Assignment Project Report

Spherical K-Means: Pattern Discovery

Name: Arun Govind
Course: AI and ML
(Batch 4)

- **Problem Statement**

  Implement K-Means clustering and then use the clusters for classification purpose.

- **Prerequisites**

  - Software:
    - Python 3 (Use anaconda as your python distributor as well)

  - Tools:
    - Numpy
    - Pandas
    - Seaborn
    - Sklearn
    - Soyclustering

- **Method Used**

  The spherical k-means algorithm, i.e., the k-means algorithm with cosine similarity, is a popular method for clustering high-dimensional text data. In this algorithm, each document as well as each cluster mean is represented as a high-dimensional unit-length vector.

  In spherical k-means, the idea is to set the center of each cluster such that it makes both uniform and minimal the angle between components.

  Spherical k-means algorithm can achieve significantly better clustering results than the batch version, especially when an annealing-type learning rate schedule is used.

- **Implementation:**

1. Load all required libraries

```
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from soyclustering import SphericalKMeans
from scipy.sparse import csr_matrix
from sklearn.metrics import accuracy_score
from sklearn.manifold import TSNE
import pandas as pd
import numpy as np
```

2. Creating a dataset for train and test.

```
#default features=20
X, y = make_classification(n_samples=1000, n_classes=4, n_clusters_per_class=1, random_state=10)
```

```
X.shape
```

```
(1000, 20)
```

```
y.shape
```

```
(1000,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

```
X_train.shape
```

```
(750, 20)
```

3. Implementing K-Means clustering

```
kmeans = KMeans(n_clusters=4).fit(X_train)
```

```
kmeans.cluster_centers_.shape
```

```
(4, 2)
```

```
kmeans.labels_.shape
```

```
(750,)
```

```
kmeans.cluster_centers_.shape
```

```
(4, 2)
```

```
y_pred = kmeans.predict(X_test)
y_pred
```

```
array([3, 2, 3, 2, 3, 0, 0, 0, 3, 3, 0, 3, 2, 0, 2, 2, 3, 3, 2, 1, 2, 0,
       2, 1, 2, 0, 1, 2, 2, 2, 2, 0, 3, 0, 0, 3, 3, 0, 0, 0, 1, 1, 3, 2,
       0, 0, 3, 2, 3, 2, 2, 0, 0, 2, 2, 3, 2, 2, 3, 2, 2, 3, 0, 2, 3, 1,
       3, 0, 1, 2, 0, 3, 2, 3, 2, 3, 3, 2, 2, 0, 1, 2, 0, 1, 1, 2, 0, 2,
       0, 0, 0, 3, 2, 2, 2, 2, 2, 3, 3, 2, 3, 2, 0, 2, 3, 3, 1, 2, 2, 2,
       1, 3, 0, 0, 2, 2, 3, 3, 0, 0, 2, 2, 0, 1, 3, 3, 0, 2, 2, 2, 2, 3,
       1, 0, 0, 1, 2, 3, 3, 3, 1, 2, 2, 0, 2, 0, 3, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 0, 3, 2, 3, 2, 2, 3, 3, 2, 1, 3, 1, 3, 3, 2, 2, 2, 0, 3, 1,
       1, 0, 0, 1, 2, 2, 3, 0, 0, 3, 3, 3, 3, 2, 0, 1, 0, 3, 0, 3, 3, 2,
       3, 3, 3, 3, 3, 2, 2, 2, 1, 1, 3, 2, 2, 3, 3, 2, 1, 2, 0, 3, 1, 0,
       3, 0, 1, 3, 2, 0, 3, 0, 2, 3, 2, 0, 1, 2, 0, 0, 1, 3, 2, 1, 1, 2,
       3, 2, 3, 1, 0, 3, 1, 2], dtype=int32)
```

4. Implementing Spherical K-Means Clustering

