Name: Tan Ngiap Chuan Alvin

AML Practicum 1: K-means Algorithm

Abstract

To implement k-means algorithm to learn the clustering partitioning method from the training data. To apply k-means group dots representing lightning strokes. To familiarize with determining the parameter for k-means.

Using M=[0,0], we computed the error as 86.62495736289799.
Using first 5 XY-coords of X as centroids (K=5), the error is reduced to 9.73227985727934
After running calcMeans() and Group() (K=5), the error is reduced to 4.006100077883563
After running K=50, the error is reduced to 0.9216408380317802
After running K=100, the error is reduced to 0.5725156609427651

Given a choice of either K=50 and K=100, we would prefer the K=50 over K=100 as the difference between the 0.92 error of K=50 and 0.57 error of K=100 is less than 0.50. In fact, both error yield an error of less than 1.0. Coupled with the visual challenge of 'over-clustering' on a plot, we would prefer the K=50 over K=100.

Introduction

Lightning networks detect individual lightning discharge events all over the world. Understanding the lightning activities over every unique geography location over space and time will help us to better manage any economic damages and human safety that could arise from lightning activities.

For each lightning event, the data is stored as (time, amount of discharge, latitude, and longitude). It shows that when and where, there is a lightening stroke with certain amount of discharge.

We will implement a k-means clustering algorithm to help the meteorologists to group the lightning strokes basing on the position.

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

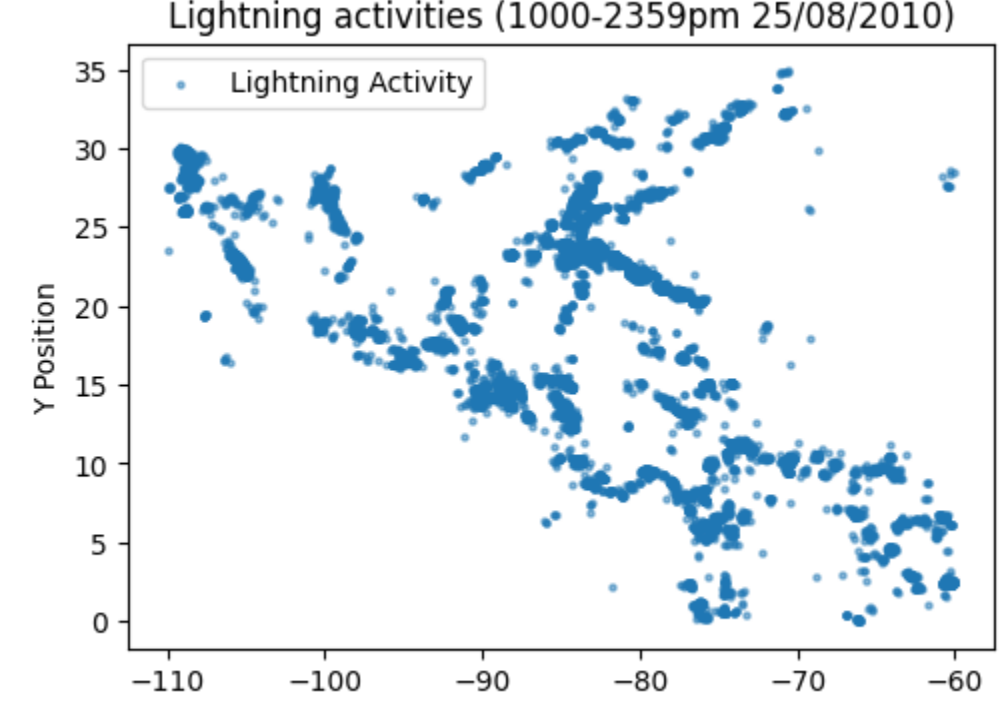
A cluster refers to a collection of data points aggregated together because of certain similarities. We define a target number k, which refers to the number of centroids needed in the dataset. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster euclidean distance. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.

