MET CS 688
Clustering II

Leila Ghaedi – Fall 2024

Creator: cemagraphics | Credit: Getty Images/iStockphoto

Clustering Evaluation Methods

Clustering Evaluation Methods

- What settings would yield the best results for cluster parameters? Which clustering algorithm demonstrates the highest accuracy when applied to our dataset? Additionally, which algorithm proves to be more efficient in terms of resources?
- Various quantitative (objective, algorithm-generated) and qualitative (subjective, involving human annotations) methods are employed to evaluate clustering results. However, there is no optimum method, and each of these evaluation approaches has its own limitations.
- Methods for evaluation include:
    - Intrinsic Evaluation
    - Extrinsic Evaluation

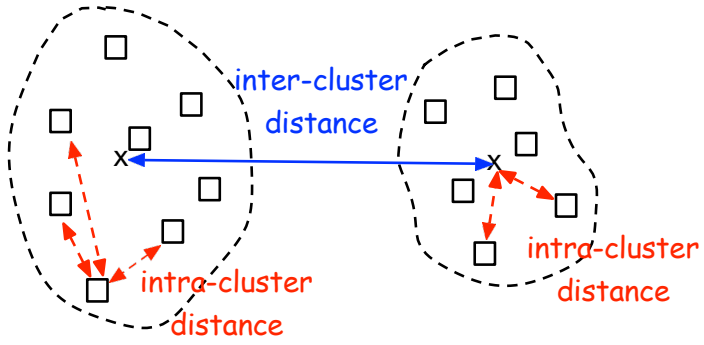
Intrinsic vs Extrinsic Evaluation Methods

- If ground truth is available, it can be used by the extrinsic methods , which compare the clustering against the ground truth and measure.
- If the ground truth is unavailable, we can use the intrinsic methods , which evaluate the goodness of a clustering by considering how well the clusters are separated.
- Ground truth can be considered as supervision in the form of “cluster labels.” Hence, extrinsic methods are also known as supervised methods , whereas intrinsic methods are unsupervised methods .

Intrinsic Methods

- The intrinsic methods are based on the fundamental intuition in clustering analysis, that is, examining how compact clusters are and how well clusters are separated.
- Many intrinsic methods take the advantage of a similarity or distance measure between objects in the data set.

  - Elbow Method
  - Silhouette Index
  - Dunn Index
  - Davies-Bouldin Index

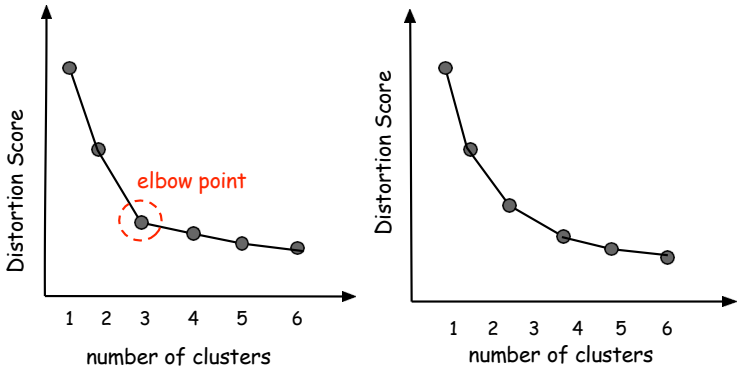


Elbow Method

- The elbow method is based on the observation that increasing the number of clusters can help to reduce the sum of within-cluster variance of each cluster.
- This is because having more clusters allows one to capture finer groups of data objects that are more similar to each other.
- However, the marginal effect of reducing the sum of within-cluster variances may drop if too many clusters are formed, because splitting a cohesive cluster into two gives only a small reduction.
- Consequently, a heuristic for selecting the right number of clusters is to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters.

Elbow Method

The elbow method is used to identify optimal number of clusters.
It is mostly used for partitioning methods, e.g. *k*-means. Elbow method presents the relation between a distortion score and the number of clusters.
The point that the elbow is constructed identifies the optimal number of clusters.
Distortion score is computed, the sum of square distances from each point to its assigned center.



Elbow method example, (Left) there is an elbow and can conclude three is the number of optimal clusters. (Right) There is no elbow and thus we can not use elbow method to determine the optimal number of clusters.

Elbow Method Example

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

In [2]:
```
iris = datasets.load_iris()
#we are usingh
df=pd.DataFrame(iris['data'])
df.head()
```

Out[2]:

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Elbow Method Example

In [3]:
```
mean = df.mean(axis=0)
std = df.std(axis=0)
# Calculate the z-score for each column in the dataframe
df_zscore = (df - mean) / std
print(df_zscore)
```

```
            0         1         2         3
0   -0.897674  1.015602 -1.335752 -1.311052
1   -1.139200 -0.131539 -1.335752 -1.311052
2   -1.380727  0.327318 -1.392399 -1.311052
3   -1.501490  0.097889 -1.279104 -1.311052
4   -1.018437  1.245030 -1.335752 -1.311052
..        ...       ...       ...       ...
145  1.034539 -0.131539  0.816859  1.443994
146  0.551486 -1.278680  0.703564  0.919223
147  0.793012 -0.131539  0.816859  1.050416
148  0.430722  0.786174  0.930154  1.443994
149  0.068433 -0.131539  0.760211  0.788031

[150 rows x 4 columns]
```

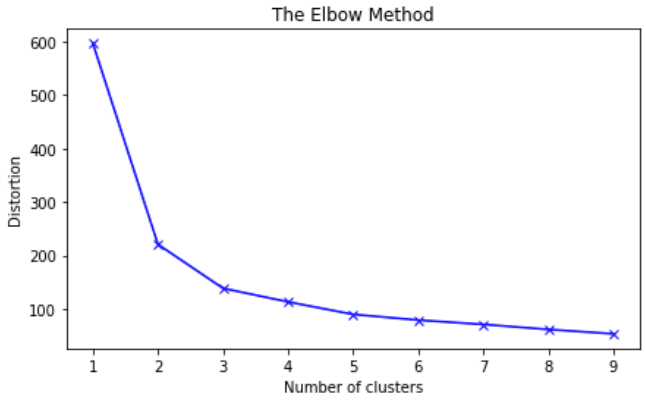
In [4]:
```
iris['target']
```

Out[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Elbow Method Example

In [5]:
```
distortions = []
# number of clusters 1 to 10
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k ,n_init=10)
    kmeanModel.fit(df_zscore)
    distortions.append(kmeanModel.inertia_)
```

In [6]:
```
plt.figure(figsize=(7,4))
plt.plot(K, distortions, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.title('The Elbow Method')
plt.show()
```



Elbow Method Example

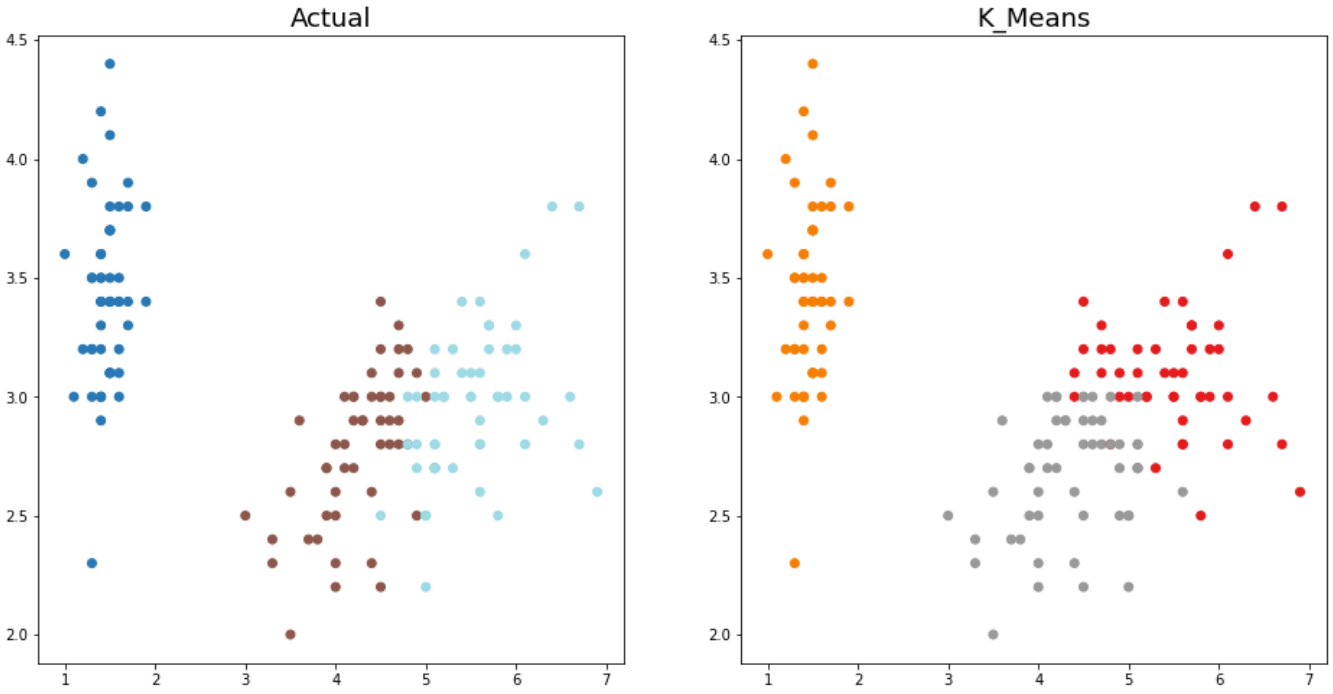
In [7]:
```
kmeanModel = KMeans(n_clusters=3 ,n_init=10)
kmeanModel.fit(df_zscore)
```

Out[7]:
```
KMeans
KMeans(n_clusters=3, n_init=10)
```

In [8]:
```
df['k_means']=kmeanModel.predict(df_zscore)
df['target']=iris['target']
fig, axes = plt.subplots(1, 2, figsize=(16,8))

axes[0].scatter(df[2], df[1], c=df['target'], cmap=plt.cm.tab20)
axes[1].scatter(df[2], df[1], c=df['k_means'], cmap=plt.cm.Set1)
axes[0].set_title('Actual', fontsize=18)
axes[1].set_title('K_Means', fontsize=18)
```

Out[8]:
```
Text(0.5, 1.0, 'K_Means')
```

Elbow Method Example



Silhouette Index

Silhouette coefficient is a score range from -1 to +1. It measures how similar is a member of a cluster is to its own cluster, i.e. cohesion, compared to other clusters, i.e. separation.

Silhouette coefficient for data object *i* is calculated based on parameters concepts, *a(i)* and *b(i).*
a(i) is the average distance between member *i* and other members of the cluster (the cluster which *i* belongs to it).
b(i) is the minimum average distance of *i* to all points in any other clusters (here we mean all clusters that *i* is not their member).

