**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | CA2 Integrated with Big Data, Individual Assignment 60% |
| **Assessment Title:** | Time Series analysis of tweet sentiment |
| **Lecturer Name:** | Dr. David McQuaid / Dr. Muhammad Iqbal |
| **Student Full Name:** | Dornel Duarte Rotta |
| **Student Number:** | 2023066 |
| **Assessment Due Date:** | Friday, 13 October 2023 |
| **Date of Submission:** | Friday, 10 November 2023 |
| **GitHub:** | https://github.com/2223066/CCT\_Master\_DataAnalytics\_CA3 |

****

**Declaration**

|  |
| --- |
| By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution. |

**Introduction**

Time-series forecasting is a technique that utilizes historical and current data to predict future values over a period. By analyzing data stored in the past, it is possible to make informed decisions that can guide business strategy and help understand future trends (Despot I, 2023).

Using Time-series forecasting, the main purpose of this study was to analyze the sentiment of tweets posted in the past in a time interval and predict/forecast these sentiments until 3 months ahead.

The dataset used had 1.6 million observations, the data was stored using two different databases, one relational (MySQL) and another no-relational (Apache Spark SQL), and using exploratory data analysis technics (EDA) such as data collect, data analysis, data preparation and visualization, time series algorithms were applied to forecast the possible sentiment of user’s tweets on the future.

Vader and TextBlob algorithms, which provide sentiment scores based on used words, were used to calculate the tweet sentiment.

Comparative performance benchmark between the MySQL and Apache Spark SQL was performed to analyze the differences in performance using a relational and no-relational database to process this type of data.

**Business and Data Understanding**

Business and Data Understanding are parts of the CRISP-DM Project Management Framework used to understand the business objectives and to have a deeper understanding of the raw data to be prepared.

**Business Understanding**

From the business perspective, the main objective for this work was to use past tweets collected from Twitter platform, analyze their sentiment which means classify as their semantic orientation as positive or negative, to finally run time-series algorithms to forecast future tweets sentiments.

Perform a benchmark test to analyze the performance of two different databases, one relational (MySQL) an another non-relational (Apache Spark SQL) is part of the scope as well.

**Data Understanding – Part 1**

The Data Understanding step involves accessing the dataset and exploring it to understand the data and decide which type of transformations and preparations is needed.

The dataset was loaded using Spark Apache where a temporary view table was used to store the data, then select queries was executed to have a first look on the data.

Printing the first 10 rows:

A screenshot of a computer program

Description automatically generated

Other queries to have a better understanding:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group by id | Group by seq | Query != ‘NO\_QUERY’ | Date = null or empty | Tweet = null or empty |
| Looking for duplicate values | Looking for duplicate values | Looking for a different value thant NO\_QUERY | Looking for missing date values | Looking for missing tweets values |
|  |  |  |  |  |

Taking a first look on the provided dataset it is possible to assume:

* Id and seq are unique.
* Query column has always the same value: “NO\_QUERY”
* Date and tweet columns always have values.

The columns seq and query have no use on the purpose for this work and can be deleted. Although seq is unique, this dataset already has id to be used as identity.

|  |  |
| --- | --- |
|  | The date value has a particular format with a mix of string and numeric values.  This must be formatted and split to help during the modeling. |

**Data Preparation**

Data preparation is the process of preparing raw data so that it is suitable for further processing and analysis. Key steps include collecting, cleaning, and labeling raw data, then exploring and visualizing it. (Amazon, 2023).

## Data Collection

The first step was to collect all data using Apache Spark SQL, which automatically uses HIVE as metastore. The initial data as collected using MySQL too.

A new database was created (dbTwitter) and a new table (tblTwitter) was placed on the dbTwitter database. This process was done in both Apache Spark SQL and MySQL, these databases with the exact same table structure and data will be used to benchmark performance differences.

The table tblTwitter was created on both databases using the same columns as the original CSV dataset to be used as base to do the data preparation:

* Id: Numeric
* Seq: Numeric
* Date: String
* Query: String
* User: String
* Twitter: String

All the data loaded on the temporary Apache SQL view was inserted into the new Apache Spark SQL table tblTwitter and then inserted into the MySQL table tblTwitter too.

**Why using Apache Spark SQL Hive and MySQL?**

Apache Spark SQL Hive is a No-relation database, it means that Hive is the best approach to deal with this tweet data because, for example, this data has no relational tables and will perform much better than a relational database.

MySQL is not the best option, there is only one table representing all the data, no relational columns, no necessity of using a foreign key (a reference to a primary key of another existing table) to “protect the data”.

Although MySQL not being the best option for handle this data, it was chosen to show the performance issues during this work.

## Data Cleaning

The first step of data cleaning was to remove all features which were considered useless:

* Seq: Once the dataset already has a unique column id, seq isn’t necessary.
* Query: It’s a column which has always the same value (NO\_QUERY).

As already mentioned, the MySQL database was chosen as the main database to perform all the data preparation on this work. Although it is not the best database, the idea was to take advantage of this scenario to start analyzing the performance of a relational database processing non-relational data (how much performance loss can be perceived).

## Data Preparation

On this phase the raw data was prepared, new columns were added, values were transformed, and new calculations were done.

