CSP 554
BIG DATA TECHNOLOGIES

Project Report

TABLE OF CONTENTS

TABLE OF FIGURES

Abstract

These are the days of innovation for a better future, and businesses must recognize and understand the importance of Big Data in order to address complicated and difficult challenges through better decision-making. The term "Big Data" refers to a collection of big datasets with data sizes ranging from petabytes to exabytes, which have a high rate of expansion and complexity, making typical database technology difficult to analyze. This project compares the various strategies for processing big data in terms of efficiency, computational power, ease of use and installation, and predictive modeling.

Introduction:

As the number of computer devices such as smartphones, tablets, laptops, sensors, and other devices grows, so does the amount of data or raw information generated. The internet has been rapidly evolving, with an increasing number of people connecting to it on a daily basis. Until the twentieth century, Big Data was measured in megabytes or gigabytes, but today, this measurement

has increased rapidly to terabytes and is slowly approaching zettabytes. Every day, the internet is predicted to generate approximately 500 billion gigabytes of data, the majority of which is unstructured (images, audio and video clips, logs, etc.) Earlier RDBMS were simple to use because the data contained in the database was in a consistent format, or structured data. This data may be a money-making machine for most businesses if it is used carefully and properly. When working with big data, there are numerous hurdles to overcome, including converting unstructured or semistructured data into structured data, data cleansing, and doing analytics on the data. Another significant difficulty is storage; whereas RDBMS searches for the schema in tables saved, this is not the case with data generated. Alternative technologies are offered to address the issues of data storage, processing, cleansing, and analysis.

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 to 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 other cases, 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". Big Data is characterized mainly by Five terms of Vs.

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. A focus on instant feedback and instant solutions has made the developers shift from batch oriented-approach to real-time streaming systems.

Variety: Variety refers to the format in which the data is generated. Be it structured, unstructured, or semi-structured data, 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 is in the form of video files, images, weblogs, sensor data, audio clips, etc.

Veracity: Veracity is the fourth V in the 5 V's of big data. It refers to the quality and accuracy of data. Gathered data could have missing pieces, may be inaccurate, or may not be able to provide real, valuable insight. Veracity, overall, refers to the level of trust there is in the collected data. Data can sometimes become messy and difficult to use. A large amount of data can cause more confusion than insights if it's incomplete. For example, in the medical field, if data about what drugs a patient is taking is incomplete, then the patient's life may be endangered.

Value: This refers to the value that big data can provide, and it relates directly to what organizations can do with that collected data. Being able to pull value from big data is a requirement, as the value of big data increases significantly depending on the insights that can be gained from them.

Organizations can use the same big data tools to gather and analyze the data, but how they derive value from that data should be unique to them.

Even though RDBMS is the most preferred tool in IT, 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 is 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, the time taken to process can be neglected, but for extensive datasets, time is a significant factor. Big data should possess a fast processing speed like real-time insights, which RDBMS can't. Processing Big Data with traditional methods would not be cost-effective and is a time-consuming process; to overcome these drawbacks, new technology was introduced called Hadoop.
Hadoop has subprojects like Hive, Pig, Spark Kafka, HBase, and Oozie, which are excellent options for Big Data Analytics. Most 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.

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 reduces tasks. 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 that 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 a 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. The 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 interprets the commands and executes them in sequential order.

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

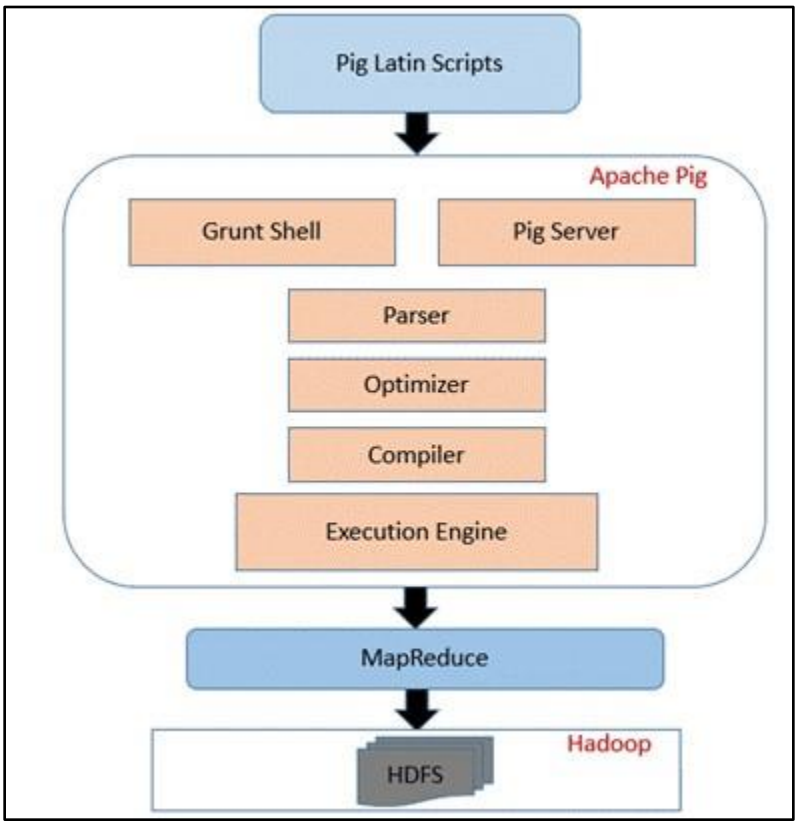
Optimizer: The DAG made by the parser is passed to the optimizer which carries out logical optimization like projection and pushdown.

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.

Features for Apache Pig:

- Pig handles the nuisance programming required for data restructuring,
- Pig Latin programs are much shorter and less complex than MapReduce programs
- Pig Latin is “compiled” into one or more MapReduce jobs
- Allows programmers to write fewer lines of code. Programmers can write 200 lines of Java code in only ten lines using the Pig Latin language.
- Apache Pig's multi-query approach reduces the development time.
- Apache pig has a rich set of datasets for performing operations like join, filter, sort, load, group, etc.
- Pig Latin language is very similar to SQL. Programmers with good SQL knowledge find it easy to write a Pig script.
- Apache Pig handles both structured and unstructured data analysis



Apache Hive

Apache Hive is an open-source data warehouse tool that is useful for analyzing large data sets. Apache Hive uses a 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.

The 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. 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. Hive has an advantage, such as indexing, which leads to faster file reading. Hive invokes MapReduce only if the query has an aggregation, join, or

sorting function, which can take one to six seconds to start the MapReduce. And also Hive is capable of handling extensive data and is faster than MySQL. Pig is not suitable for such a data set. Pig is ideal for more complex queries and more massive data sets.

Hive Components:

Metastore: A repository for metadata of Hive. It consists of information like data location, and schema along with metadata of partitions.

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

Driver: A controller is 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.

Features for Apache Hive:

Supported Computing Engine - Hive supports MapReduce, Tez, and Spark computing engines.

