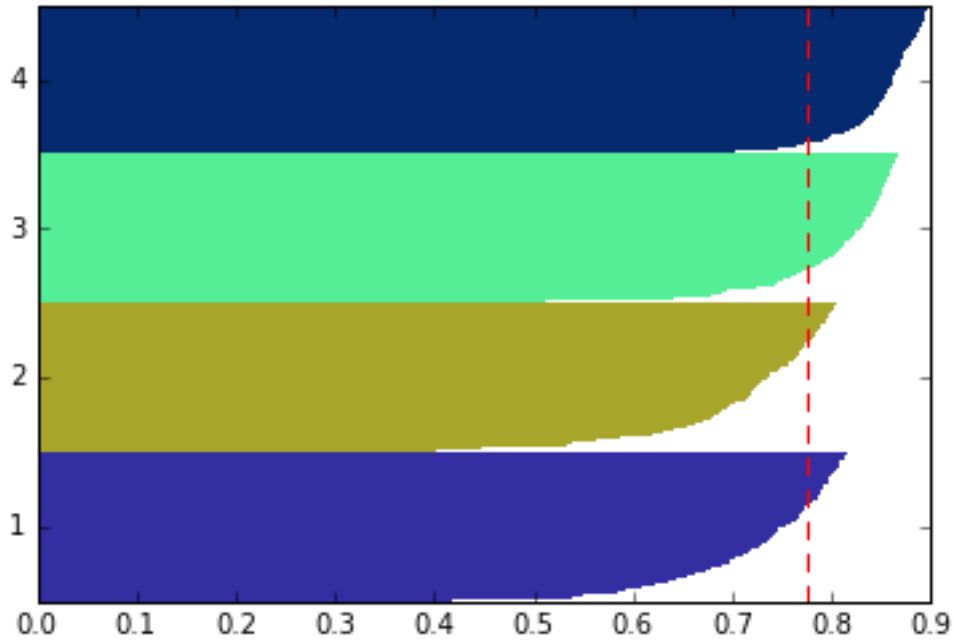
Average Silhouette Coefficient 0.715



Results for DBSCAN(eps=0.3, min_samples=5, metric="euclidean")



Average Silhouette Coefficient 0.775



Hierarchical Clustering

- There are two main approaches to hierarchical clustering
  - **Agglomerative**: Start with each individual sample as a cluster and merge the closest pairs of clusters until only one cluster remains.
  - **Divisive**: Start with one cluster that encompasses all samples and iteratively split the cluster into smaller clusters until each cluster contains just one sample.

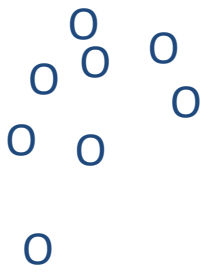
- In Agglomerative Clustering we start with a collection C of n singleton clusters.
  - Each cluster contains only a single data point $c^i = \{x^i\}$
  - Repeat until only one cluster remains:
    1. Identify the pair of clusters c^i and c^j such *min* $D(c^i, c^j)$
    2. Merge the clusters c^i and c^j into a new cluster c^{i+j}
    3. Add c^{i+j} to C and remove c^i and c^j

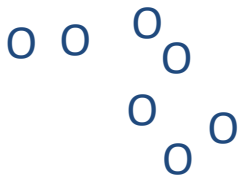
Hierarchical Clustering

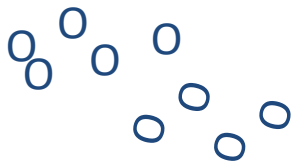
- In the previous slide we referred to a distance metric that we attempted to minimize $D(c^i, c^j)$

- There are many methods used for measuring similarity between clusters. Two of the most common are:
  - Single linkage: We compute the distances between the **most similar** members for each pair of clusters and merge the two clusters for which the distance between the most similar members is smallest.
  - Complete linkage: Instead of comparing the most similar members in each cluster we compare the **most dissimilar**.
  - There are many other distance metrics used in hierarchical clustering such the average distance between the data points of each cluster.

- Notice this process is completely deterministic.







Hierarchical Agglomerative Clustering

- Scikit Learn has an agglomerative clustering available in **sklearn.cluster.AgglomerativeClustering**

- It takes the following as arguments:
  - **n_clusters**: Step merge process once we have reached this number of clusters (remember this is a bottom up process).
  - **Affinity**: Metric used to calculate linkage. Typically Euclidean.
  - **Linkage** : Defines the linkage metric.

