CSP 554 BIG DATA TECHNOLOGIES

Project Report

Shouvik Sharma
Rahul Nair
Cheng Jiang
Yasasvi Gude

Table of Contents

List of Figures

Abstract

These are the days of innovation for a better future and companies are bounded to realize and accept the necessity of Big Data to solve complex and challenging problems through better decision making. The term Big Data refers to a collection of large datasets with a size of data present in petabytes and more, having high growth, high complexity making the process of analyzing through traditional database technologies complicated. This project talks about the different techniques that are available for processing big data and comparing those technologies in terms of efficiency, computing power, ease of use and implementation, and building predictive modeling.

Keywords: Big Data, Hadoop, Hive, Pig, Spark, Google Big Query

1. Introduction

With the increase of computing devices like smartphones, tablets, laptops, sensors, etc. there is also an increase in the size of data or raw information generated. Internet technology has been rapidly growing, with many users being connected to the internet every day. Till 20th century Big Data was referred up to megabytes or gigabytes, but today, this size has grown exponentially to terabytes; and slowing moving towards zettabytes. It is estimated that the internet generates nearly 500 Billion Gigabytes every day, which consists mostly of unstructured data (images, audio and video clips, logs, etc.) Earlier RDBMS were easy to use as the data stored in the database were in a proper format, or it can be said that they were structured data. For most companies, this data is a money-making machine if utilized carefully in a well-mannered way. There are many challenges faced while working with big data like, if they are unstructured or semi-structured, then how to convert it into structured data, data cleansing should be fast, and performing analytics on the data should be fast [4][12]. Another big challenge is the storage, RDBMS also looks for schema for the tables stored, but this is not the case for data generated. Alternative technologies are introduced to overcome the challenges of storage, processing, cleaning, and performing analytics on them.

2. Literature Review

Big Data refers to the ways to analyze data, systematic extraction of information, and deal with the dataset that is too large to deal with using the traditional software. The term was first referred in the year 2005 by Roger Mougalas from O'Reilly Media. It is referred to as a large dataset that was merely impossible to manage and process. The measurement of the big data is not fixed to one size; in some cases, 2TB can be big data, and in another case, 200 TB can also be big data. In other words, "Big Data is a circumstance where the volume, velocity, and variety of data go beyond an organization's storage or computation capacity for precise and well-timed decision making" [4]. Big Data is characterized mainly into three terms of Vs. [2][12].

Volume: it refers to the sheer size of the data. These datasets can be orders of magnitude larger than the traditional datasets. With an increase in the growth of social media, the data generated is also growing exponentially. The data generated through machines exceeds the data generated by humans.

Velocity: Velocity is the speed at which the data is generated and moves through the system. Data is frequently flowing into the system from multiple sources and is often expected to be processed in real-time. With a focus on instant feedback and instant solutions has made the developers shift from batch oriented-approach to real-time streaming system.

Variety: Variety refers to the format in which the data is generated. Be it structured, unstructured, or semi-structured data, but 70% of the data generated is unstructured data. In traditional days the information was structured like spreadsheets, databases, flat files, etc., and nowadays the share of structured data is too low, the unstructured data that today is generated are in the form of video files, images, weblogs, sensor data, audio clips, etc. [4][12].

Even though RDBMS being the most preferred tool in IT, but it failed when it comes to Big Data. One of the reasons that it failed for Big Data was that RDBMS could not handle outsized data with a variety.

RDBMS follows a very strict schema with lots of constraints for the data. As a fact, it is known that most of the data in Big Data are in an unstructured format; it will always be challenging to have a schema. And maintaining relationships for unstructured data (video, weblogs, images, audio clips, etc.) is almost impossible. For analyzing a small dataset, time taken to process can be neglected, but for extensive datasets, time is a significant factor. Big data should possess fast processing speed like real-time insights, which RDBMS can't. Processing Big Data with traditional methods would not be cost-effective and a time-consuming process; to overcome these drawbacks, new technology was introduced called Hadoop [2][4][12].

Hadoop has subprojects like Hive, Pig, Spark Kafka, HBase, Oozie, which are an excellent option for Big Data Analytics. Most of the Big Data technologies are open-source software and can be used by anyone; some vendors, on the other hand, enhance this software to a better version with paid services. Hadoop is written in Java and can run on commodity hardware, scaling up from a single node to thousands of computers, thus creating a massive cluster.

2.1 Apache Pig

In 2006, Apache Pig was developed by Yahoo's research team. Apache Pig helps in processing large datasets. The programmers will use the Pig Latin Language to process the data which is stored in the HDFS. Internally, Pig Engine converts all these scripts into a specific map and reduce task. Pig Latin and Pig Engine are the two main components of the Apache Pig tool, which outputs the required results that are always stored in the HDFS. It is an interactive execution environment which uses PigLatin, unlike in Hive, relations are expressed as data flows.The flow in Pig starts with checking the syntax of the script and gives an output in the form of a DAG (Directed Acyclic Graph). Then, DAG is passed to the logical optimizer with the help of parser. It carries out optimizations such as projections and pushdowns. It is then transferred to the compiler, which compiles the optimized DAG into a series of Map-Reduce jobs, which then gives the final output once map-reduce jobs are run [2]. Pig can be run in three different ways; all of them are compatible with local and Hadoop:

Script: Simply a file containing Pig Latin commands, identified by the .pig suffix. Pig interpret the commands and executes in sequential order [16].
Grunt: Grunt is a command interpreter. It can be typed in Pig Latin on the grunt command line, and Grunt will execute the command. It is beneficial for prototyping and "what if" scenarios.

Embedded: Pig programs can be executed as part of a java program.

Workflow

Parser: It checks the script for syntax. The output of this component will a DAG – Directed Acyclic Graph, representing PigLatin statements [17].

Optimizer: The DAG made by the parser is passed to optimizer which carries out logical optimization like projection and pushdown.

Compiler: It compiles the optimized plan into MapReduce jobs.

Execution Engine: In the final stage, the MapReduce jobs are submitted to Hadoop in a sorted manner and are then executed to derive the desired output.

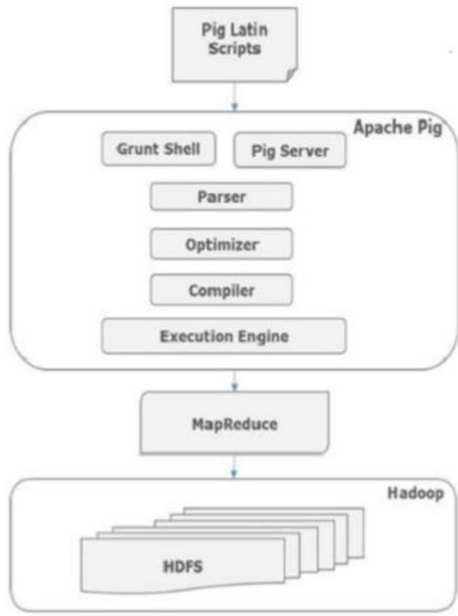


Figure 1 Apache Pig Workflow [17]

2.2 Apache Hive

Apache Hive is an open-source data warehouse tool which is useful for analyzing large data sets. Apache Hive uses query language called Hive query language, which supports ACID properties in HiveQL, with SQL commands like the update, insert, and delete. Hive query written in the command-line interface is delivered to the driver. The driver creates a session of the handle and then transfers the question to the compiler. The compiler assigns the metadata request to the database and extracts the required information. The compiler finally prepares an execution plan and shares it with the driver; subsequently, the result is transferred to the execution engine [2].

- Authors of the given research paper compare the efficiency of the MySQL server, Apache Hive, and Apache Pig. They have derived their conclusion based on the query statements and the average query time. They have used three datasets for their analysis: m1100k (movie lens 100,000 rows), m11m containing a total of 1,075,611 rows, and m110m containing a total of 10,069,372 rows [3].
- Hive has an advantage, such as indexing, which leads to faster file reading [3]. Hive invokes MapReduce only if the query has aggregation, join, or sorting function, which can take one until six seconds to start the MapReduce.
- Pig executes using a step-by-step approach. Pig works well when a query has a sophisticated type of function and many joins in the data, Pig can handle it efficiently by simultaneously executing each step and the subsequent next step. This approach does not work well in a query that has minimum joins and filters, as it can consume more time [3].

- Hive is capable of handling extensive data and is faster than MySQL. Pig is not suitable for such a data set. Pig is more ideal for more complex queries and more massive data sets [3].

Hive Components

Metastore: A repository for metadata of Hive. It consists of information like data location, schema along with metadata of partition [15].

Compiler: User for compiling Hive queries into map-reduce jobs and run them.

Driver: A controller mainly responsible for storing the generated metadata while executing HQL statements.

Optimizer: It splits tasks during the execution of map-reduce jobs, thus helping in scalability and efficiency.

Hive Shell: A terminal user for interacting with Hive to run Hive queries and is not case sensitive.