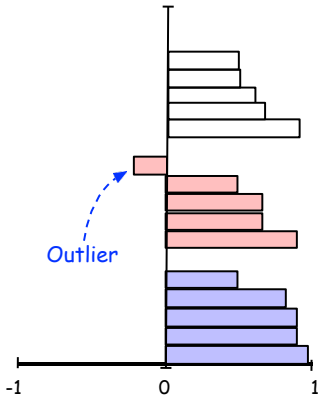
Silhouette coefficient is presented as: $$s(i) = \frac{b(i) - a(i)}{max(b(i), a(i))}$$

Silhouette Index

Some tools visualize Silhouette index, each cluster is presented with one color and you can see the red cluster has one outlier because its silhouette coefficient is negative.

After identifying Silhouette coefficient, assuming *k* is the number of clusters, we can easily calculate Silhouette index as, follows:

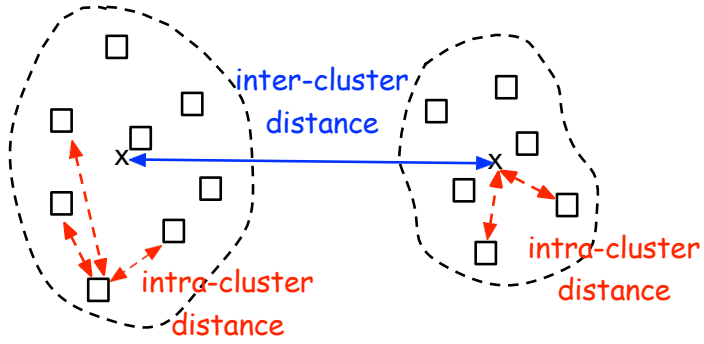
$$S_n = \frac{S_1 + S_2 + \ldots + S_k}{k}$$



Dunn Index

- Dunn Index reports the intra- class distance between two clusters divided by the maximum value of inter-class distance from one of those clusters.
- Assuming c_1, c_2, … are clusters, the DI can be written as follows:

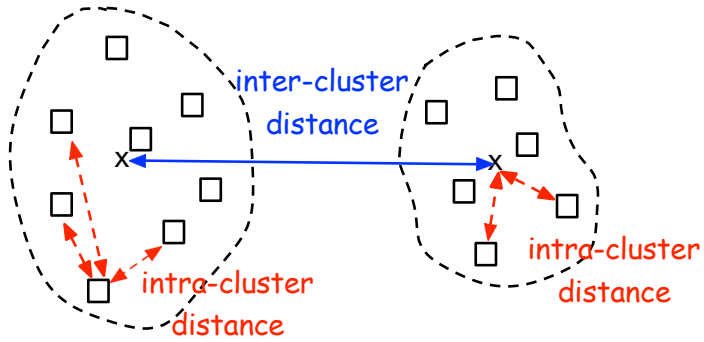
$$DI = \frac{\min \quad (\text{inter\_cluster\_disnace}(c_1, c_2))}{\max \quad (\text{intra\_cluster\_distnace}(c_1 \text{ or } c_2))}$$



- A higher Dunn index reveals that clusters' members are compact and clusters are well separated from each other.

Davies-Bouldin Index

$$DBI = mean \quad (max \quad \{ \frac{inter\_distance(c1) + inter\_distance(c2)}{intra\_distance(C1,C2)} \})$$



All intrinsic methods are all operated based on the fact that a good clustering method is having a high intra-cluster similarity (low distance) and low inter-cluster similarity (high distance).

Extrinsic Methods

- Extrinsic evaluation will be done when a ground truth dataset is available. We use unsupervised learning, when we do not have a labeled data. However, this style of evaluation requires a small labeled dataset (ground truth dataset).
  - The matching-based methods
  - The information theory-based methods
  - The pairwise comparison-based methods

The matching-based methods

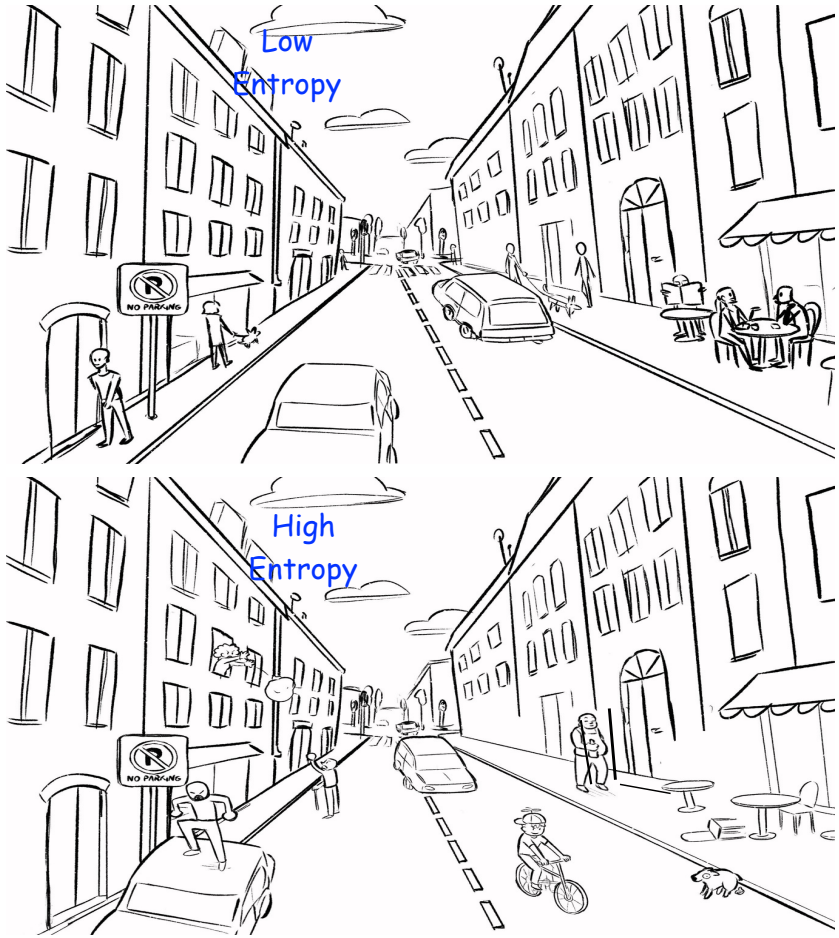
- Examines how well clustering results match the ground truth in partitioning the objects in the data set.
- For example, the purity methods assess how a cluster matches only those objects in one group in the ground truth.

The information theory-based methods

- Compare the distribution of the clustering results and that of the ground truth.
- Entropy or other measures in information theory are often employed to quantify the comparison.
- For example, we can measure the conditional entropy between the clustering results and the ground truth to measure whether there exists dependency between the information of the clustering results and the ground truth. The higher the dependency, the better the clustering results.

Entropy

- Entropy is a measure of the randomness or uncertainty in a system.
- In information theory, it quantifies the average amount of information contained in each message received.
- In statistical mechanics, it relates to the number of ways in which a system may be arranged, commonly associated with the second law of thermodynamics, which states that the entropy of an isolated system never decreases over time.



The pairwise comparison-based methods

- Treat each group in the ground truth as a class and then check the pairwise consistency of the objects in the clustering results.
- The clustering results are good if more pairs of objects of the same class are put into the same cluster, less pairs of objects of different classes are put into the same cluster, and less pairs of objects of the same class are put into different clusters.

Purity

- The higher the purity, the purer are the clusters, that is, the more objects in each cluster belong to the same group in the ground truth.

- Ci: clusters
- Gi: Ground truth groups

$$purity = \sum_{i=1}^{m} \frac{|C_i|}{n} \max_{j=1}^{l} \{ \frac{|C_i \cap G_j|}{|C_i|} \} = \frac{1}{n} \sum_{i=1}^{m} \max_{j=1}^{l} \{ |C_i \cap G_j| \}$$

Rand Index

RI is also the result of comparing our algorithm result with the ground truth dataset.

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

Rand Index

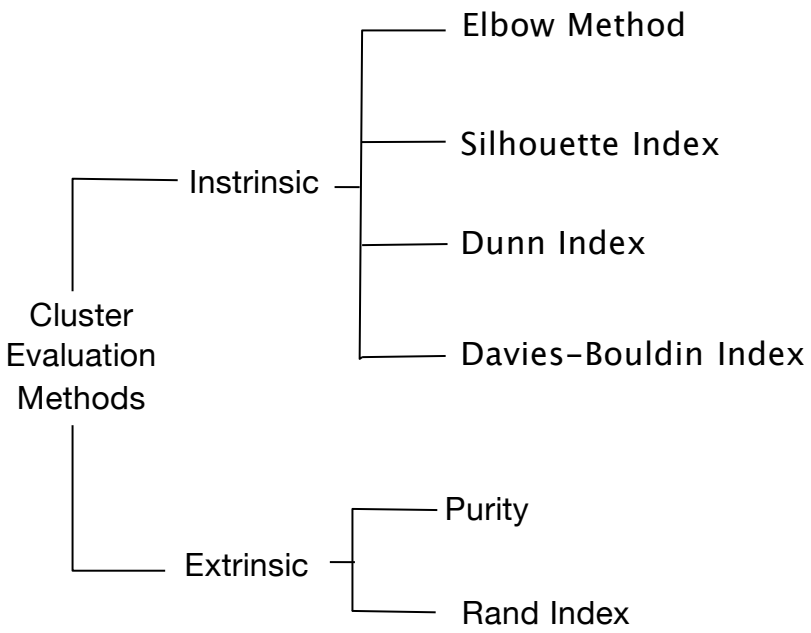
Consider C as clustering results of algorithms and G are clustering results of ground truth.

a. the number of data objects that are in the **same** clusters of C in and in the **same** clusters of G
b. the number of data objects that are in **different** clusters of C and in **different** clusters of G
c. the number of data objects that are in the **same** clusters of C and in **different** clusters of G
d. the data objects that are in the in **different** clusters of C and in the **same** clusters of G

a+ b = number of agreements, c+d = number of disagreements.

$$RI = \frac{a + b}{a + b + c + d}$$

Intrinsic and extrinsic cluster evaluation methods



Large Scale Clustering

BIRCH Clustering
(balanced iterative reducing and clustering using hierarchies)

- BIRCH is a scalable hierarchical clustering using clustering feature trees.
- There are two major difficulties in the agglomerative and divisive hierarchical clustering methods discussed so far.
  - All those methods cannot revisit any merge or split decisions made before. So, an improper decision based on limited information may lead to low quality final clustering results.
  - Scalability is a major bottleneck, since each merge or split needs to examine many possible options. To overcome those difficulties, we may conduct hierarchical clustering in multiple phases, so that clustering results can be improved over phases.
- BIRCH uses the notions of clustering feature to summarize a cluster, and clustering feature tree (CF-tree) to represent a cluster hierarchy.

BIRCH Clustering

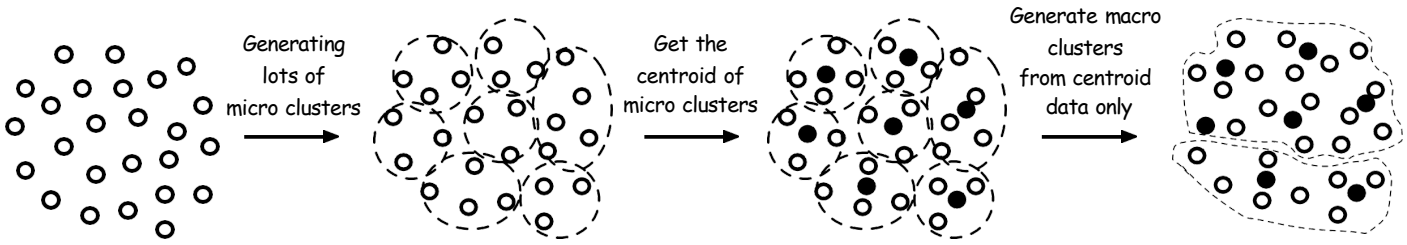
Phase1: The first phase produces a micro-clustering on the original data. First, it scans the data into the memory. This scan is used to reduce the complexity and increase the scalability. The result of the first scan (pass) is having lots of small clusters.

