**Distributed Data Processing and Sentiment Analysis of a Twitter Dataset**

Author: Jefferson William Teixeira

e-mail: [sba22201@student.cct.ie](mailto:sba22201@student.cct.ie)

Student ID: sba22201

Contents

[1 Introduction 3](#_Toc135956422)

[2 Materials and Methods 3](#_Toc135956423)

[3 Data Preparation and Visualization 4](#_Toc135956424)

[3.1 Text Processing for Sentiment Analysis 4](#_Toc135956425)

[3.2 Data Preparation for Time-Series Forecasting 6](#_Toc135956426)

[4 Data Storage Strategies 6](#_Toc135956427)

[4.1 Hadoop Setup 6](#_Toc135956428)

[4.2 MongoDB 7](#_Toc135956429)

[4.3 YCSB Benchmarking 8](#_Toc135956430)

[5 Sentiment Time-Series Forecasting 10](#_Toc135956431)

[6 Conclusions 12](#_Toc135956432)

[7 References 13](#_Toc135956433)

# Introduction

In this project, distributed data processing and machine learning (ML) were applied to a large Twitter dataset that contains comments about the war in Ukraine. The dataset was collected from a Kaggle [repository](https://www.kaggle.com/code/ssaisuryateja/eda-and-sentiment-analysis/input) and contains about 364,875 tweets on this topic. For this project, the dataset was stored using distributed environment and a NoSQL database as well as processed for sentiment analysis. The tweets were analysed using time-series forecasting to predict the overall Twitter user’s sentiment in different time intervals as per the experiments presented in this report. Moreover, a benchmarking tool called Yahoo Cloud Serving Benchmark (YCSB) was used to evaluate and compare the performance of MySQL and MongoDB databases.

This report is organized as follows: Section 2 describes the tools used to complete this project. In Section 3, the dataset and the data preparation steps for the sentiment analysis and time-series forecasting are described. In Section 4, the distributed environment and benchmarking results are discussed, while Section 5 covers the time series forecasting experiments and the dynamic dashboard implemented. Finally, in Section 6, the conclusions are summarized.

# Materials and Methods

The source code and files used in this project were hosted on GitHub under a public organization called [CCT-MastersDA](https://github.com/CCT-MastersDA). The project’s repository is called [cct-sem2-ca2](https://github.com/CCT-MastersDA/cct-ca2), which can be downloaded with the following command in any terminal: *git clone* [*https://github.com/CCT-MastersDA/cct-sem2-ca2.git*](https://github.com/CCT-MastersDA/cct-sem2-ca2.git). The source code is organized under the *jupyter* folder in the root directory which also contains a report folder for the documentation. Since GitHub is a well-known robust public repository, it was picked as the version control system for this project.

For the sentiment analysis and time-series forecasting, it was implemented a Jupyter notebook called ca2-jupyter.ipynb. This notebook depends on the following folders: *datasets*, which contains the csv files used in this project, the *images* folder, where all the generated images are stored, and the *modules* folder, which basically contains auxiliary classes for the text processing and data manipulation. The reason to separate these modules from the Jupyter notebooks was to keep the code organised and to follow the best programming practices with regards to reuse and code modularization.

The distributed environment used to store the datasets was Hadoop Filesystem (HFS) (White, 2012) and the technology implemented to access and manipulate the data was PySpark, a Python API that enables large-scale data processing in Python. The dataset was also loaded into MongoDB, a NoSQL database. The reason to use HFS was its good integration with Python, which can be leveraged by the PySpark library, so it was easier to implement the overall solution using these tools. MongoDB was selected because of its good performance in handling large datasets, which is the aim of this project.

Regarding the benchmarking approach, MongoDB and MySQL databases were compared using YCSB, an open-source specification and software package for benchmarking databases’ relative performance. For this purpose, it was implemented three bash scripts, which can be found under the *jupyter/benchmark* folder. The benchmarking-script.sh contains the code to run different workloads in Mongo and MySQL. The parse-results.sh is responsible for collecting the metrics from the results files for each execution. Finally, the exec.sh script is the trigger script that receives as input the desired number of executions. The reason to implement the scripts was to automate the performance tests, which makes it easier to run and collect the resulting metrics.

