ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

Clustering of Datasets Using K-Means Algorithm in SPARK

Subiksha N[1], Pallavi R Reddy[1], Mounica B[2]

B. E Student, Department of ISE, New Horizon College of Engineering, Bengaluru, Karnataka, India[1]

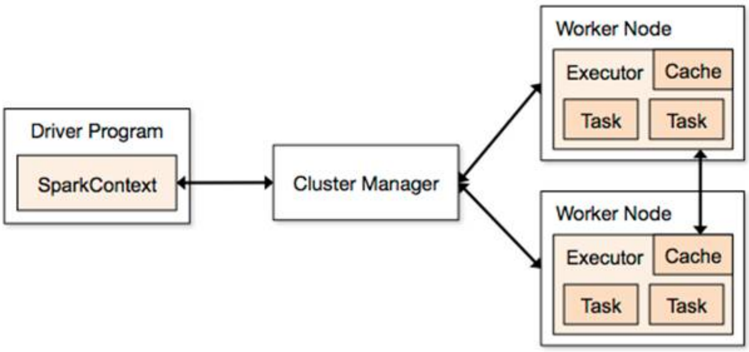
Assistant Professor, Department of ISE, New Horizon College of Engineering, Bengaluru, Karnataka, India[2]

ABSTRACT: With The growing trend of large volume of data generated every day, storing and processing the data is a great challenge. The existing solutions like SAS, R, Excel and MapReduce prove to be inefficient. MapReduce is a part of Apache Hadoop that allows distributive processing of unstructured data, where each distributive node has its own storage. Most of the algorithms are iterative in nature. But MapReduce is bane as it involves undesirable amount of read and writes to process such iterative algorithms. As an advancement, in this paper, we use the SPARK to implement algorithms like K-means, linear regression etc on real-time data. Spark is an unified engine that can run Hadoop, Mesos, cloud or standalone. It stores the intermediate result in RAM, thus avoiding read/write from/to disk. Like the MapReduce, Spark is used for batch processing. In addition, Spark can be used to handle streaming data, queries and machine learning. In this paper, we prefer to use the cloud services to access the large datasets because cloud is flexible, scalable and involves less hardware cost than using our own infrastructure which requires many software's (Hadoop, Ubuntu, Java etc) to be installed. Once the data is obtained, we run the K-means algorithm on Spark using EMR (Elastic MapReduce).

KEYWORDS: MapReduce, iterative algorithms, Spark, K-means, cloud services

I. INTRODUCTION

Hadoop is used extensively in industries to analyze data sets, as this framework is based on simple programming models. The solutions are scalable,flexible,cost-effective and fault-tolerant. The main concern is to maintain speed in processing large datasets. To overcome this Spark was introduced for improving the computational speed. In memory cluster computing is the main feature that increases the processing speed. Spark and Hadoop are different, Spark is not a modified version of Hadoop. Spark has its own cluster management. Storage and processing is the two ways in which Spark uses Hadoop. It is used only for storage as Spark has its own cluster management.Spark is free and an open source framework, written in Java.



DOI:10.15680/IJIRCCE.2017. 0505263

DATAFRAME

Collections of data that is organized into columns is a dataframe. It is equal to relational tables with some good optimization techniques. This can be constructed from an array of different sources. This was mainly designed for modern Big Data and data science applications

Some of the features of dataframes are:

- It can process the data in the form of Kilobytes till petabytes.
- This Supports different data forms and storage systems like HDFS, HIVE tables, mysql, etc.
- This can be easily integrated with all Big Data tools and frameworks using the Spark-Core component.
- This also provides an API for Python, Java, Scala, and R.

RDD (Resilient Distributed Dataset)

RDD’s are the fundamental data structures of spark. RDD is Resilient Distributed Datasets which is an immutable distributed collection of objects. RDD’s can have any type objects including Python, Java or Scala. It is a partitioned collection of records in the read-only mode. These are fault tolerant collection of elements which can be operated in parallel. There are two different ways to create RDD’s: to parallelize an existing collection or for referencing a dataset which is stored in an external storage like HDFS or HBase. To achieve faster and efficient MapReduce operations Spark uses the RDD concept.

COMPONENTS OF SPARK

Apache Spark Core- This is the engine for Spark execution in which all the functionality is built upon. The main function is it provides in-memory computing and datasets which are stored on external storage systems can be accessed, like the data can be there on cloud or in local like HDFS.