Framework - Hive is a stable batch-processing framework built on top of the Hadoop Distributed File system and can work as a data warehouse.

Easy To Code - Hive uses HIVE query language to query structure data which is easy to code. The 100 lines of java code we use to query a structure data can be minimized to 4 lines with HQL.

Declarative - HQL is a declarative language like SQL means it is non-procedural.

Structure Of Table - The table, the structure is similar to the RDBMS. It also supports partitioning and bucketing.

Supported data structures - Partition, Bucket, and tables are the 3 data structures that hive supports.

Supports ETL - Apache hive supports ETL i.e., Extract Transform and Load. Before Hive python is used for ETL.

Storage - Hive supports users to access files from HDFS, Apache HBase, Amazon S3, etc.

Capable - Hive is capable to process very large datasets of Petabytes in size.

Helps in processing unstructured data - We can easily embed custom MapReduce code with Hive to process unstructured data.

Fault Tolerance - Since we store Hive data on HDFS, so fault tolerance is provided by Hadoop.

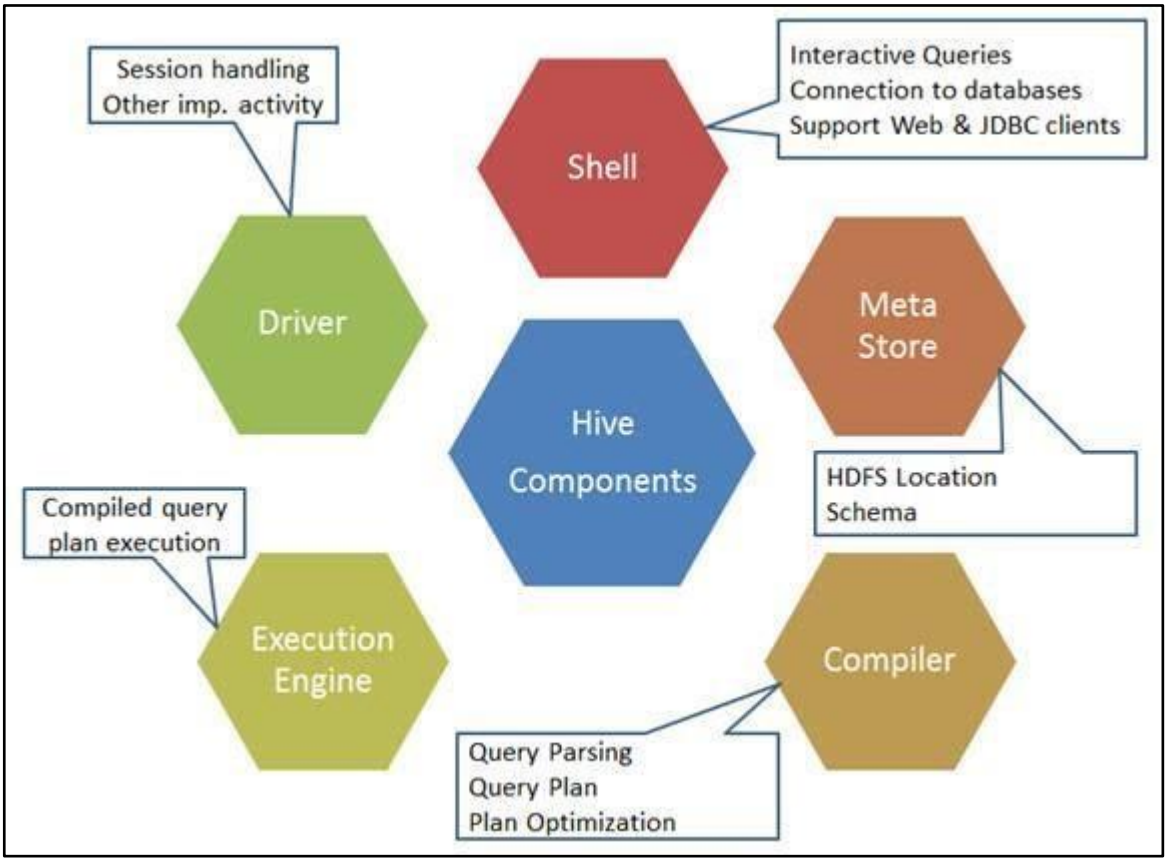


Figure 2: Hive components

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-reduce limitations, a cluster computing paradigm that forces a linear data flow 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 the database-style 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, and machine learning, but RDD in Spark handles all these workloads and works as a single engine for all requirements. These implementations use the same

optimizations as specialized engines like column-oriented processing and incremental updates; and achieve similar performance but run 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. 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 a combination of Apache Spark and Python. Deep learning has been in the limelight for quite some time in the market. Challenges in Deep Learning are exponentially rising as the 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 that 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 users to use 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 is the base 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 the existing RDD or create a new schema.

Features of Apache Spark:

Swift Processing

Using Apache Spark, we achieve a high data processing speed of about 100x faster in memory and 10x faster on the disk. This is made possible by reducing the number of read-writes to disk.

Dynamic in Nature

We can easily develop a parallel application, as Spark provides 80 high-level operators

In-Memory Computation in Spark

With in-memory processing, we can increase the processing speed. Here the data is being cached so we need not fetch data from the disk every time thus the time is saved. Spark has DAG execution engine which facilitates in-memory computation and acyclic data flow resulting in high speed.

Reusability
we can reuse the Spark code for batch-processing, join streams against historical data or run ad-hoc queries on stream state

Fault Tolerance in Spark
Apache Spark provides fault tolerance through Spark abstraction-RDD. Spark RDDs are designed to handle the failure of any worker node in the cluster. Thus, it ensures that the loss of data reduces to zero. Learn different ways to create RDD in Apache Spark.

Real-Time Stream Processing
Spark has a provision for real-time stream processing. Earlier the problem with Hadoop MapReduce was that it can handle and process data that is already present, but not the real-time data. but with Spark Streaming we can solve this problem.

Cost-Efficient
Apache Spark is a cost-effective solution for Big data problem as in Hadoop large amount of storage and a large data center is required during replication.

