ML0101EN-Clus-DBSCN-weather-py-v1

November 29, 2018

#

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:

```
<li> <b>numpy as np</b> </li>
<li> <b>DBSCAN</b> from <b>sklearn.cluster</b> </li>
<li> <b>make_blobs</b> from <b>sklearn.datasets.samples_generator</b> </li>
<li> <b>StandardScaler</b> from <b>sklearn.preprocessing</b> </li>
<li> <b>matplotlib.pyplot as plt</b> </li>
```

Remember %matplotlib inline to display plots

```
In [ ]: # 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 in
        # !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 installatio
```

```
In [1]: 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
```

```
In [2]: !conda install -c conda-forge  basemap==1.1.0  matplotlib==2.2.2  -y
```

```
Solving environment: done

## Package Plan ##
```

```
  environment location: /home/jupyterlab/conda

  added / updated specs:
    - basemap==1.1.0
    - matplotlib==2.2.2


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    openssl-1.0.2p             |       h470a237_1         3.1 MB  conda-forge
    mkl_fft-1.0.10             |          py36_0         650 KB  conda-forge
    certifi-2018.10.15         |       py36_1000         138 KB  conda-forge
    mkl_random-1.0.2           |          py36_0         1.3 MB  conda-forge
    mkl-2018.0.3               |               1       198.7 MB
    ca-certificates-2018.10.15 |       ha4d7672_0         135 KB  conda-forge
    numpy-1.15.0               |   py36h1b885b7_0          35 KB
    pyproj-1.9.5.1             |   py36h429999c_7         138 KB  conda-forge
    numpy-base-1.15.0          |   py36h3dfced4_0         4.2 MB
    conda-4.5.11               |       py36_1000         651 KB  conda-forge
    ------------------------------------------------------------
                                           Total:       209.0 MB

The following packages will be UPDATED:

    ca-certificates: 2018.8.24-ha4d7672_0  conda-forge --> 2018.10.15-ha4d7672_0  conda-forge
    certifi:         2018.8.24-py36_1001   conda-forge --> 2018.10.15-py36_1000   conda-forge
    conda:           4.5.11-py36_0         conda-forge --> 4.5.11-py36_1000       conda-forge
    mkl_fft:         1.0.1-py36h3010b51_0              --> 1.0.10-py36_0          conda-forge
    mkl_random:      1.0.1-py36h629b387_0              --> 1.0.2-py36_0           conda-forge
    numpy:           1.14.3-py36hcd700cb_1             --> 1.15.0-py36h1b885b7_0
    numpy-base:      1.14.3-py36h9be14a7_1             --> 1.15.0-py36h3dfced4_0
    openssl:         1.0.2p-h470a237_0     conda-forge --> 1.0.2p-h470a237_1      conda-forge
    pyproj:          1.9.5.1-py36_0        conda-forge --> 1.9.5.1-py36h429999c_7 conda-forge

The following packages will be DOWNGRADED:

    mkl:             2019.0-117            anaconda    --> 2018.0.3-1


Downloading and Extracting Packages
openssl-1.0.2p       | 3.1 MB    | ##################################### | 100%
mkl_fft-1.0.10       | 650 KB    | ##################################### | 100%
certifi-2018.10.15   | 138 KB    | ##################################### | 100%
mkl_random-1.0.2     | 1.3 MB    | ##################################### | 100%
mkl-2018.0.3         | 198.7 MB  | ##################################### | 100%
```

```
ca-certificates-2018 | 135 KB    | ##################################### | 100%
numpy-1.15.0         | 35 KB     | ##################################### | 100%
pyproj-1.9.5.1       | 138 KB    | ##################################### | 100%
numpy-base-1.15.0    | 4.2 MB    | ##################################### | 100%
conda-4.5.11         | 651 KB    | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

0.0.1 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 ce
  Example: 1500
 clusterDeviation: The standard deviation between the clusters. The larger the number
  Example: 0.5

```
In [3]: 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.

