Machine learning approach to support ticket forecasting from software logs

Matti Haukilintu

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Jyväskylä 01.08.2022

Supervisor

Prof. Arto Visala

Advisor

MSc Petri Pyöriä

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Matti Haukilintu

Title Machine learning approach to support ticket forecasting from software logs

Degree programme Automation and electrical engineering

Major Control, Robotics and Autonomous Systems **Code of major** ELEC3025

Supervisor Prof. Arto Visala

Advisor MSc Petri Pyöriä

Date 01.08.2022 **Number of pages** 63+9 **Language** English

Abstract

Samlink develops robotic process algorithms for its banking customers to perform mechanical tasks and improve business efficiency. Unfortunately, these robots often encounter errors, and bank clerks have to manually complete the tasks they leave behind. This results in a technical support ticket written by the clerks to Samlink's technical support team, and the development team begins to investigate the problem. The logs written by the robotic process algorithms are essential for debugging.

Due to the number and extent of the logs, debugging is very laborious. This thesis explores the possibility of utilizing a cloud-based machine learning environment to find the connection between the support tickets and log events. This will be used to develop a machine learning system that reads production logs, capable of alerting developers of a potential incoming ticket even before bank clerks themselves detect the error.

The data used in the study is first anonymized in a production environment in order to preserve data privacy. The data is then pre-processed into a cleaner format, so that it can be processed by a machine learning algorithm.

In the machine learning phase, anomaly detection is applied to identify possible log events leading to a support ticket. Random delay between the robotic process algorithm log entries and the tickets sent by the clerks is addressed by time frame compression and hybrid machine learning, which uses two algorithms at different stages of the machine learning pipeline.

This study could not prove, that there is a connection between log entries and technical support tickets, that the algorithm is able to detect. The numerous problems encountered affected the components chosen and parameters used. If these problems can be solved, it is possible to find that connection between logs and support tickets. It is also suspected that enhancing the format of the logging will improve the results. Nonetheless, further research is needed.

Keywords Machine learning , Robotic process automation, anomaly detection, log analyzing, Hybrid machine learning

Aalto University
School of Electrical Engineering

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Matti Haukilintu

Työn nimi Sovelluslokien ja vikatikettien yhteyden löytäminen koneoppimista hyödyntäen

Koulutusohjelma Automaatio- ja sähkötekniikka

Pääaine Ohjaus, robotiikka ja autonomiset järjestelmät **Pääaineen koodi** ELEC3025

Työn valvoja Prof. Arto Visala

Työn ohjaaja FM Petri Pyöriä

Päivämäärä 01.08.2022 **Sivumäärä** 63+9 **Kieli** Englanti

Tiivistelmä

Samlink kehittää pankkiasiakkailleen ohjelmistorobotteja suorittamaan mekaanisia tehtäviä ja tehostamaan liiketoimintaa. Valitettavan usein kyseiset robotit kohtaavat virheen, ja pankkivirkailijoiden on suoritettava käsin loppuun niiden jättämät tehtävät. Tästä seuraa Samlinkin tekniselle tuelle virkailijoiden kirjoittama vikatiketti, jonka perusteella kehitystiimi alkaa tutkia ongelmaa. Vianselvityksessä oleellisen tärkeitä ovat ohjelmistorobottien kirjoittamat lokit.

Lokien määrästä ja laajuudest johtuen vianselvitys on hyvin työlästä. Tässä diplomityössä tutkitaan mahdollisuutta hyödyntää pilvipalvelun koneoppimisympäristöä vikatikettien ja lokitapahtumien välisen yhteyden löytämiseksi. Tämän avulla pyritään kehittämään tuotantolokeja lukeva koneoppimisjärjestelmä, joka kykenee varoittamaan kehittäjiä mahdollisesti saapuvasta tiketistä jo ennen kuin pankkivirkailijat itse havaitsevat vian.

Tutkimuksessa käytetty data anonymisoidaan ensin tuotantoympäristössä tietosuojan säilyttämiseksi. Tämän jälkeen dataa esikäsitellään siistimpään muotoon, jotta se olisi koneoppimisalgoritmin käsiteltävissä.

Koneoppimisvaiheessa pyritään soveltamaan anomaliatunnistusta mahdollisten tikettiin johtavien lokitapahtumien tunnistamiseksi. Satunnainen viive ohjelmistorobotin lokimerkinnän ja virkailijan lähettämän tiketin välillä yritetään ratkaista aikaikkunakompressiolla ja hybridimuotoisella koneoppimisketjulla, jossa hyödynnetään kahta algoritmia ketjun eri vaiheissa.

Tämän tutkimuksen perusteella ei voitu osoittaa, että lokimerkintöjen ja vikatikettien välillä on yhteys, jonka algoritmi kykenee havaitsemaan. Lukuisat kohdatut ongelmat vaikuttivat valittuihin komponentteihin ja käytettyihin parametreihin. Mikäli nämä ongelmat voidaan ratkaista, on tuo yhteys lokien ja tikettien mahdollista löytää. Myös lokituksen muotoilun kohentamisen epäillään parantavan tuloksia. Lisätutkimuksille on joka tapauksessa tarvetta.

Avainsanat Koneoppiminen, ohjelmistorobotiikka, anomalioiden tunnistaminen, lokien analysointi

Preface

Terve,
ja kiitos kaloista.