To perform all these calculations/transformations, a new table tblTwitterPreparation was created in MySQL:

|  |  |  |
| --- | --- | --- |
| Column name | Type | Description |
| Id | Int | Id column just converted from string to int type. |
| Date | Varchar(50) | Same original string value kept as a reference. |
| User | Varchar(50) | Same original string value kept as a reference. |
| Tweet | Text | Same original string value kept being used in the sentiment analysis (where the stop words and others word transformations were done). |
| tweet\_review | Text | New text column to receive the transformed tweet text to be calculated the sentiment. |
| date\_custom | Date | New date column to receive a date time value (yyyy-mm-dd) based on the date string value. |
| Week | Char(3) | New char column to receive the week day. |
| Hour | Int | New int column to receive the hour of the tweet. May be useful to group by clauses or model calculations. |
| Minute | Int | New int column to receive the minute of the tweet. May be useful to group by clauses or model calculations. |
| Second | Int | New int column to receive the second of the tweet. May be useful to group by clauses or model calculations. |
| Day | Int | New int column to receive the day of the tweet. May be useful to group by clauses or model calculations. |
| Month | Int | New int column to receive the month of the tweet. May be useful to group by clauses or model calculations. |
| Year | Int | New int column to receive the year of the tweet. May be useful to group by clauses or model calculations. |
| sentiment\_vader | Double | New double column to receive the calculated sentiment using vader algorithm. |
| sentiment\_blob | Double | New double column to receive the calculated sentiment using textblob algorithm. |
| Index | Id | New index to help MySQL table to deliver best performance. |

**Why using a new table and why new columns?**

A new table was created to keep the original intact to be used in the benchmark performance (non-relational Spark Hive vs MySQL). Keep the original data can also be useful during the data preparation process to evaluate the transformations are correct by comparing it with the original data.

New columns can be useful during the next phases of the work, for example, having the date value split in day, month and year is easier to filter data, also its faster to filter by an int column directly instead of a substring in a date value query. There are other benefits like easier to perform group by clauses, calculations and more.

After the creation of the tblTwitterPreparation table, the next step was to insert all the data from the original tblTwitter table. The insert was done using an insert statement clause reading the data from tblTwitter.

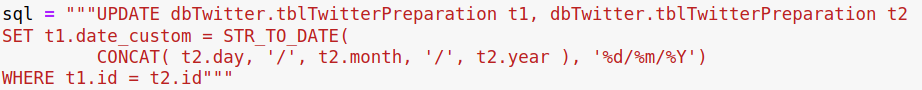
During the execution of this insert statement, the firsts transformations/calculations were done:

* Substring transformations: The original column date was split using a substring command and the values for the columns hour, minute, second, day, and year were created.
* Switch case transformation: A switch case clause was performed to convert the string month value to int: Apr = 4; May = 5; Jun = 6.
* Columns tweet, date, user just received the original value (no transformations).
* Columns tweet\_review, date\_custom, sentiment\_blob, sentiment\_vader received a null or 0 value. These columns will be calculated on the next steps.

**Date value**

As already mentioned, columns needed to be calculated, and date\_custom was one of these.

To create a new calculated value for date\_custom based on the original date column, a data cross-table update was done, this type of update statement has an implicit INNER JOIN clause, guaranteeing the same columns of each row will be updated with the appropriated value:



This in a first example how a split date value can be useful, using the previously extracted int day, month and year values, a new date value was created for each observation.

**Performance issues**

This step is a first excellent example how a relational database does not perform well using a large amount of data. This advanced query which has an implicit inner join clause plus transformations like cast, string concatenation and date conversion, took almost 20 minutes to be processed. Using Apache Spark Hive or other non-relational database certainly would improve the performance of this query.

**Sentiment calculations**

To handle with the sentiment calculations a “for each statement” was performed. The idea is to capture the tweet text value for each row, then use a function named stopwords to prepare this tweet and finally calculate the tweet sentiment to be stored.

The stopword function is a procedure which deal with the text value removing useless words like prepositions and removing punctuations. This is an important pre-step to execute the sentiment analysis on the tweet.

**Why using two different sentiment analysis?**

Two different sentiment algorithms were used to calculate the sentiment, the reason to use two options was to analyze which one is better to be used on the modeling to forecast the future sentiment. Additionally, would be interesting to compare two of the main used sentiment algorithms with a huge amount of data. Which one have better results? Which one can be better fit on the forecast analyzes?

Vader and TextBlob were the chosen libraries to deal with these sentiment calculations. Both values were calculated during each for iteration and stored using a MySQL update statement.

The tweet modeled by the stopwords functions was stored as a reference too resulting in two tweets column, the original tweet (tweet column) and the stopwords modeled tweet (tweet\_review column).

**Performance issues**

This update command was another good example of how performance can be determinant choosing the appropriate database type. Despite of being a virtual machine running on a laptop, this task took more than one hour to complete. Using a non-relational database like Apache Spark SQL Hive, this update statement would last no longer than minutes to be executed.

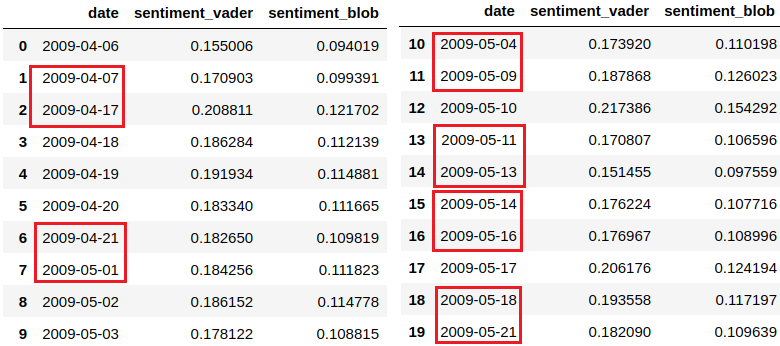
**Data Visualization**

After some important transformations being done, it was time to perform some visualizations and try to find some patterns on the data.

**Average Sentiment per day**

The first guess was to try to find some daily pattern to fit on the time series modeling.

To perform this, a select statement was executed using a group by clause to arrange the data per day and bring the average value of vader and textblob sentiment:

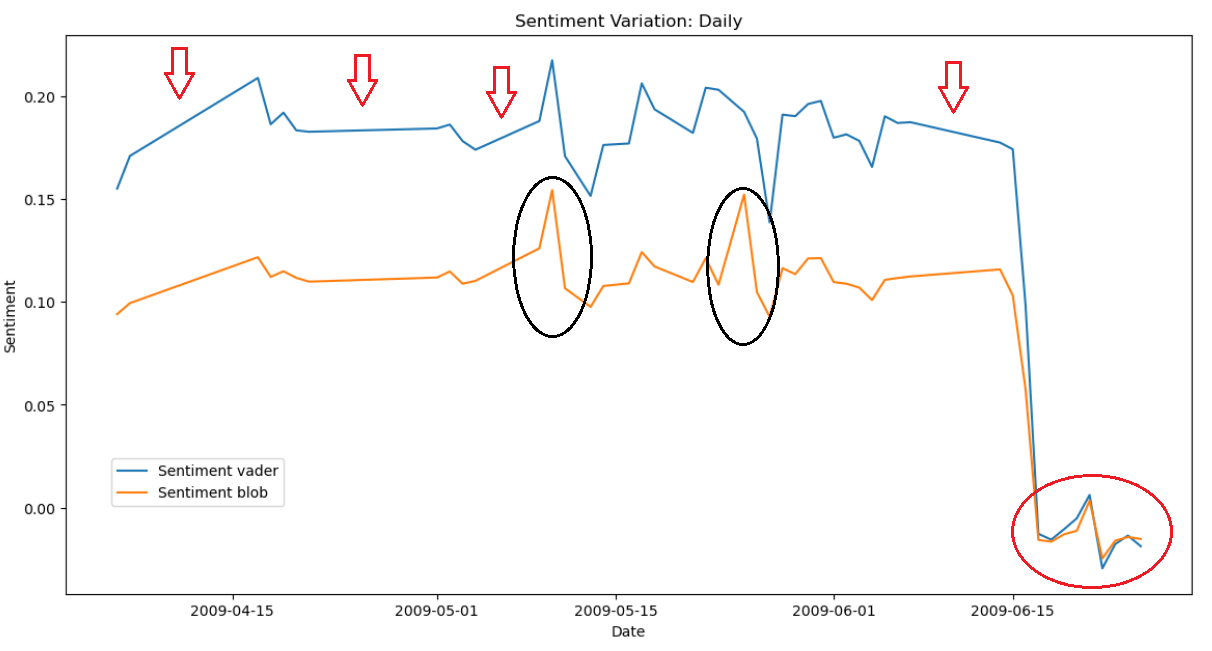


A total of 48 days was found distributed in three months (April, May and June), from 2009-04-06 to 2009-06-25, and an enormous number of days were missing:

|  |  |  |
| --- | --- | --- |
| April | May | Jun |
| 2009-04-08  2009-04-09  2009-04-10  2009-04-11  2009-04-12  2009-04-13  2009-04-14  2009-04-15  2009-04-16  2009-04-22  2009-04-23  2009-04-24  2009-04-25  2009-04-26  2009-04-27  2009-04-28  2009-04-29  2009-04-30 | 2009-05-05  2009-05-06  2009-05-07  2009-05-08  2009-05-12  2009-05-15  2009-05-19  2009-05-20  2009-05-24 | 2009-06-08  2009-06-09  2009-06-10  2009-06-11  2009-06-12  2009-06-13 |

Completing the missing values was a challenge because the dataset had 48 days and 33 missing days, totalizing a possible 81 days range, it means that considering the possible 81 days, there were around 40% of the total days missing, it’s a lot...

Plotting a graph to have more insights:



The difference between Vader and TextBlob were very similar in respect a pattern, although the difference in their values. Blob appears to have captured better a time series patterns (represented on the black circles).

It is interesting to notice the only negative values (red circle) where Vader and TextBob match almost perfectly. This may be the data noise.

On the red arrows it was possible to notice very well the big groups of missing values (entire weeks missing).

**Average Sentiment per hour**

Another possibility to find a pattern on the data is to analyze the sentiment per tweet hour.

To perform this, a select statement was executed using a group by clause to arrange the data per day/hour and bring the average value of Vader and TextBlob sentiment:

A screenshot of a calculator

Description automatically generated A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated A screenshot of a graph

Description automatically generated

In addition to having days missing from the dataset, when grouping the records by hours, it was possible to notice that practically all days in the dataset are also missing hours.

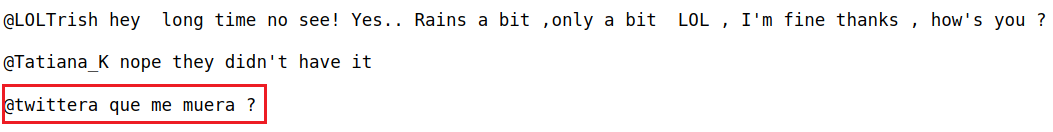
Although some hourly seasonality can be found in the data, the large lack of records can make the data unreliable.

A daily sentiment average calculation appears to be more reliable since it can reduce the inconsistency of missing data regarding hours by the day's average sentiment.

**Average Sentiment per day – excluding sentiment with zero value**

The best path to follow was using TextBlob sentiment algorithm and daily average sentiment, to verify this was analyzed a comparative of sentiment tweets using only TextBlob but excluding registers where sentiment was calculated as zero.

The idea was to remove noise because part of the tweets was useless. For example, some tweets were written in Spanish and the sentiment algorithm was prepared to analyze only English. In another example, other tweets may contain only indecipherable slang-based words.



A screenshot of a graph

Description automatically generated

Removing the zeroed sentiments did not change the pattern of values but brought the sentiment average less close to zero.

**Why daily sentiment average and Text Blob algorithm?**

Using the daily sentiment average helped in the process of correcting missing data by day and hour, for example, there was a lot of missing data per hour each day in the dataset, when averaging by day, this possible error in the data ended up being corrected when generating a daily average.

A seasoning pattern is also better noticeable when using the daily sentiment average rather than the hourly one, which made forecasting much easier.

TextBlob was the algorithm chosen because it presented a better weekly data pattern than Vader. TextBlob also presented values less close to zero on a scale of -1 to 1 in the sentiment analysis, which also helped with the forecast analysis. Finally, removing zeroed sentiment values helped to reduce noise.

**Data Preparation – Part II**

After analyzing the pattern, noise and the missing values of the dataset, the last part of the data preparation phase was to deal with that.

Basically, four different rules were applied to do that:

**Removing negative values**

The last 12 registers which represented the negative values were deleted from the dataset. It was considered bias after testing to forecast using these values. The seasonality was compromised, and these registers were considered outliers for this work.

**Including missing values based on a pattern**

Some of the missing values were solved by adding new records based on the pattern of the registers before or after.

Based on the dataset, it was possible to notice that the sentiment increases after Wednesday and decreases after Sunday, it means that Sunday generally is the highest sentiment value of the week and Wednesday is the lowest.

Then to calculate, for example, a Wednesday missing value, the average percentage decrease of the predecessors’ values of Sunday to Monday and Monday to Tuesday was applied to calculate the next average decrease and get the Wednesday value.

**Average weekday values**

Another approach was to calculate the average value of other weekdays, for example Sunday, to calculate a missing Sunday day, but respecting the average difference of the entire week. For example, if during a week the average sentiment was 1.15, the others average week was 1.12, the average Sunday must be calculated based on the rule of three to calculate the correction factor.

**Shifting an entire week**

And finally, an entire week or part of it was shifted to be closer the next one. The first April’s week is an example, where a 9 day difference could be reduced to two days by shifting these days one week ahead.

The final dataset after all these missing days added and noise removed:

A graph showing a line

Description automatically generated with medium confidence

**Modeling - Machine Learning**

## Trend and Seasonality analysis

Before starting with modeling and performing time series analysis, it was necessary to validate if the dataset had a trend and/or seasonality to be addressed:

A line graph of a graph

Description automatically generated with medium confidence

The seasonal decomposition is useful in analysis of time series affected by factors that change in time in a cyclic (periodical) manner. Running the example plots the original, trend, seasonal time series were compared. It was possible to see that the trend and seasonality information extracted from the series does seem reasonable, showing periods of high and low sentiment during the weeks of the series.

## Recursive Autoregressive Forecast

Recursive forecasting consists in creating lagged features of the target series and fitting a machine learning model on them. When forecasting further steps in the future, the predictions of the previous steps are used to create the new lagged features. Since the future values of the target are not known at the prediction time, using the previously forecasted values seems to be a good approximation of the actual values (Cerliani M, 2022).

Once the final prepared dataset to be run on this model has low number of observations, a lag forecast algorithm sounds a best choice to forecast.

The final dataset had its last preparations defining its frequency (1D) and to defining date column as an index (sorted).

The main parameters to be set were:

Steps: Is the number of steps of the dataset will be used to predict values.

Lag: How many lagged features of the target series will be created:

A blue and white squares with black text

Description automatically generated

To define the best number of steps and lags the main square error (MSE) was analyzed (lowest number is better) for different options running the algorithm:

|  |  |  |
| --- | --- | --- |
| Steps | Lags | MSE |
| 07 | 28 | 0.0002026 |
| 06 | 21 | 9.5002007 |
| 07 | 21 | 0.0001498 |
| 08 | **21** | **0.0001387** |
| 09 | 21 | 0.0001614 |
| 10 | 21 | 0.0001652 |
| 12 | 21 | 0.0001937 |
| 14 | 21 | 0.0002595 |
| 07 | 20 | 0.0002554 |
| 07 | 22 | 0.0001571 |
| 08 | 18 | 0.0002288 |
| 08 | 20 | 0.0002233 |
| 08 | 22 | 0.0001952 |
| 08 | 24 | 0.0001963 |
| 08 | 25 | 0.0002089 |
| 08 | 27 | 0.0002188 |
| 09 | 18 | 0.0002459 |
| 09 | 20 | 0.0002290 |
| 09 | 22 | 0.0001728 |
| 09 | 24 | 0.0002271 |
| 12 | 28 | 0.0003477 |
| 12 | 21 | 0.0001937 |
| 15 | 28 | 0.0005964 |
| 15 | 21 | 0.0006055 |