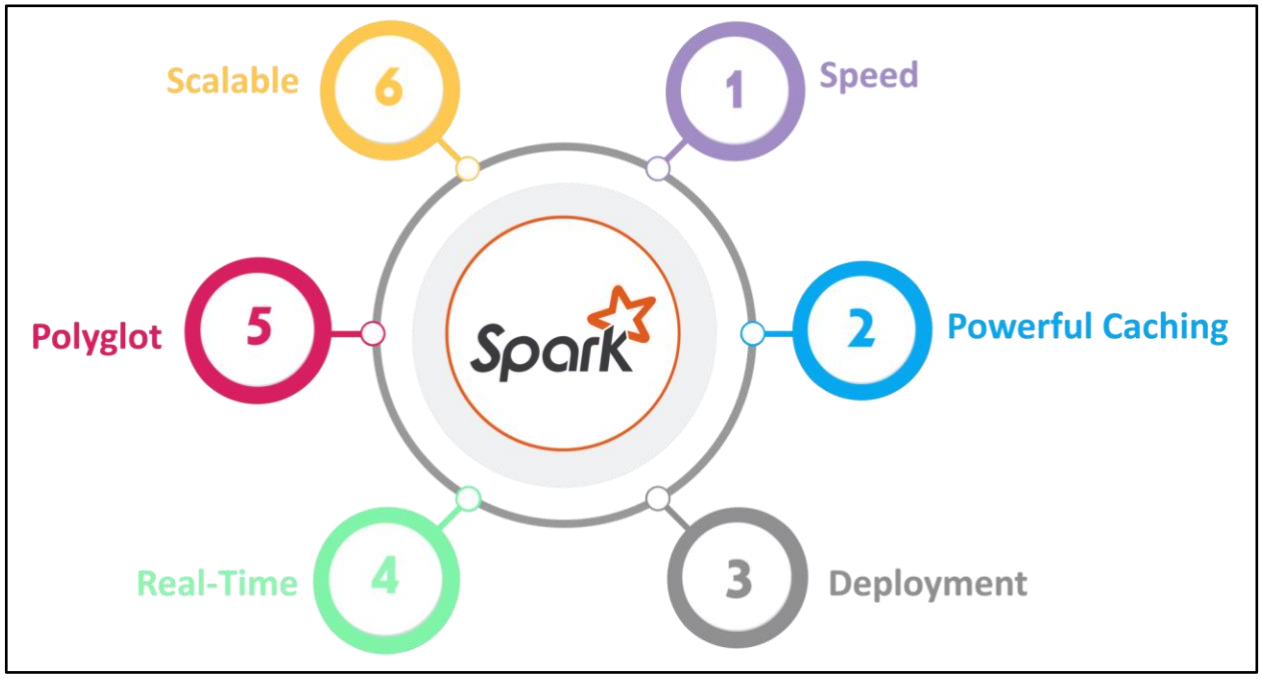


Figure 3: Spark features

Apache Pig vs Apache Hive

PIG is better for semi-structured data than Hive is for structured data. PIG is used for programming while Hive is utilized for reporting. Hive is a declarative SQL language, and PIG is a procedural language. PIG does not support partitions, whereas Hive does. PIG cannot launch a

thrift-based server, although Hive can. Hive creates tables in advance (schema) and keeps schema information in a database, whereas PIG lacks dedicated database metadata. Pig also has a COGROUP function for executing outside joins, whereas hive doesn't. Hive and PIG, on the other hand, can dynamically combine, order, and sort data.

Table 1: pig vs hive

Apache Pig	**Apache Hive**
Pig operates on the client-side of a cluster.	Hive operates on the server-side of a cluster.
Pig uses pig-latin language.	Hive uses HiveQL language.
Pig is a Procedural Data Flow Language.	Hive is a Declarative SQLish Language.
It is used to handle structured and semistructured data.	It is mainly used to handle structured data.
It does not support partitioning.	It supports partitioning.
Pig does not have a dedicated metadata database.	Hive makes use of the exact variation of dedicated SQL-DDL language by defining tables beforehand.
Pig is suitable for complex and nested data structures.	Hive is suitable for batch-processing OLAP systems.
Pig does not support schema to store data.	Hive supports schema for data insertion in tables.

Google Cloud Platform (Big Query)

Big Query is Google's data warehouse which is managed at a petabyte-scale and at a very low cost. Big Query follows a NoOps structure – which means that there is no infrastructure to manage and one doesn't need a database administrator. It follows SQL-querying systems 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.

Configurations and Architectures

OS: Windows/Ubuntu/Mac
Cloud Platform: Amazon AWS and Google GCP
Memory: 2GB
GPU: A2 – A4 VM size
Technical Skills: Good knowledge of Linux, Pig, Hive, Python, and PySpark.

Project Objectives

As previously stated, the project entails the use of various Big Data technologies such as Pig and Hive for analytics, as well as a Predictive Model for text classification using PySpark. The goal of using these technologies is to figure out which one gives better results in terms of time, GPU, and dataset size.

Acquiring Datasets

We used Google's bigQuery to access the public data of the stack overflow. The results of the big queries are stored in google's google storage (GS) for ease of access to data and efficient storage. Post the storage, the acquired data set was then copied to the clusters using google gsutil.

Step 1: Accessing the data

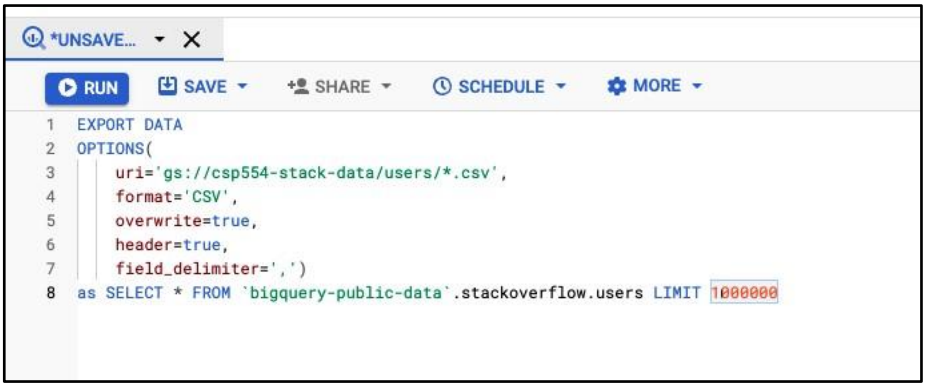


Figure 4: Accessing the data

Step 2: Confirming the data in GS

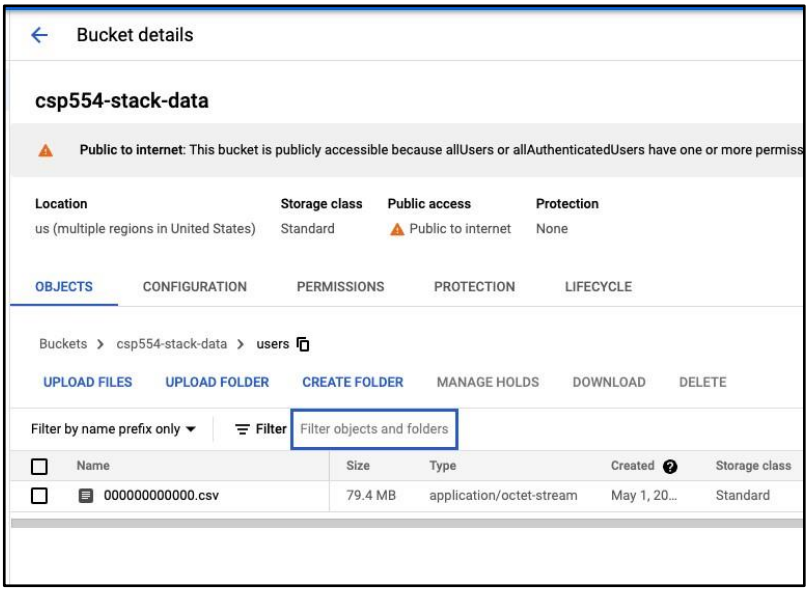


Figure 5: Confirming the data in GS

Step 3: Copy to cluster and HDFS

copy from GS buckets to the local cluster

```
[surya@stackoverflow-m:~$ gsutil cp gs://csp554-stack-data/users/000000000000.csv /home/surya/gs/users.csv
Copying gs://csp554-stack-data/users/000000000000.csv...
- [1 files][ 79.4 MiB/ 79.4 MiB]
Operation completed over 1 objects/79.4 MiB.
surya@stackoverflow-m:~$
```