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

0.0.2 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.

```
In [5]: epsilon = 0.3
        minimumSamples = 7
        db = DBSCAN(eps=epsilon, min_samples=minimumSamples).fit(X)
```

```
        labels = db.labels_
        labels
```

```
Out[5]: array([0, 1, 0, ..., 1, 1, 2])
```

0.0.3 Distinguish outliers

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

```
In [6]: # 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
```

```
Out[6]: array([ True,  True,  True, ...,  True,  True,  True])
```

```
In [7]: # Number of clusters in labels, ignoring noise if present.
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
        n_clusters_
```

```
Out[7]: 3
```

```
In [8]: # Remove repetition in labels by turning it into a set.
        unique_labels = set(labels)
        unique_labels
```

```
Out[8]: {-1, 0, 1, 2}
```

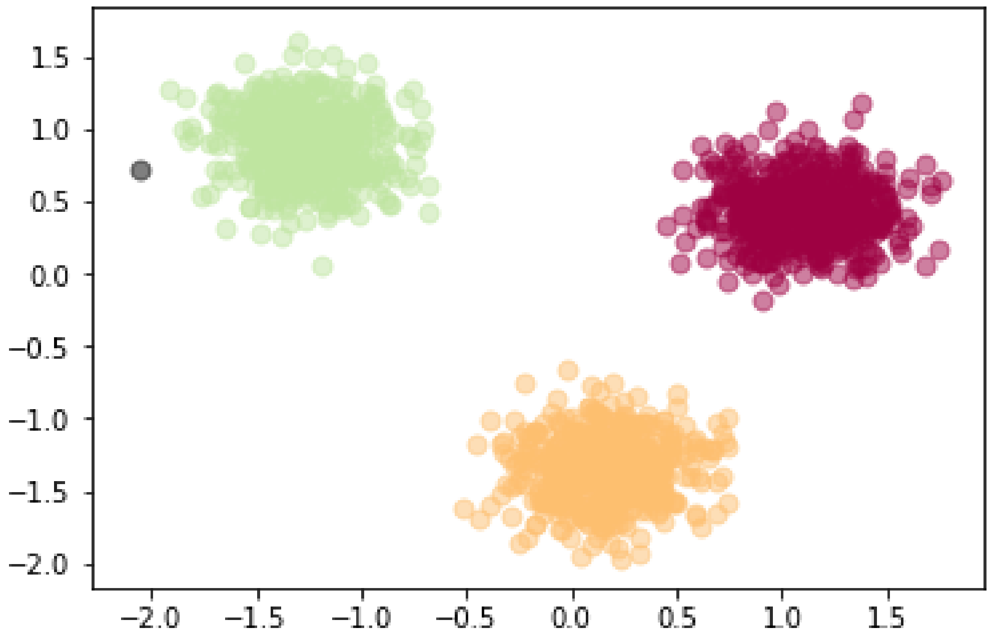
0.0.4 Data visualization

```
In [9]: # Create colors for the clusters.
        colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
        colors
```

```
Out[9]: array([[0.61960784, 0.00392157, 0.25882353, 1.        ],
               [0.99346405, 0.74771242, 0.43529412, 1.        ],
               [0.74771242, 0.89803922, 0.62745098, 1.        ],
               [0.36862745, 0.30980392, 0.63529412, 1.        ]])
```

```
In [10]: # 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]
```