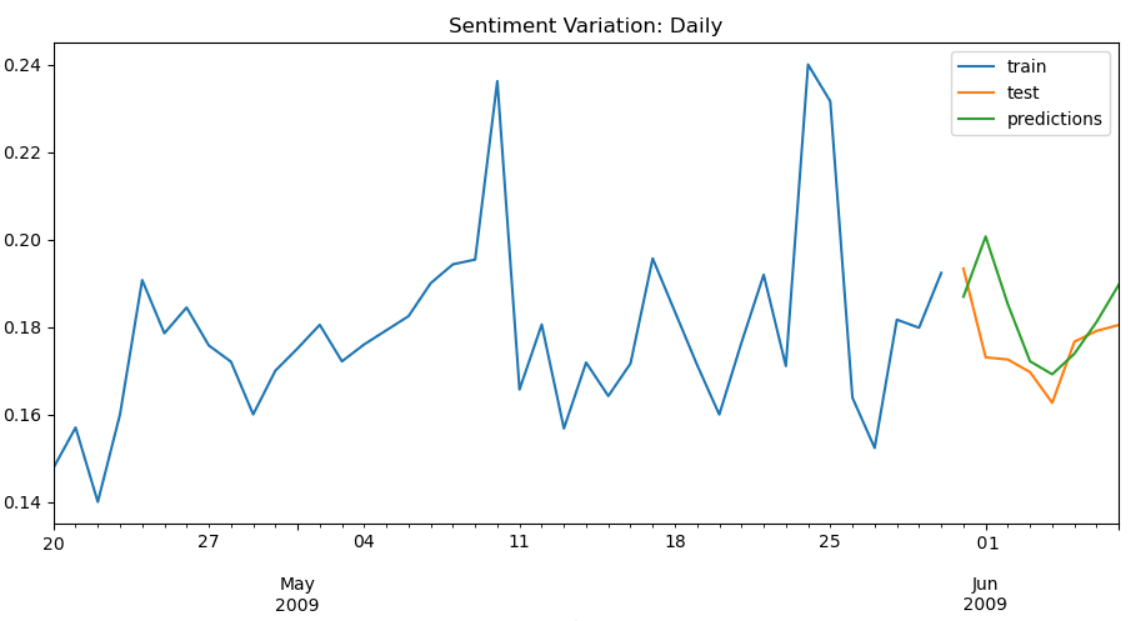
A tunning algorithm could be used to automatically define the best number of steps and lags. For this example, a manual approach was used to clarify the process of chosen the best parameters. On the next model algorithm, this tunning approach was used with ARIMA.

Predicted sentiment values for Recursive Autoregressive Forecast (8 steps):

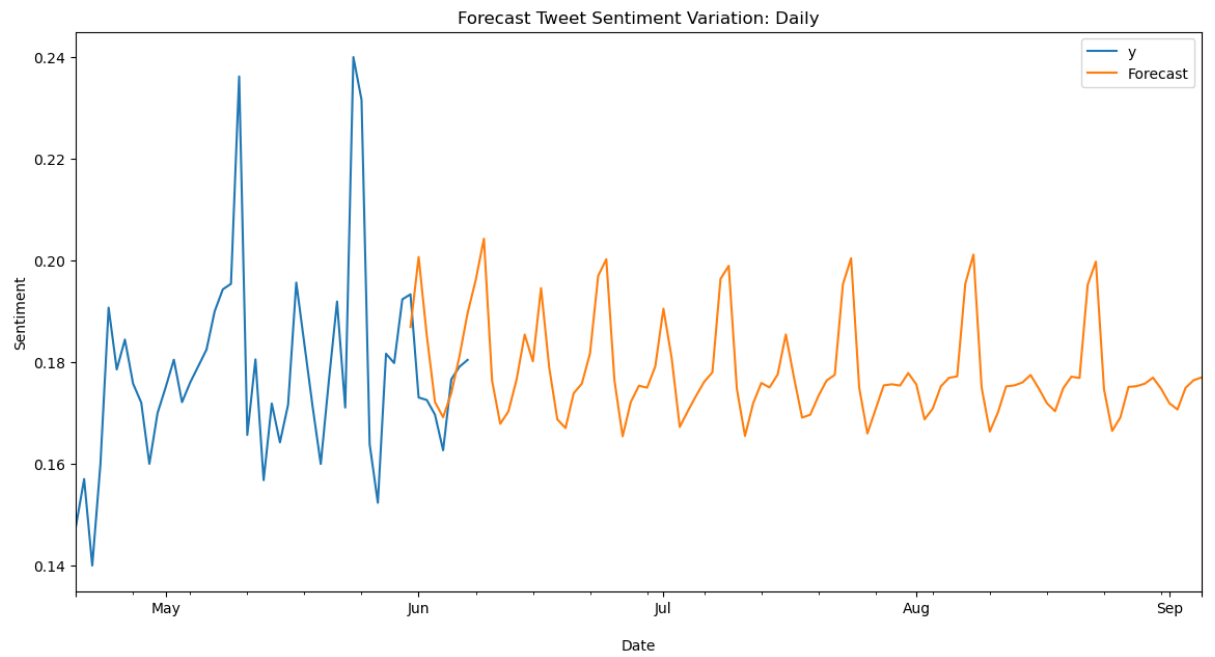
A screenshot of a computer screen

Description automatically generated

Plotting comparing the predicted sentiment values vs the original values:



Forecasting based on the model executed:



## Autoregressive Integrated Moving Average (ARIMA)

An autoregressive integrated moving average, or ARIMA, is a statistical analysis model that uses time series data to predict future trends. A statistical model is autoregressive if it predicts future values based on past values (Hayes A, 2023).

Although ARIMA requires at least 50 - but preferably more than 100 observations (Vignali V, 2023) – and this dataset had only 48 observations, ARIMA was chosen to be confronted by the results of Recursive Autoregressive algorithm.

Differently from Recursive Autoregressive which the best parameters were manually defined, using ARIMA the Akaike Information Criteria (AIC) was used for measure of a statistical model. It basically quantifies the goodness of fit, and the simplicity/parsimony of the model into a single statistic. When comparing all the possible models, the one with the lower AIC is better to be used (Keshvani A, 2013).

Testing the AIC possibilities with the sentiment dataset, the best (lower) results to be used were:

* ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC: -148.4272944748092
* ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC: -152.11004169878163

After fitting the dataset on the best suggested ARIMA configuration, a diagnose could be performed.

This step was used to diagnose the model to know if is behaving well or not. To diagnose the model, it was necessary to evaluate the residuals of the training data. The residuals are the difference between the model’s one-step-ahead predictions and the real values of the time series (Towards AI, 2022).

A collage of graphs

Description automatically generated

**Standardized residuals plot:** Shows one-step-ahead standardized residuals. In this case, the sentiment dataset model was working correctly, there was no obvious pattern in the residuals (Towards AI, 2022).

**Histogram plus estimated density plot:** Shows the distribution of the residuals.

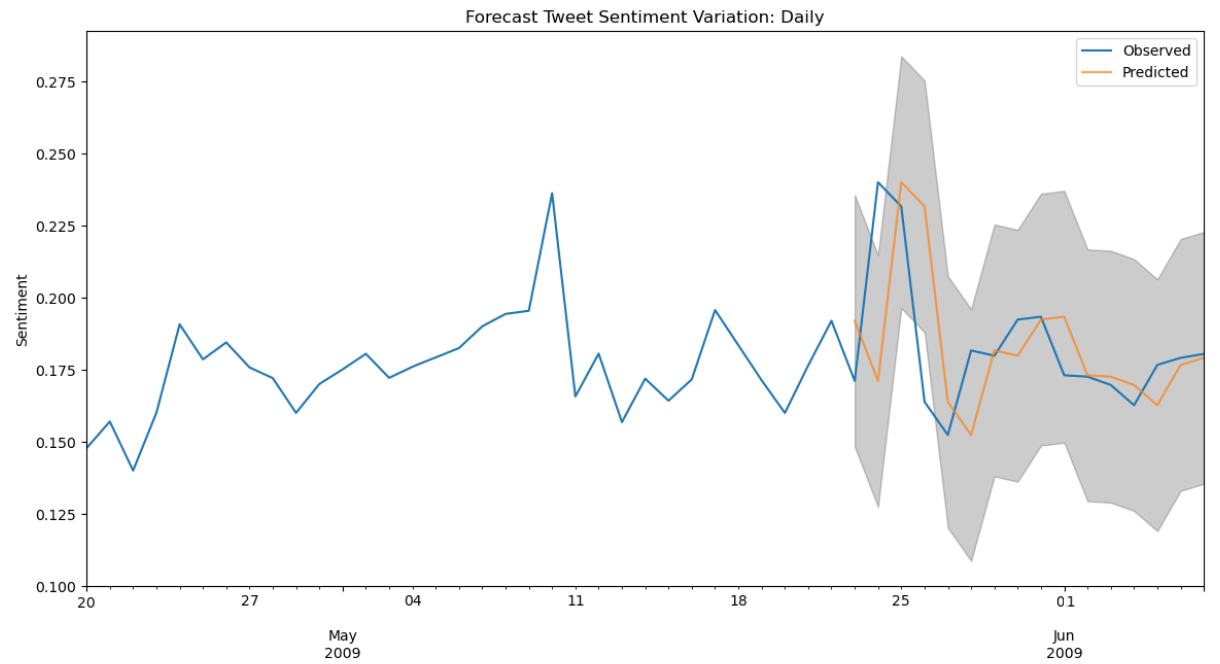
The histogram showed the measured distribution; the orange line showed a smoothed version of this histogram, and the green line shows a normal distribution. This concluded that the model was good to use because here there were small differences between the distributions, which indicate that the model was doing well (Towards AI, 2022).

**Normal Q-Q plot:** The Q-Q plot compares the distribution of the residuals to the normal distribution. It was possible to conclude that the model had a normal distribution of the residuals, all the points were lie along the red line, except for some values at the end (Towards AI, 2022).

