Time Series Outlier Detection

Master project

Aalborg University

Written by

Mik Christensen

June 3, 2021

Contents

1 Summary.

This paper will cover the importans of timeseries data in general, and some of the problems of having abnormal points in the data set. Beside describing the issues of abnormal points in the data set, we are proposing a application for testing different algorithms for detecting these abnormal points. Like describing different options for detecting abnormal points we will be covering, some of the techniques of software development. The different techniques for software development to be covered and used doing the development process are:

- Project life circle management – By Project life circle management we mean the way that the whole development process is organized in order to ensure the costumer gets the right product. Here the Waterfall and the agile method Scrum will be described, and the advantages and disadvantages will be highlighted.
- Application design – Before we as developers will be able to start a new project, we may first have to clarify to the costumer what the reequipments may by. Then it is importantant to understand the context where the new product is going to be used. This part is based on some of the techniques from the OOA&D (Object Orientated Analytic & Design) method. These techniques have the purpose to make the developer understand the domain where the new product is going to be used.
- Database design – Data redundancy or inconsistent is a huge problem to many domains. Therefor we doing this project will propose the database design method called normalization. The purpose of this method is to provide a good structure for the database, without redundant data.
- Source control – It is critical to almost all businesses to be in control when it comes source code. Different tools like GitHub and Microsoft Team Foundation Server, is trying to deal with that. And we will try to clarify why it is that important.

When it comes to abnormal data points these can not be found by using normal classification methods. Because in normal classification we relay on a set of labeled training data, which in the most cases may not be available to the timeseries data set. And in the normal classification we are trying to classify a object to be belonging to one class or another, but in the case of detecting abnormal points we consider all the objects to belong to the same class, with a number of abnormal (outliers) points. This project will cover the algorithms, for detection of outliers:

- OneClassSVM
- IsolationForest
- LocalOutlierForest
- Replicator Neural Network (RMLP)
- OmniAnomaly
- Donut

To evaluate these algorithms the ‘Outlier Score’ is introduced. Because we did not have any labels for the evaluation of the predictions, then we are not able to use the evaluation methods from the normal classification methods. The ‘Outlier Score’ is a number from 0 to 1 telling how normal the object is.

2 Abstract.

Time series data set is highly important to many business areas, that may be healthcare, finance, or the industry. Often in these cases it is not the normal data that is important to the users but instead the abnormal points. Etc. a doctor may not pay that much attention to all the normal heartbeat at an electrocardiogram (ECG) but instead the cases where the heartbeat is differing from normal. Therefor it is a huge research area to find suitable methods to finds these abnormal points.

These time series data set is often unsupervised meaning they do not contain any labeled (information about the object state) training data sets to be used for classification. That is the reason why normal classification can´t be used, in these cases and leads us to the demand of a set of algorithms that can detect these abnormal points without that kind of predefined information.

This project will cover six different algorithms for unsupervised outlier detection. These algorithms will in general base their ability to detect outliers on their ability to learn the normal distribution of the data. And they will try to do this by different methods like:

- Try if some objects can be isolated faster than other objects.
- Using neural networks to learn the normal pattern.
- Using variational autoencoders to learn the normal pattern.

3 Introduction.

Time series the term for data where the same variable has been captured many times, and the value and time stamp are stored as a sequence. This type of data is widely used in many applications such as finance, healthcare [5], or the industry manufacturing [7]. It can be measurement data from a temperature sensor, or an electrocardiogram (ECG) that is used by doctors to monitor the heartbeat. Analyses time series is important because it reveal important information of the time series generator. Figure 1 shows an example of a time series from Yahoo finance.

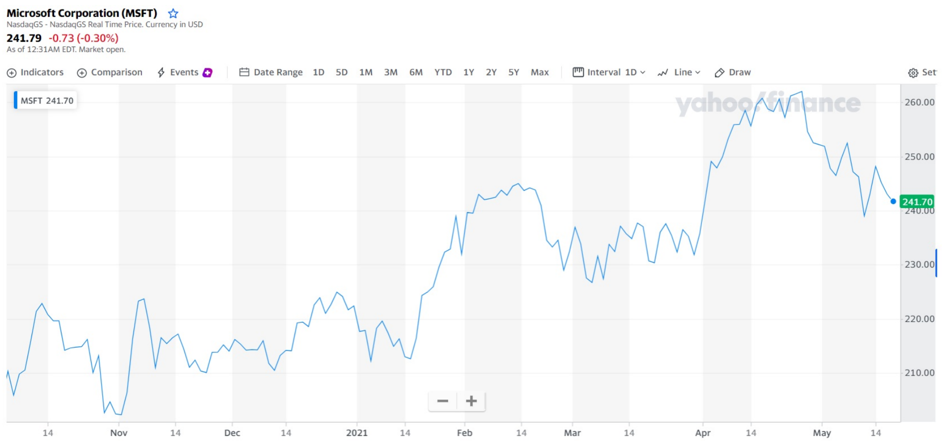


Figure 1: Example of stock time series - Microsoft stock price

Time series data often contain abnormal points, that means a few data points that is captured in the exact same way as the rest of the dataset but is significantly different from the remaining data points. These data points in the literature often referred to as abnormalities, discordant, deviants, abnormal or outliers. The fact that a single point is defined as an outlier, is not necessarily the same as being worthless. Often these objects provide abundant information about the domain where it has been captured. For example, outlier detection at time series may be useful at a hospital where a doctor is using an electrocardiogram (ECG), for monitoring the heart condition of a patient. In such situations a algorithm will be able to learn the normal pattern of the heartbeat signal, and then point out the cases where the beat signal is differing form the normal pattern. The abnormal heartbeats may indicate potential heart attacks. In industry, the production of parts for wind turbines must be tested. When performing these tests, the manufacturing process is equipped with large number of sensors, that is normally being used measure the forces at the material. Some of the ongoing projects in this field is trying to detect cracks in the material, by using outlier detection algorithms, in this case, the business case lays in if a crack is detected a soon as possible it will reduce the downtime for maintenance, and finally be able to perform the test faster. Time series outlier analysis is not a trivial task, it is often involving a huge amount (high dimensional) data. Time series outlier analysis also require expensive cost for domain experts, who have serious understanding the domain data.

Figure 2 Shows example of a time series containing outliers.

Although many algorithms are proposed for time series outlier detection, there is no study that combine all algorithms in an unified frameworks. In this project, we propose

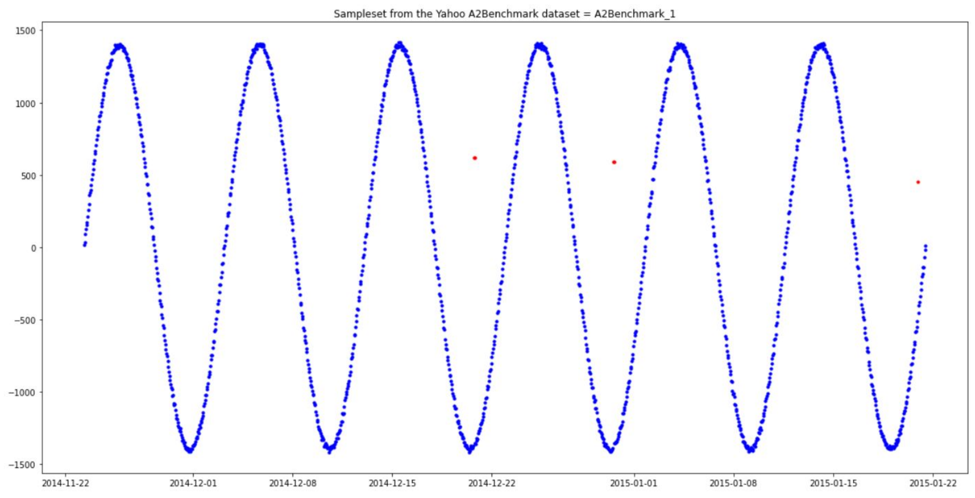


Figure 2: Global outlier example.

an unified framework, which is composed of state-of-the-art algorithm to help the domain experts can have more insight about time series anomaly detection. The framework comprise 6 algorithms which is range from tree-based method to density-based method and neural networks based methods. We employ the framework for extensive evaluation on 200 time series. These time series are collected from large number of domains. The results show that our framework is able to do something.

4 Preliminary & Problem statement.

4.1 Time Series

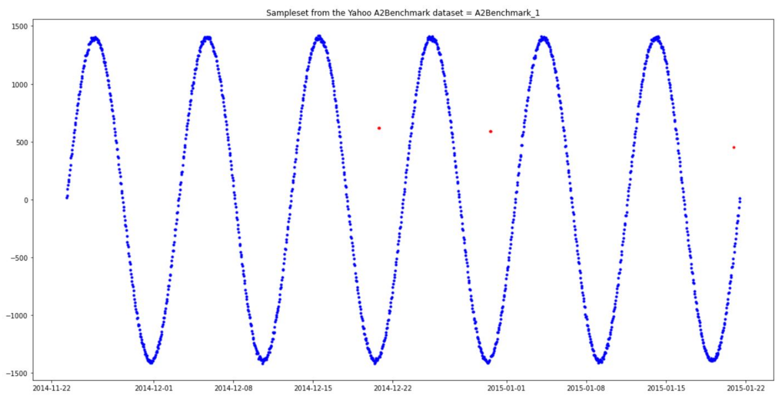
A time series $\mathcal{T} = \langle \mathbf{s}_1, \ldots, \mathbf{s}_C \rangle$ is a sequence of C observations, where each observation $\mathbf{s}_i \in \mathbb{R}^D$. If $D = 1$, $\mathcal{T}$ is *univariate.* If $D > 1$, $\mathcal{T}$ is *multivariate* (or *multidimensional*).

4.2 Time Series Outlier Detection

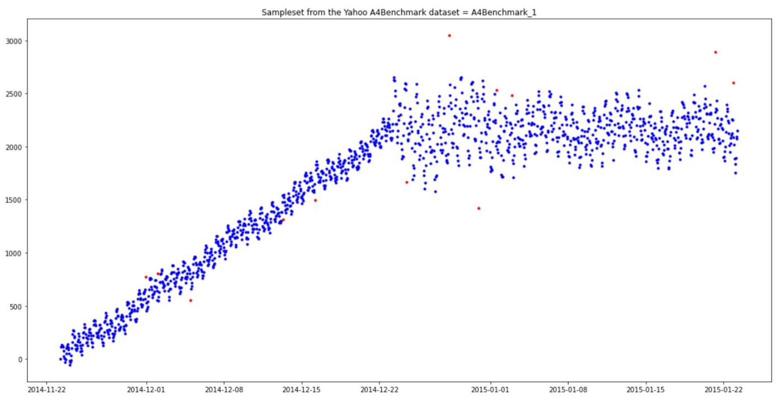
Given a time series $\mathcal{T} = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_C \rangle$, we aim at computing an outlier score $\mathcal{OS}(\mathbf{s}_i)$ for each observation $\mathbf{s}_i$ such that the higher $\mathcal{OS}(\mathbf{s}_i)$ is, the more likely it is that observation $\mathbf{s}_i$ is an outlier.

4.3 Time Series Outlier detection Benchmark Dataset

The dataset that will be used for the first phase of the project, where the application is created is the 'A2Benchmark' and 'A4Benchmark' dataset from *Yahoo* (Webscope — Yahoo Labs). The 'A2Benchmark' contains 100 synthetic time series. And the 'A4Benchmark' contains 100 real-world time series. All the time series in these two data sets contain a label for each observation for indicating if the current data point is an outlier. Figure 3a shows the synthetic time series and Figure 3b show the real-world time series



(a) Synthetic Time Series.



(b) Real-world Time Series.

Figure 3: Example of Time Series in *Yahoo.*

5 Application Overview.

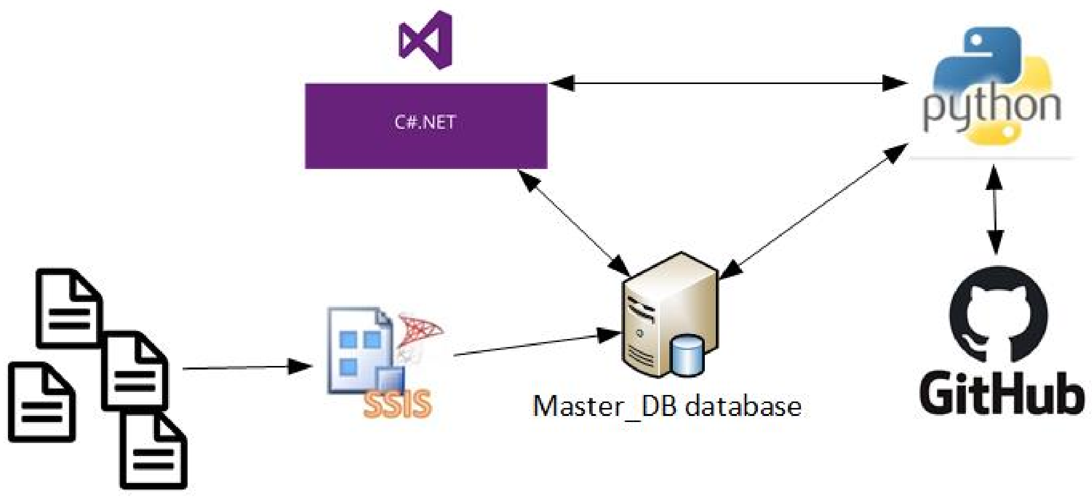


Figure 4: Application Overview.

Figure 4 shows an layout overview of the complete application:

- The Benchmark data set from Yahoo, consist of a number of .csv file, one for each time series. All these files is transformed into a normalized SQL server (Microsoft SQL Server Express 2019) structure, by using SQL Server Integration Services (SSIS).
- The user interface consist of a C#.NET application. This application is only intended to act like a data presentation layer. When a user has configured a test, all information about the particular test is stored in the SQL server, and from the interface a Python application, is then executed.
- The Python application then reads the configuration in the database for that particular test, and executes the outlier detection algorithm. The finally result is stored in the SQL server.
- When the Python application has terminated, the user interface is then showing the results from the database.
- Doing the process of creating the system a GitHub repository has been used to share the Python code with my supervisor at `https://github.com/MikChristensen/OutlierDetection_Mik_Christensen`

This project had been a one-man project, so doing my work I have not been following all the normal development strategies completely. Doing this development process, I have been trying to follow the waterfall model, because it is simple, and easy to understand. But I have not been able to follow it completely, by the reason the number of algorithms for outlier detection was increased over time. The below list shows some development strategies that may be beneficial to follow or at least have in mind doing the process:

- Project management life circle - The way that the complete team around a software project are working together.

- Application design.
- Database design.
- Source Control – A method of sharing and control software elements.

5.1 Project management life circle.

In my daily work, I work in a department where we had tried to implement the agile method called Scrum (And that means we did not follow it completely but uses the parts that make sense), because it good for supporting large ongoing projects. Another popular work model is the Waterfall model.

5.1.1 Waterfall model.

In the waterfall model a software project is starting at the top of a waterfall, as an idea, and doing its journey down the waterfall it changes state multiple times and, in the end, it ends up as a final deployed software. The overall idea is that first when one a phase has been finalized the next one in the chain can be started.

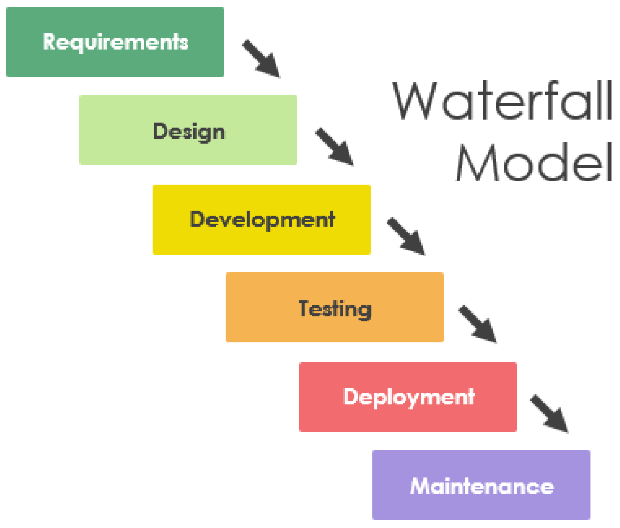


Figure 5: Waterfall model - Structure overview.

Figure 5, shows the structure overview of the Waterfall model:

- Requirements – This is the phase where the customer and developer is going to agree on the project scope. The output of this phase will be a contract that's described the project. This is meant to ensure that the costumer is receiving the expected work and the developer knew the scope of the project.
- Design – The design phase, is where the overall structure of the project is made. That means both the logical and the physical design. The logical design is covering the overall data-flow and actions in the application, then the physical part is covering the hardware part. That means if the project is requiring a SQL server or a Spark computation cluster it is in this phase these decisions is made. And the systems are ordered. Before the design step can be finished a design document must be approved.