```
X, y = make_classification(n_samples=1000, n_classes=4, n_clusters_per_class=1, random_state=10)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
spherical_kmeans = SphericalKMeans(n_clusters=4)

X_train = csr_matrix(tsne.fit_transform(X_train))
X_test = csr_matrix(tsne.fit_transform(X_test))

print(X_train.shape)

skmeans = spherical_kmeans.fit(X_train)
sy_pred = skmeans.fit_predict(X_test)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 750 samples in 0.001s...
[t-SNE] Computed neighbors for 750 samples in 0.035s...
[t-SNE] Computed conditional probabilities for sample 750 / 750
[t-SNE] Mean sigma: 1.700601
[t-SNE] KL divergence after 250 iterations with early exaggeration: 75.792328
[t-SNE] KL divergence after 300 iterations: 1.962338
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 250 samples in 0.000s...
[t-SNE] Computed neighbors for 250 samples in 0.007s...
[t-SNE] Computed conditional probabilities for sample 250 / 250
[t-SNE] Mean sigma: 1.995815
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.806404
[t-SNE] KL divergence after 300 iterations: 1.467597
(750, 2)
```

5. Graphing K-Means clustering Results

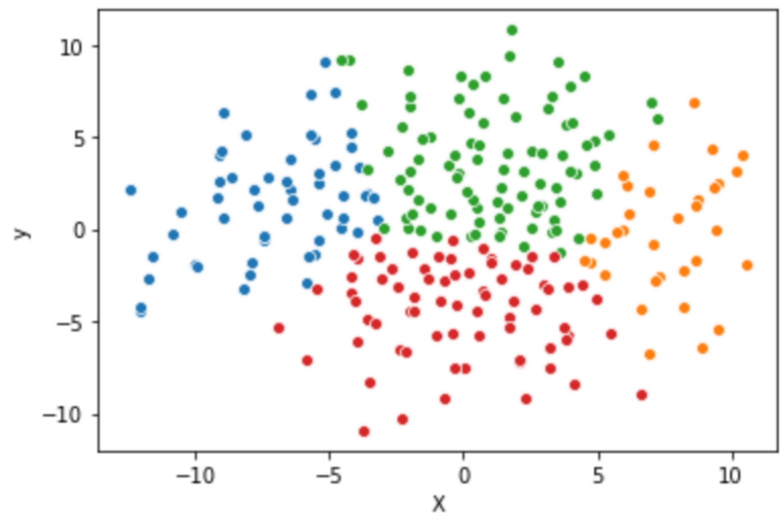
```
df = pd.DataFrame({'X': cluster1[:,0], 'y':cluster1[:,1] })
sns.scatterplot(data=df, x="X", y="y")

df = pd.DataFrame({'X': cluster2[:,0], 'y':cluster2[:,1] })
sns.scatterplot(data=df, x="X", y="y")

df = pd.DataFrame({'X': cluster3[:,0], 'y':cluster3[:,1] })
sns.scatterplot(data=df, x="X", y="y")

df = pd.DataFrame({'X': cluster4[:,0], 'y':cluster4[:,1] })
sns.scatterplot(data=df, x="X", y="y")
```

6. Graphing Spherical K-Means Results

```
df = pd.DataFrame({'X': cluster1[:,0], 'y':cluster1[:,1] })
sns.scatterplot(data=df, x="X", y="y")

df = pd.DataFrame({'X': cluster2[:,0], 'y':cluster2[:,1] })
sns.scatterplot(data=df, x="X", y="y")

df = pd.DataFrame({'X': cluster3[:,0], 'y':cluster3[:,1] })
sns.scatterplot(data=df, x="X", y="y")

df = pd.DataFrame({'X': cluster4[:,0], 'y':cluster4[:,1] })
sns.scatterplot(data=df, x="X", y="y")
```

- **Results:**

1. Resulting graph for K-Means Clustering:



2. Resulting graph for Spherical K-Means Clustering: