Student number: SBS23060

Assignment title: *MSC\_DA\_BD\_ADAv8 SB+*

*Date: 19.05.24*

*Word Count: 2,806*

**Introduction**

Both big data storage and processing and advanced data analytics are at the forefront of technological development in terms of science, business and the economy. Advanced analytics are used to predict future values and identify fraud among other use cases. The applications for these capabilities and use cases are wide-ranging. The applicability of analytics to problem solving as well as the vast growth in data generated has led to the growth of the big data storage and processing field. Large language models along with other large datasets have the ability to provide unparalleled insights to organisations, prompting the need for solutions to accommodate such large quantities of data which outweigh the capability of most individual machines. This is where distributed computing has come in to enable the storage and processing of vast quantities of data, opening up an entire field on optimal technologies to accommodate the big data needs of different organisations. This project focuses on text data in the form of tweets in order to gain insight into the sentiments of this textual data, along with predicting the values of this sentiment data which is expressed as numerical data. Big data technologies will be used as part of the project to demonstrate appropriate storage and processing architectures along with a comparison of performance between two databases, one SQL and one NoSQL database.

**Big Data Storage and Processing (BDSP)**

**Environment**

The data was first reduced to allow the file to load into the distributed environment. The BDSP activities were carried out in an Oracle virtual machine (VM) and an Ubuntu operating system. This combination provides a variety of benefits. Firstly, the isolation it provides allows for sandboxing of activities, in order to isolate any potential issues which may arise, protecting the local machine. Secondly, Ubuntu is a free open source operating system, which provides accessibility for students and reducing costs. Thirdly, Ubuntu runs on multiple host operating systems, increasing its accessibility to students and facilitating reproducibility and comparability of procedures across different practitioners.

**Architecture**

The below diagram displays the architecture and flow of the data that was explored in this project.

Figure 1: Big Data Storage & Process Architecture

The proposed architecture includes:

1. Storing the data in a Mongo DB database,
2. Utilising the Hadoop distributed file system for distributed computing capability
3. Apache spark for processing the data with Pyspark
4. Returning the processed data to Mongo DB via Hadoop and local VM
5. **Mongo DB**

Mongo DB as a NoSQL database was selected due to the benefits it provides (Parker, Poe & Vrbsky 2013) such as scalability through adding more servers to handle larger amounts of data as well as sharding to increase performance while working with a larger distributed dataset. Another benefit of Mongo DB includes its flexibility through schema-less designs, allowing developers to adapt the database to their specific project needs as well as its document-oriented format, supporting multiple datatypes and use cases. Finally Mongo DB’s rich query language allows for complex queries to be written to the database, supporting the needs of large scale complex operations and projects. In terms of trade offs, Mongo DB’s rich features can result in high overheads for storage of large scale long term projects, given the short term nature of this project this particular trade-off was deemed appropriate. Separately, the theory of CAP theorem applies to Mongo DB which prioritises availability and partition tolerance of consistency. This limitation may also affect particular use cases where consistency is paramount such as transactional systems. Given the analytical nature of this limitation was deemed minor.

1. **Hadoop/Yarn**

Hadoop/Yarn was chose as a distributed file system and cluster manager due to the distributed nature of this project and Hadoop/Yarn’s ability to handle distributed workloads for flexibility and accessibility as well as scalability for big data projects through thousands of nodes which can be deployed (Ma, Zhao & Zhao 2023). The ResourceManager feature provides high availability for the system also, aligning with MongoDB and the demands of modern big data projects. One tradeoff associated with Hadoop/Yarn is the operational overhead associated with maintaining a Yarn cluster such as monitoring, tuning and troubleshooting.

1. **Apache Spark (Pyspark)**

Apache Spark was selected for processing the data for a variety of reasons. One reason includes the in-memory computing capability which speeds up processing times, a prominent factor to consider when processing data along with the high performance of batch and streaming data which comes from the fast execution. The fault tolerance provided by resilient distributed datasets (RDDs) mitigates the risk of data loss and increases recoverability of the data. Also, the integration of Pyspark opens up the opportunity to engage with Spark in a python-friendly interface closely aligning to the familiarity of many practitioners given the popularity of the python programming language in data analytics. Latency can be an issue with Spark since it is not always real-time, which has ramifications for projects requiring real-time streaming and availability (Guo, Zhang, Jiang, et al. 2021). This project was not deemed to require real-time streaming to a major degree which is why this risk could be overlooked. There is also a serialization overhead with Pyspark as Python interacts with Java Virtual Machine (JVM), which can affect performance which was observed during this project.