Spark SQL- This is the component which provides support for structured and semi-structured data. It is placed above the Spark Core.

Spark Streaming- The data is taken in the form of mini-batches and RDD is performed on mini-batches.

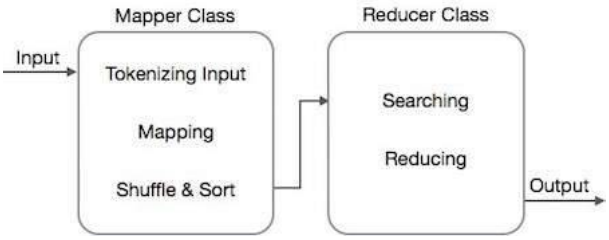
MLib(Machine Learning library)- This is placed above Spark because of distributed memory based Spark architecture. This is nine times faster than the Hadoop disk based version of Mahout.

GraphX- It is a graph processing framework. It provides an API for graph computation. Provides an optimized run time for this abstraction.

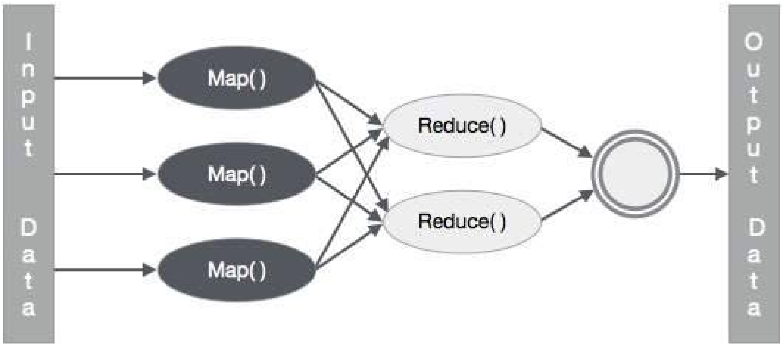
II. Related work

MapReduce

BigData can be processed in parallel on model nodes using MapReduce programming model[1].As data is accumulating in large quantities everyday it is not possible to process BigData using the traditional methods. Traditional systems use the centralized server to process and store the data. But this is not suitable for large data. So, Google developed the MapReduce algorithm. In MapReduce the task is divided into many small parts and each part is given to a computer. This algorithm performs two tasks: Map- in this one dataset is converted to another dataset. The individual elements are broken down into tuples like the key-value pairs. Reduce- The input to the reduce is the output of the Map function. This combines the key value pairs into smaller set of tuples. Reduce is performed after the Map task.



The MapReduce works well only on batch processing[2]. This is basically designed only for one specific problem domain. Often this programming paradigm is hard to understand.



III. Proposed work

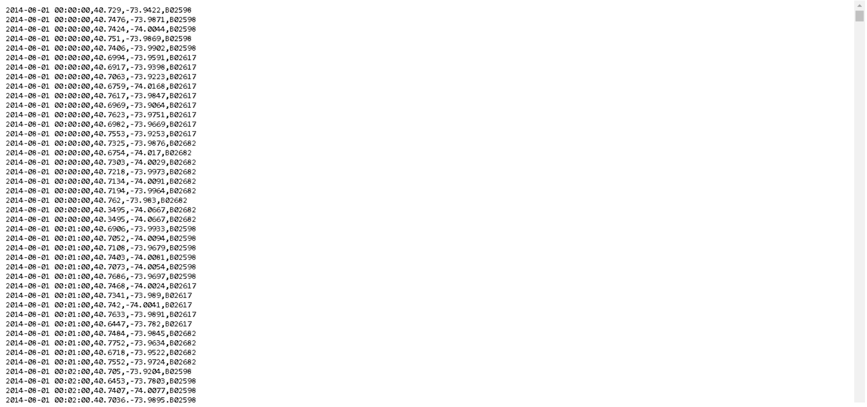
To overcome the time overhead involved in executing iterative algorithms like K-Means using MapReduce , in this paper we prefer to use a different framework called Spark. Spark is an unified engine that is 100 times faster than MapReduce.

A) *Case Study*

We consider the Uber trip data for clustering based on location and time. The input data has three features

- Data and time
- Latitude
- Longitude

The input data is in the form of CSV (Comma Separated) file. Initially, we consider five clusters as, B02617, B02598, B02682, B02764 and B02512.