Figure 6: Copy data to cluster and HDFS

Tag prediction of text classification using PySpark

We have taken the dataset from GCP BigQuery in which we preprocessed the datasets by cleaning the dataset and transferring the dataset as per our requirements. We are using PySpark for this prediction. This dataset used contains two attributes, body and tags. The body consists of what is the questions asked by the customers and tags are what category the question belongs to. The total number of tags present is 21, which will help in classifying the posts to their respective tags.

The steps to build the model are as follows:

Step 1: Importing Libraries

Importing libraries related to PySpark

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

Figure 7: Importing Libraries

Step 2: Importing Dataset

Reading the .csv file into PySpark DataFrame

```
sparkschema = StructType([StructField('post', StringType(), True), StructField('tags', StringType(), True)])
dataframe = pd.read_csv('postquestions.csv')
conf = SparkConf().setAppName("test").setMaster("local")
sc = SparkContext(conf=conf)
sqlContext = SQLContext(sc)
sparkdataframe = sqlContext.createDataFrame(dataframe, sparkschema)
sparkdataframe = sparkdataframe.filter(sparkdataframe.tags.isNotNull())
```

Figure 8: Importing Dataset

Step 3: Train and Test Dataset split

Data is split into train and test with the ratio of 80:20.

```
(train, test) = sparkdataframe.randomSplit((0.80, 0.20), seed = 100)
```

Figure 9: Train and Test Dataset split

Step 4: Removing HTML Tags

As the body consists of HTML code, we just need to remove the tags and produce the normal text, for this we used the below code

```
class HTMLTAGREMOVER(T, ICol, OCol):
  @keyword_only
  def __init__(self, iCol=None, oCol=None):
    super(HTMLTAGREMOVER, self).__init__()
    kwargs = self._input_kwargs
    self.setParams(**kwargs)
  @keyword_only
  def setParams(self, iCol=None, oCol=None):
    kwargs = self._input_kwargs
    return self._set(**kwargs)
  def _transform(self, dataset):
    def cleaingfunction(s):
      cleantext = BeautifulSoup(s).text
      return cleantext
    t = StringType()
    icol = dataset[self.getInputCol()]
    ocol = self.getOutputCol()
    return dataset.withColumn(ocol, udf(cleaingfunction, t)(icol))
nltk.download('w')
```

Figure 10: Removing HTML Tags

Step 5: List of words

Creating a list of English words to remove articles from the body column of the dataset.

```
sw = list(set(w.words('english')))
```

Figure 11: List of words

Step 6: Creating a pipeline

In the next phase, we'll define variables to build a pipeline. lI is used to convert tag string values into indexed labels, htmltagremover is used to remove HTML tags from posts, RT is used to tokenize regular expressions in posts, wordRemover is used to remove articles from posts, CV is used to create a matrix for the output from wordRemover, idf is used to calculate Inverse Document Frequency for CV, using model is used to build a Random Forest model, and finally, i is used to printing tag names.

```
lI = StringIndexer(inputCol="tags", outputCol="label").fit(train)
html_tag_remover = HTMLTAGREMOVER(inputCol="post", outputCol="untagged_post")
RT = RegexTokenizer(inputCol=html_tag_remover.getOutputCol(), outputCol="words", pattern="[^0-9a-z#+_]+")
SR = StopWordsRemover(inputCol=RegexTokenizer.getOutputCol(), outputCol="filtered_words").setStopWords(sw)
CV = CountVectorizer(inputCol=SR.getOutputCol(), outputCol="countFeatures", minDF=5)
idf = IDF(inputCol=CV.getOutputCol(), outputCol="features")
model = RandomForestClassifier(labelCol="label", featuresCol=idf.getOutputCol(), numTrees=80, maxDepth=7)
i = IndexToString(inputCol="prediction", outputCol="predictedValue")
i.setLabels(lI.labels)
```

Figure 12: Creating a pipeline

Step 7: Pipeline setting

Create a pipeline for the variables created in the previous step so that they all flow in a logical order.

```
datapipe = Pipeline(stages=[ lI,html_tag_remover, RT, SR, CV, idf,model, i])
```

Figure 13: Pipeline setting

Step 8: Training the Model

Training the model with a trained dataset

```
randomforesttmodel = datapipe.fit(train)
```

Figure 14: Training the Model

Step 9: Testing the Model

Performing prediction on the test dataset using a trained model.

```
prediction = randomforestmodel.transform(test)
```

Figure 15: Testing the Model

Step 10: Evaluation

We are now evaluating the model by the results generated by the model with the test data set.

```
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
evaluator.evaluate(prediction)
```

Figure 16: Evaluation

Queries:

Now we are using Pig and Hive to compare the results

Query 1: Accepted Answer Percentage with Posted Questions

By combining post answers and post questions and then calculating the ratio between accepted answers and total answers, a query for % of accepted answers can be created.

Pig Query:

```
grunt> postsanswers = LOAD '/user/hadoop/posts_answers.csv' Using PigStorage(',') AS (id:int,owner_user_id:int,parent_id:chararray,post_type_id:int,score:int);
22/05/02 05:25:43 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
grunt> grouped  = group postsanswers by owner_user_id;
grunt> count    = foreach grouped generate group, COUNT(postsanswers.id);
grunt> postquestions = LOAD '/user/hadoop/postsquestions.csv' Using PigStorage(',') AS (id:int,accepted_answer_id:int);
22/05/02 05:26:15 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
grunt> j = JOIN postsanswers BY id, postquestions BY accepted_answer_id;
grunt> gr  = group j by owner_user_id;
grunt> c   = foreach gr generate group, COUNT(j.postsanswers::id);
grunt> cpa = foreach grouped generate group as owner_user_id, COUNT(postsanswers.id) as postans;
grunt> caa = foreach gr generate group as owner_user_id, COUNT(j.postsanswers::id) as postaccans;
grunt> jpa= join caa by owner_user_id, cpa by owner_user_id;
grunt> per= foreach jpa generate caa::owner_user_id, caa::postaccans, cpa::postans, (caa::postaccans*100)/cpa::postans;
119327 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan  - Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
22/05/02 05:27:34 WARN newplan.BaseOperatorPlan: Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> dump per;
```

Figure 17: Pig Query 1

Hive Query:

```
hive (stackgs)> 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;
No Stats for stackgs@post_answers, Columns: owner_user_id, id
No Stats for stackgs@post_answers, Columns: owner_user_id, id
No Stats for stackgs@post_questions, Columns: accepted_answer_id
Query ID = surya_20220502023939_d82c3d40-3d88-4b4f-a49e-52d0b031247a
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1651452794346_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Map 6 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      7          7        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 5 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 4 ...... container     SUCCEEDED      3          3        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 06/06  [==========================>>] 100%  ELAPSED TIME: 25.76 s
--------------------------------------------------------------------------------
OK
```

Figure 18: hive Query 1

Query 2: Top 10 Viewed Users

We are writing a query for the top 10 viewed users in the user dataset.

