

Big Data Based Machine Learning and Predictive Analytics using Apache Mahout and Storm

Oday A. Hassen

Computer Science and Information Technology

University of Wasit, Iraq

Abstract

Machine learning refers to the intelligent and dynamic response by the software or embedded hardware programs depending upon the input data. Machine learning is the specialized domain that operates in association with the artificial intelligence to have strong predictions and analysis. Using this approach, there is no need to explicitly program the computers for specific applications rather the computing modules evaluates the dataset with its inherent behavior so that real time fuzzy based analysis can be done. The programs developed with machine learning paradigms focuses on the dynamic input and dataset so that the custom and related output can be presented to the end user. Big Data Analytics is one of the key areas of research with assorted approaches in data science and predictive analysis. A number of scenarios exist where enormous data is logged every day and needs deep evaluation for research and development. In Medical Science, there are enormous examples where processing, analysis and predictions from huge amount of data is required regularly. As per the reports from *First Post*, the data of more than 50 Peta Bytes are generated from each hospital of 500 beds in USA. In another research, it is found that one gram of DNA is equivalent to 215 petabytes in digital form. In another scenario of digital communication, the number of smart wearable gadgets increased from 26 millions in year 2014 to more than 100 millions in year 2016. This research manuscript underlines the scenarios and execution platforms for big data processing and machine learning.

Keywords:- *Big Data, Data Science, Machine Learning, Predictive Analytics*

Introduction:-

Big Data Processing [1], Machine Learning [2] and Predictive Analytics [3] are key domains of research because of enormous data in processing from assorted channels. A prominent neuroscientist *Ann-Shyn Chian* from Taiwan presented in a research that more than 1 GB per cell of brain will be required even for a very small creature on this earth. For imaging of more than 80 billion neurons in human brain, it will take around 17 million years. Now, the volume, velocity, variety of medical data can be imagined with these data figures.

Creature	No. of Neurons in Brain / Nervous System
Fly	1,35,000
Cockroach	1,000,000
Ant	2,50,000
Honey Bee	9,60,000
Cat	760,000,000
Monkey	3,246,000,000
Macaque	6,376,000,000
Human	86,000,000,000

Here, the key question comes on the evaluation of huge amount of data with enormously growing speed. To preprocess, analyze, evaluate and predict on such big data based applications, there is need to use high performance computing frameworks and libraries so that processing power of computers can be utilized with maximum throughput and performance.

A number of application domains exist where machine learning approaches are widely used including fingerprint analysis, multidimensional biometric evaluation, image forensic, pattern

recognition, criminal investigation, bioinformatics, Biomedical informatics, Computer vision, Customer relationship management, Data mining, Email filtering, Natural language processing, Automatic summarization, Automatic taxonomy construction, Robotics, Dialog system, Grammar checker, Language recognition, Handwriting recognition, Optical character recognition, Speech recognition, Machine translation, Question answering, Speech synthesis, Text simplification, Pattern recognition, Facial recognition system, Handwriting recognition, Image recognition, Search engine analytics, Recommendation system and many others.

Apache Mahout [4] and Storm [5] support powerful and performance aware real-time Distributed Computation System under Free and Open Source (FOSS) paradigm. The unbounded and free flowing data from multiple channels can be effectively logged and evaluated using Apache Storm with real time processing as compared to batch processing in Hadoop [6]. In addition, Storm is effectually adopted by number for corporate applications with the integration of any programming language without any issues of compatibility. The state of clusters and distributed environment is managed via Apache Zookeeper in the implementation of Apache Storm. The research based algorithms and predictive analytics can be executed in parallel using Apache Storm. MapReduce refers to a fault tolerant distributed high performance computational framework which is used to process and evaluate the huge amount of data. MapReduce like functions can be effectively implemented in Apache Storm using Bolts as the key logical operations are performed at the level of bolts. In many cases, the performance of Bolts in Apache Storm can be outperform MapReduce.