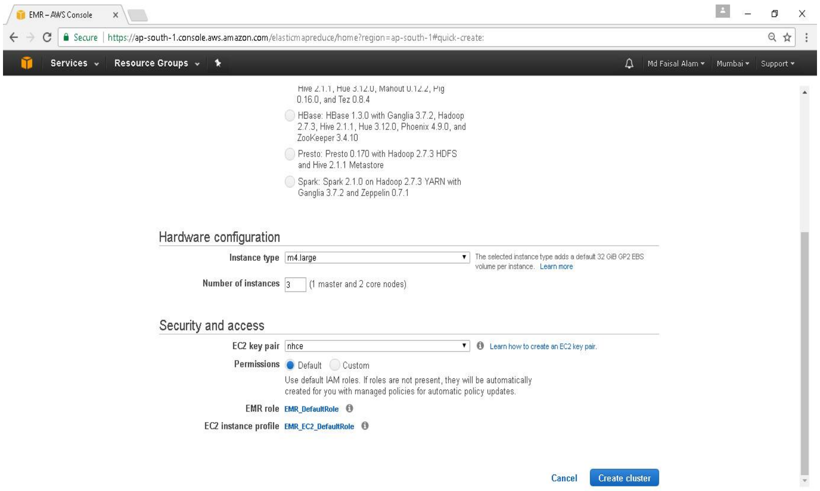
```
2014-08-01 00:00:00,40.729,-73.9422,B02598
2014-08-01 00:00:00,40.7476,-73.9871,B02598
2014-08-01 00:00:00,40.7424,-74.0044,B02598
2014-08-01 00:00:00,40.751,-73.9869,B02598
2014-08-01 00:00:00,40.7406,-73.9902,B02598
2014-08-01 00:00:00,40.6994,-73.9591,B02617
2014-08-01 00:00:00,40.6917,-73.9398,B02617
2014-08-01 00:00:00,40.7063,-73.9223,B02617
2014-08-01 00:00:00,40.6759,-74.0168,B02617
2014-08-01 00:00:00,40.7617,-73.9847,B02617
2014-08-01 00:00:00,40.6969,-73.9064,B02617
2014-08-01 00:00:00,40.7623,-73.9751,B02617
2014-08-01 00:00:00,40.6982,-73.9669,B02617
2014-08-01 00:00:00,40.7553,-73.9253,B02617
2014-08-01 00:00:00,40.7325,-73.9876,B02682
2014-08-01 00:00:00,40.6754,-74.017,B02682
2014-08-01 00:00:00,40.7303,-74.0029,B02682
2014-08-01 00:00:00,40.7218,-73.9973,B02682
2014-08-01 00:00:00,40.7134,-74.0091,B02682
2014-08-01 00:00:00,40.7194,-73.9964,B02682
2014-08-01 00:00:00,40.762,-73.983,B02682
2014-08-01 00:00:00,40.3495,-74.0667,B02682
2014-08-01 00:00:00,40.3495,-74.0667,B02682
2014-08-01 00:01:00,40.6906,-73.9933,B02598
2014-08-01 00:01:00,40.7052,-74.0094,B02598
2014-08-01 00:01:00,40.7108,-73.9679,B02598
2014-08-01 00:01:00,40.7403,-74.0081,B02598
2014-08-01 00:01:00,40.7073,-74.0054,B02598
2014-08-01 00:01:00,40.7686,-73.9697,B02598
2014-08-01 00:01:00,40.7468,-74.0024,B02617
2014-08-01 00:01:00,40.7341,-73.989,B02617
2014-08-01 00:01:00,40.742,-74.0041,B02617
2014-08-01 00:01:00,40.7633,-73.9891,B02617
2014-08-01 00:01:00,40.6447,-73.782,B02617
2014-08-01 00:01:00,40.7484,-73.9845,B02682
2014-08-01 00:01:00,40.7752,-73.9634,B02682
2014-08-01 00:01:00,40.6718,-73.9522,B02682
2014-08-01 00:01:00,40.7552,-73.9724,B02682
2014-08-01 00:02:00,40.705,-73.9204,B02598
2014-08-01 00:02:00,40.6453,-73.7803,B02598
2014-08-01 00:02:00,40.7407,-74.0077,B02598
2014-08-01 00:02:00.40.7036.-73.9895.B02598
```

B) *Implementation*

Step 1: As mentioned earlier, we prefer to use the cloud services (Amazon web services) to obtain the desired infrastructure. We create a cluster consisting of say, three “m4.large” machines, each machine having the following configuration:

vCPU Count	- 2
RAM	-8Gib
Instance Storage	-EBS (Elastic Block Storage)
Network Performance	-Moderate
EBS-Optimized	-450Mbps

In this step, we also choose the required software configuration (software’s) like Spark, Hadoop, Hive etc.



