Cluster Analysis

Agenda

1. Introduction to Unsupervised Learning
2. Clustering
3. Distance or Similarity Function
4. Types of Clustering
5. Partitioning Method
6. Clustering as an Optimization Problem¶
7. Hierarchial Clustering
8. Density Based Clustering - DBSCAN
9. Measuring Performance of Clusters

Ackknowledgement

Machine Learning Git Codebook: https://github.com/zekelabs/data-science-complete-tutorial (https://github.com/zekelabs/data-science-complete-tutorial)

1. Introduction to Unsupervised Learning

- Unsupervised Learning is a type of Machine learning to draw inferences from unlabelled datasets.
- Model tries to find relationship between data.
- Most common unsupervised learning method is clustering which is used for exploratory data analysis to find hidden patterns or grouping in data

2. Clustering

- A learning technique to group a set of objects in such a way that objects of same group are more similar to each other than from objects of other group.
- Applications of clustering are as follows
  - Automatically organizing the data
  - Labeling data
  - Understanding hidden structure of data
  - News Cloustering for grouping similar news together
  - Customer Segmentation
  - Suggest social groups

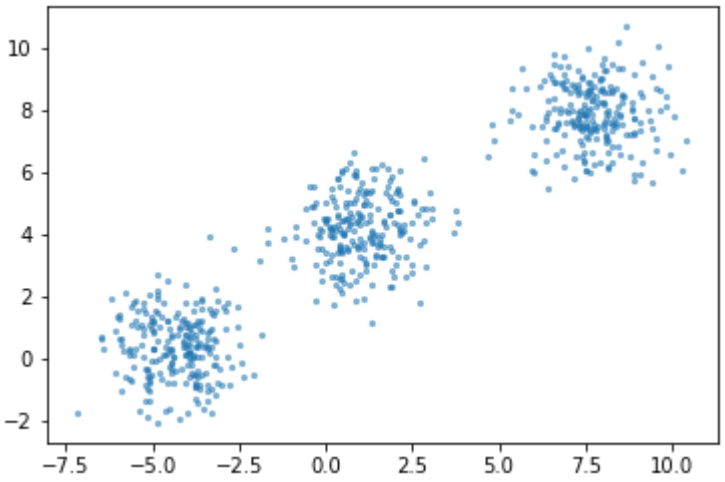
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from sklearn.datasets import make_blobs
```

- Generating natural cluster

In [3]:
```
X,y = make_blobs(n_features=2, n_samples=700, centers=3, cluster_std=1, random_s
plt.scatter(X[:,0], X[:,1], s=5, alpha=.5)
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x1d916da6588>
```



3. Distance or Similarity Function

- Data belonging to same cluster are similar & data belonging to different cluster are different.