- Development – In the Development phase the development task is divided into small chunks, that can be tested individual. When all the small chunks have been tested, the project can continue to the next step.
- Testing – The testing phase is where the whole project is tested as a complete unit. And where the end-user is able test it to find bugs.
- Deployment – In this phase the system is going to production.
- Maintenance – Here the software has been delivered to the end-user and is running in production.

Advantage and disadvantage of the Waterfall model: Advantage.

- Easy to understand.
- Easy to manage.
- Often only a few production issues.

Disadvantage.

- Not flexible.
- Not good for handling unexpected changes.
- Not good for complex for long term projects.
- Difficult to capture all requirements upfront.
- The user will first see the progress at the end.

5.1.2 Scrum.

The Waterfall model is not flexible, or suitable for large projects. That leads to another project life circle management model called Scrum. Scrum follows a completely different approach, when the Waterfall model first returns something to the user in the end of the development process, Scrum is delivering the product in small pieces. The development process is divided into small steps called 'sprints', each sprint consists of four steps:

- Planning.
- Build.
- Test.
- Deliver.

The idea of a sprint is that after each 'delivery' phase, then the user should be able to see a running application, that's going to be improved by each of the following sprints. In that the development process is open for domain changes, and the costumer / user can follow the progress. Depending on the situation or the project these sprints will be running for one to four weeks (it is the normal). A Scrum based project is based on three different roles:

- Product owner – The product owner is also the costumer, he/she got the requirements for the project.
- Scrum master – the Scrum master is a kind of a organizer for the scrum process. Who is trying to lead the team in the right direction. It may be the task of the Scrum master to ensure that the team consist of the right people.
- Team – The team will consist of developers and database administrators or other people who may be a part of the actual work. The Scrum team is intended to be self-organized, meaning it is not the Scrum masters task to tell the team members who should do what.

Figure 6 shows an overview of the Scrum framework.

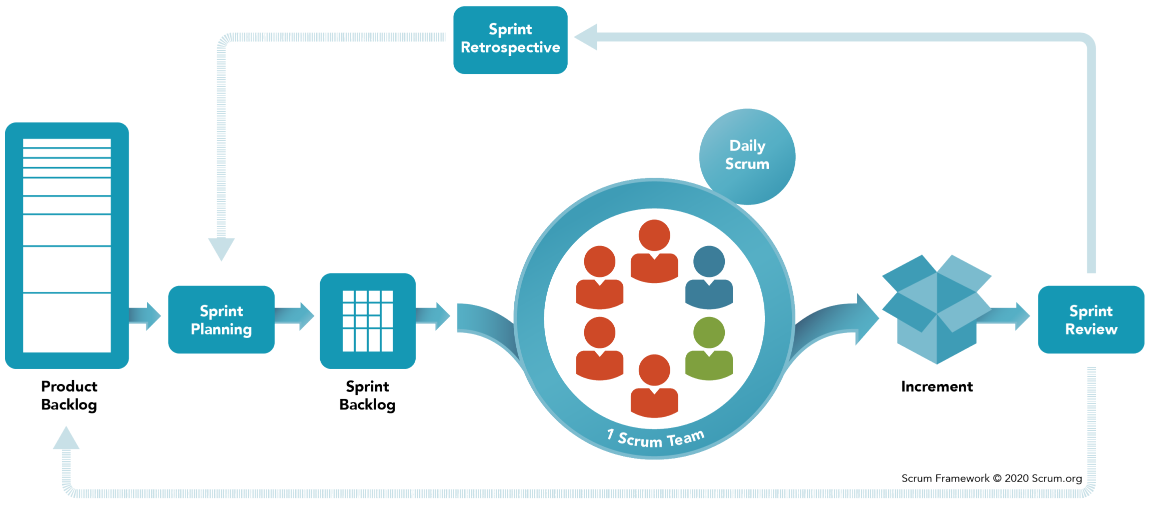


Figure 6: Scrum framework - Structure overview.

- Product backlog – The product backlog is an ordered list of unsolved task or features that is needed to improve the product.
- Sprint planning – The Sprint planning is the event that starts a new sprint. In collaboration the whole scrum team, will figure which backlog items should be implemented, and how it will be done. Another task of this event is to decide the length of the particular sprint.
- Sprint backlog – The Sprint backlog consist of the backlog items from the product backlog, that has been divided in to smaller and easier to implement tasks. It is a backlog for the developer team.
- Daily Scrum – Is a short daily event for all the developers at the project, where they have the opportunity go through the last day process.
- Increment – The goal for the Increment step is to merge all the small pieces from the sprint backlog to a running project.

- Sprint review. The running product is then reviewed, by the product owner, scrum master and the team.
- Retrospective – Is the step where the team can evaluate their work for the past week, so they can improve their work.

Advantage and disadvantage of the Scrum framework: Advantage.

- Easy to the manager or costumer to follow the process.
- Change can be applied doing the development process.

Disadvantage.

- If the development teams are to large Scrum may be a challenge.
- When the project is starting no one had the overview of the final product, that may lead to an unbeneficial code structure.
- The daily scrum meetings may frustrate the developers.
- Difficult to capture all requirements upfront.
- The user will first see the progress at the end.

5.2 Application design.

First a short real-world example from the company where I Work: 'An engineer was planning to implement a feature for variable sample rate to a measurement system. The intension was that the user of the measurement system should be able to simply slide up and down the frequency of the measurement signal.'. In this case the problem is that this new feature may work fine, in all the cases where the other engineers is only monitoring the measurements in a chart. But some of the other users may be applying some postprocessing to the same measurement channel, where they take two channel and transform those to a new calculated channel. And it may give some problems or changes to the output of the calculated channel. My intension for this small example is to make it clear that it is important to the designer of a system to understand the domain where the product is going to be used and the consequences of the different decisions.

Situations as the above one is exactly what the OOA&D design method is trying to deal with. The OOA&D stands for Object Orientated Analytic and Design and is proposed by Lars Matthiessen in the danish book 'Objekt Orienteret Analyse & Design'. The model is intended for object orientated development, but the way it analyses the domain may be applied to all other software projects to.

OOA&D tries to enforce the following principles:

- Try to modularize the domain. Meaning that a useful system must be designed to match the specific domain, and therefor it is important to understand the domain.
- Highlight the architecture. The idea is spent time on a good architecture. It is important to have a flexible and understandable architecture.

- Reuse components. Make the different modules as reusable as possible, it will speed up the development process and improve quality.

The first tasks the OOA&D suggests is to draw a 'rich' diagram of the domain, where the new model is going to be used. By rich diagram they mean a very basic drawing of the domain and situations where the new product is going to be used (Issues like the real-world example above, may have been avoided by creating such a diagram). The diagram should be used to show the different use cases in the domain, an increase the general understanding of the problem. Figure 7 shows, an example of an rich diagram.

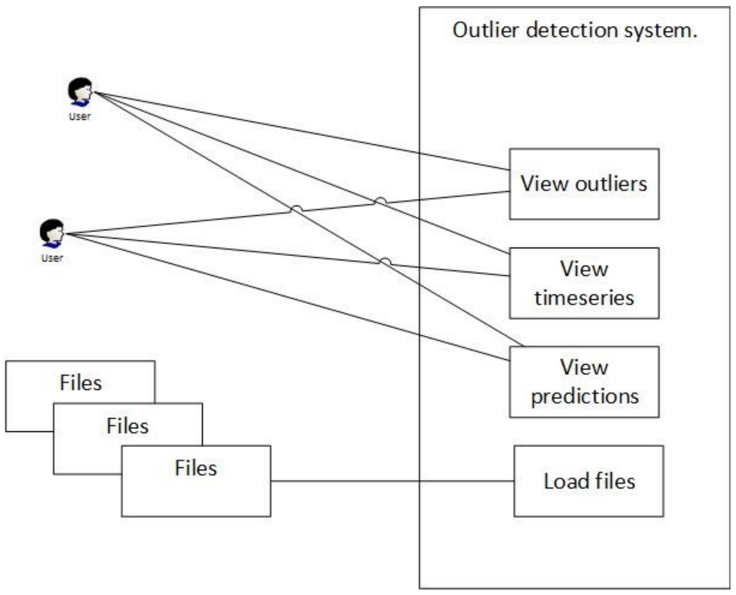


Figure 7: Rich diagram.

The next task is to modularize the domain, the model is suggesting doing this by creating a table with class candidates and actions. As shown in table 1.

	DB	**OneClassSVM**	**LOF**	**SSIS**	**Interface**
Add Configure	X				X
Read configure	X	X	X		
Display prediction					X
Display timeserie					X
Display outliers					X
Store result		X	X	X	
Store execution time		X	X	X	
Read source file				X	

Table 1: Event table

From the above table 1, the needed classes may be found. These classes can then be represented in a UML diagram. Figure 8 shows, a UML diagram is used to show relationships between classes, in the python part of the project. Where the SQL class is represented as a super class to the rest of the classes. Meaning the other classes can inherit the objects from that class.

Finally the behavior of the different objects, can be described as in figure 9.

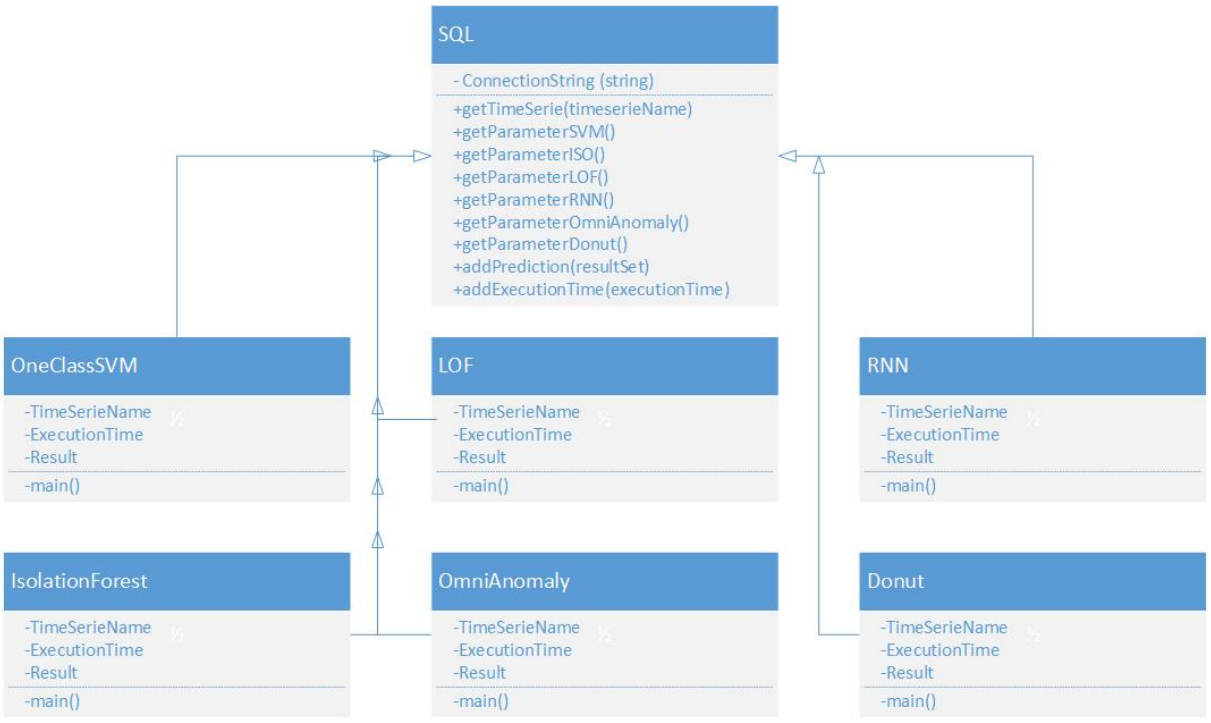


Figure 8: UML diagram for the Python part of the project.

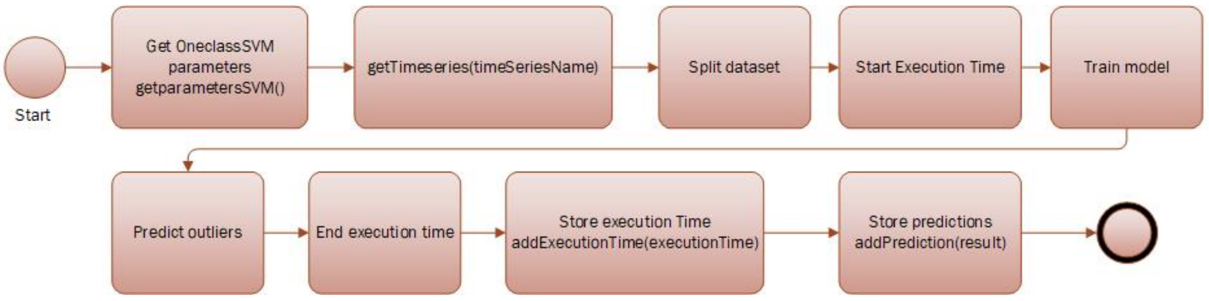

Start
Get OneclassSVM parameters getparametersSVM()
getTimeseries(timeSeriesName)
Split dataset
Start Execution Time
Train model
Predict outliers
End execution time
Store execution Time addExecutionTime(executionTime)
Store predictions addPrediction(result)

Figure 9: Behavior of the OneClassSVM class.

5.3 Database design.

Data redundancy in a database may lead to problems in many different cases, as an example if you are trying to update or delete a particular value in the database, how then to be sure that you are updating the value all the places. Or if you have a field 'firstName' in two different tables and they contain two different names for the same person which on is then the right one? To overcome that kind of problems the database may be normalized. The data normalization is a design method consist of six different steps called normal forms. It is not necessary to normalize through all the six normal forms, but instead simply just normalizing until the data quality is matching the requirements of the system. For the case of this project, I was not able to normalize further than the third normal form. The normalization steps are:

- **NF1** – Criteria for at table to be at the first normal form.
  - All columns must be atomic (only containing a single value).

  - All values of a single column must be of the same type (not only data-types but also, the meaning of the value).
  - Each column must have a unique name.
  - Each row in the table must be equally identified by a primary key. That primary key may be a natural or surrogate key.

- **NF2** – Criteria for at table to be at the second normal form.
  - Must be normalized to **1NF**
  - And there may not be any partial dependencies between the columns, a column that is only dependent on a part of the primary key.

- **NF3** – Criteria for at table to be at the third normal form.
  - Must be normalized to **2NF**
  - And there may not be any transactive dependencies between the non-primary key columns (as an example there may be a partial dependency between zip code and city).

- **BCNF** – Criteria for at table to be at the Boyce-Codd Normal Form (**BCNF**).
  - Must be normalized to **NF3**
  - If multiple natural keys are used to form the primary key, then, then all non attributes must be a part of the candidate key.

- **NF4** – Criteria for at table to be at the fourth normal form.
  - Must be normalized to **BCNF**.
  - There should be no multiple value dependencies.

Figure 10 shows the normalized table structure for the project. I had often heard people saying that they fear that the structure is going to be to complex, and they will get a performance decrease at the database when splitting up all the tables, because they in some cases risk to create long join statements. My own experiments are that is not the case, because when we have smaller tables, it is often easier to apply the right indexes. And that experiments are based on the normalization of a 130TB SQL Server containing turbine measurement data, where we in the end got much less blocked sessions.

One way to handle the complexity question, may be to apply an abstraction level in form of views and stored procedures. For this project, the interface between the different software components and the database has also been defined as a set of views and stored procedures. Another advantage using views and stored procedures will be in a security context, it is easier to provide access to a single object instead of multiple tables. That means for this small software project I had made the following objects:

- VI_DonutParameters - View to be used for the Python application to extract the last DoNut test configuration.
- VI_IsolationForestParameter - View to be used for the Python application to extract the last IsolationForest test configuration.