In [20]:
```
# import the libraries for programming
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import csv
```

In [2]:
```
def loadData(name):
    '''
    Objective: To load the file and convert to numpy array
    Input: filename (string)
    Output: X dataset (array)
    '''
    df = pd.read_csv(name, delimiter = "\t", header = None)
    return df.to_numpy()
```

In [3]:
```
file_name = '2010825.txt'                                   # get the file name
col_3 = np.zeros(16259).reshape(16259, 1)                   # create all zeros in col_3
X = np.concatenate((loadData(file_name), col_3), axis=1) # concatenate col_3 to X hor.
```

In [4]:
```
fig=plt.figure(dpi=100)
plt.scatter(X.T[0], X.T[1], alpha=0.5, label='Lightning Activity', s=5) # use X and Y
plt.legend()   # s=area, c=colors, alpha=0.5
plt.title('Lightning activities (1000-2359pm 25/08/2010)')
plt.xlabel('X Position')
plt.ylabel('Y Position')
plt.gca().set_aspect('equal')
fig.savefig("Lightning Strokes on 25 Aug 2010", format="svg")
```



In [5]:
```
def errorCompute(X, M):
    '''
    Objective: To compute the error as L2 norm
    Input: X dataset of 2 features + 1 ClusterID, mean value (M) for each cluster
           (both are arrays)
    Output: The value of objective function for the clustering (float)
    '''
    l2 = np.linalg.norm(X[:, :2] - M[X[:, 2].astype(int)], axis=1)
    return np.mean(l2)
```

In [6]:
```
def Group(X, M):
    '''
    Objective: Assign each object into a cluster bases on the current set of means (M)
    Input: X dataset of 2 features + 1 ClusterID, current mean (M)
           (both are arrays)
    Output: X with updated clusterID. Assign each object into its closest cluster
    '''
    for idx, i in enumerate(X[:,0:2]):                  # iterate over 10 rows of X, co.
        l2_norm = np.sqrt(np.square(i[0] - M[:,0]) + np.square(i[1] - M[:,1])) # get
        X[idx, 2] = np.argmin(l2_norm)                                         # popu.
    return X
```

In [7]:
```
def calcMeans(X, M):
    '''
    Objective: Update the means (M) until there is no changes in clustering result
    Input: X dataset of 2 features + 1 ClusterID, current mean (M)
           (both are arrays)
    Output: The updated mean value(M) for each cluster
    '''
    for count in range(100):
        X = Group(X, M)  # call the Group function
        centroid_XY = {i:[] for i in range(0,M.shape[0])} # create dict for xy_coords
        for i in X[:,0:2]:                   # iterate over 10 rows of X, col 1 and col
            l2_norm = np.sqrt(np.square(i[0] - M[:,0]) + np.square(i[1] - M[:,1])) #
            centroid_XY[np.argmin(l2_norm)].append(i)                              #
    #### ------------------------------------------------------------------ #####
        old = M                                              # assign current M centroid
        centroid_list = [np.array(i) for i in centroid_XY.values()] # extract xy coor
        M = np.array([0,0])             # dummy array for vstacking purpose
        for xy in centroid_list:        # for each pair of xy_coord in list
            x_pts = xy.T[0]             # get all x_coords of a centroid
            y_pts = xy.T[1]             # get all y_coords of a centroid
            row = np.array([np.mean(x_pts), np.mean(y_pts)]) # assign xy_mean pairs to
            M = np.vstack((M, row))     # create M matrix
        M = M[1:]                       # get rid of dummy array
    #### ------------------------------------------------------------------ #####
        print(f'Running {count+1} epochs in progress ...')
        if np.array_equal(old, M):  # if no change in centroids' XY coords, break loop
            print(f'Centroids unchanged after {count+1} epochs\n')
            print(f'M after {count+1} epoch:\n{M}\n')
            break
    return M, centroid_XY
```

TASKS - Part 3C: Clustering Objective Function

Given:

- n objects
- k represents number of clusters

K-mean objective function:

$d(o, centroid\&)$

Euclidean Metric in the objective function:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

TASKS - Part 3D: errCompute()

In [8]:
```
M = np.array([[0,0]]) # M.shape is (1,2)
print(f'Using M=[0,0] the error is {errorCompute(X, M)}')
```

```
Using M=[0,0] the error is 86.62495736289799
```

TASKS - Part 3E: Method Explanation

Working mechanism of the method of assigning each object to a cluster:

K-mean objective function will ...
- Groups n objects into k clusters by minimizing the E
- Find k centroids that minimize E
- Centroid is an actual object centrally located in a cluster

TASKS - Part 3F: Group()