- We need mechanisms to measure similarity & differences between data.
- This can be achieved using any of the below techniques.
  - Minkowiski breed of distance calculation:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

More complicated distances

- Manhatten (p=1), Euclidian (p=2)
- Cosine: Suited for text data

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

In [4]:
```
from sklearn.metrics.pairwise import euclidean_distances,cosine_distances,manhat
```

In [5]:
```
X = [[0, 1], [1, 1]]
```

In [6]:
```
euclidean_distances(X, X)
```

Out[6]:
```
array([[0., 1.],
       [1., 0.]])
```

In [7]:
```
euclidean_distances(X, [[0,0]])
```

Out[7]:
```
array([[1.        ],
       [1.41421356]])
```

In [8]:
```
cosine_distances(X,X)
```

Out[8]:
```
array([[0.        , 0.29289322],
       [0.29289322, 0.        ]])
```

In [9]:
```
manhattan_distances(X,X)
```

Out[9]:
```
array([[0., 1.],
       [1., 0.]])
```

4. Types of Clustering

- Partitioning methods
  - Partitions n data into k partitions
  - Initially, random partitions are created & gradually data is moved across different partitions.
  - It uses distance between points to optimize clusters.
  - KMeans & Meanshift are examples of Partitioning methods
- Hierarchical methods
  - These methods does hierarchical decomposition of datasets.
  - One approach is, assume each data as cluster & merge to create a bigger cluster
  - Another approach is start with one cluster & continue splitting
- Density-based methods
  - All above techniques are distance based & such methods can find only spherical clusters and not suited for clusters of other shapes.
  - Continue growing the cluster untill the density exceeds certain threashold.

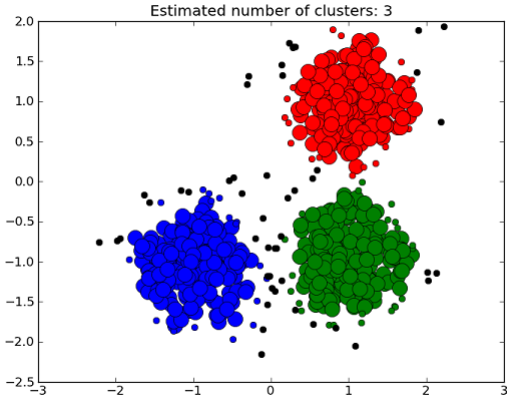
5. Partitioning Method

KMeans

