Department of
Informatics, cinema and mechatronics Engineering

DATA SCIENCE AND ENGINEERING

Real Time Anomaly Detection on Multivariate Time-Series

Supervisor:
Prof. Daniele Apiletti

Company Supervisor:
Ing. Antonio Greco
Ing. Davide Burdese

Candidate:
Erika Bosco
Mat.290262

Academic Year 2022-2023

Contents

Chapter 1

Introduction

In recent years the analysis of data gained a growing importance in society, in particular in business, as every type of company now needs the extraction of patterns and anomalies from their data, in order to make better decisions and to understand if the company is going to grow or to fail in the future.

1.1 Context

The thesis is the implementation of an anomaly detection project based on real data collected over four months.

Anomaly Detection and Time Series

The anomaly detection is a particular branch of the data analysis, as it is a problem of great practical significance across a range of real-world settings, including cyber-security, manufacturing, fraud detection, medical imaging, and others. Fundamentally, anomaly detection methods need to model the patterns in normal data to identify atypical samples. Although anomaly detection is a well-studied problem, developing effective methods for complex and high-dimensional data remains a challenge. Observations that have been recorded in an orderly fashion and which are correlated in time constitute a time series. Time series data mining aims to extract all meaningful knowledge from this data, and several mining tasks. Particularly we analysed the anomaly detection in real-time multivariate time-series, leading to an even challenger problem.

Mediamente Consulting

Mediamente, born in 2012 in the Politecnico di Torino's incubator i3p as a startup, gained the award as Startup of the year 2016. It has developed a specialization in management and analysis of companies' data, with specific solutions of Corporate Performance Management, Data Visualization, Data Integration and Data Management, which are the basis to enable the digital transformation of companies and organizations in Data Driven Company.

Mediamente is divided into more Business Units, but this thesis interest is specifically focused on the Infrastructure one, where a special project known as AMS, whose goal is to monitor in real time the performances of the customers' databases is constantly in progress.

The AMS is based on a consultant controlling a panel -in the figure- in which the various customers' monitoring tools are connected and that gives an alarm if there are some problems, so that when a database crashes the consultant can reactivate it as fast as possible. Another consultant monitors the support email, and that permit to ask for specific tasks, like performance analysis. The goal of the project implemented in the thesis is to understand when the customers' databases perform poorly in order to avoid it in the future.

It is divided in various box, indicating the importance of the system in the customers' line of business -MissionCritical is of vital importance, while Development

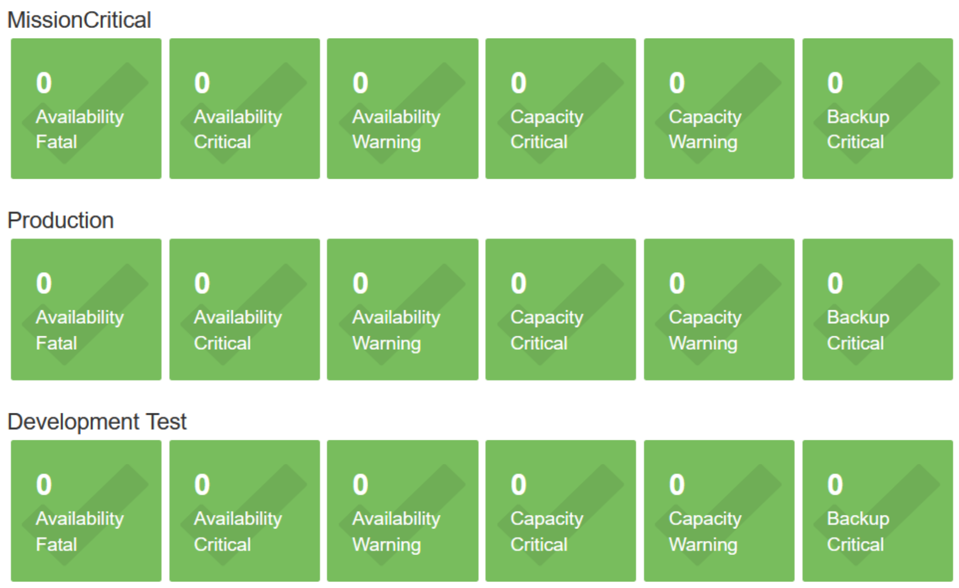

Figure 1.1: the panel used to monitor the customers' infrastructure in Mediamente

Test in the least important- and indicating the topic of the problem, availability, capacity, or backup.

1.2 Tools

The company is specialized on the brand Oracle for the databases.

Oracle Database

The Oracle Database [3] is a portable one, as it is available on every relevant platform, from Windows to UNIX/Linux to cloud-based systems. To better understand it, at first we have to distinguish between the Oracle Database that is a collection of physical operating system files or disks and an instance, which is a set of Oracle background processes or threads and a shared memory area. The Oracle Database can be of several types:

- single-instance: on the database server, you have one instance and one database

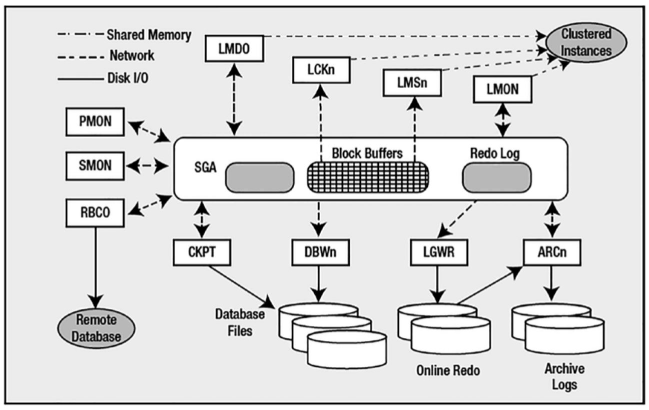

Figure 1.2: This is a simplified schema of the Oracle Database

- Real Application Cluster (RAC): which consists of one or more instances connected to one database
- Multitenant Container Database: a self-contained set of datafiles, control files, redo log files, parameter files, that only include the Oracle metadata, Oracle data, and Oracle code, with Pluggable Databases: set of datafiles only
- Multitenant RAC Database: more instances are connected to a container database, within which there are one or more pluggable databases

In particular the bank customer database we used to collect data from, was a Multitenant RAC Database, with two instances.

The Database is a Relational Database Management System (RDBMS), whose characteristic is to organize the data in table structures, each corresponding to a logic entity put in relation by common fields in different networks.

At the creation of an instance also the Shared Global Area and the Process Global Area are created, which are parts of the memory, and in which are allocated different views like the frequently accessed views and the aggregated views, respectively.

Database File

The most important files within these are the datafiles and the redo log files, but there is a long list of file type, of which we report the major ones:

- Parameter files: These files tell the Oracle instance where to find the control files, and they also specify certain initialization parameters that define how big certain memory structures are, and so on.
- Trace files: These are diagnostic files created by a server process, generally in response to some exceptional error condition.
- Alert files: These are similar to trace files, but they contain information about "expected" events, and they also alert the DBA in a single, centralized file of many database events.
- Datafiles: These are for the database; they hold your tables, indexes, and all other data segment types.
- Temp files: These files are used for disk-based sorts and temporary storage.
- Control files: These tell you where the datafiles, temp files, and redo log files are, as well as other relevant metadata about their state. They also contain backup information maintained by RMAN (Recovery Manager, the backup and recovery tool).
- Redo log files: These are the transaction logs files.
- Password files: These are used to authenticate users performing administrative activities over the network. These files are required if you want to connect remotely (over the network) as the SYS user to an instance. These files are required in a Data Guard configuration (primary and standby database).

- Change tracking file: This file facilitates a true incremental backup of Oracle data. It does not have to be located in the Fast Recovery Area, as it relates purely to database backup and recovery.
- Flashback log files: These files store "before images" of database blocks in order to facilitate the FLASHBACK DATABASE command.
- Data Pump files: These files are generated by the Oracle Data Pump Export process and consumed by the Data Pump Import process. This file format may also be created and consumed by external tables.
- Flat files: These are plain old files you can view in a text editor. It is normally used for loading data into the database

Memory Structure

The three major memory structures are the System Global Area (SGA), a large, shared memory segment that virtually all Oracle processes can access; Process Global Area (PGA), which is the part of memory that is private to a single process or thread, not accessible from other processes/threads; User Global Area (UGA), a memory associated with your session, located either in the SGA or the PGA.

Background Processes

The Oracle Database uses three types of processes:

- server processes, that perform work based on a client's request
- background processes, the most useful ones for our project, which start up with the database and perform various maintenance tasks
- slave processes, that are processes that perform extra work on behalf of either a background or a server process.

The most important background processes are:

- PMON [Process Monitor] is responsible for monitoring the other Oracle background processes and restarting them if necessary
- DBWn [database block writer] is the background process responsible for writing dirty blocks to disk
- LGWR [log writer] process is responsible for flushing to disk the contents of the redo log buffer located in the SGA

SQL Developer

SQL Developer is a graphic interpreter of SQLPlus, that, connected to an Oracle Database enables to browse, create, edit, and drop table, run SQL statements and scripts, edit and debug PL/SQL code, manipulate and export data and view and create reports. Thanks to this tool it was possible to schedule the collection of system performances data and to create pivoted history tables, where every minute took a record. It also permits to store and visualize the output data of the anomaly detection project.

Anaconda

Anaconda Distribution is a free Python/R languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics) that contains among other packages conda, a package and environment manager for your command line interface and Anaconda Navigator, a desktop application built on conda, with options to launch other development applications from your managed environments and is suitable for Windows, Linux, and macOS. At first it was implemented to remedy the problem created by the pip program of dependent packages, as it installed the new packages without checking for dependencies in other packages, risking an inability of some to work properly. [12]

Visual Studio Code

The most useful application for the project in analysis was Visual Studio Code, a source-code editor that can be used with a variety of programming languages, including C#, Java, JavaScript, Go, Node.js, Python, C++, C, Rust, and FORTRAN. Instead of a project system, it allows users to open one or more directories, which can then be saved in workspaces for future reuse. This allows it to operate as a language-agnostic code editor for any language. It supports a set of features that differs per language. Visual Studio Code can be extended via extensions, available through a central repository. Thanks to the extensions Jupyter, Jupyter Keymap, Python and Pylance it permitted us to write, debug and execute the Python code. [13]

1.3 Data

The data we used to implement the project were from a bank client, with an important amount of transaction each day. The storage is implemented with an Oracle pluggable -multitenant- database developed in a RAC structure.

Data type

The analysis was based on the metrics that the Oracle database collects in the v$sysmetric view. Collected data are numerical, timestamp data and strings. Specifically, the resulting dataframe collected had 13 columns, which were, among others:

- id: specify the id of the record
- inst_id: the instance's number of the database -1 or 2-

- begin_time: timestamp at which the metric collection is started
- end_time: finishing timestamp of the metric collection
- Metric_name: the explicit name of the metric
- Value: the value of the collected metric
- Metric_id: every metric was characterized by a specific number in order to differentiate them

However, as the database collects metrics every 60 seconds, in this disposition we had 160 records with the same begin_time and end_time, and we had some useless features, as if we have the name of the metric we did not need the metric_id and differentiating the study according to the instance we do not need the instance number specification. Therefore, we pivoted the database letting the begin time being the index and the name of each metric being the 160 columns.

Metrics

The collected metrics are of diverse types, for which we report some examples.

- Percentages, like **Host CPU utilization %**, so the percentage of CPU being used across hosts.