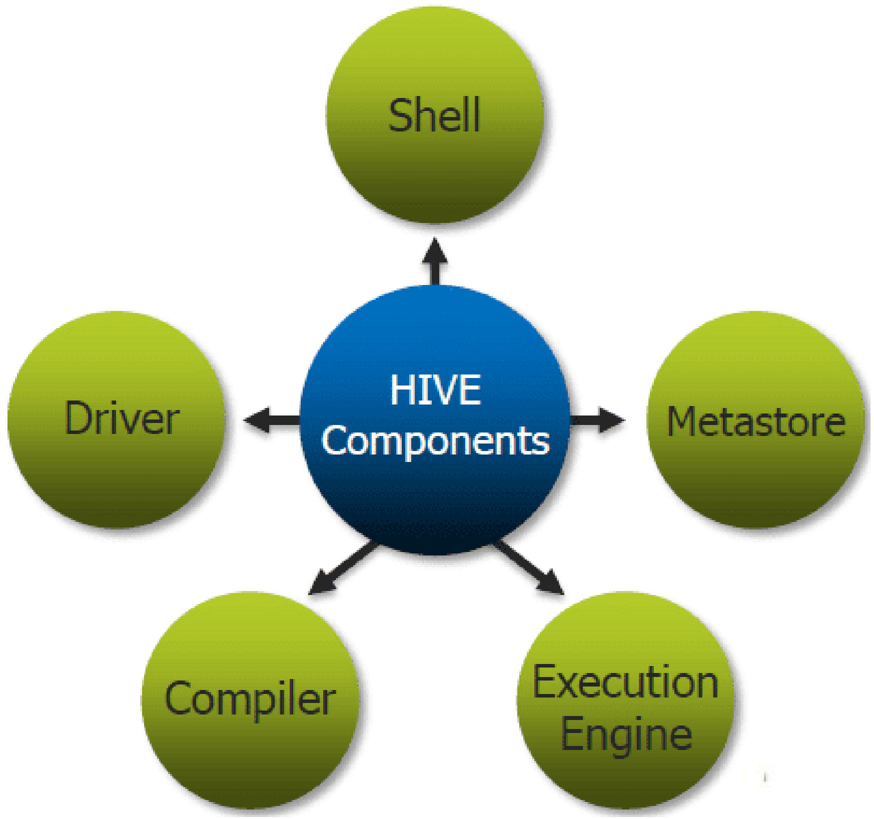


Figure 2 Hive Components [14]

2.3 Apache Spark(PySpark)

Apache spark is a distributed general-purpose cluster computing network, which has an interface for entire programming clusters with implicit data parallelism and fault tolerance. Spark has RDD – Resilient Distributed Dataset as an architectural foundation, which is a read-only multiset of data items distributed over a cluster of machines in a fault-tolerant manner. The spark came in as an alternative for map-reduces' limitations, a cluster computing paradigm that forces a linear dataflow structure on distributed programs. Apache Spark has a different approach; it implements both iterative algorithms, which visit data set multiple times in a loop and interactive data analysis which is the repetition of database styled query of data. Spark Core is the overall foundation of the project "Apache Spark," it provides distributed task dispatching, scheduling, and basic input/output functionalities through an interface (Python, Scala R, etc.) centered on the RDD abstraction. In previous map-reduce jobs, the workloads required separate engines, including SQL, streaming, graph processing, machine learning, but RDD in Spark handles all these workloads and work as a single-engine for all requirements [11][13]. These implementations use the same optimizations as specialized engines like column-oriented processing and incremental updates; and achieve similar performance but runs libraries over a common engine, making it easy and efficient to compose. A few of the benefits that Spark gives are that applications are more comfortable to develop as they use a unified API; secondly it is more suited to combine processing tasks. Third, Spark can run diverse functions over the same data, often in memory. As parallel data processing is becoming common, the composability of processing functions is also becoming an essential concern in terms of usability and performance. RDD is lazily evaluated by Spark to find an efficient plan for user computation.

When an action is called, Spark looks at the whole graph of transformations used to create an execution plan. For example, consider if there were multiple *filters* or *map* operations in a row, Spark fuses them into one pass, or it is known to spark in technical language as data is partitioned [13]. It avoids it over the network for *groupby*. Thus, users can build programs modularly without losing performance.

One of the options for performing Big Data Analytics in Spark is PySpark. PySpark is the combination of Apache Spark and Python. Deep learning has been in the limelight for quite time in market. Challenges in Deep Learning is exponentially rising as use of Deep Learning is taking gigantic leaps in numerous real business scenarios. It can’t be denied that many corporations are dependent heavily on deep learning like image language translation, self-driving cars to drone deliveries. Google is one of the few companies who have completely engulfed Deep Learning in day-to-day operations (Gmail, YouTube, Maps, Chrome, Google Assistance, Google Translation etc.). PySpark is a Python based API for Spark which enables user to user the functionalities of Spark with the power of Python libraries. Data in PySpark can be stored and accessed using RDD (Resilient Distributed Dataset) and Spark DataFrame. RDD are bases of Spark, it is fault tolerant and the data can be distributed among various nodes in a cluster. On the other hand, unlike Pandas DataFrame, Spark DataFrame is a distributed collection of structured and semi-structured data. Its functionalities are analogous to relational database tables. It can either be loaded from existing RDD or creating a new schema [18].

PySpark Features

Speed: Its nearly 100 times faster than the traditional large-scale data processing [18].

Powerful Caching: Simple programming layer in PySpark provides powerful caching and disk persistence capabilities.

Deployment: PySpark can be deployed through Mesos, Hadoop via Yarn, or Spark's own cluster manager.

Real-Time: Real-time computation and low latency because on in-memory computation.

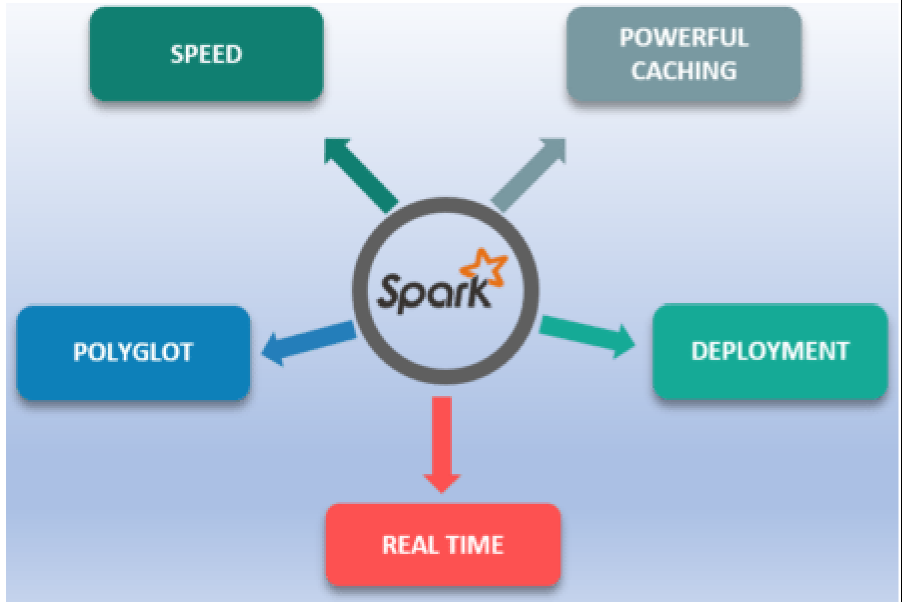


Figure 3 PySpark Features [19]

Polyglot: Supports programming in Scala, Java, Python, and R [18].

2.4 Google Cloud Platform (Big Query)

Big Query is Google's data warehouse which is managed at petabyte scale and at a very low cost. Big Query follows NoOps structure – which means that there is no infrastructure to manage and one doesn't need a database administrator. It follows SQL-querying system for fetching datasets. The main feature of the Google Big Query are as follows:

- Managing Data – Data can be pulled in the csv or json format.
- Query – The queries for extracting the datasets are expressed in the SQL dialect.
- Integration – The Big Query can be used from the Google app script.
- Access Control – The datasets can be shared with other users.

3. Data Description

The dataset has been taken from Google's Big Query, a platform to pull large datasets from a serverless warehouse. Stack Overflow is an online question and answer website for programmers. It started in fall 2018 and proved to be very helpful for developers and users across the world. This brought a rapid popularity among users; each user can ask and answer questions, they can collaboratively tag and edit questions, vote on the nature of the answers, and post comments on other individual's questions and answers.

The dataset contains 16 tables of which 9 were pulled from BigQuery in CSV format. Each CSV file has records of 200k rows in it, with variation in the number of columns for the data. There are 20 tags in total in the tag file. Tag file is used to identify any tags in a post either be a question or an answer, tag helps in classifying the nature of the post. Files such as answers.csv, questions.csv, and history.csv contain body, title, view counts, creation date, last access date, user ID, post ID, answer count, accepted answer count etc.