EMR – AWS Console
Secure
https://ap-south-1.console.aws.amazon.com/elasticmapreduce/home?region=ap-south-1#quick-create:
Services
Resource Groups
Md Faisal Alam
Mumbai
Support

Hive 2.1.1, Hue 3.12.0, Mahout 0.12.2, Pig 0.16.0, and Tez 0.8.4

○ HBase: HBase 1.3.0 with Ganglia 3.7.2, Hadoop 2.7.3, Hive 2.1.1, Hue 3.12.0, Phoenix 4.9.0, and ZooKeeper 3.4.10

○ Presto: Presto 0.170 with Hadoop 2.7.3 HDFS and Hive 2.1.1 Metastore

○ Spark: Spark 2.1.0 on Hadoop 2.7.3 YARN with Ganglia 3.7.2 and Zeppelin 0.7.1

Hardware configuration

Instance type: m4.large — The selected instance type adds a default 32 GiB GP2 EBS volume per instance. Learn more

Number of instances: 3 (1 master and 2 core nodes)

Security and access

EC2 key pair: nhce — Learn how to create an EC2 key pair.

Permissions: ● Default ○ Custom

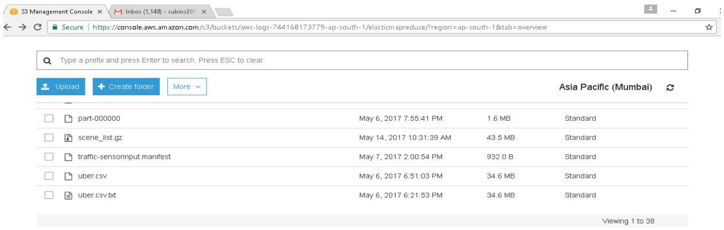
Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role: EMR_DefaultRole

EC2 instance profile: EMR_EC2_DefaultRole

We require a key pair with name of our choice to access these services from our local machine.

Step 2: Next, we upload the data on which we work to the S3 (Simple Storage Service). S3 is a storage service provided by Amazon web services. It can handle up to many terabytes of data. Once the data is uploaded, we can use the S3 for further references to it.

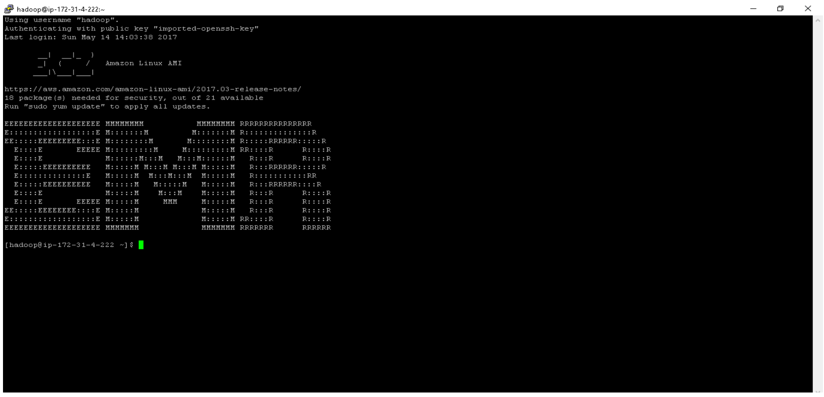

S3 Management Console
Inbox (1,149) - subins281
Secure
https://console.aws.amazon.com/s3/buckets/aws-logs-744168173779-ap-south-1/elasticmapreduce/?region=ap-south-1&tab=overview
Upload
Create folder
More
Asia Pacific (Mumbai)

Type a prefix and press Enter to search. Press ESC to clear.