Jyväskylä, July 28, 2022

Matti Haukilintu

Contents

Symbols and abbreviations

Symbols

ϵ	Error term

Operators

$A^{\mathbf{T}}$	Transpose of vector **A**
$\mathbf{f}(X_n, Y_n)$	Undefined algebraic equation between operands X_n, Y_n, for example $X_1^{Y_1} - X_2^{Y_2} - \ldots - X_n^{Y_n}$

Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
HML	Hybrid Machine Learning
RPA	Robotic Process Automation
SQL	Structured Query Language
JSON	JavaScript Object Notation
CSV	Comma-Separated Values
GDPR	General Data Protection Regulation
ADA	Anomaly Detection Algorithm
IoT	Internet of Things
UI	User Interface

1 Introduction

Artificial intelligence (AI) and machine learning (ML) have found their way into more and more fields of business. In the banking business they are already used in fraud detection, risk management and service recommendations.[1] Even though these modern technologies utilizing big data are widely used abroad, AI and ML are not yet that commonly in the Finnish banking field. Instead, many self-acting solutions are used to streamline manual labor which could be called intelligent, but are these solutions are merely highly automated processes and thus cannot be included in the category of AI. One of these technologies used in Finnish banking systems is Robotic Process Automation (RPA).

RPA operates “on the user interface of other computer systems in the way a human would do”,[2] but is strictly bounded by predefined operations, which makes it prone to unforeseen situations such as faulty input. RPA, like generally all other software, produces log to “register the automatically produced and time-stamped documentation of events, behaviors, and conditions relevant to a particular system”[3]. Logs don’t follow any standards or form guidelines which tends to make log analysis and log based problem-solving troublesome. This is also the case with RPAs developed by Oy Samlink Ab.

Oy Samlink Ab (Samlink from now on) was founded in 1994 and is now owned by Kyndryl. From the early years, while going by the name of Samcom, the company was owned by several Finnish banks for which it developed all sorts of IT solutions for them. Nowadays, Samlink offers a wide variety of banking solutions from basic banking system to end user targeted software such as Codeapp mobile application.

In addition to banking, Samlink develops multiple other IT solutions to an extensive range of customers, for example entertainment platform solutions for DNA. Even though Samlink can be considered a modern technology company, the most modern AI technologies have not yet been adopted as a part of the variety of tools used in development. However, RPA has been actively used in some banking solutions to reduce the amount of manual labor required from banking clerks.

Along with continuous development and product maintenance services, Samlink also offers a technical help desk regarding the software solutions it produces. As no IT solution comes without bugs or misbehaviour, Samlink’s service desk has to use considerable amount of labor to resolve the possible issue behind the technical support request tickets received. In many cases, the problem-solving starts by reading the log and analyzing the data written by the processes in question.

In this study, we aim to find if it is possible to utilize machine learning methods in analyzing logs created by Samlink RPA’s. Ultimately, we intend to train ML which is able to predict the arrival of a technical support ticket thus giving a warning for developers about possible issues in the production.

1.1 Background and motivation

In the field of information technology, logging is one of the most important methods in problem-solving, be it software or operating system related.[3] Typically, at least in Samlink's processes, logging is more verbose than it needs to be. This is usually because when the problem occurs, it is easier to already have the verbose logs available than setting logging to more verbose mode and trying to replicate the issue. Too verbose logging, however, leads into two problematic issues for developers.

First of, the size of log is huge and finding the critical parts related to the problem in hand takes more time. Of course, with stricter logging, pinpointing the faulty event from within the logs would be faster. However, solving the problem with only critical error messages could be more time-consuming if crucial context is missing.

Secondly, a well-designed software is able to retry the process after first failure, but logging is done in real time regardless of the process result. This means, that each process failure is logged even though said process might eventually succeed on later attempt. Thus, logs may include dozens of rows of information about a problem, which is not critical information after all. These issues make log analyzing considerably laborious.

Production logs are usually not viewed if everything is presumably working as intended. Technical support tickets are both the last and most visible indicators of an issue in the system. When a technical support ticket is received from banking clerks it means that something is wrong in a very visible way. Roughly speaking, there are two types of technical tickets that are because of a clear misbehavior of the RPA system (not, for example, due to a user error). First are the tickets that uncover an unknown bug in the system which can be either fixed or instructed to user how to avoid. Second type of tickets are somewhat pre-known issues that occur from time to time and are either fixed with updating parts of the system or by rebooting the process.

Typically, in software systems, if an issue is known and can be fixed by rebooting something, developers can create log monitors that search for certain keywords and raise an alert if they are encountered. Developers can either run a reboot manually after a log alert has been received or set up an automated script to do it immediately when such keyword has been found. However, when it comes to RPA's and technical tickets, it is hard to say what the issue in question is by reading the RPA logs word by word without context. New kind of issues can be more frequent than already confronted ones, and a clear keyword linked to a certain problem may not exist without considerable amount of false positive matches.

Machine learning algorithms are widely used to find patterns from massive amount of data making ML an ideal tool for log analysis. Patterns, however, need a connection to a visible issue to be useful. If the RPA system has encountered an error but is able to retry successfully, then no issue that requires immediate concern has actually happened. Hence, RPA log analyzing with ML can find meaningful patterns only if they relate to actual technical tickets.

If Samlink support has received a help request the issue behind the request is not fresh anymore. In the event of RPA job failing, it takes some time for the clerk to notice the issue, write a help request to first support level, which then redirects the ticket to the corresponding team. Furthermore, if the issue is noticed on friday, it takes a few more days to be handled by RPA developers due to the weekend. This leads to a noticeable delay in processes that were supposed to be dealt by RPA but which now have to be manually taken care of by clerks.

If a correlation between logs and received tickets exists and an ML algorithm is able to find it, it could be possible to create an ML-based log analyzer that can alert developers about an ongoing issue before banking clerks encounter it. With automated scripts set up to receive such alerts, some issues could even be fixed in the production automatically without human interaction. This would significantly reduce the time and labor needed from developers and bank clerks alike.