  Shared pool Free %, a metric that represents the percentage of the Shared Pool that is currently marked as free. The shared pool holds all referenced data dictionary definitions, all executed stored-object code (views, packages, procedures, triggers, functions, etc.), and all SQL code issued.

  PGA Cache Hit %, the percentage of hits in the cache of the Program Global Area.

Redo Allocation Hit Ratio, that monitors the redo log buffer hit ratio (percentage of success) against the values specified by the threshold arguments. The Redo log entries contain a record of changes that have been made to the database block buffers. The log writer (LGWR) process writes redo log entries from the log buffer to a redo log file, particularly the redo log buffer efficiency, as measured by the hit ratio, records the percentage of times users did not have to wait for the log writer to free space in the redo log buffer.

- Counted repetition per seconds, like **Redo generated per sec**, measured in bytes. It represents the amount of redo, generated per second during the sample period. Specifically, the redo log buffer is a circular buffer in the SGA. That contains the information necessary to reconstruct, or redo, changes made to the database by INSERT, UPDATE, DELETE, CREATE, ALTER, or DROP operations. ("Redo Writes (per transaction)").

  Logical reads per sec represents the number of logical reads (consistent gets, from the data buffer) per second during the sample period. A logical read is a read request for a data block from the SGA. Logical reads may result in a physical read if the requested block does not reside with the buffer cache.

  Hard parse count per sec, is the number of hard parses per second during this sample period. A hard parse takes place when a SQL statement has to be loaded into the shared pool. In this event, the Oracle Server has to allocate memory in the shared pool and parse the statement. Each time a particular SQL cursor is parsed, this count will increase by one. There are certain operations that will cause a SQL cursor to be parsed. Parsing a SQL statement breaks it down into atomic steps, which the optimizer will evaluate when generating an execution plan for the cursor.

  Enqueue deadlocks per sec, represents the number of times per second that a process detected a potential deadlock when exchanging two buffers and raised an internal error.

CR Blocks Created Per Sec, which is the Oracle metric of the number of CR (consistent read) blocks created per second. Read consistency is used when competing processes are reading and updating the data concurrently.

- Counted repetition per transactions (Txn), like **Open cursor per txn**, the number of open cursor for transaction, with cursor we specify a pointer to the context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

  Parse failure count per txn is the total number of parse failures or errors in a computer program that occurs while the program is being read by the computer per transaction.

  The **Long table scans per txn** metric represents the number of long table scans per transactions during sample period. A table is considered 'long' if the table is not cached and if its high-water mark is greater than 5 blocks.

  Leaf node splits per txn is the number of times per transaction an index leaf node was split because of the insertion of an additional value.

- Absolute values like **Current logons count**, which is the number of logons.

  Global cache blocks corrupted checks whether a corrupt block cached in an instance was received by another instance through the private interconnect. This usually results from a transmission error caused either by network problems or by adapter hardware issues.

  Temp Space Used, which is the sum of bytes used by temporary tablespaces, the ones used for special operations, particularly for sorting data results on disk and for hash joins in SQL. For SQL with millions of rows returned, the sort operation is too large for the RAM area and must occur on disk. therefore, the temporary tablespace is where this takes place.

The definitions are from the Oracle documentation [3]

Chapter 2

State of Art

2.1 Preparation

The embryonic project implemented in Mediamente had a flux starting on the loading of the data and the model in the machine, then different classes and methods train the model on the collected data and return the output tables for the anomalies and for the other computations performed in the flux.

2.2 Tables

Input

As previously stated, the input is given by the pivot of the Oracle view 'V$SYSMETRIC', or 'GV$SYSMETRIC', historicized, which shows the various metrics sampled every minute.

Model

The anomaly detection previously implemented used as detector the univariate Isolation Forest, which is an algorithm created in sklearn that 'isolates' observations randomly selecting a split value between the maximum and minimum values of the selected feature -we will analyse it better in the following chapters-. Since recursive partitioning can be represented by a tree structure, the number of splitting required to isolate a sample is equivalent to the path length from the root node to the terminating node. This path length, averaged over a forest of such random trees, is a measure of normality and the decision function random partitioning produces noticeably shorter paths for anomalies. Therefore, it is highly probable that when a random forest produces shorter path lengths for specific samples, they are anomalies. [14]

Output

The anomaly detection tool creates a table just with the records that it finds anomalous. The table created, 'ML_ALERT_HISTORY' is composed by ten columns:

- DATABASE_NAME: The database from which we collected the sysmetric's records
- INSTANCE_ID: the instance of the database
- METRIC: which metric resulted anomalous
- ANOMALY_STATUS: can be Open, Closed or Pending
- ANOMALY_VALIDATION: can be 0 or 1 if the anomaly is been validated or not -it is not functioning at the state of art of the beginning of my thesis-

- MAX_VALUE: the maximum value the specific metric assumed in the time range in study
- RESET_COUNT: counter used to pass from the Pending state to the Close one
- INS_TIME: timestamp when the record is been written
- UPD_TIME: update time
- END_TIME: time of the end of the anomaly

The tool creates also other tables, useful to better understand the anomaly. The ML_CONTEXT_TOP_CORRELATIONS is the table where are listed the metrics correlated to the anomalous ones. The columns are:

- SOURCE_METRIC: the anomalous metric correlated
- METRIC: the non-anomalous metric
- INS_TIME: timestamp when the record is been inserted in the table

Another table created by the Anomaly Detection tool is the 'Flow Manager' one, which is used to control the data flux generated by the algorithm. It has different columns that control the status of the job, not the metrics per se.

- JOBID: time of the beginning of the job, equals to the INS_TIME
- STATUS: can be 0 -job finished- or 1 -job running-
- JOB_START: when the job started
- JOB_END: time of the end of the job
- NUM_SAMPLES: number of samples read by the job

- NUM_METRICS: number of metrics read by the job
- LAST_DATA_READ: begin_time of the last sample read

2.3 Class and methods

The class are implemented to better organize the process.

Anomaly Detector

It is the principal class, which implements the anomaly detection algorithm and calls the other classes. It is composed by different methods, such as the 'init', that initialize the objects taking the cx_Oracle.connect parameters, the 'load_model', that loads the different models for the different metrics -the model implemented was the same for every metric at this point-, and the 'rolling_window_all', which is the method that permits to collect the four samples window on which 'rolling_window_features' applies the algorithm to compute the anomalies and return the update status.

Alert History Buffer

This class is used as buffer between the table 'ML_ALERT_HISTORY' and the python code, so with 'load_alert_history' it loads the open or pending anomalies, with 'get_metric' returns the status of the specified feature, than with 'collect_delta' collects the new state of an existing anomaly or inserts a new one and updates the buffer thanks to 'upd_alert_history'.

Data Collector

It is the connection between the input table of the DB and the python code. The methods 'dlt_load' that loads all data, integrated with the 'get_residual_window' one

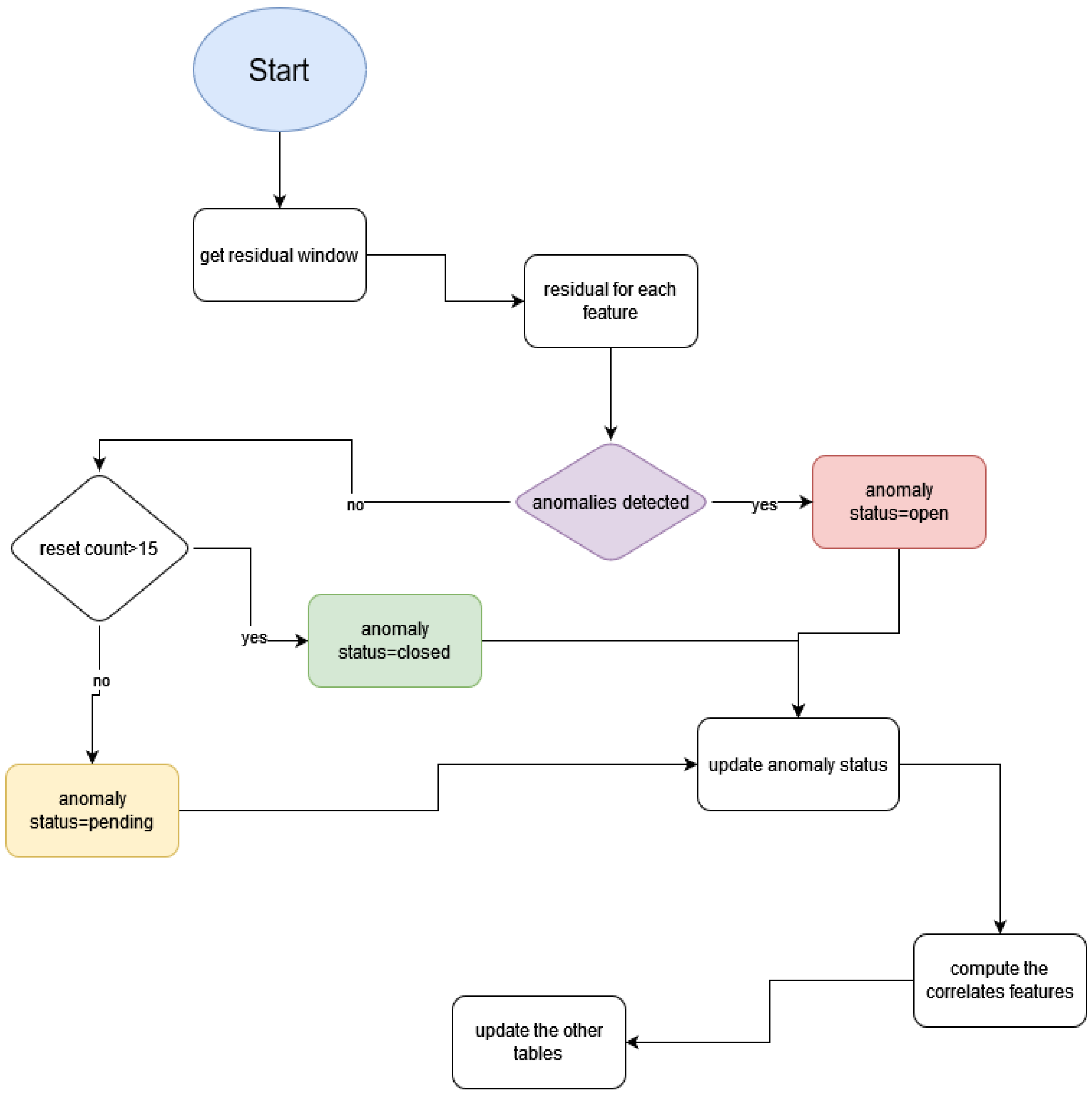

Figure 2.1: This is a simplified schema of the data flux that was implemented.

returns the dataset on which it is necessary to compute the residual

Flow Manager

This class is used to create and update the 'FLOW_MANAGER' table. It uses different methods for the current job, such as 'insert', used to insert a new record for the current job that will be updated at the end of it thanks to 'update' and 'get_job_id', that controls that it is the right record to be updated, and for the previous job, such as 'get_last_status' that return the status of the last job and 'get_last_data_read', whose output is the timestamp of the last record.

Model

It represents the machine learning model applied on the data window. The methods are 'fit', that takes a whole dataframe to better train the model, 'predict', that predicts whether and where there are anomalies in the new data and 'save' to save the parameter used for the prediction.

Tspreprocessor

This class is used to implement preliminary operations such as the STL decomposition, applied before the isolation forest.

