Terna Engineering College
Computer Engineering Department

Program: Sem VII

Course: Big Data Analytics & Computational Lab -I (BDA&CL-I)

Experiment No. 09

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Blackboard access available)

Roll No. 50	Name: AMEY THAKUR
Class: BE-COMPS-50	Batch: B3
Date of Experiment: 05-10-2021	Date of Submission: 05-10-2021
Grade :	

Aim: Implement Clustering Algorithm Using Map-Reduce.

B.1. Clustering:

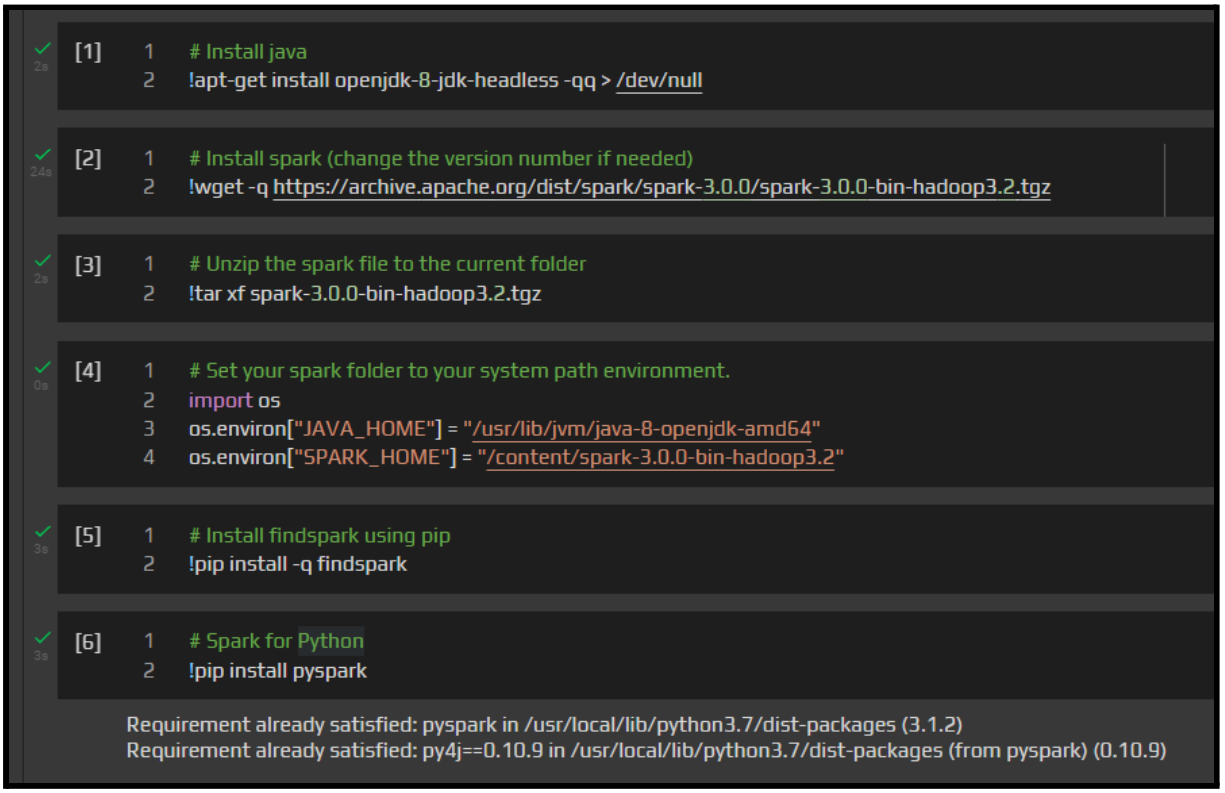
Clustering is an unsupervised learning technique, in short, you are working on data, without having any information about a target attribute or a dependent variable. The general idea of clustering is to find some intrinsic structure in the data, often referred to as groups of similar objects. The algorithm studies the data to identify these patterns or groups such that each member in a group is closer to another member in the group (lower intracluster distance) and farther from another member in a different group (higher inter-cluster distance).

B.2 Input and Output:

Data: The dataset consists of 9K active credit cardholders over 6 months and their transaction and account attributes. The idea is to develop a customer segmentation for marketing strategy.