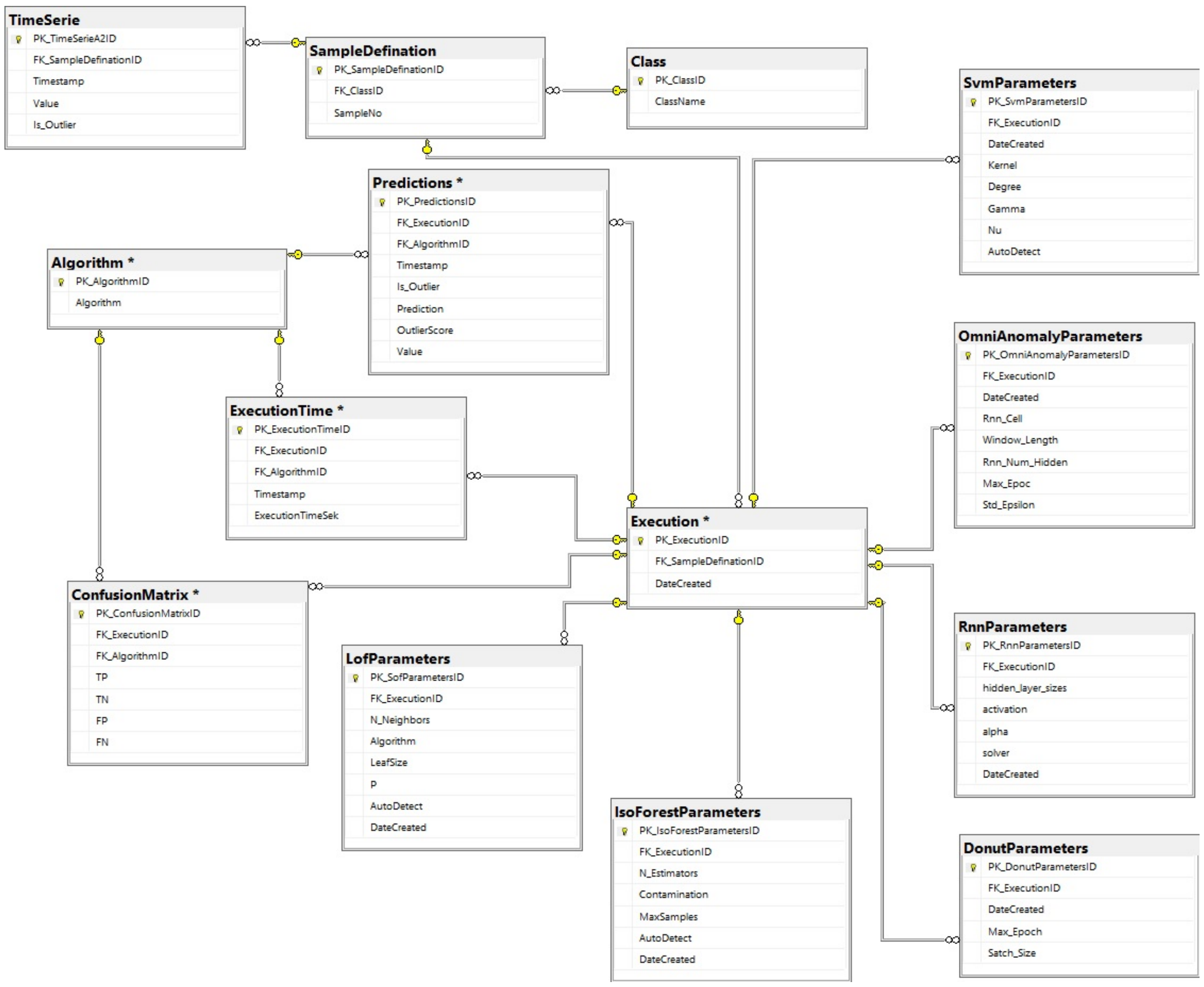


Figure 10: Normalized database structure.

- VI_LocalOutlierFactorParameter - View to be used for the Python application to extract the last LocalOutlierFactor test configuration.
- VI_OmniAnomalyParameters - View to be used for the Python application to extract the last OmniAnomaly test configuration.
- VI_RnnForestParameter - View to be used for the Python application to extract the last RNN test configuration.
- VI_SvmParameters - View to be used for the Python application to extract the last OneClassSVM test configuration.
- VI_SampleDefination - View to be used for the C#.NET user interface to extract the name of the different time series. All the time series from the *Benchmark2* and *Benchmark4* data collections contains an identification number, but that cannot be used as a meaningful name for the application. So, inside the view the belonging class of the time series and the number is combined like '*Benchmark2_1*'.
- VI_Prediction – Returns prediction of the last performed test, to be used in the user interface.

- VI_TimeSerie – View to be used in the Python code to return, all time-series in the database.
- SP_AddParametersDonut – Used by the C#.NET user interface to store the DoNut configuration.
- SP_AddParametersIsolationForest – Used by the C#.NET user interface to store the IsolationForest configuration.
- SP_AddParametersLocalOutlierFactor – Used by the C#.NET user interface to store the LocalOutlierFactor configuration.
- SP_AddParametersOmniAnomaly – Used by the C#.NET user interface to store the OmniAnomaly configuration.
- SP_AddParametersOneClassSVM – Used by the C#.NET user interface to store the OneClassSVM configuration.
- SP_AddParametersRnn – Used by the C#.NET user interface to store the RNN configuration.

It may seem to be a little bit elaborately, to create objects for all the operations interacting with the database, but it makes the life of a database administrator much simpler. Because changes can be made to the underlying table structure without breaking down the whole chain of application using the database.

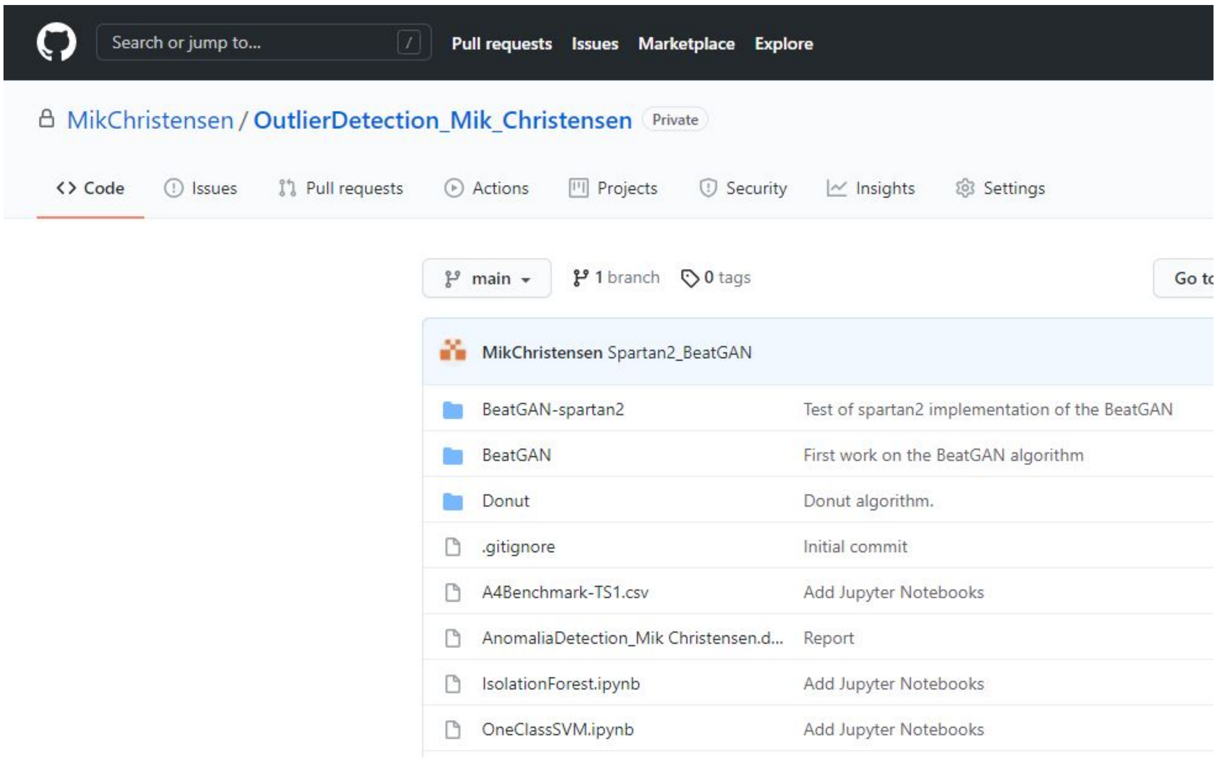
5.4 Source Control.

Source control is intended to track and manage changes to a project over time. And by project it means not only code projects, but it could also be a set of documents that often is going to be changed. The idea is to apply a system to the projects that always take care of the project so we as developers did not end up in situations where we are not able to find the code. Or as an example if multiple developers is working on the same project and they must share the code. Then they can push all their changes to the Source Control system. Another developer can then pull the changes into his own environment.

One of the advantages of using a source control system is that, as a team you can have multiple branches of the project. As an example, you may have the main branch containing the version of the code running in production. And another branch may contain all the development tasks, the two tasks can then later be merged.

For this project I had been used a free version from GitHub.com. But in the company where I work, we have been using GitHub and Microsoft Team Foundation Server, both the tools provide the ability to the development teams to setup all the stages of their development chain for their backlog items.

Figure 11 shows the GitHub repository for the project.



MikChristensen / OutlierDetection_Mik_Christensen Private

MikChristensen Spartan2_BeatGAN

BeatGAN-spartan2	Test of spartan2 implementation of the BeatGAN
BeatGAN	First work on the BeatGAN algorithm
Donut	Donut algorithm.
.gitignore	Initial commit
A4Benchmark-TS1.csv	Add Jupyter Notebooks
AnomaliaDetection_Mik Christensen.d...	Report
IsolationForest.ipynb	Add Jupyter Notebooks
OneClassSVM.ipynb	Add Jupyter Notebooks

Figure 11: GitHub repository for the project.

6 Proposed algorithms.

6.1 Unsupervised Outlier Detection.

The following section contains a set of unsupervised algorithms for outlier detection. By unsupervised algorithms it means that the algorithms did not requires any label, with information about if the object is an inlier or outlier. Unsupervised data sets is the most normal type of data to be used in timeseries outlier analytic, because it is a highly expensive task to generate that information.

6.1.1 One-Class SVM.

The book ‘Outlier Analysis [4] describes the One-Class SVM (OCSVM) algorithm as a modification of the normal Support Vector Machine (SVM) algorithm used for two classed classification. Where the main problem to the original SVM algorithm, is that it has been made to classify a set of objects to belonging to one class or another class. And another problem to the SVM algorithm is that the outlier detection area is most often an unsupervised task, where the complete data set is expected to be belonging to the same class, and no labels is available. In outlier detection the incoming data is expected to belong to the same class, but it may contain a few outliers. When the regular SVM is trying to classify the objects by separating the objects by a margin hyperplane, then the the OCSVM is trying to define a decision boundary around the normal points to excludes the outliers. Figure 12 Shows an illustration from Scikit-Learn that describes how the decision boundary is working.

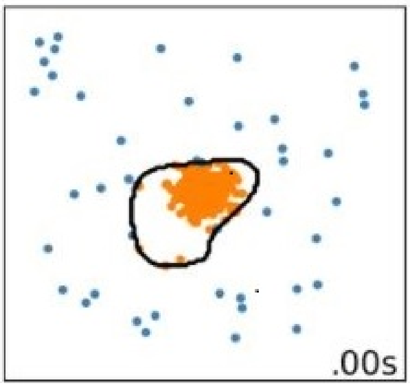


Figure 12: OCSVM - Decision boundary

$$\bar{W} \cdot \varphi(\bar{X}) - b = 0 \tag{1}$$

Equation 1 shows the equation for defining a linear decision boundary.

The above equation defines the corresponding decision boundary that's separate inliers from the outliers. $\bar{W}$ represents the input vector, and the $\varphi(\bar{X})$ defines the kernel function that is used to transform the distribution to an higher dimension. And finally the bias is represented by the b. If the value for this equation is positive the object is defined as an inlier, and if negative it will be mark as an outlier. But the above equation should be optimized, in order to ensure that as many as possible of the training samples N turns out to be positive (because all the training samples is expected to belong to the positive class). Equation 2 shows the optimized formula for defining the decision boundary.

$$J = \frac{1}{2}||\bar{X}||^2 + \frac{C}{N}\sum_{i=1}^{N}\max(b - \bar{W} \cdot \varphi(\bar{X}), 0) - b \tag{2}$$

The constant C defines the differential weight of the normal points and the outliers. That means trade-of between positive and negative classifications can be regulated by modifying the C value.

When a model has been trained a score value, can be calculated for all the points in the distribution. This score value describes how derived a value is from the decision boundary. Equation 3 shows how to express the score of deviation from the decision boundary.

$$Score(\bar{X}) = \sum_{i=1}^{N} \alpha_i \cdot K(\bar{X}, \bar{X}_i) - b \tag{3}$$

6.1.2 Local Outlier Factor.

The Local Outlier Factor (LOF) is a algorithm that makes it's assumption of an object being an outlier based on a score called local outlier score. LOF is a density-based algorithm that means that the algorithms assume that the outliers are located on low density area. The algorithm compares the local density of a point to the local density of K of its neighbours. If a point got significantly lower density than its neighbours, it is most likely to be an outlier. But to decide if the outlier factor is variating enough to be an outlier may be a decision for a domain expert. The first step is to choose a value for K, which is a user defined constant parameter. Finding the right value for K may not be that easy, because small value will make the algorithm focus on objects very near to the point. But otherwise, a large value may expand the focus area to risk much and then missing an outlier. The distance from a point to the Knearest point is called k-distance. Figure 13 illustrates the k-distance of the red point, where k=3. The distance measure can be both the Manhattan or Euclidean distance.

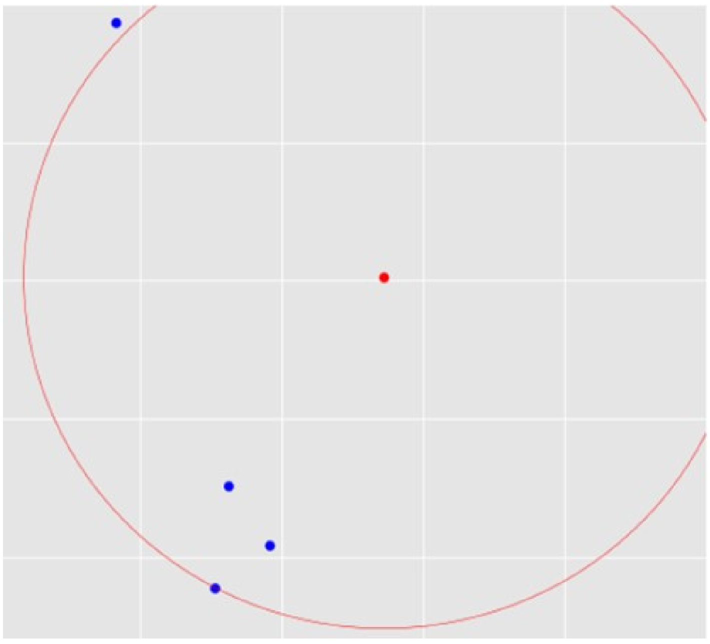


Figure 13: LOF - k-distance

The k-distance value is then used to calculate the reachability-distance. The reachability-distance is the maximum distance between two points, typically it will be equal to the K-distance. Equation 4 shows the expression for the reachability-distance.

$$reach - dist(a, b) = \max(K - Distance(b), Distance(a, b)) \tag{4}$$

Then the average reachability-distance for a point a can be calculated. The average reachability-distance is the average of reachability-distance for in the neighbourhood of a particular object. Equation 5 shows the expression for the average reachability-distance.

$$ard_K = \text{mean}_{y \in L_{k(a)}}(reach - dist(a, y)) \tag{5}$$

Finally, the LOF value can be calculated for all the object. Equation 6 shows the expression for the LOF value.

$$LOF_K(a) = \frac{ard_K(a)}{\text{mean}_{y \in L_{k(a)}} ard_K(y)} \tag{6}$$

6.1.3 Isolation Forest.

In short terms the Isolation Forest (ISF) algorithms, is based on several isolation trees, the core idea behind these isolation trees is that the object that first is being isolated will most likely being an outlier. Following the literature this way of detecting outliers should be performing similar to the LOF. This type of outlier detection algorithms is most likely to be detecting global outliers, because they are located in a spars distributed area, and therefor most likely to be quickly isolated. But just creating one single tree may not give a precise picture of the data set. So multiple trees are created and the outlierness of an object will be based on the average path of an object. And the one with the shortest average path is likely to be an outlier. Figure 14 below is illustrating that first a splitting point is randomly selected at 120 at the X axis and only one of the points Is above 120 so it is isolated by the first split. And next a split is chosen at the Y axis, which turns out to isolate the next object. That's the way the Isolation trees is created.