TopCorrelation

It is used to implement the necessary steps to compute the top correlations of a metric, thanks to 'set_window' it sets the temporal window where the correlation has to be computed and thank to 'get_top_correlation' it computes the metric with a correlation greater than 0.5.

Utilities

This last class implements the conversions, thanks to 'memory_usage' it returns the used memory, then 'convert_ts_int' converts timestamps in int number with a format yyyymmddhhmmss, as it is in this way that the timestamps are displayed on SQL. The last method 'remove_invalid_chars' removes the invalid characters for the windows files.

Chapter 3

Preprocessing Step

3.1 Feature selection

The first step of the preprocessing process is to select only the features that are useful, however as the goal is to control as much features as possible, we deleted only the ones with all the values equal to zero. To capture this kind of feature we computed the mean of the column and checked the plot. The result of this preprocessing step is a database with 155 columns, as we dropped the following features:

- PX downgraded 1 to 25% Per Sec, the parallel execution downgraded
- Streams Pool Usage Percentage, so the usage of this particular shared resource
- Captured user calls, which represents the number of logins, parses, or execute calls captured during the sample period
- Workload Capture and Replay status, which controls if the workload has been captured and replayed

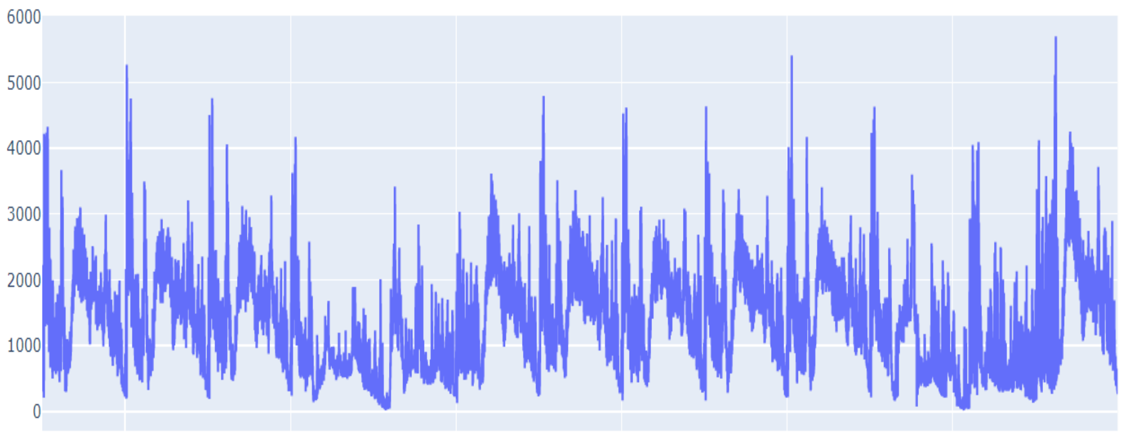

Figure 3.1: This is the plot of the CPU usage per second before the decomposition -January the 19th to January 31st-

- Global Cache Blocks Corrupted, which counts the number of blocks that encountered a corruption or checksum failure during interconnect
- VM out bytes Per Sec, which is used only when VMs are present
- Replayed user calls, which is the number of logins, parses, or execute calls replayed during the sample period

3.2 Computing the residuals

A crucial step in the preprocessing process is the decomposition of the time series. As the dataset we have is composed of time series we can filter the signal, and analyse separately the trend, the seasonality and the residual or remainder.

The decomposition is made using the STL tool of the rstl python library, which is an implementation in python of the R STL library.

Based on the paper STL: A Seasonal-Trend Decomposition Procedure Based on Loess [5], the STL algorithm compute the loess regression curve choosing a positive integer q. The positive q values of the x_i that are closest to x are selected and get

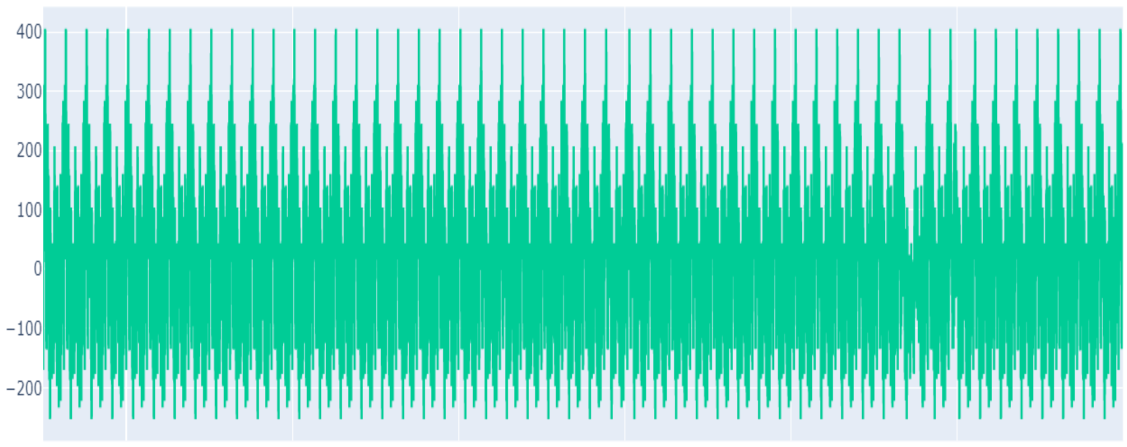

Figure 3.2: This is the plot of the seasonal component of the CPU usage per second metric.

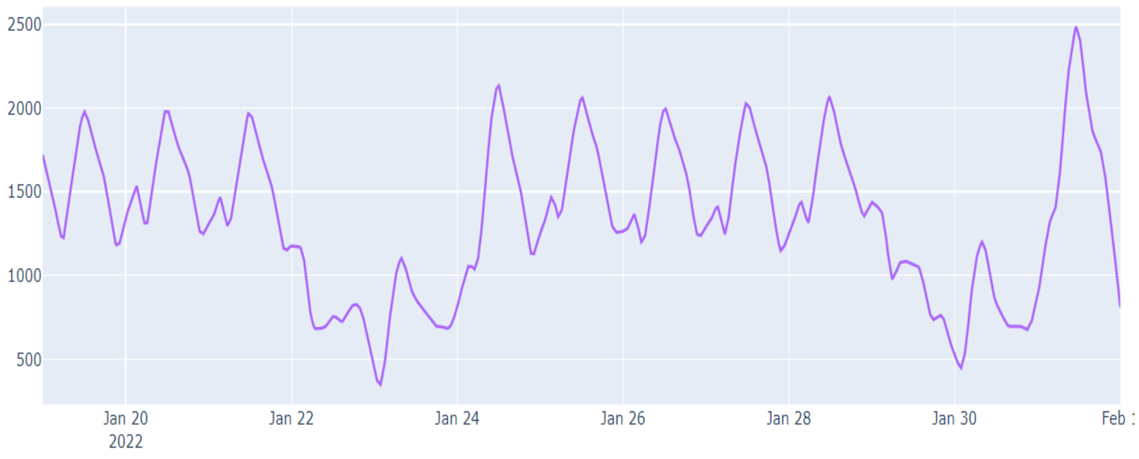

Figure 3.3: This is the plot of the trend component of the CPU usage per second metric.

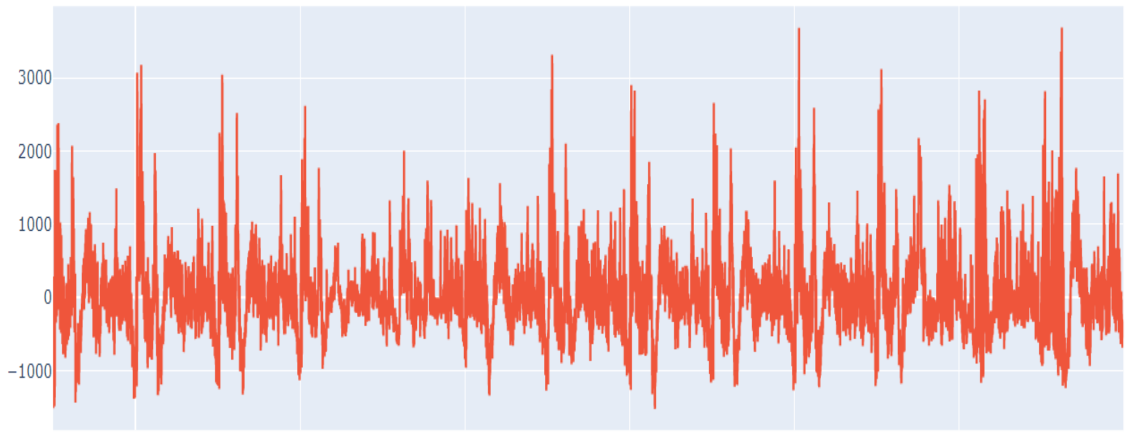

Figure 3.4: This is the plot of the residual component of the CPU usage per second metric.

a neighbourhood weight based on the distance. If $\lambda_q(x)$ are the distance of the qth farthest x_i from x, then the tricube weight function will be:

$$W(u) = \begin{matrix} (1-u^3)^3 \text{ for } 0 <= u < 1 \\ \\ 0 \text{ for } u >= 1 \end{matrix} \tag{3.1}$$

The neighbourhood weight for any x_i is

$$v_i(x) = W(\frac{|x_i - x|}{\lambda_q(x)}) \tag{3.2}$$

Thus the x_i close to x have the largest weights, which decreases as the distance increases.

The procedure itself consists in two recursive procedures, an inner loop that consists of a seasonal and trend smoothing nested inside an outer one where the remainders are computed. In the inner loop the first step is the detrending, so it is computed the difference between the point in the time-series and the point in the trend, then the cycle subseries of the detrended one is smoothed, and a low-pass filtering is applied on each subseries, which is again detrended and deseasonalized. In the outer loop the computation of the remainder is made by a subtraction of the original time-series minus the trend part and the seasonal part.

The STL parameters have to be carefully chosen, as they are the responsible for the best decomposition of the time series, therefore we chose a seasonality equals to 2880, which is the number of records collected in a day.

We have to take into account that the goal of this project is to find anomalies, therefore the trend part of the time-series, which has not spikes and the seasonal part, which is recursive, are not interesting in this study. Therefore the most useful part of the decomposition is the residual one, where the recursive and trending part of the time series are no longer taken into consideration.

An example of the plot of this decomposition is on the figures, fig 3.1 is the original time-series of the CPU Usage Per Second. After the decomposition step we have fig 3.2, the seasonal part, fig 3.3, the trend and fig 3.4, the remainder of the time-series.

3.3 Normalisation

Another major step of the preprocessing process was the normalisation of the remainder, as an important problem was that the value of the time-series can be, according to the feature into consideration, measured in hundreds, thousands, or fraction of one and the residuals inherit it.

Again, at first we deleted all the features with a standard deviation smaller than 0.001, so quite constant. Again, we dropped in this way some features, which are:

- PX downgraded 25 to 50% Per Sec, the parallel execution downgraded
- Enqueue Deadlocks Per Txn, the number of times per transaction that a pro-

cess detected a potential deadlock when exchanging two buffers and raised an internal, restorable error

- Disk Sort Per Txn, which is the number of sorts going to disk
- Enqueue Deadlocks Per Sec, which is like Enqueue Deadlocks Per Txn, but per second
- User Limit %. As the Oracle database has a maximum of users according to the license this metric controls the percentage of users reached.

The remaining features were 149.

After that, a z-score normalisation has been implemented over the database, so we computed both the mean and the standard deviation of each column -feature- of the new database containing only the residuals and every record has been subtracted by the mean and divided by the standard deviation.