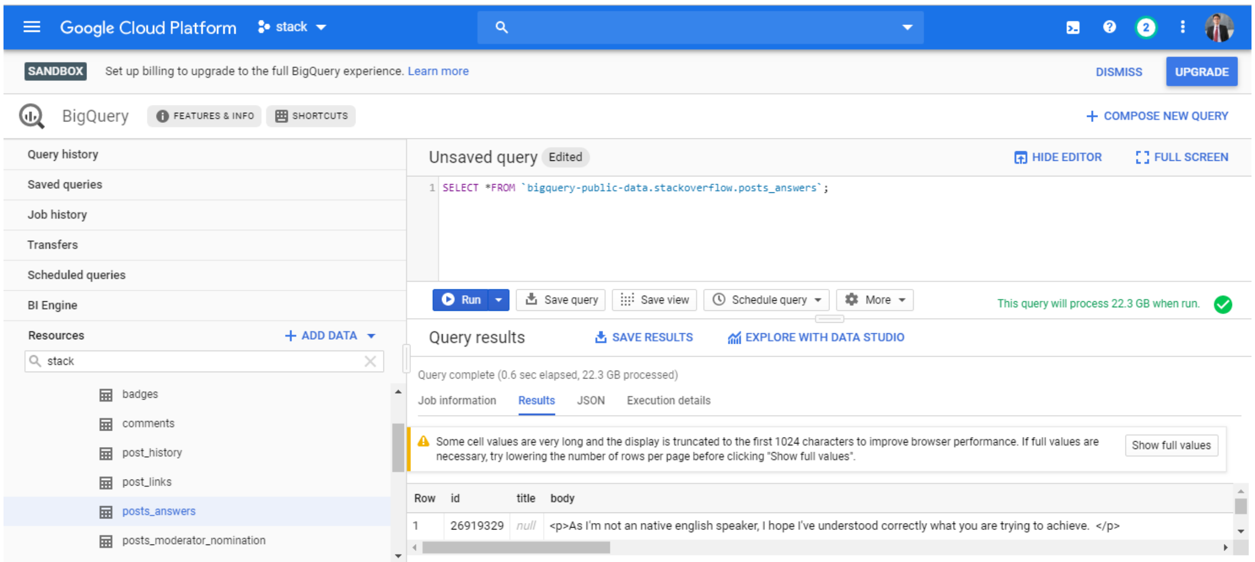


Figure 4 Google BigQuery

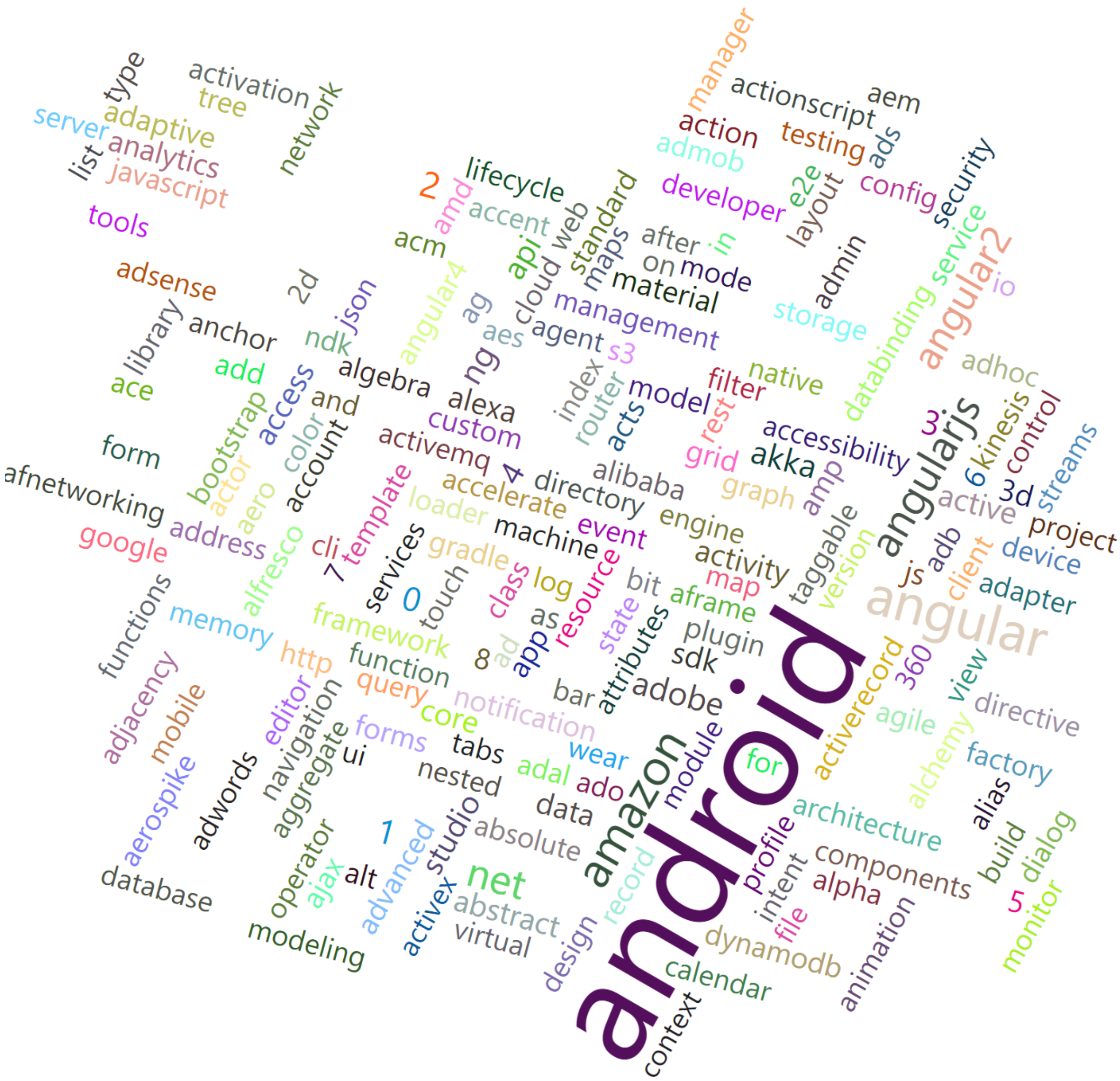


Figure 5 Word Cloud of Tags

4. System Requirements

OS : Windows/Ubuntu/Mac

Cloud Platform: Azure DevOps.

Memory: Enough memory to store data upto 2GB

GPU : A2 – A4 VM size

Technical Skills : Good knowledge of Linux, Hive, Pig and Spark.

5. Methodology

As mentioned previously, the project is about using different Big Data Technologies like Hive and Pig for analytics and Predictive Model for text classification using PySpark. Expected outcome of using these technologies is to understand which technology gives better results considering time, GPU and dataset size.

5.1 Query 1: Most Viewed Users

Writing query for top 100 most viewed users in the user dataset, column referred in this dataset was views.

5.1.1 Hive

```
hive> select A.displayName, A.views from project.user_data A
    > order by views desc limit 100;
Query ID = maria_dev_20191125084707_13cf3537-bb23-4d65-93da-7013613ec4b5
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1574613299864
_0020)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 10.32 s
--------------------------------------------------------------------------------
OK
```

Figure 6 Hive Query

5.1.2 Pig

```
maria_dev@sandbox-hdp:~
grunt> loaded = LOAD 'user.csv' Using PigStorage(',') AS (id:int,displayName:cha
rarray,creation_date:chararray, last_access_date:chararray,reputation:int,up_vot
es:int, down_votes:int, views:int);
grunt> generated = FOREACH loaded GENERATE views, displayName;
grunt> ordered = ORDER generated BY views DESC;
grunt> limited = LIMIT ordered 100
>> ;
grunt> dump limited;
```

Figure 7 Pig Query

5.2 Query 2: Most Valuable Users

Writing query for top 100 most valuable users in the user dataset, column referred in this was reputation which signifies the repute of a user.

5.2.1 Hive

```
maria_dev@sandbox-hdp:~
NULL
Time taken: 0.255 seconds, Fetched: 141312 row(s)
hive> SELECT A.DisplayName, A.Reputation FROM
    > (SELECT * FROM project.user_data ORDER BY reputation desc LIMIT 100) A;
Query ID = maria_dev_20191125085204_8239d5bb-3ddd-4750-847d-a56c0c4c3659
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1574613299864
_0021)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 9.98 s
--------------------------------------------------------------------------------
OK
```

Figure 8 Hive Query

5.2.2 Pig

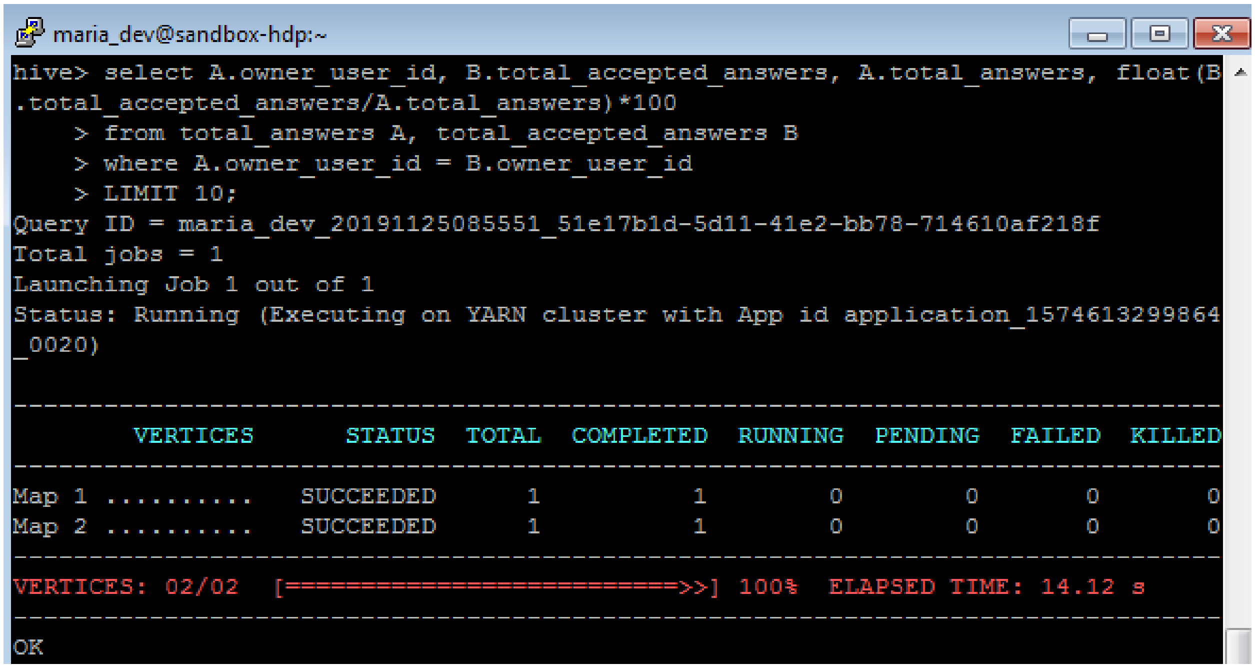
```
grunt> file = LOAD 'user.csv' Using PigStorage(',') AS (ID:int, DisplayName:char
array, CreationDate:chararray, lastAccessDate:chararray, Reputation:int, upvote:
int, downvote:int, views:int);
grunt> generated = FOREACH file GENERATE Reputation, DisplayName;
grunt> ordered_value = ORDER generated BY Reputation DESC;
grunt> limited = LIMIT ordered_value 100;
grunt> dump limited;
```

Figure 9 Pig Query

5.3 Query 3: Accepted Answer Percentage

Writing query for percentage of accepted answer by joining post answers and post questions and then finding the ration between accepted answers and total answers.