Apache Mahout: The Scalable High Performance Machine Learning Framework

Apache Mahout is the powerful and high performance machine learning framework for the implementation of machine learning algorithms. Apache Mahout is traditionally used for the integration of supervised machine learning algorithms with the target value assigned to each

input data set. Apache Mahout can be used for assorted research based applications including Social Media Extraction [7] and Sentiment Mining [8], User Belief Analytics, YouTube Analytics [9] and many related real time applications.

In Apache Mahout, a Mahout refers to the object which drives or operates the elephant. The mahout act as the master of elephant in association with Apache Hadoop and it is presented in the logo of elephant. Apache Mahout runs with the base installation of Apache Hadoop and then the machine learning algorithms are implemented with the features to develop and deploy the scalable machine learning algorithms. The prime approaches like recommender engines, classification problems and clustering can be effectively solved using mahout.

Corporate Users of Mahout includes the following

- Adobe
- Facebook
- LinkedIn
- Four Square
- Twitter
- Yahoo

Working with Apache Mahout:-

To start with the Mahout installation, first of all Apache Hadoop is required to be setup on the Linux Distribution. To get ready with Hadoop, the installation is required to be updated as follows in the Ubuntu Linux [10].

\$ sudo apt-get update

\$ sudo addgroup hadoop

\$ sudo adduser --ingroup hadoop hadoopuser1

```
$ sudo adduser hadoopuser1 sudo
$ sudo apt-get install ssh
$ su hadoopuser1
$ ssh-keygen -t rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
$ ssh localhost
```

Installing the Latest Version of Hadoop

```
$ wget http://www-us.apache.org/dist/hadoop/common/hadoop-HadoopVersion/hadoop-
HadoopVersion.tar.gz
$ tar xvfz hadoop-HadoopVersion.tar.gz
$ sudo mkdir -p /usr/local/hadoop
$ cd hadoop-HadoopVersion/
$ sudo mv * /usr/local/hadoop
$ sudo chown -R hadoopuser1:hadoop /usr/local/hadoop
```

The following files are required to be updated next

- ~/.bashrc
- core-site.xml
- hadoop-env.sh
- hdfs-site.xml
- mapred-site.xml
- yarn-site.xml

```
$ hadoop namenode -format
```

```
$ cd /usr/local/hadoop/sbin
```

```
$ start-all.sh
```

Web Interfaces of Hadoop

MapReduce: <http://localhost:8042/>

NameNode daemon: <http://localhost:50070/>

Resource Manager: <http://localhost:8088/>

SecondaryNameNode:: <http://localhost:50090/status.html>

The default port to access Hadoop is 50070 and using <http://localhost:50070/> on Web Browser

After installation of Hadoop, the setup of Mahout is required as follows.

```
$ wget http://mirror.nexcess.net/apache/mahout/0.9/mahout-Distribution.tar.gz
```

```
$ tar zxvf mahout-Distribution.tar.gz
```

Implementation of Recommender Engine Algorithm:-

Now days, we shop on the online shopping platforms like Amazon, E-Bay, SnapDeal, FlipKart and many others. We generally see that most of these online shopping platforms give us suggestions or recommendations about the products which we like or earlier purchased. This type of implementation or suggestive modeling is known as recommender engine or recommendation system. Even in YouTube, we see the number of suggestions regarding related videos which we view. Such online platforms integrate the approaches of recommendation engines by which the related best fit or most viewed items are presented to the user as recommendations.

Apache Mahout provides the platform to program and implement the recommender systems. For example, the Twitter HashTag Popularity can be evaluated and ranking can be done based on the visitor count or popularity or simply hits by the users. In YouTube, the number of viewers is the key value which determines the actual popularity of that particular video. Using Apache Mahout, such algorithms can be implemented which are covered under high performance real time machine learning.

For example, a data table which presents the popularity of products after online shopping by the users is recorded by the companies so that the overall analysis of popularity of products can be done. The rating from 0-5 is logged from the users so that the overall prominence of the product can be evaluated. This dataset can be evaluated using Apache Mahout in Eclipse IDE.

For integration of Java Code with Apache Mahout Libraries on Eclipse IDE, there are specific JAR files which are required to be added from Simple Logging Facade for Java (SLF4J).