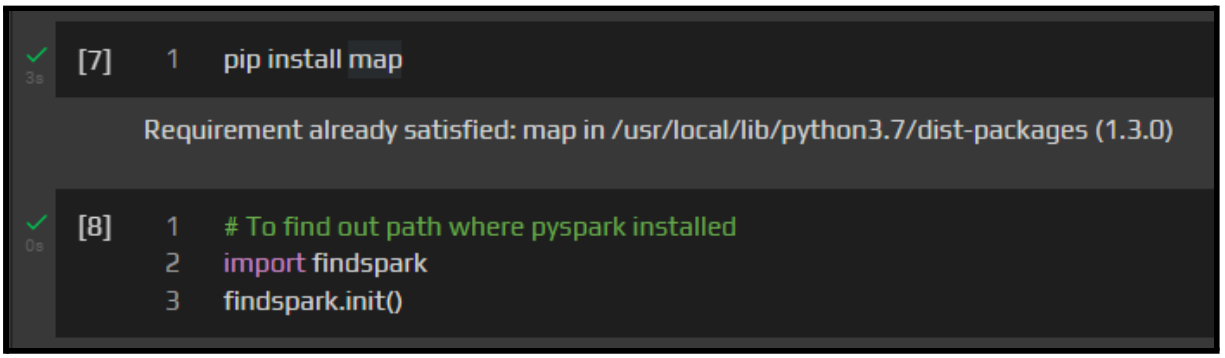
Using PySpark:

➔ **Step 1:** Installation of Hadoop and PySpark in Colab.



```
[1]  # Install java
     !apt-get install openjdk-8-jdk-headless -qq > /dev/null

[2]  # Install spark (change the version number if needed)
     !wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz

[3]  # Unzip the spark file to the current folder
     !tar xf spark-3.0.0-bin-hadoop3.2.tgz

[4]  # Set your spark folder to your system path environment.
     import os
     os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
     os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"

[5]  # Install findspark using pip
     !pip install -q findspark

[6]  # Spark for Python
     !pip install pyspark

Requirement already satisfied: pyspark in /usr/local/lib/python3.7/dist-packages (3.1.2)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark) (0.10.9)
```

➔ **Step 2:** Installing Map and find out the path where PySpark is installed.



```
[7]  pip install map

Requirement already satisfied: map in /usr/local/lib/python3.7/dist-packages (1.3.0)

[8]  # To find out path where pyspark installed
     import findspark
     findspark.init()
```

➔ **Step 3:** Schema information of the dataset.

```
[8]  1  # To find out path where pyspark installed
     2  import findspark
     3  findspark.init()
```

```
[9]  1  from pyspark.sql import SparkSession
     2  spark = SparkSession.builder.appName("Clustering using K-Means").getOrCreate()
     3  data_customer = spark.read.csv('CC GENERAL.csv', header=True, inferSchema=True)
     4  data_customer.printSchema()
```

```
root
 |-- CUST_ID: string (nullable = true)
 |-- BALANCE: double (nullable = true)
 |-- BALANCE_FREQUENCY: double (nullable = true)
 |-- PURCHASES: double (nullable = true)
 |-- ONEOFF_PURCHASES: double (nullable = true)
 |-- INSTALLMENTS_PURCHASES: double (nullable = true)
 |-- CASH_ADVANCE: double (nullable = true)
 |-- PURCHASES_FREQUENCY: double (nullable = true)
 |-- ONEOFF_PURCHASES_FREQUENCY: double (nullable = true)
 |-- PURCHASES_INSTALLMENTS_FREQUENCY: double (nullable = true)
 |-- CASH_ADVANCE_FREQUENCY: double (nullable = true)
 |-- CASH_ADVANCE_TRX: integer (nullable = true)
 |-- PURCHASES_TRX: integer (nullable = true)
 |-- CREDIT_LIMIT: double (nullable = true)
 |-- PAYMENTS: double (nullable = true)
 |-- MINIMUM_PAYMENTS: double (nullable = true)
 |-- PRC_FULL_PAYMENT: double (nullable = true)
 |-- TENURE: integer (nullable = true)
```