Pig Query:

```
grunt> users = LOAD '/user/hadoop/users.csv' Using PigStorage(',') AS (id:int,displayName:chararray,creation_date:chararray, last_access_date:chararray,reputation:int,up_votes:int, down_votes:int, views:i
nt);
22/05/02 03:41:20 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
grunt> g = FOREACH users GENERATE displayName,views;
grunt> o = ORDER g BY views DESC;
grunt> l = LIMIT o 10;
grunt> dump l;
```

Figure 19: pig Query 2

Hive Query:

```
hive (stackgs)> select display_name,views from users order by views desc limit 10;
Query ID = surya_20220502023521_8c688de5-6587-4de0-abad-2cfbbfcb0d78
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1651452794346_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.65 s
--------------------------------------------------------------------------------
OK
```

Figure 20: hive Query 2

Query 3: Looking for Spammers

Writing a query to combine spammers into one table by matching vote id from votes dataset id = 12 (as reported by the Stack Overflow community as spam) with user id from comments dataset.

Pig Query:

```
grunt> lv = LOAD '/user/hadoop/Votes.csv' Using PigStorage(',') AS (id:int,post_id:int, vote_type_id:int);
22/05/02 05:38:03 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
grunt> fv = FILTER lv BY vote_type_id ==12;
grunt> lc = LOAD '/user/hadoop/comments.csv' Using PigStorage(',') AS (id:int, post_id:int,user_id:int);
22/05/02 05:39:47 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
grunt> fc = FILTER lc BY user_id!=0;
grunt> j = JOIN fv BY post_id, fc BY id;
grunt> spam = FOREACH j GENERATE fv::post_id, fc::user_id;
grunt> dump spam;
```

Figure 21: pig Query 3

Hive Query:

```
hive (stackgs)> 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 = surya_20220502024058_bafb9cfe-e491-44f9-91a5-dc6b81a0403b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1651452794346_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Map 3 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      4          4        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 16.18 s
--------------------------------------------------------------------------------
OK
```

Figure 22: hive Query 3

Query 4: Top 10 Reputed User

Writing query for top 10 reputed users in the user dataset.

Pig Query:

```
grunt> us = LOAD '/user/hadoop/users.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)
;
22/05/02 03:53:23 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
grunt> g = FOREACH us GENERATE  displayName,reputation;
grunt> o = ORDER g BY reputation DESC;
grunt> l = LIMIT o 10;
grunt> dump l;
```

Figure 23: pig Query 4

Hive Query:

```
hive (stackgs)> select U.display_name, U.reputation from
              > (select * from users order by reputation desc limit 10) U;
Query ID = surya_20220502023838_aca02d6c-7d3b-4225-84a6-56fe0d2a04fd
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1651452794346_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.31 s
--------------------------------------------------------------------------------
OK
```

Figure 24: hive Query 4

Results:

Tag prediction of text classification using PySpark

After the running, of the evaluation model, the accuracy obtained was 85.75%.

```
0.8575645378124437
```

Figure 25: Tag prediction results

Query 1:

Pig Query Results:

```
(4376800, 1, 1,100)
(4390557,1,1,100)
(4506702, 1,2,50)
(4782114,1,3,33)
(4790671,1,1,100)
(4941538,1,3,33)
(5096562, 1, 6, 16)
(5341528,1,5,20)
(5514938,1,3,33)
(5667703,1,1,100)
(5728991,1,2,50)
(5927923, 1, 6,66)
(6209920,1,1,100)
(6357666,1,1,100)
(6996598,1,2,50)
(6996881,2,3,66)
(7196174,1,1,100)
(7305447,2,4,50)
(7495461,1,4,25)
(8559107,1,1,100)
(8692252,1,3,33)
(8857348,1,6,16)
(9122590,1,4,25)
grunt>
```

Figure 26: pig Query 1 results

Hive Query Results:

```
a.owner_user_id b.total_accepted_answers        a.total_answers _c3
903600  1       2       50.0
2382667 1       1       100.0
5577765 1       615     0.16260162
7918    1       1       100.0
43681   1       24      4.166667
1026572 1       4       25.0
2413201 1       43      2.3255813
4851092 1       3       33.333336
307295  1       2       50.0
2647909 1       1       100.0
Time taken: 27.146 seconds, Fetched: 10 row(s)
hive (stackqs)>
```

Figure 27: hive Query 1 results

Query 2:

Pig Query Results:

```
(xhxx,205976)
(T.J. Crowder,167932)
(casperOne,131600)
(Felix Kling,114657)
(paxdiablo,92073)
(xnx,83852)
(Community,76747)
(Arun P Johny,70146)
(cs95,68584)
grunt>
```

Figure 28: pig Query 2 results

Hive Query Results:

```
OK
display_name    views
Lyndsey Scott   266731
xhxx    205976
T.J. Crowder    167932
casperOne       131600
Felix Kling     114657
paxdiablo       92073
xnx     83852
Community       76747
Arun P Johny    70146
cs95    68584
Time taken: 15.895 seconds, Fetched: 10 row(s)
hive (stackgs)>
```