Regarding the setup, for the distributed environment and data storage experiments, a Linux VM was prepared with the following main required software: MongoDB, MySQL, YCSB, Hadoop, PySpark and Anaconda. The sentiment analysis and time-series forecasting were implemented in Windows and were executed in Anaconda environment.

# Data Preparation and Visualization

The dataset used has 215Mb and contains 364,875 posts about the conflict in Ukraine that were published by Twitter users within two days, between 01/04/22 and 02/04/22. Due to time and performance constraints, it was not feasible to collect data about this topic for the whole year as the data on this topic was massive. This way, to overcome this problem and perform the analysis proposed in this project, the date column was created artificially, so that it covers the period between 01/01/2022 and 31/12/2022. Thus, each tweet was assigned a random timestamp, so that there are about 999 tweets a day in the processed dataset, covering one year period.

Since the focus of this analysis is the sentiment of the users over time, this approach does not affect that result, however, it makes the analysis less accurate in terms of the timing as the original timestamps were modified. Therefore, the results presented in this report are meant to be taken as an exercise only.

The rest of this section is focused on the data preparation approach used for the sentiment analysis and time-series forecasting. The steps discussed in this section were implemented in the section 4 of the Jupyter notebook accompanying this report.

## Text Processing for Sentiment Analysis

The raw tweets are pieces of data that can contain text, hyperlinks, emojis and special characters. For the sentiment analysis, all these noises may affect the quality of the results, so they must be pre-processed first. In this context, text processing normalization strategies and algorithms were applied to this dataset. This way, four versions of the tweets were generated from the raw text: (1) cleaned with stop words, (2) cleaned without stop words, (3) *lemmatized,* and (4) *stemmerized* tweets.

Stop words do not add much information to the text, so their frequency could bias ML models or any lexicon-based sentiment extractors, therefore, they are usually removed from the dataset. *Lemmatization* and *Porter Stemmer* are common ways to extract the core meaning from the words, thus these techniques were also applied to the datasets (Müller and Guido, 2016), which also contribute to reduce the amount of noise and data to be processed. Another reason why these techniques were adopted in this project was to compare how the sentiment extractor will classify the different versions of the tweets.

For each version of the cleaned tweets, it was calculated the sentiment using Python TextBlob library (Go, Bhayani and Huang, 2009), where the possible polarities are positive, neutral, and negative. The original tweets sentiment was not calculated. The count of sentiment for each processed version of the tweet’s dataset is shown in Table 1, where the column names are explained as follows:

* tweet\_str, sent\_str: Tweets after the removal of special characters and their sentiment.
* tweet\_clr, sent\_clr: Tweets after the removal of special characters and stop words and their sentiment.
* tweet\_st, sent\_st: Tweets after the removal of special characters, stop words and application of Porter Stemmer followed by their sentiment.
* tweet\_lm, sent\_lm: Tweets after the removal of special characters, stop words and application of Lemmatizer followed by their sentiment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Sentiment*** | ***sent\_str*** | ***sent\_clr*** | ***sent\_st*** | ***sent\_lm*** |
| ***negative*** | 58,350 | 59,894 | 48,908 | 63,110 |
| ***neutral*** | 187,658 | 203,118 | 230,836 | 200,654 |
| ***positive*** | 118,625 | 101,621 | 84,889 | 100,869 |

Table 1- Sentiment count per tweet version

Overall, the different text processing techniques applied to the raw tweets did not change the big picture, which shows more neutral comments than negatives or positives. It is important to notice that this sentiment analysis alone is not able to explain what exactly Twitter users were neutral about in this dataset, as it is not possible to guarantee that all tweets are making judgements about the war, for example.

While this dataset only contains tweets about the conflict, various subjects related to it are discussed on social media, which makes it hard to extract more precise information about the users’ feelings or opinions regarding such a complex subject. This observation is further supported by the word cloud generated for each version of the processed tweets, as shown in the figures below, where the word clouds show different subjects on the war topic, such as the role of media, misinformation, among others.

|  |  |
| --- | --- |
| Figure 1- Word cloud for tweet\_str | Figure 2- Word cloud for tweet\_clr |
| Figure 3- Word cloud for tweet\_st | Figure 4- Word cloud for tweet\_lm |

From the technical perspective, the different text processing approaches have an impact on the result of the lexical sentiment extraction, as the numbers show some tweets being classified differently, but in this dataset the overall feeling was kept consistent based on the sentiment counts regardless of the text processing technique used.