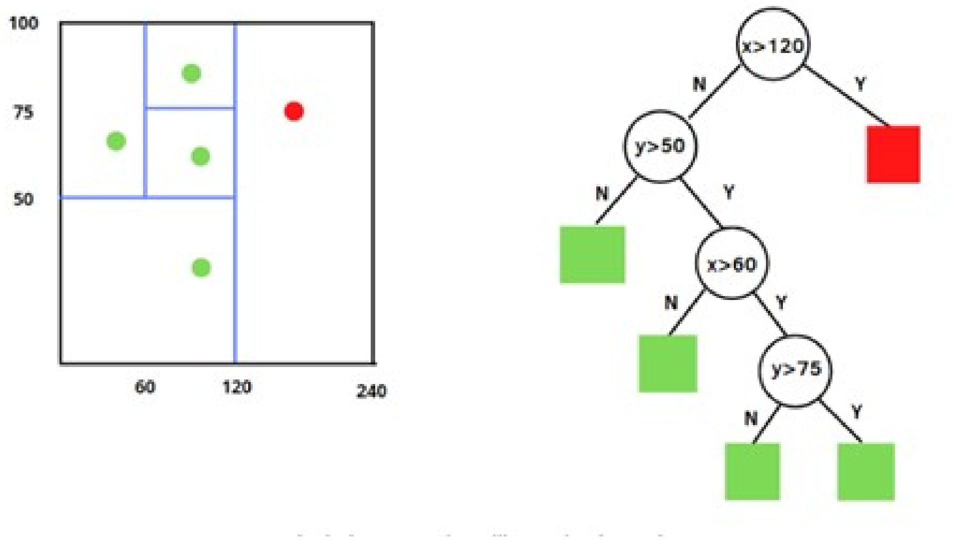


Figure 14: ISF - Data points isolated by tree.

Just because an object has the shortest average path, it is not the same as it must be an outlier. But to get an idea of how anomalous an object is, an outlier score must be calculated. And based on this outlier score a domain expert may be able to identify outliers. Equation 7 shows the expression for calculating the outlier score.

$$S(X, n) = 2 - \frac{\text{mean}(path(x))}{c(n)} \tag{7}$$

It may seems like a hard task to create trees that splits the complete training sets. But the report [8] by Zhi-Hua Zhou says that's not necessary. The ISF algorithm turns out to be performing well even when only doing the isolation process on a subset of the training data.

Algorithm 1 shows the pseudo code for generating the isolation forest doing the training process. Here the input parameters is the training data set, the number of trees in the forest and the sub sampling size. The sub sampling size describes the number of points to

Algorithm 1: Isolation Forest Algorithm

```
    Input: Input data $X$, number of trees $t$, sub-sampling size $\psi$
    Output: set of $iTrees$
1   Set height limit $l$ = ceiling($\log_2 \psi$);
2   for $i = 1$ to $t$ do
3       $X' \leftarrow sample(X, \psi)$;
4       $Forest \leftarrow Forest \cup iTree(X', 0, l)$;
5   return $Forest$
```

be isolated at the isolation trees. At line two a value for the maximal height is set, this height is used to avoid separating the tree unnecessary.

6.1.4 Replicator Multi-layer Perceptron.

Hawkins propose an outlier detection method where a multi-layer neural network is used to detect outliers [6]. Neural network is also known as replicators and that is where the algorithm got its name from. Neural Network is a way to try to transform the way the human brain works into a software application.

The paper by Simon Hawkins implements its algorithm by using the Replicator Multi-layer Perceptron (RMLP) algorithm. The RMLP algorithm is a supervised learning algorithm that tries to learn the normal distribution of the data set and based on this learning returns a classification. Figure 15 shows the overview of the MLP algorithm. In the first part we have the input layer taking our input data set. In the middle we have the hidden layer (the number of hidden layers will vary, between implementations, in order to improve the learning phase) containing three neurons, the and finally we got the output layer. All these tree layers is then connected and all the connections is applied a weight. Then inside the neurons the input value and the wight is multiplied and added a bias value the result is then inserted in to an activation function, that returns a value the algorithm will be using for its classification.

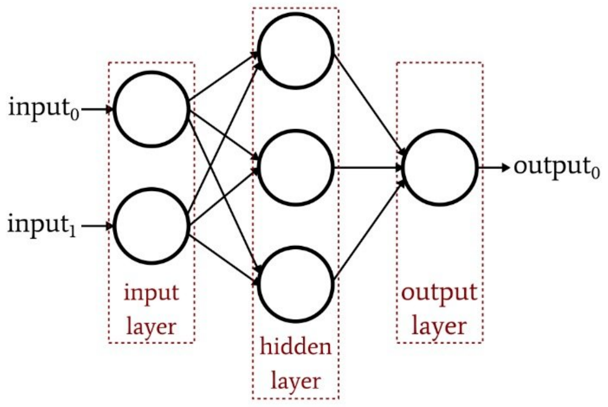


Figure 15: MLP - Structure overview

In the RMLP algorithm the input variables are also the output variables, that means that the RMLP is trying to reproduce the input pattern in the output. Doing the training phase, the reconstruction error is calculated and then used to adjust the internal weights for improving the model. Finally, the reconstruction error is then used to measure the outlierness of an object, to be understood that an object with a high reconstruction error is most likely to be an outlier, because the normal points is easier to reconstruct.

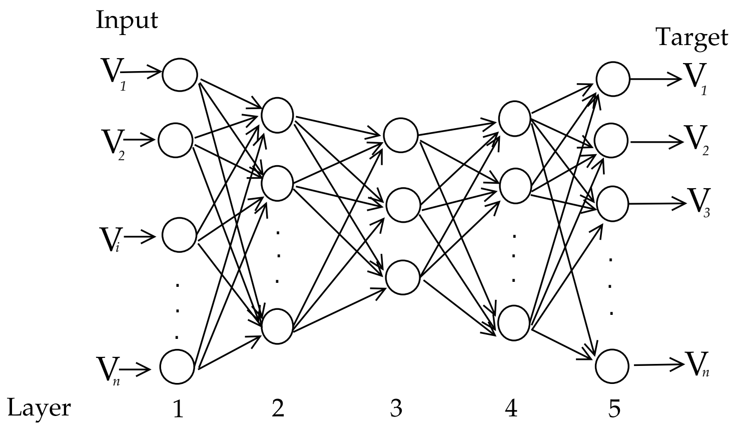


Figure 16: RMLP - Structure overview

Figure 16, shows the structure of the RMLP. At layer 1 the data set is provided, then a network of units is trying to recreate the shape in step 2-4. Then in layer 5,the reconstruction error is measured, and in the training phase this error is used to adjust the weights of the unit. For all the objects in the data set a Outlier Factor is calculated, in order to describe the outlierness the object. Equation 8, shows the expression for the outlier factor:

$$OF_i = \frac{1}{n}\sum_{j=1}^{n}(x_{ij} - o_{ij})^2 \tag{8}$$

Based on the idea of trying to recreate the pattern of the incoming data at the output level, RMLP algorithm may seems to be an autoencoder. But the construction of the two types of algorithms is different.

6.2 Variational Autoencoder based algorithms.

The following section will describe the outlier detection algorithm DoNut and OmniAnomaly, who have in common that they both has been based on Variational Autoencoder (VAE). A normal auto encoder is a tool for dimension reduction, by roughly speaking it can be described as a data compression existing of an encoder and a decoder part. The encoder will use a neural network to transform the input data to a new dimension known as a latent space. Following a decoder part will take this latent space and try to convert it back to the original state. The difference between the original data set and the decoded data is called the reconstruction error. This reconstruction error is then back-propagated to update the internal weights of the neural networks, and in this way try to improve the performance of the encoder and decoder as a pair. The normal neural network is returning a single value to describe the output. To avoid the risk of overfitting the VAE is a little bit different. The latent space of the VAE is not returning a single value but try to provide a normal distribution representation of the incoming data set. The latent state consists of two vectors one representing the mean values of the distribution and another representing the variance as shown in Figure 17.

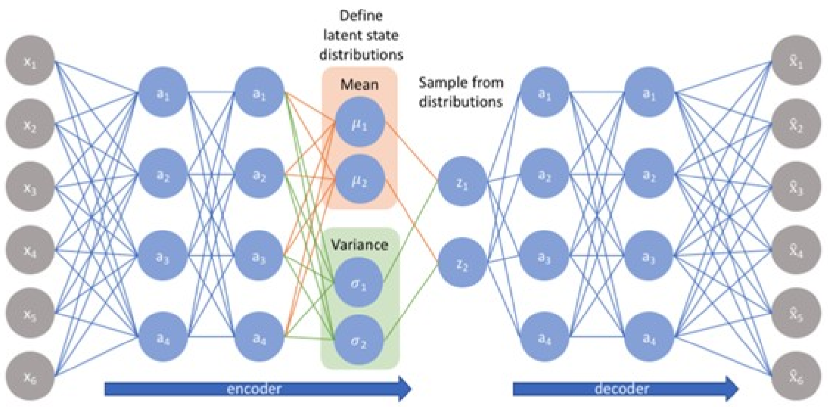


Figure 17: VAE - Structure overview

In the step called '*sample from distribution*' the two vectors from the latent space are passed into the following equation which Is known as the reparameterization.

In the reparameterization step the mean μ and the standard deviation σ is multiplied with the epsilon.

$$Z = \mu + \sigma \cdot \epsilon \tag{9}$$

6.2.1 DoNut.

About algorithm: The paper [11] 'Unsupervised Anomaly Detection via Variational Auto-Encoder for seasonal KPI in Web Applications' is describing the outlier detection algorithm DoNut (an easier understandable version was found at acolyer.org [1]). DoNut is an unsupervised algorithm based on a variational auto encoder for outlier detection, it can work completely without any labels (reason why in is unsupervised), and then it is able to take advantages of a non-complete set of labels. It may be normal that a time series is missing some data points, those missing points is filled out with null values, but the algorithm is not handling those as being outliers. The network structure of the DoNut algorithm is very similar to the one used for variational auto encoders (VAE). But the VAE is not intended to work with time series data set, to handle this problem the DoNut algorithm is prepossessing the incoming data set, dividing the data into sliding windows (Let's say the size of the sliding window is set to 90 then the first window will contain values from 0-90. The second sliding window will contain values from 1-91 and so on.). At the training phase the VAE is trained. A part of that training is to optimize the ELBO (Evidence Lower Bound). The training data set may include missing points that has been replaced with null values, these will be excluded from the ELBO calculation and a scaling factor. Therefore, it is called Modified-ELBO. Then a step called 'Missing Data Injection', is inserting some point without data (null values), to train the VAE to reconstruct these points. At the final 'Detection' phase, we would like to know how likely a observation is. Which can be estimated the reconstruction probability from the VAE. But when doing so the missing values that has been set to 'null' in the prepossessing phase may probably disturb the result, to avoid that problem the 'MCMC Imputation' step is trying to guess what the original value may had been. And then the DoNut algorithm may be able to return a probability that saying how likely an object is to be an outlier.

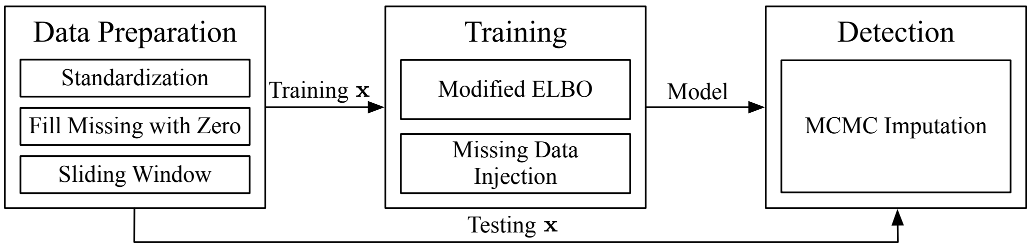


Figure 18: DoNut - Structure overview

6.2.2 Omni Anomaly.

Ya Su et al. [10] propose an unsupervised outlier detection algorithm called OmniAnomaly. This paper claims that the algorithm is outperforming other algorithms like the DoNut. The idea behind the OmniAnomaly algorithm is to is to learn the normal pattern of a time series, and then use the reconstruction probability to decide the state of the object. And based on this learned pattern compute an anomaly score for the outlierness of each object. To determine if an object is an outlier an version of POT (peak over threshold), is implemented to validate anomaly score. Figure 19, describes the overall layout of the OmniAnomaly algorithm, where the solid line is denoting the offline training and the dashed line is denoting the online testing.

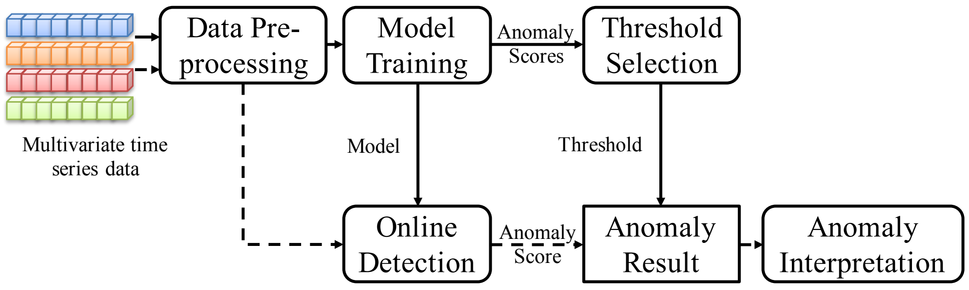


Figure 19: OmniAnomaly - Structure overview

Understanding the diagram:

- Data pre-processing – At the pre-processing phase, the data set is normalized. And then divided into a sequence of sliding windows. By sliding windows, it means that if we set the windows size to 90 then the first window will be based on the elements from 0-90, an the next window will contain elements from 1-91, and continue like that. In that way the data set is transformed into a format that can be handled by the VAE.
- Model training – First a GRU network is used to capture the most important movements in the data set. A GRU network is a neural networks that is able to capture the important part of a data set, and then remember what it has been learned earlier. Then a normal VAE is trying to learn the pattern of the time series.

The outcome of this phase is an anomaly score for each object, that describes how well the object matches the learned pattern. Unlike a normal outlier score this anomaly score denotes an object that is highly different from the model with a low score.

- Threshold selection – At the threshold selection phase, a mechanism following the principles of the Extreme Value Theory, is applied to choose a threshold to define an object as an outlier or not.
- Online detection – At the Online detection phase, the new unseen data is fitted to the trained model from the 'model training' phase. And an anomaly score is returned.
- Anomaly result – If the returned anomaly score is below the threshold from the 'threshold selection' phase it is denoted as an outlier.

A model is first trained by taking a set of historical data and then pass these through the 'Data prepossessing','Model training' and 'Threshold selection'. This part of the process is known as the offline model training. When detecting outliers at normal data the data set is first send through the 'Data prepossessing', in order to be normalized and segmented into sliding windows, then the 'Online detection' part takes the trained model from the offline 'Model training', and use that model to calculate an anomaly score. Finally the 'Anomaly Result' takes the anomaly score and apply it to the threshold from the offline 'Threshold selection', to label the objects as outliers or inliers.

7 Experiments.

7.1 Performance measures.

Most often outlier detection is an unsupervised task, and for that reason it is difficult to describe how a model performs in each case.

7.1.1 Outlier score.

Outlier Score is a probability measure for binary classification, telling how sure the model is of an object belongs to a given class. The score goes from 0 to 1, where 0 means that the model predict the object as an inlier and 1 means the object has been predicted as outlier. One of the benefits, by using an Outlier Score is that it can be applied to unsupervised data sets, and it is easy for a user to understand. The two data sets used for this task *A2Benchmark* and *A4Benchmark* contains labels for outlier detection (the data set is supervised). Then the normal measures for classifications can be performed, like the *ROC* curve. So, the outlier score will be supported by a *ROC* curve.

7.1.2 *ROC* Curve.

The outlier score was returning the probability of an object belongs to a certain class. Then the *ROC* curves are a measure function for supervised learning, and it returns a plot showing the True Positive Rate and the False Positive Rate. The best possible curve is a curve that reach as fare possible, up in the upper left corner. The False Positive Rate (*FPR*) is calculated by dividing the number of False Positive predictions, by the number of False Positive (*FP*) and True Negative (*TN*) predictions. The *FPR* (Equation 10) describes how well the model predicts the False Positives when the outcomes are negative.

$$FPR = \frac{FP}{FP + TN} \tag{10}$$

The True Positive Rate (*TPR*) is calculated by dividing the number of True Positive (*TP*) predictions, by the number of True Positive (*TP*) and False Negative (*FN*) predictions. The *TPR* (Equation 11) describes how well the model predicts the True Positives when the outcomes are positive.

$$TPR = \frac{TP}{TP + FP} \tag{11}$$

7.1.3 Extreme Value Theory (EVT).

Some algorithms [9] for outlier detection did not return an outlier score in the range of 0 to 1. Instead, they return an anomaly score, this anomaly score fare beyond this range. But this anomaly score is still needed to be able to decide if an object is an inlier or outlier. To overcome this problem the Peak Over Threshold (POT), from the Extreme Value Theorem (EVT) may be applied, to find a threshold for deciding if an object is an inlier or outlier. Basically, the POT method says that all values that exceeds a certain threshold may be an outlier. Equation 3 shows the expression, to define the POT theory:

$$F_t = P(X - t \leq x | X > t)) \frac{\sim}{t \to T} (1 + \frac{\gamma x}{\sigma(l)})^{-\frac{1}{\gamma}} \tag{12}$$

Figure 3 shows the pseudo-code for implementing the POT algorithm. The steps:

1. Initialize the procedure – pass time series $(X_1, \ldots, X_n)$ and the risk parameter (q).
2. In the initialization step, a threshold t is chosen. In practice the value of t is often set to a high empirical quantile (98%).
3. Y_t - finds all the peaks higher than the threshold t.
4. The Grimshaw trick is applied to fins extreme candidates between the values from step 3.
5. Finally the threshold is calculated.
6. Threshold is returned

The issue to the POT algorithm is that, before it can be used, we must specify a threshold. And that may be a hard task that requires some knowledge about the dataset. And then it has some limitations for streaming data. For stationary datasets another version of the POT algorithm is available, called Streaming Peak Over Threshold (SPOT). The goal for the SPOT algorithm is to detect outliers in streaming or time series data, without any knowledge about the distribution. It works by first performing the POT algorithm on the first n values, this gives an initialization threshold. That is used to detect outliers, doing the process the threshold is getting updated.