1.2 Research objectives

This research aims to pave the way for machine learning application developers inside Samlink. Multiple obstacles need to be tackled as most of the phases in this study have not yet been encountered inside the company.

First, it is crucial to construct some basic rules considering the format of log data to make it usable for ML algorithms. Log data formatting is one of the key elements in automatic log analysis applications as it is not for just machines but also for people to read.

As today more and more concern is set on anonymization, the data used for machine learning must be sanitized. As a consequence, one major objective is to create a clean dataset that is safe to use in a cloud environment without raising concern around security and privacy issues. In addition to this, data must also be clean enough so that ML algorithms are able to process it.

As mentioned, Samlink has not yet developed ML applications. In order to facilitate deployment of applications for future ML application developers, this study aims to document the process of ML deployment thoroughly enough to create a simple guide to follow in possible future ML projects in Samlink.

Finally, the main questions this research aims to answer are: *is there such a correlation between RPA run logs and technical support tickets that an ML algorithm is able to find, and can this correlation be used to forecast a ticket arrival*?

1.3 Scope

In order to limit the study to feasible length and content, it is necessary to define the scope for the thesis. Before diving in to the scope of the research objectives, we must first make one assumption regarding the data from which ML is going to find some meaning. In order to find a connection between log data and support tickets, we make a hypothesis that errors leading to tickets are visible to or parseable by ML algorithm. In addition, as we are going to utilize an anomaly detection algorithm in log analysis, we must also assume that these errors on the log are, in fact, anomalies. The results will be compared against these assumptions to the test this hypothesis.

Data anonymization

Anonymization in the context of this thesis refers to a data sanitization process purposed to edit the data into a more secure form in the point of view of privacy. In this study we aim to create a dataset that can be used in ML training. In this respect, anonymization is not the main focus of the study but only treated as a sub-phase of the data preprocessing. Nevertheless, anonymization is the most important phase of data preprocessing from the privacy perspective.

Keeping this in mind, anonymization is covered rather superficially, only enough to explain the reasons behind actions taken during the anonymization process.

Azure setup

The ML training and result scoring is done in Azure ML environment. Azure is used for ML processes because Samlink already had licenses for Azure Cloud that is used for RPA process control. Integrating existing Azure resources with ML pipelines and endpoints constructed during this study was seen as a big advantage. Thus, no other ML cloud provider was considered. However, other Azure competitors are mentioned to the extent that their existence is recognised.

As one of the objectives is to create an initial guideline for ML process commissioning, the Azure setup phase is documented in such detail reflecting the importance of this information for future developers starting Azure ML projects in Samlink.

Data requirements

Data purity in the perspective of machine learning algorithms creates challenges at the beginning of ML training. If data is not consistent, has lots of missing values or is formed in an unanticipated way, it requires considerable amount of preprocessing which slows the training process and causes errors in pipeline runs.

In order to create a baseline for Samlink ML projects, this study aims to give basic criteria for what is required from the data, so it is easily analyzable by ML algorithms.

Machine learning methods

Several different machine learning algorithms have been created for different applications. For example, to make an algorithm that can predict the price of a listed apartment[4] we could use linear regression, and in order create an algorithm that detects possible cyber threats from network traffic[5], a two-class support vector machine could be utilized. These two methods are very different in usage and have their pros and cons for different applications.

As different methods can be used in creative ways in very different applications depending on how the data is presented and how the ML problem is formed, this study focuses on just a few easily approachable training methods that were seen suitable to answer the study objectives.

When it comes to anomaly detection algorithms (ADA), only principal component analysis (PCA) is considered because PCA-based Anomaly Detection component is the only one out of the two existing anomaly detection algorithms that is usable in Azure ML Studio. As only Azure ML Studio is used during this study, no other anomaly detection algorithms are debated. The other ADA-component, One-Class

Support Vector Machine, is discussed briefly to explain its unsuitableness for the current case.

1.4 Structure

In the **Introduction** section we explained the research motivation, main objectives and scope of the study. The next section, **Background**, clarifies the general machine learning concepts and RPA terms relevant to this thesis. We also discuss the mathematical theories behind the most significant algorithms utilized in this study, and the main methods that can be used to secure data privacy.

The third section, **Research material and methods**, explains in detail the data format and contents as well as the steps used to sanitize and preformat the data for ML algorithm training. In addition, resources needed to set up the ML designing and training environment are discussed. This section is followed by **Machine learning pipeline structure**, which describes the ML pipelines set up during the study and specifies their contents in detail.

The **Results** section compares how different algorithms performed and how well the research questions could be answered. Before that we discuss the memory issue that affected greatly our component selections and data amounts. At the end of the section we evaluate the results and discuss what could have been done better.

Finally, in the section **Summary**, we summarize the research goals and outcomes.

2 Background

Machine learning, or ML, is a subcategory of the AI field and data science. Typically, ML refers to a set of technologies used to "build computers that improve automatically through experience". [6] This is generally considered a machine way to simulate the human learning process. ML usage has become more common and is nowadays widely used in many fields, not just in general information technology and computer science. This is because data can be gathered from anywhere, and where there is data to be processed, ML can be used to process it. Computer algorithms are able to find statistical correlation and patterns from places overlooked by the human mind, or in cases where the amount of data is just too much for people to process. This is why ML has proved its power in various empirical science fields, such as biology, cosmology or social science. [6]

In this section, key concepts of ML are explained briefly and several ML features that are most relevant to this study are explored. We also briefly discuss about data sensitivity and how it was addressed during this study.