The preprocessing process is outlined in figure 3.5 and some examples of the remaining time series are in the figures 3.6 and 3.7.

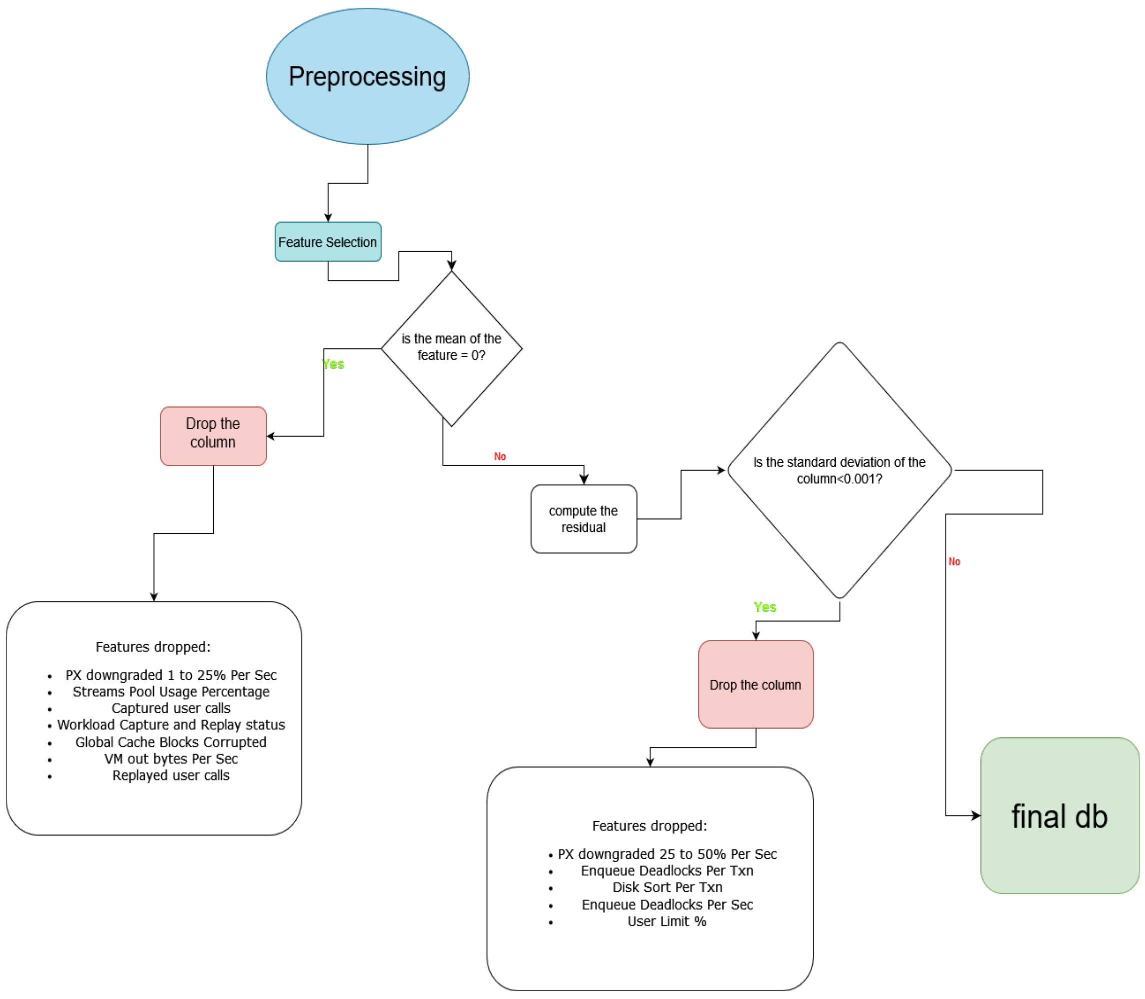
Preprocessing
Feature Selection
is the mean of the feature = 0?
Yes
Drop the column
No
compute the residual
Is the standard deviation of the column<0.001?
Yes
No
Drop the column
Features dropped:
PX downgraded 1 to 25% Per Sec
Streams Pool Usage Percentage
Captured user calls
Workload Capture and Replay status
Global Cache Blocks Corrupted
VM out bytes Per Sec
Replayed user calls
Features dropped:
PX downgraded 25 to 50% Per Sec
Enqueue Deadlocks Per Txn
Disk Sort Per Txn
Enqueue Deadlocks Per Sec
User Limit %
final db

Figure 3.5: Outline of the preprocessing step

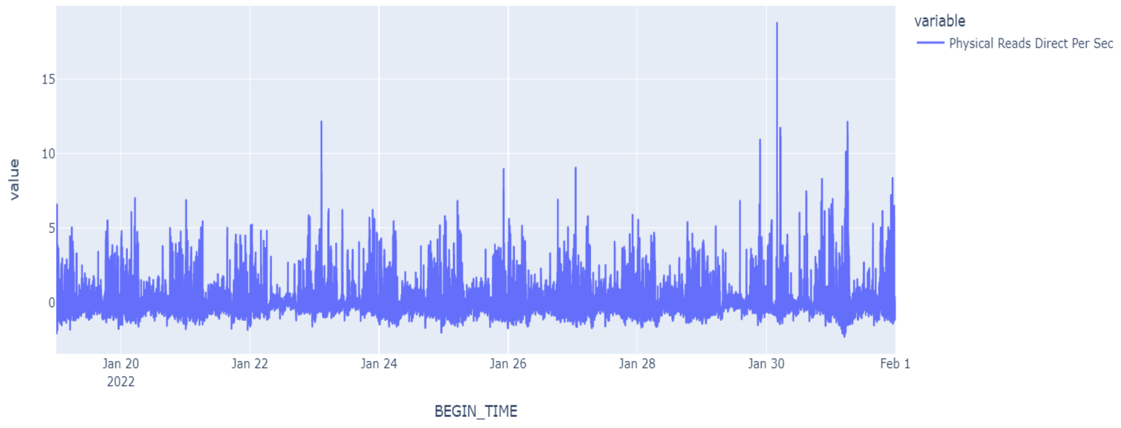

Figure 3.6: This is the plot of the residual of physical reads direct per second normalized with the z-score –January the 19th to January 31st-

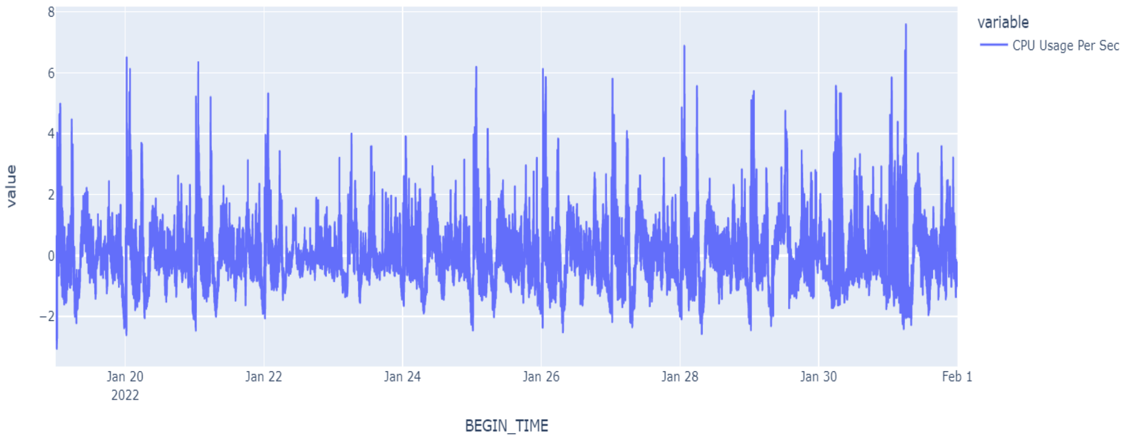

Figure 3.7: This is the plot of the residual of CPU usage per second normalized with the z-score -January the 19th to January 31st-

Chapter 4

Model Selection

As previously stated, the model used in the test project in Mediamente was the univariate Isolation Forest, a sklearn algorithm created that ‘isolates’ observations randomly selecting a split value between the maximum and minimum values. This model however, as it was, produced too many false positive but still not recognizing some of the real anomalies.

One of the problems was that the database is a multivariate time series system, so the interdependence of the metrics was not recognized by a model that analysed univariately the features. The model selection had to consider also that the model has to be unsupervised, as we do not have the possibility to label the records exactly.

The choice has been therefore to analyse only multivariate models.

In this chapter we will analyse the different models in the pyOD [8] library we chose and how they work comparing the results obtained by them at predicting the outliers in a select part of the multivariate dataset we previously cleaned.

4.1 ABOD

The approach named ABOD [4] (Angle-Based Outlier Detection) is based on the variance in the angles between the difference vectors of a point to the other points, as the general idea is that comparing the angles between pairs of distance vectors to other points helps to discern between points similar to other points and outliers.

This model is composed by different steps, the first one assigns the Angle Based Outlier Factor value to any object in the database, computing the scalar product of the difference vectors of any triple of points normalized by the quadratic product of the length of the difference vectors, so the angle is weighted less if the corresponding points are far from the query point. By this weighting factor, the distance influences the value, but only to a minor part. However, this variance weighting is important since the angle to a pair of points varies stronger for a bigger distance. The variance of this value over all pairs for the query point constitutes the angle-based outlier factor (ABOF). The equation that better explains the functioning of the ABOF is:

$$ABOF(\vec{A}) = VAR_{\vec{B},\vec{C}\in D}(\frac{\langle \bar{AB}, \bar{AC}\rangle}{\|\bar{AB}\|^2\|\bar{AC}\|^2}) \tag{4.1}$$

taking into account that the three points $\vec{A}$, $\vec{B}$, $\vec{C}$ are different.

The algorithm ABOD assigns the angle-based outlier factor ABOF to each point in the database and returns as a result the list of points sorted according to their Angle Based Outlier Factor. The top-ranked point is the utmost outlier. The next ranks are taken by border points of the cluster whilst the lowest ranks are assigned to the inner points of the cluster. Since the distance is accounted for only as a weight for the main criterion, the variance of angles, ABOD is able to concisely detect outliers even in high-dimensional data where purely distance-based approaches deteriorate in accuracy. Furthermore, ABOD allows also a different ranking of border points

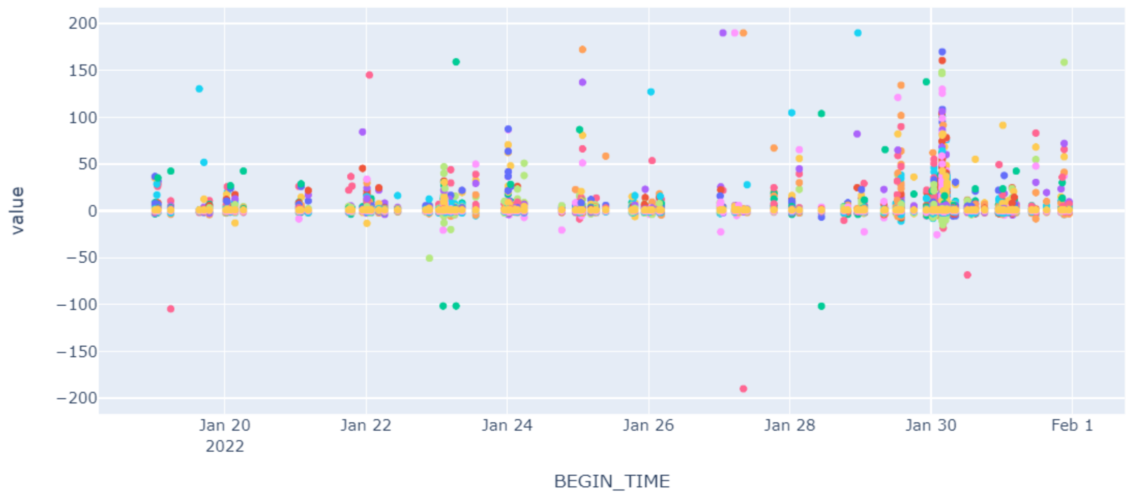

Figure 4.1: The scatter plot of all the features, in different colours each, of the anomalous timestamps found by the abod model

versus inner points of a cluster. This is not possible for most of the other outlier models. Also, most outlier detection models require the user to specify parameters that are crucial to the outcome of the approach. For unsupervised approaches, such requirements are always a drawback. Thus, a big advantage of ABOD is being completely free of parameters.