SLF4J Project

- Introduction
- Download
- Documentation
- License
- News
- Support**
- Mailing Lists
- Bug Reporting
- Source Repository
- Support offerings
- Native implementations
- Logback
- Wrapped implementations
- JDK14
- Log4j
- Simple

Simple Logging Facade for Java (SLF4J)

The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at *deployment* time.

Before you start using SLF4J, we highly recommend that you read the two-page [SLF4J user manual](#).


Note that SLF4J-enabling your library implies the addition of only a single mandatory dependency, namely *slf4j-api.jar*. If no binding is found on the class path, then SLF4J will default to a no-operation implementation.

In case you wish to migrate your Java source files to SLF4J, consider our [migrator tool](#) which can migrate your project to use the SLF4J API in just a few minutes.

In case an externally-maintained component you depend on uses a logging API other than SLF4J, such as commons logging, log4j or java.util.logging, have a look at SLF4J's binary-support for [legacy APIs](#).

Copyright © 2004-2017 QOS.ch
 We are actively looking for volunteers to proofread the documentation. Please send your corrections or suggestions for improvement to "corrections@qos.ch". See also the [instructions for contributors](#).

Figure 1: Simple Logging Facade for Java



SLF4J Project

- Introduction
- Download
- Documentation
- License
- News
- Support**
- Mailing Lists
- Bug Reporting
- Source Repository
- Support offerings
- Native implementations
- Logback
- Wrapped implementations
- JDK14
- Log4j
- Simple

Latest STABLE version

Download version 1.7.25 including *full source code*, class files and documentation in ZIP or TAR.GZ format:

- [slf4j-1.7.25.tar.gz](#)
- [slf4j-1.7.25.zip](#)

Java 9 Modularized EXPERIMENTAL version

Download version 1.8.0-alpha2 including *full source code*, class files and documentation in ZIP or TAR.GZ format:

- [slf4j-1.8.0-alpha2.tar.gz](#)
- [slf4j-1.8.0-alpha2.zip](#)

Previous versions

Previous versions of SLF4J can be downloaded from the [main repository](#).

Figure 2: Stable JAR Files from SLF54J Portal

Following is the Java Code Module with the methods which can be executed using Eclipse IDE with the JAR files of Mahout to implement Recommender Algorithm

```
DataModel dm = new FileDataModel(new File("inputdata"));
UserSimilarity us = new PearsonCorrelationSimilarity(dm);
UserNeighborhood un = new ThresholdUserNeighborhood(ThresholdValue), us, dm);
UserBasedRecommender r=new GenericUserBasedRecommender(dm, un, us);
List<RecommendedItem> rs=recommender.recommend(UserID, Recommendations);
for (RecommendedItem rc : rs) {
    System.out.println(rc);
}
```

Installing Apache Storm and Zookeeper on MS Windows Environment

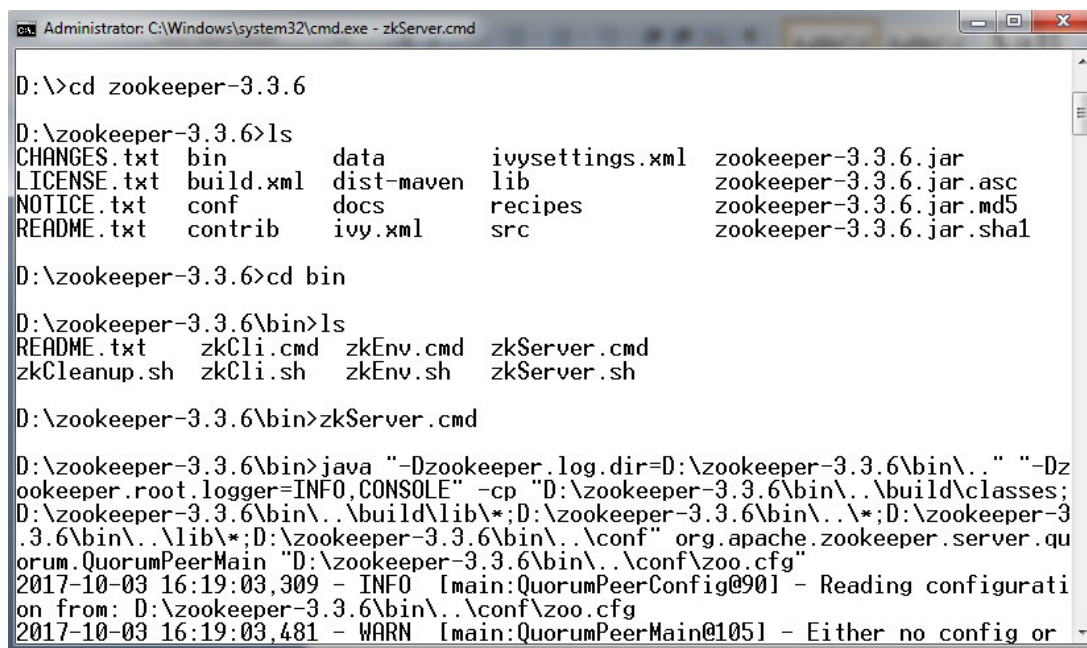
First of all, Apache Zookeeper is downloaded and installed from <https://zookeeper.apache.org/>.

