|  |  |  |  |
| --- | --- | --- | --- |
| **Programme Title:** | *MSc in Data Analytics* | | |
| **Cohort:** | *MSc in Data Analytics FT* | | |
| **Module Title(s)**: | *Advanced Data Analytics*  *Big Data Storage and Processing* | | |
| **Assignment Type:** | *Individual* | **Weighting(s)**: | *Advanced Data Analytics – 60%*  *Big Data Storage and Processing – 60%* |
| **Assignment Title:** | *MSC\_DA\_BD\_ADAv8 FT* | | |
| **Issue Date:** | *23/04/2024* | | |
| **Due Date:** | *19/05/2024* | | |
| **Late Submission Penalty:** | Late submissions will be accepted up to **5** calendar days after the deadline. All late submissions are subject to a penalty of **10%** of the mark awarded.  Submissions received more than 5 calendar days after the deadline above **will not** be accepted and a mark of 0% will be awarded. | | |
| **Method of Submission:** | **Moodle**  **Use the submission link on the Advanced Data Analytics Module page** | | |
| **Instructions for Submission:** | ***Please do not ZIP your files. ALL files must be uploaded individually (to a maximum of 20 files)***  *Expected files : Written report (word document only, NO PDF’s) ,Code files (Jupyter notebook (.ipynb) ONLY, NO PYTHON FILES), Data Files, Screencast for practical demonstration. Note that the maximum number of Jupyter Notebooks is 4* | | |
| **Feedback Method:** | **Results posted in Moodle gradebook** | | |
| **Feedback Date:** | *After exam board May/June 2024* | | |

Big data part:

1. and 3)

Data Storage and Processing Activities

In our data storage and processing activities, we leveraged the Hadoop Distributed File System (HDFS) to manage our data efficiently. HDFS was chosen for its scalability, fault tolerance, and ability to handle large datasets, making it an ideal choice for our needs. Initially, data was stored in HDFS and later inserted into a MySQL database using code executed within a Jupyter notebook. This setup facilitated seamless integration between storage and processing layers, allowing for efficient data management and retrieval.

For processing the data, we utilized Apache Spark with PySpark. The choice of Spark was driven by its powerful in-memory computing capabilities, which significantly enhance processing speed for large datasets compared to traditional disk-based processing. PySpark, the Python API for Spark, was employed to take advantage of Python's simplicity and extensive libraries. Due to disk memory constraints on our virtual machine running Ubuntu in UTM, we processed the data using Spark and stored the processed data back into HDFS. This approach ensured that the data was ready for subsequent forecast activities without exceeding the limited disk space available on the virtual machine.

Rationale and Justification

Our decision to use HDFS for data storage was driven by its robustness and ability to handle large volumes of data efficiently. HDFS provides high throughput access to data and is designed to scale out with the addition of commodity hardware, making it cost-effective. The successful integration with MySQL via a Jupyter notebook underscored HDFS's compatibility with various data management tools, enabling smooth data insertion and retrieval processes.

The need to employ Apache Spark for data processing arose from the limitations of the virtual machine's disk memory. Spark's in-memory processing capabilities allowed us to process large datasets without overwhelming the limited storage capacity. By using PySpark, we combined Spark's performance advantages with Python's ease of use and extensive ecosystem, facilitating a streamlined data processing workflow. The processed data was then saved back into HDFS, ensuring that it was accessible and ready for further analysis and forecasting tasks.

In the second Jupyter notebook, we opted to read the dataset from MongoDB Atlas, a cloud-based NoSQL database service. This choice was influenced by MongoDB's flexibility in handling unstructured data and its scalability features. We utilized Google Drive to import the CSV file created in HDFS with PySpark, ensuring a smooth data transfer process. This workflow allowed us to read the data from MongoDB and perform preprocessing using Pandas on a Mac M1, leveraging the powerful data manipulation capabilities of Pandas and the robust processing power of the Mac M1. This approach provided an efficient and effective way to prepare the data for further analysis, ensuring high performance and ease of use.

2)

A screenshot of a computer

Description automatically generatedFigure 1. Comparative betweeen MYSQL and Mongo (1000 rows)

A screenshot of a computer screen

Description automatically generated

Figure 2. Comparative betweeen MYSQL and Mongo (10000 rows)

A screenshot of a computer screen

Description automatically generated

Figure 3. Comparative betweeen MYSQL and Mongo (487928 rows)

Overall Observations:

Across all three datasets (1000 rows, 10,000 rows, and 487,928 rows), MongoDB consistently demonstrated better performance metrics compared to MySQL. MongoDB showed lower runtimes and higher throughput, indicating its efficiency in handling varying data volumes. This analysis delves into specific performance aspects observed in the benchmarks, highlighting key differences and patterns.