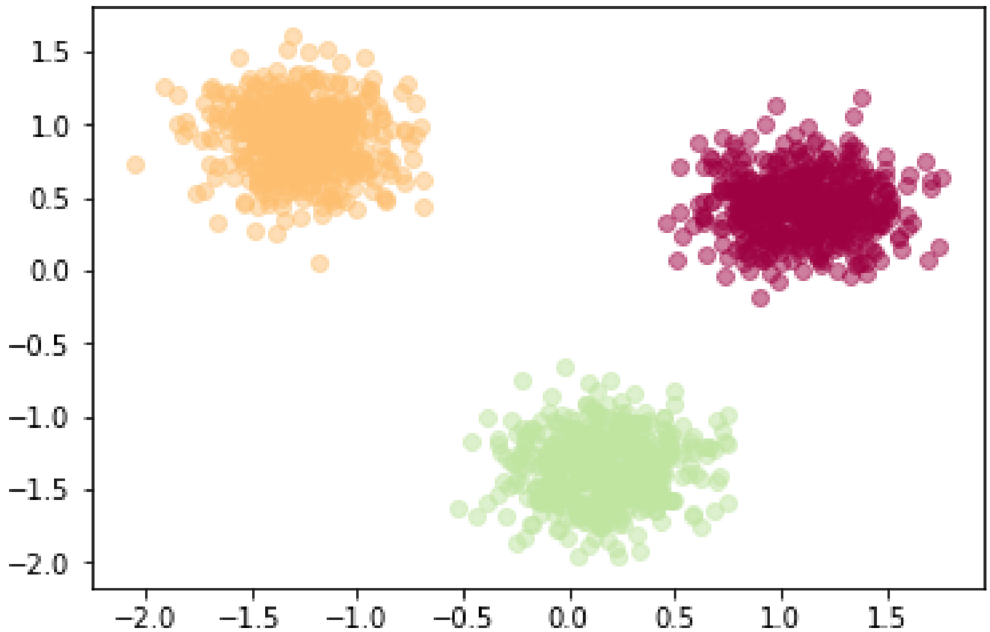
```
    plt.scatter(xy[:, 0], xy[:, 1],s=50, c=col, marker=u'o', alpha=0.5)

    # 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)
```



0.1 Practice

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

```
In [11]: # write your code here
         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()
```



Double-click **here** for the solution.

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. 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

0.1.1 About the dataset

Environment Canada
Monthly Values for July - 2015

```
<th>Name in the table</th>
<th>Meaning</th>

<td><font color = "green"><strong>Stn_Name</font></td>
<td><font color = "green"><strong>Station Name</font</td>

<td><font color = "green"><strong>Lat</font></td>
<td><font color = "green"><strong>Latitude (North+, degrees)</font></td>

<td><font color = "green"><strong>Long</font></td>
<td><font color = "green"><strong>Longitude (West - , degrees)</font></td>
```

```
<td>Prov</td>
<td>Province</td>

<td>Tm</td>
<td>Mean Temperature (řC)</td>

<td>DwTm</td>
<td>Days without Valid Mean Temperature</td>

<td>D</td>
<td>Mean Temperature difference from Normal (1981-2010) (řC)</td>

<td><font color = "black">Tx</font></td>
<td><font color = "black">Highest Monthly Maximum Temperature (řC)</font></td>

<td>DwTx</td>
<td>Days without Valid Maximum Temperature</td>

<td><font color = "black">Tn</font></td>
<td><font color = "black">Lowest Monthly Minimum Temperature (řC)</font></td>

<td>DwTn</td>
<td>Days without Valid Minimum Temperature</td>

<td>S</td>
<td>Snowfall (cm)</td>

<td>DwS</td>
<td>Days without Valid Snowfall</td>

<td>S%N</td>
<td>Percent of Normal (1981-2010) Snowfall</td>

<td><font color = "green"><strong>P</font></td>
<td><font color = "green"><strong>Total Precipitation (mm)</font></td>

<td>DwP</td>
<td>Days without Valid Precipitation</td>

<td>P%N</td>
<td>Percent of Normal (1981-2010) Precipitation</td>

<td>S_G</td>
<td>Snow on the ground at the end of the month (cm)</td>

<td>Pd</td>
<td>Number of days with Precipitation 1.0 mm or more</td>

<td>BS</td>
<td>Bright Sunshine (hours)</td>
```

```
<td>DwBS</td>
<td>Days without Valid Bright Sunshine</td>

<td>BS%</td>
<td>Percent of Normal (1981-2010) Bright Sunshine</td>

<td>HDD</td>
<td>Degree Days below 18 řC</td>

<td>CDD</td>
<td>Degree Days above 18 řC</td>

<td>Stn_No</td>
<td>Climate station identifier (first 3 digits indicate   drainage basin, last 4 characters are

<td>NA</td>
<td>Not Available</td>
```