Configure and run Zookeeper with the following commands:

```
MSWindowsDrive:\> cd zookeeper-Version
MSWindowsDrive:\ zookeeper-Version> copy conf\zoo_sample.cfg conf\zoo.cfg
MSWindowsDrive:\ zookeeper-Version> .\bin\zkServer.cmd
```

Following records are updated in zoo.cfg:

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir= MSWindowsDrive:/zookeeper-3.4.8/data
```



```

Administrator: C:\Windows\system32\cmd.exe - zkServer.cmd

D:\>cd zookeeper-3.3.6

D:\zookeeper-3.3.6>ls
CHANGES.txt  bin          data          ivysettings.xml  zookeeper-3.3.6.jar
LICENSE.txt   build.xml    dist-maven    lib              zookeeper-3.3.6.jar.asc
NOTICE.txt    conf         docs          recipes          zookeeper-3.3.6.jar.md5
README.txt    contrib      ivy.xml       src              zookeeper-3.3.6.jar.sha1

D:\zookeeper-3.3.6>cd bin

D:\zookeeper-3.3.6\bin>ls
README.txt    zkCli.cmd    zkEnv.cmd     zkServer.cmd
zkCleanup.sh  zkCli.sh     zkEnv.sh      zkServer.sh

D:\zookeeper-3.3.6\bin>zkServer.cmd

D:\zookeeper-3.3.6\bin>java "-Dzookeeper.log.dir=D:\zookeeper-3.3.6\bin\.." "-Dzookeeper.root.logger=INFO,CONSOLE" -cp "D:\zookeeper-3.3.6\bin\..\build\classes;D:\zookeeper-3.3.6\bin\..\build\lib\*;D:\zookeeper-3.3.6\bin\..\*;D:\zookeeper-3.3.6\bin\..\lib\*;D:\zookeeper-3.3.6\bin\..\conf" org.apache.zookeeper.server.quorum.QuorumPeerMain "D:\zookeeper-3.3.6\bin\..\conf\zoo.cfg"
2017-10-03 16:19:03,309 - INFO [main:QuorumPeerConfig@901] - Reading configuration from: D:\zookeeper-3.3.6\bin\..\conf\zoo.cfg
2017-10-03 16:19:03,481 - WARN [main:QuorumPeerMain@1051] - Either no config or

```

Figure 3: Execution of Commands for Initialization of Apache Zookeeper

Download and Install Apache Storm from <http://storm.apache.org/> and set STORM_HOME to *MSWindowsDrive:\apache-storm-Version* following in environment variables.

Perform the modifications in storm.yaml as follows:

```

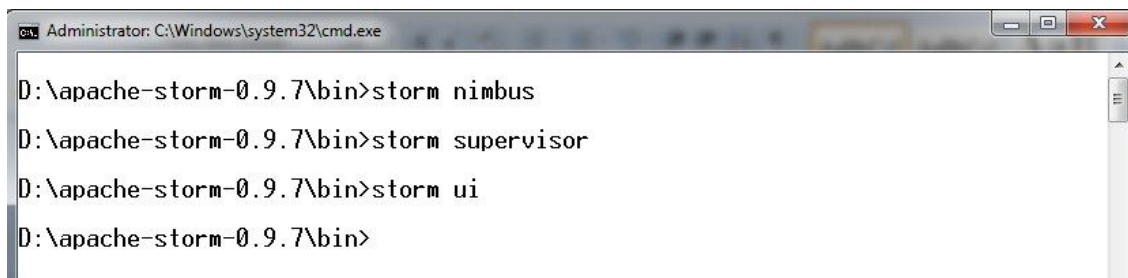
storm.zookeeper.servers:
- "127.0.0.1"

nimbus.host: "127.0.0.1"

storm.local.dir: "D:/storm/datadir/storm"

supervisor.slots.ports:
- 6700
- 6701
- 6702
- 6703

```



```

Administrator: C:\Windows\system32\cmd.exe

D:\apache-storm-0.9.7\bin>storm nimbus
D:\apache-storm-0.9.7\bin>storm supervisor
D:\apache-storm-0.9.7\bin>storm ui
D:\apache-storm-0.9.7\bin>
  
```

Figure 4: Execution of Commands for Initialization of Apache Storm

In any Web Browser, Execute the URL <http://localhost:8080> to confirm the working of Apache Storm

Storm UI

Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
0.9.7	2m 25s	1	0	10	10	0	0

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
------	----	--------	--------	-------------	---------------	-----------

Supervisor summary

Id	Host	Uptime	Slots	Used slots
6e9a5dfd-4c32-4a14-8da3-cc1e47f98c80	kumargaurav-PC	2m 10s	10	0

Nimbus Configuration

Key	Value
dev.zookeeper.path	/tmp/dev-storm-zookeeper
drpc.childopts	-Xmx768m

Figure 5: Apache Storm UI with the Base Configurations, Nodes and Cluster Information

Apache Storm is having association with number of key components and modules which work together to perform the high performance computing. These components include

Nimbus Node, Supervisor Node, Worker Process, Executor, Task and many others. Following is the brief description of key components which are used in the implementation of Apache Storm.

Components	Description
Nimbus	Master Node in the Cluster of Apache Storm. The other nodes are referred as Worker Nodes. Master Node is the key node used for the distribution of data to the worker nodes and overall monitoring
Supervisor	The nodes which accept the instructions sent by Nimbus node. Supervisor Node is having multiple processes mapped with the workers
Worker process	Execution of tasks associated with particular topology. It creates the executors to execute the tasks
Executor	Thread created to execute the process
Task	The specific task to be performed
Zookeeper Framework	A Service that is used by a group of nodes or simply clusters. Zookeeper assist the supervisor to communicate with Nimbus and maintain the states of communication
Tuple	Key data structure in Apache Storm

	supporting all formats and data types
Stream	The sequence of tuples in unordered perspectives
Spout	Used to accept the input data from different channels like Twitter Streaming, Bioinformatics, Medical, Satellite Data and many others
Bolt	These are Logical Processing Units in Apache Storm

Extraction and Analytics of Twitter Social Media using Apache Storm

To extract the live data from Twitter Social Media, the APIs of Twitter4j are used which provides the programming interface to connect with Twitter Servers. In Eclipse IDE, the code of Java can be programmed for predictive analysis and evaluation of the tweets fetched from real time streaming channels. As social media mining is one of the key segment of research for extraction and prediction of popularity, the following code snippets are used to extract the real time streaming and evaluation of user sentiments.

MyTwitterSpout.java

```
public class MyTwitterSpout extends BaseRichSpout {
    SpoutOutputCollector _collector;
    LinkedBlockingQueue<Status> queue = null;
    TwitterStream _twitterStream;
    String myconsumerKey;
    String myconsumerSecret;
    String myaccessToken;
```

