SWE404/DMT413
BIG DATA ANALYTICS

Lecture 10: Unsupervised Learning Algorithms

Lecturer: Dr. Yang Lu

Email: luyang@xmu.edu.my

Office: A1-432

Office hour: 2pm-4pm Mon & Thur

Outlines

- Clustering Algorithms
  - k-means
  - Spectral Clustering
- Principle Component Analysis

CLUSTERING ALGORITHMS

Basic Concepts

- Cluster: A collection of data objects.
  - Similar (or related) to one another within the same cluster.
  - Dissimilar (or unrelated) to the objects in other clusters.
- Clustering (or cluster analysis, data segmentation, ...)
  - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters.

Applications of Clustering

- Biology: taxonomy of living things.
  - kingdom, phylum, class, order, family, genus and species.
- Information retrieval: document clustering.
  - Automatic document organization, topic extraction and fast information retrieval or filtering.
- Land use: Identification of areas of similar land use in an earth observation database.
  - Satellite image analysis.
- Marketing: product grouping, customer segmentation.
  - Analytics can characterize their customer groups based on the purchasing patterns.

Requirements of Clustering

- **Scalability** − We need highly scalable clustering algorithms to deal with large databases.
- **Ability to deal with different kinds of attributes** − Algorithms should be capable to be applied on any kind of data such as interval-based (numerical) data, categorical, and binary data.
- **Discovery of clusters with attribute shape** − The clustering algorithm should be capable of detecting clusters of arbitrary shape. They should not be bounded to only distance measures that tend to find spherical cluster of small sizes.
- **High dimensionality** − The clustering algorithm should not only be able to handle low-dimensional data but also the high dimensional space.
- **Ability to deal with noisy data** − Databases contain noisy, missing or erroneous data. The real data is very dirty. Some algorithms are sensitive to such data and may lead to poor quality clusters.
- **Interpretability** − The clustering results should be interpretable, comprehensible, and usable. As a big data analyst, you should use clustering analysis to provide some insights to increase business value.

k-Means

- Given a set of data points $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n$, where each observation is a d-dimensional real vector, *k-means clustering* aims to partition the n data points into $k(\leq n)$ clusters $C = \{C_1, C_2, \dots, C_k\}$ so as to minimize the within-cluster sum of squares (WCSS).
- Formally, the objective is to minimize:

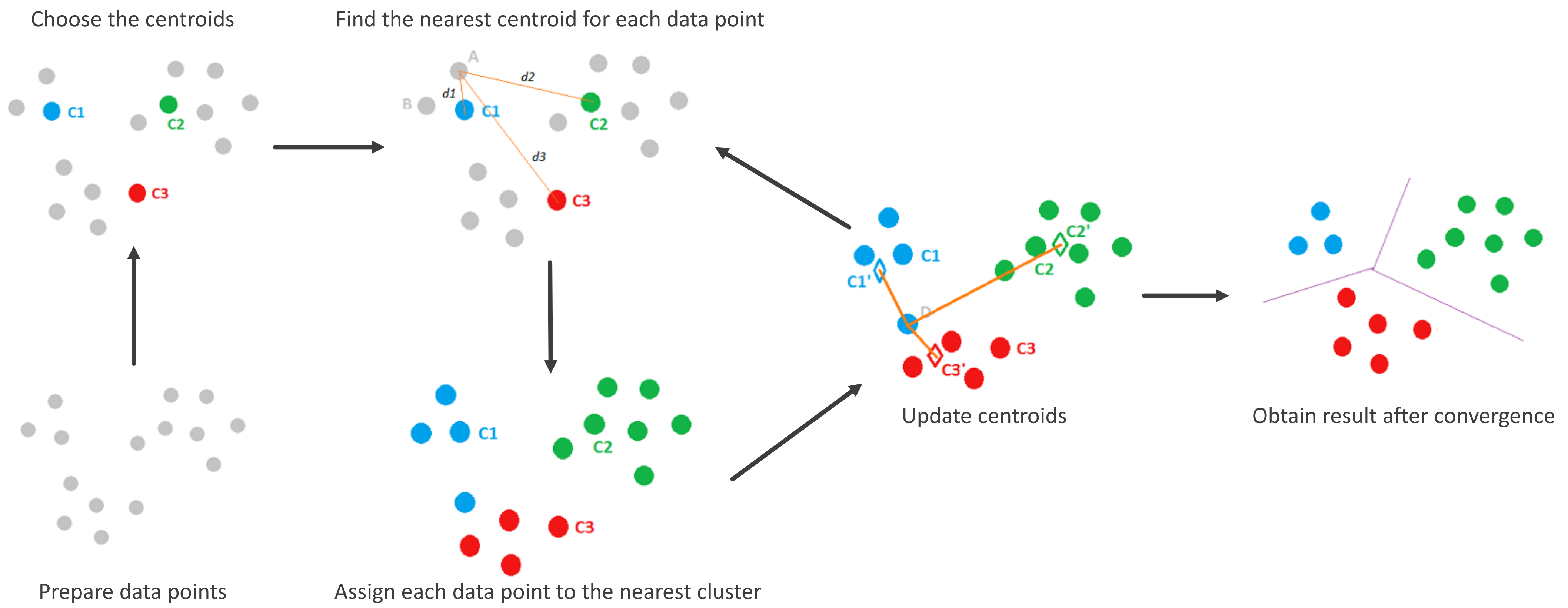
$$J = \sum_{j=1}^{k} \sum_{i=1}^{n_j} (\boldsymbol{x}_i - \boldsymbol{c}_j)^2$$

  - $\boldsymbol{c}_j$ is called the centroid of the jth cluster, which is calculated by the mean of points in C_i.
  - n_j is the number of data points assigned to the jth cluster.

k-Means Pseudocode

1. Determine the number of clusters k and obtain the data points $\boldsymbol{x}_i, i = 1, \dots, n$.
2. Choose the centroids $\boldsymbol{c}_1, \boldsymbol{c}_2, \dots, \boldsymbol{c}_k$ randomly.
3. Repeat steps 4 and 5 until convergence or until the end of a fixed number of iterations.
4. For each data point $\boldsymbol{x}_i$:
   - Find the nearest centroid $\boldsymbol{c}_j$ among $\boldsymbol{c}_1, \boldsymbol{c}_2, \dots, \boldsymbol{c}_k$.
   - Assign the point to that jth cluster.
5. For each cluster $j = 1, \dots, k$:
   - The centroid $\boldsymbol{c}_j$ is updated by the mean of all points assigned to jth cluster.

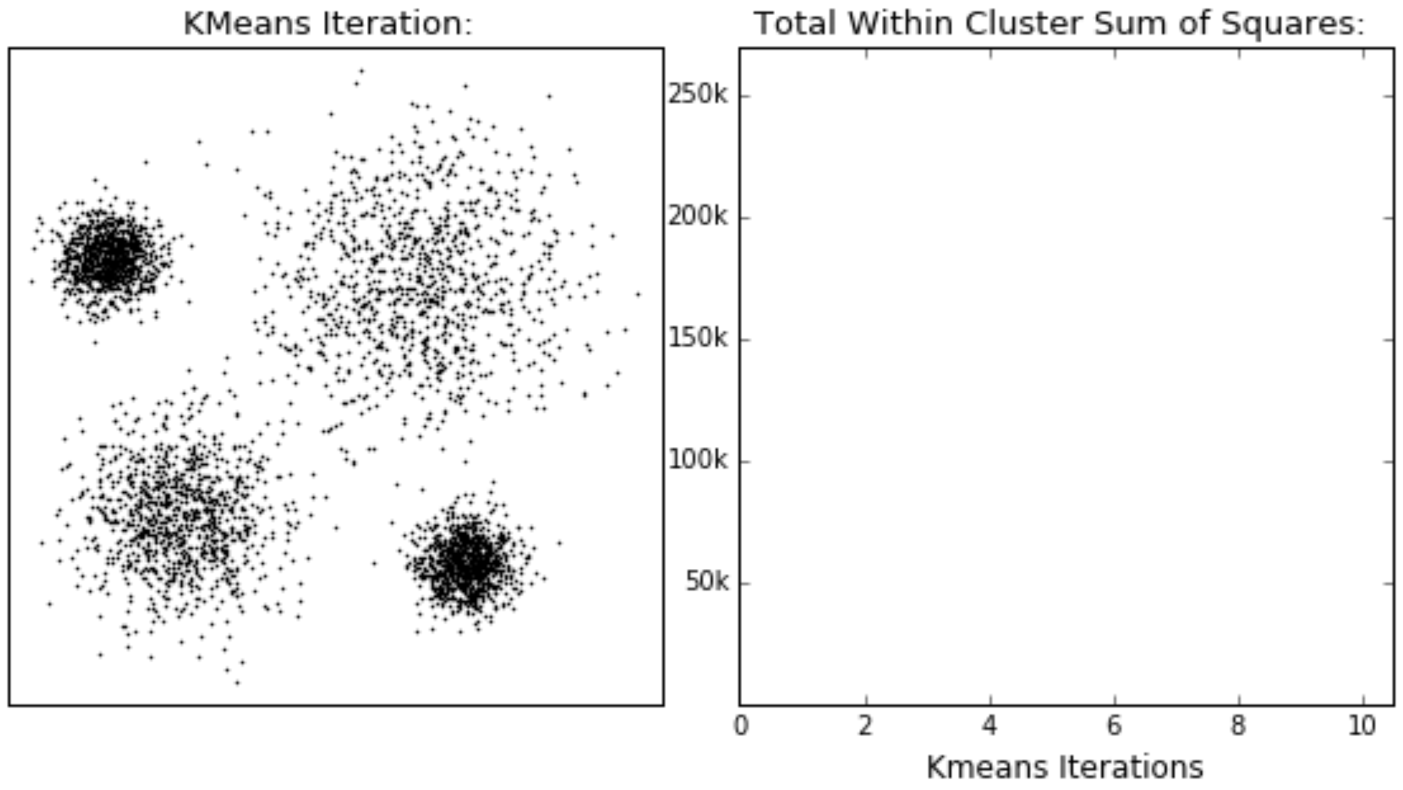
Step-by-step Example of k-Means


Choose the centroids
C1
C2
C3
Find the nearest centroid for each data point
A
B
d1
d2
d3
C1
C2
C3
Update centroids
C1
C1'
C2
C2'
C3
C3'
D
Obtain result after convergence
Prepare data points
Assign each data point to the nearest cluster
C1
C2
C3

Convergence of k-Means

- We can measure the difference of J between each consecutive iterations. If the difference is less than a pre-defined threshld, we stop and determine that the convergence is achieved.
- k-means usually quickly converges to a local minimum.

k-means Example



Imgae source: https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/

How to Choose k

- Because we don’t have the label information, the ideal number of clusters is unknown when we are doing clustering.
- It is necessary to select a proper k that best fits the data.
  - This is one of the most difficult problems for all clustering algorithm.
- A simple solution is called the elbow method. Try a different number of clusters and plot the total within-cluster sum of square.
  - In this example, the elbow point is at $k = 4$.
  - Even though the within-cluster distance decreases after 4, the improvement is not obvious.