➔ **Step 4:** All attributes under consideration are numerical or discrete numeric, hence we need to convert them into features using a Vector Assembler. Since customer id is an identifier that won't be used for clustering, we first extract the required columns using .columns, pass it as an input to Vector Assembler, and then use the transform() to convert the input columns into a single vector column called a feature.

```
[10]  1  data_customer=data_customer.na.drop()
```

```
[11]  1  from pyspark.ml.feature import VectorAssembler
      2  data_customer.columns
      3  assemble=VectorAssembler(inputCols=[
      4  'BALANCE',
      5  'BALANCE_FREQUENCY',
      6  'PURCHASES',
      7  'ONEOFF_PURCHASES',
      8  'INSTALLMENTS_PURCHASES',
      9  'CASH_ADVANCE',
     10  'PURCHASES_FREQUENCY',
     11  'ONEOFF_PURCHASES_FREQUENCY',
     12  'CASH_ADVANCE_FREQUENCY',
     13  'CASH_ADVANCE_TRX',
     14  'PURCHASES_TRX',
     15  'CREDIT_LIMIT',
     16  'PAYMENTS',
     17  'MINIMUM_PAYMENTS',
     18  'PRC_FULL_PAYMENT',
     19  'TENURE' ], outputCol='features')
     20  assembled_data=assemble.transform(data_customer)
     21  assembled_data.show(2)
```

```
+-------+-----------+----------------+---------+---------------+--------------------+-----------+------------------
|CUST_ID|    BALANCE|BALANCE_FREQUENCY|PURCHASES|ONEOFF_PURCHASES|INSTALLMENTS_
+-------+-----------+----------------+---------+---------------+--------------------+-----------+------------------
| C10001|  40.900749|        0.818182|     95.4|            0.0|                95.4|        0.0|         0.166667|
| C10002|3202.467416|        0.909091|      0.0|            0.0|                 0.0| 6442.945483|           0.0|
+-------+-----------+----------------+---------+---------------+--------------------+-----------+------------------
only showing top 2 rows
```

➔ **Step 5:** Now that all columns are transformed into a single feature vector we need to standardize the data to bring them to a comparable scale.

```
[12]  1  from pyspark.ml.feature import StandardScaler
      2  scale=StandardScaler(inputCol='features',outputCol='standardized')
      3  data_scale=scale.fit(assembled_data)
      4  data_scale_output=data_scale.transform(assembled_data)
      5  data_scale_output.show(2)
```

```
+-------+----------+----------------+---------+--------------+--------------------+-----------+-------------------+
|CUST_ID|   BALANCE|BALANCE_FREQUENCY|PURCHASES|ONEOFF_PURCHASES|INSTALLMENTS_
+-------+----------+----------------+---------+--------------+--------------------+-----------+-------------------+
| C10001| 40.900749|        0.818182|    95.4|           0.0|                95.4|        0.0|           0.166667|
| C10002|3202.467416|        0.909091|     0.0|           0.0|                 0.0|6442.945483|                0.0|
+-------+----------+----------------+---------+--------------+--------------------+-----------+-------------------+
only showing top 2 rows
```

➔ **Step 6:** Now that our data is standardized we can develop the K Means algorithm.

