

File Edit View Run Kernel Git Tabs Settings Help

ML0101EN-Clus-DBSCN-wr

+ X □ ▶ ⟲ ⟳ Markdown git Run as Pipeline Python



Density-Based Clustering

Estimated time needed: 25 minutes

Objectives

After completing this lab you will be able to:

- Use DBSCAN to do Density based clustering
- Use Matplotlib to plot clusters

Most of the traditional clustering techniques, such as k-means, hierarchical and fuzzy clustering, can be used to group data without supervision. However, when applied to tasks with arbitrary shape clusters, or clusters within cluster, the traditional techniques might be unable to achieve good results. That is, elements in the same cluster might not share enough similarity or the performance may be poor. Additionally, Density-based Clustering locates regions of high density that are separated from one another by regions of low density. Density, in this context, is defined as the number of points within a specified radius.

In this section, the main focus will be manipulating the data and properties of DBSCAN and observing the resulting clustering.

Import the following libraries:

- numpy as np
- DBSCAN from sklearn.cluster
- make_blobs from sklearn.datasets.samples_generator
- StandardScaler from sklearn.preprocessing
- matplotlib.pyplot as plt

Remember %matplotlib inline to display plots

```
[ ]: # Notice: For visualization of map, you need basemap package.
# if you dont have basemap install on your machine, you can use the following Line to install it
!conda install -c conda-forge basemap matplotlib==3.1 -y
# Notice: you might have to refresh your page and re-run the notebook after installation
```

```
[ ]: import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

Data generation

The function below will generate the data points and requires these inputs:

- centroidLocation: Coordinates of the centroids that will generate the random data.
 - Example: input: [[4,3], [2,-1], [-1,4]]
- numSamples: The number of data points we want generated, split over the number of centroids (# of centroids defined in centroidLocation)
 - Example: 1500
- clusterDeviation: The standard deviation between the clusters. The larger the number, the further the spacing.
 - Example: 0.5

```
[ ]: def createDataPoints(centroidLocation, numSamples, clusterDeviation):
    # Create random data and store in feature matrix X and response vector y.
    X, y = make_blobs(n_samples=numSamples, centers=centroidLocation,
                      cluster_std=clusterDeviation)

    # Standardize features by removing the mean and scaling to unit variance
    X = StandardScaler().fit_transform(X)
    return X, y
```

Use createDataPoints with the 3 inputs and store the output into variables X and y.

```
[ ]: X, y = createDataPoints([[4,3], [2,-1], [-1,4]], 1500, 0.5)
```

Modeling

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. This technique is one of the most common clustering algorithms which works based on density of object. The whole idea is that if a particular point belongs to a cluster, it should be near to lots of other points in that cluster.

It works based on two parameters: Epsilon and Minimum Points

Epsilon determine a specified radius that if includes enough number of points within, we call it dense area

minimumSamples determine the minimum number of data points we want in a neighborhood to define a cluster.

```
[ ]: epsilon = 0.3
minimumSamples = 7
db = DBSCAN(eps=epsilon, min_samples=minimumSamples).fit(X)
labels = db.labels_
labels
```

Distinguish outliers

Lets Replace all elements with 'True' in core_samples_mask that are in the cluster, 'False' if the points are outliers.

```
[ ]: # First, create an array of booleans using the labels from db.
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
```

```

core_samples_mask[db.core_sample_indices_] = True
core_samples_mask

[ ]: # Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_clusters_

[ ]: # Remove repetition in labels by turning it into a set.
unique_labels = set(labels)
unique_labels

```

Data visualization

```

[ ]: # Create colors for the clusters.
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))

[ ]: # Plot the points with colors
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = 'k'

    class_member_mask = (labels == k)

    # Plot the datapoints that are clustered
    xy = X[class_member_mask & core_samples_mask]
    plt.scatter(xy[:, 0], xy[:, 1], s=50, c=[col], marker=u'o', alpha=0.5)

    # Plot the outliers
    xy = X[class_member_mask & ~core_samples_mask]
    plt.scatter(xy[:, 0], xy[:, 1], s=50, c=[col], marker=u'o', alpha=0.5)

```

Practice

To better understand differences between partitional and density-based clustering, try to cluster the above dataset into 3 clusters using k-Means.
Notice: do not generate data again, use the same dataset as above.