2.1 Machine learning algorithms and training

An algorithm is a finite sequence of (typically) mathematical operations that are used to solve a specific problem, generally by repetition of certain steps until the problem resolves. [7] Algorithms are the main component inside machine learning. By iterating through all the data points, an algorithm is able to, for example, find repeating patterns, mathematical or logical connections, or unusual anomalies that would appear seemingly normal for the human eye.

Algorithms operate on a set of rules and parameters. In order to utilize an algorithm to solve a problem, the algorithm is first trained by tuning these parameters to fit the current case. Usually, ML algorithms can be trained in three ways: supervised, unsupervised, and reinforcement learning. [6] Even more training methods exist that usually combine those mentioned. [8, 9] For the sake of simplicity, we focus only on the three main methods.

In **supervised learning**, the algorithm is given data with ready answers on how the data needs to be interpreted. Algorithm then tries to figure out the rules behind how the given data and the correct answers are related. [8] In **unsupervised learning**, on the other hand, the algorithm does not get model data from which to train itself, but instead it tries to find clusters or groups inside the data that are linked together more closely than to other data points. [4] **Reinforcement learning** refers to a method where a computer program is given a goal and provided feedback as a reward. This reward is what the program aims to maximize by adjusting the parameters it has been given. [8]

In ML, there are multiple algorithms to solve different problems and no jack-of-all-trades algorithm exists. Each algorithm is suitable for a certain type of problem. To simplify, algorithms are usually divided into three or four categories based on the problem type. [10]

Regression algorithms predict values and are typically used with supervised learning. A usual example of a regression problem is house price prediction using typical house features such as building year, location, number of rooms etc. . The algorithm then assigns each feature a weight value which determine the final price of the house. [10]

Classification algorithms predict categories and are also used most commonly with supervised learning. Depending on the algorithm, they can predict between two or more categories. Examples of classification problems could be spam mail identification with two class classification, or flower species recognition from images with multiclass classification. [10]

Clustering algorithms use unsupervised learning to find structures inside data. This is done, for instance, by first providing the amount of clusters to search to the algorithm, which then calculates a center point for each cluster so that they are as far away from each other as possible, while the data points surrounding each center are as close to each other as possible. [9] This could be used, for example, to find meaningful customer segments from transaction data in order to improve targeted advertising. [11]

Dimension reduction algorithms are a separate type of algorithms used with unsupervised learning, but they are usually combined with other algorithms to solve the main problem. With dimension reduction, main algorithm calculations are streamlined by first reducing the amount of feature dimensions. [12]

These four ML problem types and most known algorithms of each type are presented in the figure 1.

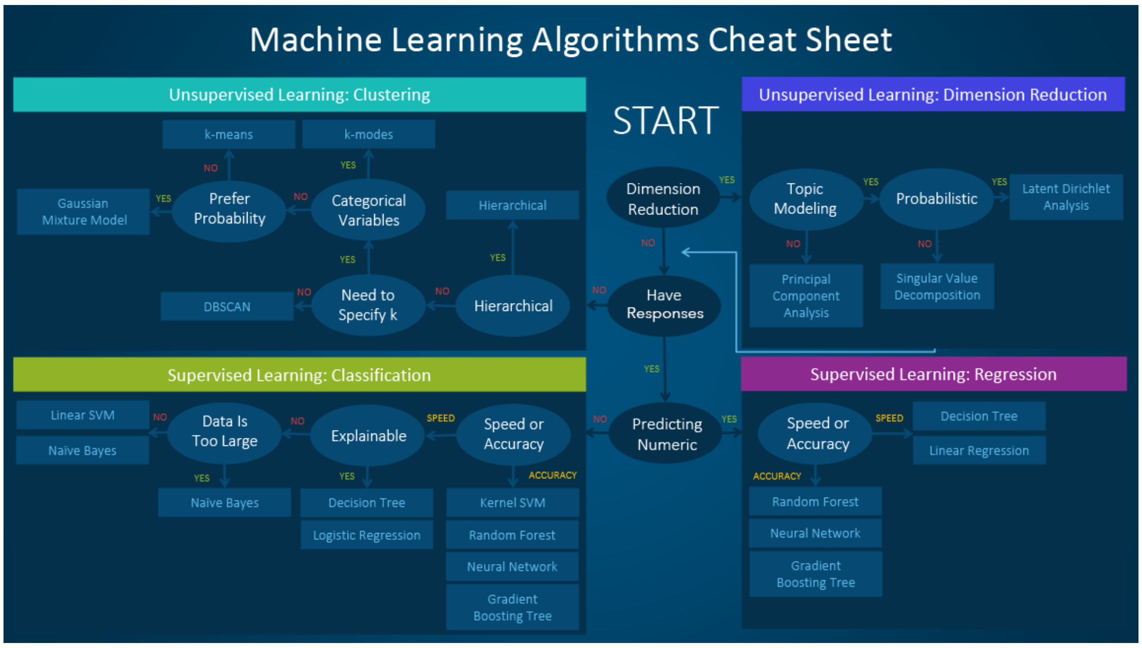


Figure 1: Machine learning cheatsheet for algorithm choosing[12]

This study focuses on anomaly detection, which, simply put, is a clustering problem where anomalies are rare incidents outside common clusters. However, in this study we utilize a PCA-based anomaly detection algorithm, where PCA refers to Principal Component Analysis, and which is a dimension reduction algorithm. [12] PCA is discussed in more detail later in this section. In addition, we aim to find a connection between anomalies and incident tickets by their amount in a timeframe, which makes the topic in the end a regression problem.