Runtime and Throughput:

MongoDB exhibited significantly lower runtimes and higher throughput across all datasets. For the smallest dataset (1000 rows), MongoDB's runtime was much shorter, and its throughput was considerably higher. As the dataset size increased to 10,000 rows and further to 487,928 rows, these differences became even more pronounced. MongoDB maintained high throughput and low runtime even with larger datasets, underscoring its efficiency in handling both small and large volumes of data.

Latency Metrics:

Latency is a critical performance metric, particularly for operations like inserts. Across all datasets, MongoDB consistently demonstrated lower average, minimum, and maximum latencies compared to MySQL. This trend was evident in the 1000 rows, 10,000 rows, and 487,928 rows datasets. The lower latencies in MongoDB indicate faster response times and more efficient processing of insert operations, contributing to its superior performance.

Garbage Collection:

MongoDB and MySQL both exhibited garbage collection (GC) activities, but the impact on performance varied. MongoDB had more frequent GC events but with minimal impact on the overall runtime. In contrast, MySQL had fewer GC events, but due to its longer runtimes, the relative impact was negligible. This trend was consistent across all datasets, showing MongoDB's better handling of garbage collection.

Insert Operations:

MongoDB handled insert operations more efficiently across all datasets, with consistently lower latencies and higher throughput. MySQL's performance, while functional, was marked by higher latencies and occasional errors in larger datasets. This was evident in both small and large datasets, highlighting MongoDB's robustness in managing insert-heavy workloads.

Cleanup Operations:

During cleanup phases, MongoDB exhibited lower latencies, indicating faster and more efficient cleanup processes. This trend was consistent across all datasets, reinforcing MongoDB's overall efficiency in handling different types of operations compared to MySQL.

Advance Data Analysis part:

1)

In the initial phase of Exploratory Data Analysis (EDA), it is essential to understand the structure and type of the data we are working with. Using PySpark, the df.printSchema() function provides a comprehensive view of the dataset’s schema, detailing the data types of each column and whether null values are allowed. This is particularly useful in identifying columns that may need further attention, such as those with complex data types or unexpected nullability. In a similar vein, when working with Pandas, the `df.dtypes` attribute offers a quick glance at the data types of each column in the DataFrame. This helps in confirming that the data is in the expected format and highlights any discrepancies that need to be addressed before further analysis. Additionally, the df.show() method is used to display the first few rows of the dataset, providing a snapshot that aids in visually inspecting the data. This can help in spotting any obvious issues like missing values, incorrect data types, or inconsistencies right from the start.

To further deepen the understanding of the dataset, the df.describe() method can be used. This method provides summary statistics for numeric columns, including count, mean, standard deviation, minimum, and maximum values, as well as quartiles. These statistics are essential for identifying the distribution and central tendencies of the data, detecting outliers, and understanding the overall spread. Combining these methods—df.printSchema(), df.dtypes, df.show(), and df.describe()—provides a robust foundation for EDA, ensuring a comprehensive initial assessment of the dataset's structure and characteristics. These steps are crucial in setting the foundation for effective data wrangling and model development.

Data wrangling is a critical step in preparing the dataset for analysis and modeling. One common issue encountered is missing dates in the time series data, which can lead to inaccurate forecasting if not handled properly. Interpolation is a powerful technique used to estimate and fill in these missing values based on existing data points. This ensures a continuous time series without gaps, which is essential for accurate sentiment analysis and forecasting.

Additionally, it was necessary to remove duplicated IDs of comments to maintain the integrity of the dataset. Duplicate entries can skew the results and lead to biased predictions, so identifying and removing them is crucial. Another important aspect of data wrangling is feature engineering, which helps to identify relevant and irrelevant columns for the analysis. For instance, the "flag" column, where all values were the same, was deemed irrelevant and removed. In contrast, columns like "date" and "text" were identified as essential for forecasting. The "date" column provides the temporal context, while the "text" column is crucial for natural language processing (NLP) to analyze sentiment.