Algorithm 3: Pseudo-Code for the POT algorithm.

```
   Input: $X_1, X_2, \ldots, X_n$, $q$
   Output: $z_q$, $t$
1  $t \leftarrow$ SetInitialThreshold($X_1, X_2, \ldots, X_n$);
2  $\mathbf{Y}_t \leftarrow \{X_i - t | X_i > t\}$;
3  $\hat{\gamma}, \hat{\sigma} \leftarrow$ Grimshaw($\mathbf{Y}_t$);
4  $z_q \leftarrow$ CalcThreshold($q$, $\hat{\gamma}$, $\hat{\sigma}$, $n$, $N_t$, $t$);
5  return $z_q$, $t$
```

7.2 One-class SVM experiments.

The implementation of the OCSVM algorithm is based on the Scikit-learn implementation [3] of the algorithm. This implementation takes the following parameters:

- *kernel* – the kernel function is used to specify the function to transform the data set to higher dimension.
- *degree* – is used for the 'poly' kernel.
- *gamma* – defines the force of each object to the boundary.
- *nu* – the proportion of expected outliers.

The outcome of this implementation:

- *predict* – Returns an array where -1 means outliers and 1 means inliers..
- *score_samples* - A score value defining diversion from the decision boundary.

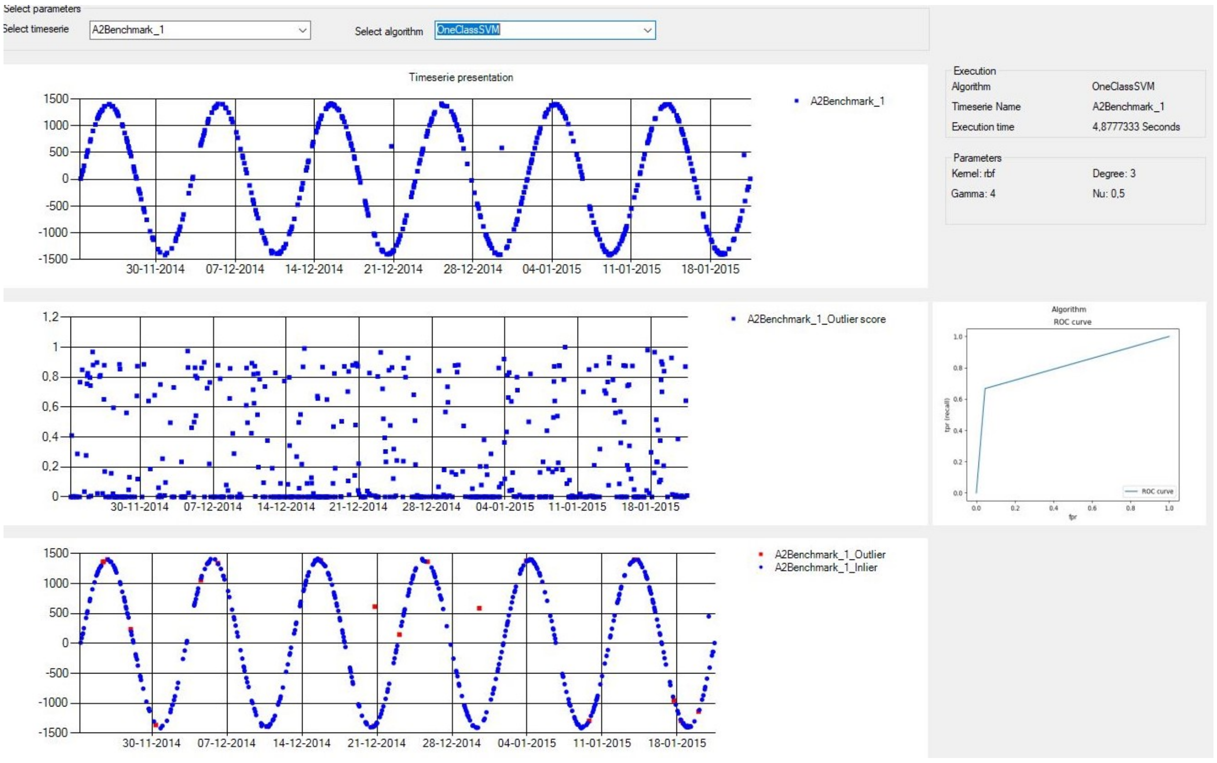
To gather some insight into the performance of the OCSVM algorithm, I was running the algorithm a few times against the synthetic data set *A2Benchmark_1*, and all the time adjusting the parameter. When a parameter with a positive effect to the *ROC* value was found, then it was optimized in that direction.

kernel	**degree**	**gamma**	**nu**	**ROC**	**execution time**
rbf	3	3.0	0.5	0.54	0.3
rbf	2	3.0	0.5	0.32	0.3
rbf	4	3.0	0.5	0.42	0.3
rbf	3	2.0	0.5	0.27	0.5
rbf	3	4.0	0.5	0.31	0.4
rbf	3	4.0	0.5	0.62	0.3
rbf	3	5.0	0.5	0.28	0.3
rbf	3	4.0	0.2	0.41	0.3
rbf	3	4.0	0.5	0.46	0.3
linear	3	3.0	0.5	0.34	137.9
linear	2	3.0	0.5	0.65	407.2
linear	1	3.0	0.5	0.35	284.9
linear	2	2.0	0.5	0.64	153.2
linear	1	3.0	0.2	0.41	44.5
linear	1	3.0	0.7	0.40	211.8

Table 2: OCSVM - Execution results for the synthetic data set

The table 2, shows the results of all these tests. I was expecting the '*nu*' value to have a stronger effect to the result. And then I find it very suppressing that changes to kernel function, was giving that high difference in the execution time. To me it may seems like '*linear*' kernel function struggling a lot to do the job, without generating a better result.

When using the '*linear*' kernel function, the algorithm is trying to catch the normal pattern by a straight line. But the '*rbf*' is trying to shape a decision boundary as a curve


Select parameters

Select timeserie: A2Benchmark_1

Select algorithm: OneClassSVM

Execution

Algorithm	OneClassSVM
Timeserie Name	A2Benchmark_1
Execution time	4,8777333 Seconds

Parameters

Kernel: rbf	Degree: 3
Gamma: 4	Nu: 0,5

Figure 20: OCSVM - Execution test for the synthetic data set

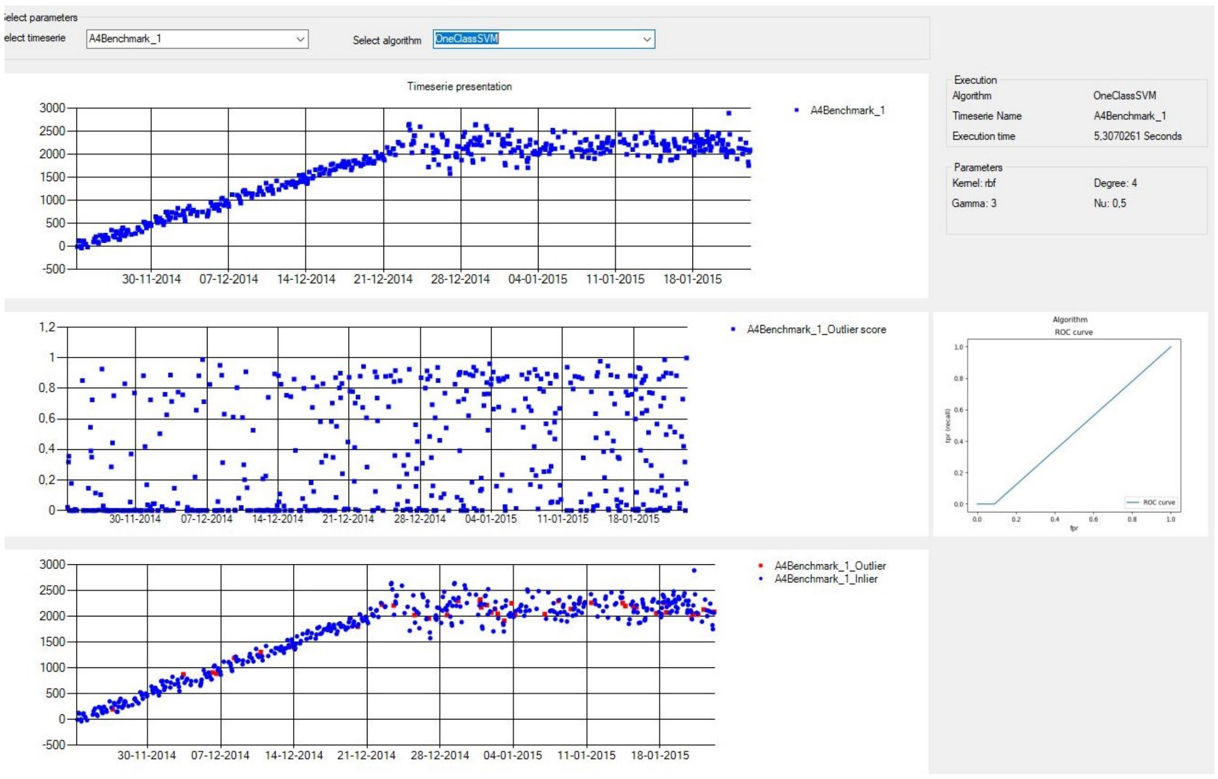
Figure 20, shows one of the test results for the OCSVM prediction on a synthetic data set. The algorithm succeeds predicting the outliers but it also got a few misclassified objects. Which also has been supported by the *ROC* curve and the Outlier factor.

Like for the synthetic *A2Benchmark_1* data set several tests have been performed to the real-world time series *A4Benchmark_1*. Below the table shows the result set. By a mistake I was changing the parameter '*Degree*' for the '*rbf*' and '*linear*' kernel, following the Scikit-learn documentation this parameter is ignored for all other kernel function than the '*poly*'. But still there seems to be a relatively high difference in the *ROC* score, from time to time. I assume that this may be related to the amount of data available for this test. When looking at the test results for the real-world data set in table 3 below, it may be interesting to see that the execution time for both the '*rbf*' and '*linear*' kernel function stays the same. Regarding the best *ROC* score it seems to be slightly better than for the synthetic data set. A possible reason for this may be that the '*rbf*' kernel finds it easier to create a decision boundary when the distribution is more dense.

kernel	**degree**	**gamma**	**nu**	**ROC**	**execution time**
rbf	3	3.0	0.5	0.59	0.1
rbf	2	3.0	0.5	0.49	0.1
rbf	4	3.0	0.5	0.77	0.1
rbf	5	3.0	0.5	0.57	0.1
rbf	4	2.0	0.5	0.61	0.1
rbf	4	4.0	0.5	0.74	0.1
rbf	4	3.0	0.2	0.51	0.1
rbf	4	3.0	0.7	0.16	0.1
linear	3	3.0	0.5	0.32	0.5
linear	2	3.0	0.5	0.38	0.4
linear	1	3.0	0.5	0.42	0.6
linear	1	2.0	0.5	0.01	0.6
linear	1	4.0	0.5	0.46	0.5
linear	1	5.0	0.5	0.39	0.6
linear	1	4.0	0.2	0.48	0.4
linear	1	4.0	0.1	0.57	0.4

Table 3: OCSVM - Execution results for the real-world data set

Figure 21, shows the test result for the OCSVM prediction on a real-world data set. Here it was not possible, to get the same positive result as shown in the test table above. By looking at the picture (Figure 21), we see that if we compare two lowest charts the '*Outlier score*' and the '*Predicted inliers / outliers*', in the beginning of the data set where the data distribution is going up, then the amount of objects in the '*Outlier score*' chart, close to one is less than the rest of the chart. That observation can also be reflected in the number of red dots in the '*Predicted inliers / outliers*' chart for the same area.



Select parameters
Select timeserie: A4Benchmark_1
Select algorithm: OneClassSVM

Execution
Algorithm: OneClassSVM
Timeserie Name: A4Benchmark_1
Execution time: 5,3070261 Seconds

Parameters
Kernel: rbf
Degree: 4
Gamma: 3
Nu: 0,5

Figure 21: OCSVM - Execution test for the real-world data set

7.3 Local Outlier Factor experiments.

The task of this project will be based on the implementation from Scikit-learn. This implementation takes the following parameters:

- *n_neighbors* – Value for K, describes how many objects to be in the local area.
- *algorithm* – Algorithm used to compute the nearest neighbours. (*ball_tree*, *kd_tree*, bruteforce)
- *leaf_size* – Parameter to be used to control the creation of the isolation trees in the *ball_tree* or *kd_tree* algorithm. This parameter can be used to improve the execution time.
- *p* – Describes the distance function to be used for calculating the distance between to objects. (Manhattan – *p* value of 1) or (Euclidean – *p* value of 2)

This Scikit-learn implementation, is not returning an outlier score in the range of 0 to 1. That normally will have been used to describe the outlierness of an object. Instead, it returns a '*sample score*' for each object that has been observed to be in the range of -52 to 1, 2. Doing the implementation process I had been in a discussion to myself how to handle this issue. And my final decision turned out to be that I will print out this '*sample score*' for two reasons:

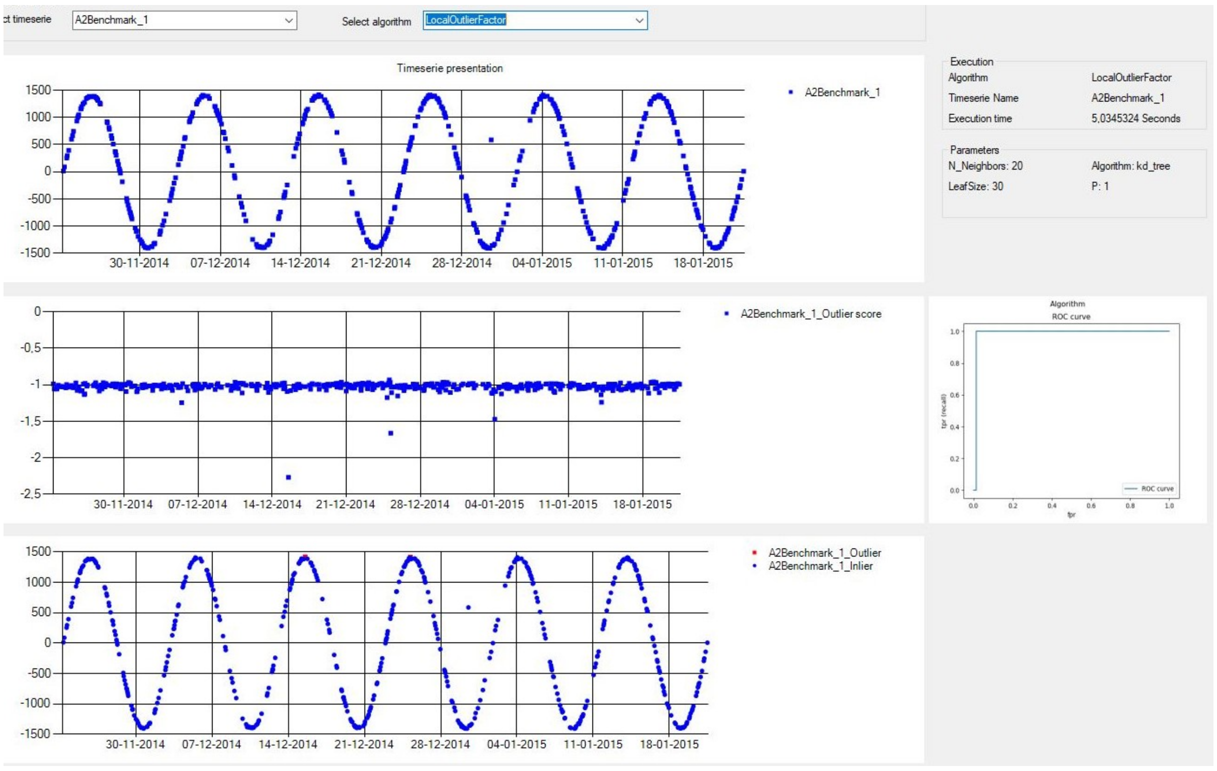
- The algorithms is returning a matrix telling if the particular object has been detected as an inlier or a outlier. So I do not have to use this outlier score to decide if it may be an outlier or not.
- In a real-world situation, a person having domain knowledge, may have to detect outliers from this behaviour. And even if the algorithm was able to return a real outlier score, the it may still require a domain expert to decide when an object is an inlier or an outlier.