□ part-000000	May 6, 2017 7:55:41 PM	1.6 MB	Standard
□ scene_list.gz	May 14, 2017 10:31:39 AM	43.5 MB	Standard
□ traffic-sensorinput.manifest	May 7, 2017 2:00:54 PM	932.0 B	Standard
□ uber.csv	May 6, 2017 6:51:03 PM	34.6 MB	Standard
□ uber.csv.txt	May 6, 2017 6:21:53 PM	34.6 MB	Standard

Viewing 1 to 38

Step 3: Next, we install PuTTy to access the cloud services from our local machine using key value pair created in step 1. This instantiates EMR (Elastic Map Reduce) which has all the required software's like Spark, Mahout, Hive pre installed.



```
Using username "hadoop".
Authenticating with public key "imported-openssh-key"
Last login: Sun May 14 14:03:38 2017

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
18 package(s) needed for security, out of 21 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-4-222 ~]$
```

Step 4: Using vi commands, we type the code for the K-means algorithm[3] in any programming language. We consider, Python.

Methods/functions used	Meaning
1. Dataframe	Dataframe consists of the Dataset in named columns.
2. Dataframe.show()	Displays the first 20 rows of the dataset.
3. VectorAssembler (*InputCols, OutputCol*)	It combines the various columns specified in *InputCols* parameter into a single column in the form of list with the name given in *OutputCol* parameter.
4. StringIndexer (*InputCos, OutputCol*)	It converts the string values into integers.
5. Randomsplit([n1,n2])	Splits the input data into two parts as specified in the parameter.Ex- n1=0.8 , n2=0.2
6. Kmeans (*k, seed, featuresCol*)	Performs the clustering into 'k' clusters on the parameter *featuresCol* with any intial seed value.
7. saveAsTextFile (output_path)	Saves the output to the specified location in the S3.
8. Dataframe.cache()	Stores the dataframe in the cache memory.

Step 5: Next, we run the code using the following command
spark-submit codefile.py inputfile_path outputfile_path

codefile.py is the file containing python code
inputfile_path is the S3 location of the raw input data
outputfile_path is the S3 location of the final output.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

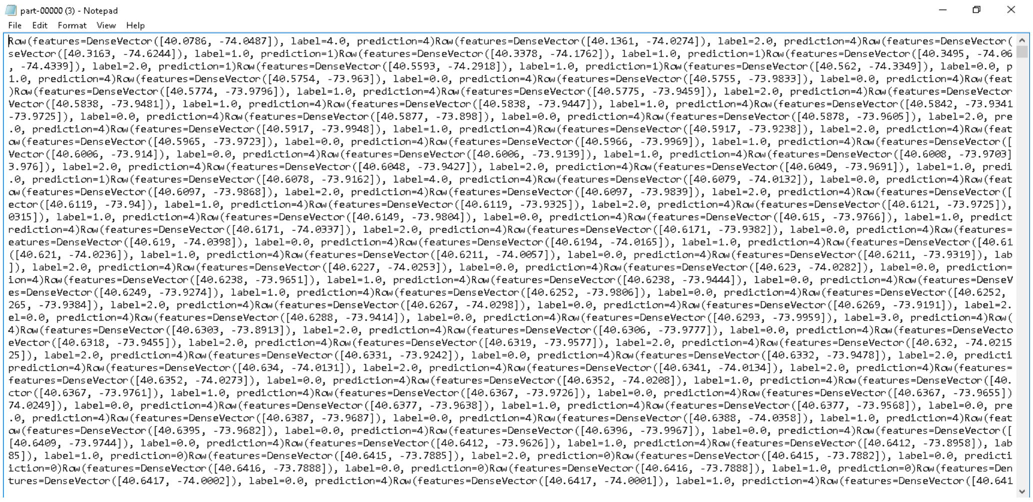
(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