\_id index ids date flag user text \

263571 664a0fb9e1eba02a48f04b01 index ids date flag user text

527585 664a0fc1e1eba02a48f4524f index ids date flag user text

790636 664a0fc9e1eba02a48f855da index ids date flag user text

1054110 664a0fd0e1eba02a48fc5b0c index ids date flag user text

1317457 664a0fd8e1eba02a48005fc0 index ids date flag user text

1581913 664a0fdfe1eba02a480468c8 index ids date flag user text

timestamp sentiment\_score date\_string

263571 timestamp sentiment\_score date

527585 timestamp sentiment\_score date

790636 timestamp sentiment\_score date

1054110 timestamp sentiment\_score date

1317457 timestamp sentiment\_score date

1581913 timestamp sentiment\_score date

Thanks to Exploratory Data Analysis (EDA) and data wrangling, it was possible to fix the problem with the data from MongoDB that had irregularities in data types as shown in the text box. These irregularities included inconsistent data formats and mixed data types within columns, which can cause significant issues during analysis. By thoroughly examining the dataset and applying necessary transformations, such as converting data types to ensure consistency, EDA and data wrangling helped to standardize the data. This process not only improved data quality but also facilitated more accurate and reliable forecasting and sentiment analysis.

Furthermore, the "date" column was originally in string format, which is not suitable for time series analysis. It was necessary to convert this column into a proper date type to enable accurate time-based calculations and ensure compatibility with forecasting models like ARIMA, SARIMA, and LSTM. This conversion facilitates the correct chronological ordering of data points and is essential for the effective application of time series forecasting techniques.

Additionally, it was necessary to create a new column, "sentiment\_score," which quantifies the sentiment derived from the text. This new feature is pivotal for the next steps in the analysis and modeling process, providing a numerical representation of sentiment that can be used in time series forecasting models. By performing these data wrangling steps, the dataset is transformed into a clean and structured format, ready for detailed analysis and model development.

For time series forecasting of sentiment data, selecting appropriate machine learning models is crucial. ARIMA (AutoRegressive Integrated Moving Average) is a widely used model that captures the temporal dependencies in the data, making it suitable for forecasting based on historical sentiment trends. ARIMA operates by decomposing the time series into autoregressive and moving average components while integrating differencing to make the data stationary. This model is effective in capturing linear patterns in the time series data and can provide reliable short-term forecasts for the sentiment scores.

For data with seasonal patterns, SARIMA (Seasonal ARIMA) extends ARIMA by including seasonal components, providing a more accurate forecast for data exhibiting regular fluctuations over time. The justification for using SARIMA analysis in this dataset lies in its ability to model and forecast time series data that exhibit both seasonal patterns and autocorrelation. SARIMA models are specifically designed to handle time series data with repeating patterns or cycles at fixed intervals, effectively capturing and modeling these seasonal variations. Additionally, SARIMA incorporates autoregressive and moving average components to capture the autocorrelation in the data, accounting for dependencies between past and present values. This makes SARIMA suitable for forecasting data with complex temporal structures, providing robust and reliable forecasts even in the presence of noise and irregularities.

LSTM (Long Short-Term Memory) networks, a type of recurrent neural network, are highly effective in capturing long-term dependencies in time series data. LSTMs are designed to overcome the vanishing gradient problem, enabling them to learn and remember long-term sequences. This makes LSTMs particularly suitable for handling complex patterns and non-linear relationships in the sentiment data, offering robust performance in time series forecasting tasks. LSTMs can learn from the sequential nature of the data, making them powerful for predicting future sentiment based on historical sentiment sequences.

In this project, ARIMA, SARIMA, and LSTM models were implemented and evaluated to determine the best approach for forecasting sentiment at 1 day, 3 days, and 7 days intervals. The choice of models was based on the characteristics of the dataset and the need to capture different types of dependencies and patterns in the sentiment data. By leveraging these models, we aim to generate reliable forecasts that provide valuable insights for decision-making and strategic planning.

2)

Hyperparameter tuning plays a crucial role in optimizing the performance of machine learning models, ensuring that they generalize well to unseen data and produce accurate predictions. In this project, two commonly used techniques for hyperparameter tuning were employed: Grid Search and Random Search, tailored to the specific requirements of each model.

For the ARIMA and SARIMA models, hyperparameters such as the order of autoregressive (p), integrated (d), and moving average (q) terms, as well as seasonal components, greatly influence the model's forecasting accuracy. Grid Search was chosen as one of the techniques due to its exhaustive search over a predefined grid of hyperparameters. This method systematically evaluates all combinations of hyperparameters within the specified grid and identifies the optimal set that yields the best performance. By exhaustively searching the hyperparameter space, Grid Search ensures thorough exploration but can be computationally expensive, especially for large parameter grids.