- Minimizing creteria : within-cluster-sum-of-squares.

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

- The centroids are chosen in such a way that it minimizes within cluster sum of squares.
- The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean of the samples in the cluster. μ



KMeans Algorithm

1. Initialize k centroids.
2. Assign each data to the nearest centroid, these step will create clusters.
3. Recalculate centroid - which is mean of all data belonging to same cluster.
4. Repeat steps 2 & 3, till there is no data to reassign a different centroid.

Animation to explain algo - http://tech.nitoyon.com/en/blog/2013/11/07/k-means/ (http://tech.nitoyon.com/en/blog/2013/11/07/k-means/)

6. Clustering as an Optimization Problem

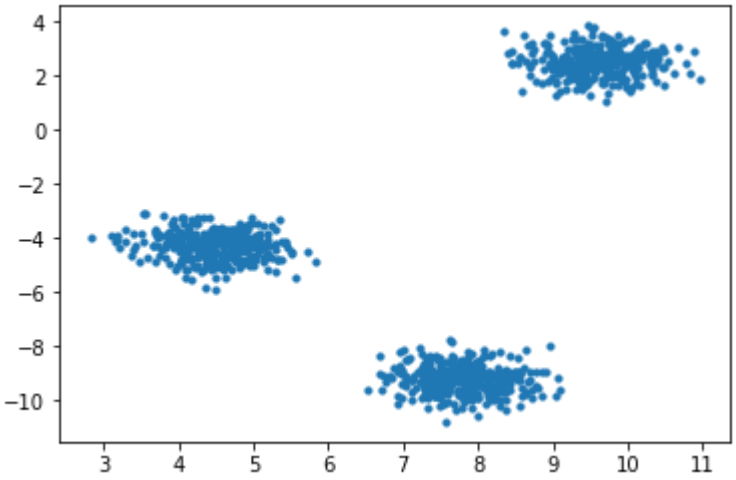
- Maximize inter-cluster distances
- Minimize intra-cluster distances

In [10]:
```
from sklearn.datasets import make_blobs, make_moons
```

In [11]:
```
X,y = make_blobs(n_features=2, n_samples=1000, cluster_std=.5)
```

In [12]:
```
plt.scatter(X[:,0], X[:,1],s=10)
```

Out[12]:
```
<matplotlib.collections.PathCollection at 0x1d916ec6a20>
```



In [13]:
```
from sklearn.cluster import KMeans, MeanShift
```

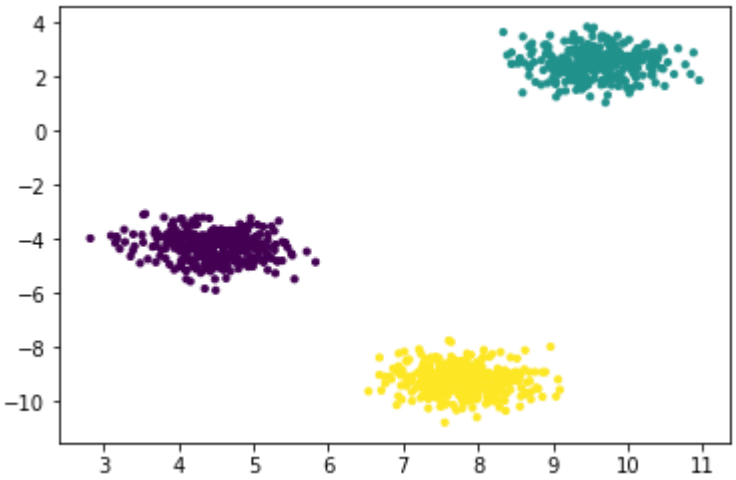
In [14]:
```
kmeans = KMeans(n_clusters=3)
```

In [15]:
```
kmeans.fit(X)
```

Out[15]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [16]:
```
plt.scatter(X[:,0], X[:,1],s=10, c=kmeans.predict(X))
```

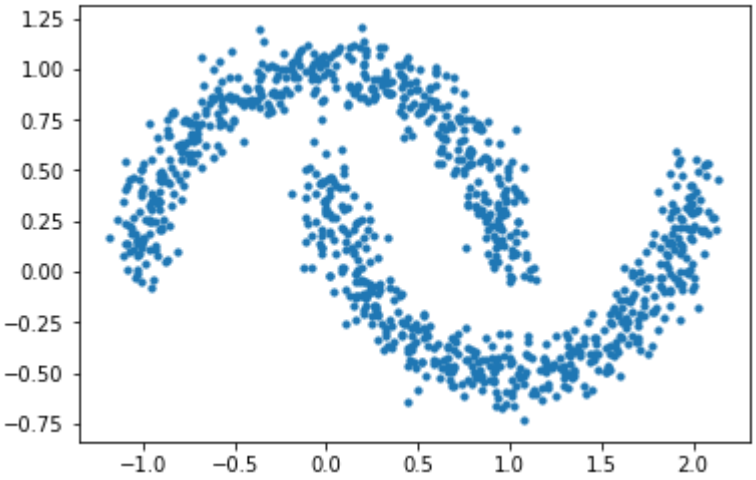
Out[16]:
```
<matplotlib.collections.PathCollection at 0x1d91774d048>
```



In [17]:
```
X, y = make_moons(n_samples=1000, noise=.09)
```

In [18]:
```
plt.scatter(X[:,0], X[:,1],s=10)
```

Out[18]:
```
<matplotlib.collections.PathCollection at 0x1d9177b7ef0>
```