```
String myaccessTokenSecret;
String[] mykeyWords;
public MyTwitterSpout(String myconsumerKey, String myconsumerSecret,
String myaccessToken, String myaccessTokenSecret, String[] mykeyWords) {
    this.myconsumerKey = myconsumerKey;
    this.myconsumerSecret = myconsumerSecret;
    this.myaccessToken = myaccessToken;
    this.myaccessTokenSecret = myaccessTokenSecret;
    this.mykeyWords = mykeyWords;
}
public MyTwitterSpout() { }
@Override
public void open(Map conf, TopologyContext context,
SpoutOutputCollector collector) {
    queue = new LinkedBlockingQueue<Status>(1000);
    _collector = collector;
    MyStatusListener listener = new MyStatusListener() {
        @Override
        public void onStatus(Status MyStatus) {
            queue.offer(status);
        }
    };
    @Override
    public void onDeletionNotice(StatusDeletionNotice sdn) {}
    @Override
    public void onTrackLimitationNotice(int i) {}
    @Override
    public void onScrubGeo(long l, long ll) {}
```

```
@Override
public void onException(Exception ex) {}

@Override
public void onStallWarning(StallWarning arg0) {
}

ConfigurationBuilder MyCB = new ConfigurationBuilder();
MyCB.setDebugEnabled(true)
.setOAuthMyconsumerKey(myconsumerKey)
.setOAuthMyconsumerSecret(myconsumerSecret)
.setOAuthMyaccessToken(myaccessToken)
.setOAuthMyaccessTokenSecret(myaccessTokenSecret);
_twitterStream = new TwitterStreamFactory(cb.build()).getInstance();
_twitterStream.addListener(listener);
if (mykeyWords.length == 0) {
    _twitterStream.sample();
} else {
    FilterQuery MyQuery = new FilterQuery().track(mykeyWords);
    _twitterStream.filter(query);
} }

@Override
public void nextTuple() {
    MyStatus ret = queue.poll();

    if (ret == null) {
        Utils.sleep(50);
    } else {
        _collector.emit(new Values(ret));
    }
}
```

```
    } }  
    @Override  
    public void close() {  
        _twitterStream.shutdown();  
    }  
    @Override  
    public Map<String, Object> getComponentConfiguration() {  
        Config ret = new Config();  
        ret.setMaxTaskParallelism(1);  
        return ret;  
    }  
    @Override  
    public void ack(Object id) {}  
    @Override  
    public void fail(Object id) {}  
    @Override  
    public void declareOutputFields(OutputFieldsDeclarer declarer) {  
        declarer.declare(new Fields("tweet"));  
    }  
}
```

MyHashtagBolt.java

```
public class MyHashtagBolt implements IRichBolt {  
    private OutputCollector collector;  
    @Override  
    public void prepare(Map conf, TopologyContext context, OutputCollector collector) {  
        this.collector = collector;  
    }  
}
```


@Override

```

public void execute(Tuple tuple) {
    Status tweet = (Status) tuple.getValueByField("tweet");
    for(HashtagEntity hashtag : tweet.getHashtagEntities()) {
        System.out.println("Hashtag: " + hashtag.getText());
        this.collector.emit(new Values(hashtag.getText()));
    } }

```

@Override

```

public void cleanup() {}

```

@Override

```

public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("hashtag"));
}

```

@Override

```

public Map<String, Object> getComponentConfiguration() {
    return null;
} }

```

MyBolt.java

```

public class MyBolt implements IRichBolt {
    Map<String, Integer> counterMap;
    private OutputCollector collector;
    @Override
    public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
        this.counterMap = new HashMap<String, Integer>();
        this.collector = collector;
    }
}

```

```
@Override
public void execute(Tuple tuple) {
    String key = tuple.getString(0);
    if(!counterMap.containsKey(key)){
        counterMap.put(key, 1);
    }else{
        Integer c = counterMap.get(key) + 1;
        counterMap.put(key, c);
    }
    collector.ack(tuple);
}

@Override
public void cleanup() {
    for(Map.Entry<String, Integer> entry:counterMap.entrySet()){
        System.out.println("Result: " + entry.getKey()+" : " + entry.getValue());
    } }

@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("hashtag"));
}

@Override
public Map<String, Object> getComponentConfiguration() {
    return null;
} }
```

MyApacheStorm.java

