PyClustering: K-Means Tutorial

Pyclustering Library Tutorial
Theme: K-Means

Library Version: 0.8.0
Author: Andrei Novikov
E-Mail: pyclustering@yandex.ru

PY Clustering

Agenda

- PyClustering K-Means Features;
- Input format for K-Means;
- Data clustering using K-Means;
- Additional parameters of the algorithm;
- Data clustering result visualization;
- Image segmentation by K-Meas;

PyClustering K-Means Features

- Python implementation based on numpy.
- C/C++ implementation – core library that is supported for 32, 64-bit Windows, Linux.
- K-Means observer to collect information about clustering process on each iteration.
- K-Means visualizer to display and animate K-Means specific results.

C/C++ implementation can be used without python if integration with C/C++ project is required. Use sources from „*pyclustering/ccore*“.

Import K-Means

K-Means algorithm and its features can be imported from „**pyclustering.cluster.kmeans**“:

```
from pyclustering.cluster.kmeans import kmeans
from pyclustering.cluster.kmeans import kmeans_observer
from pyclustering.cluster.kmeans import kmeans_visualizer
```

Input data format for K-Means

K-Means algorithm uses array_like data format, for example, list where each element is point that is represented by coordinates. Here is an example of 1-D data:

dataset = [[0.25], [0.43], [1.34], [-0.56]]

Example of 2-D data:

dataset = [[0.1, 0.2], [0.1, 0.6], [0.2, -1.4]]

Example of 3-D data:

dataset = [[0.6, 0.1, 0.4], [0.2, 0.1, 0.9]]

Clustering by K-Means algorithm

K-Means uses two general parameters for clustering: input data and initial cluster centers.

```
from pyclustering.cluster.kmeans import kmeans
from pyclustering.cluster.center_initializer import kmeans_plusplus_initializer;
from pyclustering.utils import read_sample;

data = [ [0.1], [0.2], [0.5], [0.3], [1.5], [1.8], [1.3], [1.6], [1.5] ]
initial_centers = kmeans_plusplus_initializer(data, 2).initialize()
instance = kmeans(data, initial_centers)
instance.process()      # perform processing

clusters = instance.get_clusters();   # allocated clusters
centers = instance.get_centers();    # cluster centers
```

Output result of K-Means

K-Means algorithm returns allocated clusters and corresponding centers:

- Clusters are represented by list of clusters and each cluster contains object indexes from dataset, for example:

  [[0, 1, 4], [2, 3, 5]], where 0, 1, 4 – object indexes that forms the first cluster, and 2, 3, 5 – forms the second.

- Centers are represented by list of centers where each center is a point represented by coordinates.

Library core usage

By default core of pyclustering library that is called „ccore“ used for processing (C/C++ implementation of the algorithm). If platform is not supported then python implementation is used. There is special parameter that can be used to switch on/off core usage:

```
# switch on core (switch on by default, and it is ignored if impossible to use core
instance = kmeans(data, initial_centers, ccore=True)

# switch off core (python implementation is used instead)
instance = kmeans(data, initial_centers, ccore=False)
```

K-Means result visualization

Output result can be shown by common visualizer „**cluster_visualizer**“ from „**pyclustering.cluster**“ or by K-Means visualizer:

```
instance = kmeans(sample, start_centers, 0.0001)

instance.process()
clusters = instance.get_clusters()
centers = instance.get_centers()

# visualize clustering results
kmeans_visualizer.show_clusters(sample, clusters, centers, start_centers);
```

Stop condition – tolerance

K-Means calculates center changes on current and previous iterations and if difference is less than threshold value defined by parameter „tolerance“ than clustering process is over.

Example with small data points:

```
data = [  [0.0000001], [0.00000023], [0.00000051], [0.0000002] ]
tolerance = 0.000000001
instance = kmeans(data, initial_centers, tolerance)
```

But in case of very big or very small data values normalization should be used.

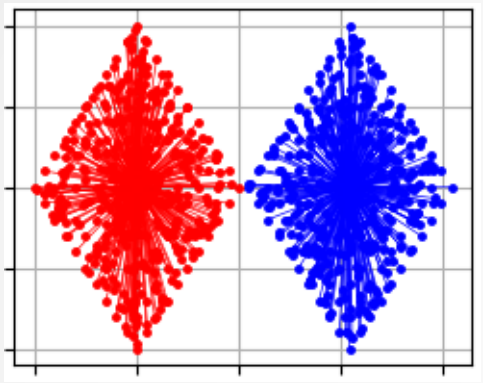
K-Means visualizer

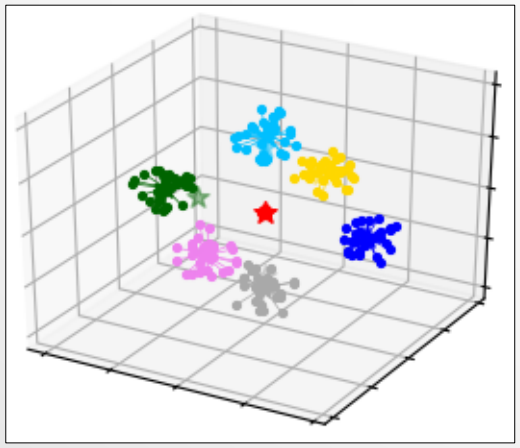
Output result can be shown by common visualizer „**cluster_visualizer**“ from „**pyclustering.cluster**“ or by K-Means visualizer:

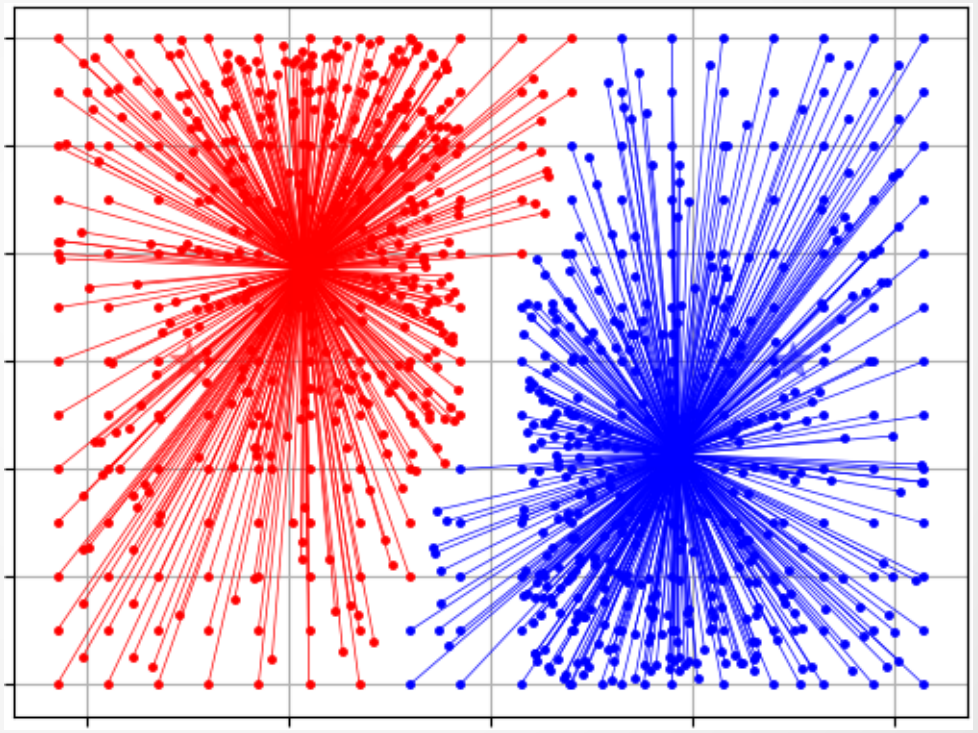
```
instance = kmeans(sample, start_centers, 0.0001)

instance.process()
clusters = instance.get_clusters()
centers = instance.get_centers()

# visualize clustering results
kmeans_visualizer.show_clusters(sample, clusters, centers, start_centers);
```

Example of result visualization

Here several examples of K-Means visualization:







K-Means result animation

K-Means clustering process can be visualized as an animation and saved to file. „kmeans_observer“ is used to collect information of each step of processing and the observer is used by „kmeans_visualizer“ then.

```
observer = kmeans_visualizer()
instance = kmeans(data, start_centers, observer=observer)
instance.process()

# animate clustering process
kmeans_visualizer.animate_cluster_allocation(sample, observer);

# or save animation to file
kmeans_visualizer.animate_cluster_allocation(data, observer, save_movie=file.mp4);
```

K-Means for segmentation

K-Means can consider image as a data where each point represent pixel with three coordinate (RGB), or with four coordinate (RGBA). Here is an example of image segmentation using K-Means algorithm:

```
from pyclustering.utils import draw_image_mask_segments, read_image
from pyclustering.cluster.kmeans import kmeans;

data = read_image(file_path_to_image);
kmeans_instance = kmeans(data, start_centers);
kmeans_instance.process();

clusters = kmeans_instance.get_clusters();
draw_image_mask_segments(source, clusters);
```

Image segmentation results

Here several examples of image segmentation by K-Means algorithm using **pyclustering**.

References and Links

- Official pyclustering github repository:
  https://github.com/annoviko/pyclustering
- Official pypi pyclustering page:
  https://pypi.python.org/pypi/pyclustering/0.8.0
- Official pyclustering web-site:
  https://pyclustering.github.io/

Thank you for your attention

Thank you!

PY Clustering