CCT

MSc In Data Analytics CA2

Contents

[1 Executive Summary 3](#_Toc149852250)

[1.1 Introduction 3](#_Toc149852251)

[1.2 Dataset Description 3](#_Toc149852252)

[2 Data Storage and Cleaning 3](#_Toc149852253)

[2.1 Dataset Storage 3](#_Toc149852254)

[2.2 Data Cleaning 4](#_Toc149852255)

[3 Reference 11](#_Toc149852256)

[Annexture 12](#_Toc149852257)

# Executive Summary

## Introduction

This project is an integrated assessment used for analysing a large dataset derived from the Twitter API named “ProjectTweets.csv”. The dataset comprises od 1,600,00.0 tweets, and the primary objective was to investigate the sentiment shift over the recorded time-period. A time series forecast was required for projecting the sentiment across 1 week, 1 month, and 3 months into the future.

## Dataset Description

The Twitter dataset comprises of five key attributes: ”ids” (tweet ID), “date” (tweet date), “flag” (query status), “user” (tweeting user), and “text” (tweet content).

# Data Storage and Cleaning

## Dataset Storage

The dataset “ProjectTweets.csv” was stored into the Hadoop Distributed File System (HDFS), a reliable and scalable storage system designed to span across multiple clusters of commodity hardware (Databricks, 2023). The storage procedure followed these key steps:

* HDFS Initialization: Using the command “start-dfs.sh”, the HDFS was initialized. This command ensured all background processes have been successfully started and are currently active.
* YARN Initialization: The YARN (Yet Another Resource Negotiator) resource management platform was initiated using the “start-yarn.sh” command.
* Dataset storage: The dataset was saved in HDFS using the command “-put” and using “user1” folder.
* Pyspark Integration: To facilitate data processing and analytics, Pyspark was used. Pyspark is an interface for Apache Spark in Python with powerful big data processing capabilities.
* Jupyter Notebook Access: The dataset analysis was executed using Jupyter notebook. By starting Pyspark, a Jupyter notebook was automatically launched and made accessible at the address “http://localhost:8889/notebooks/Downloads/Ca2.ipynb”

A screenshot of a computer

Description automatically generated

Figure 2.1: Jupyter Notebook Access

In summary, the dataset was securely stored in HDFS. YARN together with Pyspark were then used to establish a robust analytical environment through the Jupiter notebook interface.

## Data Cleaning

The initial row of the dataset served as the header, which was preserved and saved under the name "header\_row". Subsequently, it was added back into the dataset with the header names being updated to: "index", "user\_id", "timestamp", "query", "username", and "tweet\_text". In order for the header to be updated the “withColumnRenamed” Pyspark dataframe method was used (Spark, 2023).

The "query" column was removed as it didn’t hold any important information.

### Missing Data

Using pyspark sql functions “when”, “count” and “col” each column was checked for missing values. The output indicated that the dataset doesn’t have missing values (Naveen, 2022).

A screen shot of a computer

Description automatically generated.

Figure 2.2: Missing Data Output

### Tweet Text Cleaning

The tweet\_text column was cleaned using “withColumn” dataframe function (Zach, 2023). The following steps were taken:

* URL removal: All URLs starting with http:// or https:// were removed from the text.
* Hashtags removal: Hashtags, which are words starting with “#” were removed
* Special characters removal: Special characters including &,\*,%,$,~,@ and ! were removed. It was later decided to retain “!” and “?” because of their potential significance in sentiment analysis.

A screenshot of a computer

Description automatically generated

Figure 2.3: Tweet Text Cleaning

### Timestamp Transformation

The dataset contained timestamps in the format "Mon Apr 06 22:19:45 PDT 2009" (StringType). All timestamps were converted into a standard datetime format to support time series sentiment analysis. Due to the changes made in Spark 3.0 datetime parsing bahaviour, the parsing policy was changed to “LEGACY” (Datacamp, 2022) to ensure compatibility. Using PySpark's in-built datetime functions (“unix\_timestamp”, ”from\_unixtime”, “timestamptype”), the timestamps were transformed into the format "2009-04-07 06:19:45".

A white paper with black numbers and black text

Description automatically generated

Figure 2.4: Timestamp transformation.

### Column Renaming

The “tweet\_text” column was dropped after creating the “cleaned\_text” column. The latter was renamed to “tweets”. The user\_”id column” was renamed to “userid”.

### Text Normalization

The “tweets” column was converted to lowercase to maintain consistency and reduce redundancy during sentiment analysis, as the same word could have been treated as distinct if written in uppercase or lowercase.