## Data Preparation for Time-Series Forecasting

The final dataset contains the processed tweets and the synthetic timestamp. At this stage, the dataset was also verified for null and missing values, before being configured for a time-series. For the experiments in this report, only the *sent\_clr* sentiment values were used, which means that only the sentiment associated with the tweets after the removal of stop words and special characters were considered for the time-series forecasting. The same experiment could be replicated for the other versions, but since their polarity does not change overall, as explained in the previous section, only one version was evaluated.

As part of this step, the approach taken was to calculate the average polarity of the tweets per day, so that the time-series had a daily frequency. This strategy was adopted because of the purpose of the analysis, which is to forecast the sentiment of users in the next 7, 30 and 90 days, hence the data had to be aggregated. The generated time-series is shown in Figure 5, where the highlighted last 30 days of the year were used as predictors for the forecaster, which is explained in more detail in Section 5.

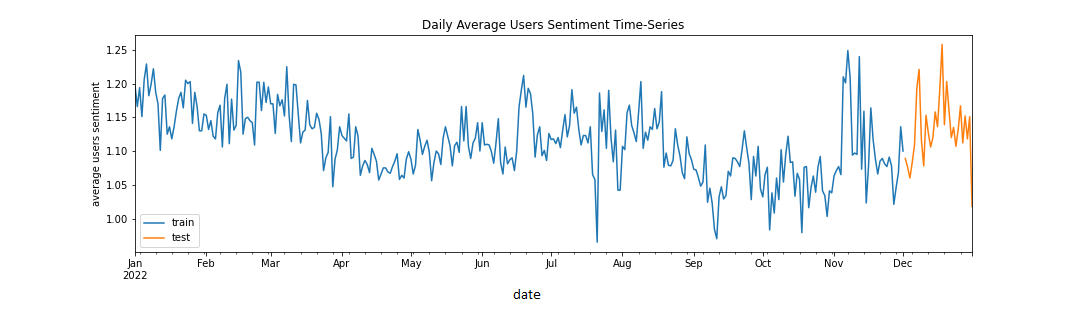


Figure 5- Average user's sentiment in 2022 (Daily)

# Data Storage Strategies

As part of the big data analysis proposed for this project, the dataset was stored in a Hadoop distributed environment and then transferred to MongoDB to demonstrate the application of a NoSQL database in the context of this analysis. Moreover, YCSB benchmarking tool was used to compare the performance of MySQL with MongoDB. This experiment was executed in the VM, and the steps were implemented in section 3 of the Jupyter notebook accompanying this report. Below it is presented a summary of the tasks executed.

## Hadoop Setup

In this step, the original dataset was uploaded to a Hadoop environment configured in the VM. The following screenshot shows how the HFS was initialized.

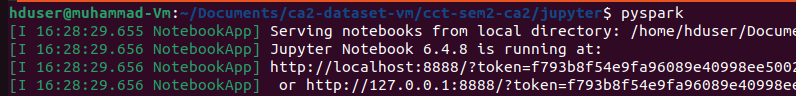
A screenshot of a computer program

Description automatically generated with medium confidence

Using the Hadoop command line interface, the datasets were uploaded to the filesystem, as shown in the screenshot below, where the datasets are stored under user1.



In section 3.1 of the Jupyter notebook, it is demonstrated how the dataset was manipulated using PySpark API, where the dataset was loaded into main memory to be transferred to MongoDB, as part of the next experiment. Some queries were also executed to demonstrate how data can be handled from Hadoop in Python. The screenshot below shows PySpark being initialized in the VM.



## MongoDB

In section 3.2 of the Jupyter notebook, the code to transfer the dataset from Hadoop to MongoDB was implemented. For this demonstration, Mongo database was first initialized as shown in the screenshot below.

A screen shot of a computer screen

Description automatically generated with low confidence

The screenshot below shows the databases and collections stored initially in the local environment. For this experiment, the database name is cct-ca2 and the collection where the tweets were stored is called twitter.

A screenshot of a computer

Description automatically generated

The database was removed so the experiment could be restarted without issues.



As part of the commands implemented in section 3.2 of the Jupyter notebook, a sample of the original dataset containing 1,000 rows were selected from the data frame extracted from Hadoop. The reason for the sampling was performance issues, as more rows would take too much time to process. For this operation, the MongoClient Python library was used, which received a dictionary from the data frame obtained using PySpark. As a result, a collection called twitter was recreated in the database as shown in the screenshots below.