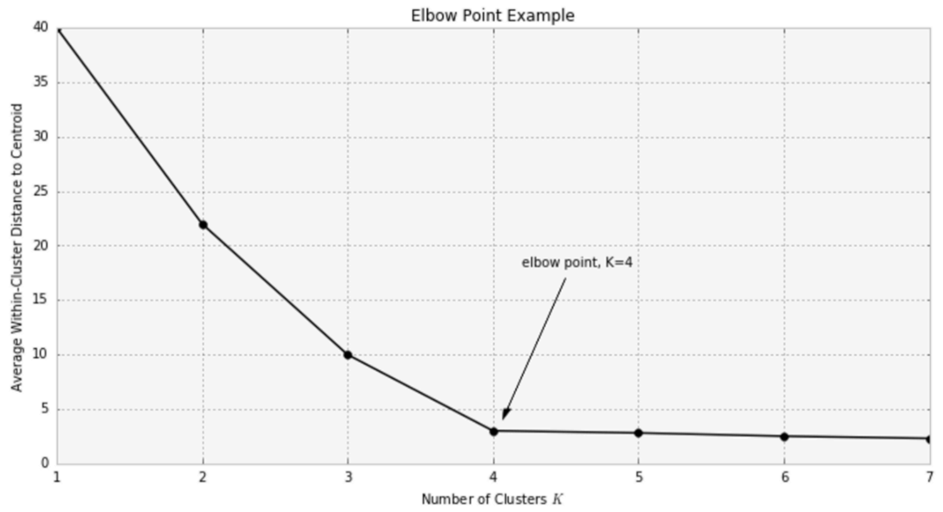


Image source: https://towardsdatascience.com/k-means-clustering-introduction-to-machine-learning-algorithms-c96bf0d5d57a

Unlucky Centroids

- Choosing poorly the initial centroids will take longer to converge or get stuck on local optima which may result in bad clustering.
  - In the figure, the blue and red stars are unlucky centroids.
- There are two solutions:
  - Distribute the initial centroids over the space.
  - Try different sets of random centroids, and choose the best set.

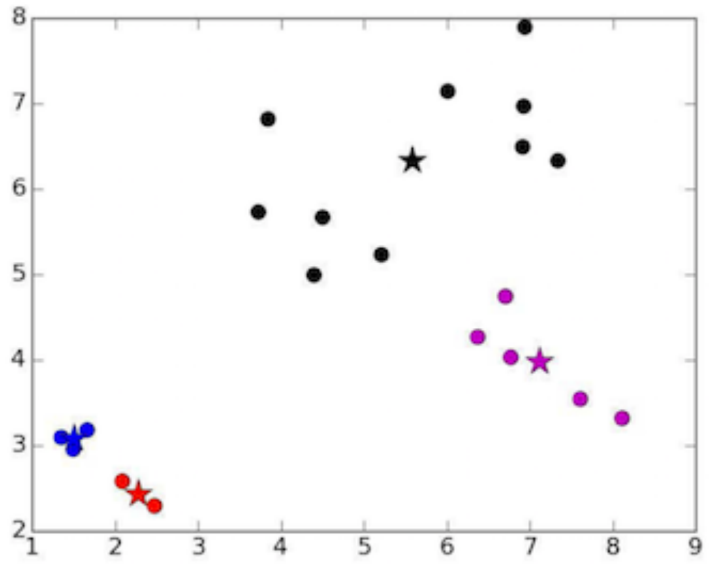


Image source: https://towardsdatascience.com/clustering-using-k-means-algorithm-81da00f156f6

k-means++

- k-means++ is proposed to solve the centroid initialization problem.
- The algorithm is as follows:
  1. Choose one centroid uniformly at random among the data points.
  2. For each data point x, compute $D(x)$, the distance between x and the nearest centroid that has already been chosen.
  3. Choose one new data point at random as a new centroid, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$.
  4. Repeat Steps 2 and 3 until k centroids have been chosen.
  5. Now that the initial centroids have been chosen, proceed using standard k-means clustering.

k-means++ Example

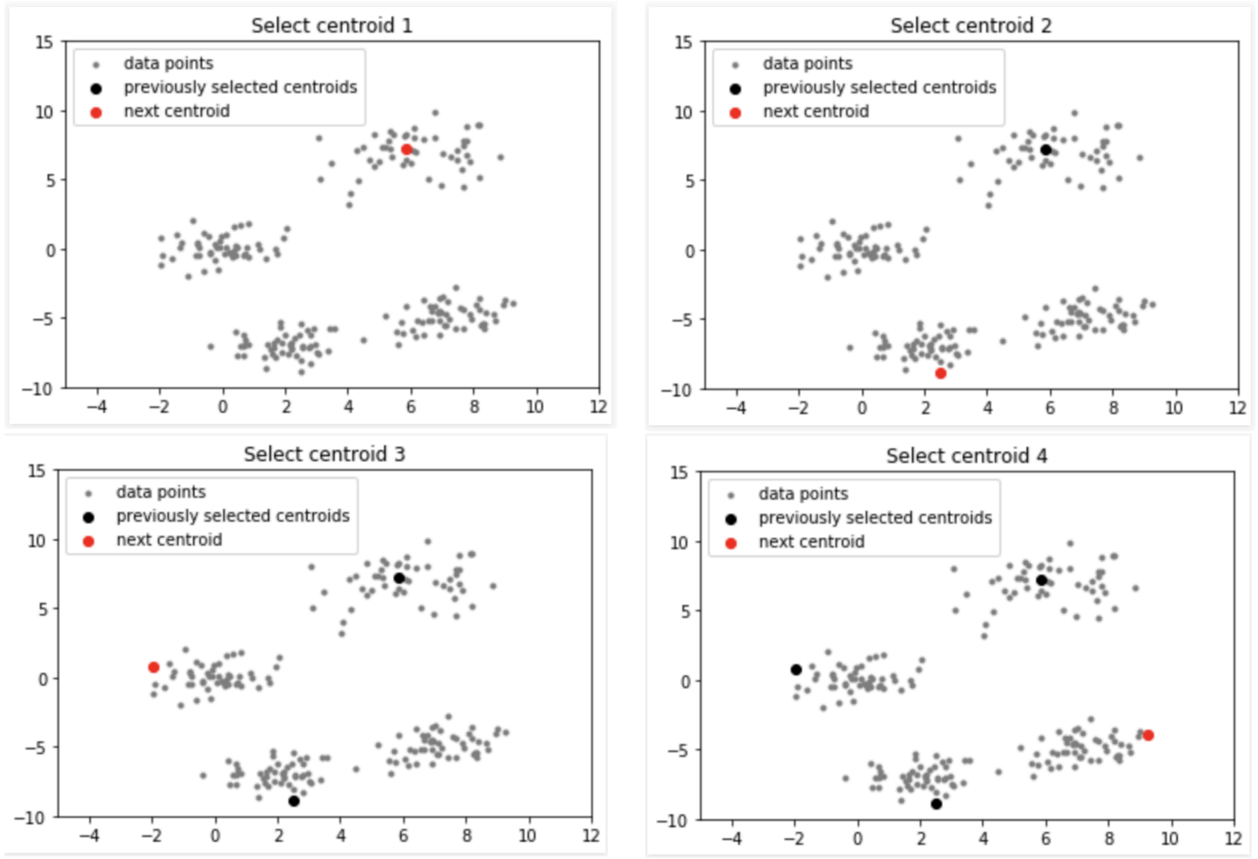
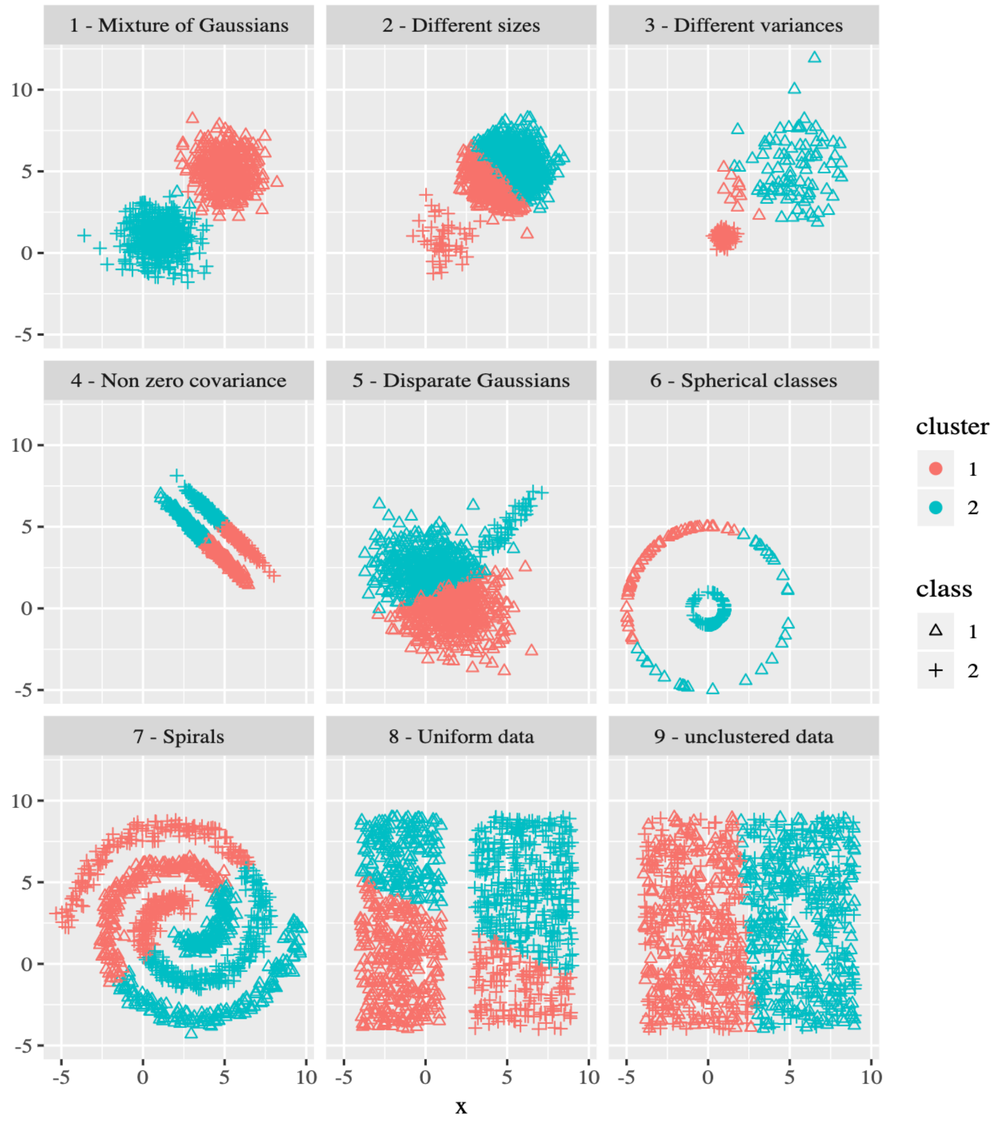


Image source: https://www.geeksforgeeks.org/ml-k-means-algorithm/



k-means deals with different kinds of data distribution

Image source: https://smorbieu.gitlab.io/k-means-is-not-all-about-sunshines-and-rainbows/#k-means-assumptions-and-criterion

Advantages and Disadvantages

- Advantages:
  - Relatively simple to implement.
  - Efficient and able to scale to large data sets.
  - Guarantees convergence.
- Disadvantages:
  - Choosing k manually.
  - Being dependent on initial values.
  - Unable to handle non-spherical clusters.
  - Scaling with number of dimensions.