IV. Results

The result of the clustering is stored in S3. It consists of a column named “prediction” which displays the cluster to which the algorithm has assigned the data.



```
Row(features=DenseVector([40.0786, -74.0487]), label=4.0, prediction=4)Row(features=DenseVector([40.1361, -74.0274]), label=2.0, prediction=4)Row(features=DenseVector(
seVector([40.3163, -74.6244]), label=1.0, prediction=1)Row(features=DenseVector([40.3378, -74.1762]), label=1.0, prediction=1)Row(features=DenseVector([40.3495, -74.06
, -74.4339]), label=2.0, prediction=1)Row(features=DenseVector([40.5593, -74.2918]), label=1.0, prediction=1)Row(features=DenseVector([40.562, -74.3349]), label=0.0, p
1.0, prediction=4)Row(features=DenseVector([40.5754, -73.963]), label=0.0, prediction=4)Row(features=DenseVector([40.5755, -73.9833]), label=0.0, prediction=4)Row(feat
)Row(features=DenseVector([40.5774, -73.9796]), label=1.0, prediction=4)Row(features=DenseVector([40.5775, -73.9459]), label=2.0, prediction=4)Row(features=DenseVector
Vector([40.5838, -73.9481]), label=1.0, prediction=4)Row(features=DenseVector([40.5838, -73.9447]), label=1.0, prediction=4)Row(features=DenseVector([40.5842, -73.9341
-73.9725]), label=0.0, prediction=4)Row(features=DenseVector([40.5877, -73.898]), label=0.0, prediction=4)Row(features=DenseVector([40.5878, -73.9605]), label=2.0, pre
.0, prediction=4)Row(features=DenseVector([40.5917, -73.9948]), label=1.0, prediction=4)Row(features=DenseVector([40.5917, -73.9238]), label=2.0, prediction=4)Row(feat
ow(features=DenseVector([40.5965, -73.9723]), label=0.0, prediction=4)Row(features=DenseVector([40.5966, -73.9969]), label=1.0, prediction=4)Row(features=DenseVector([
Vector([40.6006, -73.914]), label=0.0, prediction=4)Row(features=DenseVector([40.6006, -73.9139]), label=1.0, prediction=4)Row(features=DenseVector([40.6008, -73.9703]
3.976]), label=2.0, prediction=4)Row(features=DenseVector([40.6048, -73.9427]), label=2.0, prediction=4)Row(features=DenseVector([40.6049, -73.9691]), label=1.0, predi
.0, prediction=1)Row(features=DenseVector([40.6078, -73.9162]), label=4.0, prediction=4)Row(features=DenseVector([40.6079, -74.0132]), label=0.0, prediction=4)Row(feat
ow(features=DenseVector([40.6097, -73.9868]), label=2.0, prediction=4)Row(features=DenseVector([40.6097, -73.9839]), label=2.0, prediction=4)Row(features=DenseVector([
ector([40.6119, -73.94]), label=1.0, prediction=4)Row(features=DenseVector([40.6119, -73.9325]), label=2.0, prediction=4)Row(features=DenseVector([40.6121, -73.9725]),
0315]), label=1.0, prediction=4)Row(features=DenseVector([40.6149, -73.9804]), label=0.0, prediction=4)Row(features=DenseVector([40.615, -73.9766]), label=1.0, predict
rediction=4)Row(features=DenseVector([40.6171, -74.0337]), label=2.0, prediction=4)Row(features=DenseVector([40.6171, -73.9382]), label=0.0, prediction=4)Row(features=
eatures=DenseVector([40.619, -74.0398]), label=0.0, prediction=4)Row(features=DenseVector([40.6194, -74.0165]), label=1.0, prediction=4)Row(features=DenseVector([40.61
([40.621, -74.0236]), label=1.0, prediction=4)Row(features=DenseVector([40.6211, -74.0057]), label=0.0, prediction=4)Row(features=DenseVector([40.6211, -73.9319]), lab
]), label=2.0, prediction=4)Row(features=DenseVector([40.6227, -74.0253]), label=0.0, prediction=4)Row(features=DenseVector([40.623, -74.0282]), label=0.0, prediction=
ion=4)Row(features=DenseVector([40.6238, -73.9651]), label=1.0, prediction=4)Row(features=DenseVector([40.6238, -73.9444]), label=0.0, prediction=4)Row(features=DenseV
es=DenseVector([40.6249, -73.9274]), label=1.0, prediction=4)Row(features=DenseVector([40.6252, -73.9806]), label=0.0, prediction=4)Row(features=DenseVector([40.6252,
265, -73.9384]), label=2.0, prediction=4)Row(features=DenseVector([40.6267, -74.0298]), label=0.0, prediction=4)Row(features=DenseVector([40.6269, -73.9191]), label=2.
el=0.0, prediction=4)Row(features=DenseVector([40.6288, -73.9414]), label=0.0, prediction=4)Row(features=DenseVector([40.6293, -73.9959]), label=3.0, prediction=4)Row(
4)Row(features=DenseVector([40.6303, -73.8913]), label=2.0, prediction=4)Row(features=DenseVector([40.6306, -73.9777]), label=0.0, prediction=4)Row(features=DenseVecto
eVector([40.6318, -73.9455]), label=2.0, prediction=4)Row(features=DenseVector([40.6319, -73.9577]), label=2.0, prediction=4)Row(features=DenseVector([40.632, -74.0215
25]), label=2.0, prediction=4)Row(features=DenseVector([40.6331, -73.9242]), label=0.0, prediction=4)Row(features=DenseVector([40.6332, -73.9478]), label=2.0, predicti
prediction=4)Row(features=DenseVector([40.634, -74.0131]), label=2.0, prediction=4)Row(features=DenseVector([40.6341, -74.0134]), label=2.0, prediction=4)Row(features=
features=DenseVector([40.6352, -74.0273]), label=0.0, prediction=4)Row(features=DenseVector([40.6352, -74.0208]), label=1.0, prediction=4)Row(features=DenseVector([40.
ctor([40.6367, -73.9761]), label=1.0, prediction=4)Row(features=DenseVector([40.6367, -73.9726]), label=0.0, prediction=4)Row(features=DenseVector([40.6367, -73.9655])
74.0249]), label=0.0, prediction=4)Row(features=DenseVector([40.6377, -73.9638]), label=1.0, prediction=4)Row(features=DenseVector([40.6377, -73.9568]), label=0.0, pre
.0, prediction=4)Row(features=DenseVector([40.6387, -73.9687]), label=0.0, prediction=4)Row(features=DenseVector([40.6388, -74.0358]), label=1.0, prediction=4)Row(feat
ow(features=DenseVector([40.6395, -73.9682]), label=0.0, prediction=4)Row(features=DenseVector([40.6396, -73.9967]), label=0.0, prediction=4)Row(features=DenseVector([
[40.6409, -73.9744]), label=0.0, prediction=4)Row(features=DenseVector([40.6412, -73.9626]), label=1.0, prediction=4)Row(features=DenseVector([40.6412, -73.8958]), lab
85]), label=1.0, prediction=0)Row(features=DenseVector([40.6415, -73.7885]), label=2.0, prediction=0)Row(features=DenseVector([40.6415, -73.7882]), label=0.0, predicti
iction=0)Row(features=DenseVector([40.6416, -73.7888]), label=0.0, prediction=0)Row(features=DenseVector([40.6416, -73.7888]), label=1.0, prediction=0)Row(features=Den
tures=DenseVector([40.6417, -74.0002]), label=0.0, prediction=4)Row(features=DenseVector([40.6417, -74.0001]), label=1.0, prediction=4)Row(features=DenseVector([40.641
```