```
public class MyApacheStorm {
```

```

public static void main(String[] args) throws Exception{
    String myconsumerKey = args[0];
    String myconsumerSecret = args[1];
    String myaccessToken = args[2];
    String myaccessTokenSecret = args[3];
    String[] MyArguments = args.clone();
    String[] mykeyWords = Arrays.copyOfRange(arguments, 4, MyArguments.length);
    MyConfig MyConfig = new MyConfig();
    MyConfig.setDebug(true);
    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout("twitter-spout", new MyTwitterSpout(myconsumerKey,
        myconsumerSecret, myaccessToken, myaccessTokenSecret, mykeyWords));
    builder.setBolt("twitter-hashtag-reader-bolt", new MyHashtagBolt())
        .shuffleGrouping("twitter-spout");
    builder.setBolt("twitter-hashtag-counter-bolt", new MyBolt())
        .fieldsGrouping("twitter-hashtag-reader-bolt", new Fields("hashtag"));
    LocalCluster MyCluster = new LocalCluster();
    MyCluster.submitTopology("MyApacheStorm", MyConfig,
        builder.createTopology());
    Thread.sleep(10000);
    MyCluster.shutdown();
} }

```

Conclusion:-

The research problems can be solved effectively using Apache Mahout with the customized algorithms in multiple applications including Malware Predictive Analytics, User Sentiment Mining, Rainfall Predictions, Network Forensic and Network Routing with deep analytics.

Now days, the integration of deep learning approaches can be embedded in the existing algorithms so that higher degree of accuracy and optimization in the results can be achieved. The extraction of datasets from live satellite channels and cloud delivery points can be implemented with the integrated approach of Apache Storm to have effectual predictions on specific paradigms. As an example, the live streaming data of longitude and latitude from smart gadget with the deep learning based approach can be implemented to predict the upcoming position of specific person. In bioinformatics and medical sciences, the probability of a specific disease in a person can be predicted with the neural network based learning of historical medical records and health parameters using Apache Storm. Besides, these there are enormous domains and areas where big data analytics including Banking Datasets, Rainfall Predictions and many others can be done with the effectual results having contribution to the social cause.

References:-

- [1] Provost F, Fawcett T. Data science and its relationship to big data and data-driven decision making. *Big Data*. 2013 Mar 1;1(1):51-9.
- [2] Carbonell JG, Michalski RS, Mitchell TM. An overview of machine learning. In *Machine learning 1983* (pp. 3-23). Springer Berlin Heidelberg.
- [3] Burbidge R, Trotter M, Buxton B, Holden S. Drug design by machine learning: support vector machines for pharmaceutical data analysis. *Computers & chemistry*. 2001 Dec 31;26(1):5-14.
- [4] Schelter S, Owen S. Collaborative filtering with apache mahout. *Proc. of ACM RecSys Challenge*. 2012.
- [5] Iqbal MH, Soomro TR. Big data analysis: Apache storm perspective. *International journal of computer trends and technology*. 2015 Jan:9-14.
- [6] O'Malley O. Terabyte sort on apache hadoop. Yahoo, available online at: <http://sortbenchmark.org/Yahoo-Hadoop.pdf>, (May). 2008 May:1-3.

- [7] Imran M, Elbassuoni S, Castillo C, Diaz F, Meier P. Practical extraction of disaster-relevant information from social media. In Proceedings of the 22nd International Conference on World Wide Web 2013 May 13 (pp. 1021-1024). ACM.
- [8] Go A, Huang L, Bhayani R. Twitter sentiment analysis. Entropy. 2009 Jun 6;17:252.
- [9] Atwong CT. A social media practicum: An action-learning approach to social media marketing and analytics. Marketing Education Review. 2015 Jan 2;25(1):27-31.
- [10] Noll MG. Running hadoop on ubuntu linux (single-node cluster). Mar-2013.[Online]. Available: <http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-nodecluster/>. 2007.

ABOUT THE AUTHOR



Oday A. Hassen received the B.Sc. degree in mathematics from the Department of Mathematics, University of Mustansiriyah, Iraq in 2004, and M.Sc from department of information technology, science college, Alexandria University, Egypt. Currently he is PHD student in university Technical Melaka (UTeM), Malaysia. His research and professional interests include image processing, cryptography, authentication and security technologies.