Typically, the data used in algorithm training is divided in two parts. One part is used for the training process, and the other is used to validate the results of the training. These data parts must not overlap, but the algorithm is given data to validate that it has not seen before. [13] For example, in supervised learning the key values the algorithm is trained to find are hidden in the validation data. The resulting values produced by the algorithm are compared to the hidden values and the difference between the estimate and the real value can be used to determine how well the current trained algorithm compares to others. However, in this study, we are going to break that rule of non-overlapping training and validation data. The reason for this is explained further in section 4.2.

2.2 Cloud ML platforms

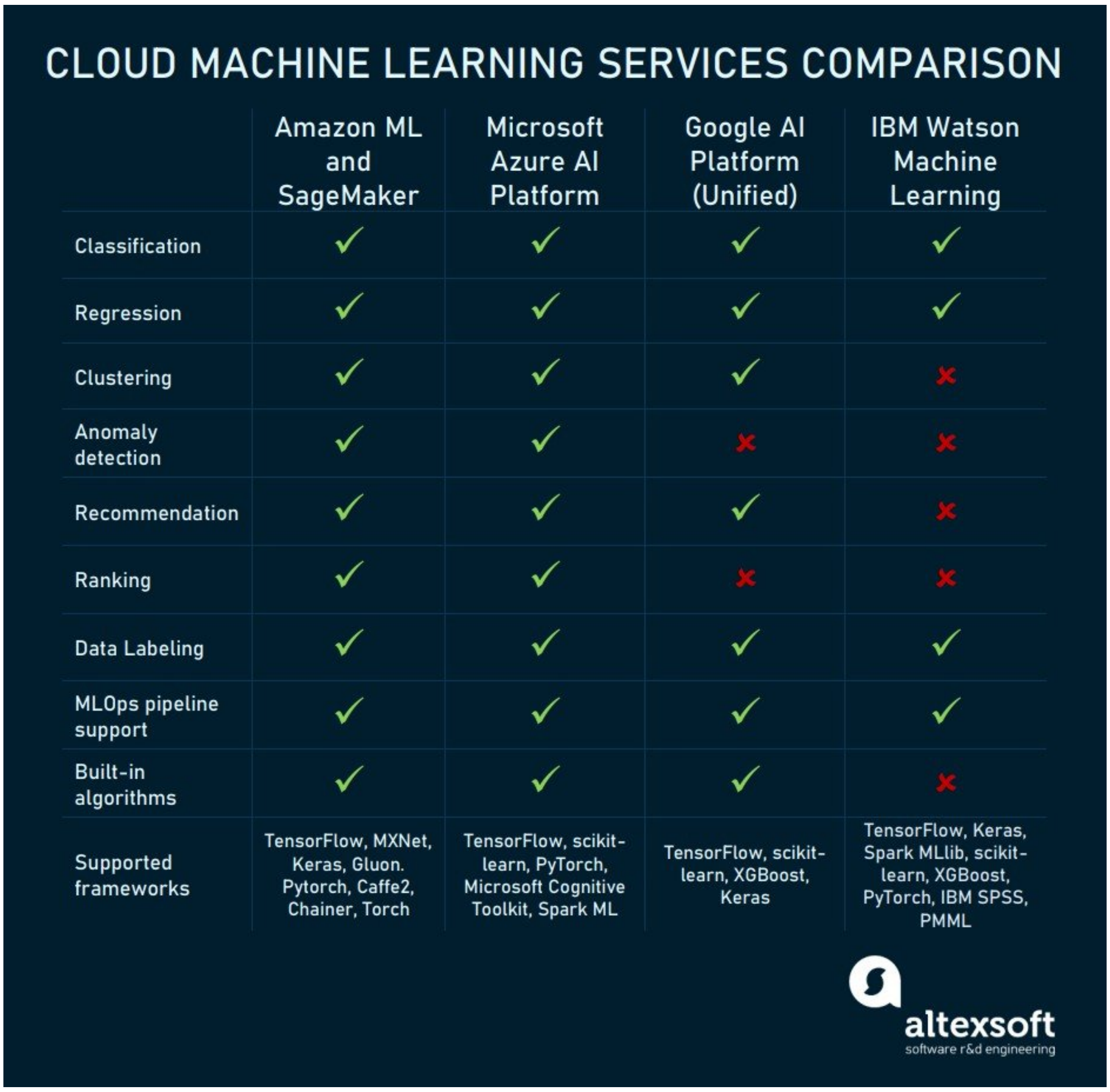
Machine learning algorithms are not light to operate. ML is at its best with big data where the large amount of data points makes it easier for algorithms to find repeating patterns more reliably. [14] Data amount, however, requires huge resources in terms of memory and computing power. Especially with online applications where real time analysis of new input data is required with small latency, cloud computing can make a big difference in terms of processing speed.

The online market offers several solutions for ML computing in cloud. Most notable service providers for MLaaS (Machine Learning as a Service) are Google, Amazon, IBM, and Microsoft. Differences of each service provider are listed in table 2.