To gather some insight into the performance of the LOF algorithm, I was running the algorithm a few times against the synthetic data set *A2Benchmark_1*, and all the time adjusting the parameters. When a parameter with a positive effect to the *ROC* value was found, then it was optimized in that direction. Below table 4 shows, the test results for the first test case. Doing these tests, I was very impressed about the effect of changing the algorithm.

algorithm	**p**	**n_neighbors**	**leaf_size**	**ROC**	**execution time**
ball_tree	1	20	30	0.76	0.36
kd_tree	1	20	30	0.91	0.31
bruteforce	1	20	30	0.56	0.36
kd_tree	2	20	30	0.59	0.32
kd_tree	1	15	30	0.48	0.33
kd_tree	1	25	30	0.88	0.36
kd_tree	1	20	20	0.77	0.34
kd_tree	1	20	40	0.58	0.35

Table 4: LOF - Execution results for the synthetic data set

Unfortunately, I was not able to recreate the fine result from the table above, when creating the screenshot for the report. The only outlier in this test data set, was not captured by the algorithm, but at least the algorithm did not misclassify any other objects. If we look at the '*Original time series*' and '*Outlier score*' charts in '28-12-24'. Then we may see the object that should have been mark as being an outlier in the original dataset. And in the same area for the '*Outlier score*' chart we may find a object at -1,50, and nearly all other objects is located around -1. Unfortunately, the documentation from Scikit-learn did give that much information about this score. But it seems like the smallest values is most likely to be outliers.



Execution
Algorithm LocalOutlierFactor
Timeserie Name A2Benchmark_1
Execution time 5,0345324 Seconds

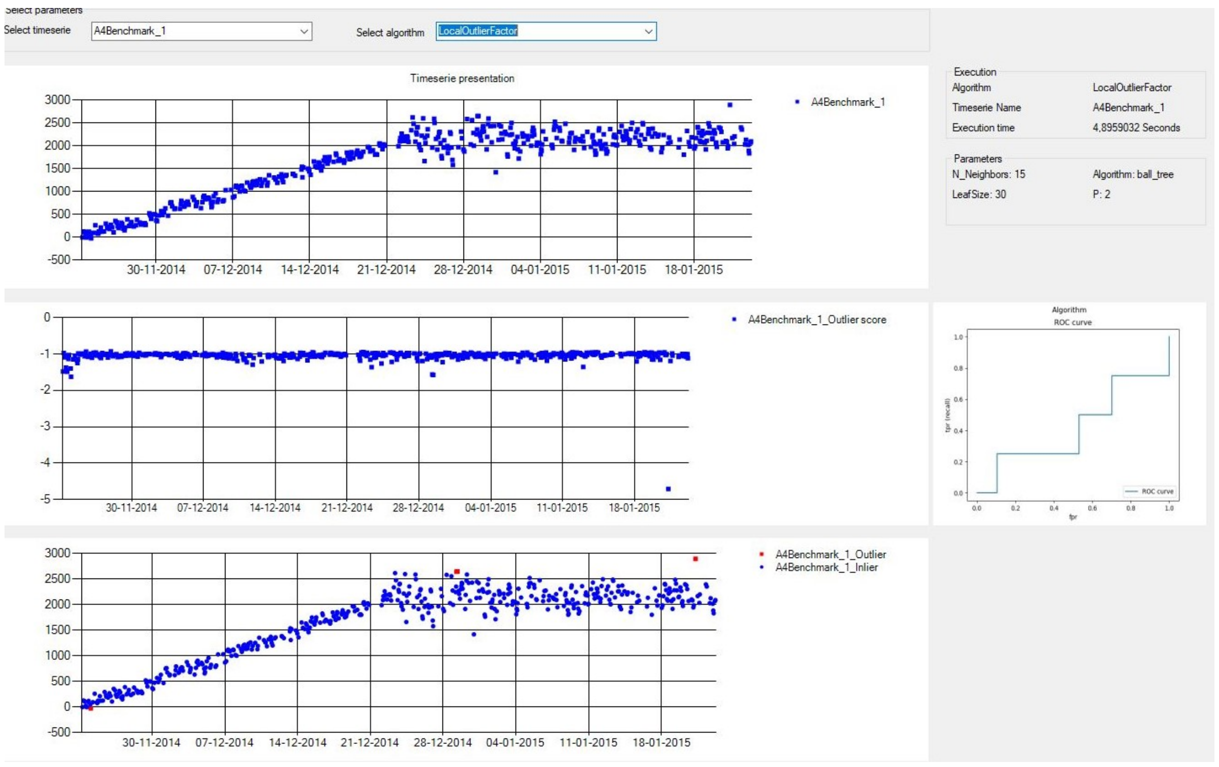
Parameters
N_Neighbors: 20 Algorithm: kd_tree
LeafSize: 30 P: 1

Figure 22: LOF - Execution test for the synthetic data set

Like for synthetic data set *A2Benchmark_1* I was trying to find a good set of matching parameter for the real-world data set *A4Benchmark_1*. Table 5 shows the results, for the test performed on the real-world data set. When looking at the *ROC* curve it seems to have a hard job by separating the objects. But if we are looking at the '*Predicted inliers / outlier*' (figure 23) chart it actually captures one of the outliers, but sadly it did also misclassify a number of objects which may lead to that bare curve.

algorithm	**p**	**n_neighbors**	**leaf_size**	**ROC**	**execution time**
ball_tree	1	20	30	0.38	0.36
kd_tree	1	20	30	0.27	0.42
bruteforce	1	20	30	0.27	0.40
ball_tree	2	20	30	0.45	0.40
ball_tree	2	15	30	0.95	0.43
ball_tree	2	10	30	0.41	0.35
ball_tree	2	15	20	0.53	0.40
ball_tree	2	15	40	0.30	0.35

Table 5: LOF - Execution results for the real-world data set


Select parameters

Select timeserie: A4Benchmark_1

Select algorithm: LocalOutlierFactor

Execution

Algorithm	LocalOutlierFactor
Timeserie Name	A4Benchmark_1
Execution time	4,8959032 Seconds

Parameters

N_Neighbors: 15	Algorithm: ball_tree
LeafSize: 30	P: 2

Figure 23: LOF - Execution test for the real-world data set

7.4 Isolation Forest experiments.

For the IF implementation for the project, I had chosen the Scicit-learn package. This implementation takes the following parameters:

- *n_estimator* – Numbers of trees in the forest.
- *max_samples* - Number of objects to be used for training the individual trees.
- *contamination* – Is used to control the proportion of inliers / outliers.

  The outcome of this implementation:
- *predict* – Returns an array where -1 means outliers and 1 means inliers
- *score_samples* - The lower the output score is, then the point is more likely to be an outlier.

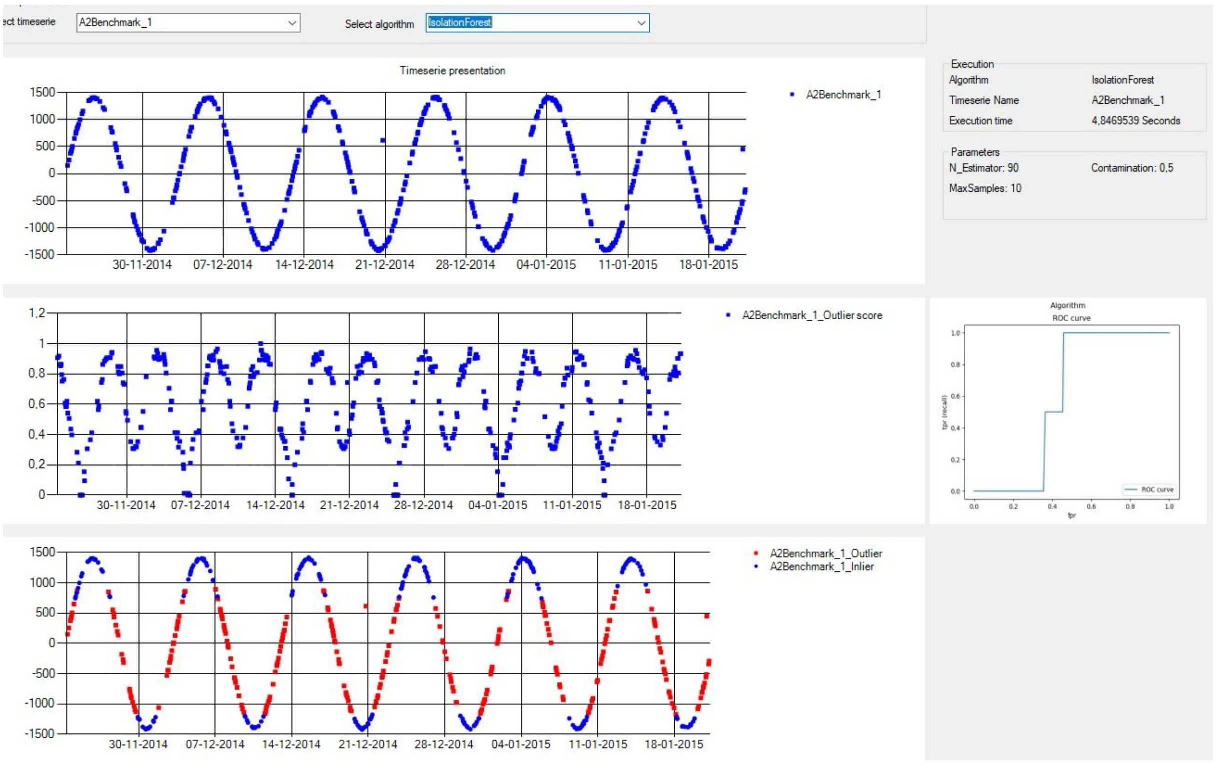
To gather some insight into the performance of the IF algorithm, I was running the algorithm a few times against the synthetic data set *A2Benchmark_1*, and all the time adjusting the parameter. When a parameter with a positive effect to the *ROC* value was found, then it was optimized in that direction. Table 6, shows the result of the different executions of the algorithm. I was expecting to see that changing the number of estimators was having a large impact to the performance. But that was not the case, instead it turns out that changing the contamination values was having the largest impact to the performance, which may make sense because the algorithm may use this as a threshold.

n_estimators	**contamination**	**max_samples**	**ROC**	**execution time**
100	0.1	10	0.55	0.98
110	0.1	10	0.54	0.80
90	0.1	10	0.56	0.72
80	0.1	10	0.54	0.65
90	0.1	20	0.56	0.79
90	0.1	30	0.54	0.73
90	0.01	10	0.50	0.84
90	1.0	10	0.49	0.68
90	0.5	10	0.76	0.68
90	0.3	10	0.15	0.75

Table 6: IF - Execution results for the synthetic data set

By looking at the '*Predicted inliers / outlier*' of the Figure 24, it seems like the algorithm was having a hard job by distinguish between the object in a spars curve pattern. But it is interesting to notify that the red dots are located next to each other at the same position at the curves.

Like for the synthetic data set *A2Benchmark_1* I was trying to find a good matching set of parameters for the real-world data set *A4Benchmark_1*. Table 7, is showing the result. The most interesting things for the second test, is to see that the missclassified object is in a very dense area of the distribution (See chart '*Predicted inliers / outlier*' at Figure 25. I was expecting the outlier at '23-12-2014' to be classified as an outlier object


ect timeserie: A2Benchmark_1

Select algorithm: IsolationForest

Execution

Algorithm	IsolationForest
Timeserie Name	A2Benchmark_1
Execution time	4,8469539 Seconds

Parameters

N_Estimator: 90

Contamination: 0,5

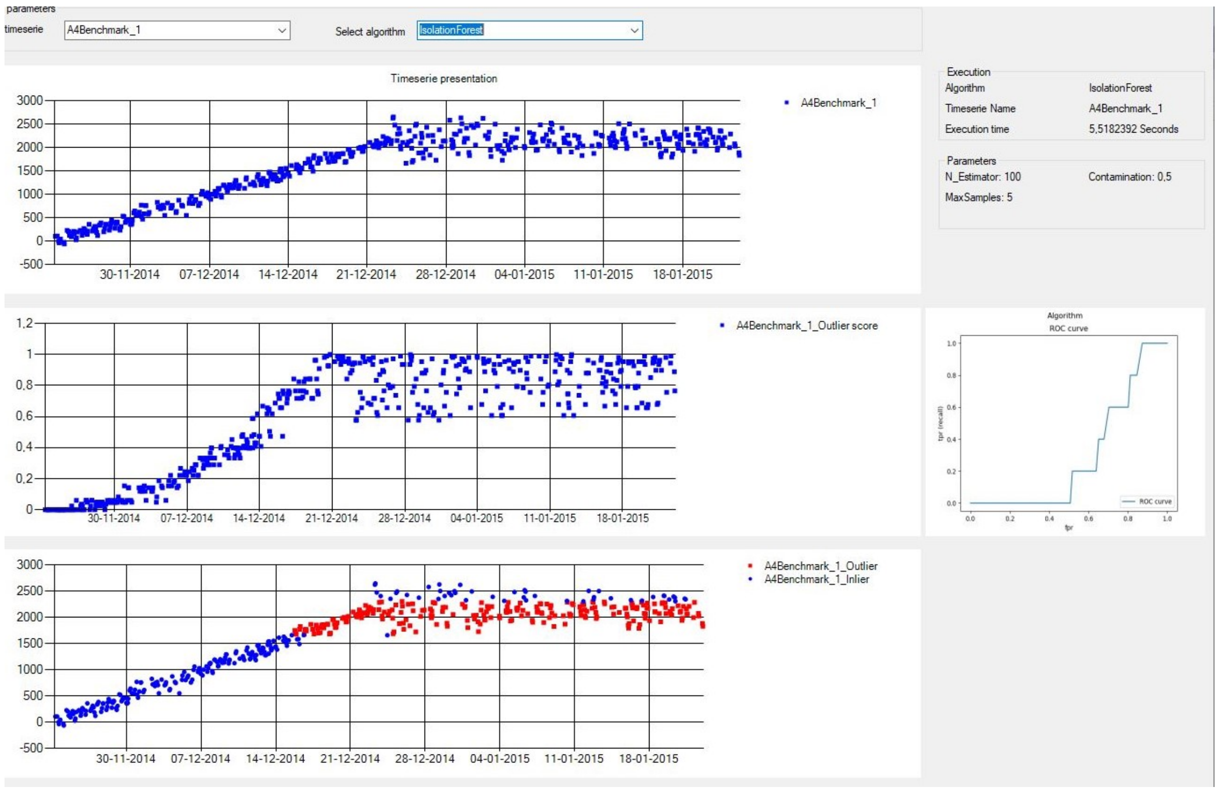
MaxSamples: 10

Figure 24: IF - Execution test for the synthetic data set

because of it position fare away from the rest of the objects, but instead a large amount of objects in the middle of the data set was captured as outliers. That pour result is also reflected in the ROC curve.

n_estimators	**contamination**	**max_samples**	**ROC**	**execution time**
100	0.1	10	0.55	0.81
110	0.1	10	0.54	0.73
90	0.1	10	0.53	0.79
80	0.1	20	0.39	0.72
90	0.1	5	0.56	0.73
90	0.1	3	0.55	0.69
90	0.5	10	0.29	0.75
90	0.3	10	0.40	0.75
90	0.09	10	0.55	0.76
90	0.05	10	0.50	0.74

Table 7: IF - Execution results for the real-world data set


parameters

timeserie: A4Benchmark_1

Select algorithm: IsolationForest

Execution

Algorithm	IsolationForest
Timeserie Name	A4Benchmark_1
Execution time	5,5182392 Seconds

Parameters

N_Estimator: 100   Contamination: 0,5

MaxSamples: 5

Figure 25: IF - Execution test for the real-world data set

7.5 Replicator Multi-layer Perceptrion experiments.

For the implementation of the RMLP algorithm in this project, I have been used the multiple layer perceptron algorithm from Scikit-learn. And for the experiments of testing this algorithm, I made the following parameters available:

- *hidden_layer_sizes* – Defines the size of the hidden layer.
- *activation* – The activation function is used to define the out of each unit.
- *alpha* – Is used to control the border, that is defines the different classes.
- *solver* – The ways that the internal weights is updated.