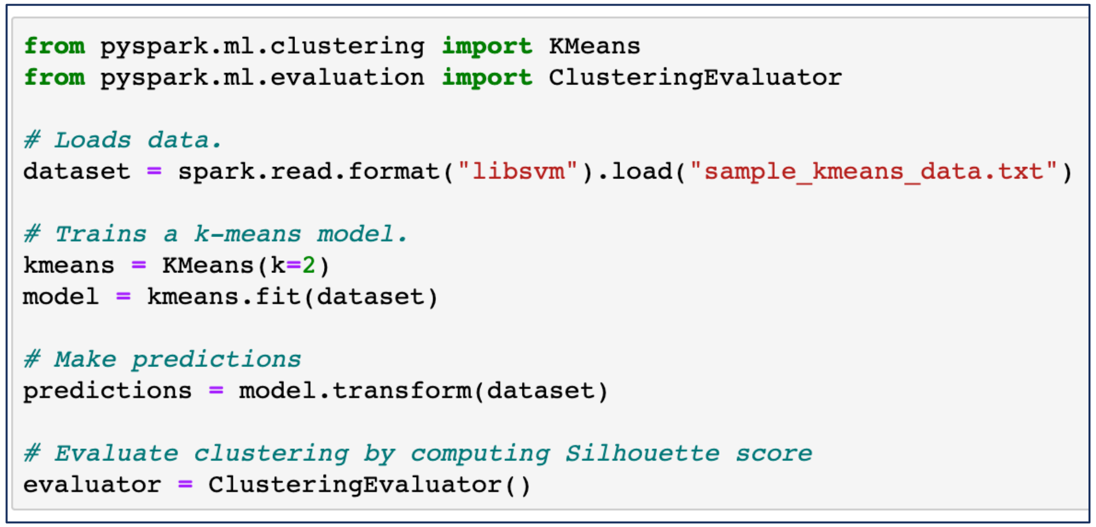
MLlib API

class `pyspark.ml.clustering.`**`KMeans`**(*featuresCol='features', predictionCol='prediction', k=2, initMode='k-means||', initSteps=2, tol=0.0001, maxIter=20, seed=None, distanceMeasure='euclidean'*) [source]

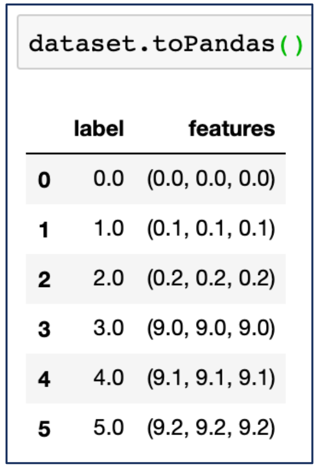
- It is implemented with a k-means++ like initialization mode (the k-means|| algorithm by Bahmani et al).
- Commonly used hyperparameter:
  - **k:** The number of clusters to create. Must be > 1.
  - **initMode:** The initialization algorithm. This can be either "random" to choose random points as initial cluster centroids, or "k-means||" to use a parallel variant of k-means++.
  - **initSteps:** The number of steps for k-means|| initialization mode. Must be > 0.
  - **distanceMeasure:** the distance measure. Supported options: 'euclidean' and 'cosine'.
  - **maxIter:** max number of iterations (>= 0).
  - **tol:** the convergence tolerance for iterative algorithms (>= 0).

Source: https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.clustering.KMeans

MLlib Example

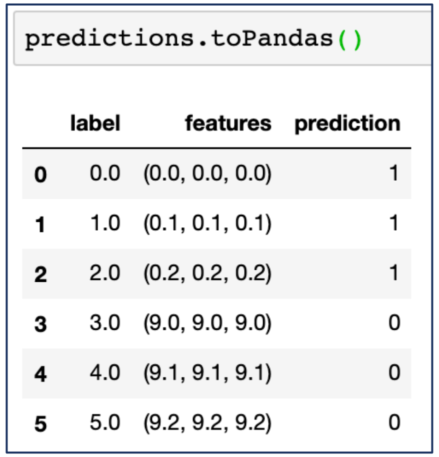


```python
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

# Loads data.
dataset = spark.read.format("libsvm").load("sample_kmeans_data.txt")

# Trains a k-means model.
kmeans = KMeans(k=2)
model = kmeans.fit(dataset)

# Make predictions
predictions = model.transform(dataset)

# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
```



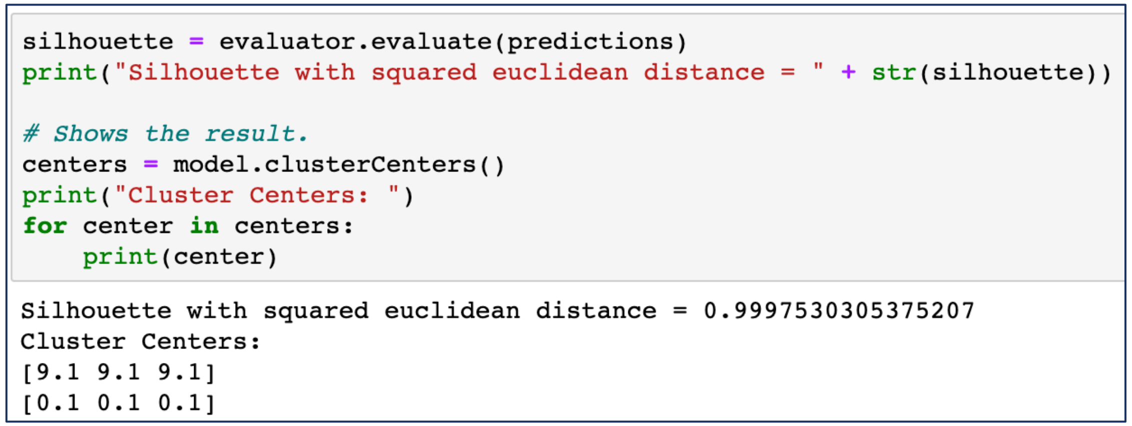
```
dataset.toPandas()
```

	label	features
0	0.0	(0.0, 0.0, 0.0)
1	1.0	(0.1, 0.1, 0.1)
2	2.0	(0.2, 0.2, 0.2)
3	3.0	(9.0, 9.0, 9.0)
4	4.0	(9.1, 9.1, 9.1)
5	5.0	(9.2, 9.2, 9.2)



```
predictions.toPandas()
```

	label	features	prediction
0	0.0	(0.0, 0.0, 0.0)	1
1	1.0	(0.1, 0.1, 0.1)	1
2	2.0	(0.2, 0.2, 0.2)	1
3	3.0	(9.0, 9.0, 9.0)	0
4	4.0	(9.1, 9.1, 9.1)	0
5	5.0	(9.2, 9.2, 9.2)	0



```python
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Silhouette with squared euclidean distance = 0.9997530305375207
Cluster Centers:
[9.1 9.1 9.1]
[0.1 0.1 0.1]
```

SPECTRAL CLUSTERING

Deal with Non-Spherical Data

- One of the problems of k-means is that it is not able to deal with non-spherical data.
- When the data is non-spherical, using centroids to represent the cluster center is meaningless.

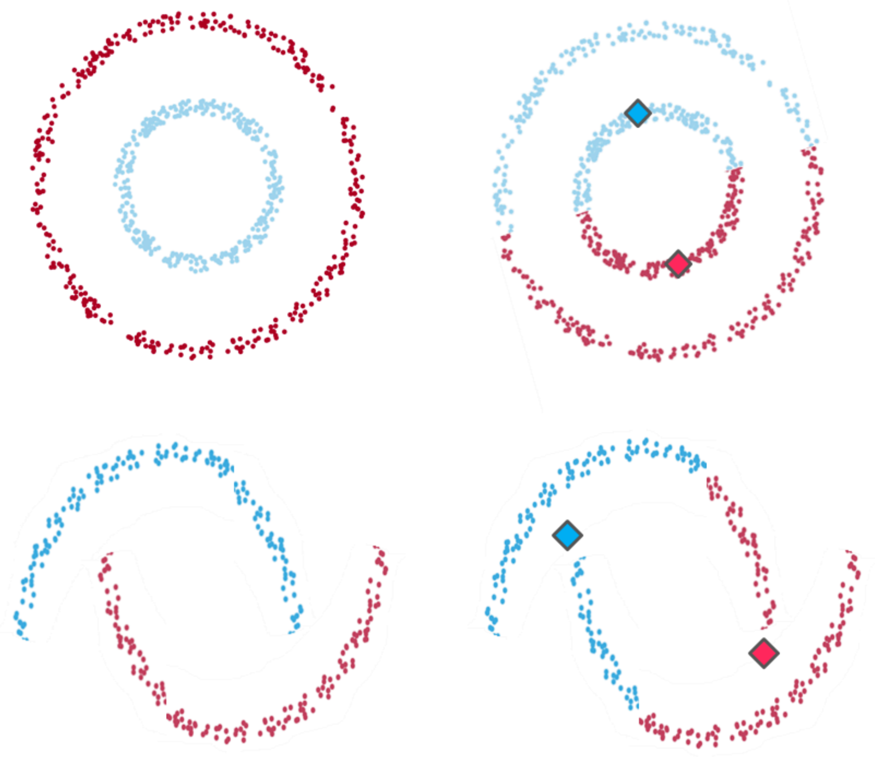
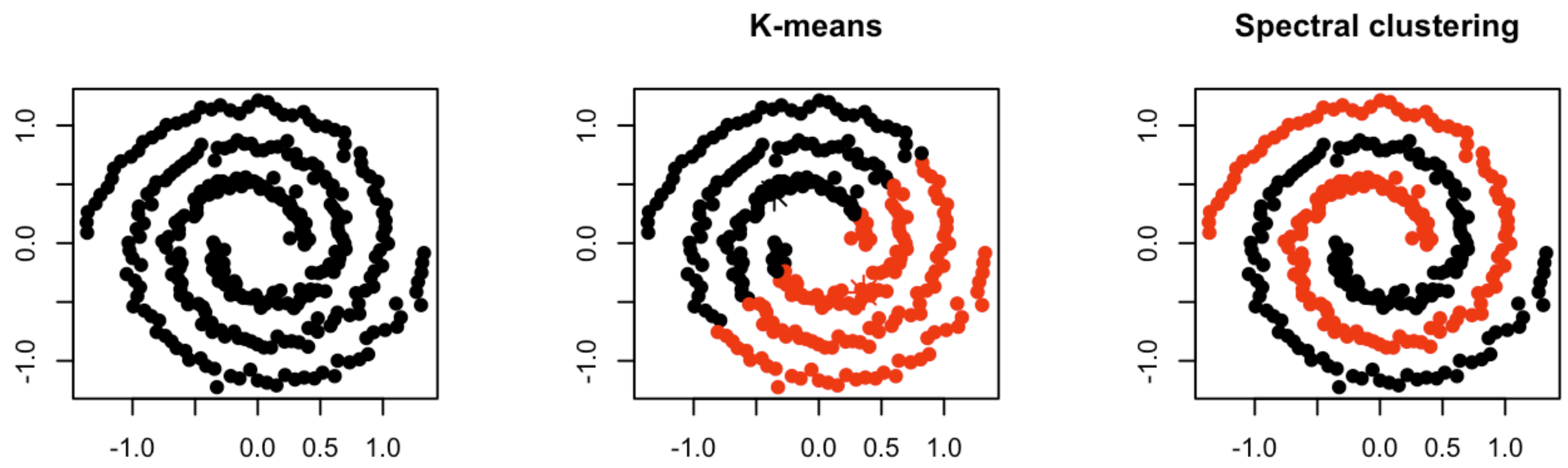


Image source: https://www.kdnuggets.com/2019/05/guide-k-means-clustering-algorithm.html

k-means vs. Spectral Clustering



- The result produced by spectral clustering is wonderful!
  - How to achieve that?

Image source: http://scalefreegan.github.io/Teaching/DataIntegration/practicals/p2.html

Spectral Clustering

- Instead of clustering data points in their original Euclidean space, cluster them in the space spanned by the "significant" eigenvectors of the normalized Laplacian matrix.
- Affinity matrix: a matrix A where A_{ij} is the similarity between data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.
  - E.g. heat kernel: $A_{ij} = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/\sigma^2)$.
- Normalized Laplacian matrix: $W = I - D^{-1}A$.
  - I is the identity matrix.
  - D is a diagonal matrix where $D_{ii} = \sum_{j=1}^{n} A_{ij}$.

Review of Eigenvector and Eigenvalue

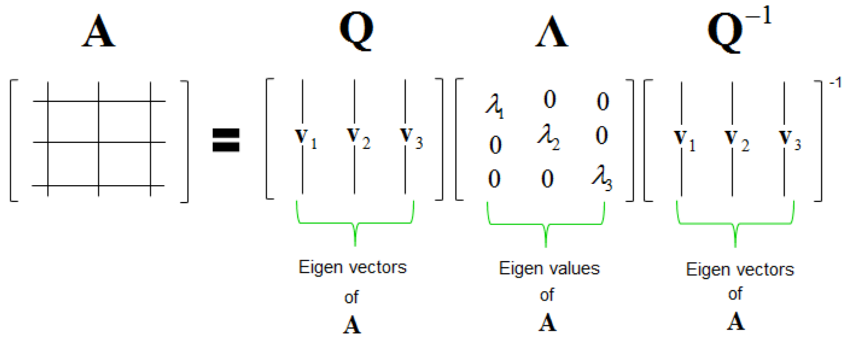
- A (non-zero) vector $\boldsymbol{v}$ of dimension N is an *eigenvector* of a square $N\times N$ matrix A if it satisfies the linear equation

$$A\boldsymbol{v} = \lambda\boldsymbol{v}$$

  where λ is a scalar, termed the *eigenvalue* corresponding to $\boldsymbol{v}$.