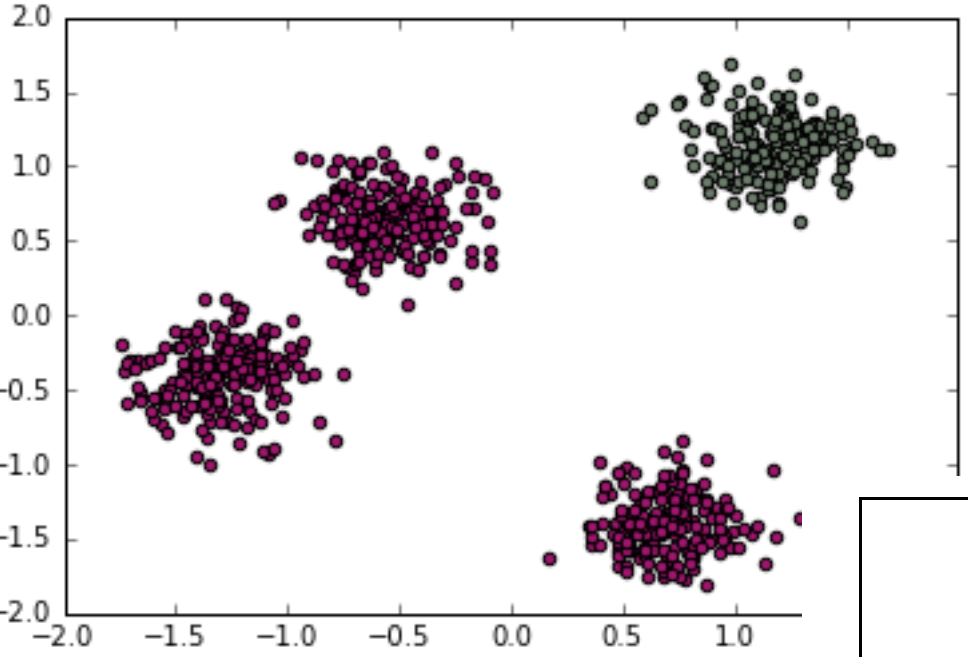
- It returns a single array that specifies the cluster index for each data point.

- Notice we must specify the n_clusters to retain.
- How do we decide the number of clusters?
- We will return to the same dataset we used for DBScan for illustration.

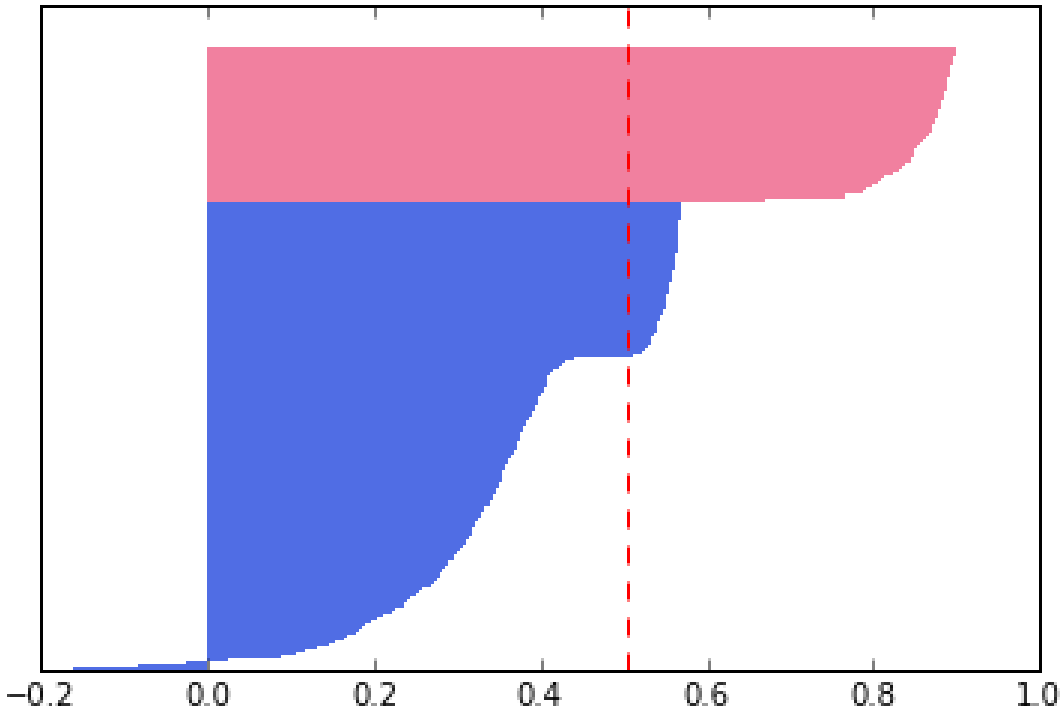
Agglomerative Clustering – Complete Linkage

```
from sklearn.cluster import AgglomerativeClustering


def main():
    # Generate sample data
    centers = [[0.5, 1], [-1, -1], [3, -3], [4, 2]]
    X, y = make_blobs(n_samples=750, centers=centers, cluster_std=0.4, random_state=0)
    X = StandardScaler().fit_transform(X)


    clusters = AgglomerativeClustering(n_clusters = 2, affinity="euclidean", linkage='complete')
    results = clusters.fit_predict(X)
    plotCluster(X, results)

    # you must have more than one cluster in your results to run this code
    generateSilhouettePlot(X, results)
```