In [9]:
```
M=np.copy(X[0:5,0:X.shape[1]-1])
X = Group(X, M)
error = errorCompute(X, M)
print(f'Using first 5 XY-coords of X as centroids, the error is {error}')
```

```
Using first 5 XY-coords of X as centroids, the error is 9.73227985727934
```

In [21]:
```
# save the inital means of 5 centroids for K=5 means clustering
with open('K5_initial_mean.csv', 'w') as f:
    write = csv.writer(f)
    write.writerows(val for val in M)
```

TASKS - Part 3G: Describe the method to complute new means

Steps to calculate the new means based on current clustering:

- Step1: Randomly select k number of centroids
- Step2: Assign each sample into the class represented by the closest centroids
- Step3：Update centroids as mean of the cluster based on the latest set of assigned data points
- Step4：Repeat step2 and step3 until convergence

TASKS - Part 3H: Run k-means with K=5

In [10]:
```
# print(M)
M, xy_5 = calcMeans(X, M)
```

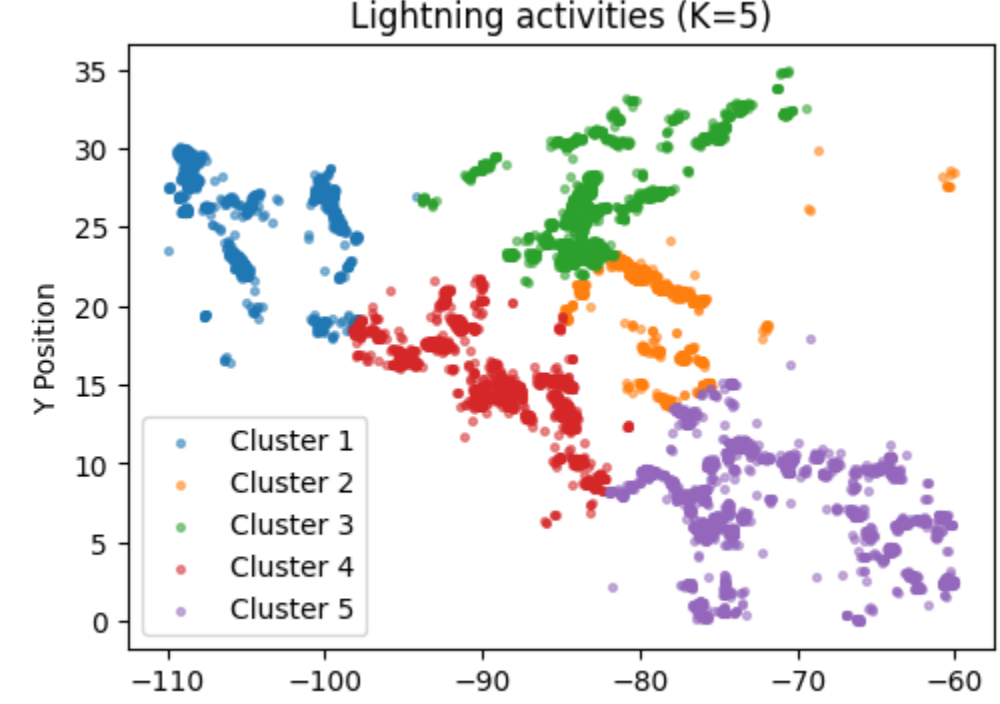
```
Running 1 epochs in progress ...
Running 2 epochs in progress ...
Running 3 epochs in progress ...
Running 4 epochs in progress ...
Running 5 epochs in progress ...
Running 6 epochs in progress ...
Running 7 epochs in progress ...
Running 8 epochs in progress ...
Running 9 epochs in progress ...
Running 10 epochs in progress ...
Running 11 epochs in progress ...
Running 12 epochs in progress ...
Running 13 epochs in progress ...
Running 14 epochs in progress ...
Running 15 epochs in progress ...
Running 16 epochs in progress ...
Centroids unchanged after 16 epochs

M after 16 epoch:
[[-105.19604652   25.33132344]
 [ -79.39863196   21.23281229]
 [ -82.67640192   26.06156976]
 [ -89.16882233   15.14357792]
 [ -72.13285047    8.21465361]]
```

In [11]:
```
error = errorCompute(X, M)
print(f'After running calcMeans() and Group(), the error is {error}')
```

```
After running calcMeans() and Group(), the error is 4.006100077883563
```