Amazon's new SageMaker service has replaced the old Amazon Machine Learning service, and is very much like Azure Machine Learning service produced by Microsoft. Compared to SageMaker and Azure, the Google AI Platform is missing anomaly detection and ranking abilities. IBM Watson has even less features, as demonstrated in table 2. [15]

Azure, however, has one major advantage compared to SageMaker and other competitors, which is the UI environment of ML Studio. Most of the MLaaS providers' solutions have some sort of no-code to low-code design features which makes pipeline designing easy. Azure ML Studio lets the developer design and deploy full ML pipelines with drag-and-drop user interface. [15, 16]

Each component in the pipeline designer can be tuned to certain extent. ML Studio has a predefined set of ready algorithms to use. An example of Azure ML Studio interface is shown in figure 3. Data to the ML Studio environment can be imported from local storage, but also from various other Azure services such as storage accounts with table and blob data. Trained ML pipeline can be published as a cloud endpoint and inserted into a wider operation chain combining it with other Azure services, like data storages and cloud computing resources. This allows the designer to use ML computing capabilities with existing production environments



CLOUD MACHINE LEARNING SERVICES COMPARISON

	Amazon ML and SageMaker	Microsoft Azure AI Platform	Google AI Platform (Unified)	IBM Watson Machine Learning
Classification	✓	✓	✓	✓
Regression	✓	✓	✓	✓
Clustering	✓	✓	✓	✗
Anomaly detection	✓	✓	✗	✗
Recommendation	✓	✓	✓	✗
Ranking	✓	✓	✗	✗
Data Labeling	✓	✓	✓	✓
MLOps pipeline support	✓	✓	✓	✓
Built-in algorithms	✓	✓	✓	✗
Supported frameworks	TensorFlow, MXNet, Keras, Gluon. Pytorch, Caffe2, Chainer, Torch	TensorFlow, scikit-learn, PyTorch, Microsoft Cognitive Toolkit, Spark ML	TensorFlow, scikit-learn, XGBoost, Keras	TensorFlow, Keras, Spark MLlib, scikit-learn, XGBoost, PyTorch, IBM SPSS, PMML

altexsoft
software r&d engineering

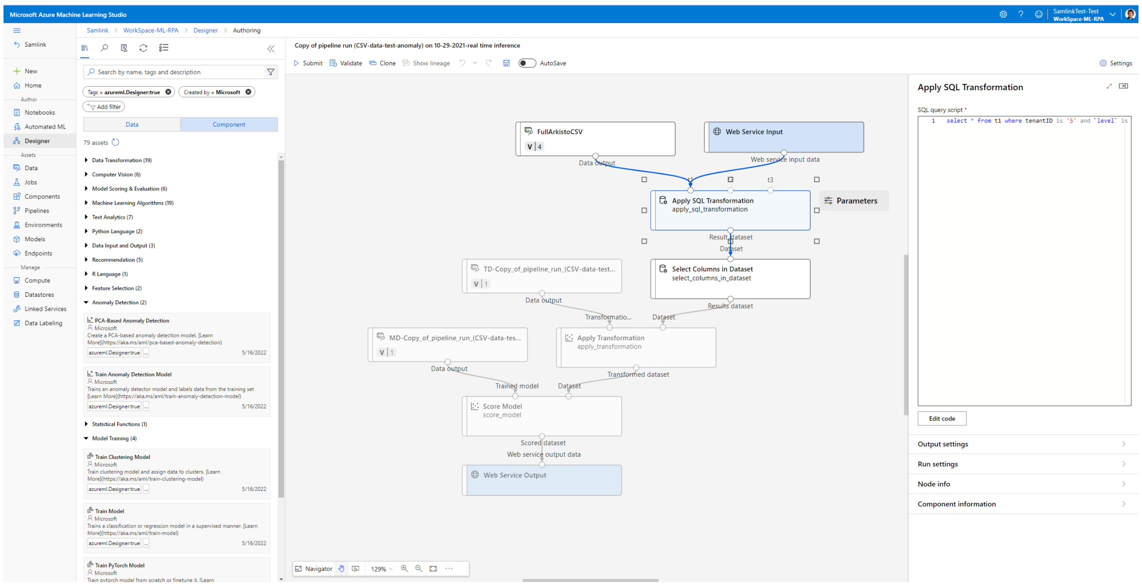
Figure 2: Machine learning as a Service comparison. [15]

utilizing services such as IoT, API, or Kubernetes.

2.3 Regression analysis

Regression analysis is a typical approach in statistical science, and thus, in machine learning too. Algorithms based on regression analysis are used to find relationships between a set of variables, providing the means to “predict values of one variable when given values of the others” making them a fundamental component in the ML field. [17]

Regression algorithms intend to create a mathematical model that explains the relations of the data. This usually means an algebraic equation which can be



```
select * from t1 where tenantID is '5' and `level` is
```

Figure 3: With drag-and-drop pipeline designer it is easy to get started with ML programming in Azure ML Studio, and visualizing the process helps understand all pipeline components and their relations to each other.

generalized in the following form,

$$Y = f(X_m, \beta_p) + \epsilon \tag{1}$$

where Y is the data feature we are looking to find relation to, f is some function of m independent variables (X_m), and p coefficients (β_p). X_m can be opened as $X_1, ..., X_m$, and β_p as $\beta_1, ..., \beta_p$. Note, that m and p do not have to be equal. A single X_i refers to a value of ith data row. The ϵ refers to error term. The goal is to determine the coefficients β_p in order to find a model that explains the data. [18]

As an example, one of the best known principles for coefficient value solving is the least square principle, which aims to minimize the sum of squared errors (SSE). The smaller the SSE is, the closer each data point is to the suggested model, and the better the model explains the data and can be used to make predictions. The equation to solve in the least square principle derived from the generalized form,

$$SSE = \Sigma\epsilon^2 = \Sigma[(Y - f(X_1, ..., X_m, \beta_1, ..., \beta_p))]^2. \tag{2}$$

Exact solution for this does not usually exist as there can be more coefficients than independent variables $(p > m)$), and minimizing the SSE does not always result into a linear equation. This is typically the case in ML, and thus the model must be found by iterative search process. As mentioned in the section 2.1, this is what algorithms are build for.