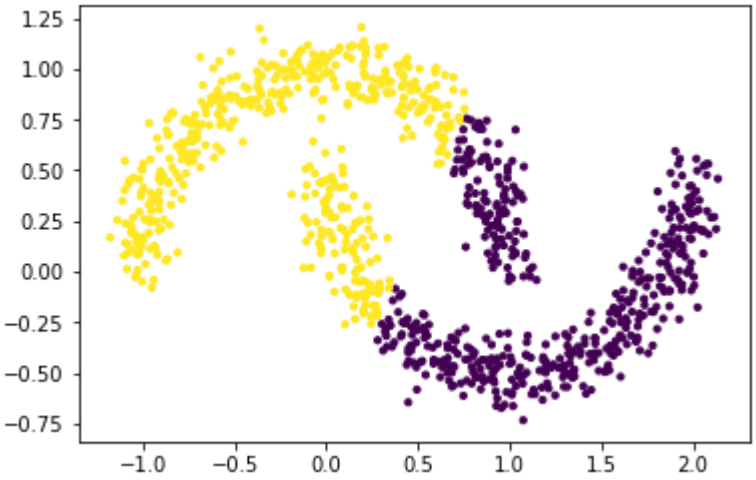
In [19]:
```
kmeans = KMeans(n_clusters=2)
```

In [20]:
```
kmeans.fit(X)
```

Out[20]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [21]:
```
plt.scatter(X[:,0], X[:,1],s=10, c=kmeans.predict(X))
```

Out[21]:
```
<matplotlib.collections.PathCollection at 0x1d9178305c0>
```



Limitations of KMeans

- Assumes that clusters are convex & behaves poorly for elongated clusters.
- Probability for participation of data to multiple clusters.
- KMeans tries to find local minima & this depends on init value.

7. Hierarchial Clustering

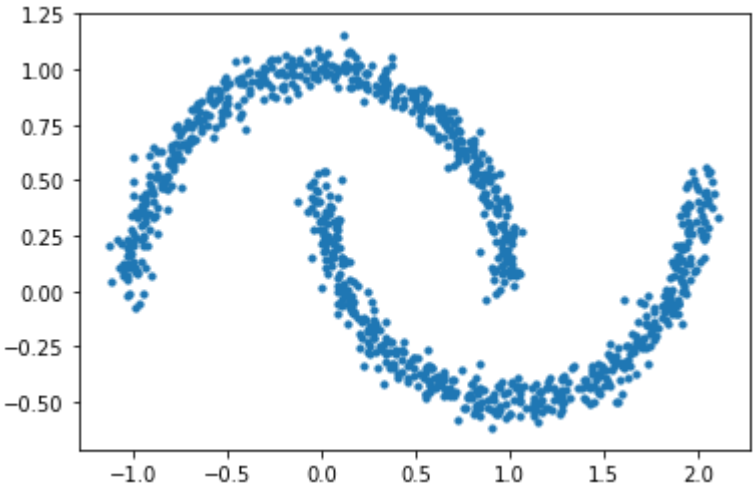
- A method of clustering where you combine similar clusters to create a cluster or split a cluster into smaller clusters such they now they become better.
- Two types of hierarchaial Clustering
  - Agglomerative method, a botton-up approach.
  - Divisive method, a top-down approach.

Agglomerative method

- Start with assigning one cluster to each data.
- Combine clusters which have higher similarity.
- Differences between methods arise due to different ways of defining distance (or similarity) between clusters. The following sections describe several agglomerative techniques in detail.
  - Single Linkage Clustering
  - Complete linkage clustering
  - Average linkage clustering
  - Average group linkage

In [22]:
```
X, y = make_moons(n_samples=1000, noise=.05)
plt.scatter(X[:,0], X[:,1],s=10)
```

Out[22]:
```
<matplotlib.collections.PathCollection at 0x1d917896908>
```



In [23]:
```
from sklearn.cluster import AgglomerativeClustering
```

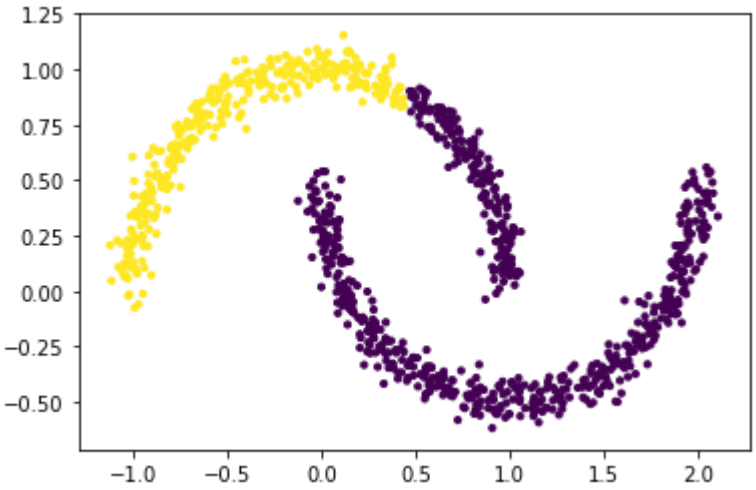
In [24]:
```
agc = AgglomerativeClustering(linkage='ward')
```

In [25]:
```
agc.fit(X)
```

Out[25]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')
```