Phase2: In this phase the algorithm uses an arbitrary clustering algorithm (e.g. k-mean, dbscan,…) to performs the global clustering. This pass improves the clustering results of the first pass and creates macro-clustering results. This pass applies a clustering algorithm (any arbitrary clustering) on CF-Tree leaf nodes and not the original dataset. In simple words, it takes the centroid of each cluster in the previous pass and performs a clustering only on centroid data.

BIRCH Clustering

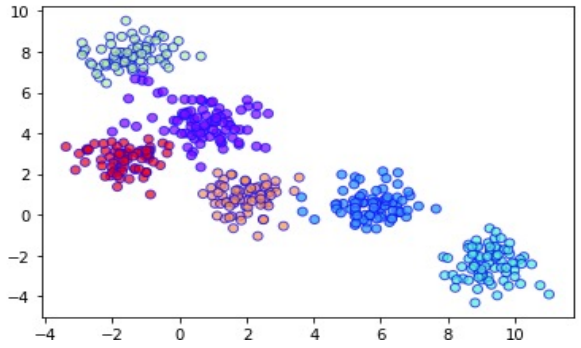
Phase1: The first phase produces a micro-clustering on the original data. First, it scans the data into the memory. This scan is used to reduce the complexity and increase the scalability. The result of the first scan (pass) is having lots of small clusters.

Phase2: In this phase the algorithm uses an arbitrary clustering algorithm (e.g. k-mean, dbscan,…) to performs the global clustering. This pass improves the clustering results of the first pass and creates macro-clustering results. This pass applies a clustering algorithm (any arbitrary clustering) on CF-Tree leaf nodes and not the original dataset. In simple words, it takes the centroid of each cluster in the previous pass and performs a clustering only on centroid data.



Birch Example

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import Birch

# Create the data
X, clusters = make_blobs(n_samples=450, centers=6, cluster_std=0.70, random_state=0)

# Create the BIRCH model
# branching factor is the maximum number of CF subclusters in each node
# radius of subclusters should be lass than threshold
brc = Birch(n_clusters=6, branching_factor=20,  threshold=0.4)

# Fit the model to the data
brc.fit(X)

# Predict the labels for the data
labels = brc.predict(X)

# Plot the data
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='rainbow', alpha=0.7, edgecolors='b')
plt.show()
```



BIRCH advantage/ disadvantage

Advantage:

A capability of BIRCH is that it can be used for data streams, and when a new data arrives, the algorithm can dynamically calculate its CF and assign it to its related cluster. BIRCH scales linearly.

Disadvantage:

Each CF-node in the CF-Tree can hold limited number of entries.

A CF-Tree node is not always a representation of a cluster.

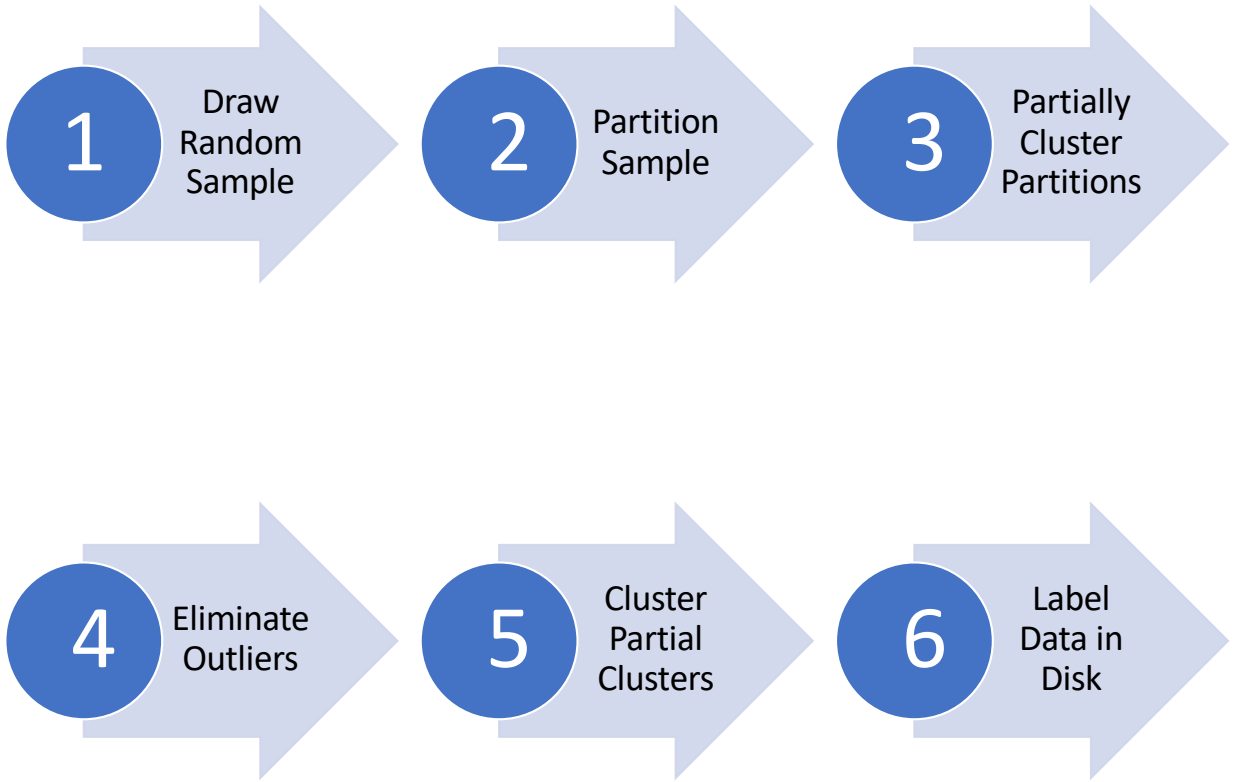
Since it uses the notion of radius or diameter to control the boundary of a cluster and if the dataset does not have a spherical shape, it might not perform well.

Large Scale Clustering

CURE (Clustering Using REpresentatives)

- CURE is another efficient data clustering algorithm for large databases. Cure is an extension of K-means.
- Compared with K-means clustering it is more robust to outliers and able to identify clusters having non-spherical shapes and size variances.
- K-means uses square error method. If there is large differences in sizes or geometries of different clusters, the square error method could split the large clusters to minimize the square error, which is not always correct.

CURE (Clustering Using REpresentatives)



CURE (Clustering Using REpresentatives)

Data Subset Selection

To initiate the CURE algorithm, an initial subset of data points needs to be chosen from the dataset being analyzed. These randomly selected points will act as potential representatives for producing robust clusters.

Hierarchical Clustering

Next, these representative points are clustered hierarchically using either agglomerative or divisive techniques. Agglomerative clustering gradually merges similar representatives until reaching one central representative per cluster while divisive clustering splits them based on dissimilarities.

Cluster Shrinkage

Once all clusters are obtained through hierarchical clustering, each cluster's size is reduced by reducing the outlier’s weights in relation to their distance from their respective representative points. This process helps eliminate irrelevant noise and focuses on more relevant patterns in each individual cluster.

Final Data Point Assignment

After shrinking the initial clusters down to their core components, all remaining nonrepresentative points are assigned to their nearest existing representative based on Euclidean distance or other suitable measures consistent with specific applications.

CURE (Clustering Using REpresentatives)

Random Sampling: The first step in the CURE algorithm entails randomly selecting a subset of data points from the given dataset. This random sampling ensures that representative samples are obtained across different regions of the data space rather than being biased toward particular areas or clusters.

Hierarchical Clustering: Next comes hierarchical clustering on the sampled points. Employing techniques such as Single Linkage or Complete Linkage hierarchical clustering methods helps create initial compact clusters based on their proximity to each other within this smaller dataset.

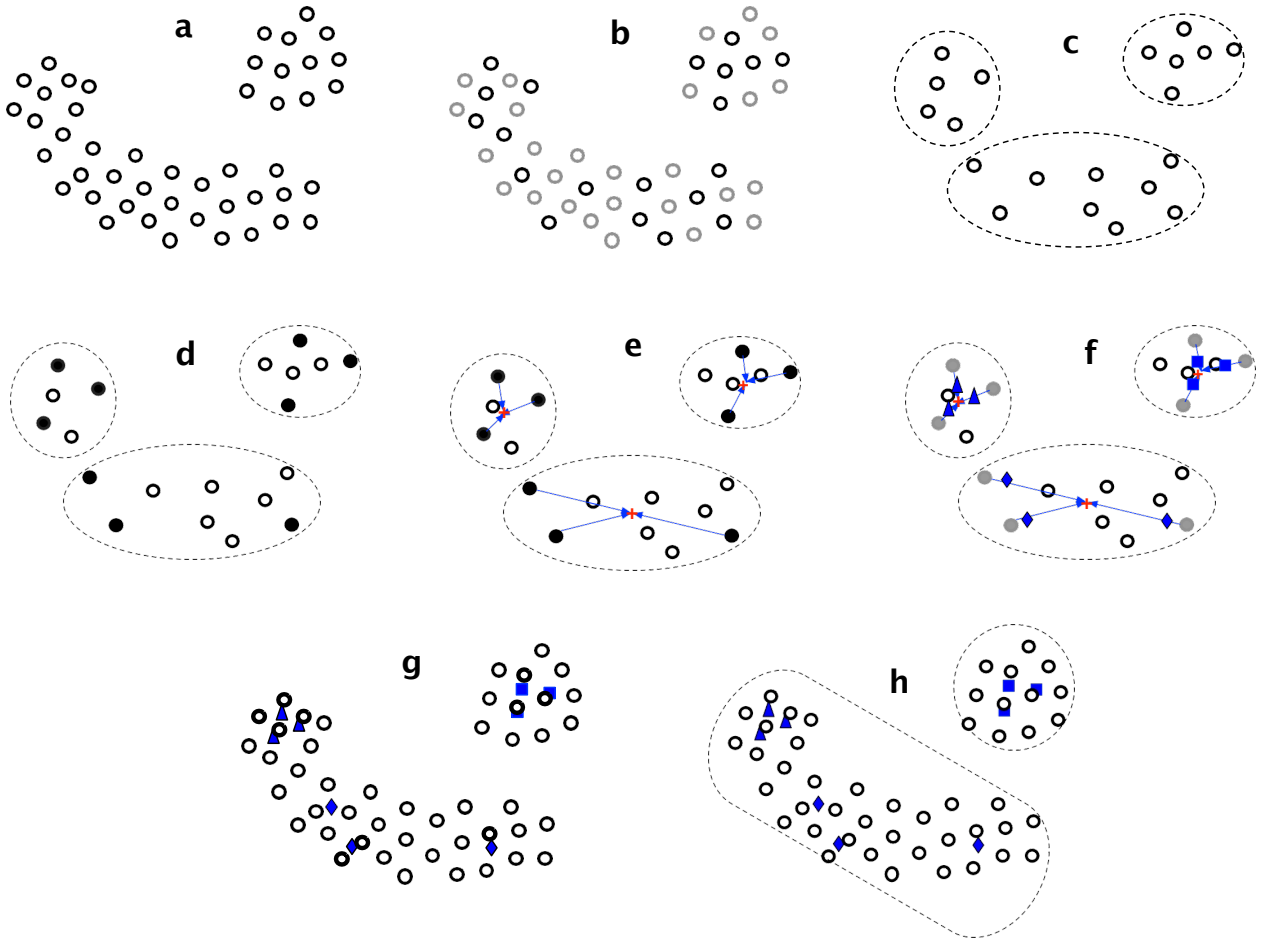
Distance Measures: CURE leverages distance measures to compute distances between clusters during merging operations while maintaining an efficient runtime. Euclidean distance is commonly used due to simplicity; however, other distance metrics like Manhattan can be employed depending on domain−specific requirements.

CURE (Clustering Using REpresentatives)

Merging Representative Points: With cluster centroids determined through hierarchical clustering, CURE focuses on merging representative points from various sub−clusters into a unified set by employing partial aggregations and pruning appropriately. This consolidation facilitates a significant reduction in computation time by making subsequent operations more concise.

Cluster Refinement and Splitting: After merging representatives, refinement takes place through exchanging outliers among aggregated sets for better alignment with true target structures within each merged group. Subsequently, splitting occurs, when necessary, by forming new individual agglomerative groups representing modified substructures unaccounted for during earlier hierarchies.

Final Membership Assignment: Lastly, assigning remaining objects outside formed aggregates follows suit − specifically those not captured effectively via either mergers or refinements. These yet−to−beclustered points are linked with the cluster identifiers of their nearest representative points, finalizing the overall clustering process.



CURE is useful to deal with large dataset also, it reduces the impact of outliers.

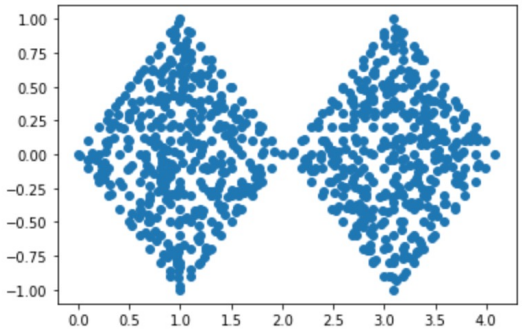
Cure Clustering in Python

- https://pyclustering.github.io/docs/0.10.1/html/dc/d6d/classpyclustering_1_1cluster_1_1cure_1_1cure.html

Cure Clustering in Python

In [1]:
```
from pyclustering.cluster import cluster_visualizer
from pyclustering.cluster.cure import cure
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pyclustering.samples.definitions import SIMPLE_SAMPLES,FCPS_SAMPLES,FAMOUS_SAMPLES
from pyclustering.utils import read_sample, timedcall
```

In [2]:
```
sample = read_sample(FCPS_SAMPLES.SAMPLE_TWO_DIAMONDS)
X=sample