In addition to Grid Search, Random Search was employed for ARIMA and SARIMA hyperparameter tuning. Random Search randomly samples hyperparameters from predefined distributions and evaluates their performance. While not as exhaustive as Grid Search, Random Search can efficiently explore a wide range of hyperparameter combinations, often yielding comparable results with fewer iterations. This makes Random Search particularly well-suited for scenarios where computational resources are limited or when the hyperparameter space is large and complex. In this project, Random Search complemented Grid Search by providing a more efficient search strategy and helping to identify promising regions of the hyperparameter space.

For the LSTM model, which is a type of deep learning model, hyperparameter tuning is equally important for optimizing its architecture and training process. Keras Tuner, a hyperparameter optimization library for Keras, was employed with a Random Search strategy. Keras Tuner uses Bayesian optimization to efficiently search the hyperparameter space, adapting its search based on previous evaluations to focus on promising regions. This method is well-suited for deep learning models like LSTM, where the hyperparameter space is high-dimensional and nonlinear. By leveraging Keras Tuner with Random Search, optimal configurations for the LSTM model can be efficiently identified, leading to improved forecasting performance.

3)

A graph with a line

Description automatically generated

Figure 1. Sentiment score over time

Fdfdfd

fddfd

The analysis of sentiment changes over the dataset revealed several important trends and patterns, despite the presence of gaps in the data. To address these gaps, interpolation techniques were employed to estimate missing values, ensuring a continuous time series suitable for accurate sentiment analysis and forecasting.

Initially, the sentiment scores were progressing well, showing a stable trend. However, gaps in the data posed a challenge for continuous analysis. To manage this, interpolation was applied, filling in the missing dates with estimated sentiment scores based on the surrounding data points. This method ensured that the time series remained intact, allowing for reliable analysis and preventing the gaps from skewing the overall sentiment trends.

As depicted in Figure 1, the sentiment scores experienced a significant breakdown in mid-June, shifting from a generally positive or neutral trend to a negative tendency. This sudden change in sentiment required careful examination to understand the underlying causes and its implications for future forecasts. The breakdown in sentiment could be attributed to specific events, news, or social media discussions during that period, leading to a more negative sentiment among the data sources.

Given the trends observed, forecasting sentiment for the upcoming periods (1 day, 3 days, and 7 days) was crucial. The models, tuned with Grid Search and Random Search techniques, were able to account for the interpolated data and the identified negative shift. The forecasts aimed to predict how the sentiment would evolve post the mid-June breakdown, providing insights into whether the negative trend would persist or if there would be a recovery.

4)

The table below presents the forecasted sentiment scores for 1 day, 3 days, and 7 days intervals using LSTM, ARIMA, and SARIMA models. These forecasts are essential for understanding how sentiment is likely to evolve over short-term and medium-term periods, providing valuable insights for decision-making processes.

Figure 2. Table of results per day

This table clearly shows the results of sentiment forecasting for the specified dates. The LSTM model, known for its capability to capture complex patterns in time series data, provides a slight upward trend in sentiment over the 7-day period. The ARIMA model, which relies on past values and assumes a linear structure, predicts a generally negative trend in sentiment. Similarly, the SARIMA model, which extends ARIMA to account for seasonality, forecasts a stable but negative sentiment trend.

The importance of presenting these results in a well-structured table cannot be overstated. It allows for easy comparison of different models' forecasts, highlighting differences and similarities in their predictions. This structured presentation helps stakeholders to make informed decisions based on the expected sentiment trends.

Adhering to Tufts Principles in the design of the dashboard is crucial for effectively communicating the forecast results. These principles emphasize clarity, accuracy, and relevance, which are essential for creating an intuitive and informative dashboard (Tufte, 2001).

In terms of clarity, the dashboard should present information in a clear and understandable manner. This involves using clean layouts, legible fonts, and appropriate color schemes. Each chart and table should be well-labeled, and captions should provide necessary context to understand the data at a glance. Ensuring that the data is presented clearly helps users quickly grasp the key insights without confusion.

Accuracy is another vital principle. The data presented on the dashboard must be accurate and up-to-date. This requires careful handling of data, ensuring that any gaps are appropriately managed, as was done using interpolation in this project. Accurate data representation builds trust and reliability in the dashboard's insights, allowing users to confidently base their decisions on the information provided.

Relevance is crucial for maintaining the focus of the dashboard on presenting the most pertinent information to the users. It should highlight key trends, such as the significant sentiment drop in mid-June, and provide actionable insights. Interactive features, such as filters and drill-downs, can help users explore data more deeply according to their needs. By focusing on relevant information, the dashboard ensures that users are not overwhelmed with extraneous details and can concentrate on the most important aspects of the data.