To gather some more insight into the performance of the RMLP algorithm, I was running the algorithm a few times against the synthetic data set *A2Benchmark_1*, and all the time adjusting the parameter. When a parameter with a positive effect to the *ROC* value was found, then it was optimized in that direction. And as shown in table 8, it seems to be doing relatively fine. When trying to recreate the pattern, at least it return a *ROC* score at 0.94, which is much better than some of the other algorithms.

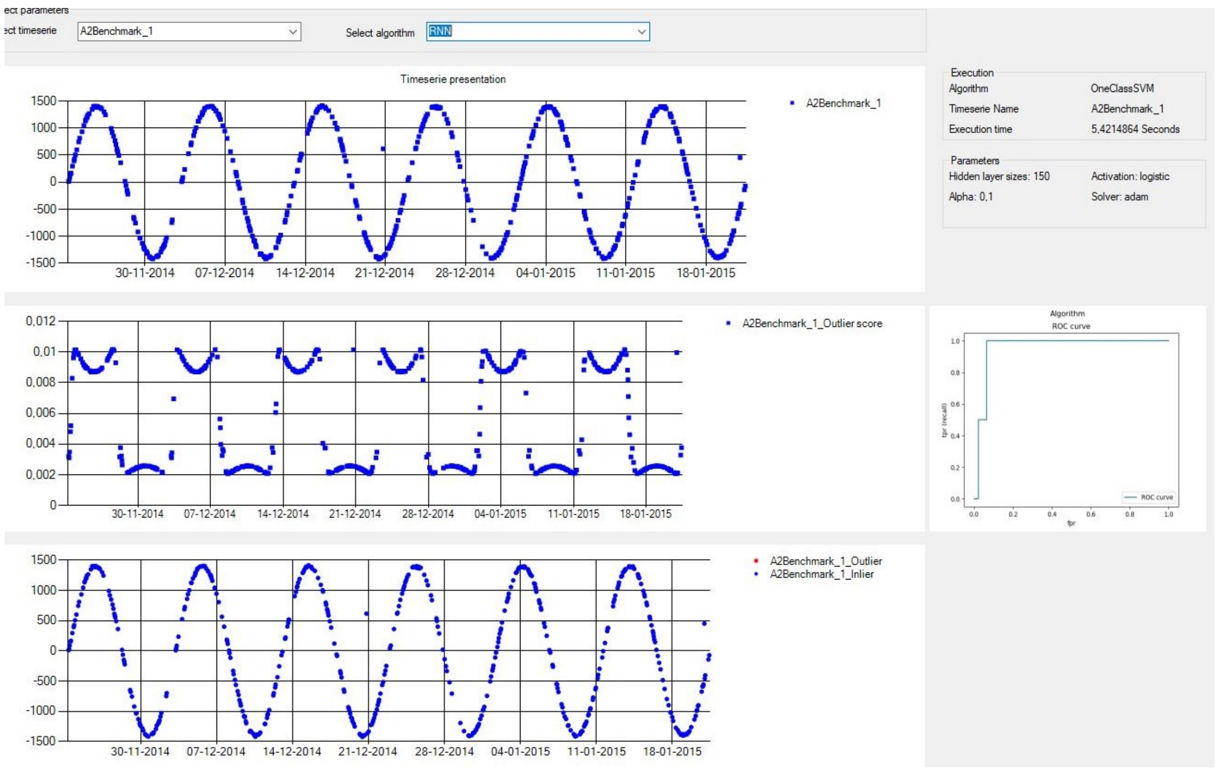
hidden_layer_sizes	**activation**	**alpha**	**solver**	**ROC**	**execution time**
100	*relu*	0.1	*adam*	0.59	0.16
150	*relu*	0.1	*adam*	0.85	0.35
160	*relu*	0.1	*adam*	0.73	0.18
150	*tanh*	0.1	*adam*	0.71	1.14
150	*logistic*	0.1	*adam*	0.94	0.81
160	*logistic*	0.1	*adam*	0.93	0.79
150	*identity*	0.1	*adam*	0.60	1.03
160	*logistic*	0.2	*adam*	0.33	1.01
150	*logistic*	0.1	*sgd*	0.66	0.93
150	*logistic*	0.1	*lbfgs*	0.65	0.21

Table 8: RMLP - Execution results for the synthetic data set

When trying to recreate (see figure 26) the fine result from the table above, it looks like we are getting a similar result, by comparing the *ROC* value and the *ROC* curve. But unfortunately the algorithm was not able to catch the two outlier objects.

Like for the synthetic data set *A2Benchmark_1* I was trying to find a good matching set of parameters for the real-world data set *A4Benchmark_1*, as shown in table 9. Here it might be worth noting the effect of changing the size of the hidden layers. And the fact that in the report 'Outlier Detection Using Replicator Neural Network' [6], they were only testing with three hidden layers.

Figure 27 shows the performance of the RMLP algorithm at the real-world data set *A4Benchmark_1*. The test result from the above test table seems to be reflected at the *ROC* curve. By looking at the '*Outlier score*' in the beginning the algorithm seems to be having a hard job learning the pattern, but it quickly falls to the same level as for the rest of the data set. But again, is disappointing to see that it was not able to catch the real outliers.



ect parameters

ect timeserie: A2Benchmark_1

Select algorithm: RNN

Execution

Algorithm	OneClassSVM
Timeserie Name	A2Benchmark_1
Execution time	5,4214864 Seconds

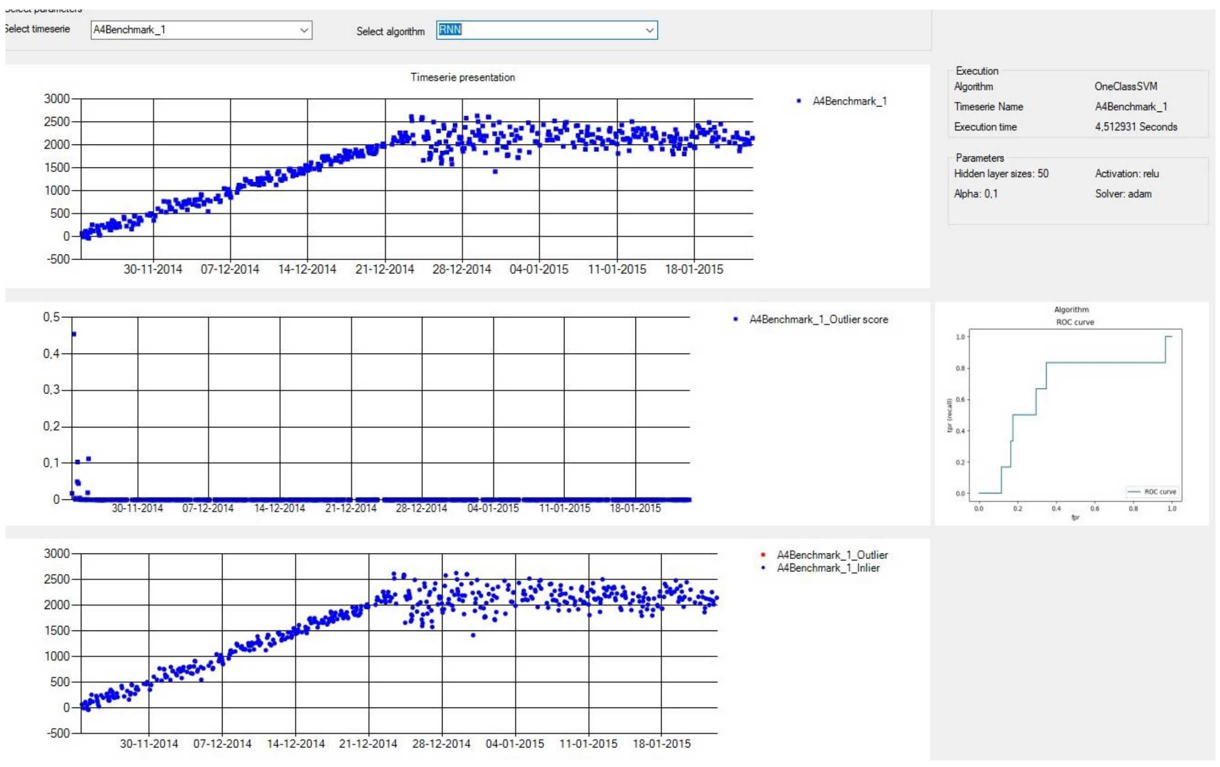
Parameters

Hidden layer sizes: 150	Activation: logistic
Alpha: 0.1	Solver: adam

Figure 26: RMLP - Execution test for the synthetic data set

hidden_layer_sizes	activation	alpha	solver	ROC	execution time
100	*relu*	0.1	*adam*	0.52	0.14
150	*relu*	0.1	*adam*	0.34	0.18
110	*relu*	0.1	*adam*	0.18	0.99
50	*relu*	0.1	*adam*	0.83	0.49
40	*relu*	0.1	*adam*	0.50	0.49
50	*tanh*	0.1	*adam*	0.57	0.42
50	*logistic*	0.1	*adam*	0.58	0.44
50	*identity*	0.1	*adam*	0.43	0.45
50	*relu*	0.2	*adam*	0.36	0.08
50	*relu*	0.1	*sgd*	0.49	0.24
50	*relu*	0.1	*lbfgs*	0.51	0.54

Table 9: RMLP - Execution results for the real-world data set


Select timeserie: A4Benchmark_1

Select algorithm: RNN

Execution

Algorithm	OneClassSVM
Timeserie Name	A4Benchmark_1
Execution time	4,512931 Seconds

Parameters

Hidden layer sizes: 50	Activation: relu
Alpha: 0,1	Solver: adam

Figure 27: RMLP - Execution test for the real-world data set

7.6 DoNut experiments.

The implementation of the DoNut algorithm in this project will be based on the implementation from NetManAIOps/donut [2]. This implementation takes the following set of parameters:

- *batch_size* – Number of training objects, for the model to work through, before the internal parameters of the model is updated.
- *max_epoch* - Controls the number of times the model is looping through, all the objects and tries to improve itself.

The output of this model is an anomaly score, that did not look like the normal outlier score that is goes from 0 to 1. To be able to determine if an object is an outlier the SPOT algorithm was applied. SPOT stands for Streaming Peak Over Threshold and is a part of the Extreme Value Theory. The SPOT algorithm is a modification of the POT algorithm.

To gather some insight into the performance of the DoNut algorithm, I was running the algorithm a few times, and all the time adjusting the parameter. When a parameter with a positive effect to the *ROC* value was found, then it was optimized in that direction. Table 10 shows, the results of these test. To these test it is clear the most important parameter, is the '*max_epoch*' by starting at 10 and returns a *ROC* value a 0.0. But was changed to 40 the returned *ROC* value was 0.80. I find it very surprising that it was the smallest '*batch_size*' value that returns the best result.

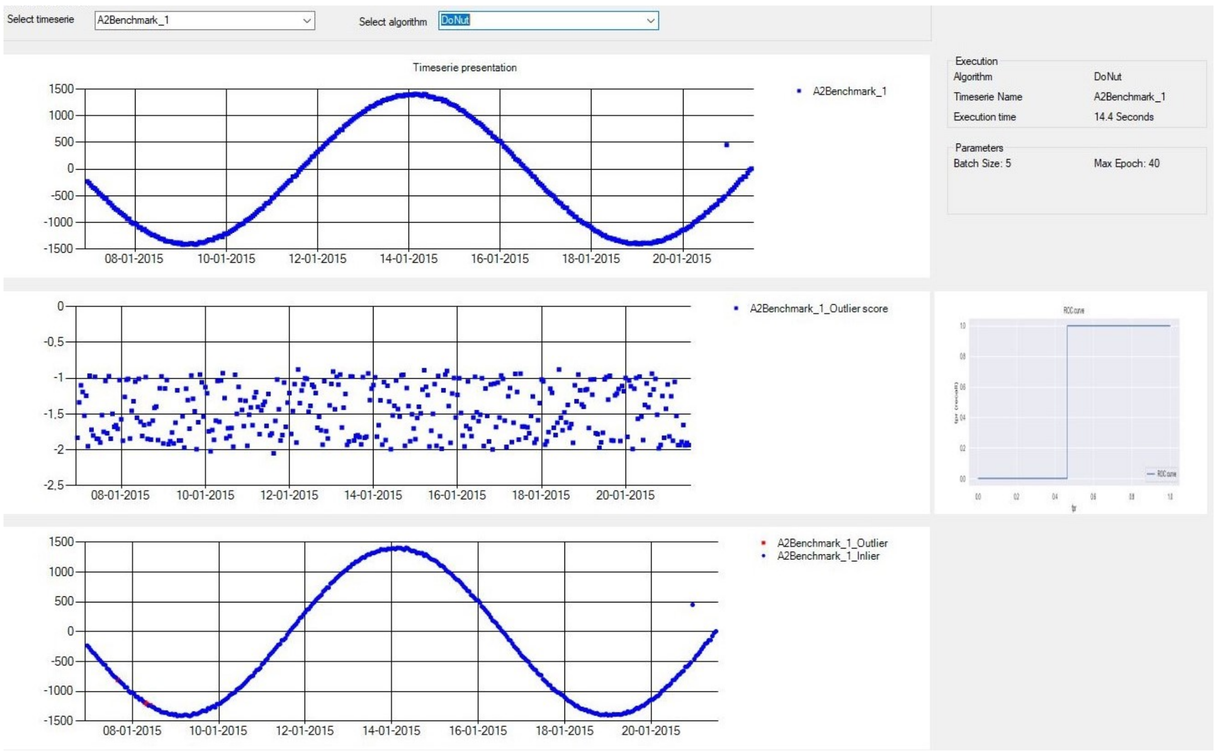
max_epoch	**batch_size**	**ROC**	**execution time**
10	10	0.00	6.8
20	10	0.39	7.7
30	10	0.62	10.5
40	10	0.80	8.4
50	10	0.59	12.5
40	20	0.47	7.7
40	5	0.98	16.1

Table 10: Donut - Execution results for the synthetic data set

Figure 28, shows the DoNut algorithm, executed against the synthetic time series *A2Benchmark_1*. I was hoping to see an algorithm that was able to capture the pattern of the sinus like curve, but when looking at the 'Outlier score' chart it did not seems to be the case.

Table 11, shows the DoNut algorithm, tested against the real-world time series *A4Benchmark_1*. By searching the *ROC* column shows that it seems like DoNut algorithm find it hard to capture normal the pattern of the time series.

Figure 29, shows the performance of the DoNut algorithm executed against the real-world data set. Here the algorithm fails to capture the two outliers but instead it was missclassifying a group of objects in the beginning of the time series.