A computer screen shot of white text

Description automatically generated with low confidence

## YCSB Benchmarking

In this step, Mongo and MySQL databases were compared using a benchmarking tool called YCSB. For this experiment, three workloads were prepared with different proportions of data reading and updating, as shown in Table 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Workload** | **#Operations** | **Read Proportion** | **Update Proportion** |
| workloada | 1000 | 50% | 50% |
| workloadb | 1000 | 95% | 5% |
| workloadc | 1000 | 100% | 0% |

Table 2 - Workloads description

The experiment was performed in the VM with MongoDB and MySQL up and running. The experiment was executed 5 times using the bash scripts implemented to facilitate and automate the process. The reason to run the tests several times was to avoid outliers in the metrics. In fact, for a more elaborated benchmarking strategy, factors like cache and other processes running in the machine during the performance tests should be considered. However, for the sake of simplicity, the approach taken in this project was to calculate the average metrics of the 5 executions, without further precautions.

In the section 3.3 of the Jupyter notebook accompanying this report, it is shown how the resulting csv file was processed to get the average metrics of the executions. The graphs below illustrate the results for the Runtime, Throughput and Average Latency metrics only. The x-axis represents each workload executed in Mongo or MySQL database, while the y-axis represents the average metric calculated for each test.

|  |  |
| --- | --- |
| Figure 6- Runtime metric | A screenshot of a computer screen  Description automatically generated with low confidence  Figure 7- Throughput metric |
| Figure 8- Latency metric |  |

These results show how fast a NoSQL database, such as MongoDB, can be when compared to a relational database like MySQL. For all the metrics demonstrated in this report, MongoDB performed better than MySQL. For instance, MySQL runtime, which indicates how long each workload took to complete in each database, is almost 600% higher than the MongoDB one. It is also possible to see that MongoDB performed better in terms of throughput, a metric that indicates the number of operations that can be processed per second, which is much higher for the NoSQL database. In terms of latency, MongoDB also performed better as this result indicates that the database does not take too much time to respond or complete an operation as compared to MySQL.

Although the results indicate that MongoDB can be very efficient, it doesn’t mean that MySQL is a poor system. The performance differences can be explained by the specificities of each database, for example, in a relational database, several checks and operations are executed to guarantee the ACID properties, which makes this type of system very robust and reliable, hence more expensive to run. On the other hand, NoSQL databases like MongoDB organise the data in a much simpler way, which means some qualities of a database, like consistency, are not completely guaranteed, resulting in a lighter and faster system. The decision on which one is better will ultimately depend on the requirements of the application.

# Sentiment Time-Series Forecasting

In this step, a recursive multi-step forecasting approach was used to project the average daily user's sentiment at 1 week, 1 month and 3 months going forward. The implementation of the time-series forecasting discussed in this section was implemented in section 4.5 of the accompanying Jupyter notebook.

To make the predictions, a forecaster object was created using Random Forest Regressor algorithm, which is basically an ensemble of decision trees that make predictions based on a series of binary choices according to the input features. The regressor instance used the previous 30 days as predictors to start the model and the forecast of the next 7, 30 and 90 days were calculated using the predict method of the algorithm. The results are depicted in Figure 9, Figure 10 and Figure 11 respectively.