- Put all the eigenvectors $\boldsymbol{v}$ in $n\times n$ matrix Q whose ith column is the eigenvector v_i of A and all the eigenvalues λ in the diagonal of Λ:

$$A = Q\Lambda Q^{-1}$$



XIAMEN UNIVERSITY MALAYSIA 廈門大學馬來西亞分校

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY

厦门大学计算机科学系 Computer Science Department of Xiamen University

Image source: https://www.sharetechnote.com/html/Handbook_EngMath_Matrix_EigenDecomposition.html

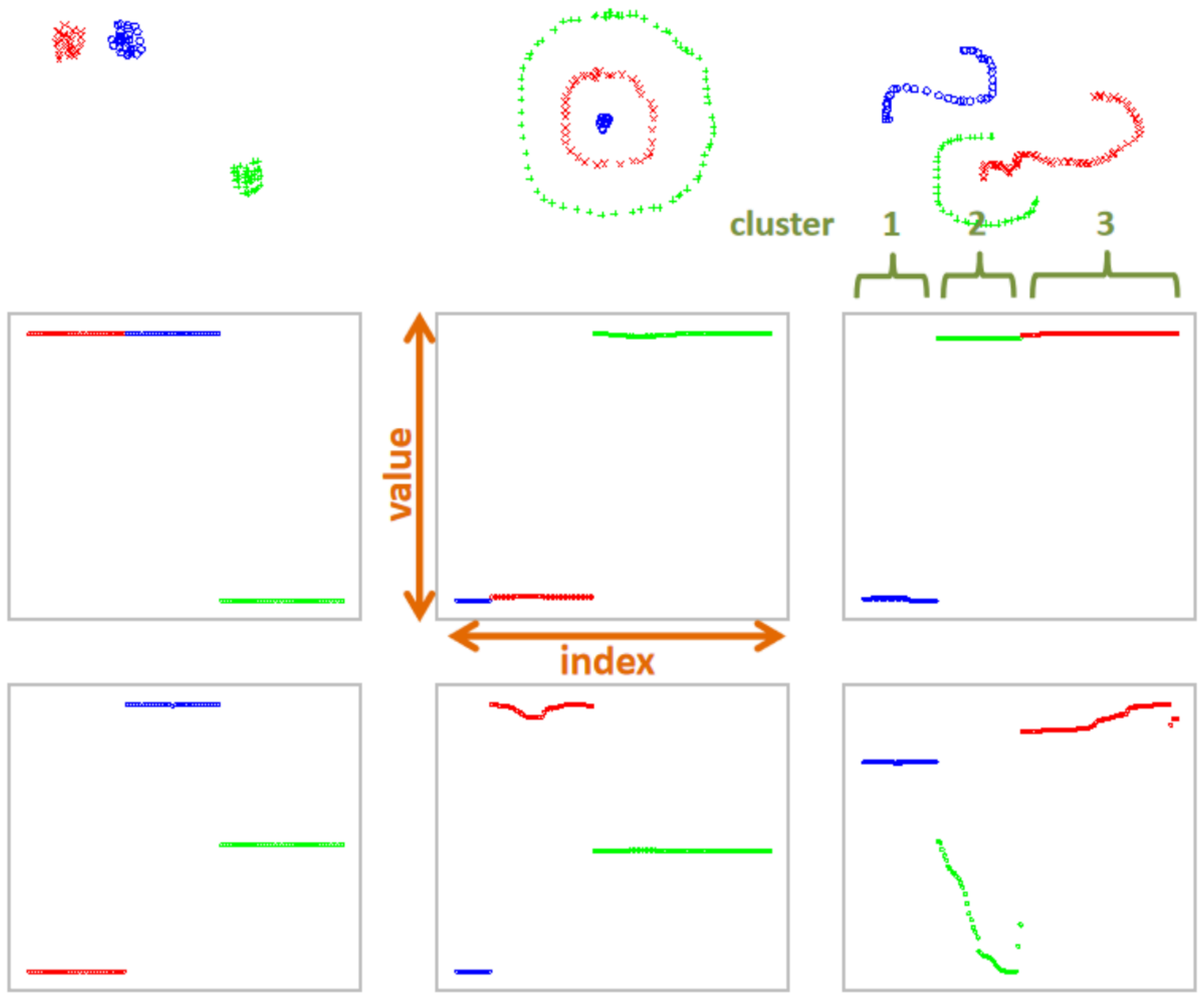
Review of Eigenvector and Eigenvalue

- When A is a real symmetric matrix, the eigenvectors can be chosen such that they are orthogonal to each other.
  - They can be used to represent the low-dimensional embedding of the matrix A.

Original data space

1st eigenvector

2nd eigenvector



Spectral Clustering

1. Choose k and similarity function s.
2. Derive A from s, let $W = I - D^{-1}A$.
3. Find eigenvectors and corresponding eigenvalues of W.
4. Pick the k eigenvectors of W with the smallest corresponding eigenvalues as "significant" eigenvectors.
5. Project the data points onto the space spanned by these vectors.
6. Run k-means on the projected data points.

Solution for Big Data

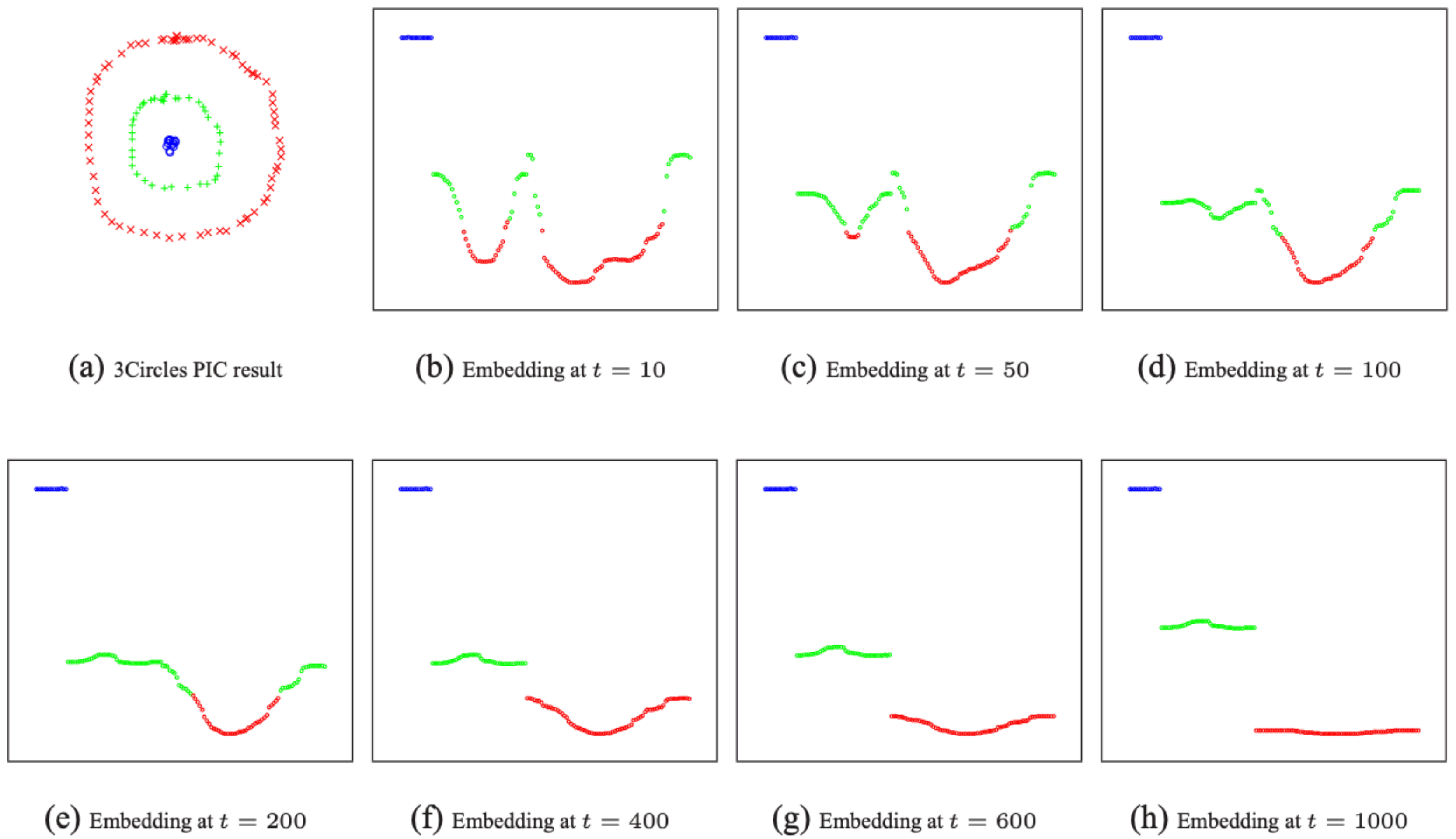
- When the data is very large, it is impossible to calculate it eigenvectors.
- Can we find a low-dimensional embedding for clustering, as spectral clustering, but without calculating these eigenvectors?
  - Use the power of distributed computing with iterations.

Power Iteration Clustering

- Power Iteration Clustering (PIC) is a simple iterative method for finding the dominant eigenvector of a matrix:

$$\boldsymbol{v}^{t+1} = cW\boldsymbol{v}^{t}$$

  - We set $W = D^{-1}A$, which is called row-normalized affinity matrix.
  - $\boldsymbol{v}^{t}$ is the vector at iteration t; $\boldsymbol{v}^{0}$ is typically a random vector.
  - c is a normalizing constant to avoid $\boldsymbol{v}^{t}$ from getting too large or too small.
- Typically converges quickly, and is fairly efficient if W is a sparse matrix.



(a) 3Circles PIC result (b) Embedding at $t = 10$ (c) Embedding at $t = 50$ (d) Embedding at $t = 100$

(e) Embedding at $t = 200$ (f) Embedding at $t = 400$ (g) Embedding at $t = 600$ (h) Embedding at $t = 1000$

Figure 1: Clustering result and the embedding provided by $\mathbf{v}^t$ for the 3Circles dataset. In (b) through (h), the value of each component of $\mathbf{v}^t$ is plotted against its index.

Image source: Lin, Frank, and William W. Cohen. "Power iteration clustering”, ICML, 2010.