Select timeserie: A2Benchmark_1

Select algorithm: DoNut

Execution

Algorithm	DoNut
Timeserie Name	A2Benchmark_1
Execution time	14.4 Seconds

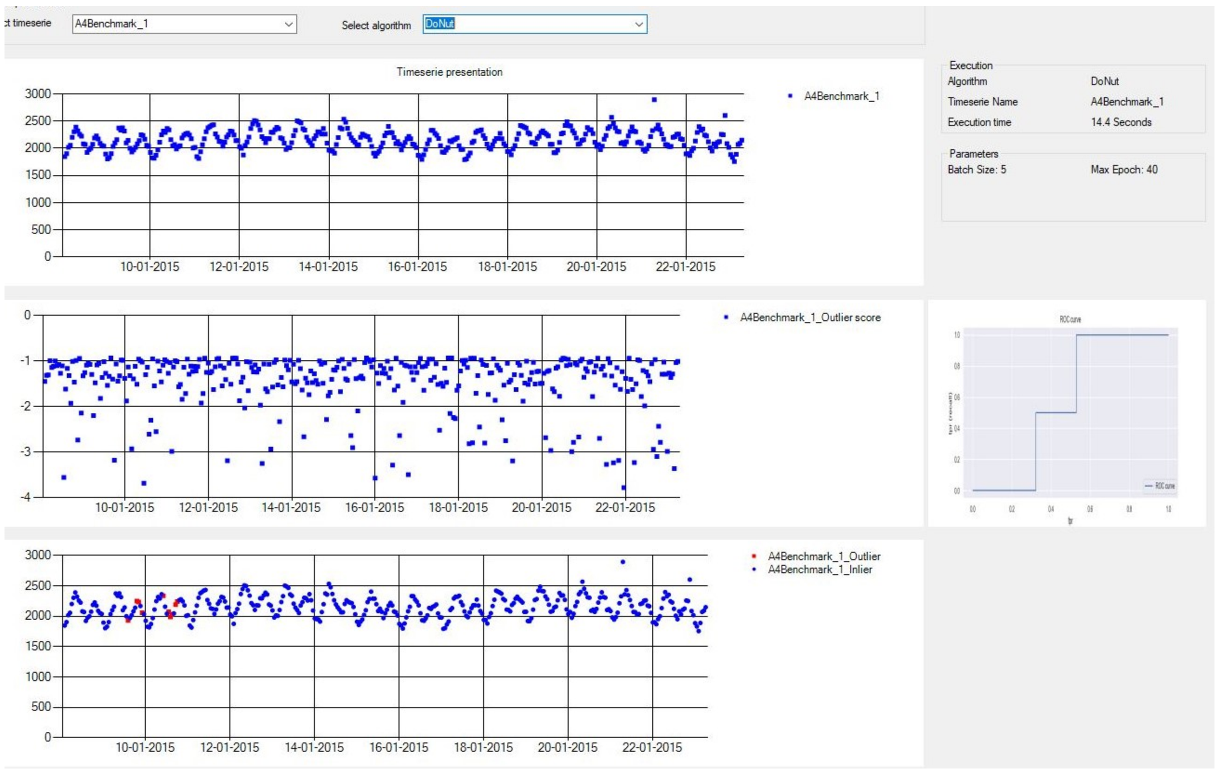
Parameters

Batch Size: 5 Max Epoch: 40

Figure 28: DoNut - Execution test for the synthetic data set

max_epoch	batch_size	ROC	execution time
10	10	0.44	6.2
20	10	0.11	8.8
5	10	0.50	4.8
2	10	0.51	4.5
2	20	0.26	3.8
2	5	0.40	4.5
100	10	0.28	17.40
100	100	0.24	6.4

Table 11: DoNut - Execution results for the real-world data set



ct timeserie A4Benchmark_1

Select algorithm DoNut

Execution

Algorithm	DoNut
Timeserie Name	A4Benchmark_1
Execution time	14.4 Seconds

Parameters

Batch Size: 5 Max Epoch: 40

Figure 29: DoNut - Execution test for the real-world data set

7.7 OmniAnomaly experiments.

The implementation of OmniAnomaly in this project will be based on the implementation from (NetManAIOps, n.d.). This implementation takes the following set of parameters:

- *rnn_cell* – An RNN Cell is performing the same task as neurons in a neural network, but the RNN Cell can take a state and remember information's from the past. For this test I had been working with GRU and LSTM.
- *rnn_number_hidden*
- *windows_length* – The length of each sliding windows.
- *max_epoch* - Controls the number of times the model is looping through, all the objects and tries to improve itself.

To gather some insight into the performance of the OmniAnomaly algorithm, I was running the algorithm a few times against the synthetic data set *A2Benchmark_1*, and all the time adjusting the parameters. When a parameter with a positive effect to the *ROC* value was found, then it was optimized in that direction. Table 12 shows, the results of these test. In the literature the OmniAnomaly is claimed to be able to out-perform the DoNut algorithm. In this case they seems to be performing equally, when it comes to the *ROC* value. But when it comes to the execution time the OmniAnomaly algorithm seems to be much slower than the DoNut algorithm.

rnn_cell	**windows_length**	**rnn_num_hidden**	**max_epoch**	**ROC**	**execution time**
GRU	200	500	20	0.46	530.52
GRU	150	500	20	0.70	522.69
GRU	100	500	20	0.51	551.61
GRU	50	500	20	0.35	515.56
GRU	80	500	20	0.26	544.39
GRU	150	400	20	0.66	496.52
GRU	150	500	10	0.98	528.85

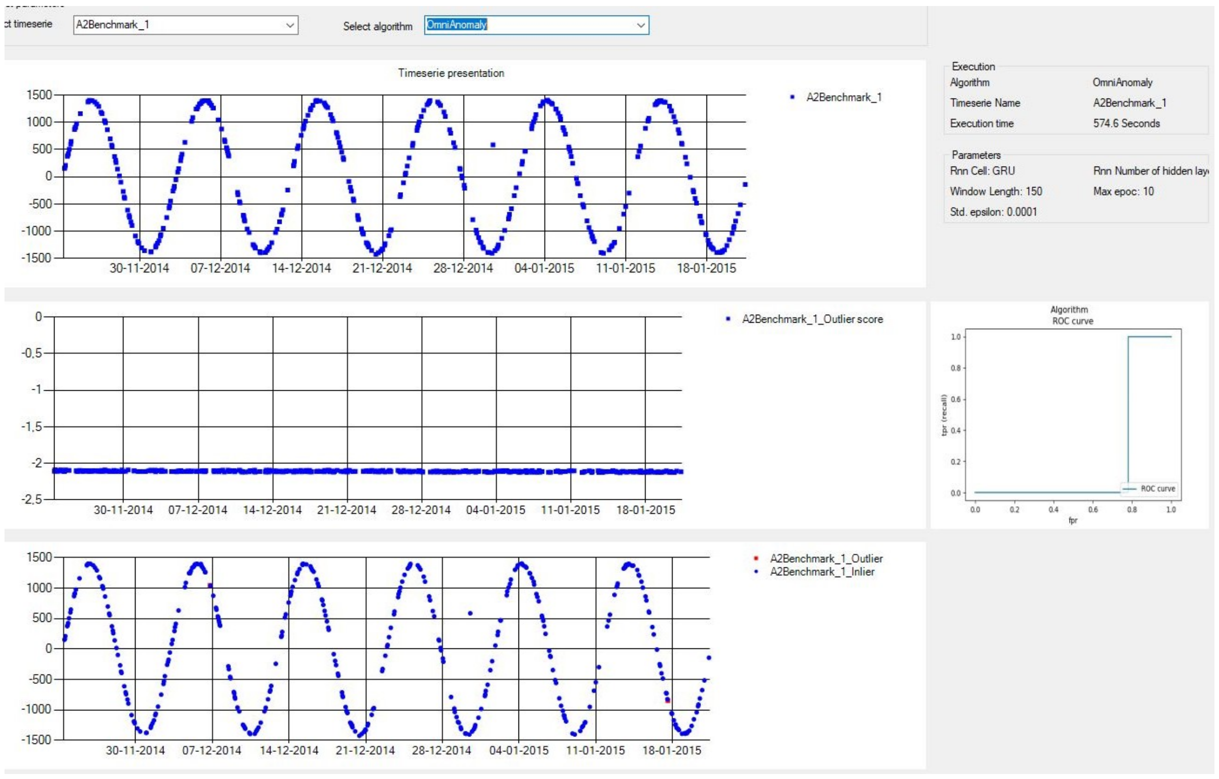
Table 12: OmniAnomaly - Execution results for the synthetic data set

As shown in figure 30, I was trying to recreate the fine result, without success. Here it the algorithm was not able to capture the outliers nor it was missclassifying any objects. The ROC curve shows a ROC value near 0.20 which I did not seem reflected in the chart.

For the real-world *A4Benchmark_1* data set I was doing the same test as shown in table 13.Here the algorithm was not performing as god as in the synthetic tests.

When looking at the table 13 showing the result of the tests performed on the real-world data set, it is clear that the algorithm find it hard to capture the normal pattern.

Figure 31, shows the OmniAnomaly algorithm executed at same data set. And the same badly result seems to be reflected, specifically when we look at the *ROC* curve. Here the bad result from the above earlier tests is reflected, in the *ROC* curve showing a low true positive rate. And the chart shows non of the original outliers was captured by the algorithm.



ct timeserie: A2Benchmark_1
Select algorithm: OmniAnomaly

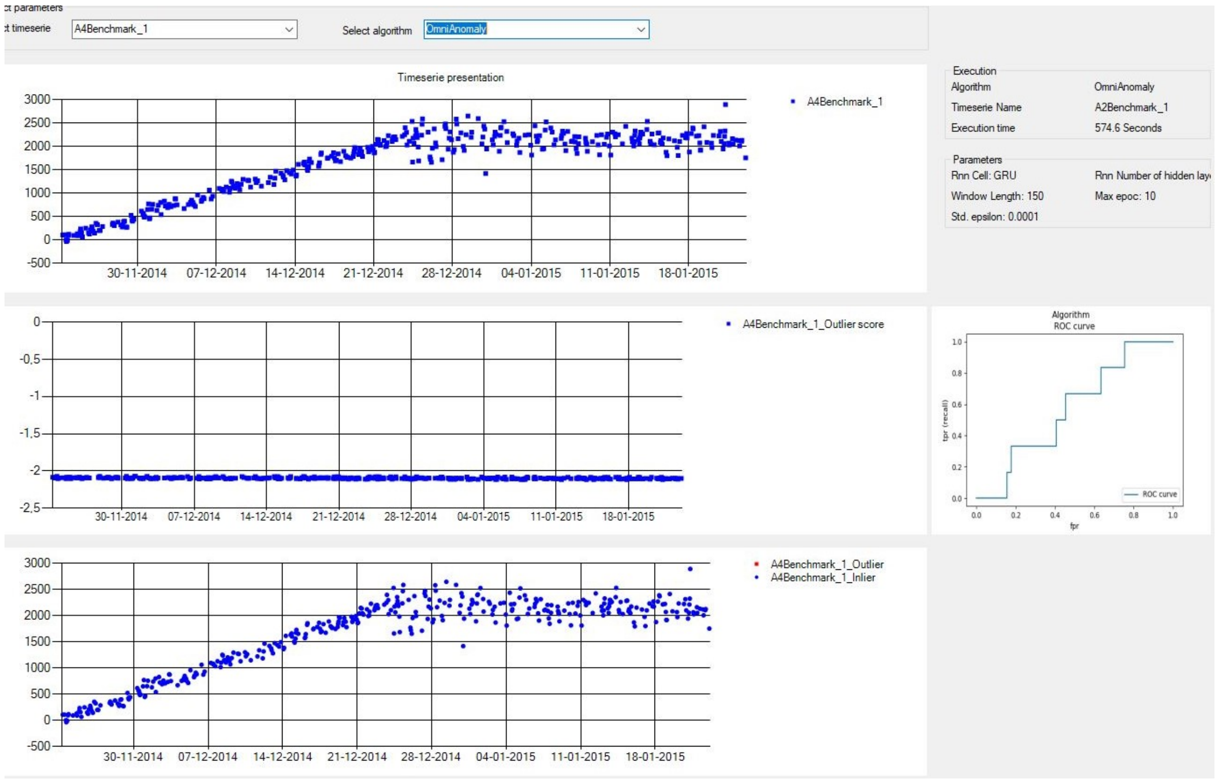
Execution
Algorithm: OmniAnomaly
Timeserie Name: A2Benchmark_1
Execution time: 574.6 Seconds

Parameters
Rnn Cell: GRU
Rnn Number of hidden lay
Window Length: 150
Max epoc: 10
Std. epsilon: 0.0001

Figure 30: OmniAnomaly - Execution test for the synthetic data set

rnn_cell	windows_length	rnn_num_hidden	max_epoch	ROC	execution time
GRU	150	500	20	0.19	574.57
GRU	100	500	20	0.39	580.50
GRU	50	500	20	0.38	581.30
GRU	10	500	20	0.27	548.00
GRU	150	400	20	0.36	531.94
GRU	100	400	20	0.57	537.67
GRU	100	300	20	0.76	572.56
GRU	100	200	20	0.62	553.95
GRU	100	300	10	0.46	600.29
GRU	100	300	40	0.59	457.44
GRU	100	300	30	0.37	552.77

Table 13: OmniAnomaly - Execution results for the real-world data set


ct parameters

ct timeserie: A4Benchmark_1

Select algorithm: OmniAnomaly

Execution

Algorithm	OmniAnomaly
Timeserie Name	A2Benchmark_1
Execution time	574.6 Seconds

Parameters

Rnn Cell: GRU	Rnn Number of hidden lay
Window Length: 150	Max epoc: 10
Std. epsilon: 0.0001	

Figure 31: OmniAnomaly - Execution test for the real-world data set

8 Conclusion

Algorithm	**Synthetic data set.**	**Real-world data set**
OCSVM	The best ROC value was 0.64. It was acting robust, when executing multiple times. But it may have a hart job, creating a decision boundary, around the spread objects.	The best ROC value was 0.77. It seems to be performing better when the objects are closely distributed.
IF	The best ROC value was 0.76	The best ROC value was 0.56. The result was not impressive, but when executing the algorithm multiple times, the result was stable.
LOF	The best ROC value was 0.91. LOF is made for detecting global outliers. And it seems to be the same case here.	The best ROC value was 0.9. LOF is made for detecting global outliers. And it seems to be the same case here. But when executing the algorithm multiple times, the result was differing.
RMLP	The best ROC value was 0.94. Following the ROC curve, it seems like the RMLP algorithm is good for learning the pattern, it was just not reflected in the predicted outliers.	The best ROC value was 0.83. Comparing to the synthetic data set, it seems to have a harder job detecting the normal pattern.
DoNut	The best ROC value was 0.98. And an execution time at 16.1 sec.	The best ROC value was 0.51. The bad performance at the real-world data set bay be related to the way the data is split in to training and test. Because of the sliding window the first 70% of the data is used for training.
OmniAnomaly	The best ROC value was 0.98. It seems to be good at detecting the pattern, but it got a very high execution time.	The best ROC value was 0.76.The bad performance at the real-world data set bay be related to the way the data is split in to training and test. Because of the sliding window the first 70% of the data is used for training. I was the slowest executed test a 572.56 sec.

Table 14: Comparison of outlier detection algorithm.

When starting the project, I thought the algorithm to be used for outlier detection was the same as used for normal classification task. But doing the project I learned that, it was not completely the case. The classification is the task of classifying the objects to belonging to one class or another class, but in outlier analytic all the objects is expected to be belonging to the same class, with some outliers. The algorithm to be used in this project is very different from each other like the time series data set. Therefor to compare the algorithm I had summarized my experiments in the table 14 below.

When looking at the comparisons table above it is important to have in main, that this test has been performed on two relatively small data sets a only 1400 points. And special for the DoNut and OmniAnomaly test at performed against the real-world data set the result may change when they are executed at a larger data set. The problem to this data set is that the data distribution I changing, and when start using the first 70% of the samples for training and the last 30% for testing. Then the algorithm may see the training an testing data sets as two different distributions.

Regarding my future work, in this area. Then I would like to implement the DoNut algorithm at one of our blade-test rigs at Siemens Gamesa Renewable Energy, in order to see if it can be implemented for detection of cracks in the surface.

References

[1] blog.acolyer.org/ unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. `https://blog.acolyer.org`. Accessed: 2021-05-21.

[2] NetManAIOps donut. `https://github.com/NetManAIOps/donut`. Accessed: 2021-05-21.

[3] Scikit-Learn oneclasssvm. `https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html`. Accessed: 2021-05-21.

[4] Aggarwal, C. C. *Outlier Analysis - Second Edition.* Springer, 2017.

[5] Chuah, M. C., and Fu, F. ECG anomaly detection via time series analysis. In *Frontiers of High Performance Computing and Networking ISPA 2007 Workshops, ISPA 2007 International Workshops SSDSN, UPWN, WISH, SGC, ParDMCom, HiPCoMB, and IST-AWSN Niagara Falls, Canada, August 28 - September 1, 2007, Proceedings* (2007), P. Thulasiraman, X. He, T. L. Xu, M. K. Denko, R. K. Thulasiram, and L. T. Yang, Eds., vol. 4743 of *Lecture Notes in Computer Science*, Springer, pp. 123–135.

[6] Hawkins, S., He, H., Williams, G. J., and Baxter, R. A. Outlier detection using replicator neural networks. In *Data Warehousing and Knowledge Discovery, 4th International Conference, DaWaK 2002, Aix-en-Provence, France, September 4-6, 2002, Proceedings* (2002), Y. Kambayashi, W. Winiwarter, and M. Arikawa, Eds., vol. 2454 of *Lecture Notes in Computer Science*, Springer, pp. 170–180.

[7] Liu, D., Zhen, H., Kong, D., Chen, X., Zhang, L., Yuan, M., and Wang, H. Sensors anomaly detection of industrial internet of things based on isolated forest algorithm and data compression. *Sci. Program. 2021* (2021), 6699313:1–6699313:9.

[8] Liu, F. T., Ting, K. M., and Zhou, Z. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy* (2008), IEEE Computer Society, pp. 413–422.

[9] Siffer, A., Fouque, P., Termier, A., and Largouët, C. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017* (2017), ACM, pp. 1067–1075.

[10] Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., and Pei, D. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019* (2019), A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, Eds., ACM, pp. 2828–2837.

[11] Xu, H., Chen, W., Zhao, N., Li, Z., Bu, J., Li, Z., Liu, Y., Zhao, Y., Pei, D., Feng, Y., Chen, J., Wang, Z., and Qiao, H. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. *CoRR abs/1802.03903* (2018).