5.3.1 Hive



```
maria_dev@sandbox-hdp:~
hive> select A.owner_user_id, B.total_accepted_answers, A.total_answers, float(B
.total_accepted_answers/A.total_answers)*100
    > from total_answers A, total_accepted_answers B
    > where A.owner_user_id = B.owner_user_id
    > LIMIT 10;
Query ID = maria_dev_20191125085551_51e17b1d-5d11-41e2-bb78-714610af218f
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1574613299864
_0020)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Map 2 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 14.12 s
--------------------------------------------------------------------------------
OK
```

Figure 10 Hive Query

5.3.2 Pig

maria_dev@sandbox-hdp:~

```
grunt> post_answers_file = LOAD 'post_answers_file_1.csv' Using PigStorage(',')
AS (id:int,
>> owner_user_id:int,
>> parent_id:chararray,
>> post_type_id:int,
>> score:int);
2019-11-25 09:33:45,534 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> grpd_post_answers  = group post_answers_file by owner_user_id;
2019-11-25 09:33:45,816 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> cnt_post_answers   = foreach grpd_post_answers generate group, COUNT(post
_answers_file.id);
2019-11-25 09:33:46,188 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> post_questions_file = LOAD 'post_questions_file_1.csv' Using PigStorage('
,') AS (id:int,
>> accepted_answer_id:int);
2019-11-25 09:33:46,537 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> join_accepted_answers = JOIN post_answers_file BY id, post_questions_file
 BY accepted_answer_id;
2019-11-25 09:33:46,983 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> grpd_accepted_answers  = group join_accepted_answers by owner_user_id;
2019-11-25 09:33:47,304 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> cnt_accepted_answers   = foreach grpd_accepted_answers generate group, CO
UNT(join_accepted_answers.post_answers_file::id);
2019-11-25 09:33:47,627 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> cnt_post_answers_1 = foreach grpd_post_answers generate group as owner_us
er_id, COUNT(post_answers_file.id) as post_ans;
2019-11-25 09:33:48,003 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> cnt_accepted_answers_1 = foreach grpd_accepted_answers generate group as
owner_user_id, COUNT(join_accepted_answers.post_answers_file::id) as post_acc_an
s;
2019-11-25 09:33:48,456 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - E
ncountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> join_percent_answers= join cnt_accepted_answers_1 by owner_user_id, cnt_p
ost_answers_1 by owner_user_id;
```

Figure 11 Pig Query

5.4 Query 4: Marking Spammers

Writing query for marking spammers by matching vote id from votes dataset id = 12 (as described by Stack Overflow community a spam) with user id in comments dataset and grouping into one table.

5.4.1 Hive

```
hive> select A.post_id, B.user_id from votes A , comments B
    > where A.post_id=B.id AND (A.vote_type_id==12 AND B.user_id!=0);
Query ID = maria_dev_20191125090030_bb3293fe-6df5-42ca-935b-e56f6271653c
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1574613299864
_0020)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Map 2              SUCCEEDED      0          0        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 01/02  [==========================>>] 100%  ELAPSED TIME: 9.45 s
----------------------------------------------------------------------------------------------
OK
Time taken: 11.036 seconds
hive>
```

Figure 12 Hive Query

5.4.2 Pig

```
maria_dev@sandbox-hdp:~
grunt> loaded_votes = LOAD 'votes_data.csv' Using PigStorage(',') AS (id:int,post_id:int, vote_type_id:int);
grunt> filtered_votes = FILTER loaded_votes BY vote_type_id ==12;
grunt> loaded_comments = LOAD 'data_comments.csv' Using PigStorage(',') AS (id:int, creation_date:chararray,post_id:int,user_id:int ,score:float);
grunt> filtered_comments = FILTER loaded_comments BY user_id!=0;
grunt> join_relation = JOIN filtered_votes BY post_id, filtered_comments BY id;
grunt> spammers = FOREACH join_relation GENERATE filtered_votes::post_id, filtered_comments::user_id;
grunt> dump spammers;
2019-11-25 09:39:39,401 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: HASH_JOIN,FILTER
2019-11-25 09:39:39,581 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-11-25 09:39:39,664 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, Const
antCalculator, GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, PartitionFilterOptimizer, PredicatePushdownOp
timizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
2019-11-25 09:39:39,732 [main] INFO  org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for loaded_votes: $0
2019-11-25 09:39:39,739 [main] INFO  org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for loaded_comments: $1, $2, $4
2019-11-25 09:39:39,829 [main] INFO  org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS Old Gen) of size 699400192 to monitor. collectio
nUsageThreshold = 489580128, usageThreshold = 489580128
2019-11-25 09:39:40,052 [main] INFO  org.apache.pig.backend.hadoop.executionengine.tez.TezLauncher - Tez staging directory is /tmp/maria_dev/staging and
resources directory is /tmp/temp-948637447
2019-11-25 09:39:40,165 [main] INFO  org.apache.pig.backend.hadoop.executionengine.tez.plan.TezCompiler - File concatenation threshold: 100 optimistic? f
alse
2019-11-25 09:39:40,392 [main] INFO  org.apache.pig.builtin.PigStorage - Using PigTextInputFormat
2019-11-25 09:39:40,506 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 57
2019-11-25 09:39:40,506 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 57
2019-11-25 09:39:40,555 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths (combined) to process : 3
2019-11-25 09:39:40,643 [main] INFO  org.apache.pig.builtin.PigStorage - Using PigTextInputFormat
2019-11-25 09:39:40,702 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 61
2019-11-25 09:39:40,702 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 61
2019-11-25 09:39:40,720 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths (combined) to process : 3
```

Figure 13 Pig Query

5.5 Tag Prediction

Building a tag prediction model using PySpark. The dataset used contains two attributes, posts and tags. The total number of tags present are 21, which will help in classifying the posts to their respective tags. The steps to build the model are as follows:

5.5.1 Step 1: Importing Libraries

Importing libraries related to PySpark

```
import pyspark
from pyspark.sql.types import *
import pandas as pd
from pyspark.sql import SQLContext
from bs4 import BeautifulSoup
from pyspark import keyword_only
from pyspark.ml import Transformer
from pyspark.ml.param.shared import HasInputCol, HasOutputCol
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression, OneVsRest, RandomForestClassifier
from pyspark.ml.feature import IDF, StringIndexer, StopWordsRemover, CountVectorizer, RegexTokenizer, IndexToString
import nltk
from nltk.corpus import stopwords
```

Figure 14 Library Import

5.5.2 Step 2: Importing Dataset

Reading the .csv file into PySpark DataFrame

```
# creating schema for the dataframe
qSchema = StructType([StructField('post', StringType(), True),
                      StructField('tags', StringType(), True)])
d = pd.read_csv("/Users/rahulnair/desktop/stack-overflow-data.csv")

sqlContext = SQLContext(sc)
sdf = sqlContext.createDataFrame(d, qSchema)

# filtering out null values
sdf = sdf.filter(sdf.tags.isNotNull())
print("the dataframe for the data: ", sdf)
```

Figure 15 Dataset Import

5.5.3 Step 3: Data Split

Data is split into train and test with ratio of 75:25.

```
# Splitting the data
(train, test) = sdf.randomSplit((0.75, 0.25), seed = 100)
```

Figure 16 Data Split

5.5.4 Step 4: Removing HTML Tags

A class is created called *BsTextExtractor* to remove html tags from the posts.

```
# For removing HTML tags
class BsTextExtractor(Transformer, HasInputCol, HasOutputCol):

    @keyword_only
    def __init__(self, inputCol=None, outputCol=None):
        super(BsTextExtractor, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    @keyword_only
    def setParams(self, inputCol=None, outputCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _transform(self, dataset):

        def f(s):
            cleaned_post = BeautifulSoup(s).text
            return cleaned_post

        t = StringType()
        out_col = self.getOutputCol()
        in_col = dataset[self.getInputCol()]
        return dataset.withColumn(out_col, udf(f, t)(in_col))
```

Figure 17 HTML Tag Removal

5.5.5 Step 5: List of Stopwords

Creating a list of English Stopwords to remove articles from the post.

```
# list of stopwords to be removed from the posts
StopWords = list(set(stopwords.words('english')))
```

Figure 18 Stopwords

5.5.6 Step 6: Define Variables

Defining variables to create pipeline in next step. These variables include *labelIndexer* for converting tag string values into *indexed labels*, *bs_text_extractor* for removing html tags from post, *RegexTokenizer* for tokenizing regular expressions in posts, *StopwordRemover* for removing articles from posts, *CountVectorizer* for creating matrix for the output from *StopwordRemover*, *idf* for calculating Inverse Document Frequency for *CountVectorizer*, using *rf* to build Random Forest model and lastly *idx_2_string* to print tag names in the output.

```
labelIndexer = StringIndexer(inputCol="tags", outputCol="label").fit(train)
bs_text_extractor = BsTextExtractor(inputCol="post", outputCol="untagged_post")
RegexTokenizer = RegexTokenizer(inputCol=bs_text_extractor.getOutputCol(), outputCol="words", pattern="[^0-9a-z#+_]+")
StopwordRemover = StopWordsRemover(inputCol=RegexTokenizer.getOutputCol(), outputCol="filtered_words").setStopWords(
    StopWords)
CountVectorizer = CountVectorizer(inputCol=StopwordRemover.getOutputCol(), outputCol="countFeatures", minDF=5)
idf = IDF(inputCol=CountVectorizer.getOutputCol(), outputCol="features")
rf = RandomForestClassifier(labelCol="label", featuresCol=idf.getOutputCol(), numTrees=100, maxDepth=4)
idx_2_string = IndexToString(inputCol="prediction", outputCol="predictedValue")
idx_2_string.setLabels(labelIndexer.labels)
```

Figure 19 Define Variables

5.5.7 Step 7: Pipeline

Building pipeline for the variables defined in step 6, so that all variables flow in a sequential manner.

```
# creating the pipeline
pipeline = Pipeline(stages=[
    labelIndexer,
    bs_text_extractor,
    RegexTokenizer,
    StopwordRemover,
    CountVectorizer,
    idf,
    rf,
    idx_2_string])

