SCRIPT

September 29, 2020

1 Stage 1: Preparing Inputs

[ ]:
```
#import libraries
import pandas as pd
import numpy as np
from numpy import percentile
from numpy import unique
from numpy import where
import matplotlib as mpl
from matplotlib import pyplot
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt
import seaborn as sns
import seaborn as sns; sns.set(font_scale=1.2)
from sklearn.ensemble import IsolationForest
from sklearn.cluster import MiniBatchKMeans
from sklearn import metrics
```

[ ]:
```
# load data
AD6=pd.read_csv("C:/599_Research/FINAL_RESEARCH_and_PPT/THESIS_SUBMISSION/
↪APPENDIX/2_SINGLE ATTRIBUTE SCRIPTS/DATA/AD6.csv")
```

2 Stage 2 : Pre-processing & Execution

3 Pre-Processing

[ ]:
```
#inspect - example on df asc south
AD6.head()
AD6.describe()
```

[ ]:
```
#Visualize the data for the CUMULATIVE (OR ANY OTHER ATTRIBUTE), changing the␣
↪hue allows you to visualize any attribute
sns.set(style="whitegrid")
plt.scatter(AD6['LONG'],AD6['LAT'], c= AD6['D20200131'], s=1)
```

[ ]:
```
###specific to GMM!
#GMM
#the Akaike information criterion (AIC) or the Bayesian information criterion␣
↪(BIC).
X = np.array(list(zip(AD6['D20190125'],AD6['D20200131'])))
n_components = np.arange(1, 21)
models = [GMM(n, covariance_type='full', random_state=0).fit(X) for n in␣
↪n_components]
plt.plot(n_components, [m.bic(X) for m in models], label='BIC')
plt.plot(n_components, [m.aic(X) for m in models], label='AIC')
plt.legend(loc='best')
plt.xlabel('n_components');
```

[ ]:
```
def SelBest(arr:list, X:int)->list:
    '''
    returns the set of X configurations with shorter distance
    '''
    dx=np.argsort(arr)[:X]
    return arr[dx]
```

[ ]:
```
#Silhouette Score
X = np.array(list(zip(AD6['D20190125'],AD6['D20200131'])))
n_clusters=np.arange(2, 20)
sils=[]
sils_err=[]
iterations=20
for n in n_clusters:
    tmp_sil=[]
    for _ in range(iterations):
        gmm=GMM(n, n_init=2).fit(X)
        labels=gmm.predict(X)
        sil=metrics.silhouette_score(X, labels, metric='euclidean')
        tmp_sil.append(sil)
    val=np.mean(SelBest(np.array(tmp_sil), int(iterations/5)))
    err=np.std(tmp_sil)
    sils.append(val)
    sils_err.append(err)

plt.errorbar(n_clusters, sils, yerr=sils_err)
plt.title("Silhouette Scores", fontsize=20)
plt.xticks(n_clusters)
plt.xlabel("N. of clusters")
plt.ylabel("Score")
```

4 Algorithm Execution¶

[ ]:
```
#MiniBatch_Kmeans
# define dataset
X = np.array(list(zip(AD6['D20190125'],AD6['D20200131'])))
# define the model
MiniBatch_model = MiniBatchKMeans(n_clusters=6)
# fit the model
MiniBatch_model.fit(X)
# assign a cluster to each example
yhat = MiniBatch_model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)



import timeit

start = timeit.default_timer()

# All the program statements
stop = timeit.default_timer()
execution_time = stop - start

print("Program Executed in "+str(execution_time)) # It returns time in seconds

#map the labels to colors
c= ['b', 'r', 'y', 'g', 'c', 'm', 'e','f', 'u', 'd', 'a', 'h']
colors = [c[i] for i in yhat]

#Plot clusters with coordinates
figure(num=None, figsize=(10, 8), dpi=100, facecolor='w', edgecolor='k')
pyplot.scatter(AD6['LONG'], AD6['LAT'], c=yhat, s=10, cmap='viridis')
plt.savefig('AD6_MINIBATCH_6.png')
```

5 Stage 3: Outputs and Assessment

[ ]:
```
#MINIBATCH kmeans
# Number of clusters in labels, ignoring noise if present.
labels = MiniBatch_model.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)

print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, labels, metric='sqeuclidean'))
```

```
#Calinski-Harabasz Index
print("Calinski Harabasz Score: %0.3f"
      % metrics.calinski_harabasz_score(X, labels))
#Davies Bouldin Index
print("Davies Bouldin Index: %0.3f"
      % metrics.davies_bouldin_score(X, labels))

cluster_map = pd.DataFrame()
cluster_map['data_index'] = AD6.index.values
cluster_map['cluster'] = MiniBatch_model.labels_

cluster_map[cluster_map.cluster == 4]

# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

6 STAGE 4: EXPORT TO LAYER FOR ARCPRO: CSV TO SHP¶

[ ]:
```
data.to_csv('C:/599_Research/ARTIFICIAL/Permian/SA_SHP_OUTPUTS/
 ↪asc_south_kmeans_PERMIAN_VEL_6_withlabels.csv')
```

[ ]:
```
# MakeXYLayer.py
# Description: Creates an XY layer and exports it to a layer file

# import system modules
import arcpy
from arcpy import env

# Set environment settings
env.workspace = "C:/599_Research/ARTIFICIAL/Permian/SA_SHP_OUTPUTS"

# Set the local variables
in_Table = "asc_south_kmeans_PERMIAN_VEL_6_withlabels.csv"
x_coords = "LONG"
y_coords = "LAT"
z_coords = "HEIGHT"
out_Layer = "asc_south_kmeans_PERMIAN_VEL_6_withlabels_layer"
```

```
saved_Layer = r"C:/599_Research/ARTIFICIAL/Permian/SA_SHP_OUTPUTS/
↪asc_south_kmeans_PERMIAN_VEL_6_MLOUTPUT.shp"

# Set the spatial reference
spRef = arcpy.SpatialReference(4326)

# Make the XY event layer...
arcpy.MakeXYEventLayer_management(in_Table, x_coords, y_coords, out_Layer,␣
↪spRef, z_coords)

# Save to a layer file
arcpy.SaveToLayerFile_management(out_Layer, saved_Layer)
```