Figure 29: Hive Query 2 results

The Graph below show us Top 10 viewed users

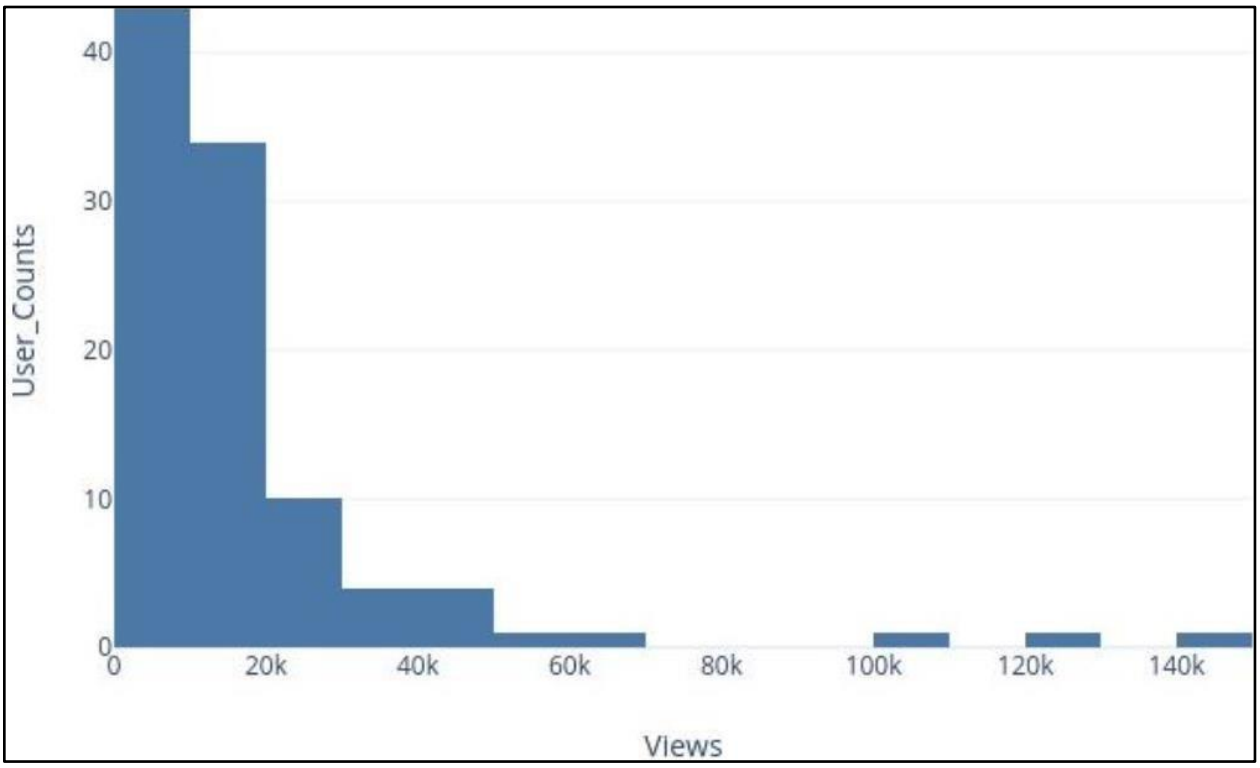


Figure 30: Top 10 viewed users

Query 3:

Pig Query Results:

```
(17291147,544557)
(17291147,544557)
(17364812,1640682)
(17414027,1137077)
(17424651,57695)
(17426438,1041948)
(17504062,1657196)
(17539158,80111)
(17772224,1733677)
(17001071,1260025)
(17998977,1114370)
(18448880,1520106)
(18854186,390041)
(19234161,409214)
(19286756,1269727)
(19296202,211665)
(19317600,1922108)
(19365264,1891521)
(19399600,1929986)
(19419965,1246764)
(19426869,1839235)
(19587803,312480)
(19639745,367273)
(19726400,324584)
(19818055,1784834)
(19838210,799586)
(19848546,1283124)
(19859543,1170677)
(19931230,477997)
(19950685,1684658)
(20023415,1062764)
(20023471,1501051)
(20028156,1607446)
(26057117,331059)
(20117853,216074)
```

Figure 31: Pig query 3 results

Hive Query Results:

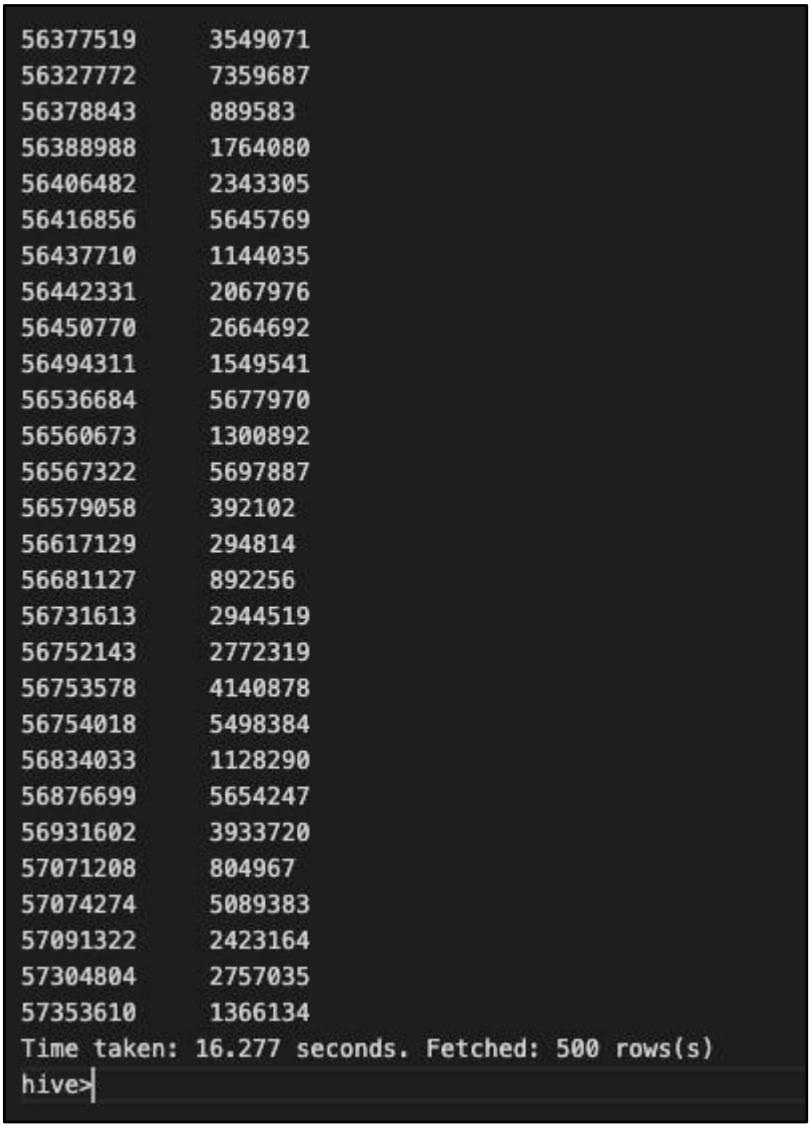


Figure 32: Hive query 3 results

Query 4:

Pig Query Results:

```
(T.J. Crowder,940041)
(paxdiablo,805833)
(Felix Kling,745407)
(Daniel Roseman,562640)
(Erwin Brandstetter,524074)
(Remy Lebeau,493061)
(kennytm,486660)
(Cascabel,445857)
(Rob,387537)
(Pascal MARTIN,383375)
grunt>
```

Figure 33: Pig query 4 results

Hive Query Results:

```
u.display_name  u.reputation
Eugene Podskal  9997
Lonnie Price    9995
Evik James      9995
Michael Gundlach        99920
Kevin Le - Khnle        9990
narupo  999
weeix   999
Kyle    999
ilia timofeev   999
Chirag Swadia   999
Time taken: 7.109 seconds, Fetched: 10 row(s)
hive (stackgs)>
```