```

Figure 20 Pipeline

5.5.8 Step 8: Fit Model

Fitting the model on the train dataset.

```
# fitting the model
model = pipeline.fit(train)
```

Figure 21 Fitting Model

5.5.9 Step 9: Prediction Model

Performing prediction on test dataset using model from the previous step.

```
# performing the prediction
predictions = model.transform(test)
```

Figure 22 Model Prediction

5.5.10 Step 10: Model Evaluation

Evaluating the model

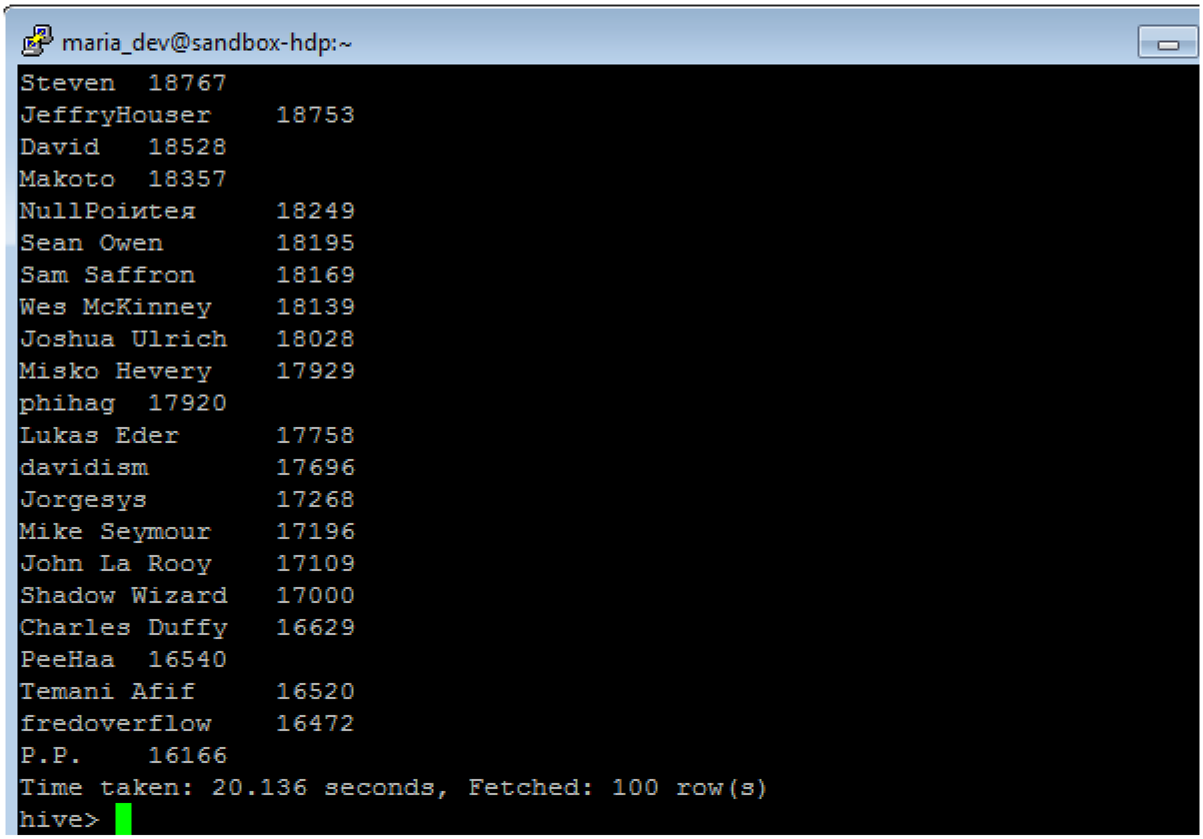
```
# evaluating the model
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
evaluator.evaluate(predictions)
```

Figure 23 Model Evaluation

6. Results

6.1 Result 1: Most Viewed Users

6.1.1 Hive



```
maria_dev@sandbox-hdp:~
Steven  18767
JeffryHouser    18753
David   18528
Makoto  18357
NullPoiиteя     18249
Sean Owen       18195
Sam Saffron     18169
Wes McKinney    18139
Joshua Ulrich   18028
Misko Hevery    17929
phihag  17920
Lukas Eder      17758
davidism        17696
Jorgesys        17268
Mike Seymour    17196
John La Rooy    17109
Shadow Wizard   17000
Charles Duffy   16629
PeeHaa  16540
Temani Afif     16520
fredoverflow    16472
P.P.    16166
Time taken: 20.136 seconds, Fetched: 100 row(s)
hive>
```

Figure 24 Hive Results

6.1.2 Pig

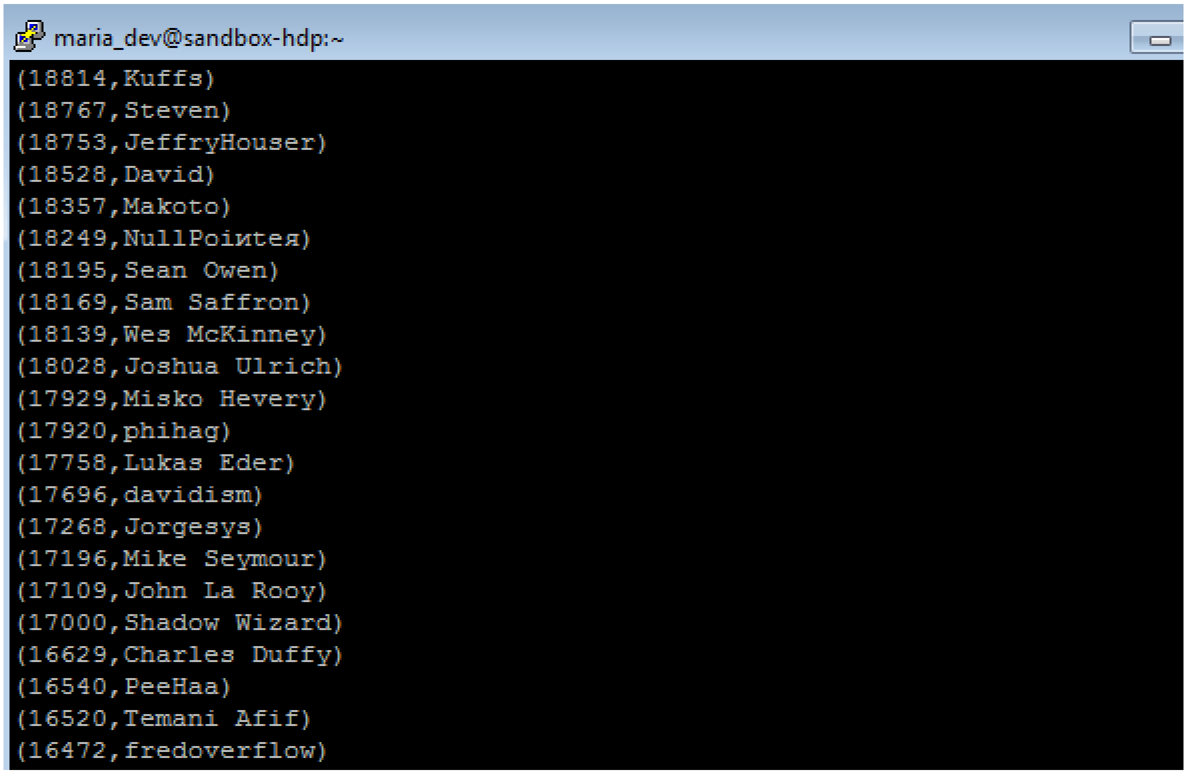


Figure 25 Pig Results

6.1.3 Graph

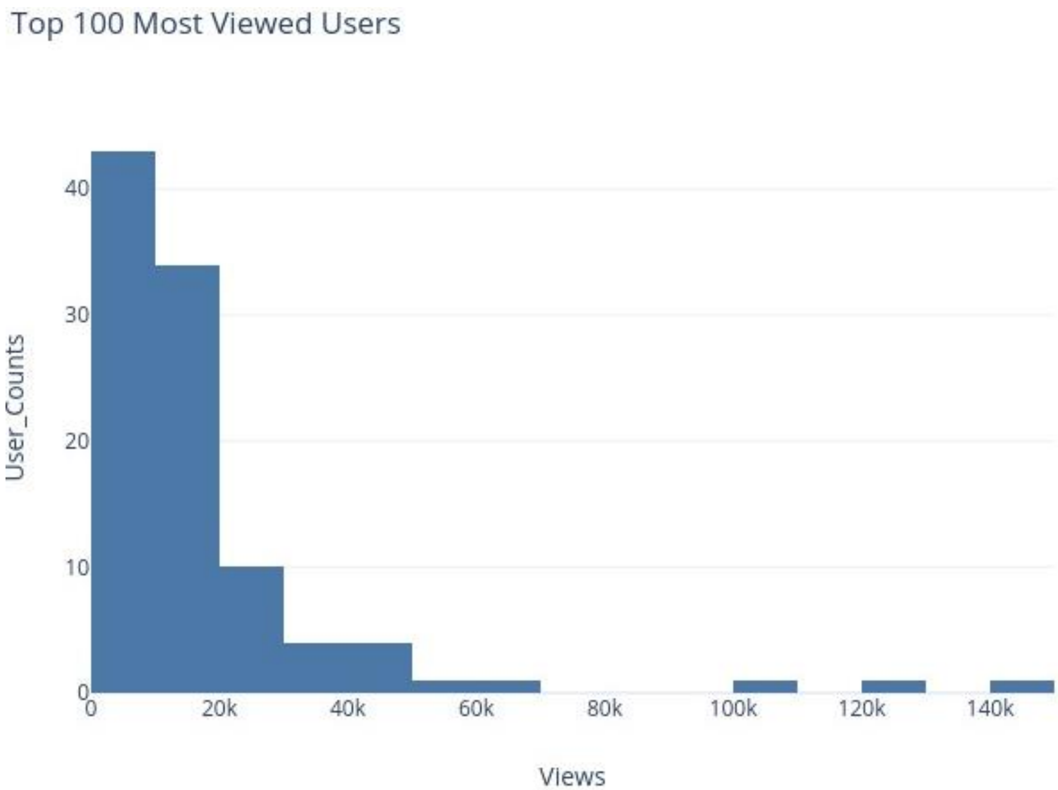


Figure 26 Top 100 Most Viewed Users

6.2 Result 2: Most Valuable Users

6.2.1 Hive

maria_dev@sandbox-hdp:~

```
Steven  18767
JeffryHouser    18753
David   18528
Makoto  18357
NullPoiиteя     18249
Sean Owen       18195
Sam Saffron     18169
Wes McKinney    18139
Joshua Ulrich   18028
Misko Hevery    17929
phihag  17920
Lukas Eder      17758
davidism        17696
Jorgesys        17268
Mike Seymour    17196
John La Rooy    17109
Shadow Wizard   17000
Charles Duffy   16629
PeeHaa  16540
Temani Afif     16520
fredoverflow    16472
P.P.    16166
Time taken: 20.136 seconds, Fetched: 100 row(s)
hive>
```