0.1.2 1-Download data

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

```
In [12]: !wget -O weather-stations20140101-20141231.csv https://s3-api.us-geo.objectstorage.soft
         print('Download Complete!')
```

```
--2018-11-29 18:17:43--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cogni
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net).
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.n
HTTP request sent, awaiting response... 200 OK
Length: 129821 (127K) [text/csv]
Saving to: weather-stations20140101-20141231.csv

weather-stations201 100%[===================>] 126.78K  --.-KB/s   in 0.08s

2018-11-29 18:17:44 (1.46 MB/s) - weather-stations20140101-20141231.csv saved [129821/129821]

Download Complete!
```

0.1.3 2- Load the dataset

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

```
In [13]: import csv
         import pandas as pd
         import numpy as np
```

```
#filename='weather-stations20140101-20141231.csv'

#Read csv
pdf = pd.read_csv('weather-stations20140101-20141231.csv')
pdf.head(5)
```

```
Out[13]:                Stn_Name     Lat      Long Prov   Tm  DwTm    D    Tx  DwTx  \
0               CHEMAINUS  48.935 -123.742   BC  8.2   0.0  NaN  13.5   0.0
1   COWICHAN LAKE FORESTRY  48.824 -124.133   BC  7.0   0.0  3.0  15.0   0.0
2            LAKE COWICHAN  48.829 -124.052   BC  6.8  13.0  2.8  16.0   9.0
3         DISCOVERY ISLAND  48.425 -123.226   BC  NaN   NaN  NaN  12.5   0.0
4      DUNCAN KELVIN CREEK  48.735 -123.728   BC  7.7   2.0  3.4  14.5   2.0

     Tn   ...      DwP    P%N  S_G    Pd  BS  DwBS  BS%    HDD  CDD   Stn_No
0   1.0   ...      0.0    NaN  0.0  12.0 NaN   NaN  NaN  273.3  0.0  1011500
1  -3.0   ...      0.0  104.0  0.0  12.0 NaN   NaN  NaN  307.0  0.0  1012040
2  -2.5   ...      9.0    NaN  NaN  11.0 NaN   NaN  NaN  168.1  0.0  1012055
3   NaN   ...      NaN    NaN  NaN   NaN NaN   NaN  NaN    NaN  NaN  1012475
4  -1.0   ...      2.0    NaN  NaN  11.0 NaN   NaN  NaN  267.7  0.0  1012573

[5 rows x 25 columns]
```

0.1.4 3-Cleaning

Lets remove rows that don't have any value in the **Tm** field.

```
In [14]: pdf = pdf[pd.notnull(pdf["Tm"])]
         pdf = pdf.reset_index(drop=True)
         pdf.head(5)
```