A screenshot of a computer code

Description automatically generated

Figure 2.5: Cleaned Dataset.

### Removing Duplicate Rows

The removal of duplicate rows was achieved in Spark using the “distinct()” and the “dropDuplicates()” method. First the dataset was interrogated using the distinct().count() function. The results showed that the dataset has 1,600,000.0 of distinct rows, meaning that no duplicates were identified. A further analysis done specifically on the “username” and “tweets” columns revealed otherwise. After discarding the duplicates, 1,588,174 rows remained. To determine which rows were duplicated, the “subtract” function was utilized. A visualization of these rows was achieved using the “show” function. To validate the accuracy of this method, a selection of rows was randomly cross referenced against the original dataset. All spot checks confirmed that the method used was successful.

## Comparative Analysis Using YCSB Benchmarking Tool

YCSB is a good tool for assessing the performance of widely used database such as MySQL and MongoDB. This is attributed to the extensive adoption of these databases and the robustness of their respective YCSB connectors or drivers (Pandey, 2020). Databases like Cassandra and HBase are designed with particular architectural features that are optimized for certain use cases, such as handling very large volumes of data spread across many servers (Mehta, 2021). Cassandra, for instance, is a wide-column store (Scylla, 2023) that excels at managing large datasets with high write and read throughput across distributed systems. Similarly, HBase offers efficient storage and rapid access to big data tables, and it’s particularly good for sequential read/write operations. While YCSB can certainly be used to benchmark these databases, their specialized nature might mean that a benchmarking tool specifically developed for their unique architecture would more accurately showcase their performance advantages in the scenarios they are built for. In other words, although YCSB provides a general framework for performance testing, a more customized approach might be required to fully tap into and evaluate the distinctive capabilities and performance optimizations of databases like Cassandra and HBase.

YCSB, MySQL, and MongoDB have been configured and prepared for the performance analysis. The cleaned dataset, as mentioned in section 2.2, will serve to populate the newly created MySQL database. This database will subsequently be employed as a workload within YCSB to produce results that will reflect on the performance metrics of both MySQL and MongoDB.

### Database Preparation

First the cleaned dataset named “distinct\_rows\_based\_on\_colums” was saved in HDFS user 1 folder using the “write.csv” function using the path “/user1/cleaned\_ProjectTweet.csv”.

A screenshot of a computer screen

Description automatically generated

Figure 2.6: HDFS user 1 folder containing cleaned\_ProjectTweets.csv.

MySql was accessed using the command "-u root -p" and a new database named tweetsDB was established. Subsequently, a table meant for the cleaned dataset was created. An attempt to load the cleaned dataset directly from HDFS was made, but it resulted in an error. The allowed directory specified by the "secure-file-priv" option was then checked using the "show" command. Finally, efforts were made to save the file on the local system and then save it in the specified location “var/lib/mysql-files/”.

A screenshot of a computer program

Description automatically generated

Figure 2.7: TweetsDB new database.

The output files in Spark’s DataFrame when “write.csv” method is used are split in multiple parts and files start with “part- “. In order to get the file from HDFS the parts had to be merged using the “getmerge” command and save the output as a csv file on the local directory “Lab05”.



Figure 2.8: Retrieving the file from HDFS.

The merged file was then saved in the specified location and previous commands required to populate the database were used.

When the cleaned dataset was loaded in mysql an error was generated. The error was related to column 6968. The potential reason for this is that mysql interprets the emoticons or punctuation as a new row. This information might be relevant for the sentiment analysis and cannot be removed. In order to continue the test with YCSB a new dataset was created with only 1000 rows, and this was analysed further.

A screenshot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

Figure 2.9: Error loading the dataset.

Errors were still generated even with a reduced size of the dataset. The dataset was further cleaned and the process of saving the content of the .cvs in MySQL reinitiated. Finally, it worked and the results are indicated in the below snip.

A screen shot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

Figure 2.10: Database table generated.

The YCSB workload was loaded however a couple of errors were generated. These errors were related to YCSB trying to find the table “usertable”. The “tweetsDB” database has a different table name. The “db.properties” has been updated with the new table name and the database will need to change the headers to YCSB expected values field01 etc.

A computer screen shot of a program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Figure 2.11: YCSB errors and table fields corrected.

A new column “YCSB\_KEY” was added to the database with data type of “VARCHAR(255)” as a primary key. The YCSB workload was changed to reduce the field count to 4 instead of 10. The YCSB finally run successfully, and the results are listed below.