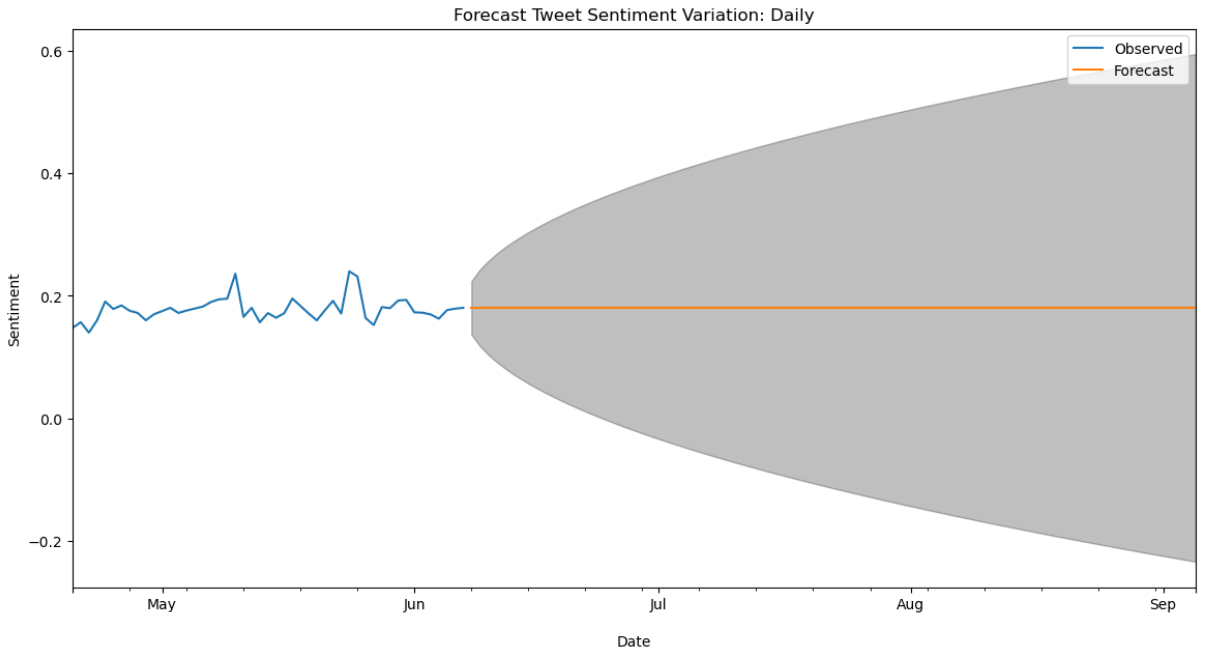
**Correlogram plot:** Is the ACF plot of the residuals rather than the data. 95% of the correlations for lag greater than zero should not be significant (within the blue shades). If there is a significant correlation in the residuals (not the case here), it means that there is information in the data that was not captured by the model (Towards AI, 2022).

After analyzing the best AIC configuration, the model was considered good to run the forecast, additional to that, the mean square error was excellent too: 0.000729.

Comparing the predicted sentiment values with the original values:



Forecast based on the ARIMA model executed:



Unfortunately, the forecast using ARIMA didn’t bring a useful result. After trying different possibilities and ARIMA parameters variations, the forecasted values were always “flat”. Maybe the reason is the small number of observations in the sentiment dataset, although others could find good results using sometimes less observations than in this study.

**Performance Comparison: Apache Spark SQL vs MySQL**

During the execution of this work, data was collected and stored in two different databases:

Apache Spark SQL Hive: Non-relational database.

MySQL: Relational database.

As should be expected, during the execution of this work was possible to notice a huge difference in performance when executing queries like select and update statements.

To have a clearer comparison of the performance differences, select queries were performed using two commands: CProfile and %time.

CProfile and %time are powerful time-consuming modules which capture how much time a python command takes to execute.

To perform this test, some rules and metrics were adopted:

* Both databases have the same table structure (columns) and same amount of data (observations).
* Select queries were used to perform the tests. The exact same queries were executed in both databases.
* Time consuming were the metric used (CProfile and %time results captured).
* To run the tests in each database, the Virtual machine were restarted as well Jupyter’s kernel.
* A first iteration (query execution) was performed to "start" the services and avoid test bias. Usually, the first select statement on a database takes longer.

**Executed Queries**

**Query 1:**

A select statement bringing all columns and all rows from the table (1.600.000 observations):



**Query 2:**

A select statement bringing custom columns and all rows from the table plus performing several string transformations (1.600.000 observations):

A screenshot of a computer program

Description automatically generated

**Query 3:**

A select statement bringing custom columns and all rows from the table plus performing several numeric calculations:

A screenshot of a computer

Description automatically generated

**Results (in seconds)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | First Query | | Second Query | | Third Query | |
|  | CProfile | %Time | CProfile | %Time | CProfile | %Time |
| Apache Spark SQL | 0.051 | 0.013 | 0.225 | 0.062 | 0.090 | 0.053 |
| MySQL | 61.66 | 20.60 | 93.85 | 28.70 | 104.31 | 43.70 |

The performance difference was huge, Apache Spark SQL Hive could perform much better than MySQL, even the more complex queries involving string transformations or numeric calculations barely affected the performance.

%Time took less time to execute probably because it was executed after CProfile and some cache was used by MySQL to re-execute the query, adding to that, CProfile does more like to map all the functions calls involved on the process which takes longer to execute.

As already mentioned, the performance difference was expected because Hive should be the appropriate chose when using non-relational data.