In [26]:
```
plt.scatter(X[:,0], X[:,1],s=10,c=agc.labels_)
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x1d917951240>
```



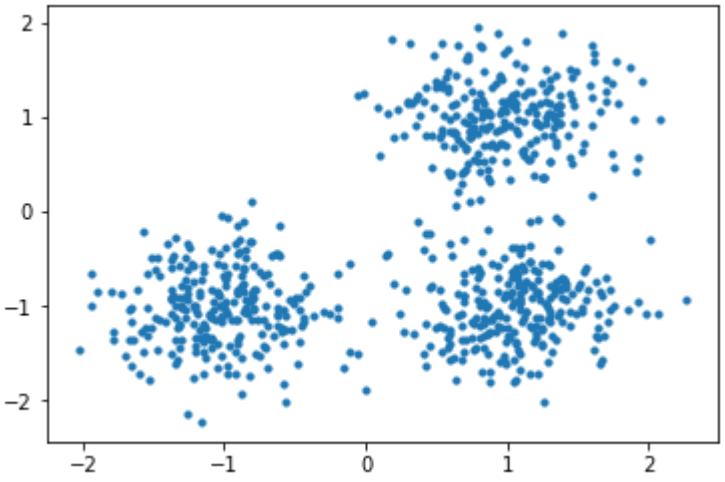
8. Density Based Clustering - DBSCAN

See an animation at https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/ (https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/)

In [27]:
```
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                            random_state=0)
```

In [28]:
```
plt.scatter(X[:,0], X[:,1],s=10)
```

Out[28]:
```
<matplotlib.collections.PathCollection at 0x1d9179b1da0>
```



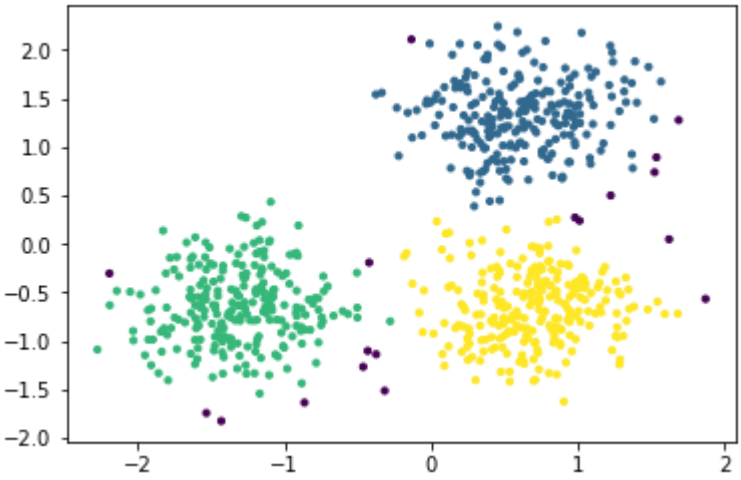
In [29]:
```
from sklearn.cluster import DBSCAN
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)

db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
```

In [31]:
```
plt.scatter(X[:,0], X[:,1],s=10,c=labels)
```

Out[31]:
```
<matplotlib.collections.PathCollection at 0x1d917a166a0>
```



9. Measuring Performance of Clusters

- Two forms of evaluation
- supervised, which uses a ground truth class values for each sample.
  - completeness_score
  - homogeneity_score
- unsupervised, which measures the quality of model itself
  - silhoutte_score
  - calinski_harabaz_score

completeness_score

- A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster.
- Accuracy is 1.0 if data belonging to same class belongs to same cluster, even if multiple classes belongs to same cluster

In [32]:
```
from sklearn.metrics.cluster import completeness_score
```

In [33]:
```
completeness_score( labels_true=[10,10,11,11],labels_pred=[1,1,0,0])
```

Out[33]:
```
1.0
```

- The acuracy is 1.0 because all the data belonging to same class belongs to same cluster

In [34]:
```
completeness_score( labels_true=[11,22,22,11],labels_pred=[1,0,1,1])
```

Out[34]:
```
0.3836885465963443
```

- The accuracy is .3 because class 1 - [11,22,11], class 2 - [22]

In [35]:
```
print(completeness_score([10, 10, 11, 11], [0, 0, 0, 0]))
```

```
1.0
```

homogeneity_score

- A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.