The difference vector to the most similar object in the nearest group of points provides the divergence quantitatively for each attribute and, thus, explains why, or how much, the point is an outlier. [4]

The ABOD model was trained by the customer database and subsequently used to search for the outliers in the January the 18^{th}, January the 31^{st} set, giving as a result approximately two hundred points as outliers. The figures shows where we found them.

4.2 Feature Bagging

The Feature Bagging method [6] is composed by a series of T rounds, in every round t, the outlier detection algorithm is called and presented with a distinct set of features F_t that is used in distance computation. It is based on the division of the set of features F_t into a collection of feature bags, randomly selected from the original dataset, such that also the number of features in F_t is randomly chosen between (d/2) and (d-1), where d is the number of features in the original data set. The chosen outlier detection algorithm is the Local Outlier Factor model, which consists into giving each record an outlier factor taking into account also the density of the point neighbourhood, as in the CBLOF, but instead of clustering it, using all the records in the feature bag.

Every subset, as a result of the LOF, outputs different outlier score vector AS_t that reflects the probability of each data record from the dataset S of being an outlier. For example, if $AS_t(i) > AS_t(j)$, the data record x_i has higher probability of being outlier than the data record x_j. At the end of the procedure, after T rounds, there are T outlier score vectors each corresponding to a single round. The function COMBINE is then used to coalesce these T outlier score vectors $AS_t, t = \bar{1,T}$ into a unique anomaly score vector AS_{FINAL}, which is lastly used to assign a final probability of being an outlier to every data record from the data set. The COMBINE function is implemented through the computation of the average of the record's different outlier factors, leading to a single value of "anomaly score", so how much is this record anomalous, and is computed for each record.

The decisive step is represented by the label of the records as anomalous or in-bounds points, whether they are greater than a threshold or not.

The fig. 4.2 is the graphical representation of the 180 anomalous points found in the customer dataset between January the 18th and January the 31st by the feature

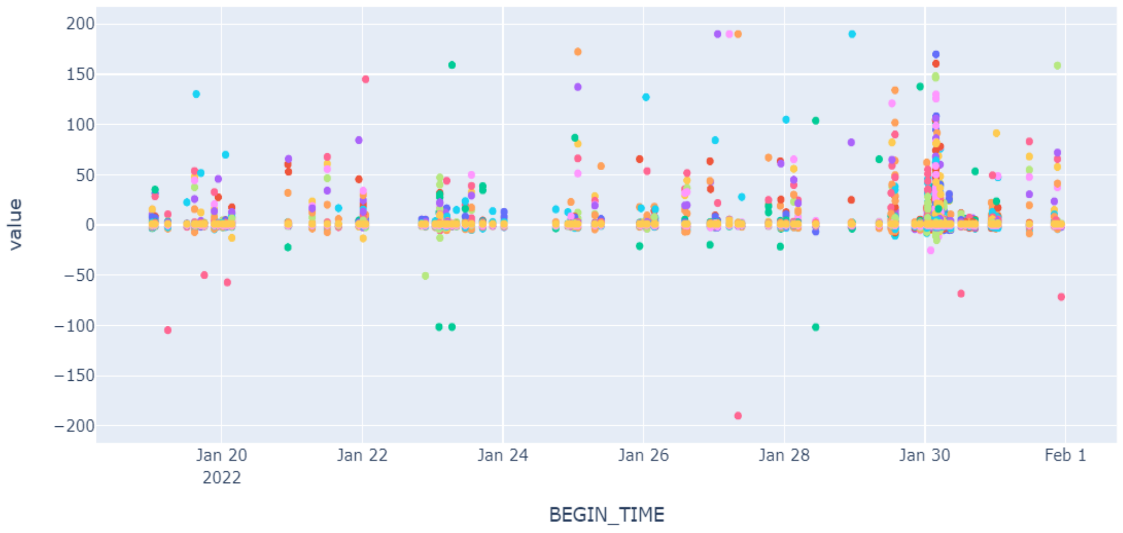

Figure 4.2: The scatter plot of all the features, in different colours each, of the anomalous timestamps found by the feature bagging model

bagging approach.

4.3 PCA

The Principal Component Analysis can be used to solve a general version of the problem that aims to find a k-dimensional hyperplane (for any value of $k < d$) that minimises the squared projection error over the remaining $(d - k)$ dimensions.

In principal component analysis, the $d \times d$ covariance matrix over d-dimensional data is computed, where the (i, j)th entry is equal to the covariance between the dimensions i and j for the set of N observation. Considering a multidimensional data set D of dimensionality d and size N, the covariance matrix Σ is:

$$\Sigma = \frac{D^T D}{N} \tag{4.2}$$

the first step is to transform the data into a new axis system of representation. The orthonormal eigenvectors provide the axes directions along which the data should

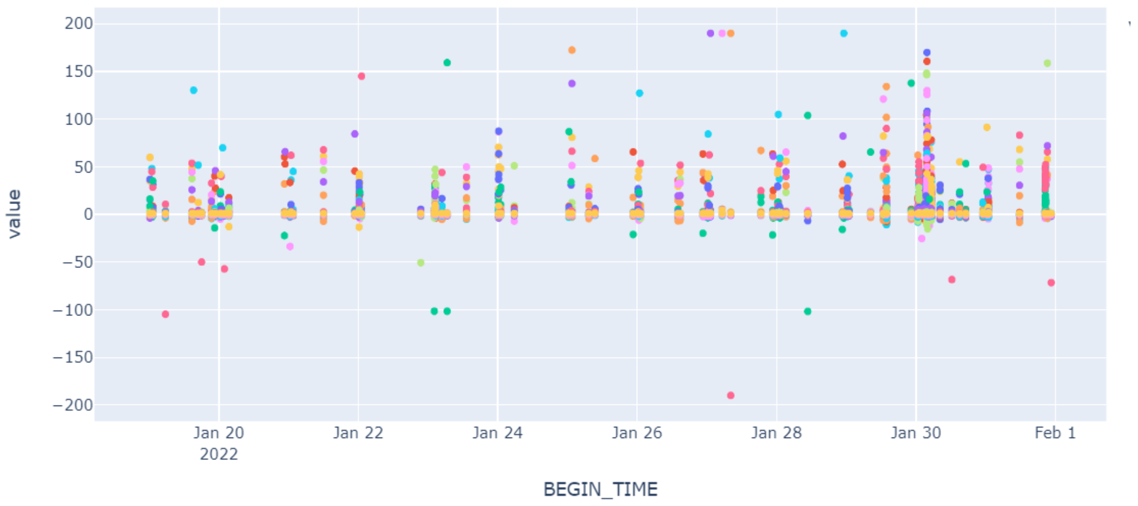

Figure 4.3: The scatter plot of all the features, in different colours each, of the anomalous timestamps found by the PCA model

be projected.

The important part about Principal Component Analysis is that the largest eigenvectors provide the key directions of global correlation. These directions are referred to as the principal components, which are uncorrelated and retain most of the data variance. However, in real settings, it is common for a large fraction of the eigenvalues to be very small. This means that most of the data aligns along a much lower-dimensional subspace, which is very convenient from the perspective of outlier analysis because the observations that do not respect this alignment can be assumed to be outliers.

In the end the algorithm computes the distance between the data points and the hyperplane built by the eigenvectors. If the distance is large the data point can be identified as an outlier.

In fig 4.3 we show the results of the pca model prediction of the anomalous points found in the customer dataset between January the 18^{th} and January the 31^{st}

4.4 GMM

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities as given by the equation

$$p(x|\lambda) = \sum_{i=1}^{M} w_i g(x|\mu_i, \sum i) \tag{4.3}$$

where x is a feature, w_i, i = 1, . . . , M are the mixture weights, so the likelihood that a particular data point belongs to each class, and $g(x|\mu_i, \sum i)$, i = 1, . . . , M are the component Gaussian densities, each is a Gaussian function of the form

$$g(x|\mu_i, \sum i) = \frac{1}{(2\pi)^{\frac{D}{2}} |\sum i|^{\frac{1}{2}}} exp\{-\frac{1}{2}(x-\mu_i)' \sum_i^{-1} (x-\mu_i)\} \tag{4.4}$$

. The mixture weights satisfy the constraint that $\sum_{i=1}^{M} w_i = 1$

The complete Gaussian mixture model is parameterized by the mean vectors, covariance matrices and mixture weights from all component densities. These parameters are collectively represented by the notation, $\lambda = w_i, \mu_i, \sum i$ i = 1, . . . , M.

GMMs are often used in biometric systems, most notably in speaker recognition systems, due to their capability of representing a large class of sample distributions. One of the powerful attributes of the GMM is its ability to form smooth approximations to arbitrarily shaped densities. The classical uni-modal Gaussian model represents feature distributions by a position (mean vector) and an elliptic shape (covariance matrix) and a vector quantizer (VQ) or nearest neighbour model represents a distribution by a discrete set of characteristic templates. A GMM acts as a hybrid between these two models by using a discrete set of Gaussian functions, each with their own mean and covariance matrix, to allow a better modelling capability.

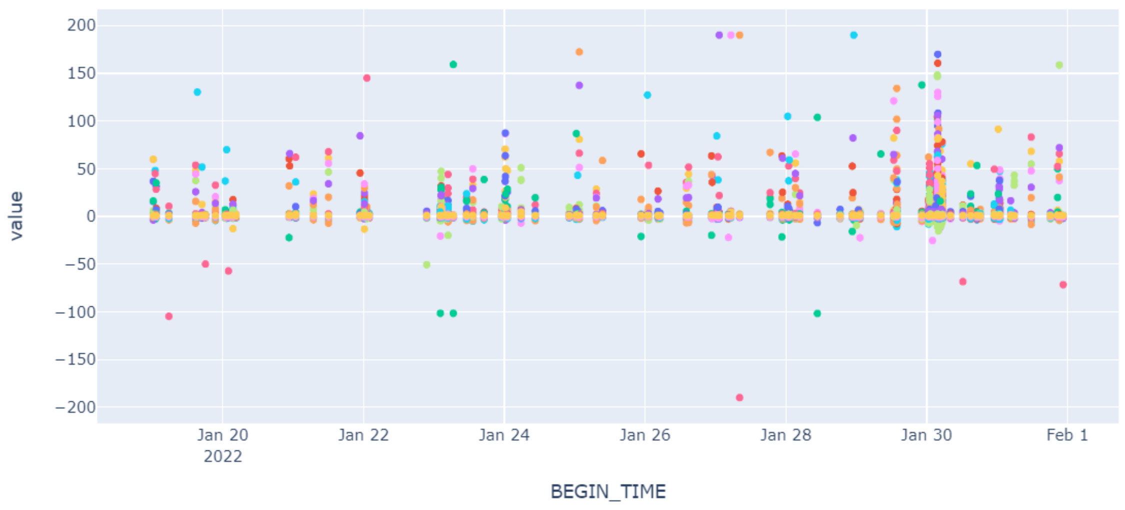

Figure 4.4: The scatter plot of all the features, in different colours each, of the anomalous timestamps found by the gaussian mixture model

GMM parameters are estimated from training data using the iterative Expectation-Maximization (EM) algorithm or Maximum A Posteriori (MAP) estimation from a prior model.