plt.scatter(list(zip(*X))[0], list(zip(*X))[1])
plt.show()
```



Cure Clustering in Python

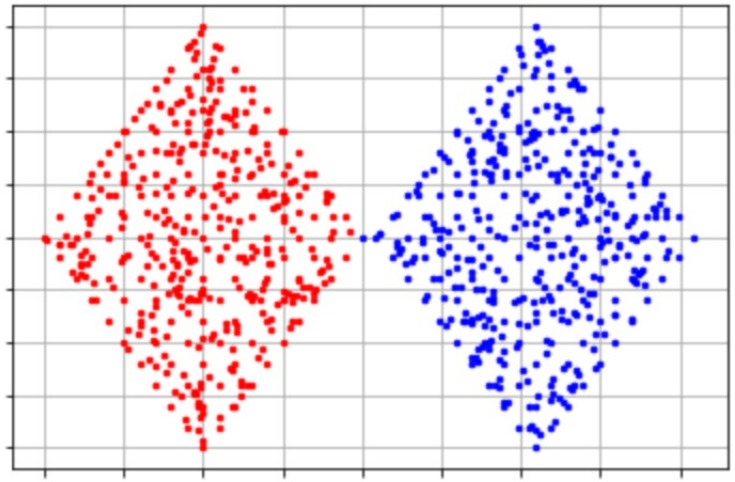
In [3]:
```
# create cure object
cure_clusters=cure(X,number_cluster=2, number_represent_points=4)
# cluster the data
cure_clusters.process()
# get cluster labels
clusters=cure_clusters.get_clusters()
#print(clusters)
```

In [4]:
```
# get representative points
representative = cure_clusters.get_representors()
print(representative)
```

```
[[[0.5009373934837094, 0.002216667919799515], [1.0009373934837094, -0.49778333208020054], [1.0009373934837094, 0.50
22166679197995], [1.4619933934837093, 0.015105667919799514]], [[2.540691882793018, -0.0010253366583541119], [3.0856
91882793018, -0.5010253366583541], [3.085691882793018, 0.49897466334164586], [3.585691882793018, -0.001025336658354
1119]]]
```

Cure Clustering in Python

```
In [5]: visualizer= cluster_visualizer()
        visualizer.append_clusters(clusters, X)
        visualizer.show()
```

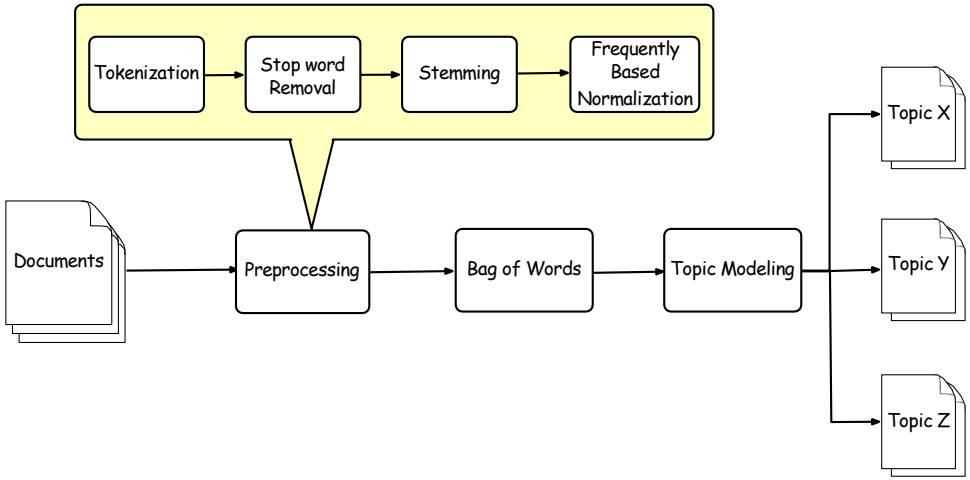


Topic Modeling

Topic modeling are methods that cluster textual documents based on the appearance and similarities between their words (terms). In other words, topic modeling is trying to identify the abstract document topics based on the frequency of their words (terms).

To cluster text documents, which are unstructured data, we usually use topic modeling.

Topic modeling is operated based on the concept of Bag of Words. Which means the algorithm should convert a raw text document into a Bag of Words, then fed into the topic modeling algorithm.



Topic Modeling

- There are two main ways to do topic modeling:
  1. LSA: Latent Semantic Analysis
  2. LDA: Latent Dirichlet Allocation

Latent Semantic Analysis (LSA)

LSA or LSI (Latent Semantic Indexing) assumes that there is a hidden (latent) relation between terms. To use LSA, first we need to convert our documents into document-term matrix. But, before conversion into document-term matrix, we need to preform the necessary text processing steps, which we have previously explained, including tokenization, stemming, etc.

D1: I am lifting a heavy chicken today; it is really heavy.
D2: Today, the heavy chicken eats lots of food.
D3: It is hard to lift heavy chickens and easy to lift light chickens.

	D1	D2	D3
I	1	0	0
lift	1	0	1
heavy	2	1	1
chicken	1	1	1
today	1	2	1
eat	0	1	0
...	...	...	...

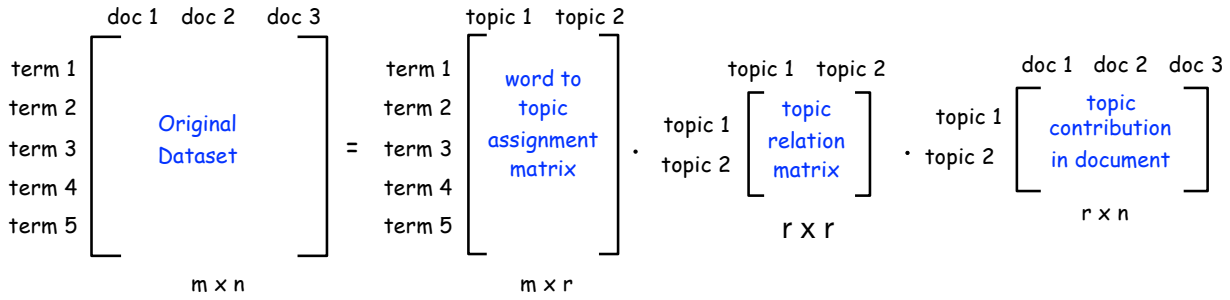
Latent Semantic Analysis (LSA)

- Build a Document-Term Matrix (X), where each entry X_{ij} is a raw count of j-th word appearing in the i-th document. However, In practice, we use TF-IDF Vectorizer to assign weights to each in the document.
- Apply Singular Value Decomposition (SVD) to find few latent topics to capture the relationship between words and documents.
- Then we pick top-k topics, (i.e) $X = U_k * S_k * V_k$.

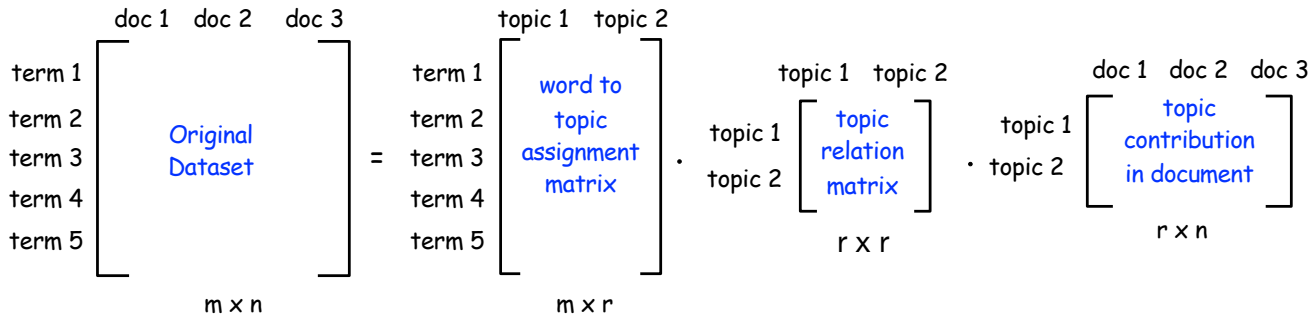
Latent Semantic Analysis (LSA)

The next step is to apply SVD on the term-document matrix.

After term-document matrix is ready, it will be **decomposed by the SVD** and we will get three matrix as follows. One matrix is word to topic assignment, the second one is topic relation and the third one is topic contribution in the document.



Latent Semantic Analysis (LSA)



- The topic contribution in document matrix is the result that we are looking for. It describe the contribution of identified topic in each document. Topic is something that the system will identify. Nevertheless, the topic will be not explicitly provided by the algorithm, we should look on documents and manually create a name for a topic.

- For example, a topic which includes document about “gear”, “break”, “wheel”, “seat”, “auto-pilot”, etc. could be named as “car” topic, or a topic which includes documents about “cheesecake”, “moon cake”, “custards”, “cupcake” could be named “dessert”.

Latent Semantic Analysis (LSA)

- pip install --upgrade genism
- Gensim is a short form for the generate similarity, it is an open-source fully specialized python library written to represent documents vectors efficiently.
- Genism is designed to be used in Topic modeling tasks to extract semantic topics from documents, Genism is your tool in case you're want to process large chunks of textual data, it uses algorithms like Word2Vec, FastText, Latent Semantic Indexing (LSI, LSA, LsiModel), Latent Dirichlet Allocation (LDA, LdaModel) internally.

https://www.datacamp.com/tutorial/discovering-hidden-topics-python

LSA Example

In [1]:
```
import pandas as pd
from gensim import corpora
from gensim.models import LsiModel
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from gensim.models.coherencemodel import CoherenceModel
import matplotlib.pyplot as plt
```

In [2]:
```
# Load the dataset
datafile = 'abcnews-date-text.csv'