In [12]:
```
fig=plt.figure(dpi=100)
for key, value in xy_5.items(): # extract xy coords of each centroid into a list
    xx = np.array(value).T[0]   # get the x-coordinates
    yy = np.array(value).T[1]   # get the y-coordinates
    plt.scatter(xx, yy, alpha=0.5, label=f'Cluster {key+1}', s=7) # use X and Y pts to
plt.legend()   # s=area, c=colors, alpha=0.5
plt.title('Lightning activities (K=5)')
plt.xlabel('X Position')
plt.ylabel('Y Position')
plt.gca().set_aspect('equal')
```



TASKS - Part 3I: Run k-means with K=50

In [13]:
```
M_50_initial = np.copy(X[0:50,0:X.shape[1]-1])
X = Group(X, M_50_initial)
M_50_final, xy_50 = calcMeans(X, M_50_initial)
error_50 = errorCompute(X, M_50_final)
print(f'After running K=50, the error is {error_50}')
```

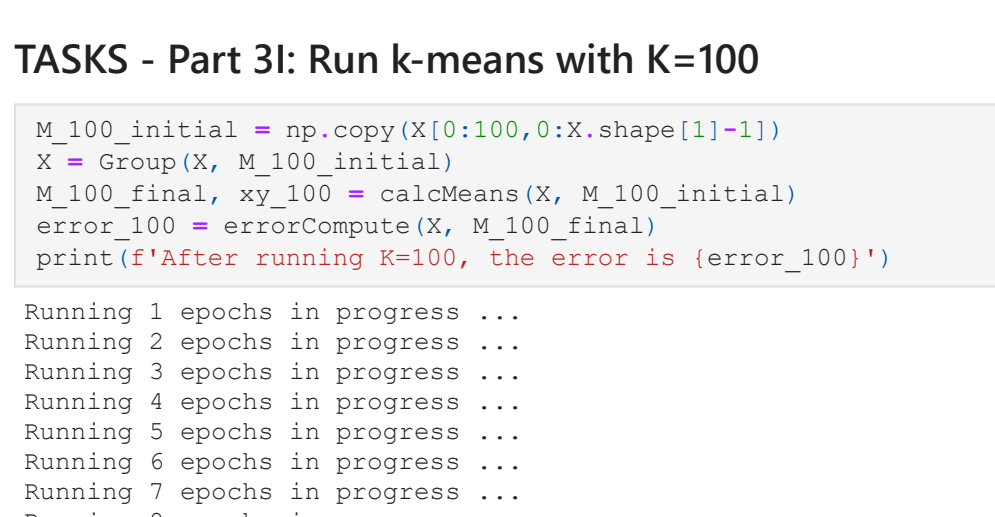
```
Running 1 epochs in progress ...
Running 2 epochs in progress ...
Running 3 epochs in progress ...
Running 4 epochs in progress ...
Running 5 epochs in progress ...
Running 6 epochs in progress ...
Running 7 epochs in progress ...
Running 8 epochs in progress ...
Running 9 epochs in progress ...
Running 10 epochs in progress ...
Running 11 epochs in progress ...
Running 12 epochs in progress ...
Running 13 epochs in progress ...
Running 14 epochs in progress ...
Running 15 epochs in progress ...
Running 16 epochs in progress ...
Running 17 epochs in progress ...
Running 18 epochs in progress ...
Running 19 epochs in progress ...
Running 20 epochs in progress ...
Running 21 epochs in progress ...
Running 22 epochs in progress ...
Running 23 epochs in progress ...
Running 24 epochs in progress ...
Running 25 epochs in progress ...
Running 26 epochs in progress ...
Running 27 epochs in progress ...
Running 28 epochs in progress ...
Running 29 epochs in progress ...
Running 30 epochs in progress ...
Running 31 epochs in progress ...
Running 32 epochs in progress ...
Running 33 epochs in progress ...
Running 34 epochs in progress ...
Running 35 epochs in progress ...
Running 36 epochs in progress ...
Running 37 epochs in progress ...
Running 38 epochs in progress ...
Running 39 epochs in progress ...
Running 40 epochs in progress ...
Running 41 epochs in progress ...
Running 42 epochs in progress ...
Running 43 epochs in progress ...
Running 44 epochs in progress ...
Running 45 epochs in progress ...
Running 46 epochs in progress ...
Running 47 epochs in progress ...
Running 48 epochs in progress ...
Running 49 epochs in progress ...
Running 50 epochs in progress ...
Running 51 epochs in progress ...
Centroids unchanged after 51 epochs

M after 51 epoch:
[[ -84.7389125    19.3018125 ]
 [ -77.39584816   20.7592168 ]
 [ -78.49712022   21.20796239]
 [ -76.591028     20.353078  ]
 [ -78.16112282   16.79331141]
 [ -76.19389595   20.26360811]
 [ -65.92858719    9.19971421]
 [ -87.6119435    23.1034785 ]
 [ -84.72164199   23.38912212]
 [ -79.73289043   21.99343625]
 [ -94.87245458   16.61237802]
 [ -71.648375     18.34376667]
 [ -82.79636174   30.94092957]
 [ -80.27879824   21.91315101]
 [ -98.951745     24.55810813]
 [-100.01265236   27.06484764]
 [ -83.18848297   27.63980694]
 [ -80.7654463    22.55715953]
 [ -83.70167223   26.69518742]
 [-108.68069672   27.91651434]
 [-108.80797555   29.57409095]
 [-107.49652703   26.18966712]
 [ -83.06901191   23.46813219]
 [ -84.80363812   14.97322875]
 [ -75.88477643    5.88295255]
 [ -84.57852866   12.63942287]
 [ -82.26415547    9.3395829 ]
 [ -98.33800618   18.47403527]
 [ -83.70395278   21.13268611]
 [ -76.77201377   13.85983912]
 [ -83.99314402   25.35000897]
 [ -84.53583333   16.57163333]
 [ -85.90992338   15.3249987 ]
 [ -90.23685527   13.90498776]
 [ -73.83000415   31.46666024]
 [ -92.26037544   18.36728697]
 [ -81.31540649   22.81809375]
 [ -81.56724892   23.03282312]
 [-105.38383355   22.49086018]
 [ -91.18610086   28.15762672]
 [ -62.68410276    4.14629632]
 [ -87.87937437   14.6774191 ]
 [ -88.46873966   13.94254852]
 [ -88.6248982    15.13816475]
 [ -87.09392069   12.99242529]
 [ -79.14428444   27.27676222]
 [ -90.42772747   15.00335495]
 [ -81.33970889   26.05422   ]
 [ -89.54610345   14.2103181 ]
 [ -73.2636527    10.69507914]]

After running K=50, the error is 0.9216408380317802
```