The GMM not only provides a smooth overall distribution fit, its components also clearly detail the multi-modal nature of the density.

To estimate the parameters of the GMM, λ, which in some sense best matches the distribution of the training feature vectors there are several techniques. The method used is the expectation-maximization (EM) algorithm.

The basic idea of the EM algorithm is, beginning with an initial model λ, to estimate a new model $\bar{\lambda}$, such that $p(X|\bar{\lambda}) \geq p(X|\lambda)$. The new model then becomes the initial model for the next iteration and the process is repeated until some convergence threshold is reached.

Figure 4.4 represents the anomalies found by the GMM in the selected dataset.

4.5 CBLOF

The Cluster Based Local Outlier Factor [7] aims to solve both requests to find clusters for the dataset and to find outliers. The algorithm first assigns to each object an outlier factor, measured by both the size of the cluster the object belongs to and the distance between the object and its closer cluster. To determine the cluster of belonging the CBLOF uses the KMeans, which computes the Euclidean distance between two objects as

$$D = \sum [(x_i - z_i)^2]^{\frac{1}{2}} \tag{4.5}$$

and the smaller the Euclidean distance is, the more similar the two points are. Taking a K number of centroids and computing the distances determines, with recursive steps, to which cluster a point belongs to. The result will be a K number of clusters, divided in Small and Large ones according to a parameter b, boundary of clusters.

The definition of Small and Large Clusters, given two parameters α and β, derives from

$$\begin{gathered}(|C_1| + |C_2| + ... + |C_b|) \geq |D| * \alpha \\ |C_b|/|C_{b+1} \geq \beta\end{gathered} \tag{4.6}$$

where D is the dataset dimension. Then the set of large clusters is defined as $LC = \{C_b | i \leq b\}$ and the set of small clusters: $SC = \{C_j | j > b\}$

The second part of the CBLOF is the FindCBLOF algorithm, which detects outliers. The data points are treated differently whether they belong to a Small or a Large cluster. So if C_i belongs to SC, and t is a data point $CBLOF = |C_i| * min(distance(t, C_j))$ where C_j belongs to a Large Cluster while if C_i belongs to a Large Cluster $CBLOF = |C_i| * (distance(t, C_i)$. The point is labelled as outlier

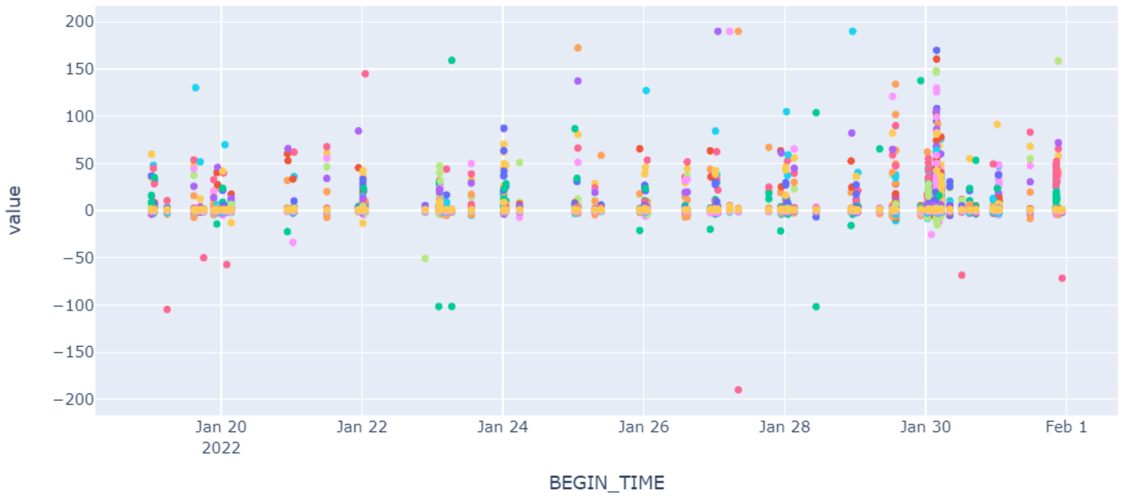

Figure 4.5: The scatter plot of all the features, in different colours each, of the anomalous timestamps found by the cblof model

if the CBLOF is higher than the threshold.

We therefore trained the model on the selected database, and find the outliers points in fig. 4.5.

4.6 Isolation Forest

The Isolation Forest [10] model used to find anomalies is based on two different steps. The first one builds isolation trees using sub-samples of the given training set. The second passes test instances through isolation trees to obtain an anomaly score for each record.

In the training step, iTrees are constructed by recursively partitioning a sub-sample X' until all instances are isolated. Particularly an iTree is built, with X' as the input data, e the current tree height and l the height limit by iteratively randomly selecting an attribute q that belongs to Q -the set of features- and select a split point p between the maximum and the minimum value of that feature having as result of the filter

$$\begin{gathered}\{X_l = filter(X', q < p)\} \\ \{X_r = filter(X', q \geq p)\}\end{gathered} \tag{4.7}$$

two nodes, the $Left = iTree(X_l, e+1, l)$ and the $Right = iTree(X_r, e+1, l)$ or, if the sub-sample X' cannot be splitted again because $e > l$ or $|X| < 1$ taking the node as the final leaf.

A forest is itself an iterative process where at each step a subsample of the feature is selected. Subsampling size ψ controls the training data size. We use $\psi = 256$ as the default value for our experiment, as used in the Isolation Forest paper. [11].

The anomaly score s is then derived from the expected path length $E(h(x))$ for each test instance. $E(h(x))$ are derived by passing instances through each iTree in an iForest. A single path length $h(x)$ is derived by counting the number of edges e from the root node to a terminating node as instance x traverses through an iTree. When x is terminated at an external node, where $Size > 1$, the return value is $e + C(Size)$, where the adjustment accounts for an unbuilt subtree beyond the tree height limit.

In fig. 4.6, we can see the customer dataset scatter plot for anomalous timestamp only, from the 18^{th} to the 31^{st} of January.

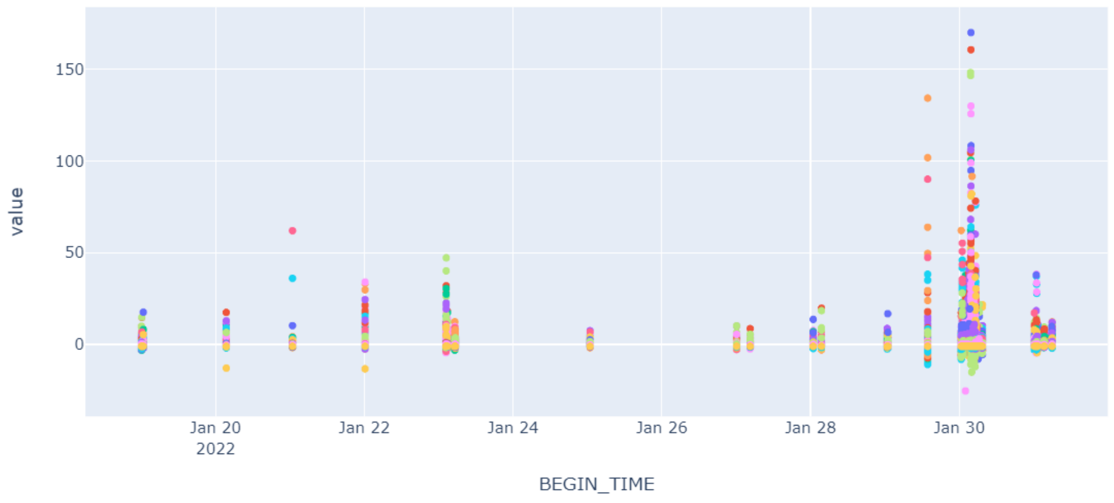

Figure 4.6: The scatter plot of all the features, in different colours each, of the anomalous timestamps found by the isolation forest model

Chapter 5

Data Validation and Hyperparameter Tuning

The different models have different results as anomalous points, therefore they had to be checked with a validation set.

5.1 Hyperparameter Tuning

The **hyperparameter tuning** consisted in trying different contamination parameters and seeing the different results. The best results are obtained by a contamination of 0.005, thanks to which the anomalous points were between 150 and 250 in the time analysed.

5.2 Data Validation

The data validation part was implemented using a different dataset as validation set. It was a dataset composed by the different values of db_time, so the database time, computed by oracle as

$$DB\_Time = CPU\_time + I/O\_time + Non - idle\_wait\_time \tag{5.1}$$

which consist of the amount of elapsed time (in microseconds) spent performing database user-level calls, not including time spent on instance background processes [3], recorded every half an hour, with a total of 6939 records for the period between the 1st of January and the 27th of May.

The initial dataset we had had five columns:

- instance: the database instance the record has been taken
- date: an int value of the day of the year -yyyymmdd-
- start_time: an int representation of the start hour -hhmm-
- end_time: the finish hour represented by an int variable -hhmm-
- value: the db_time value itself computed in the period between the start_time and the end_time

however, as the values have been recorded only in an instance and every recording lasted half an hour, we dropped both the column 'instance' and 'start_time'.

The dataset itself contained both anomalous and inbound points, therefore we had to clean it and take only the outlier points.

In order to clean the data from recording error, such as records with an end of recording a minute later and data with two different values at the same recording time we modify the end_time to be standardized and we dropped one of the two values, according to the choice of the Mediamente dba who monitored the database in the specific day. Also, we dropped the Nan values.

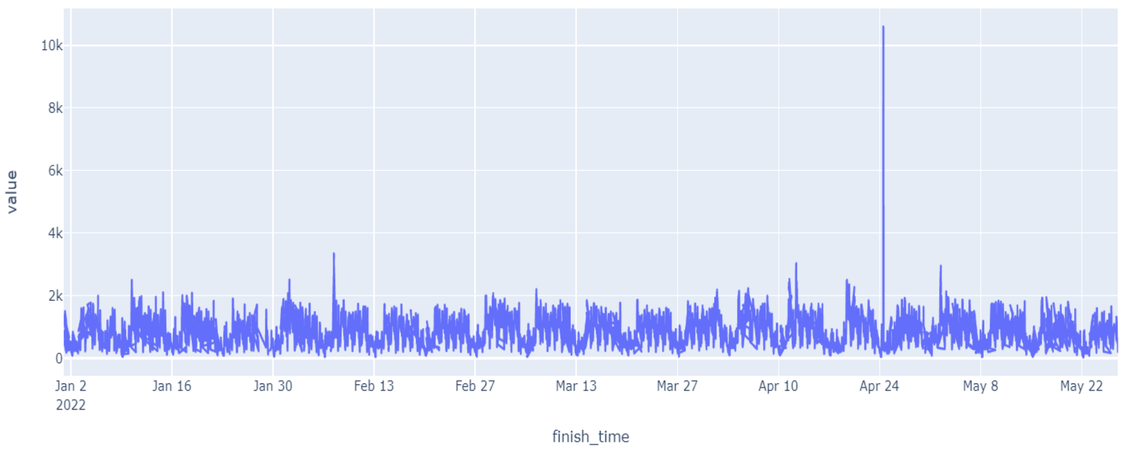

Figure 5.1: db_time cleaned database

The cleared database has been plotted to better visualize it, after merging the date and time columns in order to have a a timestamp formatted as 'finish_time'.