Figure 27 Hive Results

6.2.2 Pig

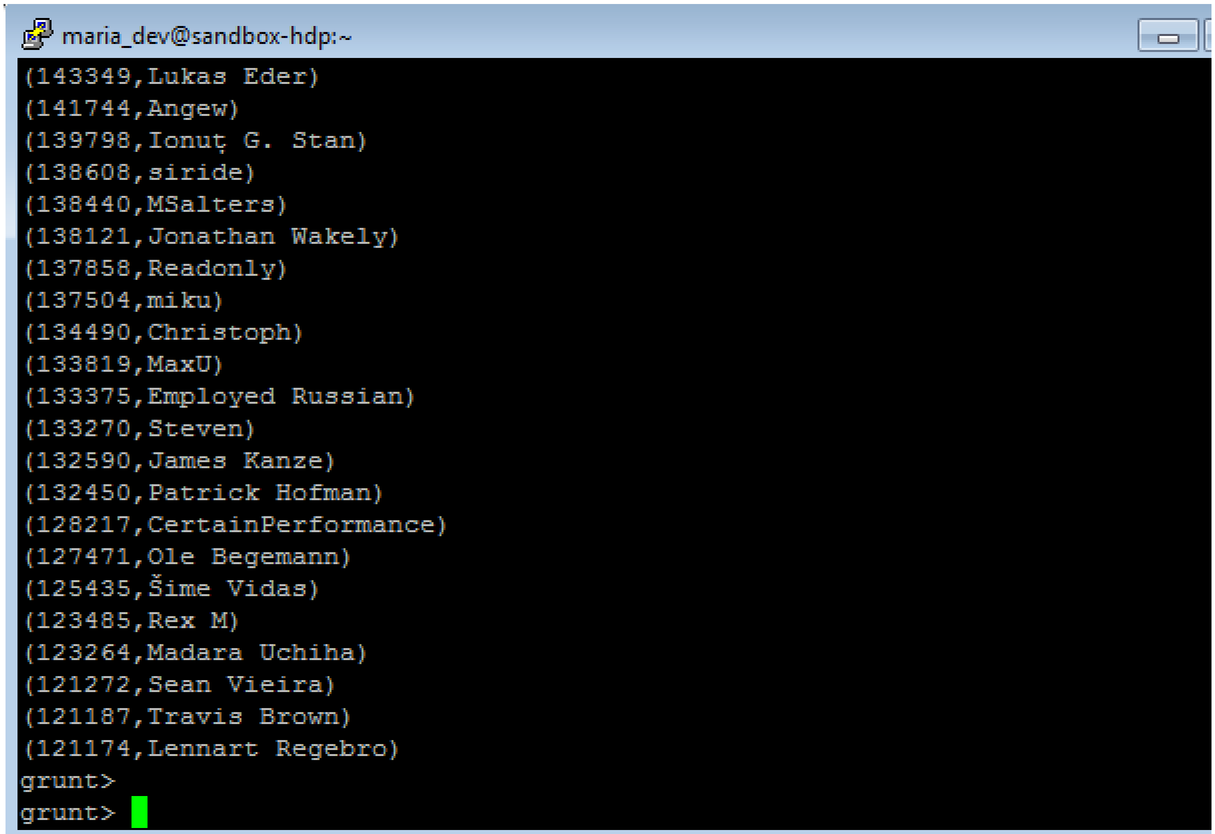


Figure 28 Pig Results

6.2.3 Graph

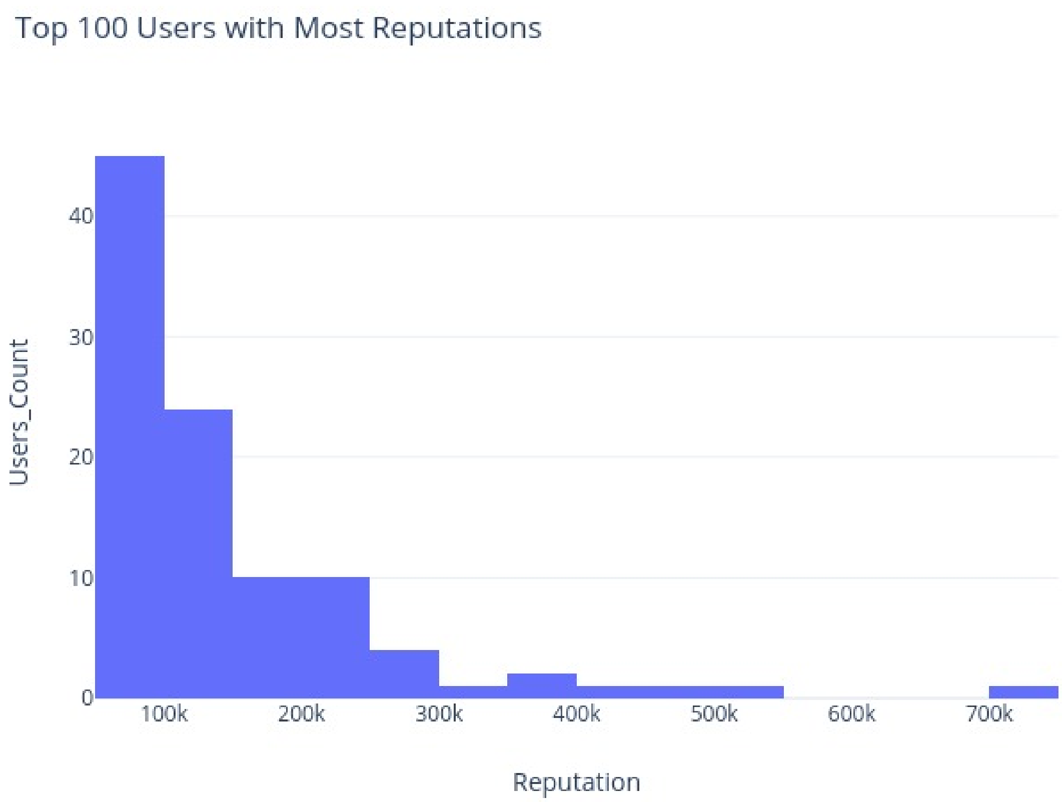


Figure 29 Top 100 Reputed Users

6.3 Result 3: Accepted Answer Percentage

6.3.1 Hive

```
13      19      35      54.285717
95      3       4       75.0
101     1       2       50.0
133     1       1       100.0
259     1       1       100.0
267     8       13      61.538464
303     1       1       100.0
304     1       2       50.0
340     2       2       100.0
350     1       1       100.0
Time taken: 16.528 seconds, Fetched: 10 row(s)
hive>
```