Power Iteration Clustering

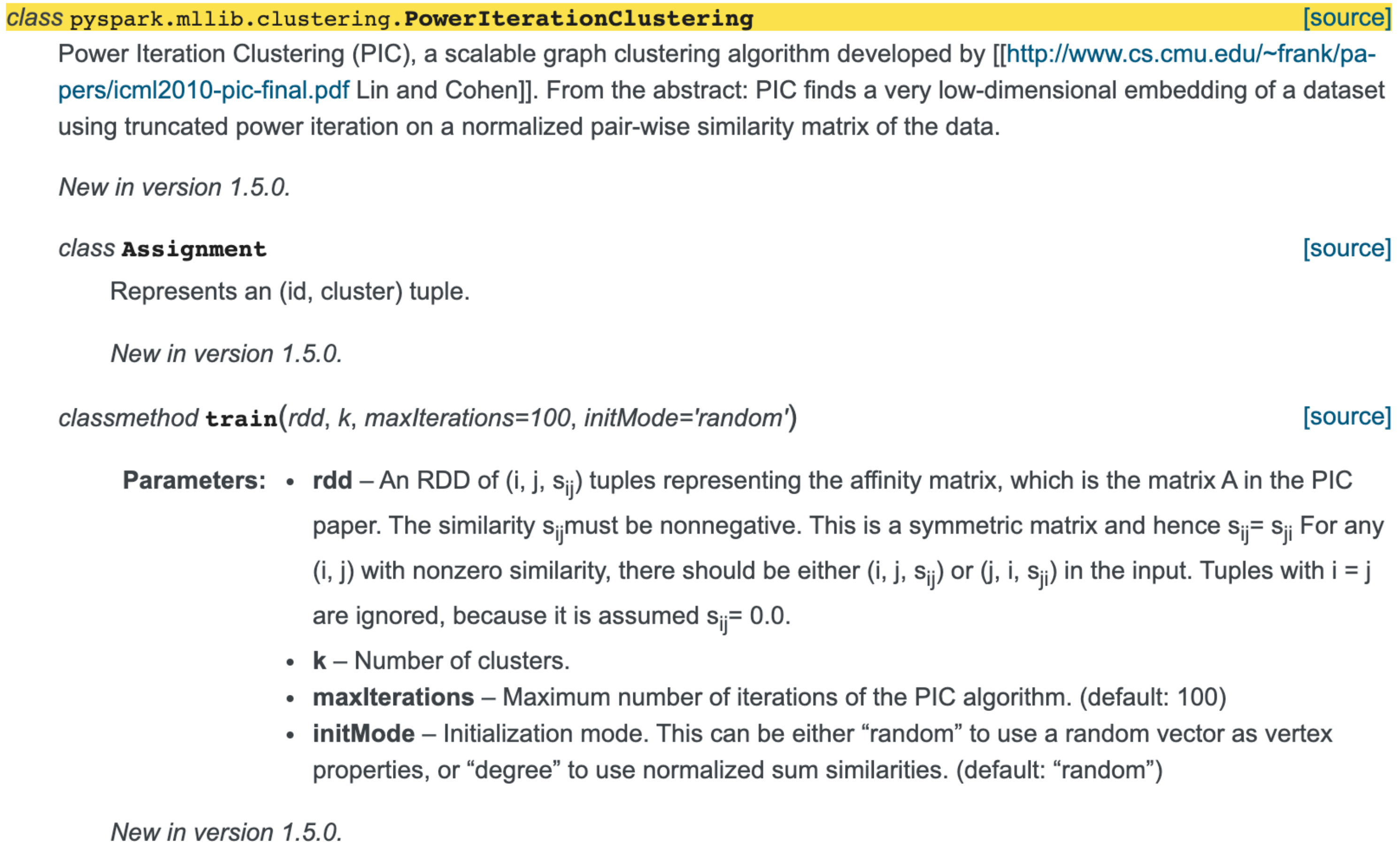
- Input: A row-normalized affinity matrix W and the number of clusters k.
- Output: Clusters $C_1, C_2, \ldots, C_k$.
- Pick an initial vector $\boldsymbol{v}^0$.
- Repeat
  - Set $\boldsymbol{v}^{t+1} \leftarrow \frac{W\boldsymbol{v}^t}{\|W\boldsymbol{v}^t\|_1}$.
  - Set $\delta^{t+1} \leftarrow |\boldsymbol{v}^{t+1} - \boldsymbol{v}^t|$.
  - Increment t.
  - Stop when $|\delta^t - \delta^{t-1}| \approx 0$.
- Use k-means to cluster points on $\boldsymbol{v}^t$ and return clusters $C_1, C_2, \ldots, C_k$.

Advantages and Disadvantages

- Advantages:
  - Elegant, and well-founded mathematically.
  - Handle clusters with any shape.
- Disadvantages
  - Very noisy datasets cause problems
    - Informative eigenvectors need not be in top few.
    - Performance can drop suddenly from good to terrible.
  - Computational cost is relative high.

MLlib API

- Currently, PIC is only available on RDD-based MLlib.



class `pyspark.mllib.clustering.PowerIterationClustering` [source]

Power Iteration Clustering (PIC), a scalable graph clustering algorithm developed by [[http://www.cs.cmu.edu/~frank/papers/icml2010-pic-final.pdf Lin and Cohen]]. From the abstract: PIC finds a very low-dimensional embedding of a dataset using truncated power iteration on a normalized pair-wise similarity matrix of the data.

New in version 1.5.0.

class **`Assignment`** [source]

Represents an (id, cluster) tuple.

New in version 1.5.0.

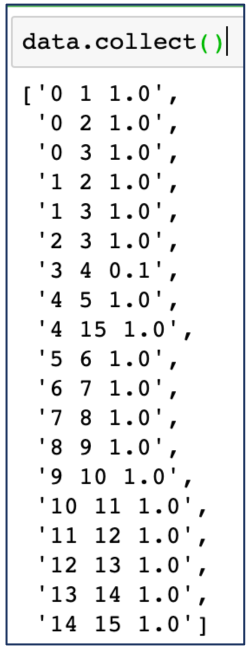
classmethod **`train`**(*rdd, k, maxIterations=100, initMode='random'*) [source]

Parameters:
- **rdd** – An RDD of (i, j, s_{ij}) tuples representing the affinity matrix, which is the matrix A in the PIC paper. The similarity s_{ij}must be nonnegative. This is a symmetric matrix and hence s_{ij}= s_{ji} For any (i, j) with nonzero similarity, there should be either (i, j, s_{ij}) or (j, i, s_{ji}) in the input. Tuples with i = j are ignored, because it is assumed s_{ij}= 0.0.
- **k** – Number of clusters.
- **maxIterations** – Maximum number of iterations of the PIC algorithm. (default: 100)
- **initMode** – Initialization mode. This can be either “random” to use a random vector as vertex properties, or “degree” to use normalized sum similarities. (default: “random”)

New in version 1.5.0.

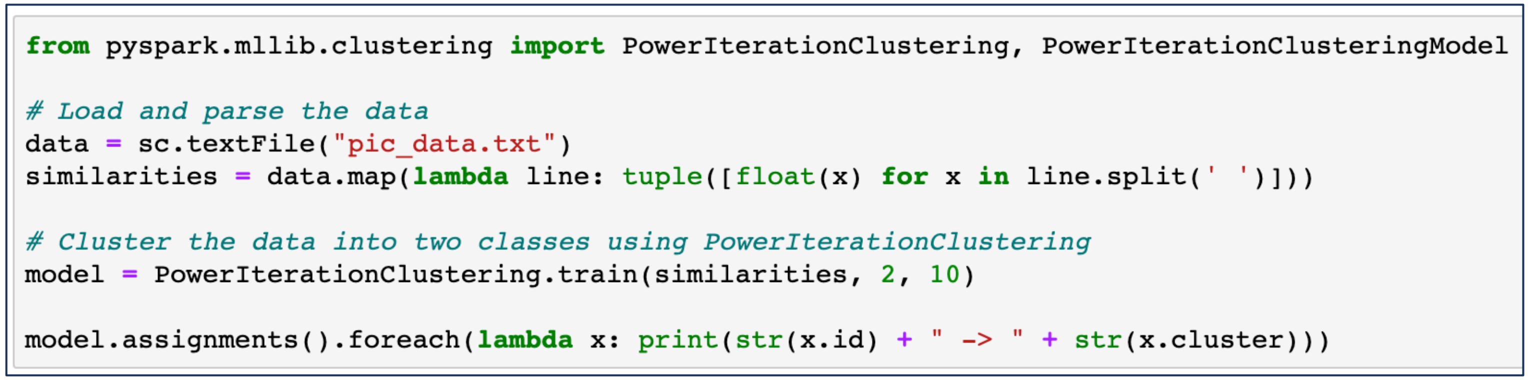
Source: https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html#pyspark.mllib.clustering.PowerIterationClustering

MLlib Example

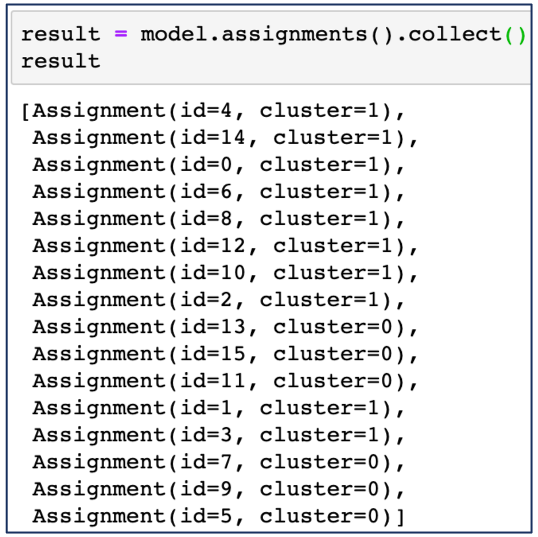


```
data.collect()
```

```
['0 1 1.0',
 '0 2 1.0',
 '0 3 1.0',
 '1 2 1.0',
 '1 3 1.0',
 '2 3 1.0',
 '3 4 0.1',
 '4 5 1.0',
 '4 15 1.0',
 '5 6 1.0',
 '6 7 1.0',
 '7 8 1.0',
 '8 9 1.0',
 '9 10 1.0',
 '10 11 1.0',
 '11 12 1.0',
 '12 13 1.0',
 '13 14 1.0',
 '14 15 1.0']
```



```
from pyspark.mllib.clustering import PowerIterationClustering, PowerIterationClusteringModel

# Load and parse the data
data = sc.textFile("pic_data.txt")
similarities = data.map(lambda line: tuple([float(x) for x in line.split(' ')]))

# Cluster the data into two classes using PowerIterationClustering
model = PowerIterationClustering.train(similarities, 2, 10)

model.assignments().foreach(lambda x: print(str(x.id) + " -> " + str(x.cluster)))
```



```
result = model.assignments().collect()
result
```

```
[Assignment(id=4, cluster=1),
 Assignment(id=14, cluster=1),
 Assignment(id=0, cluster=1),
 Assignment(id=6, cluster=1),
 Assignment(id=8, cluster=1),
 Assignment(id=12, cluster=1),
 Assignment(id=10, cluster=1),
 Assignment(id=2, cluster=1),
 Assignment(id=13, cluster=0),
 Assignment(id=15, cluster=0),
 Assignment(id=11, cluster=0),
 Assignment(id=1, cluster=1),
 Assignment(id=3, cluster=1),
 Assignment(id=7, cluster=0),
 Assignment(id=9, cluster=0),
 Assignment(id=5, cluster=0)]
```

Source: https://spark.apache.org/docs/latest/mllib-clustering.html#power-iteration-clustering-pic

Evaluation Metrics for Clustering

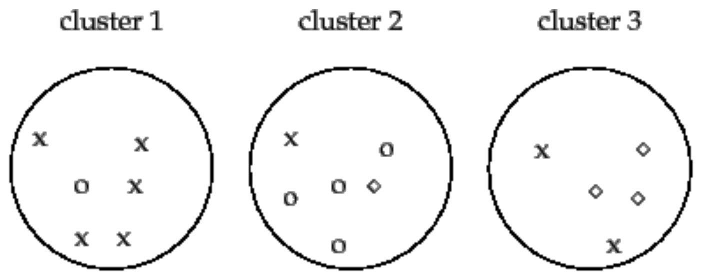
- Generally, there are two categories of metrics to evaluate the performance of a clustering algorithm:
  - External criterion: evaluate the clustering results with ground truth.
  - Internal criterion: evaluate the clustering results without ground truth.

External Criterion: Purity

- We define $\Omega = \{\omega_1, \omega_2, \dots, \omega_K\}$ as the set of obtained clusters and $C = \{c_1, c_2, \dots, c_J\}$ as the set of ground truth classes.
- Purity is a simple and transparent evaluation measure.
- Each cluster is assigned to the class which is most frequent in the cluster, and then count the number of correct assignments and dividing by n. Formally:

$$purity(\Omega, C) = \frac{1}{n} \sum_k \max_j |\omega_k \cap c_j| .$$

External Criterion: Purity



► **Figure 16.1** Purity as an external evaluation criterion for cluster quality. Majority class and number of members of the majority class for the three clusters are: x, 5 (cluster 1); o, 4 (cluster 2); and ◇, 3 (cluster 3). Purity is $(1/17) \times (5+4+3) \approx 0.71$.

- High purity is easy to achieve when the number of clusters is large.
  - In particular, purity is 1 if each data point gets its own cluster.
- Thus, we cannot use purity to trade off the quality of the clustering against the number of clusters.