In the implementation, we split the input data into two parts – training data and test data, thus giving a touch to supervised machine learning. We then allow the algorithm to predict the clusters for the test data. The accuracy of the predictions is checked using a function available in Spark-

evaluator=MulticlassClassificationEvaluator(labelCol="label",predictionCol="predicted_cluster", metricName="accuracy")
accuracy_lg = evaluator.evaluate(prediction)

This gives the accuracy of the algorithm.

V. Conclusion and Future Work

In this paper, we discussed how to apply the clustering algorithm to large datasets using Spark in order to improve the time of execution by caching the intermediate results in RAM. We have worked on structured and semi-structured data sets so far. This algorithm can be extended to take inputs in the form of unstructured data sets like images and video, which requires the use of other software’s integrated with Spark like Tensorflow.

References

1. Madhavi Vaidya, ‘Parallel Processing of cluster by Map Reduce’, International Journal of Distributed and Parallel Systems (IJDPS) Vol.3, No.1, January 2012.
2. Jeffrey Dean and Sanjay Ghemawat, ‘MapReduce: Simplified Data Processing on Large Clusters’, Google, Inc.
3. Jyoti Yadav, Monika Sharma, ‘A Review of K-mean Algorithm’, International Journal of Engineering Trends and Technology (IJETT) – Volume 4 Issue 7- July 2013.
4. V Srinivas Jonnalagadda, P Srikanth, Krishnamachari Thumati, Sri Hari Nallamala, ‘A Review Study of Apache Spark in Big Data Processing’, International Journal of Computer Science Trends and Technology (IJCST) – Volume 4 Issue 3, May - Jun 2016
5. Abhishek Bhattachary, Shefali Bhatnagar, ‘Big Data and Apache Spark: A Review’, International Journal of Engineering Research & Science (IJOER), Vol-2, Issue-5 May- 2016

DOI:10.15680/IJIRCCE.2017. 0505263