raw_data = pd.read_csv(datafile, parse_dates=[0])
reindexed_data = raw_data['headline_text']
reindexed_data.index = raw_data['publish_date']

reindexed_data.head()
```

Out[2]:
```
publish_date
2003-02-19    aba decides against community broadcasting lic...
2003-02-19       act fire witnesses must be aware of defamation
2003-02-19       a g calls for infrastructure protection summit
2003-02-19             air nz staff in aust strike for pay rise
2003-02-19        air nz strike to affect australian travellers
Name: headline_text, dtype: object
```

LSA Example

In [3]:
```
# preprocessing steps
# tokenize, removing stopwords, and stemming

tokenizer = RegexpTokenizer(r'\w+')
# create English stop words list
en_stop = set(stopwords.words('english'))
# Create p_stemmer of class PorterStemmer
p_stemmer = PorterStemmer()
# list for tokenized documents in loop
texts = []
# loop through document list
for i in reindexed_data:
   # clean and tokenize document string
    raw = i.lower()
    tokens = tokenizer.tokenize(raw)
    # remove stop words from tokens
    stopped_tokens = [i for i in tokens if not i in en_stop]
    # stem tokens
    stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]
    # add tokens to list
    texts.append(stemmed_tokens)
prep_data= texts
```

LSA Example

In [4]:
```
# Creating the term dictionary of our courpus,
# where every unique term is assigned an index.
dictionary = corpora.Dictionary(prep_data)
# Converting list of documents (corpus) into Document Term Matrix using dictionary prepared above.
doc_term_matrix = [dictionary.doc2bow(doc) for doc in prep_data]
```

In [5]:
```
# generate LSA model
lsamodel = LsiModel(doc_term_matrix, num_topics=7, id2word = dictionary)  # train model
print(lsamodel.print_topics(num_topics=7, num_words=5))
```

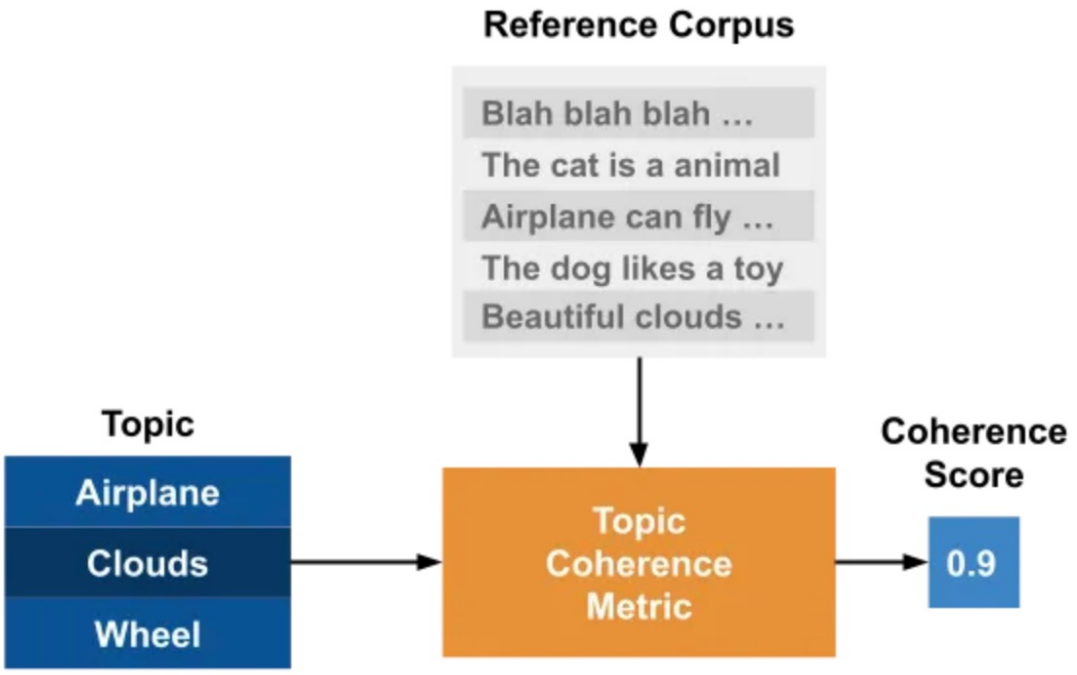
```
[(0, '0.689*"polic" + 0.467*"man" + 0.224*"charg" + 0.153*"new" + 0.124*"court"'), (1, '0.768*"new" + -0.401*"man"
+ 0.177*"plan" + -0.168*"charg" + 0.131*"council"'), (2, '-0.671*"polic" + 0.519*"man" + 0.301*"new" + 0.246*"char
g" + 0.163*"court"'), (3, '0.496*"govt" + -0.485*"new" + 0.364*"plan" + 0.317*"say" + 0.202*"council"'), (4, '-0.78
1*"govt" + 0.428*"plan" + 0.227*"say" + 0.225*"council" + 0.146*"us"'), (5, '-0.641*"plan" + 0.483*"us" + 0.391*"sa
y" + 0.185*"kill" + 0.152*"iraq"'), (6, '-0.656*"us" + 0.433*"council" + -0.389*"plan" + 0.253*"say" + -0.153*"kil
l"')]
```

Topic Coherence Metric

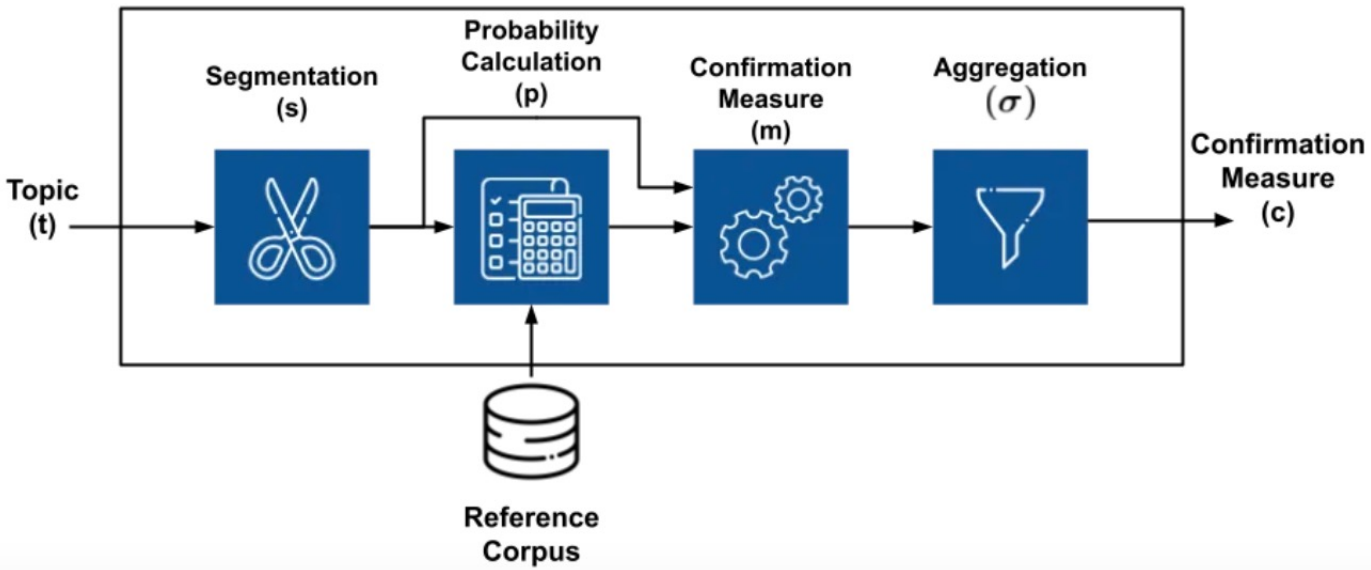
- Calculate topic coherence for topic models.
  - Topic 1: Cat, dog, home, toy. (Probably a good topic)
  - Topic 2: Super, nurse, brick. (Probably a bad topic)
- What a Topic Coherence Metric assesses is how well a topic is 'supported' by a text set (called reference corpus). It uses statistics and probabilities drawn from the reference corpus, especially focused on the word's context, to give a coherence score to a topic.

https://radimrehurek.com/gensim/models/coherencemodel.html
https://towardsdatascience.com/understanding-topic-coherence-measures-4aa41339634c

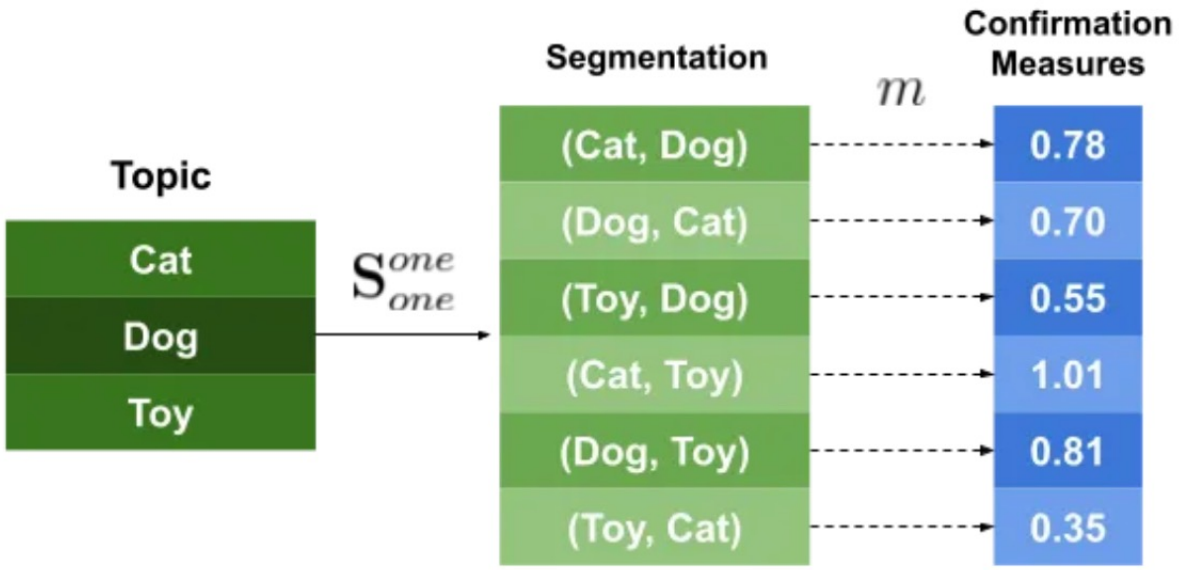
Topic Coherence Metric



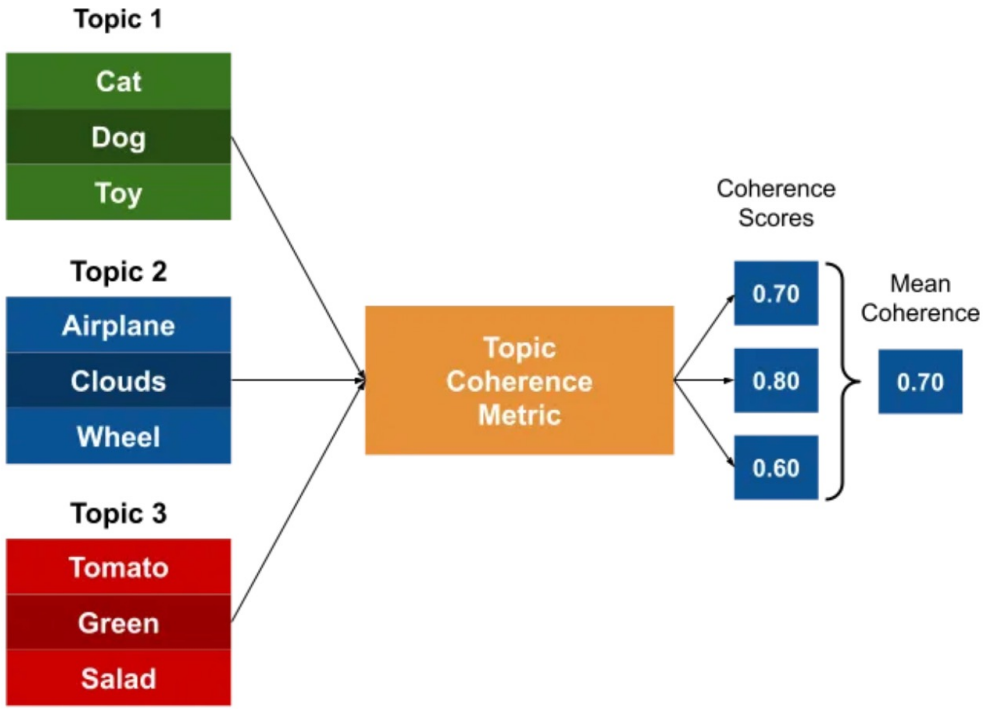
Topic Coherence Metric



Topic Coherence Metric



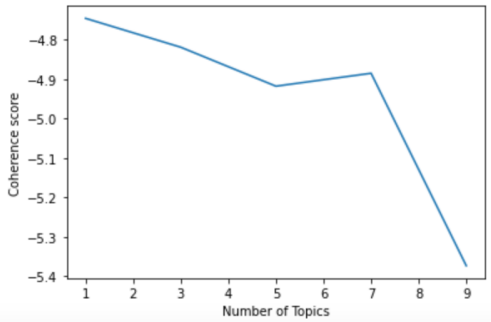
Topic Coherence Metric



LSA Example- Coherence Metric

In [6]:
```
model_list=[]
coherence_values=[]
start,stop,step=1,10,2
for num_topix in range(start, stop, step):
    # generate LSA model
    model = LsiModel(doc_term_matrix, num_topics=num_topix, id2word = dictionary)  # train model
    model_list.append(model)
    coherencemodel = CoherenceModel(model=model, texts=prep_data, dictionary=dictionary, coherence='u_mass')
    coherence_values.append(coherencemodel.get_coherence())
```

In [7]:
```
x = range(start, stop, step)
plt.plot(x, coherence_values)
plt.xlabel("Number of Topics")
plt.ylabel("Coherence score")