Image source: https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html

External Criterion: Normalized Mutual Information (NMI)

- A measure that allows us to make this tradeoff is NMI:

$$NMI(\Omega, C) = \frac{I(\Omega; C)}{[H(\Omega) + H(C)]/2}.$$

- I is mutual information:

$$I(\Omega; C) = \sum_k \sum_j P(\omega_k \cap c_j) \log \frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j)} = \sum_k \sum_j \frac{|\omega_k \cap c_j|}{n} \log \frac{n|\omega_k \cap c_j|}{|\omega_k||c_j|}.$$

- H is entropy:

$$H(\Omega) = -\sum_k P(\omega_k) \log P(\omega_k) = -\sum_k \frac{|\omega_k|}{n} \log \frac{|\omega_k|}{n}.$$

External Criterion: Normalized Mutual Information (NMI)

- Mutual Information tells us the reduction in the entropy of class labels that we get if we know the cluster labels. (Similar to Information gain in decision trees).
- Since it’s normalized we can measure and compare the NMI between different clustering results having different number of clusters.
- A step-by-step NMI calculation with examples can be found here.

Internal Criterion: Silhouette Coefficient

- If we don’t have ground truth labels, we can evaluate the quality of clusters by within-cluster and between-cluster similarity.
- Silhouette Coefficient is the default evaluation metric in MLlib.
- The Silhouette Coefficient s for a single data point is then given as:

$$s = \frac{b - a}{\max(a, b)}.$$

  - a is the mean distance between this data point and all other data points in the same cluster.
  - b is the mean distance between this data point and all other data points in other clusters.
- The Silhouette Coefficient for a dataset is given as the mean of the Silhouette Coefficient for each sample.

More about Clustering

- If you are interested in clustering algorithms and related issues, check sklearn clustering documentation and the wikipedia page.

PRINCIPLE COMPONENT ANALYSIS

Curse of Dimensionality

- In machine learning, we often have high-dimensional data.
  - If we're recording 60 different metrics for each of our shoppers, we're working in a space with 60 dimensions.
  - If we're analyzing grayscale images sized 50x50, we're working in a space with 2,500 dimensions.
  - If the images are RGB-colored, the dimensionality increases to 7,500 dimensions (one dimension for each color channel in each pixel in the image).

Curse of Dimensionality

- As the number of features increases, the classifier's performance increases as well until we reach the optimal number of features.
- Adding more features based on the same size as the training set will then degrade the classifier's performance.

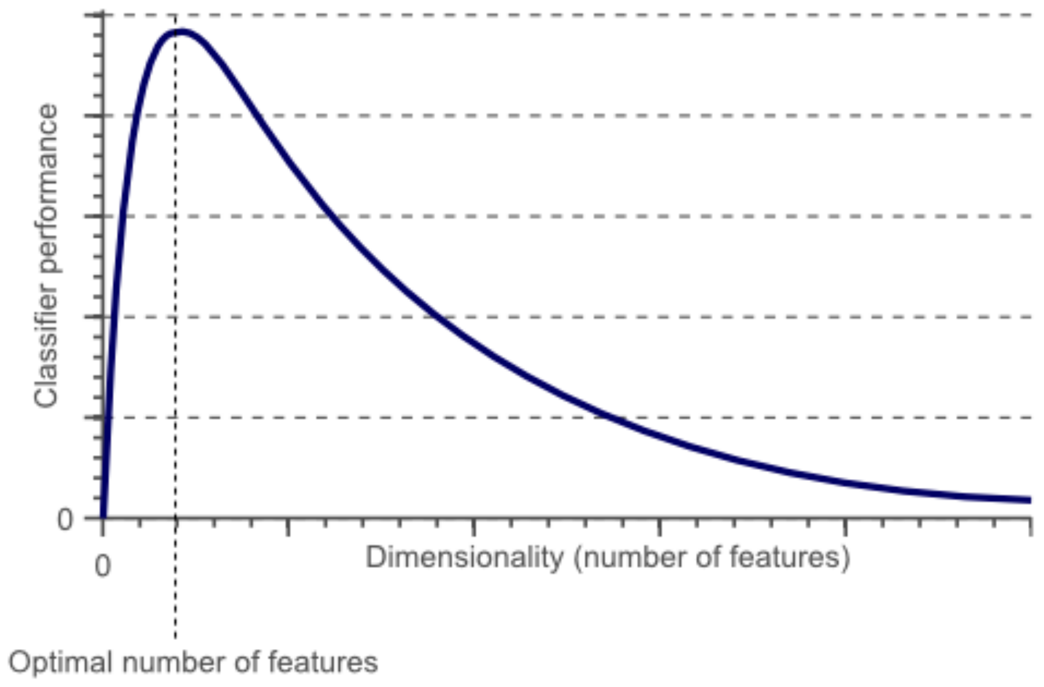


Image source: https://builtin.com/data-science/curse-dimensionality

Dimensionality Reduction

- To get rid of the curse of dimensionality, a process called *dimensionality reduction* was introduced.
- Dimensionality reduction techniques can be used to filter only a limited number of significant features needed for training.
- *Principal Components Analysis (PCA)* is a dimensionality reduction technique that enables you to identify correlations and patterns in a data set so that it can be transformed into a data set of significantly lower dimension **without loss of any important information**.

General Idea of Principal Components Analysis

- PCA is done by transforming the variables to a new set of variables, which are known as the *principal components*.
- The principal components are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order.
- So, in this way, the 1st principal component retains maximum variation that was present in the original components.
- The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

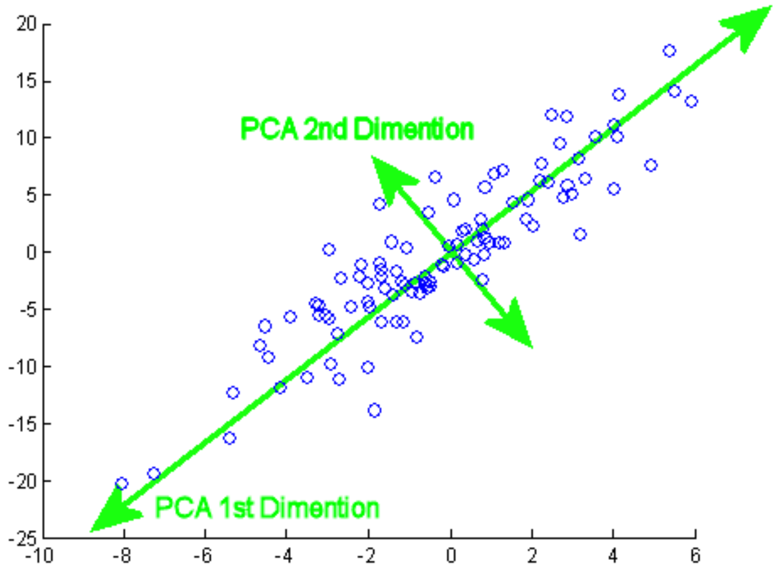


Image source: https://medium.com/@raghavan99o/principal-component-analysis-pca-explained-and-implemented-eeab7cb73b72

Steps of PCA

1. Standardize the dataset.
2. Calculate the covariance matrix for the features in the dataset.
3. Calculate the eigenvalues and eigenvectors for the covariance matrix.
4. Sort eigenvalues and their corresponding eigenvectors.
5. Pick k eigenvectors with top k eigenvalues to form a matrix.
6. Transform the original matrix.

Standardize

- Let the d-dimensional data matrix be $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_d]$.
- For the jth feature $\boldsymbol{x}_j$, subtract the mean μ_j and then divide it by its standard deviation.

$$\widehat{\boldsymbol{x}}_j = \frac{\boldsymbol{x}_j - \mu_j}{\sigma_j}.$$

- After that, each feature has 0 mean and 1 standard deviation.
- The standardized data matrix is $\hat{X} = [\widehat{\boldsymbol{x}}_1, \widehat{\boldsymbol{x}}_2, \ldots, \widehat{\boldsymbol{x}}_d]$.

Covariance Matrix

- Then calculate the covariance matrix C:

$$C = \begin{bmatrix} Cov(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_1) & Cov(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2) & \cdots & Cov(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_d) \\ Cov(\hat{\boldsymbol{x}}_2, \hat{\boldsymbol{x}}_1) & Cov(\hat{\boldsymbol{x}}_2, \hat{\boldsymbol{x}}_2) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ Cov(\hat{\boldsymbol{x}}_d, \hat{\boldsymbol{x}}_1) & Cov(\hat{\boldsymbol{x}}_d, \hat{\boldsymbol{x}}_2) & \cdots & Cov(\hat{\boldsymbol{x}}_d, \hat{\boldsymbol{x}}_d) \end{bmatrix}$$

Eigenvalues and Eigenvectors

- Calculate the eigenvalues and eigenvectors for the covariance matrix C.

$$C = Q\Lambda Q^{-1}$$

- Sort eigenvalues and their corresponding eigenvectors.

$$Q = [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_d]$$
$$\Lambda = diag([\lambda_1, \lambda_2, \dots, \lambda_d])$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$.

Data Transormation

- Pick k eigenvectors with top k eigenvalues to form a matrix Q_k.

$$Q_k = [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k]$$

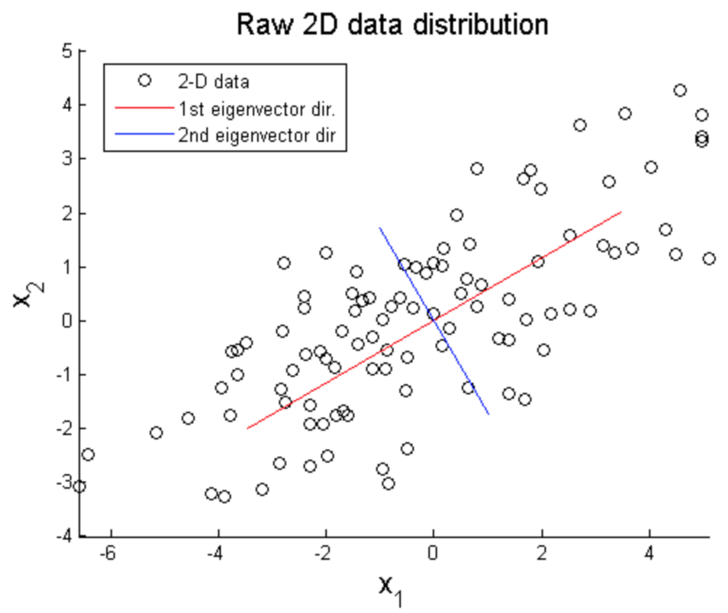
- Transform the original matrix to get the dimensional reduced data Y.

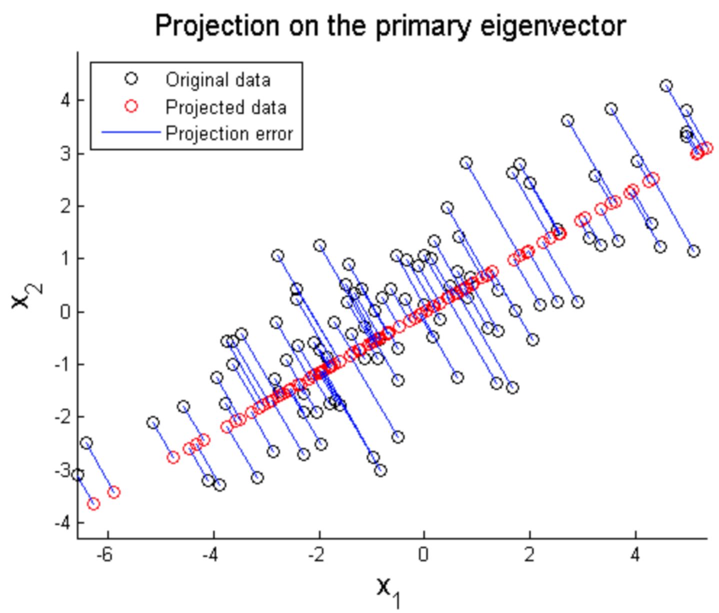
$$Y = \hat{X} Q_k$$

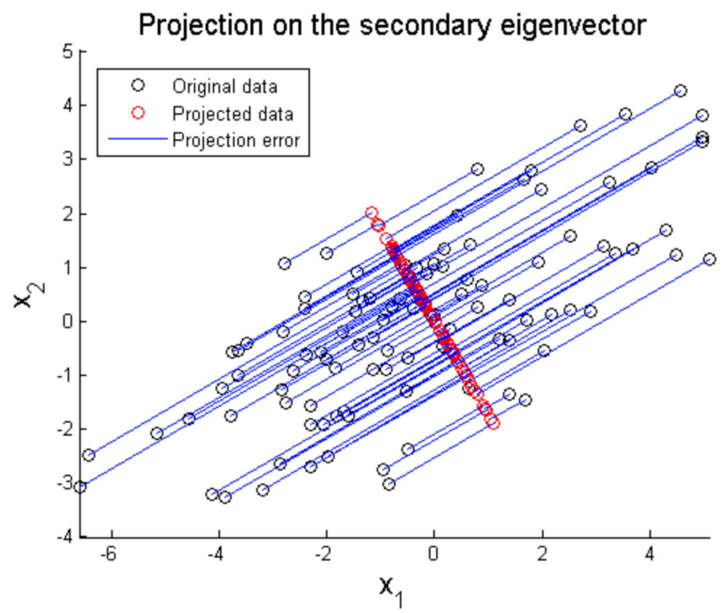
  where $\hat{X}$ is an $n \times d$ matrix and Q_k is a $d \times k$ matrix.