Figure 30 Hive Results

6.3.2 Pig

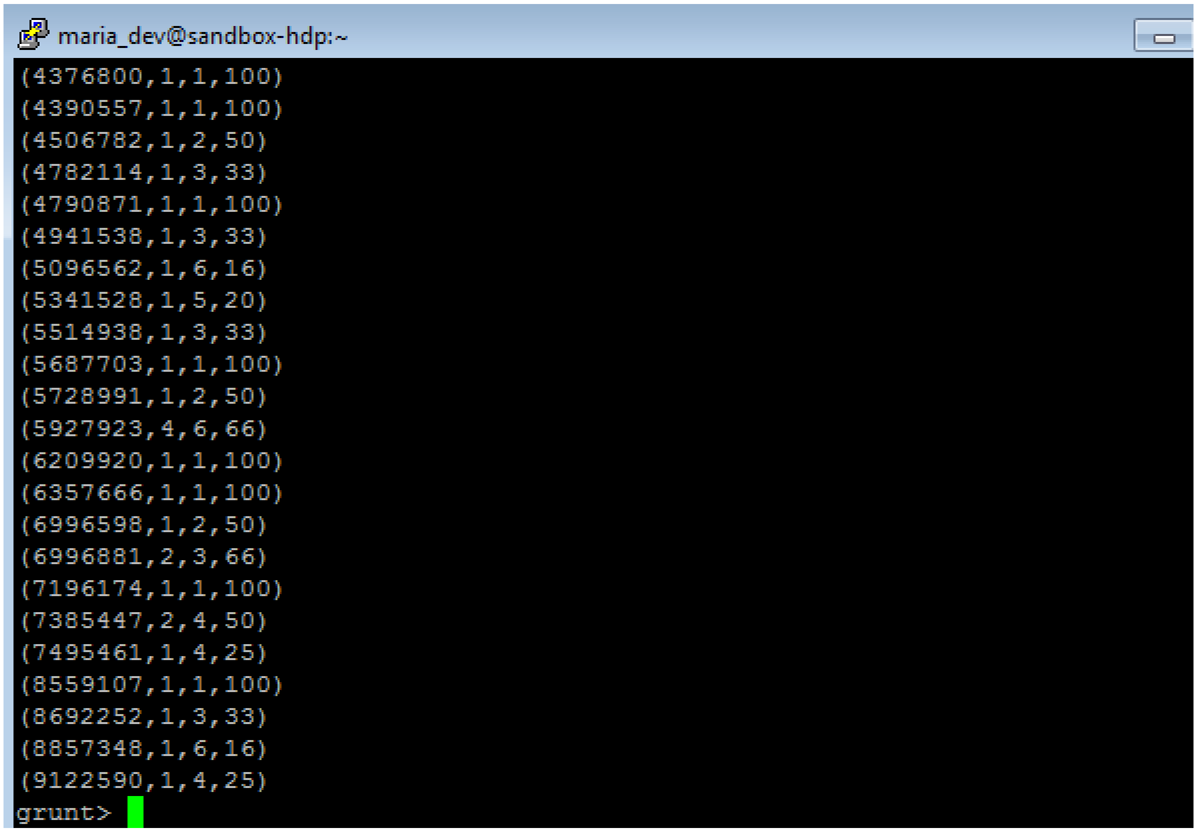
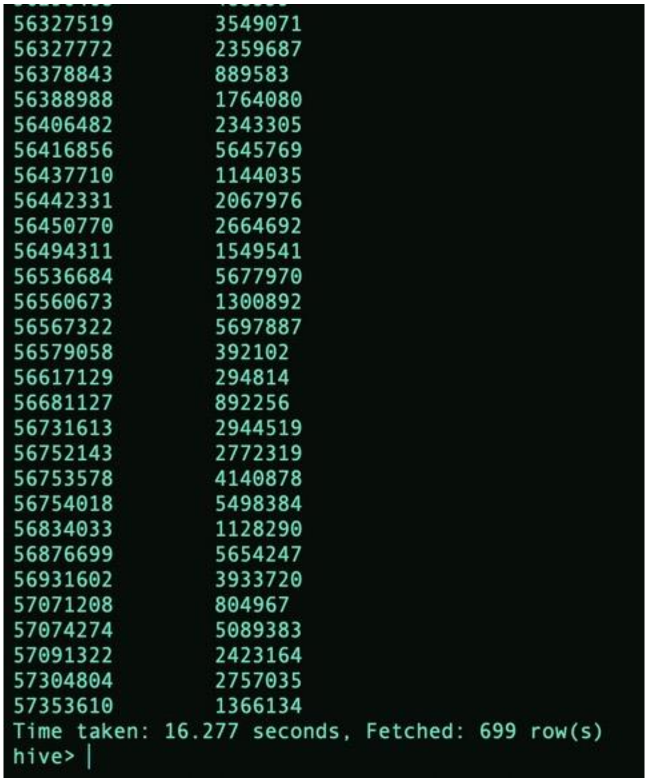


Figure 31 Pig Results

6.4 Result 4: Marking Spammers

6.4.1 Hive



```
56327519        3549071
56327772        2359687
56378843        889583
56388988        1764080
56406482        2343305
56416856        5645769
56437710        1144035
56442331        2067976
56450770        2664692
56494311        1549541
56536684        5677970
56560673        1300892
56567322        5697887
56579058        392102
56617129        294814
56681127        892256
56731613        2944519
56752143        2772319
56753578        4140878
56754018        5498384
56834033        1128290
56876699        5654247
56931602        3933720
57071208        804967
57074274        5089383
57091322        2423164
57304804        2757035
57353610        1366134
Time taken: 16.277 seconds, Fetched: 699 row(s)
hive>
```

Figure 32 Hive Results

6.4.2 Pig

```
(17276295,1505086)
(17276295,1505086)
(17291147,544557)
(17291147,544557)
(17291147,544557)
(17291147,544557)
(17291147,544557)
(17291147,544557)
(17364812,1640682)
(17414027,1137672)
(17424651,57695)
(17426438,1041948)
(17504062,1657196)
(17539158,88111)
(17772224,1233627)
(17801871,1260625)
(17990977,1114320)
(18448880,1520186)
(18854186,398041)
(19234161,499214)
(19286756,1269727)
(19296202,211665)
(19317600,1922108)
(19365264,1891521)
(19399600,1929986)
(19419965,1246764)
(19420869,1839235)
(19587803,312480)
(19639745,367273)
(19720400,324584)
(19818055,1784834)
(19830210,799586)
(19848546,1283124)
(19859543,1170677)
(19931230,477997)
(19950685,1684058)
(20023415,1062764)
(20023471,1501051)
(20028156,1607446)
(20057117,331059)
(20117853,216074)
(20121545,1062238)
(20129936,383635)
(20143274,1140682)
(20191935,1721135)
(20218202,1879076)
```

Figure 33 Pig Results

6.5 Prediction Model

After the running the evaluation model the accuracy obtained was **74.35%**.

```
0.7433501044446787
```

7. Challenges

Prediction Model

During the model prediction there were few challenges faced,

- Firstly, PySpark had compatible issues with Python 3.7, PySpark is compatible with version 2.6. Because of the version incompatibility it creates issues in distributing memory for parallel processing. PySpark (v 2.4.4) works well Python 2.6.
- Another challenge was splitting the data on the basis of delimiter and storing it in RDD was loading wrong values into the dataframe, to overcome this problem, an additional process is done by using “Pandas” dataframe which is later converted into “Spark” dataframe.
- Another problem was using Apache Zeppelin through maria_dev for data visualization, but large amount of latency was experienced when running simple queries. This might be because of the disk size allotment of virtual machine during deployment. An alternate would be to use Zeppelin on local machines, but due to the system restrictions it was not done.

Visualizing

To visualize the dataset, there are many tools available for visualization, of which one is Neo4j. Neo4J is a high performance, NoSQL graphics database that stores structured data on the network instead of tables. The issue arose when files like users, post answers and posts were inserted into Neo4j and couldn’t generate more than a single relationship in these tables. In the case of users, it was only able to show relationship for one specific user instead of the whole dataset.

8. Future Scope

Creating a GUI for tag predictor to make it more user-friendly.

Develop a recommender system for suggesting the best answer available for the question posted by the user.

Connecting PySpark directly to Google’s Big Query platform for model building using DataProc.

9. Conclusion

Given the circumstances of PySpark memory issues, it could be understood that one has to either upgrade or downgrade the Python version in local with respect to the version of Spark using. Secondly, when the parameters (such as trees and depth) are changed the accuracy also changes move upwards, but after certain limit the accuracy remained in the range of 70-75% due to unavailability of the more memory. Even after multiple iteration, the accuracy remained in between 70% to 75%.

As from the analysis done on the stack overflow dataset. Hive was much easier for user with SQL language background. It makes use of exact variation of the SQL language by defining the tables before hand and storing the schema details in any local database. Whereas in the case of Pig, there is no dedicated metadata database and the schemas or data types have to be defined in the script itself.

Hive Hadoop Component is completely used for structured data whereas Pig Hadoop Component is completely used for semi-structured data. Since, the Stack Overflow Dataset is large, Apache Hive is more suitable for the dataset.

10. References

1. S. Aravinth, A. Haseenah Begam, S. Shanmugapriyaa, and S. Sowmya, "An Efficient HADOOP Frameworks SQOOP and Ambari for Big Data Processing," *IJIRST*, vol. 1, no. 10, 2015. [Accessed 10 October 2019].
2. A. Fuad, A. Erwin, and H. Ipung, "Processing performance on Apache Pig, Apache Hive and MySQL cluster," *IEEE*, no. 10110920147010600, 2019. Available: 10.1109/ICTS.2014.7010600 [Accessed 17 October 2019].
3. V. Saminath and M. Sangeetha, "Internals of Hadoop Application Framework and Distributed File System," *IJSRP*, vol. 5, no. 7, 2015. [Accessed 19 October 2019].
4. R. Gadder and V. Namavaram, "A Survey on Evolution of Big Data with Hadoop," *IJRISE*, vol. 3, no. 6, 2017. [Accessed 11 November 2019].
5. D. Sarkar, "SQL at Scale with Apache Spark SQL and DataFrames—Concepts, Architecture, and Examples," *Medium*, 2018. [Online]. Available: https://towardsdatascience.com/sql-at-scale-with-apache-spark-sql-and-dataframes-concepts-architecture-and-examples-c567853a702f. [Accessed: 11- Nov- 2019].
6. A. Khutal, "Soccer Data Analysis Using Apache Spark SQL (Use Case) |", *AcadGild*, 2019. [Online]. Available: https://acadgild.com/blog/soccer-data-analysis-using-apache-spark-sql-use-case. [Accessed: 11- Nov- 2019].
7. Lin, J., Lin, T. and Schaedler, P. (n.d.). Predicting the best Answers for Questions on Stack Overflow.
8. Madeti, P. (2019). Using Apache Spark's MLlib To Predict Closed Questions on Stack Overflow.
9. D. Movshovitz-Attias, Y. Movshovitz-Attias, P. Steamiest, and C. Faloutsos, "Analysis of the Reputation System and User Contributions on a Question Answering Website: StackOverflow," 2013. [Accessed 13 October 2019].
10. S. Wang, D. Lo, and L. Jiang, "An Empirical Study on Developer Interactions in StackOverflow," 2014. [Accessed 12 October 2019].
11. X. Meng et al., "MLlib: Machine Learning in Apache Spark," *Journal of Machine Learning in Apache Spark*, 2016. [Accessed 25 October 2019].
12. A. Bhardwaj, A. Kumar, Y. Narayan and P. Kumar, "Big data emerging technologies: A CaseStudy with analyzing twitter data using apache hive," *IEEE*, 2015. [Accessed 11 November 2019].
13. M. Zaharia et al., "Apache Spark," *Communications of the ACM*, vol. 59, no. 11, pp. 56-65, 2016. Available: 10.1145/2934664 [Accessed 19 November 2019].