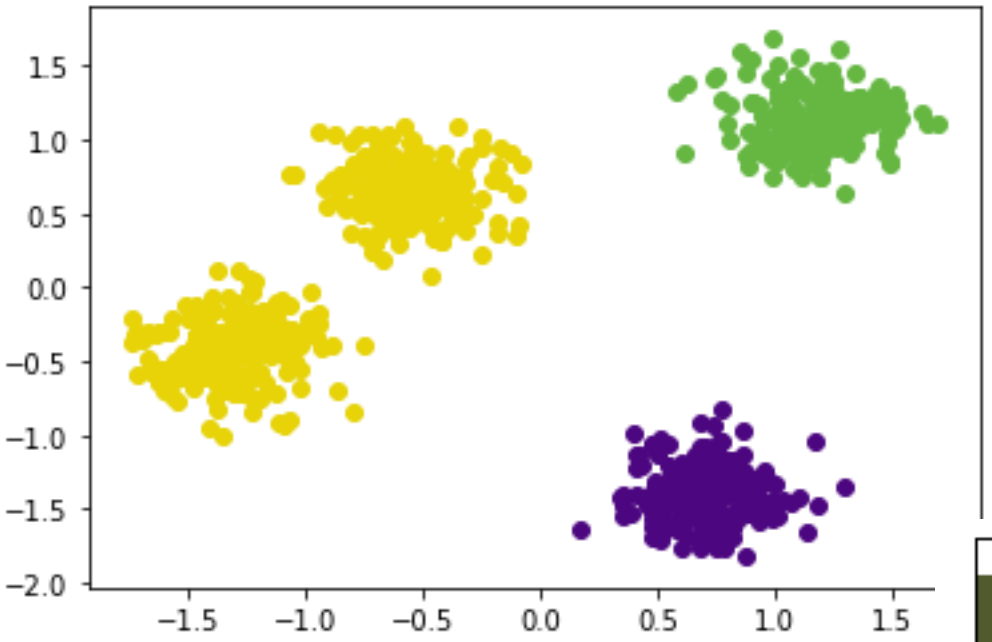
AgglomerativeClustering(n_clusters = 2, affinity="euclidean", linkage='complete')