```
[ ]: # write your code here
```

▼ Click here for the solution

```

from sklearn.cluster import KMeans
k = 3
k_means3 = KMeans(init = "k-means++", n_clusters = k, n_init = 12)
k_means3.fit(X)
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot(1, 1, 1)
for k, col in zip(range(k), colors):
    my_members = (k_means3.labels_ == k)
    plt.scatter(X[my_members, 0], X[my_members, 1], c=col, marker=u'o', alpha=0.5)
plt.show()

```

Weather Station Clustering using DBSCAN & scikit-learn

DBSCAN is specially very good for tasks like class identification on a spatial context. The wonderful attribute of DBSCAN algorithm is that it can find out any arbitrary shape cluster without getting affected by noise. For example, this following example cluster the location of weather stations in Canada. <Click 1> DBSCAN can be used here, for instance, to find the group of stations which show the same weather condition. As you can see, it not only finds different arbitrary shaped clusters, can find the denser part of data-centered samples by ignoring less-dense areas or noises.

let's start playing with the data. We will be working according to the following workflow:

1. Loading data
 - Overview data
 - Data cleaning
 - Data selection
 - Clustering

Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. [Learn more here](#).

About the dataset

Environment Canada Monthly Values for July - 2015

Name in the table	Meaning
Stn_Name	Station Name
Lat	Latitude (North+, degrees)
Long	Longitude (West -, degrees)
Prov	Province
Tm	Mean Temperature (°C)
DwTm	Days without Valid Mean Temperature
D	Mean Temperature difference from Normal (1981-2010) (°C)
Tx	Highest Monthly Maximum Temperature (°C)
DwTx	Days without Valid Maximum Temperature
Tn	Lowest Monthly Minimum Temperature (°C)
DwTn	Days without Valid Minimum Temperature
S	Snowfall (cm)
DwS	Days without Valid Snowfall
S%N	Percent of Normal (1981-2010) Snowfall
P	Total Precipitation (mm)
DwP	Days without Valid Precipitation
P%N	Percent of Normal (1981-2010) Precipitation

S_G	Snow on the ground at the end of the month (cm)
Pd	Number of days with Precipitation 1.0 mm or more
BS	Bright Sunshine (hours)
DwBS	Days without Valid Bright Sunshine
BS%	Percent of Normal (1981-2010) Bright Sunshine
HDD	Degree Days below 18 °C
CDD	Degree Days above 18 °C
Stn_No	Climate station identifier (first 3 digits indicate drainage basin, last 4 characters are for sorting alphabetically).
NA	Not Available

1-Download data

To download the data, we will use `!wget`. To download the data, we will use `!wget` to download it from IBM Object Storage.

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: [Sign up now for free](#)

```
[ ]: !wget -O weather-stations20140101-20141231.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/weather-stations/20140101-20141231.csv
```

2- Load the dataset

We will import the .csv then we creates the columns for year, month and day.

```
[ ]: import_csv
import_pandas_as_pd
import_numpy_as_np

filename='weather-stations20140101-20141231.csv'

#Read_csv
pdf = pd.read_csv(filename)
pdf.head(5)
```

3-Cleaning

Lets remove rows that dont have any value in the `Tm` field.

```
[ ]: pdf = pdf[pd.notnull(pdf["Tm"])]
pdf = pdf.reset_index(drop=True)
pdf.head(5)
```

4-Visualization

Visualization of stations on map using basemap package. The matplotlib basemap toolkit is a library for plotting 2D data on maps in Python. Basemap does not do any plotting on its own, but provides the facilities to transform coordinates to a map projections.

Please notice that the size of each data points represents the average of maximum temperature for each station in a year.