14. “Introduction to Apache Hive | Edureka.co", *Edureka*, 2019. [Online]. Available: https://www.edureka.co/blog/introduction-to-apache-hive/comment-page-1/ . [Accessed: 24- Nov- 2019].
15. P. Krishnan, "Hive - Big Data", *Big Data*, 2019. [Online]. Available: https://bigdatacurls.com/hive/ . [Accessed: 25- Nov- 2019].
16. J. Hurwitz, A. Nugent, F. Halper, and M. Kaufman, "Hadoop Pig and Pig Latin for Big Data - dummies," *dummies*, 2019. [Online]. Available: https://www.dummies.com/programming/big-data/hadoop/hadoop-pig-and-pig-latin-for-big-data/. [Accessed: 22- Nov- 2019].
17. "Apache Pig - Architecture - Tutorialspoint", *Tutorialspoint.com*, 2019. [Online]. Available: https://www.tutorialspoint.com/apache_pig/apache_pig_architecture.htm . [Accessed: 22- Nov- 2019].
18. P. SINGH *LEARN PYSPARK*. [S.l.]: APRESS, 2019, pp. 183-186.
19. "Introduction to Spark With Python: PySpark for Beginners - DZone Big Data", *dzone.com*, 2019. [Online]. Available: https://dzone.com/articles/introduction-to-spark-with-python-pyspark-for-begi. [Accessed: 25- Nov- 2019].

Appendix A (Prediction Modelling)

```
import
pyspark
          from pyspark.sql.types import *
          import pandas as pd
          from pyspark.sql import SQLContext
          from bs4 import BeautifulSoup
          from pyspark import keyword_only
          from pyspark.ml import Transformer
          from pyspark.ml.param.shared import HasInputCol, HasOutputCol
          from pyspark.sql.functions import udf
          from pyspark.sql.types import StringType
          from pyspark.ml import Pipeline
          from pyspark.ml.classification import LogisticRegression, OneVsRest,
          RandomForestClassifier
          from pyspark.ml.feature import IDF, StringIndexer, StopWordsRemover,
          CountVectorizer, RegexTokenizer, IndexToString
          import nltk
          from nltk.corpus import stopwords

          # creating schema for the dataframe
          qSchema = StructType([StructField('post', StringType(), True),
                               StructField('tags', StringType(), True)])
          d = pd.read_csv("/Users/rahulnair/desktop/stack-overflow-data.csv")

          sqlContext = SQLContext(sc)
          sdf = sqlContext.createDataFrame(d, qSchema)

          # filtering out null values
          sdf = sdf.filter(sdf.tags.isNotNull())
          print("the dataframe for the data: ", sdf)

          # Splitting the data
          (train, test) = sdf.randomSplit((0.75, 0.25), seed = 100)

          # For removing HTML tags
          class BsTextExtractor(Transformer, HasInputCol, HasOutputCol):
```

```python
    @keyword_only
    def __init__(self, inputCol=None, outputCol=None):
        super(BsTextExtractor, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    @keyword_only
    def setParams(self, inputCol=None, outputCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _transform(self, dataset):

        def f(s):
            cleaned_post = BeautifulSoup(s).text
            return cleaned_post

        t = StringType()
        out_col = self.getOutputCol()
        in_col = dataset[self.getInputCol()]
        return dataset.withColumn(out_col, udf(f, t)(in_col))

nltk.download('stopwords')

# list of stopwords to be removed from the posts
StopWords = list(set(stopwords.words('english')))

labelIndexer = StringIndexer(inputCol="tags", outputCol="label").fit(train)
bs_text_extractor = BsTextExtractor(inputCol="post", outputCol="untagged_post")
RegexTokenizer = RegexTokenizer(inputCol=bs_text_extractor.getOutputCol(),
outputCol="words", pattern="[^0-9a-z#+_]+")
StopwordRemover = StopWordsRemover(inputCol=RegexTokenizer.getOutputCol(),
outputCol="filtered_words").setStopWords(
    StopWords)
CountVectorizer = CountVectorizer(inputCol=StopwordRemover.getOutputCol(),
outputCol="countFeatures", minDF=5)
idf = IDF(inputCol=CountVectorizer.getOutputCol(), outputCol="features")
rf = RandomForestClassifier(labelCol="label", featuresCol=idf.getOutputCol(),
numTrees=100, maxDepth=4)
idx_2_string = IndexToString(inputCol="prediction", outputCol="predictedValue")
idx_2_string.setLabels(labelIndexer.labels)

# creating the pipeline
pipeline = Pipeline(stages=[
    labelIndexer,
    bs_text_extractor,
    RegexTokenizer,
    StopwordRemover,
    CountVectorizer,
    idf,
    rf,
    idx_2_string])

# fitting the model
model = pipeline.fit(train)

# performing the prediction
predictions = model.transform(test)

# convert spark dataframe to Pandas Dataframe
```

```
qwerty = predictions.toPandas()
print("the Predictions are: ", qwerty)

# evaluating the model
evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="f1")
evaluator.evaluate(predictions)
```

Appendix B

Hive

Query 1

Selecting displayname and views from users table and ordering the result by number of views in descending order and showing top 100 viewed users.

select A.displayName, A.views from project.user_data A

order by views desc limit 100;

Query 2

-- most valuable customer

Selecting DisplayName and Reputation from users table and ordering the result by Reputation in descending order and showing top 100 users.

SELECT A.DisplayName, A.Reputation FROM

(SELECT * FROM project.user_data ORDER BY reputation desc LIMIT 100) A;

Query 3

computing the total number of answers given by the user.

create table project.total_answers as

SELECT owner_user_id, count(distinct id) as total_answers FROM project.post_answers group by owner_user_id;

computing the total number correct answers given by the user.

create table project.total_accepted_answers as

SELECT A.owner_user_id as owner_user_id, count(distinct A.id) as total_accepted_answers FROM user_answers.post_answers A,

user_answers.post_questions B

where A.id=B.accepted_answer_id

group by A.owner_user_id;

number of users

select A.owner_user_id, B.total_accepted_answers, A.total_answers, float(B.total_accepted_answers/A.total_answers)*100

from total_answers A, total_accepted_answers B

where A.owner_user_id = B.owner_user_id

LIMIT 10;

Query 4

-- spam post

select A.post_id, B.user_id from votes A , comments B

Joining the postId of votes table with id of comments table a nd checking whether the voteTypeId from votes is 12 (which is confirming post has been voted spam) and usersId is not equal to 0

where A.post_id=B.id AND (A.vote_type_id==12 AND B.user_id!=0);

Appendix C

Pig

Query 1

Loading users dataset to find the users which have highest number of views by other users

loaded = LOAD 'user.csv' Using PigStorage(',') AS (id:int,displayName:chararray,creation_date:chararray, last_access_date:chararray, reputation:int, up_votes:int, down_votes:int, views:int);

generating the views and displayname for the user.

generated = FOREACH loaded GENERATE views, displayName;

ordered = ORDER generated BY views DESC;

categorizing the top 100 most viewed users

limited = LIMIT ordered 100

dump limited;

STORE limited INTO '/pigresults/mostViewedUsers';

Query 2

```
file = LOAD 'user.csv' Using PigStorage(',') AS (ID:int, DisplayName:chararray, CreationDate:chararray,
lastAccessDate:chararray, Reputation:int, upvote:int, downvote:int, views:int);

generated = FOREACH file GENERATE Reputation, DisplayName;

ordered_value = ORDER generated BY Reputation DESC;

limited = LIMIT ordered_value 100;

dump limited;

STORE limited INTO '/pigresults/most_Valuable_User' USING PigStorage(',');
```

Query 3

```
# Loading post_answers dataset from the google query table

post_answers_file = LOAD 'post_answers_file_1.csv' Using PigStorage(',') AS (id:int,

owner_user_id:int,

parent_id:chararray,

post_type_id:int,

score:int);


# Creating

grpd_post_answers  = group post_answers_file by owner_user_id;

cnt_post_answers   = foreach grpd_post_answers generate group, COUNT(post_answers_file.id);


post_questions_file = LOAD 'post_questions_file_1.csv' Using PigStorage(',') AS (id:int,

accepted_answer_id:int);


join_accepted_answers = JOIN post_answers_file BY id, post_questions_file BY accepted_answer_id;

grpd_accepted_answers  = group join_accepted_answers by owner_user_id;

cnt_accepted_answers   = foreach grpd_accepted_answers generate group,
COUNT(join_accepted_answers.post_answers_file::id);


cnt_post_answers_1 = foreach grpd_post_answers generate group as owner_user_id,
COUNT(post_answers_file.id) as post_ans;
```

cnt_accepted_answers_1 = foreach grpd_accepted_answers generate group as owner_user_id, COUNT(join_accepted_answers.post_answers_file::id) as post_acc_ans;

obtaining the percentage

join_percent_answers= join cnt_accepted_answers_1 by owner_user_id, cnt_post_answers_1 by owner_user_id;

x = foreach join_percent_answers generate cnt_accepted_answers_1::owner_user_id, cnt_accepted_answers_1::post_acc_ans, cnt_post_answers_1::post_ans, (cnt_accepted_answers_1::post_acc_ans*100)/cnt_post_answers_1::post_ans;

dump x;

Query 4

-- spam post

Loading the votes data where each post has a tag whether

loaded_votes = LOAD 'votes_data.csv' Using PigStorage(',') AS (id:int,post_id:int, vote_type_id:int);

filtered_votes = FILTER loaded_votes BY vote_type_id ==12;

loaded_comments = LOAD 'data_comments.csv' Using PigStorage(',') AS (id:int, creation_date:chararray,post_id:int,user_id:int ,score:float);

filtered_comments = FILTER loaded_comments BY user_id!=0;

join_relation = JOIN filtered_votes BY post_id, filtered_comments BY id;

spammers = FOREACH join_relation GENERATE filtered_votes::post_id, filtered_comments::user_id;

STORE gener INTO '/pigresults/topSpammers';