plt.show()
```



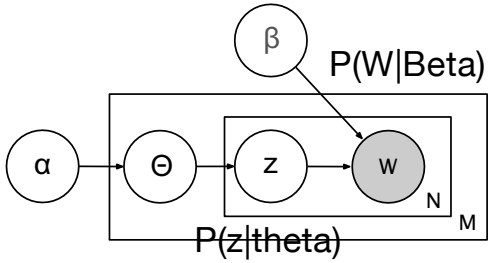
Latent Dirichlet Allocation (LDA)

- LDA is a significant improvement from LSA in the context that LSA considers no probabilistic determination inside the document structures. In LDA, this is the major change.
- The LDA model tries to find groups of words (the topics) that appear together frequently. LDA also requires that each document can be understood as a "mixture" of a subset of the topics.

1. Every document is a mixture of topics. We imagine that each document may contain words from several topics in particular proportions. For example, in a two-topic model we could say "Document 1 is 90% topic A and 10% topic B, while Document 2 is 30% topic A and 70% topic B."
2. Every topic is a mixture of words. For example, we could imagine a two-topic model of American news, with one topic for "politics" and one for "entertainment." The most common words in the politics topic might be "President", "Congress", and "government", while the entertainment topic may be made up of words such as "movies", "television", and "actor". Importantly, words can be shared between topics; a word like "budget" might appear in both equally.

Latent Dirichlet Allocation (LDA)

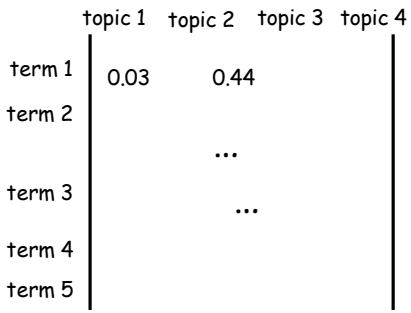
- This figure presents the probabilistic model for LDA in a plate notation (another name for probabilistic graphical model).
- When we draw an arrow from one node to another node it presents the probability dependence. A box around nodes means variables inside the box will be repeated *N* times or *M* times and anything in grey means that it is observable variable (*W* is observable variable).
- α and *β* are Dirichlet distribution parameters and most LDA packages do this by default. But, if our initial result of LDA does not make sense, we could start to tune these two parameters as well. *Θ* is the topic distribution for document *m*, *z* is the topic for the *n*-th word in document *m*, and *w* is specific term. Except *W* (words) everything else is unknown.

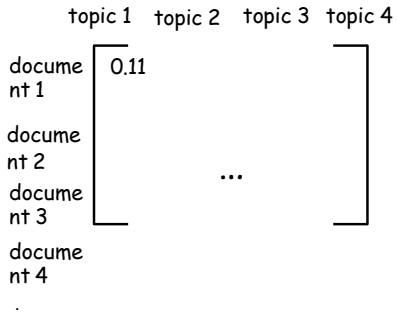


Latent Dirichlet Allocation (LDA)

- The result of LDA are two matrices as it has been shown in the following.

- The matrix on the left presents the probability of terms appearances inside each document. The matrix on the right presents the topic appearance probabilities in each document. Each data point of these matrices is a probability value.





LDA Example

In [8]:
```
lda_model = gensim.models.LdaMulticore(doc_term_matrix,
                                       num_topics=3,
                                       id2word = dictionary,
                                       passes = 4,
                                       workers=2)
```

In [9]:
```
#Here it should give you a ten topics as example shown below image
for idx, topic in lda_model.print_topics(-1):
    print("Topic: {} \nWords: {}".format(idx, topic))
    print("\n")
```

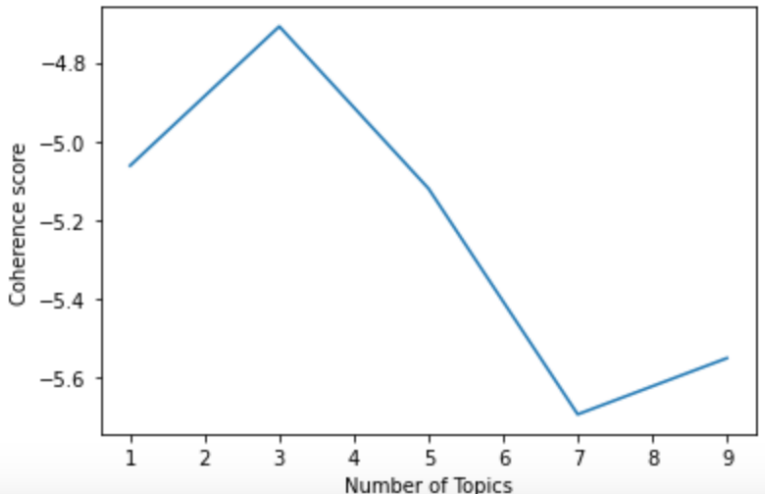
```
Topic: 0
Words: 0.009*"say" + 0.008*"new" + 0.007*"govern" + 0.007*"call" + 0.006*"wa" + 0.006*"plan" + 0.005*"nsw" + 0.00
5*"chang" + 0.005*"market" + 0.004*"sa"


Topic: 1
Words: 0.015*"australia" + 0.009*"win" + 0.008*"court" + 0.008*"trump" + 0.008*"day" + 0.007*"world" + 0.006*"count
ri" + 0.006*"open" + 0.006*"face" + 0.005*"first"