**Conclusion**

The dataset provided to be analyzed as a time series was a big challenge. The missing dates made the process extremely difficult and were definitive for the massive reduction of observations in the final dataset.

Despite the total number of initial observations being 1,600,000, when generating a frequency of daily data, even with corrections through the inclusion of some missing values, the final number of observations was only 48 records, which had a strong impact on the results of forecasts.

The decision to use the frequency by sentiment average per day and not by hour was decided due to the extremely large missing dates, practically 40% of the total possible if all dates were present. Using daily rather than hourly averages made the values that needed to be entered more precise to compensate missing dates.

Although these challenges found, the forecast results were promising, mainly with the use of the "Recursive Autoregressive Forecast" algorithm which, with its characteristic of using step lags, became possible to mitigate the low number of observations. The mean square error was excellent, resulting in only 0.00013.

The ARIMA algorithm wasn’t satisfactory, the forecast results were impossible to define, even with an excellent mean square error (0.000729) and excellent ARIMA diagnosis results after applying the best AIC configuration. The only reasonable explanation was that probably it was because the small number of observations available on the final sentiment dataset (less than 50).

Regarding benchmark performance results, it was expected that a non-relational database like Spark Hive would perform better in tests with large volumes of data without relational data (stored in just one table). What was perhaps not expected was the significant difference in results. Variations from 60 to 100 times faster in favor of the non-relational database, in this case Apache Spark SQL Hive.

Even though after table column index being created to tunning performance of the MySQL database, the results had little significant changes. Having relational data seems to be a considerate penalty to the overall performance.

**References**

GitHub: <https://github.com/2223066/CCT_Master_DataAnalytics_CA3>

Rimmalapudi (2023). Spark SQL Create a Table. Available at: <https://sparkbyexamples.com/apache-hive/spark-sql-create-a-table/> (Accessed: 18 Octobre 2023).

W3Schools (2023). SQL Tutorial. Available at: <https://www.w3schools.com/sql/default.asp> (Accessed: 18 Octobre 2023).

Stack Overflow (2023). PySpark: Converting a Temporary View to a Table and save as .CSV in local drive. Available at: <https://stackoverflow.com/questions/55448754/pyspark-converting-a-temporary-view-to-a-table-and-save-as-csv-in-local-drive> (Accessed: 19 Octobre 2023).

A2Hosting (2023). How to connect to MySQL using Python. Available at: <https://www.a2hosting.com/kb/developer-corner/mysql/connecting-to-mysql-using-python/> (Accessed: 20 Octobre 2023).

Janesch G (2020). PySpark + MySQL Tutorial. Available at: <https://towardsdatascience.com/pyspark-mysql-tutorial-fa3f7c26dc7> (Accessed: 21 Octobre 2023).

Navlani A (2019). Tutorial: Inserting Records and DataFrames Into a SQL Database. Available at: <https://www.dataquest.io/blog/sql-insert-tutorial/> (Accessed: 22 Octobre 2023).

Li S (2018). An End-to-End Project on Time Series Analysis and Forecasting with Python. Available at: <https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b> (Accessed: 23 Octobre 2023).

Stack Exchange (2014). Negative values for AIC in General Mixed Model. Available at: <https://stats.stackexchange.com/questions/84076/negative-values-for-aic-in-general-mixed-model> (Accessed: 24 Octobre 2023).

Keith M (2022). Forecast with ARIMA in Python More Easily with Scalecast. Available at: <https://towardsdatascience.com/forecast-with-arima-in-python-more-easily-with-scalecast-35125fc7dc2e> (Accessed: 25 Octobre 2023).

Despot I (2023) What Is Time-Series Forecasting?. Available at: <https://www.timescale.com/blog/what-is-time-series-forecasting/> (Accessed: 28 Octobre 2023).

IBM (2021) CRISP-DM Help Overview. Available at: <https://www.ibm.com/docs/it/spss-modeler/saas?topic=dm-crisp-help-overview> (Accessed: 28 Octobre 2023).

Data Science Process Alliance (2023) What is CRISP-DM? Available at: <https://www.datascience-pm.com/crisp-dm-2> (Accessed: 28 Octobre 2023).

Statistics.com (2023) Seasonal Decomposition. Available at: <https://www.statistics.com/glossary/seasonal-decomposition/> (Accessed: 30 Octobre 2023).

Cerliani M (2022) How to Improve Recursive Time Series Forecasting. Available at:: <https://towardsdatascience.com/how-to-improve-recursive-time-series-forecasting-ff5b90a98eeb> (Accessed: 30 Octobre 2023).

Hayes A (2023) Autoregressive Integrated Moving Average (ARIMA) Prediction Model. Available at:: <https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp> (Accessed: 30 Octobre 2023).

Vignali V (2023) Practical Minimum Sample Size for Road Crash Time-Series Prediction Models. Available at: <https://www.hindawi.com/journals/ace/2020/6672612/> (Accessed: 30 Octobre 2023).

Keshvani A (2013). Using AIC to Test ARIMA Models Models. Available at: <https://coolstatsblog.com/2013/08/14/using-aic-to-test-arima-models-2/> (Accessed: 30 Octobre 2023).

Towards AI (2022). Time Series Forecasting with ARIMA Models In Python [Part 2]. Available at: <https://towardsai.net/p/l/time-series-forecasting-with-arima-models-in-python-part-2> (Accessed: 30 Octobre 2023).