To improve the efficiency of algorithm calculations, the general regression model can be converted into matrix form. The same function in equation 1 can thus be represented as,

$$Y = XB + E \tag{3}$$

where Y is an $n \times 1$ matrix of values in the data, X is an $n \times (m+1)$ matrix of independent variables, B is an $(m+1) \times 1$ matrix of unknown coefficient parameters, and finally, E is an $n \times 1$ matrix of error parameters. Thus, the equation of squared errors (equation 2) that is to be minimized can be written as

$$E^T E = (Y - XB)^T (Y - XB) = Y^T Y - 2B^T X^T Y + B^T X^T XB. \tag{4}$$

In order to minimize this, we can take the derivative of matrix B,

$$\frac{\partial (E^T E)}{\partial B} = -2X^T Y + 2X^T XB \tag{5}$$

and equating to zero gives

$$(X^T X)\hat{B} = X^T Y. \tag{6}$$

The solutions of coefficient parameters are thus

$$\hat{B} = (X^T X)^{-1} X^T Y. \tag{7}$$

Ultimately, there usually is no one perfect model to describe real world data. As an example, the function f in the general regression model can describe the relation of X_i and β_i as a product of both terms. This would mean, that

$$f(X_1, ..., X_m, \beta_1, ..., \beta_m) = \beta_0 + \beta_1 x_1 + ... + \beta_m x_m \tag{8}$$

which would be a linear regression model. Example of an algorithm using this method is visualized in figure 4. However, real world problems cannot always be explained with a linear model. Fitting a polynomial curve to the data may improve the results of regression to a certain degree, but it also has its limits.

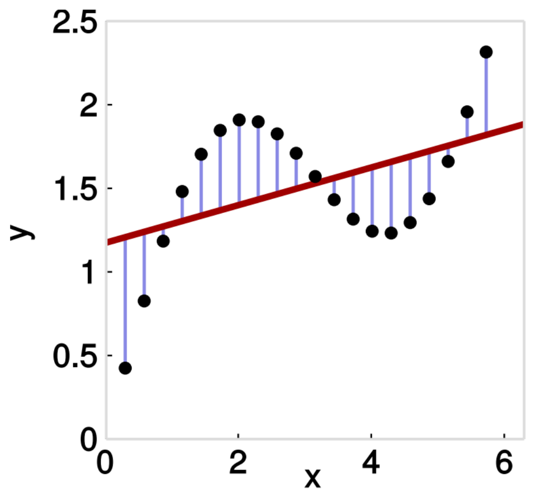


Figure 4: Linear least squares algorithm. Algorithm tries to fit a linear model (red line) on the data points, represented by the black dots, by minimizing the difference between all the data points and the model estimate. [19]

Consider an algorithm that tries to keep a car driving on a straight road in its lane. The model output is how much we should turn the steering wheel based on how much we are off from the straight line. We might have constant parameters such as wind effect, continuous error of the wheel axes, and tilt of the road. However, we also have parameters that change over time and are dependent on other factors, such as the weight of the car, and the temperature and wear of the tires. Therefore, with the same location at the road, same car and same weather conditions, our result could be different depending on how long has been driven before the current moment which affects the tire temperature and thus pressure. Finding a model that best fits the data depends also on the type of problem. There are numerous algorithms to suit different situations, and usually the best algorithm can be found only by trial and error.

2.4 PCA-based anomaly detection

Principal Component Analysis, or PCA, is a machine learning technique used to analyze data and explain the variance in it. PCA analyzes data with multiple variables and looks for correlations among them. The final output PCA gives is a new feature space, i. e. a smaller set of features (variables), called *principal components*. [20]

In other words, PCA works by reducing the dimensionality of the data. First, data must be standardized so that the mean of each variable is zero and the scale of each feature is the same.

$$Standardized\ data\ point = \frac{data\ point\ value - mean\ value\ of\ the\ feature}{standard\ deviation\ of\ the\ feature} \tag{9}$$

In practice, the data points of each variable, or column, is shifted so that their center is at 0 and the scale is adjusted to match all the variables.

Next, a covariance matrix for all the columns is determined. As it is known, covariance value between variables (or dimensions) is calculated with

$$cov(X, Y) = \frac{\Sigma_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)} \tag{10}$$

where X and Y are different columns or variables, $\bar{X}$ and $\bar{Y}$ their mean values (which both are zero after standardization), n is the number of rows or values in the data, and so X_i is the ith data point in the column X. [21]

Covariance matrix is formed by calculating the covariance value between each of the dimensions (including self). For example, for 3 dimensional data with dimensions x, y, and z, the covariance matrix would be

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}. \tag{11}$$

Generally, the covariance matrix for n dimensions would then be, of course

$$C = \begin{pmatrix} cov(1,1) & \cdots & cov(1,n) \\ \vdots & \ddots & \vdots \\ cov(n,1) & \cdots & cov(n,n) \end{pmatrix}. \tag{12}$$

Next, we find the eigenvectors and eigenvalues for the square-shaped covariance matrix. As defined, for $n \times n$ dimensional matrix A, if exists a vector x that satisfies

$$Ax = \lambda x, \tag{13}$$

where λ is a scalar value, such vector x is an *eigenvector* of matrix A, and λ is an *eigenvalue*, forming the *eigenpair* of (λ, x). [22]

Eigenvectors have a property, that either a matrix has zero of them, or there are n eigenvectors for $n \times n$ dimensional matrix. Because in this case the eigenvectors are for the covariance matrix of the original (standardized) data, they actually create a new feature space made of principal components. More over, eigenvalues describe their importance, so that the greater the eigenvalue is, the more significant the eigenvector is describing the original data in principal component feature space. When finding eigenpairs, we actually find the largest possible variance in the data. Each eigenvector describes a principal component that is a linear combination of the original variables. In figure 5 we can see an example of how eigenvectors of the covariance matrix define a new axis system or new feature dimensions. [23] Each eigenvector, or a principal component, is perpendicular with one another. Thus, the data can be represented in the new axis system formed by eigenvectors.