In [36]:
```
from sklearn.metrics.cluster import homogeneity_score
```

In [37]:
```
homogeneity_score([0, 0, 1, 1], [1, 1, 0, 0])
```

Out[37]:
```
1.0
```

In [38]:
```
homogeneity_score([0, 0, 1, 1], [0, 1, 2, 3])
```

Out[38]:
```
0.9999999999999999
```

In [39]:
```
homogeneity_score([0, 0, 0, 0], [1, 1, 0, 0])
```

Out[39]:
```
1.0
```

- Same class data is broken into two clusters

silhoutte_score

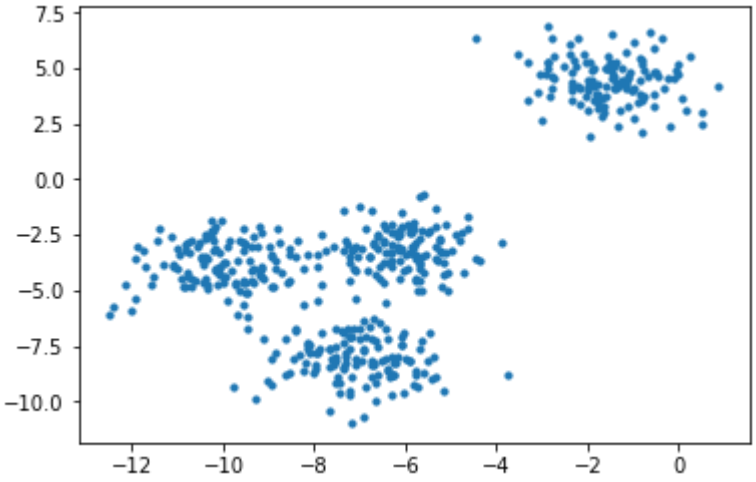
- The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample.
- The Silhouette Coefficient for a sample is (b - a) / max(a, b). To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of.

Example : Selecting the number of clusters with silhouette analysis on KMeans clustering

In [40]:
```
from sklearn.datasets import make_blobs
X, Y = make_blobs(n_samples=500,
                  n_features=2,
                  centers=4,
                  cluster_std=1,
                  center_box=(-10.0, 10.0),
                  shuffle=True,
                  random_state=1)
```

In [41]:
```
plt.scatter(X[:,0],X[:,1],s=10)
```

Out[41]:
```
<matplotlib.collections.PathCollection at 0x1d917a86748>
```



In [42]:
```
range_n_clusters = [2, 3, 4, 5, 6]
```

In [43]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [44]:
```
for n_cluster in range_n_clusters:
    kmeans = KMeans(n_clusters=n_cluster)
    kmeans.fit(X)
    labels = kmeans.predict(X)
    print (n_cluster, silhouette_score(X,labels))
```

```
2 0.7049787496083262
3 0.5882004012129721
4 0.6505186632729437
5 0.5745566973301872
6 0.4504666294372765
```

- Optimal number of clusters seems to be 2

calinski_harabaz_score

- The score is defined as ratio between the within-cluster dispersion and the between-cluster dispersion.

In [45]:
```
from sklearn.metrics import calinski_harabaz_score

for n_cluster in range_n_clusters:
    kmeans = KMeans(n_clusters=n_cluster)
    kmeans.fit(X)
    labels = kmeans.predict(X)
    print (n_cluster, calinski_harabaz_score(X,labels))
```

```
2 1604.112286409658
3 1809.991966958033
4 2704.4858735121097
5 2282.1610642050177
6 2043.5729191612152
```