In the preprocessing step we had to understand if a record was anomalous comparing it to the other record collected on the same day of the week, as the db_time recorded in a Sunday is different from the one collected in a Monday, but it is right to be different as the database is more used in workdays. Because of this behaviour we could not use a threshold.

We divided therefore the dataset in seven different subsets, one each day of the week and pivoted each in order to have the different hour of collection as columns.

The following step was to decide which records had to be considered anomalous. As the metric db_time is a symptom of the database reactivity and when it is too high it means that the database is not performing well, we decided to find the anomalous point normalizing each column of the new subsets with the z-score normalization and then taking only the points with a value greater or equal to 3 σ.

The number of anomalous points found between the 1st of January and the 25th of May was 396, represented in the fig. 5.2

In the analysed time range, between 18th January and 31st January the anoma-

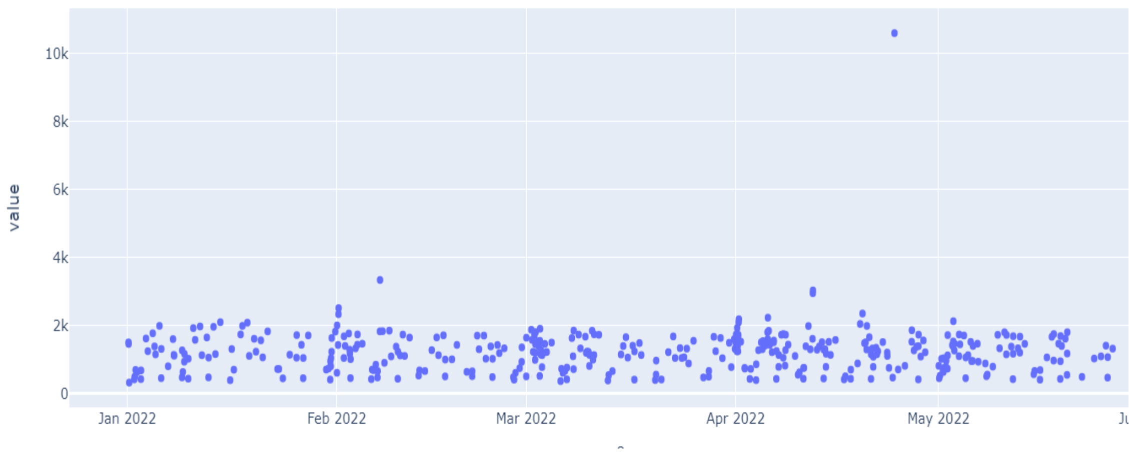

Figure 5.2: Anomalous points find in the db_time dataset

lous points were 25, taking into account that the recorded time was every half an hour.

The anomalies found were then confirmed by the Mediamente's employees that monitored the databases in those days.

We therefore compared the anomalous points found by the different models with the ones found in the data validation set.

To understand the result, we have to take into account that the anomalies found by the model should be both a certain range of time before the actual anomalous event and during the anomaly, as the idea is that the anomalous metrics predict the real anomalies. To confirm this idea, we can think that if a metric like **Enqueue deadlocks per sec** has a high value the deadlocks will probably lead to a database problem in the near future, but on the other hand if the dataset is not working well the metrics will have anomalous behaviour.

The first model we tested was the ABOD one, which had as result the following plot, where the red dots were the data validation set points and the blue lines the anomalies found by the ABOD model.

We can see that the ABOD model had a lot of false positive, so we discarded it

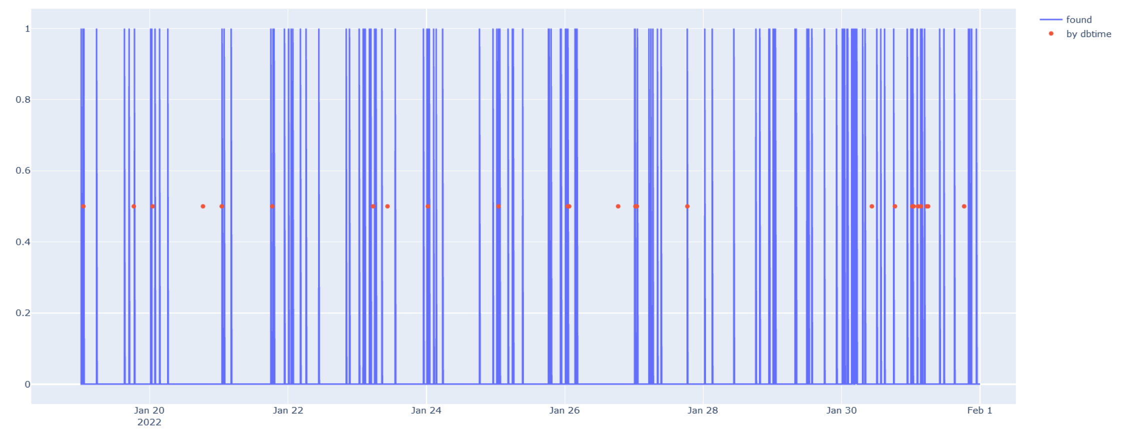

Figure 5.3: Abod model compared to real anomalies

as it was not the best one.

Then we proceeded to compare the CBLOF model anomalies with the real one.

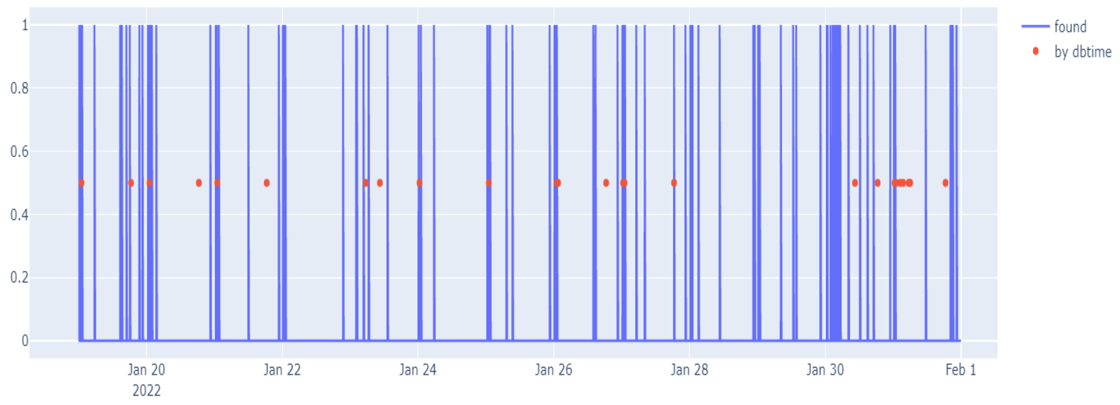

Figure 5.4: Anomalies found by the CBLOF model compared to the real anomalies

This model correctly predicts quite all the anomalies that really took place, however also the CBLOF model predicted a number of anomalies greater than the real number, so a great part of the predicted ones was false positive. We decided that the CBLOF model was not the best one.

The results obtained by the Gaussian Mixture Model were with even a greater number of false positive than the other tested models, so we could not accept it as

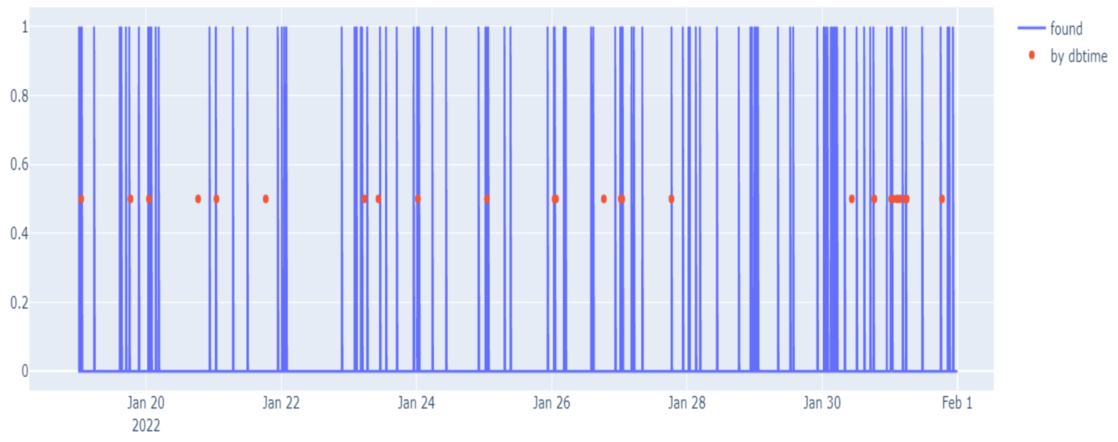

Figure 5.5: Anomalies found by the Gaussian Mixture Model model compared to the real anomalies

the model to complete the Mediamente architecture.

We proceeded to test the PCA model, that resulted in an important number of false positive.

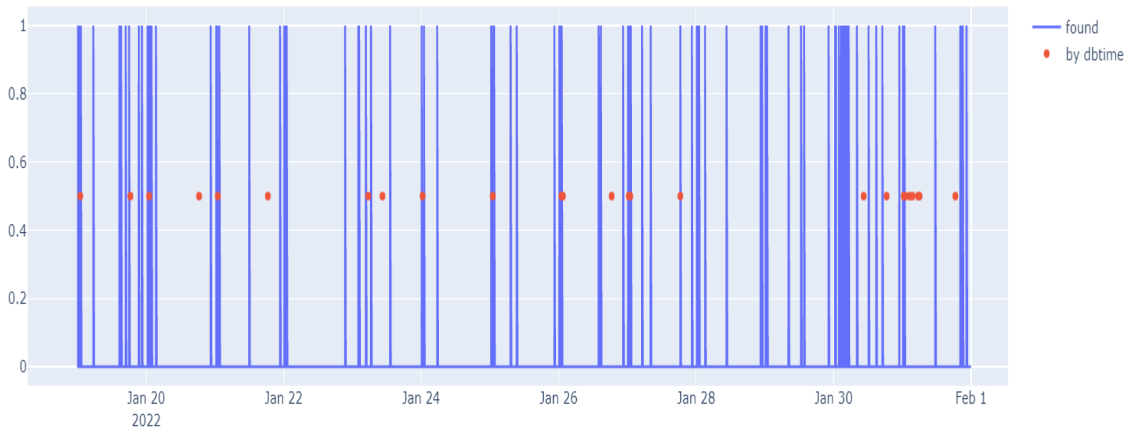

Figure 5.6: Anomalies found by the Principal Component Analisys model compared to the real anomalies

Also, the Feature Bagging model obtained a lot of false positive, so even if it well predicted the true anomalies, we discarded this model as well.