**a.** **Hadoop Distributed File System (HDFS)**

Since HDFS interacts with Pyspark and stores the data processed through Spark, HDFS was deemed a natural fit to transport the data out of Spark and back toward its storage destination. Aforementioned benefits such as horizontal scaling and fault tolerance are additional benefits of utilising HDFS to transport data out of Spark.

**b. Local Virtual Machine**

The local virtual machine was also utilised for transporting the data back from Spark and into its destination in Mongo DB. The local environment provides an accessible interaction point to perform MongoImport operations from, allows an ease of use interaction for the practitioner in moving files into Mongo DB. Utilizing the local virtual machine allows for convenient exporting of the file where it can be placed on the local machine for further analysis also.

1. **Mongo DB**

Mongo DB was chosen as the database to store the relevant file due to the benefits outlined previously around scalability, flexibility and rich query language as well as sharding to increase performance.

**Steps Taken**

Screenshots are shared separately of the steps taken and outputs as part of the data storage and processing activities, but can broadly be outlined below:

1. Import sentiment\_analysis.csv file into ‘Test’ database and ‘sentiment’ collections for MongoDB
2. Checking the data loaded correctly in the ‘test’ database within MongoDB shell
3. Moving sentiment\_analysis.csv file into Lab04 directory for apache spark processing
4. Starting up dfs.sh and yarn.sh for cluster mode
5. Ensuring appropriate ipynb file and csv file are in HDFS for processing activity
6. Launching pyspark
7. Confirming yarn mode is running from within pyspark
8. Reading in and displaying csv file, shortened file to stay within VM capacity
9. Counting number of lines in file
10. Processing text data and then displaying word counts in descending order
11. Export data out to dfs
12. Copying sentiment\_pyspark.csv to local folder
13. Merged the split dataframe into one file ‘merged\_sentiment\_pyspark.csv’
14. Imported merged\_sentiment\_pyspark.csv file to MongoDB
15. Successfully imported, initial query to confirm

**SQL vs NoSQL Database Comparison**

Two databases were selected for performance comparison. The NoSQL database selected was Mongo DB due to the benefits previously outlined in this project regarding its capabilities. The steps and results are shared separately, with analysis of results shown below.

**Yahoo Cloud Serving Benchmark (YCSB)**

YCSB was selected as the tool to conduct this comparison due to its standardized approach which provides like for like comparisons across databases and their capabilities. Since this tool is widely accepted its widespread use also increases its strength as a reference point in comparing databases’ performance. It can also be utilised in both local and cloud environments which provides the flexibility to compare databases across widespread implementations. One limitation which must be noted of YCSB is its focus on NoSQL databases over SQL databases. This can result in YCSB not testing the full extent of relational databases’ capabilities such as strong consistency vs eventual consistency of data, which leads to bias in favour of NoSQL models. This bias must be considered when evaluating results which supports the extensibility capability it possesses to address any gaps (Cooper, Silberstein, Tam, et al. 2010).

**Database Results**

A couple of major metrics are examined below, however a more comprehensive list of results are available separate to this report.



Table 1: YCSB Database Results

The throughput operation was far superior on Mongo DB compared with MySQL. Mongo DB’s document oriented structure can provide more flexibility with schema-less design which can result in more efficient throughput compared with MySQL which may require joins and multiple tables when dealing with different data structures (Győrödi, Győrödi, Pecherle, et al. 2015). MySQL has outperformed Mongo DB in the majority of workloads on average latency. This may be as a result of MySQL’s transactional ACID model which is mature and possesses strong transaction processing capabilities.

**Advanced Data Analytics**

**Exploratory Data Analysis (EDA)**

As part of the EDA of the imported dataset, the timezone characters of the date column were removed as they were upsetting attempts to transform the column values into datetime values as well as epoch values. It was necessary to transform this field to datetime values in preparing the data for the analysis which was to come later in the project. The data was then sorted by time which is a prerequisite for timeseries analysis, in order for the analysis to be completed affectively. In order to avoid regular contributors skewing the sentiment of the dataset, duplicate user entries were removed from the dataset in order to obtain unique user tweets only. This approach was seen as more desirable to maintain an even spread of individual’s views as opposed to certain cohorts of subjects saturating and skewing the sentiment models. Given this was timeseries analysis and not panel data, unnecessary columns were removed from the dataset including ‘ids’, ‘flag’ and ‘user’ which were not considered relevant to gauging sentiment distribution values over a timeseries. Given textual data was to be analysed the next step was to engage in text preprocessing, which improves text classification (HaCohen-Kerner, Miller & Yigal 2020). This is required in order to present the data in a manner which is compatible with the analytical models that were to follow as part of the analysis. Operations performed as part of this preprocessing include removing user mentions, which contain no value toward the sentiment model. Any word containing ‘http’ was removed also as these were deemed to be links which offer no linguistic value to the models also. Removing of special characters was employed for the same reason along with tokenizing the text in order to separate the body of text into individual words which can be analysed. Stopwords were then removed to further reduce noise in the text as well as lemmatization being performed on the data to reduce words to their meaningful root, in order to preserve signal and remove noise from the data.