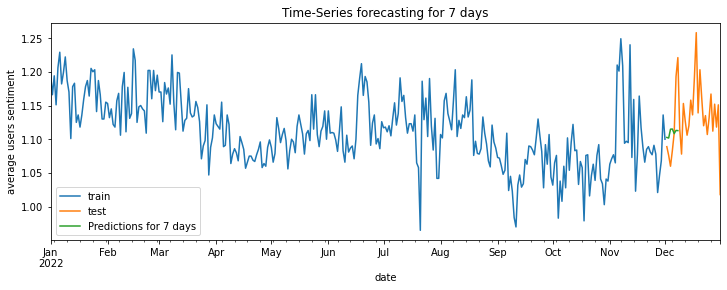


Figure 9 - Time-series prediction for the next 7 days

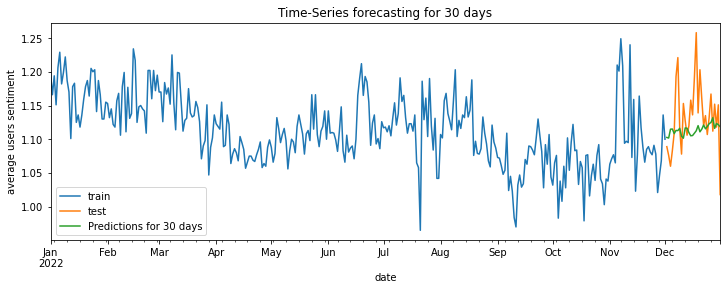


Figure 10 - Time-series prediction for the next 30 days

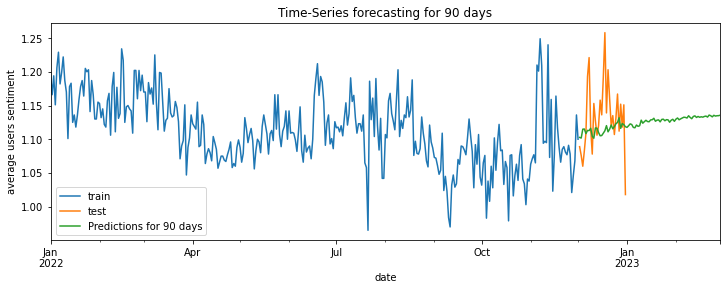


Figure 11 - Time-series prediction for the next 90 days

By following this approach, the mean squared error for each of the predictions were only 0.0024, 0.0028 and 0.0038 respectively, which suggests that the model predictions are closer to the actual values, indicating better accuracy. More experiments could have been made using different start values, but 30 days provided a satisfactory result, so it was used in this experiment. Regarding the model used, it was picked due to its simplicity to implement, so it worked as a starting point to understand the capabilities of time-series forecasting in this project.

Moreover, based on these results and by analysing the average sentiment of users in the time-series, the tendency is that the user’s feelings about the war are likely to remain in the neutral zone for the time window selected. The reason for that apparent stability can be the fact that the original dataset does not cover a long period of time, as the raw tweets were obtained from just two days and further adapted for this project. This way, converting the data back to its original time window, it is reasonable to assume that the user’s overall sentiment about the conflict in Ukraine did not change as such.

The results of the predictions can also be visualized using the dynamic dashboard implemented in section 4.6 of the accompanying Jupyter notebook. The dashboard was implemented in TkInter Python library and contains a dropdown that allows the user to select which forecasting they want to see, as illustrated in Figure 12. The reason why TKInter approach was selected for the dashboard was its flexibility, which allows the dashboard developer to create basically any functionality as in an UI software.

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure 12- Dynamic dashboard screenshot

# Conclusions

In this project, big data and ML techniques were applied to a Twitter dataset for sentiment analysis and time-series predictions. In the big data experiments, it was demonstrated how HFS, and a NoSQL database can work together to store the dataset used in this project. Moreover, benchmarking techniques were applied to compare the performance of MongoDB and MySQL when dealing with the same workloads, which required the implementation of bash scripts to automate the performance tests and to parse the results. This experiment showed the hight performance that a NoSQL database can achieve when compared to a relational database, not forgetting that each type of database has its own advantages and disadvantages, so that the decision on which one to use is always dependent on the application requirements.

For the time-series sentiment forecasting, the results of the Twitter dataset about the war in Ukraine showed that most of the users represented in that dataset were expressing a so-called neutral sentiment. However, caution is required when analysing this result as many sub-topics could be discussed around the topic in the dataset, which could affect the polarity calculated by the lexicon-based algorithms. In fact, this was verified by the word clouds extracted from the dataset, which showed terms related to several topics around the conflict main theme. Finally, different forecasting was performed on the time-series data, which showed that the tendency of the average daily sentiment of users would remain in the neutral zone for the short-term.

# References

Go, A., Bhayani, R. and Huang, L. (2009) ‘Twitter Sentiment Classiﬁcation using Distant Supervision’.

Müller, A.C. and Guido, S. (2016) *Introduction to machine learning with Python: a guide for data scientists*. First edition. Sebastopol, CA: O’Reilly Media, Inc.

White, T. (2012) *Hadoop: the definitive guide*. Third edition. Beijing: O’Reilly.