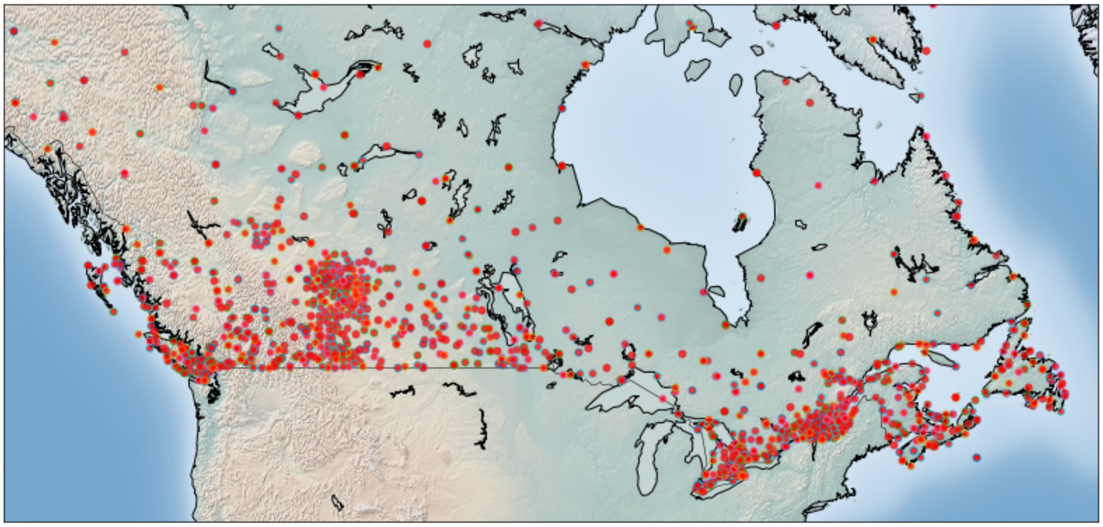
```
Out[14]:                Stn_Name     Lat      Long Prov   Tm  DwTm    D    Tx  DwTx  \
0               CHEMAINUS  48.935 -123.742   BC  8.2   0.0  NaN  13.5   0.0
1   COWICHAN LAKE FORESTRY  48.824 -124.133   BC  7.0   0.0  3.0  15.0   0.0
2            LAKE COWICHAN  48.829 -124.052   BC  6.8  13.0  2.8  16.0   9.0
3      DUNCAN KELVIN CREEK  48.735 -123.728   BC  7.7   2.0  3.4  14.5   2.0
4        ESQUIMALT HARBOUR  48.432 -123.439   BC  8.8   0.0  NaN  13.1   0.0

     Tn   ...      DwP    P%N  S_G    Pd  BS  DwBS  BS%    HDD  CDD   Stn_No
0   1.0   ...      0.0    NaN  0.0  12.0 NaN   NaN  NaN  273.3  0.0  1011500
1  -3.0   ...      0.0  104.0  0.0  12.0 NaN   NaN  NaN  307.0  0.0  1012040
2  -2.5   ...      9.0    NaN  NaN  11.0 NaN   NaN  NaN  168.1  0.0  1012055
3  -1.0   ...      2.0    NaN  NaN  11.0 NaN   NaN  NaN  267.7  0.0  1012573
4   1.9   ...      8.0    NaN  NaN  12.0 NaN   NaN  NaN  258.6  0.0  1012710

[5 rows x 25 columns]
```

0.1.5 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 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.

```
In [15]: 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'

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

         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()

         #Visualization1
         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, a
         #plt.text(x,y,stn)
         plt.show()
```



0.1.6 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.

In [16]:
```
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)
```

Out[16]:

	Stn_Name	Tx	Tm	Clus_Db
0	CHEMAINUS	13.5	8.2	0

```
1  COWICHAN LAKE FORESTRY  15.0  7.0        0
2           LAKE COWICHAN  16.0  6.8        0
3      DUNCAN KELVIN CREEK  14.5  7.7        0
4        ESQUIMALT HARBOUR  13.1  8.8        0
```

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

```
In [17]: set(labels)
```