The Isolation forest model is the only one that, even if it also has some false positive it has a smaller amount of it and a greater part of the predicted anomalies

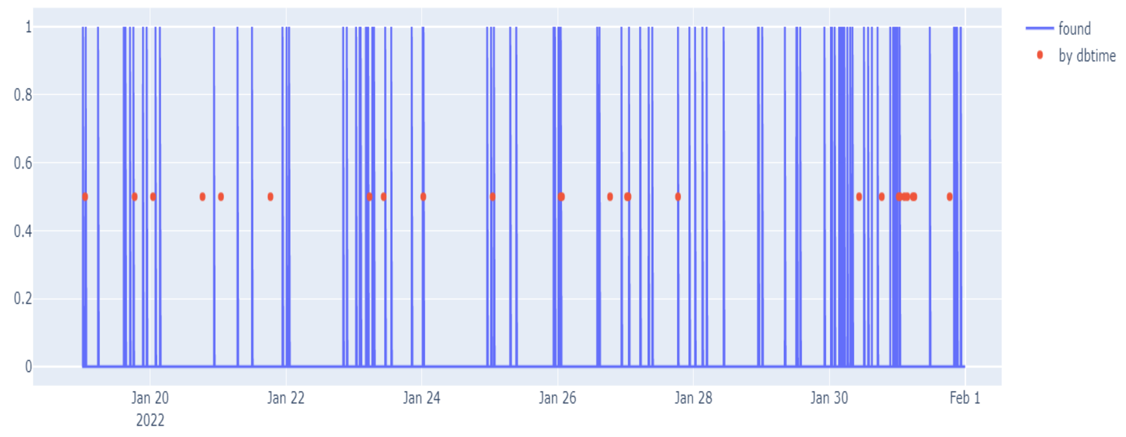

Figure 5.7: Anomalies found by the Feature Bagging model compared to the real anomalies

that correspond to the real ones or to a little time before a real anomaly, which is the goal of this project.

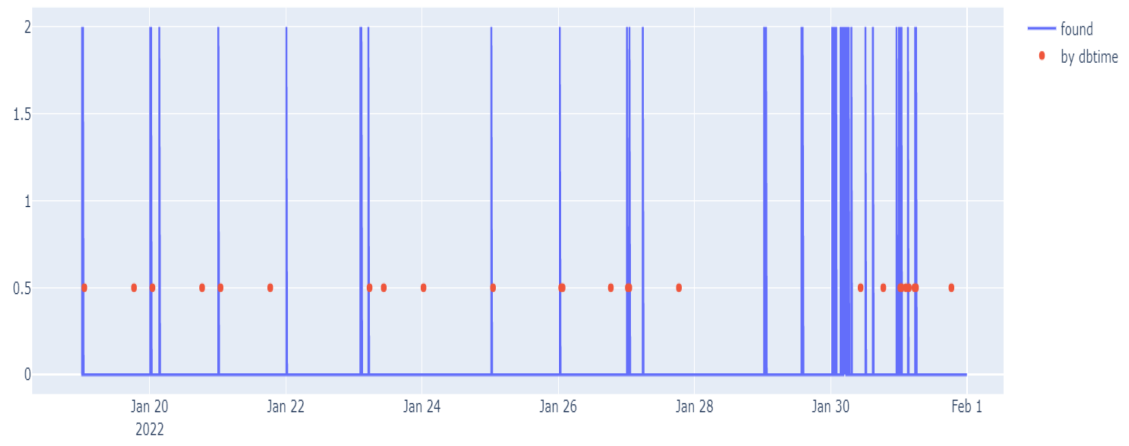

Figure 5.8: Anomalies found by the Isolation Forest model compared to the real anomalies

We compared the different models in a plot, where the different lines correspond to how likely it is to find an anomaly in the corresponding timestamp or in the near following time, according to the specific model.

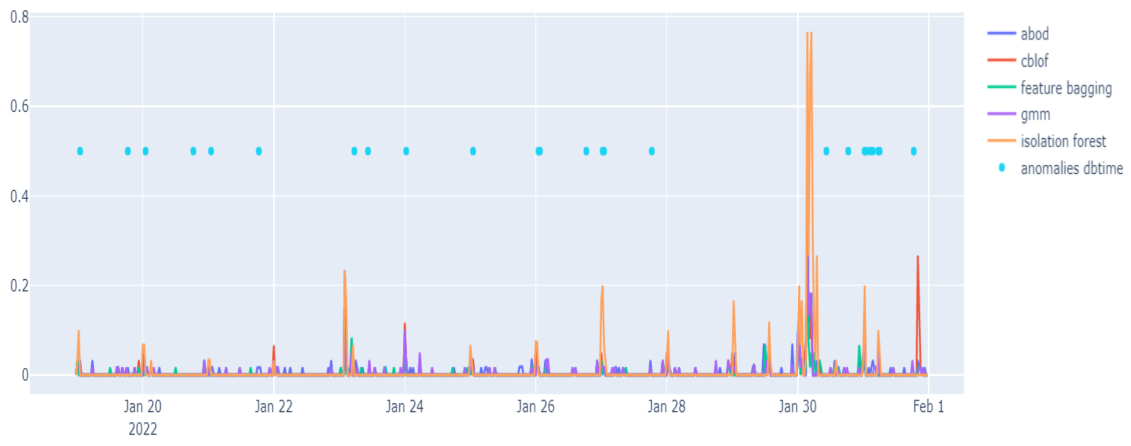

Figure 5.9: Probability to have an anomaly according to the different models and real anomalies, in light blue points

In the end we computed the Accuracy [15] as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.2}$$

where TP=True Positive, TN=True Negative, FP=False Positive and FN=False Negative and the Precision as

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

counting as True Positive not only the exact timestamp where the real anomaly is, but also the preceding and following half an hour of the real anomalies time. The results are shown in table 5.1.

As both Precision and Accuracy are higher in the isolation forest compared to the other models we chose the multivariate isolation forest as the best performing model in our research and implemented it in the pre-existent architecture.

To better understand the prediction capacity of the model we chose, we trained the Isolation Forest with the dataset of January and predicted the anomalies in

Model	Precision	Accuracy
abod	0.2307	0.7936
feature bagging	0.1473	0.7872
pca	0.2262	0.8208
gaussian mixture model	0.2448	0.8144
cblof	0.2168	0.8192
isolation forest	0.3255	0.8704

Table 5.1: Result obtained by the different models in precision and accuracy.

February. The result is the following, considering that the value -1 as the anomalous one and the value 1 as the not anomalous, leading us to affirm that the isolation model is the good one to be used in the Mediamente project.

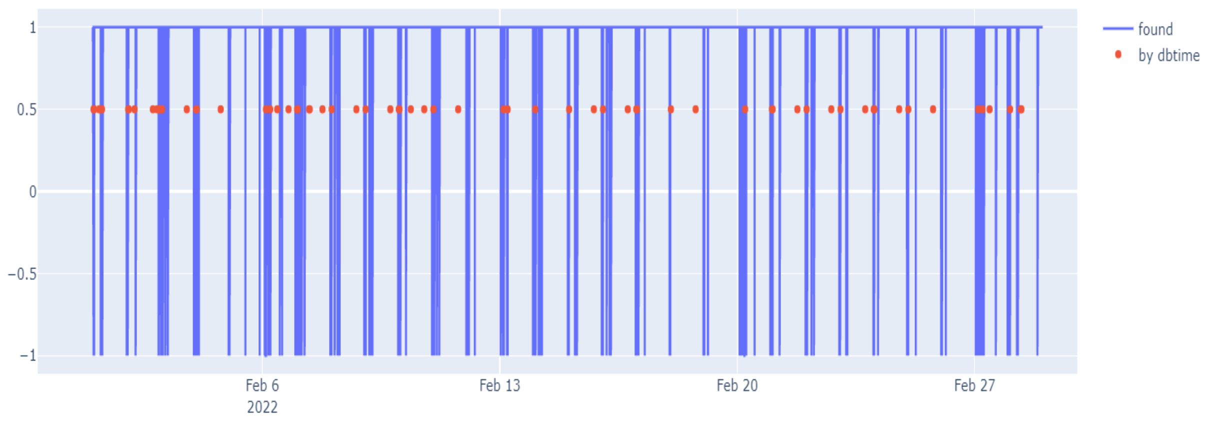

Figure 5.10: Anomalies found by the Isolation Forest model compared to the real anomalies in the February month

Chapter 6

Conclusions

The main goal of this thesis was to search the best model to find anomalies in real time in a multivariate time-series dataset, so we studied and chose the ideally best models and evaluated them. The best one was the multivariate isolation forest that well predicted the anomalies also when we tested in the February dataset and not only in the test dataset of thirteen days in January.

Also, this model has a small computing time, so it can take place in quite real time, which is a fundamental goal in this project and the main one that lead us to decide to not try autoencoder or other type of Neural Network, to avoid the augmenting of computing time.

The study and the research of the different models has been especially important to have a better performing tool, as it is the core of the prediction part.

As the tool will be used to parallel the existent panel the employee of Mediamente use, which right now is used to highlights the databases and connectivity components problems when they happen, the anomaly detection tool will be the tool that permits to avoid the crashes, letting the Mediamente workers act before it happens and prevent them.

The next step is therefore to implement the anomaly detection tool in the panel in use.

Another goal now is to search for a system view also in the SQLServer database system to analyse and to use it as dataset to predict the anomalies, in order to have an anomaly detection tool also for this kind of databases. It is not an obvious transaction, as the system works differently, even if they are both relational databases, the SQL server system has different focus on collected system views and system tables compared to Oracle ones.

Bibliography

[1] *Scikit-learn: Machine learning in Python*, Pedregosa and Varoquaux and Gramfort and Michel and Thirion and Grisel and Blondel and Prettenhofer and Weiss and Dubourg and Vanderplas and Passos and Cournapeau and Brucher and Perrot and Duchesnay, year 2011.

[2] *Collaborative data science*, Plotly Technologies Inc., Montreal QC, year 2015 `https://plot.ly`.

[3] *Oracle Documentation* `https://docs.oracle.com/en/`.

[4] *Angle-Based Outlier Detection in High-dimensional Data*, Hans-Peter Kriegel and Matthias Schubert and Arthur Zimek, Ludwig-Maximilians.

[5] *STL: A Seaonal-Trend Decomposition Procedure* Based on Loess, Robert B. Cleveland and Williamo S. Cleveland and Jean E McRae and Irma Terpenning.

[6] *Feature Bagging for Outlier Detection*, Aleksandar Lazarevic and Vipin Kumar.

[7] *Discovering Cluster Based Local Outliers*, Zengyou He and Xiaofei Xu and Shengchun Deng.

[8] *PyOD: A Python Toolbox for Scalable Outlier Detection*, Zhao, Yue and Nasrullah, Zain and Li, Zheng, Journal of Machine Learning Research, 2019, `http://jmlr.org/papers/v20/19-011.html`.

[9] *Gaussian Mixture Models* Douglas Reynolds.

[10] *Isolation-based Anomaly Detection* Fei Tony Liu and Kai Ming Ting and Zhi-Hua Zhou.

[11] *Isolation Forest* Fei Tony Liu and Kai Ming Ting and Zhi-Hua Zhou.

[12] *Anaconda Documentation* `https://docs.anaconda.com/2`.

[13] *Visual Studio Documentation* `https://code.visualstudio.com/docs`.

[14] *scikit-learn 1.2.1 documentation* sklearn.ensemble.IsolationForest — `https://scikit-learn.org`.

[15] *Advanced Data Mining Techniques* David L. Olson and Dursun Delen January 2008.

Acknowledgements

I would like to thank all the people that stand by me. Particularly my family, who supported me during all my life, and still they are doing, Veronica, my sister, who is the best role model, Alberto, for his patience, and all my friends.