Topic: 2
Words: 0.015*"man" + 0.014*"polic" + 0.009*"charg" + 0.009*"year" + 0.008*"us" + 0.008*"fire" + 0.007*"attack" + 0.
007*"death" + 0.006*"murder" + 0.006*"kill"
```

LDA Example- Coherence Metric

In [10]:
```
model_list=[]
coherence_values=[]
start,stop,step=1,10,2
for num_topix in range(start, stop, step):
    # generate LDA model
    model = gensim.models.LdaMulticore(doc_term_matrix, num_topics=num_topix,
                                       id2word = dictionary, passes = 4,
                                       workers=2) # train model
    model_list.append(model)
    coherencemodel = CoherenceModel(model=model, texts=prep_data, dictionary=dictionary, coherence='u_mass')
    coherence_values.append(coherencemodel.get_coherence())
```

LDA Example- Coherence Metric

In [11]:
```
x = range(start, stop, step)
plt.plot(x, coherence_values)
plt.xlabel("Number of Topics")
plt.ylabel("Coherence score")
plt.show()
```



Types of Clustering Algorithms

- K-representative Clustering, Partitioning
- Density Based Clustering
- Hierarchical Clustering
- Probabilistic Clustering

Probability vs Likelihood

- Probability means what is the chance of observing *X* in the given sample dataset. This means that in a given dataset, what is the chance of observing *X*?

- Likelihood means given the observed subset *X* of a dataset, what is the best distribution parameters, that fit the given *X*.

- Randy Gallistel provides a good comparison as follows: ”Probability attaches to possible results; likelihood attaches to hypotheses“.

Probabilistic model-based clustering

- In most of the cluster analysis methods we have discussed so far, each data object can be assigned to only one of the clusters.
- This kind of strict cluster assignments are required in some applications, such as assigning customers to marketing strategies.
- In some other applications, this rigid requirement may not be desirable. In this section, we demonstrate the need for fuzzy or flexible cluster assignment in some applications and introduce a general method to compute probabilistic clusters and assignments.
- In probabilistic clustering the assignment of points to clusters is “soft”, in the sense that the membership of a data point x in a cluster Ck is given as a probability, denoted by pk(x).

Probabilistic model-based clustering Use Cases

Clustering product reviews:

Imagine that an e-commerce company has an online store, where customers not purchase online and create reviews of products.

- Not every product receives reviews.
- Some products may have many reviews.
- A review may involve multiple products.

In this context, a group of products, services, or issues that are highly related could be clustered together.

Assigning a review to one cluster exclusively would not work well for this task.

Suppose there is a cluster for “cameras” and another for “computers.” What if a review talks about the compatibility between a camera and a computer? The review is related to both clusters; however, it does not exclusively belong to either cluster.

Maximum Likelihood Estimation (MLE) Approach

- Maximum Likelihood Estimation is an approach to estimating the distribution of data, and an algorithm that implements it is Expectation Maximization.

- For example, we have a small part of a dataset, and the original dataset has a distribution (e.g. Gaussian distribution). We don't know what are the parameters (mean and standard deviation) of that Gaussian distribution, because there could be infinite numbers of Gaussian distributions. The MLE is a procedure that tries to identify the mean, and standard deviation, by using the sample dataset and constructing a distribution that is the closest match to the original dataset distribution.

- The *goal* of MLE is to find the best distribution parameters that fit the original dataset (population).

Maximum Likelihood Estimation (MLE) Approach

- The maximum likelihood estimation method aims to solve the following optimization problem, which says that we should choose the best weight vector w that maximizes the likelihood of the training set. The intuition is that we want to find the optimal model parameter (i.e., the weight vector w) so that there is the highest "chance" (i.e., the likelihood or the probability) of observing the entire training set.
- It means we calculate the inverse of minimization, which is equal to maximization. In simple words, minimizing the error is equivalent to maximizing the log-likelihood.

Expectation Maximization

- Latent variable problem: Although MLE does not have access to the original dataset, it assumes the dataset is complete or fully observed.

- Nevertheless, part of the dataset could be missing in the observation subset that is used by MLE. Those missing parts could construct parameters that are known as latent variables. Here, latent variables refer to parameters that do not exist in the observed dataset.

- Expectation Maximization (EM) algorithm implements the MLE, when there is a latent variable in the dataset.

- In summary, EM algorithm is recommended to use for MLE implementation when we are dealing with missing data in our observed dataset.

EM Algorithm

- The objective of EM, like the MLE objective, is to find unknown parameters that find the best distribution fit to the original dataset (approximate maximum likelihood).
- E-step makes an initial guess of parameters for the expected distribution.
- M-step starts after the E-step, and when newly observed data will be fed into the model. In this step, the EM algorithm tweaks the estimated parameters (from E-Step) to cover newly observed data as well.
- From M-step, the process will be repeated until the created distribution does not change in E-step or M-step and it reaches a stable state (converged), or a maximum threshold of iteration reaches.

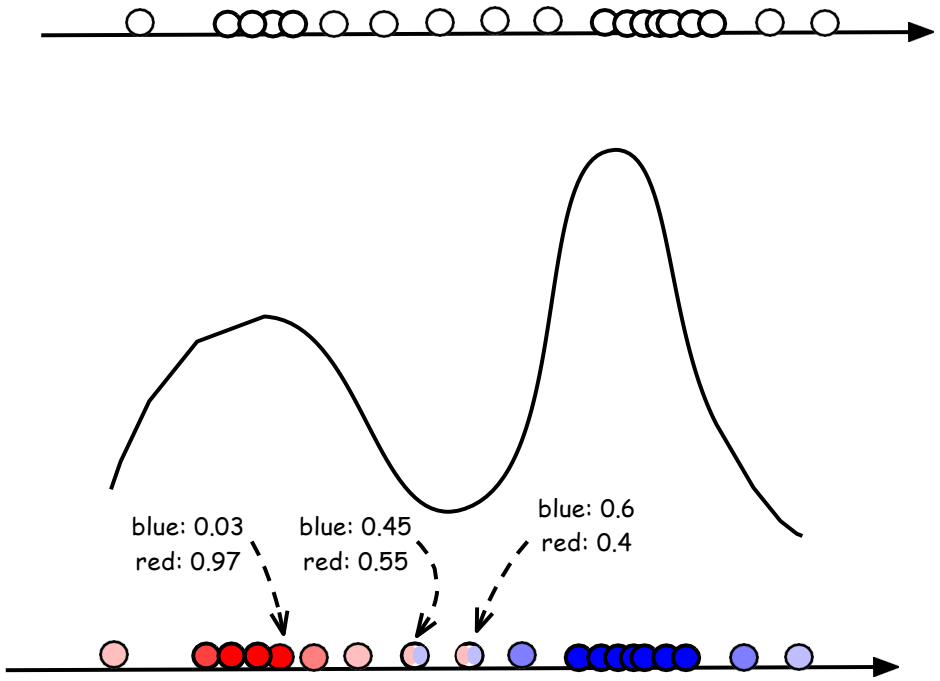
Probabilistic Clustering

Described clustering methods assume each data point will be assigned to one cluster and a single data point cannot participate in more than one cluster.

Probabilistic clustering allows a data point to be a member of more than one cluster. This is called soft clustering or probabilistic clustering.

- This type of clustering assumes the dataset is a mixture of two or more probability distributions. Each distribution presents a cluster, and distribution is described by a density function with a weight (mixing coefficient).

- In other words, mixture model clustering methods assume that a dataset can be divided into probability distributions (clusters) and clusters are following a known distribution, e.g. Gaussian. Therefore, we can say a dataset is composed of a combination of different distributions. A mixture model is a weighted sum of distributions.



On dimensional data distribution on top and its distribution on the bottom.

How to convert a dataset into a combination of different distributions?

Maximum Likelihood Estimation

Problem: We don't have access to the entire dataset, but we have access to a portion of the dataset, and by using this available portion, we would like to estimate a distribution that the entire dataset fits in it.

We use MLE to determine the best distribution that presents the entire dataset, by using the sample dataset.

The MLE method of GMM tries to identify the mean, μ, and standard deviation, δ, to construct a distribution which is the closest match to the dataset.

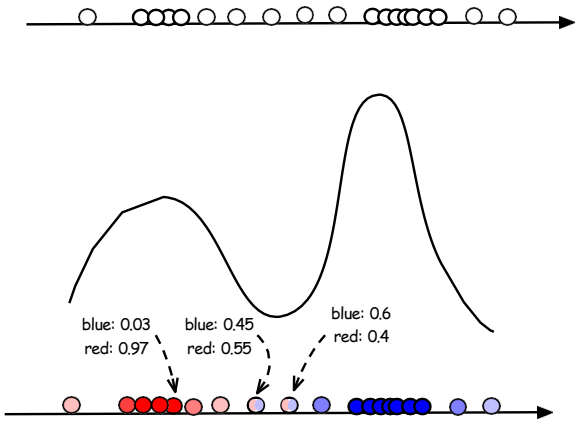
Gaussian Mixture Model

- A Gaussian mixture model (GMM) is a probabilistic model that represents data as a mixture of Gaussian distributions. GMMs are used for clustering, density estimation, and dimensionality reduction. They are a powerful algorithm for discovering underlying patterns in a dataset.
- GMM is an example of a probabilistic model-based clustering. That is, we assume that the probability density function of each cluster follows a Gaussian distribution. Suppose there are k clusters. The two parameters for the probability density function of each cluster are center, μ , and standard deviation.

Gaussian Mixture Model

- If we assume all result clusters should have a gaussian distribution, we can use Gaussian Mixture Model (GMM) to cluster the dataset.

- As it has been described each cluster is modeled by a distribution. The Gaussian distribution is defined by mean (correspond to center of the distribution) and standard deviation (how spread are the data around the center).

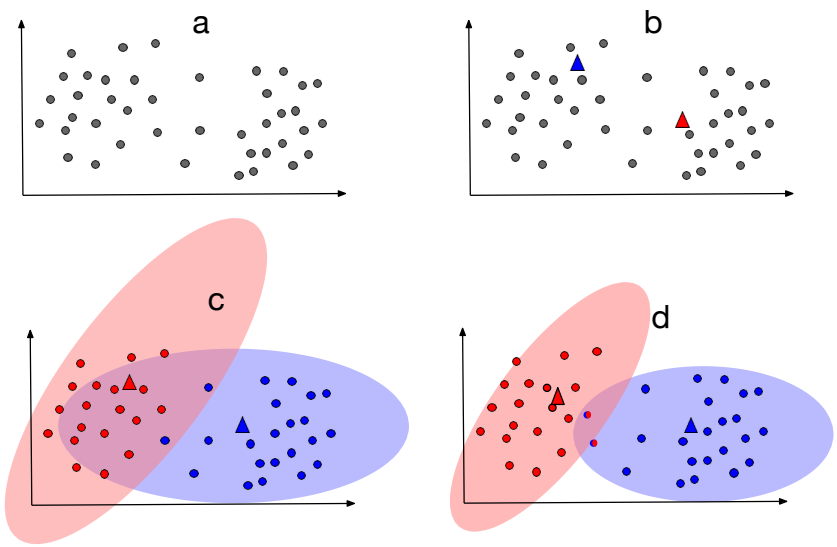


Gaussian Mixture Model

- To identify distributions (clusters) of a dataset which constitute the clustering, the clustering is defined as a parameter estimation problem. As we have described, we are seeking to identify the parameter and GMM uses EM algorithm to accomplish this task.

- In GMM each cluster is characterized by mean vector, covariance matrix and associated probability.

- The goal of GMM is to represent each sub-dataset as its own distribution (or mixture component). The entire dataset could then be represented as a mixture of *n* Gaussian distributions, and *n* is the number of clusters.



EM is the main challenge and its computational complexity is similar to k-mean, O(m.n.i)

Expectation step: This step compares the distances (in probabilities) from all other points to **the two selected points**. Then based on the calculated **probabilities**, each data point will be assigned to one or more clusters.

Maximization step: This step **moves the mean points**, i.e. triangles, into the center (mean) of each cluster. Afterward, again the algorithm reassigns data points based on the new locations of means. This process iteratively continues until there will be no changes in mean points, they stay at their location, and then both clusters were finalized.

In other words, the EM algorithm stops the iteration, either by observing that mean and SD are not moving or by reaching a predefined threshold, which a user can specify this threshold.

GMM Example

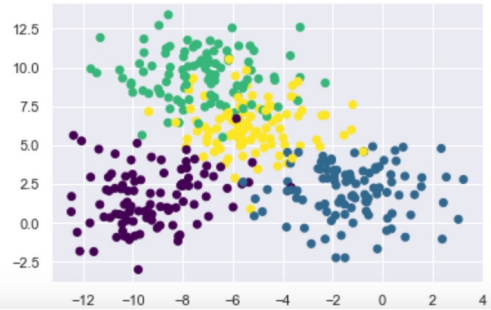
- https://www.kaggle.com/code/vipulgandhi/gaussian-mixture-models-clustering-explained

GMM Example 1

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn import mixture
```