Figure 34: hive query 4 results

Graph of Top 10 Reputed Users

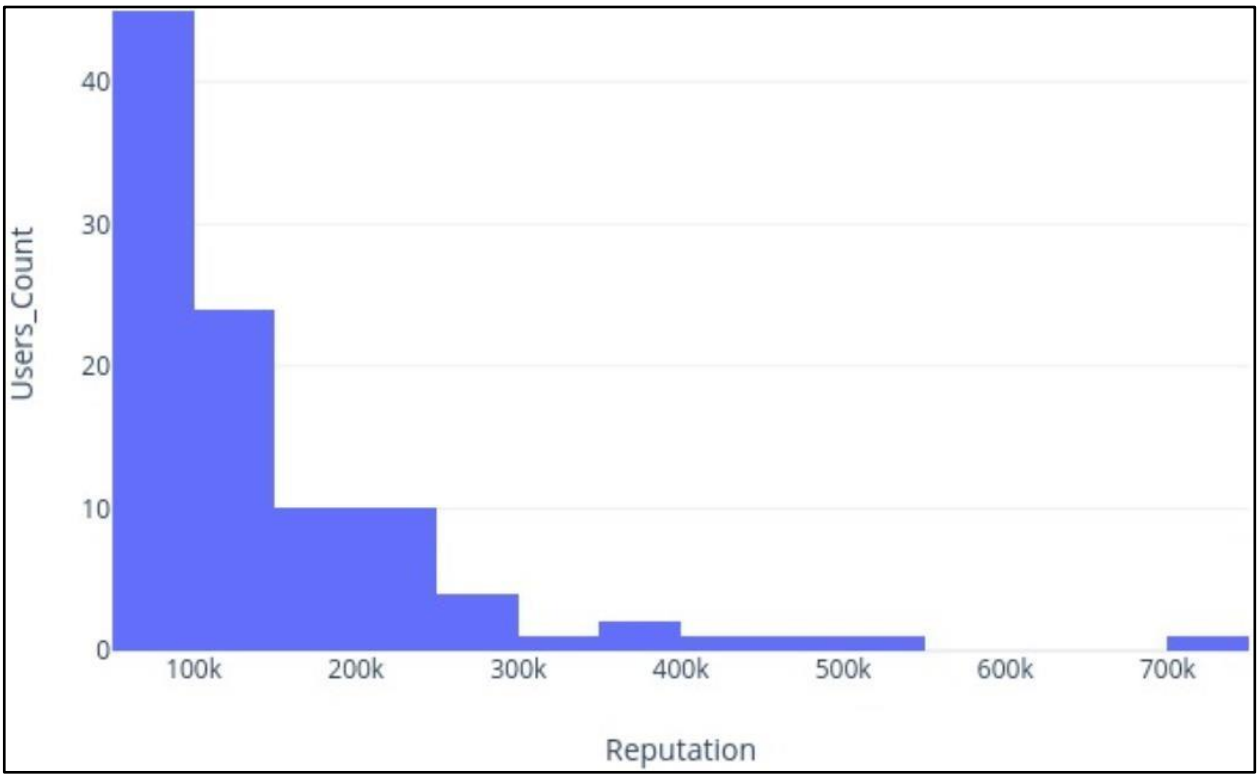


Figure 35: Graph of Top 10 Reputed Users

Experience and challenges

- Firstly, PySpark had compatible issues with Python versions. We had tried different versions, and at last PySpark (3.0) was compatible with the python version (3.9). For running the code we had to use Google colab as the dataset is not cleaned, we had done the cleaning of the dataset which had taken a lot of time as we used a larger dataset about 10 lakhs row, to get efficient results. Due to processing speed, we had taken a lot of time to compute.
- Another obstacle was dividing data by delimiter and saving it in RDD, which resulted in incorrect values being loaded into the data frame. To solve this problem, an extra procedure is used, which uses the "Pandas" data frame, which is then transformed into the "Spark" data frame.
- As we used a large Dataset when we were computing the queries for Pig and Hive, we got some errors timeout that computation power is not sufficient to run the queries, for this again we increased the computation capacity but still, we had to wait at least 20 sec to get the results.
- We are reading the datasets in Pig and Hive Tables, we had faced some challenges, to overcome this, we had used only the required columns for the computation, which helped us to increase the efficiency.
- Reading dataset from Google's Big query had challenges and limitations of download, to which we had to use google's google storage buckets for the storage of the required data set.

Ideas for Future Investigation

Creating a GUI for tag predictor to make it more user-friendly. And also Develop a recommendation system to suggest quality answers. And connecting PySpark directly to Google's Big Query platform for model building using DataProc. Bringing more insights as there is a large dataset, which gets better results helps the company.

Summary:

Given the circumstances surrounding PySpark memory problems, it is logical to presume that the Python version in local must be upgraded or downgraded in relation to the Spark version in use. Second, as the parameters (such as trees and depth) are changed, the accuracy improves, but at a certain point, because to a lack of extra memory, the accuracy remains in the 85 percent region. Even after multiple cycles, the accuracy maintained between 75 and 85 percent.

According to a study of the stack overflow dataset. Hive was a lot easier for people who were already familiar with SQL. It takes use of an exact form of the SQL language by building the tables ahead of time and storing the schema details in any local database. Pig, on the other hand, lacks a separate metadata database, necessitating the creation of schemas and data types within the script itself.

Pig Hadoop Component is for semi-structured data only, whereas Hive Hadoop Component is for structured data. Because of the size of the Stack Overflow Dataset, Apache Hive is a better fit.