In [23]:
```
# save the inital means of 50 centroids for K=50 means clustering
with open('K50_initial_mean.csv', 'w') as f:
    write = csv.writer(f)
    write.writerows(val for val in M_50_initial)
```

In [14]:
```
fig=plt.figure(dpi=100)
for key, value in xy_50.items(): # extract xy coords of each centroid into a list
    xx = np.array(value).T[0]   # get the x-coordinates
    yy = np.array(value).T[1]   # get the y-coordinates
    plt.scatter(xx, yy, alpha=0.5, label=f'Cluster {key+1}', s=7) # use X and Y pts to
# plt.legend()
plt.title('Lightning activities (K=50)')
plt.xlabel('X Position')
plt.ylabel('Y Position')
plt.gca().set_aspect('equal')
```



TASKS - Part 3I: Run k-means with K=100

In [15]:
```
M_100_initial = np.copy(X[0:100,0:X.shape[1]-1])
X = Group(X, M_100_initial)
M_100_final, xy_100 = calcMeans(X, M_100_initial)
error_100 = errorCompute(X, M_100_final)
print(f'After running K=100, the error is {error_100}')
```

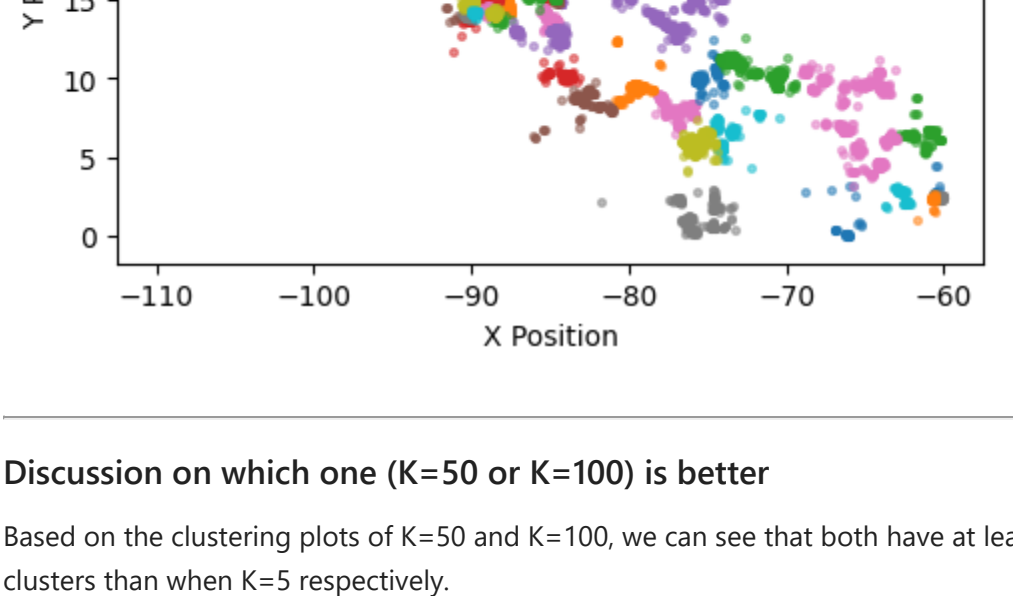
```
Running 1 epochs in progress ...
Running 2 epochs in progress ...
Running 3 epochs in progress ...
Running 4 epochs in progress ...
Running 5 epochs in progress ...
Running 6 epochs in progress ...
Running 7 epochs in progress ...
Running 8 epochs in progress ...
Running 9 epochs in progress ...
Running 10 epochs in progress ...
Running 11 epochs in progress ...
Running 12 epochs in progress ...
Running 13 epochs in progress ...
Running 14 epochs in progress ...
Running 15 epochs in progress ...
Running 16 epochs in progress ...
Running 17 epochs in progress ...
Running 18 epochs in progress ...
Running 19 epochs in progress ...
Running 20 epochs in progress ...
Running 21 epochs in progress ...
Running 22 epochs in progress ...
Running 23 epochs in progress ...
Running 24 epochs in progress ...
Running 25 epochs in progress ...
Running 26 epochs in progress ...
Running 27 epochs in progress ...
Running 28 epochs in progress ...
Running 29 epochs in progress ...
Running 30 epochs in progress ...
Running 31 epochs in progress ...
Running 32 epochs in progress ...
Running 33 epochs in progress ...
Running 34 epochs in progress ...
Running 35 epochs in progress ...
Running 36 epochs in progress ...
Running 37 epochs in progress ...
Centroids unchanged after 37 epochs

M after 37 epoch:
[[ -84.7824       19.22793137]
 [ -77.09416455   20.76595182]
 [ -77.79826203   20.88001646]
 [ -76.62870909   20.29045455]
 [ -79.49712877   16.85135616]
 [ -77.40846384   20.72964811]
 [ -65.82672062    9.68958247]
 [ -82.87471154   22.68534615]
 [ -83.13237292   23.74765885]
 [ -80.26940806   21.89584417]
 [ -91.78371769   20.51901308]
 [ -76.2814225    20.2239525 ]
 [ -83.13913651   28.1982619 ]
 [ -80.51401071   22.34227321]