- Thus, Y is an $n \times k$ matrix, a k-dimensional data matrix.

PCA Example

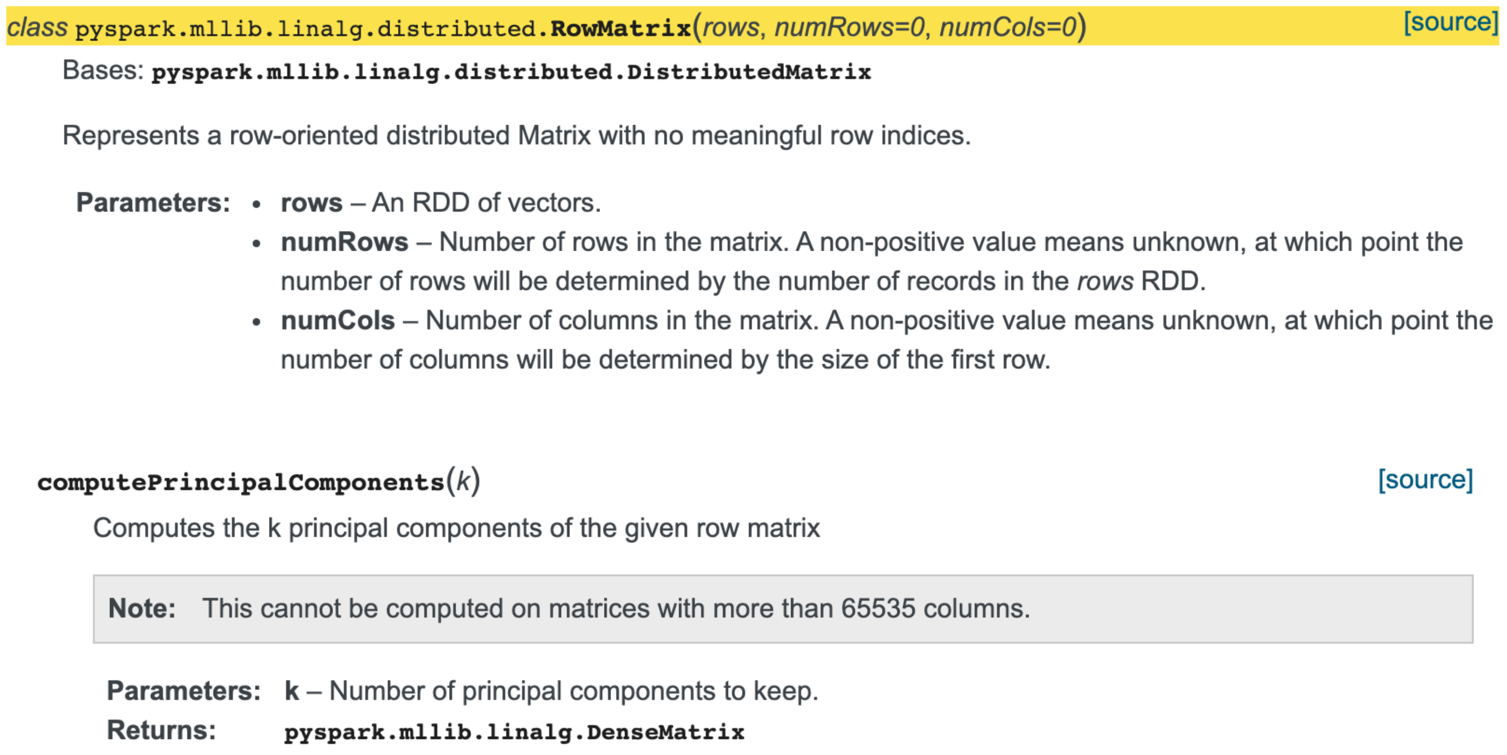

Raw 2D data distribution
2-D data
1st eigenvector dir.
2nd eigenvector dir
x2
x1
5
4
3
2
1
0
-1
-2
-3
-4
-6
-4
-2
0
2
4


Projection on the primary eigenvector
Original data
Projected data
Projection error
x2
x1
4
3
2
1
0
-1
-2
-3
-4
-6
-4
-2
0
2
4


Projection on the secondary eigenvector
Original data
Projected data
Projection error
x2
x1
5
4
3
2
1
0
-1
-2
-3
-4
-6
-4
-2
0
2
4

MLlib API

- Currently, PCA is only available on RDD-based MLlib.
- PCA is implemented as a function of class `RowMatrix`.
- There are some other useful functions to do operations on matrix:
  - `multiply()`
  - `columnSimilarities()`
  - `computeCovariance()`
  - `computeSVD()`



class **pyspark.mllib.linalg.distributed.RowMatrix**(*rows*, *numRows=0*, *numCols=0*) [source]

Bases: `pyspark.mllib.linalg.distributed.DistributedMatrix`

Represents a row-oriented distributed Matrix with no meaningful row indices.

Parameters:
- **rows** – An RDD of vectors.
- **numRows** – Number of rows in the matrix. A non-positive value means unknown, at which point the number of rows will be determined by the number of records in the *rows* RDD.
- **numCols** – Number of columns in the matrix. A non-positive value means unknown, at which point the number of columns will be determined by the size of the first row.

computePrincipalComponents(*k*) [source]

Computes the k principal components of the given row matrix

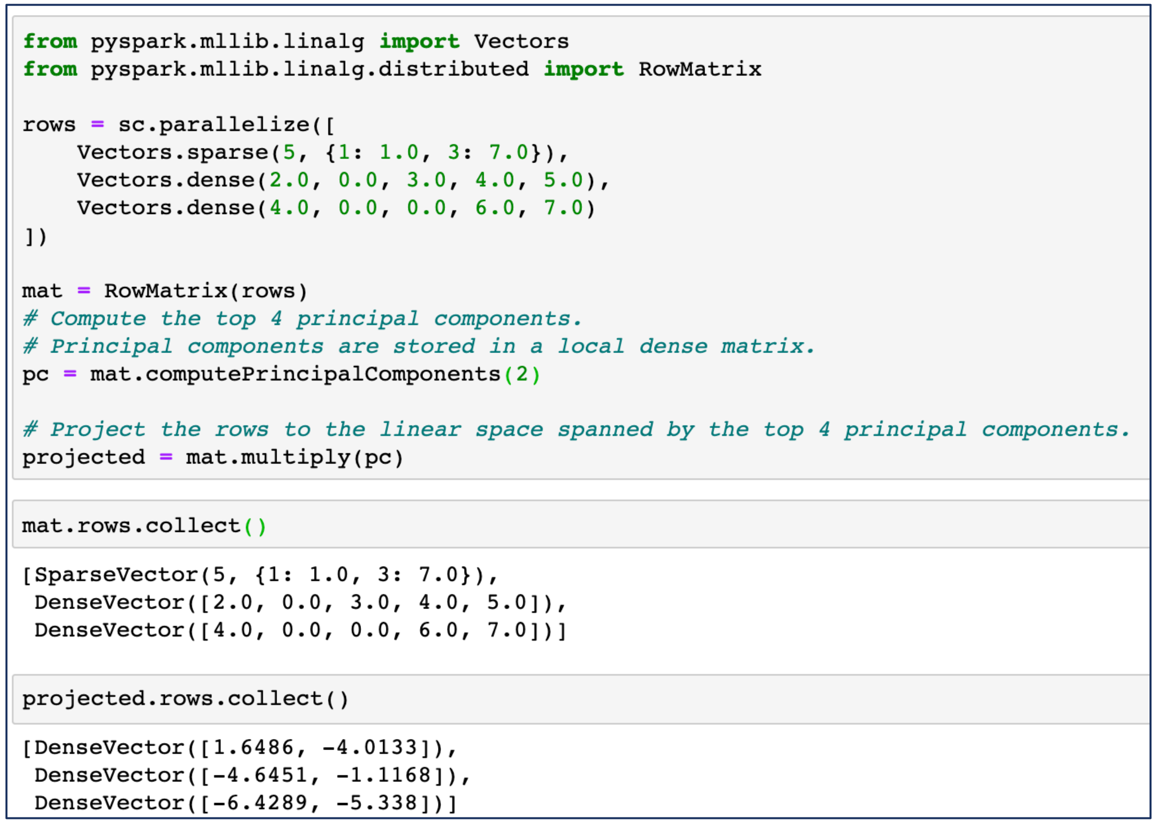
Note: This cannot be computed on matrices with more than 65535 columns.

Parameters: **k** – Number of principal components to keep.
Returns: `pyspark.mllib.linalg.DenseMatrix`

XIAMEN UNIVERSITY MALAYSIA 厦門大學馬來西亞分校 厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Source: https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html#pyspark.mllib.linalg.distributed.RowMatrix

MLlib Example



```
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.linalg.distributed import RowMatrix

rows = sc.parallelize([
    Vectors.sparse(5, {1: 1.0, 3: 7.0}),
    Vectors.dense(2.0, 0.0, 3.0, 4.0, 5.0),
    Vectors.dense(4.0, 0.0, 0.0, 6.0, 7.0)
])

mat = RowMatrix(rows)
# Compute the top 4 principal components.
# Principal components are stored in a local dense matrix.
pc = mat.computePrincipalComponents(2)

# Project the rows to the linear space spanned by the top 4 principal components.
projected = mat.multiply(pc)
```

```
mat.rows.collect()
```

```
[SparseVector(5, {1: 1.0, 3: 7.0}),
 DenseVector([2.0, 0.0, 3.0, 4.0, 5.0]),
 DenseVector([4.0, 0.0, 0.0, 6.0, 7.0])]
```

```
projected.rows.collect()
```

```
[DenseVector([1.6486, -4.0133]),
 DenseVector([-4.6451, -1.1168]),
 DenseVector([-6.4289, -5.338])]
```

XIAMEN UNIVERSITY MALAYSIA 厦門大學馬來西亞分校

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY

厦门大学计算机科学系 Computer Science Department of Xiamen University

Source: https://spark.apache.org/docs/latest/mllib-dimensionality-reduction#principal-component-analysis-pca

Feature Normalization

- For numerical features, each feature has different scale.
- For example, as features of a house:
  - Price is at the scale of \$100,000.
  - Size is at the scale of 1,000 ft^2.
  - Distance to the downtown is at the scale of 10km.
  - House age is at the scale of 10 years.
- The features with high magnitudes will weigh in a lot more in weighted combination or distance calculations than features with low magnitudes.
- The preprocessing step to align features to the same scale is called *feature normalization* or *feature scaling*.

Image source: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

Feature Normalization

- Feature normalization is a general requirement for many machine learning algorithms.
  - For the algorithms that needs to use gradient descent, e.g. logistic regression, SVMs, perceptrons, neural networks etc., if features are on different scales, certain weights may update faster than others.
  - For the algorithms that needs to calculate distances between data points, e.g. k-nearest neighbor, k-means etc., the feature with large scale will dominant the distance.
- In fact, tree-based classifier are probably the only classifiers where feature scaling doesn’t make a difference.