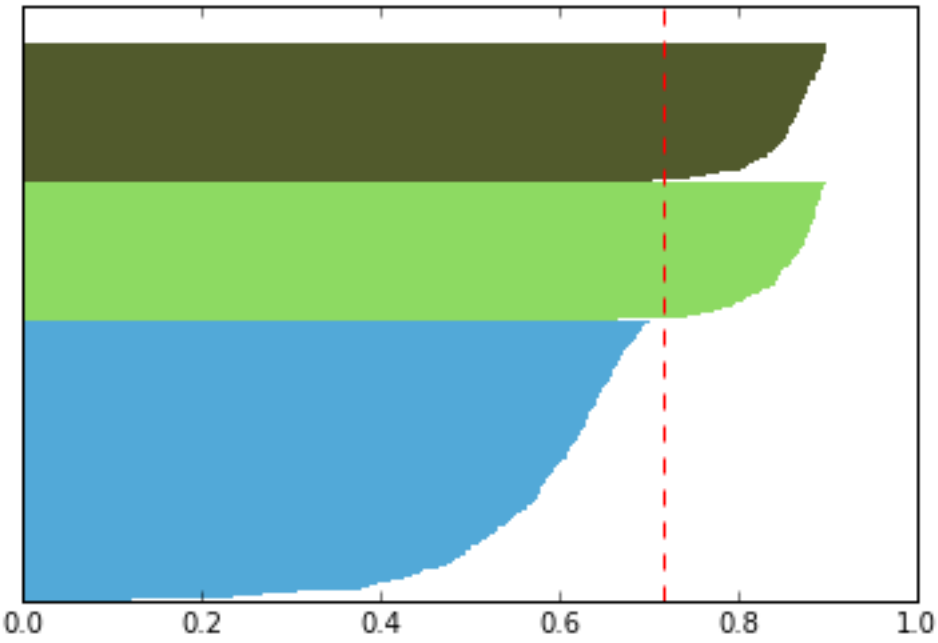
Average Silhouette Coefficient 0.503



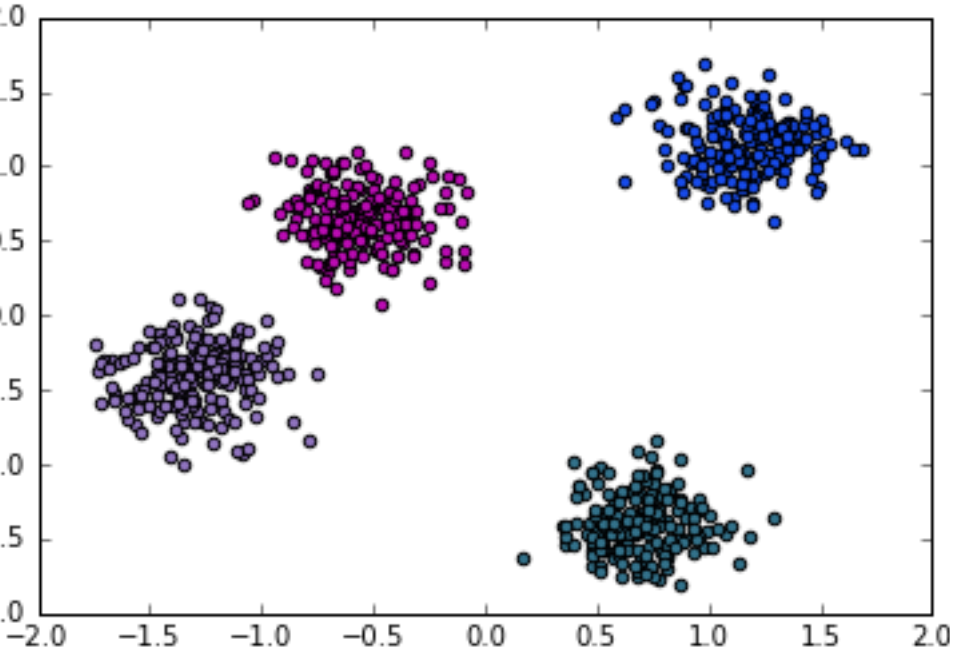
AgglomerativeClustering(n_clusters = 3, affinity="euclidean", linkage='complete')