**Sentiment Analysis**

Vader Sentiment Analyzer was chosen as the tool to perform the initial sentiment analysis. Vader was selected because of the benefits it brings such as ability to process negation in sentences as well capture amplifiers and diminishing words, which affect the sentiment score. Vader is also a popular tool for sentiment analysis and has been built upon in the literature (Barik & Misra 2024). These inputs to the Vader analyzer are output into an aggregated compound sentiment score between negative one and positive one, allowing flexibility to the practitioner to consider which threshold would constitute positive, negative and neutral sentiments. Vader is also computationally fast and lightweight, due to it’s rule based lexicon on which it’s pretrained. Vader is also computationally less expensive than machine learning approaches to sentiment analysis. Due to the inconsistency of the data inputs within the dataset, it was necessary to average the sentiment scores per week in order to be able to establish a consistent pattern which can be analysed and represented visually. Matlotlib library was utilised to visualize the sentiment of the tweets over time, with labels and titles boldened and increased in size in order to create contrast and improve the readability of the line graph. The graph shows a decline in sentiment in late June/early July on the final two weeks of the time period examined.

A graph showing the average sentiment

Description automatically generated

Figure 2. Weekly Average sentiment of tweets

Attempts were also made to display the missing values and sentiments over time, to give an idea of the imputation or other methods which may be required for forecasting models which are to follow. These graphs were built using matplotlib and also include horizontal threshold lines in one of the graphs to represent the different sentiments between positive, neutral and negative with -0.05 <= 0.5 the threshold decided upon due to general practise. Forward and back filling were employed to fill in empty datapoints which may upset forecasting models at a later stage in the project.

**ARMA Forecasting**

ARMA was selected as a baseline forecasting model against which more advanced models could be compared against in forecasting future sentiment numerical values. ARMA models are capable of capturing short term dependencies through its autoregressive nature as well as long term dependencies through the moving average, this flexibility renders it suitable in providing baseline forecasts. The ARMA model produced 0.158 as a consistent prediction spanning across the 1st, 3rd and 7th days going forward. The lack of variability in the output lay grounds of suspicion and perhaps the limitations of the autoregressive approach of ARMA and also ARIMA, which does not take into account the seasonality of the data when making predictions. Due to the suspicions of this output timeseries cross validation (TSCV) was performed in order to validate the output. TSCV is specialised in evaluating the performance of timeseries forecasting models, preserving temporal order as well as capturing seasonality which ARMA models lack. The inconsistency of the results produced by the cross validation suggests caution which must be exercised over the reliability of the predictions made by the model. A source of improvement for this forecasting model would be to perform a more thorough analysis pre-forecasting such as decomposition of the timeseries in terms of seasonality, detrending and making the data stationary if necessary which can prove productive in improving autoregressive models (WEST 1997).

**Long Short Term Memory (LSTM) Network**

An LSTM network was employed due to its ability to capture temporally dependencies in timeseries data and its ability to mitigate the vanishing gradient problem which is often observed in recurrent neural networks (RNNs) (Hochreiter & Schmidhuber 1997). The data structure was transformed into an array in order to be compatible with the neural network model. Training data was converted to numpy arrays in order to comply with the neural network as well transforming its dimensionality to 3D which is an additional requirement. A baseline keras model was initially created with lower number parameters, which can be tuned in later iterations of the model. The results will be measured with mean squared error which is a widely used metric to assess for regression tasks (Hodson 2022). The results of iterations with hyperparameter tuning are detailed in the tables below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layers | Batchsize | Epochs | Lags | RMSE |
| 5 neurons | 5 | 5 | 40 | 0.0522 |
| 3 neurons |  |  |  |  |
| 2 neurons |  |  |  |  |
| 1 neuron |  |  |  |  |

Table 2A: First LSTM result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layers | Batchsize | Epochs | Lags | RMSE |
| 10 neurons | 10 | 10 | 40 | 0.0605 |
| 5 neurons |  |  |  |  |
| 4 neurons |  |  |  |  |
| 2 neurons |  |  |  |  |
| 1 neuron |  |  |  |  |