Appendix

Tag prediction of text classification using PySpark:

```
1. import nltk
2. import pyspark
3. import pandas as pd
4. from bs4 import BeautifulSoup
5. from pyspark.ml import Pipeline
6. from pyspark.sql.types import *
7. from pyspark import keyword_only
8. from nltk.corpus import stopwords
9. from pyspark.sql import SQLContext
10. from pyspark.ml import Transformer
11. from pyspark.sql.functions import udf
12. from pyspark.sql.types import StringType
13. from pyspark import SparkConf, SparkContext, SQLContext
14. from pyspark.ml.param.shared import HasInputCol, HasOutputCol
15. from pyspark.ml.classification import LogisticRegression, OneVsRest,
    RandomForestClassifier
16. from pyspark.ml.feature import IDF, StringIndexer, StopWordsRemover, CountVectorizer,
    RegexTokenizer, IndexToString
```

```
17.

18. sparkschema = StructType([StructField('post', StringType(), True), StructField('tags',
    StringType(), True)])
19. dataframe = pd.read_csv('postquestions.csv')
20. conf = SparkConf().setAppName("test").setMaster("local")
21. sc = SparkContext(conf=conf)
22. sqlContext = SQLContext(sc)
23. sparkdataframe = sqlContext.createDataFrame(dataframe, sparkschema)
24. sparkdataframe = sparkdataframe.filter(sparkdataframe.tags.isNotNull())
25.
26. (train, test) = sparkdataframe.randomSplit((0.80, 0.20), seed = 100)
27.
28.
29. class HTMLTAGREMOVER(T, ICol, OCol):
30.  @keyword_only
31.  def __init__(self, iCol=None, oCol=None):
32.    super(HTMLTAGREMOVER, self).__init__()
33.    kwargs = self._input_kwargs
34.    self.setParams(**kwargs)
35.  @keyword_only
36.  def setParams(self, iCol=None, oCol=None):
37.    kwargs = self._input_kwargs
38.    return self._set(**kwargs)
39.  def _transform(self, dataset):
40.    def cleaingfunction(s):
41.      cleantext = BeautifulSoup(s).text
42.      return cleantext
43.    t = StringType()
44.    icol = dataset[self.getInputCol()]
45.    ocol = self.getOutputCol()
46.    return dataset.withColumn(ocol, udf(cleaingfunction, t)(icol))
47. nltk.download('w')
48.
49. sw = list(set(w.words('english')))
50. lI = StringIndexer(inputCol="tags", outputCol="label").fit(train)
51. html_tag_remover = HTMLTAGREMOVER(inputCol="post", outputCol="untagged_post")
52. RT = RegexTokenizer(inputCol=html_tag_remover.getOutputCol(), outputCol="words",
    pattern="[^0-9a-z#+_]+")
53. SR = StopWordsRemover(inputCol=RegexTokenizer.getOutputCol(),
    outputCol="filtered_words").setStopWords(sw)
54. CV = CountVectorizer(inputCol=SR.getOutputCol(), outputCol="countFeatures", minDF=5)
55. idf = IDF(inputCol=CV.getOutputCol(), outputCol="features")
56.  model  =  RandomForestClassifier(labelCol="label",  featuresCol=idf.getOutputCol(),
    numTrees=80, maxDepth=7)
57. i = IndexToString(inputCol="prediction", outputCol="predictedValue")
58. i.setLabels(lI.labels)
59.
60. datapipe = Pipeline(stages=[ lI,html_tag_remover, RT, SR, CV, idf,model, i])
61.
62. randomforesttmodel = datapipe.fit(train)
63.
```

```
prediction = randomforestmodel.transform(test)

topd = prediction.toPandas()
print("the Predictions are: ",topd)

evaluator = MulticlassClassificationEvaluator(labelCol="label",
    predictionCol="prediction", metricName="f1")
evaluator.evaluate(prediction)
```

Pig Queries

Query 1:

```
postsanswers = LOAD '/user/hadoop/posts_answers.csv' Using PigStorage(',') AS
(id:int,owner_user_id:int,parent_id:chararray,post_type_id:int,score:int);
grouped  = group postsanswers by owner_user_id;
count    = foreach grouped generate group, COUNT(postsanswers.id);
postquestions = LOAD '/user/hadoop/postsquestions.csv' Using PigStorage(',') AS
(id:int,accepted_answer_id:int);
j = JOIN postsanswers BY id, postquestions BY accepted_answer_id;
gr  = group j by owner_user_id;
c   = foreach gr generate group, COUNT(j.postsanswers::id);
cpa = foreach grouped generate group as owner_user_id, COUNT(postsanswers.id) as
postans;
caa = foreach gr generate group as owner_user_id, COUNT(j.postsanswers::id) as
postaccans;
jpa= join caa by owner_user_id, cpa by owner_user_id;
per= foreach jpa generate caa::owner_user_id, caa::postaccans, cpa::postans,
(caa::postaccans*100)/cpa::postans;
dump per;
```

Query 2:

```
    users = LOAD '/user/home/hadoopuser.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);
    g = FOREACH users GENERATE views, displayName;
    o = ORDER g BY views DESC;
    l= LIMIT ordered 10;
    dump l;
```

Query 3:

```
lv = LOAD '/user/hadoop/Votes.csv' Using PigStorage(',') AS (id:int,post_id:int,
vote_type_id:int);
fv = FILTER lv BY vote_type_id ==12;
lc = LOAD '/user/hadoop/comments.csv' Using PigStorage(',') AS (id:int,
post_id:int,user_id:int);
fc = FILTER lc BY user_id!=0;
j = JOIN fv BY post_id, fc BY id;
spam = FOREACH j GENERATE fv::post_id, fc::user_id;
dump spam;
```

Query 4:

```
    us = LOAD '/home/hadoop/user.csv' Using PigStorage(',')
    AS (ID:int, displayName:chararray, creationDate:chararray, lastAccessDate:chararray,
    reputation:int, upvote:int, downvote:int, views:int);
```

```
g = FOREACH us GENERATE displayName,reputation;
o = ORDER g BY Reputation DESC;
l = LIMIT ordered_value 10;
dump l;
```

Hive Queries:

Query 1:

```
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 2:

```
SELECT display_name, views
FROM users
ORDER BY views desc
LIMIT 10;
```

Query 3:

```
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 4:

```
SELECT U.display_name, U.reputation
FROM (
   SELECT * FROM users
   ORDER BY reputation DESC
   LIMIT 10) U;
```

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. 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-dataframesconcepts-architecture-and-examples- c567853a702f. [Accessed: 11- Nov-2019].

6. Khutal, "Soccer Data Analysis Using Apache Spark SQL (Use Case) |", AcadGild, 2019. [Online]. Available: https://acadgild.com/blog/soccer-data-analysis-using-apache-sparksql- 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. 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. 5665, 2016. Available: 10.1145/2934664 [Accessed 19 November 2019].
14. https://benalexkeen.com/multiclass-text-classification-with-pyspark/
15. "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].
16. P. Krishnan, "Hive - Big Data", Big Data, 2019. [Online]. Available: https://bigdatacurls.com/hive/ . [Accessed: 25- Nov- 2019].
17. 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-forbig- data/. [Accessed: 22- Nov- 2019].
18. "Apache Pig - Architecture - Tutorialspoint", Tutorialspoint.com, 2019. [Online]. Available: https://www.tutorialspoint.com/apache_pig/apache_pig_architecture.htm . [Accessed: 22- Nov- 2019].
19. P. SINGH LEARN PYSPARK. [S.l.]: APRESS, 2019, pp. 183-186.
20. "Introduction to Spark With Python: PySpark for Beginners - DZone Big Data", dzone.com,
21. 2019. [Online]. Available: https://dzone.com/articles/introduction-to-spark-with-python-pyspark-for-begi. [Accessed: 25- Nov- 2019].
22. https://www.upgrad.com/blog/apache-pig-architecture-in-hadoop/#Apache_Pig_Features
23. https://www.geeksforgeeks.org/apache-hive-features-and-limitations/
24. https://data-flair.training/blogs/apache-spark-features/
25. https://www.projectpro.io/article/difference-between-pig-and-hive-the-two-keycomponents-of-hadoop-ecosystem/
26. https://www.geeksforgeeks.org/difference-between-pig-and-hive/
27. Lecture Slides.