A computer screen shot of a black background

Description automatically generated

Figure 2.12: Successful execution of YCSB load operation for workload.

A screenshot of a computer program

Description automatically generated

Figure 2.13: Successful execution of YCSB load operation for workload.

While trying to save the output file named “ycsb\_output.txt” another error was generated.

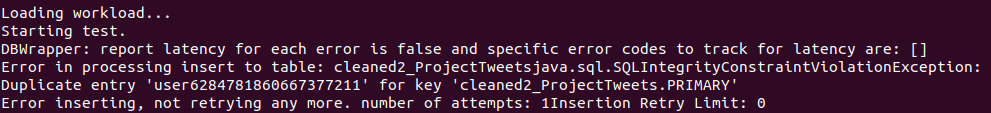


Figure 2.14: YCSB error while loading the results.

A new empty database was create named “BenchTest” and a table named “usertable” following the YCSB requirements. All settings were changed to match the new database.

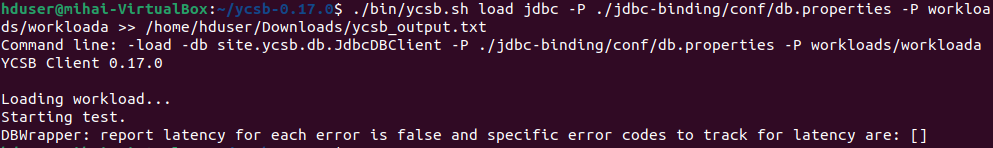


Figure 2.15: YCSB successfully loading the results in the output file.

A few changes were made to the workload properties to better reflect our dataset:

* Changed the number of record count to the number of tweets 1,600,000.00. The record count was changed to 16,000.0 as the previous setup would take approximately 4h (this is an approximation based on the 9,366ms latency for 1000 count)
* Changed the operation count to the same number as well to see how long it takes to load the tweets into the database.
* As sentiment analysis and time series analysis typically involves aggregating, summarizing and querying existing data and because data will be used to populate a dashboard the read-heavy workload was used (95% read and 5% update).
* The Zipfian distribution was kept matching a real-world scenario where a few items are very popular and accessed frequently.

Note: See video demonstration <https://vimeo.com/881370713/2bd7f5b176?share=copy>

The first observation indicates that MangoDB is much faster than MySQL. For same amount of records, it only took 3seconds to completed (vs 143seconds for MySQL).

MySQL’s throughput was approximately 112 operation per second, which is quite low in comparison with MongoDB’s throughput of 5291 operations per second.

A computer screen shot of a program

Description automatically generated

Figure 2.16: YCSB MySQL results.

The average latency for inserting operations in MySQL was 8850 ms, minimum latency 4708 and maximum latency of 201,343. MongoDB demonstrated a significant lower average insert latency of 145.83ms and same goes for minimum and maximum.

Overall, these results illustrate MongoDB’s superior speed in this scenario, handling large volumes of data with lower latency and higher throughput compared to MySQL.

A screenshot of a computer program

Description automatically generated

Figure 2.17: YCSB MongoDB results.

# Sentiment Analysis

Two cleaned datasets were generated, named "cleaned\_ProjectTweets" (which retains potentially sentiment-relevant punctuation) and "cleanedt\_ProjectTweets" (which is entirely free of punctuation and includes only words). Due to loading errors with the first dataset, only the second one will be utilized moving forward.

An initial review of the dataset was conducted. To minimize noise and randomness in the data, stop words were first removed using the "stopwords" function from NLTK. Next, the “word\_tokenize”, “FreqDist”, and “most\_common” functions were employed to identify the 10 most frequently occurring words across all tweets. Interestingly, most of these common words seem to have a positive connotation.

A white background with black text

Description automatically generated

Figure 3.1: Most common words using “FreqDist”.

NLTK “Concordance” was used for four positive key words and the corresponding antonyms. The results show that we have more positive tweets (150,956) than we have negative tweets (32,195).

A close-up of a text

Description automatically generated

A close-up of a text

Description automatically generated

Figure 3.2: Positive vs Negative tweets using “Concordance”.

Applying “Trigrams” hasn’t provide any additional input.

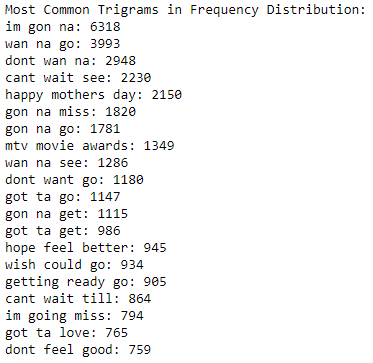


Figure 3.3: Positive vs Negative tweets using “Concordance”.