 [ -98.951745     24.55810813]
 [ -99.99059487   27.06706374]
 [ -82.9626       28.01258475]
 [ -80.88477527   22.6460086 ]
 [ -83.61402727   30.66350779]
 [-108.60293562   28.10157811]
 [-108.82411735   29.60611862]
 [-108.75611299   26.17309266]
 [ -82.50318962   23.58181246]
 [ -84.9902415    15.27942585]
 [ -84.30415116   14.83946977]
 [ -84.49165714   14.92962381]
 [ -84.98047748   13.7499018 ]
 [-106.03123077   19.00438846]
 [ -83.75933095   20.69776905]
 [ -76.93098961   16.76588961]
 [ -83.34838313   25.58357407]
 [ -84.38392      16.66466   ]
 [ -85.92836622   15.33130541]
 [ -90.29078727   13.67934182]
 [ -73.65061173   31.44537653]
 [ -91.2511125    14.039975  ]
 [ -81.29274687   22.79164479]
 [ -81.5309303    23.14641061]
 [-104.98407692   26.48548333]
 [ -91.18610086   28.15762672]
 [ -60.53188824    2.98935882]
 [ -87.86505596   14.67954974]
 [ -88.23816354   13.74143646]
 [ -88.56222      15.04812863]
 [ -87.08997209   12.98383023]
 [ -79.17674352   27.14546389]
 [ -89.17797612   14.21884627]
 [ -81.14021688   26.05385455]
 [ -88.58022045   14.07827955]
 [ -73.78173933    6.57600787]
 [ -66.23902308    0.59644872]
 [ -79.84955707   22.12873463]
 [ -84.63940435   15.12164565]
 [-100.22938901   18.68901099]
 [ -97.38938441   18.36500376]
 [ -84.26560093   25.19275185]
 [ -77.07533854    8.04692682]
 [ -85.79746762   24.61570762]
 [ -83.32865364   23.18937351]
 [-105.3679016    22.59912221]
 [ -75.1426       10.13160047]
 [ -79.68473851    9.52591622]
 [ -72.82048238   10.86752368]
 [ -76.43272083   20.298975  ]
 [ -92.29981854   18.03407124]
 [ -79.67308459   21.89591199]
 [ -83.78371055   26.67748443]
 [ -60.1860386     2.45548947]
 [ -83.32127668   27.26270316]
 [ -62.73737241    2.68988966]
 [ -76.098        20.269905  ]
 [ -71.648375     18.34376667]
 [ -61.19602456    6.46248158]
 [ -75.96316      20.40405   ]
 [ -76.76167278   13.88464833]
 [ -89.98474474   16.16358684]
 [ -64.67498095    5.59639683]
 [ -90.20600778   13.96640359]
 [ -90.21244177   14.78664051]
 [ -89.78132609   14.16838406]
 [ -81.44402488   22.91288259]
 [ -60.72389024    2.33664146]
 [ -83.69165882   21.48698529]
 [ -84.32816061   23.3868697 ]
 [ -83.42945142   23.56344009]
 [ -78.40582451   21.19611275]
 [ -78.73220276   21.26320552]
 [ -80.34091538   31.46343365]
 [ -78.0656       24.2044     ]
 [ -94.61052083   16.57180179]
 [ -95.27441165   16.64937379]
 [ -84.75228273   23.79173364]
 [ -84.73166723   22.70806555]
 [ -84.07879382   10.21151685]
 [ -84.46798917   12.60675792]
 [ -82.79934877    8.58659015]
 [ -81.78840645   22.9781129 ]
 [ -75.63513411    1.36864112]
 [ -75.48174516    6.04387788]
 [ -87.4002486    23.23371564]]

After running K=100, the error is 0.5725156609427651
```