In [2]:
```
#Creating the test data for clustering
# y is the actual label for blobs
# the gmm model is agnostic to y

X, y = make_blobs(n_samples=400, centers=4, n_features=4, cluster_std=1.8, random_state=820)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')
```

Out[2]: <matplotlib.collections.PathCollection at 0x7f9671642dc0>



GMM Example 1

In [3]:
```
gmm = mixture.GaussianMixture(n_components=4)
gmm.fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=10, cmap='viridis')
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x7f96717092e0>
```

GMM Example 1

Gaussian mixture model contains a probabilistic model under the hood, it is also possible to find probabilistic cluster assignments. In Scikit-Learn this is done using the predict_proba method.

```
In [4]: probs = (gmm.predict_proba(X)).round(3)
        probs[100:120]

Out[4]: array([[1.   , 0.   , 0.   , 0.   ],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 0.   , 0.998, 0.002],
               [0.   , 0.   , 0.   , 1.   ],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 0.   , 0.995, 0.005],
               [0.   , 0.   , 1.   , 0.   ],
               [0.962, 0.   , 0.   , 0.038],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 0.   , 0.996, 0.004],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 1.   , 0.   , 0.   ],
               [0.   , 1.   , 0.   , 0.   ],
               [0.998, 0.   , 0.   , 0.002],
               [0.   , 1.   , 0.   , 0.   ],
               [0.995, 0.   , 0.002, 0.003],
               [0.887, 0.   , 0.001, 0.111]])
```

GMM Example 1

In [5]:
```
# visualize the effect of the probabilities
size = 10 * probs.max(1) ** 2  # square emphasizes differences
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=size)
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x7f96507d84c0>
```

GMM Example 2

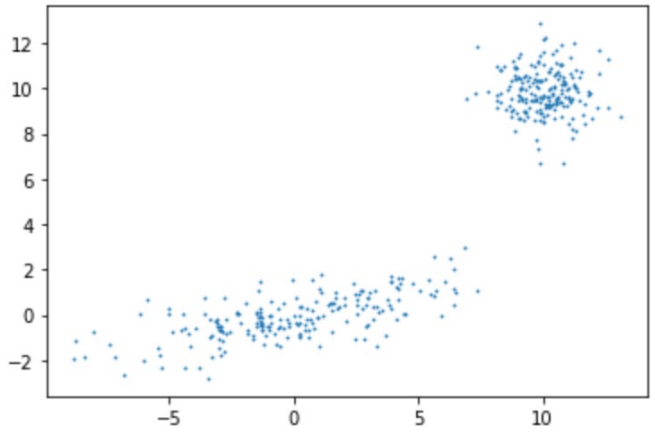
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from sklearn import mixture
```

In [2]:
```
n_samples=200
# generate random sample, two components
np.random.seed(70)
# generate spherical data centered on (20, 20)
shifted_gaussian  =  np.random.randn(n_samples,  2) + np.array([10, 10])
# generate zero centered stretched Gaussian data
C = np.array([[0., -0.7], [3.5, .7]])
stretched_gaussian = np.dot(np.random.randn(n_samples,2), C)

# concatenate the two datasets into the final training set
X_train = np.vstack([shifted_gaussian, stretched_gaussian])
```

GMM Example 2

In [3]:

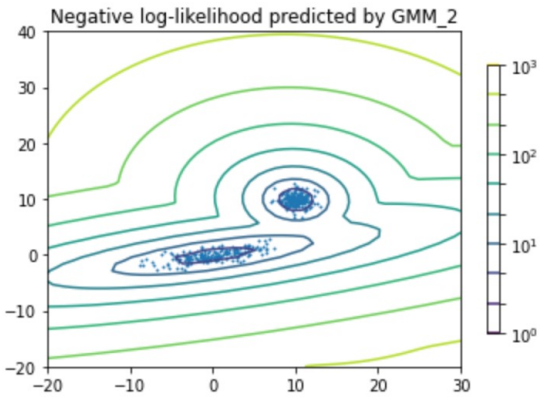
```
# fit a Gaussian Mixture Model with two components
gmm_2 = mixture.GaussianMixture(n_components=2, covariance_type='full')
gmm_2.fit(X_train)

plt.scatter(X_train[:,0], X_train[:,1], 0.8)
plt.show()
```



GMM Example 2

In [4]:
```
#display predicted scores by the model as a contour plot
x=np.linspace(-20, 30)
y=np.linspace(-20, 40)
X,Y=np.meshgrid(x,y)
XX = np.array([X.ravel(), Y.ravel()]).T
Z = -gmm_2.score_samples(XX)
Z = Z.reshape(X.shape)
CS = plt.contour(X,Y,Z, norm=LogNorm(vmin=1.0, vmax=1000.0),levels=np.logspace(0,3,10))
CB = plt.colorbar(CS, shrink=0.8, extend='both')
plt.scatter(X_train[:,0],  X_train[:,1],  .9)
plt.title('Negative log-likelihood predicted by GMM_2')
plt.axis('tight')
```

Out[4]: (-20.0, 30.0, -20.0, 40.0)



Fuzzy C-Mean

- It is very similar to k-mean and starts by defining some random points in the space. This algorithm operates as follows:

1. Similar to k-mean it starts by randomly selecting *c* data points, as centroid of clusters, and *c* is a hyper parameter which defines the number of clusters.

2. It calculates the fuzzy membership of each node in a cluster and assign the probabilities of membership to each node.

3. Next, it computes the centroid of each clusters and move the centroid points toward the center of each cluster.

4. Step (2) and (3) repeats iteratively until centroid points do not change or the maximum number of iterations (given by the user) reached.

Fuzzy C-Mean Example

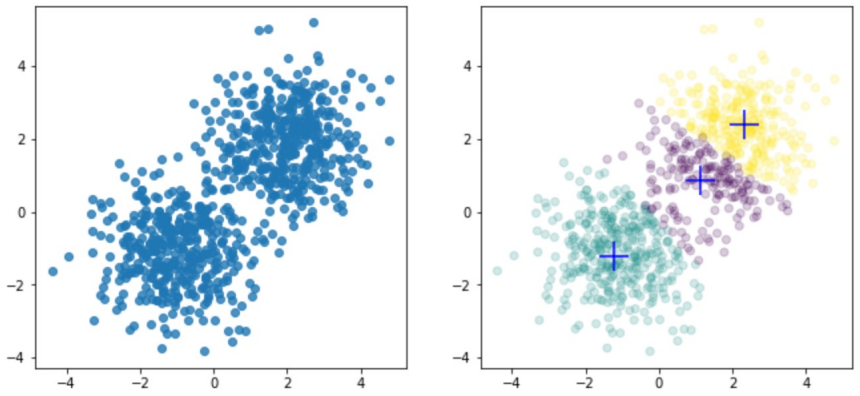
- pip install fuzzy-c-means

In [1]:
```
%matplotlib  inline
import  numpy  as  np
from fcmeans import FCM
from matplotlib import pyplot as plt
```

In [2]:
```
n_samples = 400
X = np.concatenate((np.random.normal((-1,-1), size=(n_samples,2)),
                    np.random.normal((2,2), size=(n_samples,2))   ))
fcm = FCM(n_clusters=3)
fcm.fit(X)

# outputs
fcm_centers = fcm.centers
fcm_labels = fcm.predict(X)
```

Fuzzy C-Mean Example

In [3]:
```
# outputs
fcm_centers = fcm.centers
fcm_labels = fcm.predict(X)
# plot  result
f, axes = plt.subplots(1, 2, figsize=(11,5))
axes[0].scatter(X[:,0], X[:,1], alpha=.8)
axes[1].scatter(X[:,0], X[:,1], c=fcm_labels, alpha=.2)
axes[1].scatter(fcm_centers[:,0],fcm_centers[:,1],
                marker="+", s=500,  c='b')

plt.savefig('fuzzy-output.jpg')
plt.show()
```



Fuzzy C-Mean Example

In [4]:
```
# look at soft (fuzzy) predictions
soft_prediction_labels = fcm.soft_predict(X)
soft_prediction_labels
```

Out[4]:
```
array([[0.01565674, 0.97882972, 0.00551354],
       [0.09677115, 0.85863035, 0.0445985 ],
       [0.14714758, 0.78989339, 0.06295903],
       ...,
       [0.18521577, 0.03695805, 0.77782618],
       [0.03300345, 0.00566055, 0.961336  ],
       [0.18697768, 0.05196247, 0.76105985]])
```

Category	Algorithm Name	Input Parameters	Descriptions
Partition based	k-means	1 number of clusters 2 threshold for iteration (optional)	the most used clustering algorithm due to its simplicity
	k-medoids	1 number of clusters 2 threshold for iteration (optional)	very similar to k-mean but more tolerant to outliers
Density based	DBScan	1 epsilon 2 minimum points	can clusters data points with convex shape
	OPTICS	1 minimum points 2 epsilon (search distance)	OPTICS reduces sensitivity of the epsilon parameter (in comparison to DBScan). It can also handles clusters with different densities.
Hierarchical	SLINK (agglomerative)	No parameter required	Both of these methods are computationally in-efficient, but they are baseline algorithms to hierarchical clusterings. For efficient hierarchical clustering check BIRCH and CURE.
	DIANA (divisive)	No parameter required	
Hierarchical for Large Dataset	BIRCH	1 CF-Tree branching factor 2 CF-Tree threshold 3 CF-Tree max. number of entries in leaf node	very scalable and be able to undo what has been done in its previous step. Sometimes CF-Tree can not fit into the memory and we get out of memory error
	CURE	1- number of clusters	Useful to handle large datasets with non-convex shapes.
Probabilistic	GMM	1- number of clusters	enables soft clustering, operates based on the two steps of expectation and maximization
	Fuzzy C-Mean	1- number of clusters	enables soft clustering, operates very similar k-mean

A summary of described clustering algorithms.

+ topic modeling (LSA and LDA)

References

- Galli, Soledad. Python Feature Engineering Cookbook: Over 70 recipes for creating, engineering, and transforming features to build machine learning models. Packt Publishing. Kindle Edition.
- Dey, Sandipan. Image Processing Masterclass with Python : 50+ Solutions and Techniques Solving Complex Digital Image Processing Challenges Using Numpy, Scipy, Pytorch and Keras (English Edition). BPB Publications. Kindle Edition.
- Han, Jiawei,Pei, Jian,Tong, Hanghang. Data Mining (The Morgan Kaufmann Series in Data Management Systems). Elsevier Science. Kindle Edition.