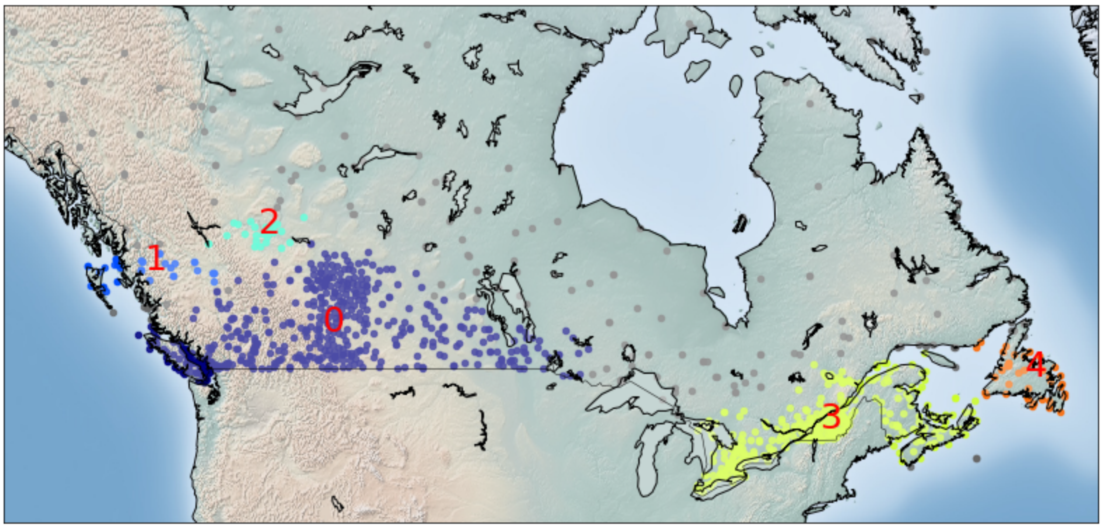
```
Out[17]: {-1, 0, 1, 2, 3, 4}
```

0.1.7 6- Visualization of clusters based on location

Now, we can visualize the clusters using basemap:

```
In [19]: 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 (ll
                     urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (ur

         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.
             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)))
```

```
Cluster 0, Avg Temp: -5.538747553816051
Cluster 1, Avg Temp: 1.9526315789473685
Cluster 2, Avg Temp: -9.195652173913045
Cluster 3, Avg Temp: -15.300833333333333
Cluster 4, Avg Temp: -7.769047619047619
```



0.1.8 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:

```
In [20]: 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)
```

Out[20]:

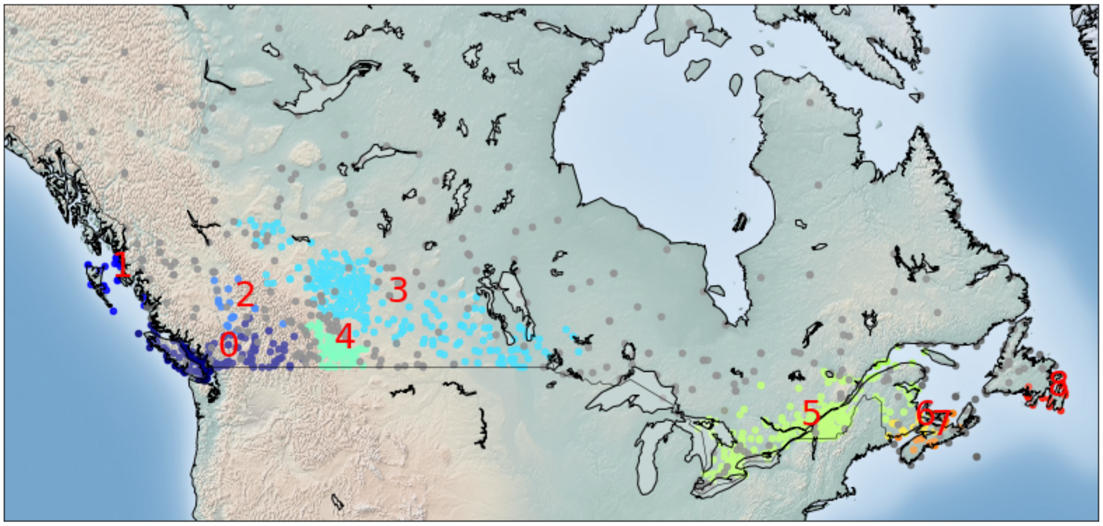
	Stn_Name	Tx	Tm	Clus_Db
0	CHEMAINUS	13.5	8.2	0
1	COWICHAN LAKE FORESTRY	15.0	7.0	0
2	LAKE COWICHAN	16.0	6.8	0
3	DUNCAN KELVIN CREEK	14.5	7.7	0
4	ESQUIMALT HARBOUR	13.1	8.8	0

0.1.9 8- Visualization of clusters based on location and Temperture

In [21]:
```
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 (ll
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (ur

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.
    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)))
```

```
Cluster 0, Avg Temp: 6.221920529801334
Cluster 1, Avg Temp: 6.790000000000001
```

```
Cluster 2, Avg Temp: -0.49411764705882355
Cluster 3, Avg Temp: -13.877209302325586
Cluster 4, Avg Temp: -4.186274509803922
Cluster 5, Avg Temp: -16.301503759398482
Cluster 6, Avg Temp: -13.599999999999998
Cluster 7, Avg Temp: -9.753333333333334
Cluster 8, Avg Temp: -4.258333333333334
```



0.2 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

0.2.1 Thanks for completing this lesson!

Notebook created by: Saeed Aghabozorgi