**Density-Based Clustering**

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**

# 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==1.1.0 matplotlib==2.2.2 -y

# Notice: you maight 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

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.

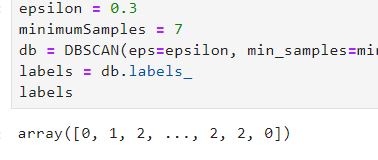
It works based on two parameters: Epsilon and Minimum Points  
**Epsilon** determine a specified radius that if includes enough number of points within, we call it dense area  
**minimumSamples** determine the minimum number of data points we want in a neighborhood to define a cluster.

db=DBSCAN(eps=0.3,min\_samples=7)

db.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 labels of db

core\_samples\_mask=np.zeros\_like(db.labels\_,dtype=bol)

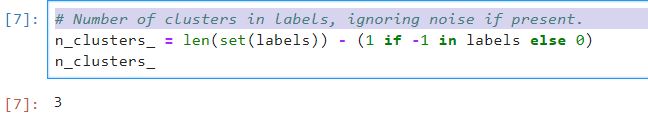
core\_samples\_mask[db.core\_samples\_indices]=True

core\_samples\_mask

# Number of clusters in labels, ignoring noise if present.

n\_clusters\_=len(set(labels))-(1 if -1 in labels else 0)

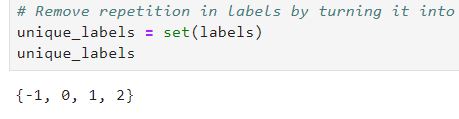
n\_clusters\_



#get unique labels

unique\_labels=len(set(labels))

unique\_labels



### Data visualization

color=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)

**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
* Clusteing

Download data

!wget -O weather-stations20140101-20141231.csv <https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/weather-stations20140101-20141231.csv>

import csv

import pandas as pd

import numpy as np

filename='weather-stations20140101-20141231.csv'

pdf=pd.read\_csv(filename)

pdf.head()



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()

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 it’s 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[‘Lat’]>llat & pdf[‘Lat’]<ulat & pdf[‘Long’]>llon & pdf[‘Long’]<ulon ]

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()

xs,ys=my\_map(np.asarray(pdf.Long), np.asarray(pdf.Lat))

pdf[‘xm’]=xs.toList()

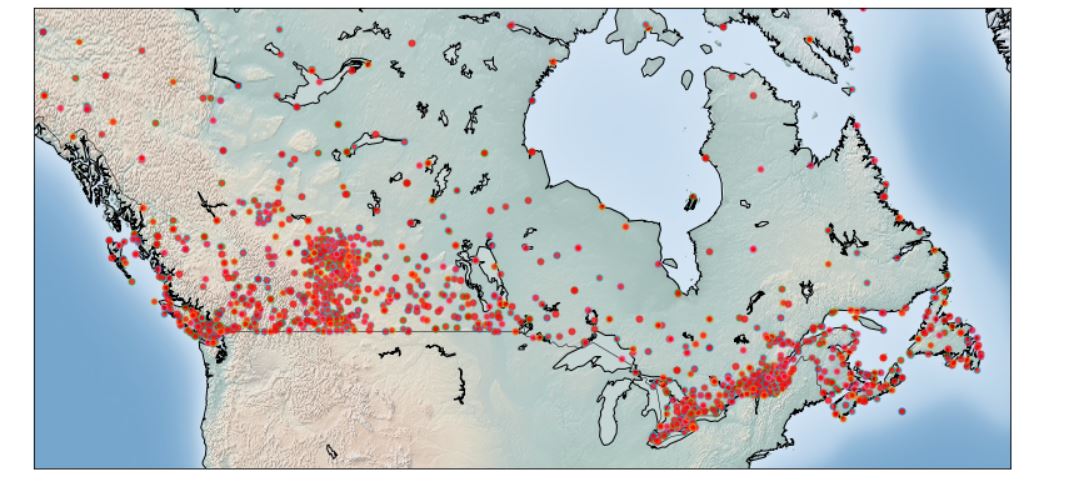
pdf[‘ym’]=ys.toList()

for index,row in pdf.iterrows():

# x,y = my\_map(row.Long, row.Lat)

my\_map.plot(row.xm, row.ym,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)

### 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=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 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)

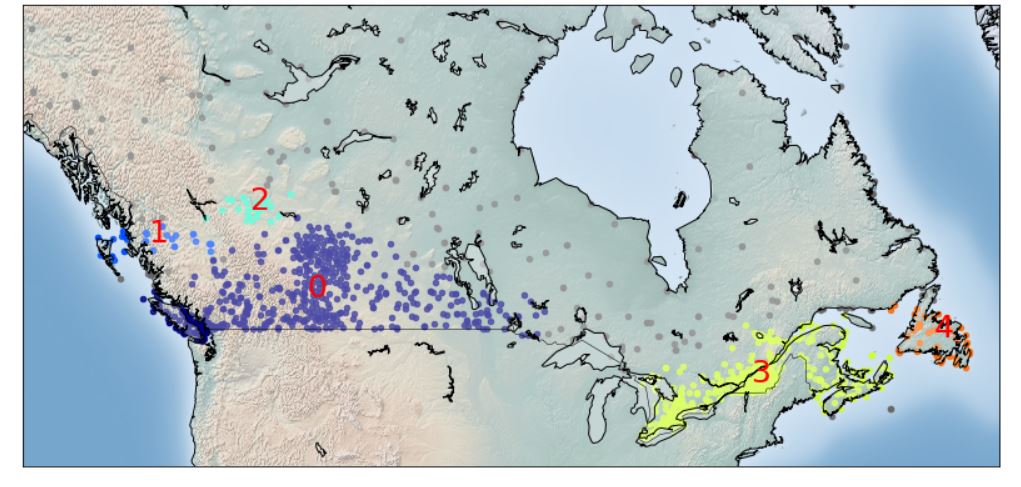
if clust\_number != -1:

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 Temperture

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=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 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)))