Z-score Normalization

- *Z-score normalization*, aka *standardization*, is that the features will be rescaled so that they'll have the properties of a standard normal distribution with

$$\mu = 0 \text{ and } \sigma = 1$$

- μ is the mean (average) and σ is the standard deviation from the mean. Standard scores (also called *z*-scores) of the samples are calculated as follows:

$$z = \frac{x - \mu}{\sigma}.$$

Min-Max Scaling

- *Min-max scaling* scales the data to a fixed range - usually 0 to 1.
- A Min-max scaling is typically done via the following equation:

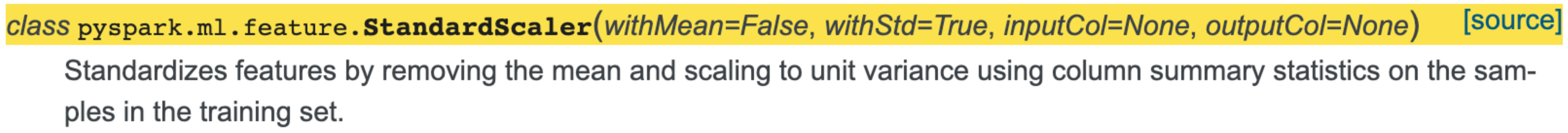
$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Min-max scaling will cause problems when a feature has outliers.
  - E.g. a feature with all values in [0,10] except a outlier 10000. After min-max scaling, the outlier becomes 1 and all the other values are in [0,0.001].

Feature Normalization

- One important thing to notice is that, when the data is seperated into training and test set, it is improper to do feature normalization on them together.
  - Information of test set will be utilized.
- One should scale the training data and use the scaling information (e.g. μ, σ, x_{min}, x_{max}) to scale the test data.

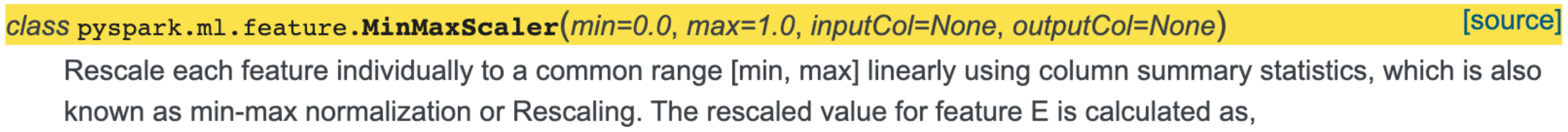
MLlib API



class `pyspark.ml.feature.`**`StandardScaler`**(*withMean=False, withStd=True, inputCol=None, outputCol=None*) [source]

Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set.

- Parameters:
  - **withMean**: data with mean.
  - **withStd**: Scale to unit standard deviation.



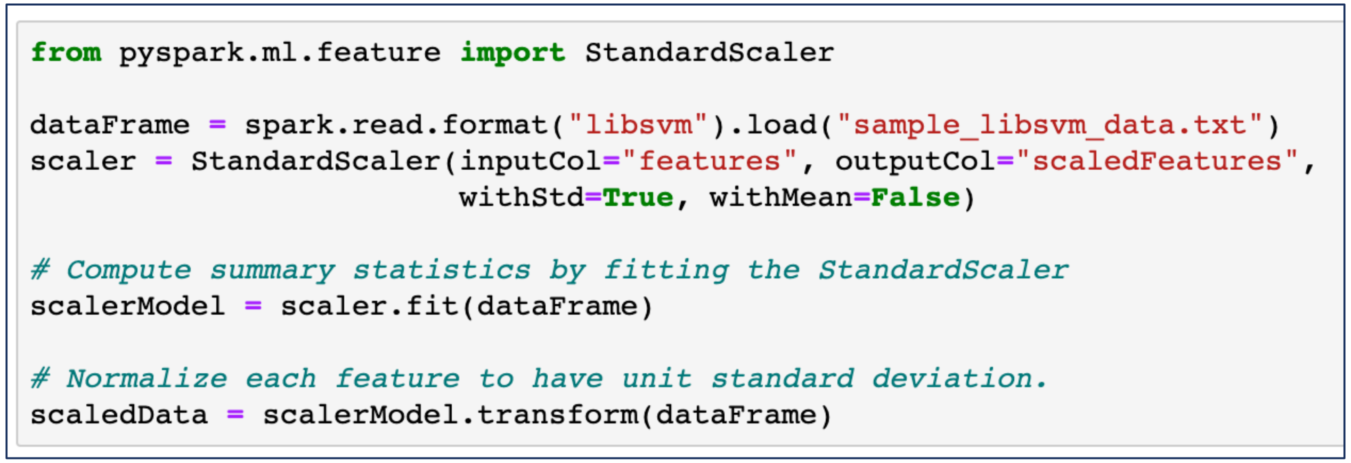
class `pyspark.ml.feature.`**`MinMaxScaler`**(*min=0.0, max=1.0, inputCol=None, outputCol=None*) [source]

Rescale each feature individually to a common range [min, max] linearly using column summary statistics, which is also known as min-max normalization or Rescaling. The rescaled value for feature E is calculated as,

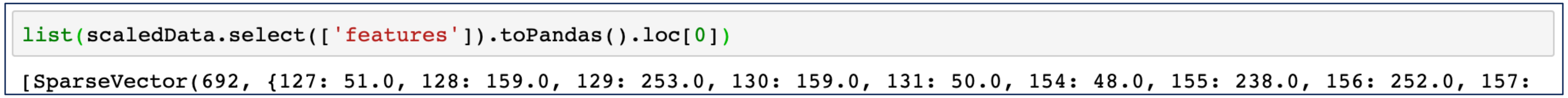
- Parameters:
  - **max**: Upper bound of the output feature range.
  - **min**: Lower bound of the output feature range.

Source: https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.feature.MinMaxScaler

MLlib Example

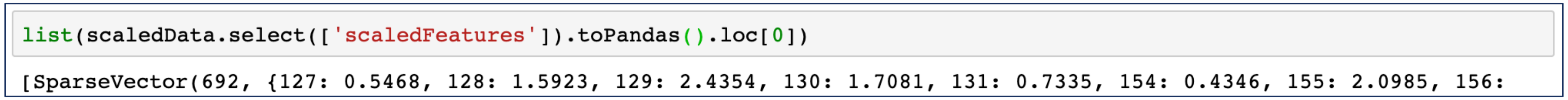


```python
from pyspark.ml.feature import StandardScaler

dataFrame = spark.read.format("libsvm").load("sample_libsvm_data.txt")
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)

# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(dataFrame)

# Normalize each feature to have unit standard deviation.
scaledData = scalerModel.transform(dataFrame)
```



```python
list(scaledData.select(['features']).toPandas().loc[0])
```

```
[SparseVector(692, {127: 51.0, 128: 159.0, 129: 253.0, 130: 159.0, 131: 50.0, 154: 48.0, 155: 238.0, 156: 252.0, 157:
```

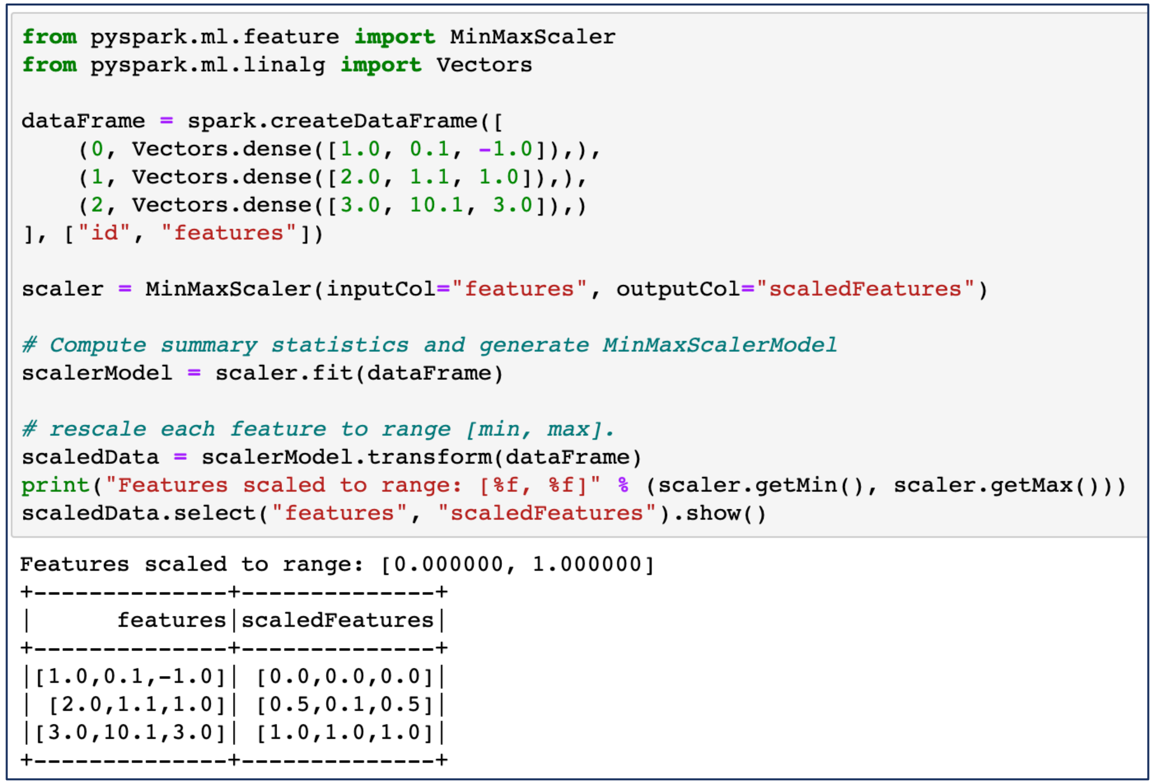


```python
list(scaledData.select(['scaledFeatures']).toPandas().loc[0])
```

```
[SparseVector(692, {127: 0.5468, 128: 1.5923, 129: 2.4354, 130: 1.7081, 131: 0.7335, 154: 0.4346, 155: 2.0985, 156:
```

Source: https://spark.apache.org/docs/latest/ml-features.html#standardscaler

MLlib Example



```
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.linalg import Vectors

dataFrame = spark.createDataFrame([
    (0, Vectors.dense([1.0, 0.1, -1.0]),),
    (1, Vectors.dense([2.0, 1.1, 1.0]),),
    (2, Vectors.dense([3.0, 10.1, 3.0]),)
], ["id", "features"])

scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

# Compute summary statistics and generate MinMaxScalerModel
scalerModel = scaler.fit(dataFrame)

# rescale each feature to range [min, max].
scaledData = scalerModel.transform(dataFrame)
print("Features scaled to range: [%f, %f]" % (scaler.getMin(), scaler.getMax()))
scaledData.select("features", "scaledFeatures").show()
```

```
Features scaled to range: [0.000000, 1.000000]
+--------------+--------------+
|      features|scaledFeatures|
+--------------+--------------+
|[1.0,0.1,-1.0]| [0.0,0.0,0.0]|
| [2.0,1.1,1.0]| [0.5,0.1,0.5]|
|[3.0,10.1,3.0]| [1.0,1.0,1.0]|
+--------------+--------------+
```

Source: https://spark.apache.org/docs/latest/ml-features.html#minmaxscaler

Conclusion

After this lecture, you should know:

- What is clustering.
- How k-means works.
- What is the difference between k-means and spectral clustering.
- Why do we need dimensionality reduction.
- Why do we need feature normalization.

Assignment 4

- Assignment 4 is released. The deadline is **18:00, 29th June**.

Thank you!

- Reference:
  - PIC paper: Lin, Frank, and William W. Cohen. "Power iteration clustering”, ICML, 2010.
  - sklearn clustering documentation: https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation.

Acknowledgement: Thankfully acknowledge slide contents shared by Prof. Yongxuan Lai