In [24]:
```
# save the inital means of 100 centroids for K=100 means clustering
with open('K100_initial_mean.csv', 'w') as f:
    write = csv.writer(f)
    write.writerows(val for val in M_100_initial)
```

In [16]:
```
fig=plt.figure(dpi=100)
for key, value in xy_100.items(): # extract xy coords of each centroid into a list
    xx = np.array(value).T[0]   # get the x-coordinates
    yy = np.array(value).T[1]   # get the y-coordinates
    plt.scatter(xx, yy, alpha=0.5, label=f'Cluster {key+1}', s=7) # use x & y pts to p
# plt.legend()
plt.title('Lightning activities (K=100)')
plt.xlabel('X Position')
plt.ylabel('Y Position')
plt.gca().set_aspect('equal')
```



Discussion on which one (K=50 or K=100) is better

Based on the clustering plots of K=50 and K=100, we can see that both have at least 10 and 20 times more clusters than when K=5 respectively.

However, given a choice of either K=50 and K=100, we would prefer the K=50 over K=100 as the difference between the 0.92 error of K=50 and 0.57 error of K=100 is less than 0.50. In fact, both error yield an error of less than 1.0. Coupled with the visual challenge of 'over-clustering' on a plot, we would prefer the K=50 over K=100.

Clustering is useful and effective if we can find a right balance of error and visual-challenge of 'over-clustering', and given a choice, we would prefer K=5 for simple and easy labelling of lightning clusters.