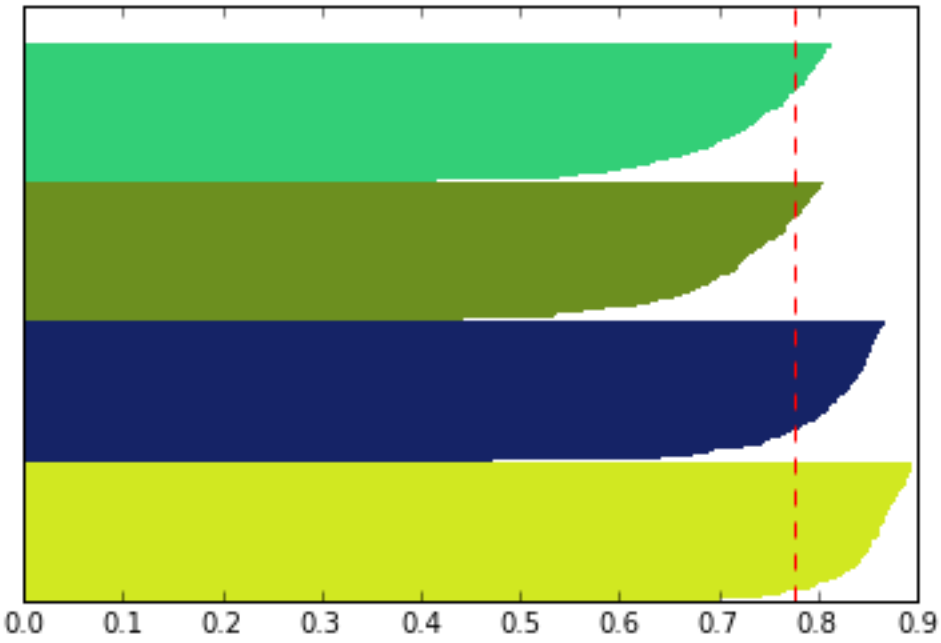
Average Silhouette Coefficient 0.715



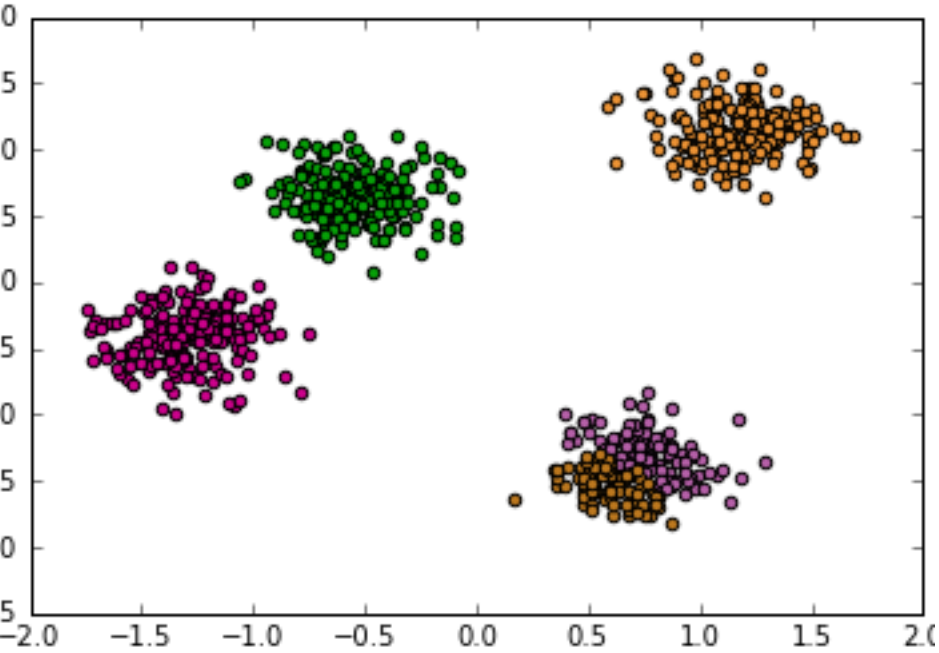
AgglomerativeClustering(n_clusters = 4, affinity="euclidean", linkage='complete')



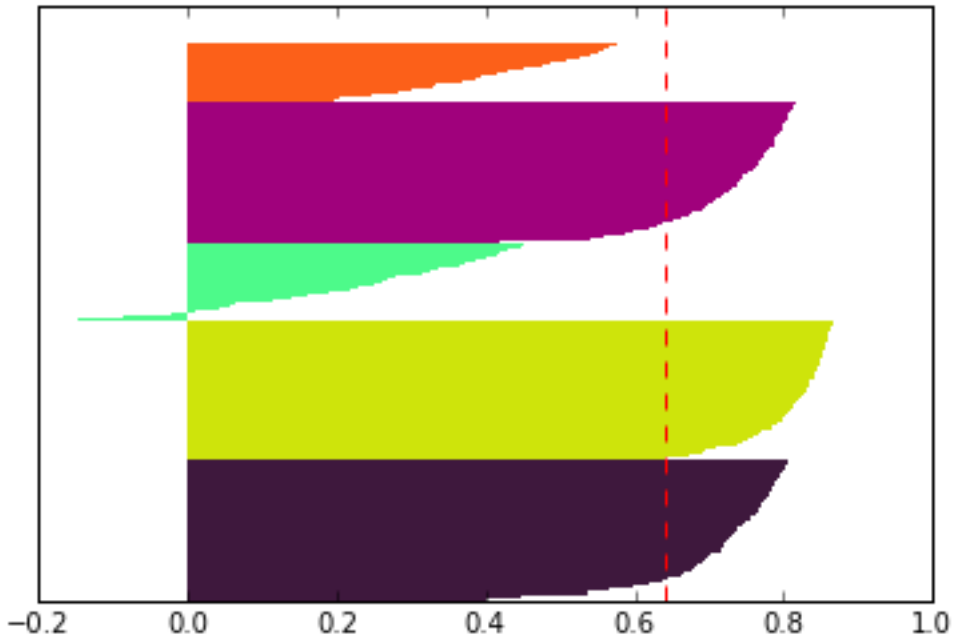
Average Silhouette Coefficient 0.775



AgglomerativeClustering(n_clusters = 5, affinity="euclidean", linkage='complete')



Average Silhouette Coefficient 0.640



Unsupervised Learning (Clustering)

- Unsupervised learning we are taking feature data with the objective of identifying underlying patterns in that data.

- We looked 3 different categories:
  - Partitioning methods (k-Means, k-Means++)
  - Density-based methods (DB Scan)
  - Hierarchical methods (AgglomerativeClustering)

- Metrics
  - Elbow Plot (Partitioning only)
  - Silhouette