The NLTK Sentiment Intensity Analyzer along with the “vader\_lexicon” was used to produce polarity scores. These scores label sentiments as negative, positive, neutral, and give a combined compound score. The compound score combines the individual polarity scores into a single metric that ranges from -1 (highly negative) to +1 (highly positive). The compound score was used to gauge the overall sentiment of a dataset. The average compound score was 0.13 which implies that the sentiment across the dataset was marginally positive. Since the score is nearer to zero, it indicates a sentiment that is more neutral overall.

BERT was used to predict sentiment and classify each prediction as positive, negative, or neutral. However, based on progress bar output the total iteration required were 1584469 and running at 1.48 iteration per second it would take approximative 300h or 12 days. The process was stopped.



Figure 3.4: BERT model progress.

If a cluster or computers or cloud resources were available, distributed computing framework like Apache Spark could be used to improve speed. Processing can then occur on each chunk of data in parallel, significantly speeding up the time it takes to run through the entire dataset.

Comparison Between SIA and BERT:

VADER Sentiment Analysis (SIA): It's a lexicon and rule-based sentiment analysis tool that is specifically adapted to sentiments expressed in social media. It's lightweight and fast, but it might not always capture the nuanced context of the text as it relies on predefined rules and a static lexicon (Mahreen, 2022).

BERT: Provides a deep understanding of the context within the text, potentially leading to more accurate sentiment predictions. However, it's much more resource-intensive and can be slower, as demonstrated (Yalçın, 2020).

While BERT is state-of-the-art and its likely to produce more accurate results due to its understanding of context and language nuance, the time and resources needed might not justify the potential increase in accuracy, especially if the dataset is very large.

Due to the project's time and resource limitations, the decision was made to proceed with the results from VADER, as they were acquired relatively fast and with significantly lower computational requirements.

A Word Cloud Dashboard was created to showcase the most frequent words associated with negative, positive, or neutral sentiments based on the “compound” column generated with SIA.

A close up of words

Description automatically generated

Figure 3.5: SIA dashboard for sentiment analysis.

# Time Series

The data preparation continued under this section by removing columns that were not essential and checking for missing values or duplicates for the other columns. The “polarity\_scores” column was removed as the sentiment scores were already extracted. The “userid” column contains only numerical values and since the purpose of this analysis is with trends over time rather than individual user sentiment trends this column was removed. This simplified our dataset further and helped focus the analysis on the temporal aspects and sentiment scores.

By plotting the number of tweets per day it was evident that some days have many more tweets than other.

A graph of blue lines

Description automatically generated

A graph with red lines

Description automatically generated

Figure 4.1: Tweets per day plot.

The fact that some days have many more tweets than others can have an important impact on applying ARIMA, LSTM, and GRU for time series modelling of the dataset. Uneven distribution of tweets per day can lead to data imbalance. Some days may have a lot of data points, while others have very few. This can affect the performance of the models. Models like ARIMA are well-suited for capturing linear trends and seasonality, while LSTM and GRU are more capable of capturing complex patterns. When splitting the dataset into training, validation, and test sets, it's essential to ensure that the distribution of tweets per day is representative in each set. Randomly splitting your data might lead to a bias if certain days dominate one of the sets. Evaluation metrics that account for the data's imbalanced nature will be used. For example, using metrics like Mean Absolute Error (MAE) or Mean Absolute Percentage Error (MAPE) may provide a better assessment of model performance than simple accuracy.

A box plot was generated for weekly compound values and number of weekly number of tweets. The median line above zero suggests that the sentiment is generally positive for that week. The weeks with high volume of tweets have more extreme sentiment scores (both positive and negative).

A graph of different colored squares

Description automatically generated

A graph of a growing graph

Description automatically generated with medium confidence

Figure 4.2: Weekly compound sentiment and number of tweets.

A time series can be modelled as an additive or multiplicative (Radečić, 2021). The multiplicative decomposition would fail because it’s not appropriate for series with zero or negative values. The additive decomposition showed that the long-term movement in daily average was relatively smooth, indicating that any long-term changes in sentiment are gradual rather then abrupt. The residual component was irregular and this represents the noise in the dataset.

A graph of a graph

Description automatically generated with medium confidence

Figure 4.3: Additive decomposition.

# Reference

# Annexture

A screenshot of a computer

Description automatically generated

Note: the total count includes section title, references etc.

Github project location: https://github.com/cuculicu/CA2-Second-semester