## Methods:

1. **def \_\_init\_\_( self, nClusters, maxIterations=300, initialCentroidMethod = 'defaultDense',**

**method = ''defaultDense', oversamplingFactor =0.5, nRounds=5,**

**accuracyThreshold = 0.0001, gamma = 1.0, distanceType = 'euclidean',**

**assignFlag = True, dtype = float64):**

Constructor to set Kmeans compute parameters

*parameters:*

**nClusters:** default: None

number of centroids to compute

**maxIterations:** default: 300

maximum number of iterations

**initialCentroidMethod:** default**:** ‘**defaultDense**’

Initial centroid assignment method**.** Refer here for other available methods

**method:** default**:** 'dense'

final centroid computation mode. Refer here for other available methods

**oversamplingFactor**: default: 0.5

applicable only if initialCentroidMethod is ‘parallelPlusDense’, ‘parallelPlusCSR’

A fraction of nClusters in each of nRounds of parallel K-Means++. *L*=nClusters\*oversamplingFactor points are sampled in a round

**nRounds:** default**:** 5

applicable only if initialCentroidMethod is ‘parallelPlusDense’, ‘parallelPlusCSR’

The number of rounds for parallel K-Means++. (*L*\*nRounds) must be greater than nClusters.

**accuracyThreshold:** default: 0.0001

The threshold for termination of the algorithm.

**gamma:** default:1.0

The weight to be used in distance calculation for binary categorical features.

**distanceType:** default: 'euclidean'

The measure of closeness between points being clustered.

**assignFlag**: default: True

Flag that enables cluster assignments for clustered data points.

1. **def compute(self, trainData):**

*parameters:*

train data values(type nT)

*returns*:

training results object

Attributes - centroidResults, clusterAssignments, objectiveFunction

1. **def predict(self, centroidResults, testData):**

*parameters:*

centroidResults(type nT), test data feature values(type nT)

*returns*:

predicted values of type nT

1. **def compress(self, arrayData):**

*parameters:*

serialized numpy array

*returns*:

Compressed numpy array

1. **def decompress(self, arrayData):**

*parameters:*

deserialized numpy array

*returns*:

decompressed numpy array

1. **def serialize(self, data, fileName=None, useCompression=False):**

*parameters:*

Method 1: data(type nT/model)

-Returns serialized numpy array

Method 2: data(type nT/model), fileName(.npy file to save serialized array to disk)

- Saves serialized numpy array as "fileName" argument

Method 3: data(type nT/model), useCompression = True

-Returns compressed numpy array

Method 4: data(type nT/model), fileName(.npy file to save serialized array to disk), useCompression = True

-Saves compressed numpy array as "fileName" argument

1. **def deserialize(self, serialObjectDict=None, fileName=None, useCompression=False):**

*parameters:*

serialized/ compressed numpy array or serialized/ compressed .npy file

*returns*:

deserialized/ decompressed numeric table/model