Aim: Implementing K means clustering using pyspark

Theory:
K-means is a popular clustering algorithm used in machine learning and data analysis to partition a dataset into a predefined number of clusters. Each data point is assigned to the cluster with the nearest mean (center). This method is advantageous for its simplicity, efficiency, and ease of interpretation.

Advantages of K-means:

1. Simplicity: K-means is easy to understand and implement. It is a straightforward and intuitive algorithm for clustering.
2. Efficiency: K-means is computationally efficient and works well with large datasets, making it suitable for a wide range of applications.
3. Scalability: It can be used in both low-dimensional and high-dimensional data spaces, making it versatile for various data types.
4. Fast Convergence: K-means usually converges quickly to a local minimum, making it efficient for finding cluster centers.
5. Ease of Interpretation: The results of K-means are easy to interpret, as data points are assigned to a single cluster.

Disadvantages of K-means:

1. Sensitive to Initialization: The results of K-means can vary depending on the initial positions of cluster centers. Poor initializations can lead to suboptimal clustering.
2. Assumes Circular Clusters: K-means assumes that clusters are spherical and equally sized, which may not hold true for all datasets. It can perform poorly on elongated or irregularly shaped clusters.
3. Fixed Number of Clusters: K-means requires specifying the number of clusters (K) in advance, which can be challenging when the true number of clusters is unknown.
4. Outlier Sensitivity: Outliers can significantly impact K-means clustering results, causing cluster centers to be skewed.
5. May Converge to Local Optima: K-means converges to a local minimum, so different initializations may lead to different solutions, and it might not always find the global optimum.

Applications of K-means:

1. Image Compression: Used to reduce the number of colors in an image while preserving visual quality, making it efficient for image compression.
2. Customer Segmentation: Segments customers based on purchasing behavior or demographics for targeted marketing.
3. Anomaly Detection: Identifies anomalies or outliers by treating them as data points that do not fit into any cluster.
4. Document Clustering: Clusters documents (e.g., news articles) into related topics in natural language processing.
5. Genomic Data Analysis: Clusters genes with similar expression patterns in bioinformatics and genomics research.
6. Recommendation Systems: Groups users or items in recommendation systems to make personalized recommendations.
7. Geographical Data Analysis: Finds natural groupings in geographical data, such as classifying geographic regions.
8. Market Segmentation: Segments markets and customers into distinct groups for targeted advertising and product development.

Steps:

1. Initialize a Spark Session: Create a SparkSession to configure Spark settings.
2. Load Data: Load the dataset into a PySpark DataFrame using read methods.
3. Data Preparation: Perform data preprocessing, which includes data cleaning, feature selection, and transformation.
4. Feature Vectorization: Use PySpark's VectorAssembler to convert data into feature vectors.
5. K-Means Model Configuration: Create an instance of the K-Means model with the desired number of clusters (K) and other options.
6. Model Training: Fit the K-Means model to the data to calculate cluster centers and assign data points to clusters.
7. Cluster Assignment: Use the trained model to assign data points to clusters.
8. Cluster Centers: Retrieve cluster centers from the model.
9. Conclusion Analysis: Evaluate clustering quality using metrics, visualize clusters and centroids, and use clusters for downstream tasks.
10. Cleanup: Stop the Spark session to release resources.

Conclusion:
In conclusion, K-means is a straightforward and efficient clustering algorithm with advantages in simplicity, speed, and interpretability. However, it has limitations such as sensitivity to initialization, assumption of spherical clusters, and the need to specify the number of clusters in advance. Therefore, its suitability for a specific problem should be carefully evaluated. In cases where K-means is not a good fit due to its assumptions and limitations, exploring other clustering algorithms such as hierarchical clustering, DBSCAN, or Gaussian Mixture Models might be necessary. The choice of clustering algorithm should be based on the dataset's characteristics and analysis goals.

Notebook

October 15, 2023

[1]:
```
!pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
                             316.9/316.9
MB 4.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-
packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) … done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl
size=317425344
sha256=becbdb1b43bfef238ce1b44af35fe0b575967185f2a2761c361f074ecaa8266d
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9a
c9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

[2]:
```
import pandas as pd
import numpy as np

from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
```

[4]:
```
spark = SparkSession.builder.appName("KMeansExamp1e").getOrCreate()
```

[5]:
```
roll_numbers = range(1, 73)
bda_values = np.random.randint(8, 21, size = 72)
nlp_values = np.random.randint(8, 21, size = 72)
ml_values = np.random.randint(8, 21, size = 72)
ir_values = np.random.randint(8, 21, size = 72)
plm_values = np.random.randint(8, 21, size = 72)
cs_values = np.random.randint(8, 21, size = 72)
```

[6]:
```
data = pd.DataFrame({
    'Roll Number': roll_numbers,
    'BDA': bda_values,
    'NLP': nlp_values,
    'ML': ml_values,
    'IR': ir_values,
    'PLM': plm_values,
    'CS': cs_values,
})
```

[7]:
```
data.to_csv('data.csv', index = False)
data
```

[7]:

	Roll Number	BDA	NLP	ML	IR	PLM	CS
0	1	17	17	17	11	20	16
1	2	18	12	13	17	19	14
2	3	18	20	12	10	13	14
3	4	11	10	17	20	8	16
4	5	13	17	20	14	14	20
..	…	…	…	..	..	…	..
67	68	18	20	19	17	9	19
68	69	11	16	10	9	9	10
69	70	11	14	9	13	18	14
70	71	12	19	14	9	17	16
71	72	19	8	8	16	9	17

[72 rows x 7 columns]

[8]:
```
data = spark.read.csv("data.csv", header = True, inferSchema = True)
```

[9]:
```
feature_cols = ["BDA", "NLP", "ML", "IR", "PLM", "CS"]
vec_assembler = VectorAssembler(inputCols = feature_cols, outputCol =␣
 ↪"features")
data = vec_assembler.transform(data)
```

[10]:
```
kmeans = KMeans().setK(4)
kmeans = kmeans.setSeed(1)
model = kmeans.fit(data)
```

[11]:
```
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
  print(center)
```

```
Cluster Centers:
[15.64285714 16.64285714 17.21428571 12.71428571 15.85714286 15.64285714]
[11.1875 13.625  12.75   10.4375 11.     12.4375]
```

```
[13.76923077 13.38461538 11.         15.76923077 14.42307692 15.46153846]
[15.5    11.125  16.5625 15.8125 10.25   11.6875]
```

[12]:
```
predictions = model.transform(data)
predictions.select("prediction").show( )
```

```
+----------+
|prediction|
+----------+
|         0|
|         2|
|         0|
|         3|
|         0|
|         2|
|         1|
|         3|
|         3|
|         2|
|         3|
|         1|
|         1|
|         2|
|         1|
|         2|
|         2|
|         1|
|         2|
|         2|
+----------+
only showing top 20 rows
```

[13]:
```
spark.stop()
```