Table 2B: 2nd LSTM result

Given the rate at which the loss decreased with each epoch, it was decided an increase in the number of epochs would bring superior results to the predictions of the model. In addition, an additional layer as well as additional neurons and an increased batchsize was explored with in an attempt to improve the prediction scores in terms of RMSE. It was found that the baseline model outperformed the second model which underwent hyperparameter tuning. The superior performance of the baseline model was also evident in the loss function evident in the training of the model. It’s possible the increase in batchsize may have decreased the amount of recurrence which can occur in the model, which can diminish the ability of the model to exploit its short term memory function and capture temporal dependencies in the data. Improvements could be made to the model in terms of lag exploration which can affect the number of observations fed into the model and well as further exploration around numbers of layers and neurons in the network. Cross validation could also validate the credibility of the RMSE result.

**Conclusion**

This project aimed to store and process big data in a distributed environment, including HDFS and YARN resource manager as well as processing the data in Apache Spark with the use of Pyspark along with initial and final storage in Mongo DB due to its NoSQL capabilities. The dataset was explored and analysed for sentiment using Vader Sentiment Analyzer which found the data to become negative in the final weeks of the time period. ARMA forecasting was performed on the dataset which produced consistent predictions for the 1st, 3rd and 7th day going forward but were proven unreliable through the use of cross-validation. Finally the dataset fed into an LSTM network which trained on the model and produced RMSE results on the predictions of the period going forward which included the 1st, 3rd and 7th day going forward. After hyperparameter tuning it was found the original baseline LSTM network displayed superior performance with smaller numbers detailed in the parameters of the model.

**Bibliography**

Barik, K. and Misra, S. (2024) Analysis of customer reviews with an improved VADER lexicon classifier. *Journal of Big Data*. [Online] 11 (1), 1–29. Available at: doi:10.1186/s40537-023-00861-x.

Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., et al. (2010) Benchmarking cloud serving systems with YCSB. In: *Proceedings of the 1st ACM symposium on Cloud computing*. SoCC ’10. [Online]. 10 June 2010 New York, NY, USA, Association for Computing Machinery. pp. 143–154. Available at: doi:10.1145/1807128.1807152 (Accessed: 19 May 2024).

Guo, Y., Zhang, Z., Jiang, J., Wu, W., et al. (2021) Model averaging in distributed machine learning: a case study with Apache Spark. *VLDB Journal International Journal on Very Large Data Bases*. [Online] 30 (4), 693–712. Available at: doi:10.1007/s00778-021-00664-7.

Győrödi, C., Győrödi, R., Pecherle, G. and Olah, A. (2015) A comparative study: MongoDB vs. MySQL. In: *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. [Online]. June 2015 pp. 1–6. Available at: doi:10.1109/EMES.2015.7158433 (Accessed: 19 May 2024).

HaCohen-Kerner, Y., Miller, D. and Yigal, Y. (2020) The influence of preprocessing on text classification using a bag-of-words representation. *PLoS ONE*. [Online] 15 (5), e0232525–e0232525. Available at: doi:10.1371/journal.pone.0232525.

Hochreiter, S. and Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Computation*. [Online] 9 (8), 1735–1780. Available at: doi:10.1162/neco.1997.9.8.1735 (Accessed: 14 April 2024).

Hodson, T.O. (2022) Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not. *Geoscientific Model Development*. [Online] 15 (14), 5481–5487. Available at: doi:10.5194/gmd-15-5481-2022 (Accessed: 19 May 2024).

Ma, C., Zhao, M. and Zhao, Y. (2023) An overview of Hadoop applications in transportation big data. *Journal of Traffic and Transportation Engineering (English Edition)*. [Online] 10 (5), 900–917. Available at: doi:10.1016/j.jtte.2023.05.003 (Accessed: 19 May 2024).

Parker, Z., Poe, S. and Vrbsky, S.V. (2013) Comparing NoSQL MongoDB to an SQL DB. In: *Proceedings of the 51st ACM Southeast Conference*. ACMSE ’13. [Online]. 4 April 2013 New York, NY, USA, Association for Computing Machinery. pp. 1–6. Available at: doi:10.1145/2498328.2500047 (Accessed: 19 May 2024).

WEST, M. (1997) Time series decomposition. *Biometrika*. [Online] 84 (2), 489–494. Available at: doi:10.1093/biomet/84.2.489 (Accessed: 19 May 2024).