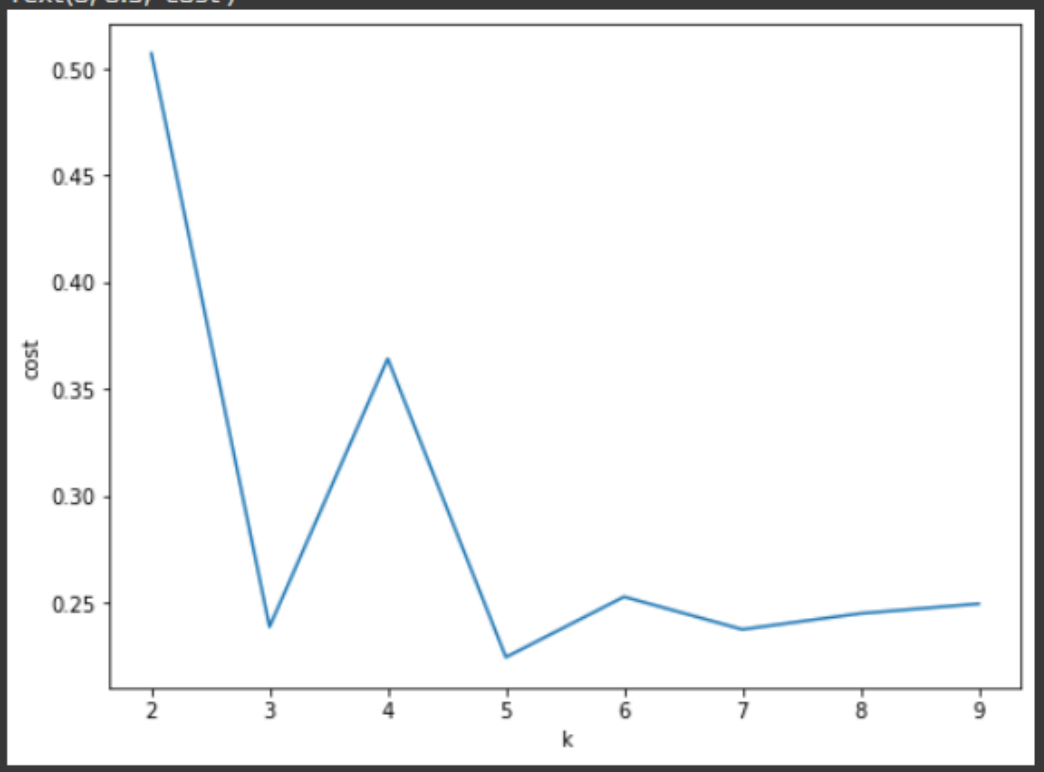
```
[13]  1  from pyspark.ml.clustering import KMeans
      2  from pyspark.ml.evaluation import ClusteringEvaluator
      3  silhouette_score=[]
      4  evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='standardized', \
      5                                  metricName='silhouette', distanceMeasure='squaredEuclidean')
      6  for i in range(2,10):
      7
      8      KMeans_algo=KMeans(featuresCol='standardized', k=i)
      9
     10      KMeans_fit=KMeans_algo.fit(data_scale_output)
     11
     12      output=KMeans_fit.transform(data_scale_output)
     13
     14      score=evaluator.evaluate(output)
     15
     16      silhouette_score.append(score)
     17
     18      print("Silhouette Score:",score)
```

```
Silhouette Score: 0.5072826123334669
Silhouette Score: 0.23859448410725492
Silhouette Score: 0.36422379640356556
Silhouette Score: 0.22433597716882983
Silhouette Score: 0.2526487106651543
Silhouette Score: 0.2373718932736326
Silhouette Score: 0.2448404473492103
Silhouette Score: 0.2493487881988621
```

➔ **Step 7:** After Visualizing the silhouette score.

```
[14]  1  #Visualizing the silhouette scores in a plot
      2  import matplotlib.pyplot as plt
      3  fig, ax = plt.subplots(1,1, figsize =(8,6))
      4  ax.plot(range(2,10),silhouette_score)
      5  ax.set_xlabel('k')
      6  ax.set_ylabel('cost')
```

```
Text(0, 0.5, 'cost')
```



B.3 Observations and learning:

I observed that K-means clustering is a classical clustering algorithm that uses an expectation-maximization like technique to partition a number of data points into k clusters and MapReduce is a style of computing that has been implemented in this system.

B.4 Conclusion:

We have successfully implemented k-means clustering using Hadoop Map Reduce along in python.

B.5 Question of Curiosity:

1. Explain clustering strategies?

Ans:

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

2. What are the clustering applications?

Ans:

Clustering techniques can be used in various areas or fields of real-life examples such as data mining, web cluster engines, academics, bioinformatics, image processing & transformation, and many more and emerged as an effective solution to the above-mentioned areas.

3. How is clustering different from classification?

Ans:

BASIS FOR COMPARISON	CLASSIFICATION	CLUSTERING
Basic	This model function classifies the data into one of numerous already defined definite classes.	This function maps the data into one of the multiple clusters where the arrangement of data items relies on the similarities between them.
Involved in	Supervised learning	Unsupervised learning
Training sample	Labelled data is provided.	Unlabeled data was provided.