```
[ ]: from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

llon=-140
ulon=-50
llat=40
ulat=65

pdf = pdf[(pdf['Long'] > llon) & (pdf['Long'] < ulon) & (pdf['Lat'] > llat) &(pdf['Lat'] < ulat)]

my_map = Basemap(projection='merc',
                  resolution = 'l', area_thresh = 1000.0,
                  llcrnrlon=llon, llcrnrlat=llat, #min longitude,(llcrnrlon).and.latitude,(llcrnrlat)
                  urcrnrlon=ulon, urcrnrlat=ulat) #max longitude,(urcrnrlon).and.latitude,(urcrnrlat)

my_map.drawcoastlines()
my_map.drawcountries()
# my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To collect data based on stations.....
xs,ys = my_map(np.asarray(pdf.Long),np.asarray(pdf.Lat))
pdf['xm']=xs.tolist()
pdf['ym']=ys.tolist()

#Visualization!
for index, row in pdf.iterrows():
    x,y = my_map(row.Long, row.Lat)
    my_map.plot(x,y,markerfacecolor = ([1,0,0]), marker='o', markersize= 5, alpha = 0.75)
    plt.text(x,y,stn)
plt.show()
```

5- Clustering of stations based on their location i.e. Lat & Lon

DBSCAN form sklearn library can runs DBSCAN clustering from vector array or distance matrix. In our case, we pass it the Numpy array Clus_dataSet to find core samples of high density and expands clusters from them.

```
[ ]: from sklearn.cluster import DBSCAN
import_sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN
db = DBSCAN(eps=0.15, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"] = labels
```

```

realClusterNum=len(set(labels))-(1 if -1 in labels else 0)
clusterNum = len(set(labels))

# A sample of clusters
pdf[["Stn_Name","Tx","Tm","Clus_Db"]].head(5)

```

As you can see for outliers, the cluster label is -1

```
[ :]: set(labels)
```

6- Visualization of clusters based on location

Now, we can visualize the clusters using basemap:

```

[ :]: from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
                  resolution='l', area_thresh = 1000.0,
                  llcrnrlon=lllon, llcrnrlat=lllat, #min_longitude_(llcrnrlon) and latitude_(llcrnrlat)
                  urcrnrlon=ulon, urcrnrlat=ulat) #max_longitude_(urcrnrlon) and latitude_(urcrnrlat)

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))

#Visualization1
for clust_number in set(labels):
    c=((0.4,0.4,0.4)) if clust_number == -1 else colors[np.int(clust_number)]
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color=c, marker='o', s= 20, alpha = 0.85)
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red')
        print ('Cluster '+str(clust_number)+'. Avg Temp: '+str(np.mean(clust_set.Tm)))

```

7- Clustering of stations based on their location, mean, max, and min Temperature

In this section we re-run DBSCAN, but this time on a 5-dimensional dataset:

```

[ :]: from sklearn.cluster import DBSCAN
import sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym','Tx','Tm','Tn']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"]=labels

realClusterNum=len(set(labels))-(1 if -1 in labels else 0)
clusterNum = len(set(labels))

# A sample of clusters
pdf[["Stn_Name","Tx","Tm","Clus_Db"]].head(5)

```

8- Visualization of clusters based on location and Temperature

```

[ :]: from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
                  resolution='l', area_thresh = 1000.0,
                  llcrnrlon=lllon, llcrnrlat=lllat, #min_longitude_(llcrnrlon) and latitude_(llcrnrlat)
                  urcrnrlon=ulon, urcrnrlat=ulat) #max_longitude_(urcrnrlon) and latitude_(urcrnrlat)

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))

#Visualization1
for clust_number in set(labels):
    c=((0.4,0.4,0.4)) if clust_number == -1 else colors[np.int(clust_number)]
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color=c, marker='o', s= 20, alpha = 0.85)
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red')
        print ('Cluster '+str(clust_number)+'. Avg Temp: '+str(np.mean(clust_set.Tm)))

```

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: [SPSS Modeler](#)

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter

notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at [Watson Studio](#)

Thank you for completing this lab!

Author

Saeed Aghabozorgi

Other Contributors

[Joseph Santarcangelo](#)

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-11-03	2.1	Lakshmi	Updated url of csv
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

© IBM Corporation 2020. All rights reserved.