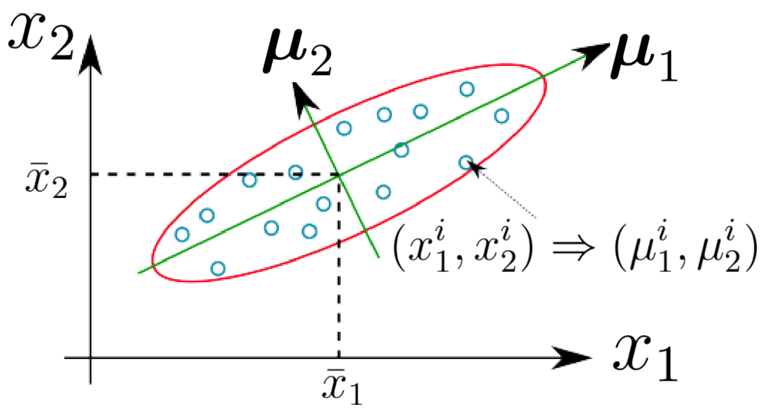


Figure 5: Principal components form a new axis system. In this two dimensional example, eigenvectors μ_1 and μ_2 of the covariance matrix define new feature dimensions. Thus, data point (x_1^i, x_2^i) of dimensions x_1 and x_2 can be presented as a point (μ_1^i, μ_2^i) of the dimensions μ_1 and μ_2. The origin point of each principal component axis is at the mean values of the data ($\bar{x}_1$ and $\bar{x}_2$) as the data is standardized. [23]

By comparing the eigenvalues, we can determine a set of eigenvectors, or principal components, that form a new feature space with fewer dimensions than the original data, without losing a significant amount of information. We can calculate a comparison value for each eigenvalue λ_i with

$$Percentage\ of\ variance = \frac{\lambda_i}{\Sigma\lambda} \cdot 100 \tag{14}$$

Organized from highest to lowest, we can make a scree plot to visualize the eigenvalue ratios for all principal component. Example of an 8 dimensional data (which thus has 8 principal components), with their eigenvalue variances ordered can be seen in figure 6.

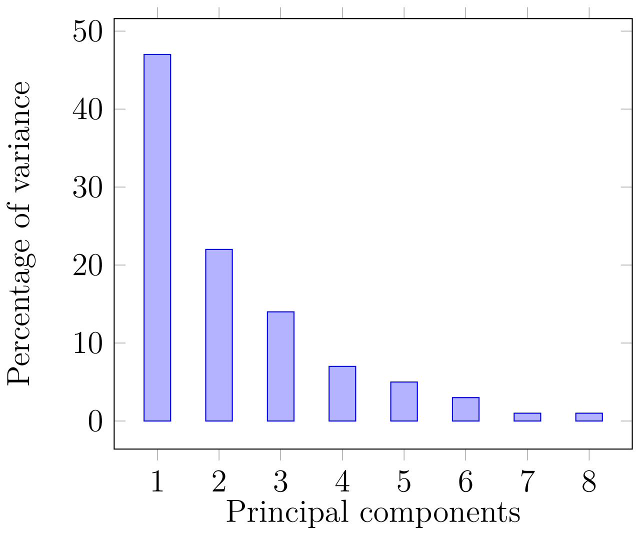


Figure 6: Significance of principal components. In this example, eigenvalues, or the variance of each principal component, are ordered on a scree plot. Values are first converted to a percentage of the sum of all values in order to visualize both their relationship to each other and their total share of information held by the corresponding eigenvectors.

Practically, as principal component space describes the data and eigenvalues describe the importance of each component, the percentage of variance is the portion of the total information each principal component holds. Simply put, in our example in figure 6, the first two principal components hold almost 70% of the original information, the first three over 80%, and the last seven hold less than 20% of the information. Thus, by creating a new data set using the first three components, we can keep 80% of the information while reducing the dimensions from 8 to 3, which would be significant improvement in terms of needed computing resources. The amount of top principal components we choose to keep depends on the situation, but one guideline is to select all components that describe more than a single variable's worth of information. For our 8 dimensional example, that would mean 1/8th, or 12.5%, meaning that we would select top 3 principal components and discard the rest that are under 12.5% of variance. The remaining vectors form a new matrix called *feature vector*:

$$Feature\ vector\ (F) = (x_1 x_2 ... x_p) \tag{15}$$

where x_i is the ith eigenvector or feature component we chose to keep, and p is the amount of them, where $p < n$, n being the original amount of dimensions of the data.

Finally, we reorient our data from the original axes to the space defined by the principal components with the following equation:

$$Final\ data = F^T A^T \tag{16}$$

where A is the original data set with standardized data, which is transposed as well as the feature vector multiplying it.[24, 25, 21]

With principal components describing the distribution and variance of the data majority, we can use PCA to detect anomalies. This is done by analyzing each input and computing its "projection on the eigenvectors, together with a normalized reconstruction error. The normalized error is used as the anomaly score. The higher the error, the more anomalous the instance is" [20]. Normalized error means, that each anomaly score is between zero and one. Figure 7 illustrates the steps from data standardizing to PCA reconstruction error calculating.

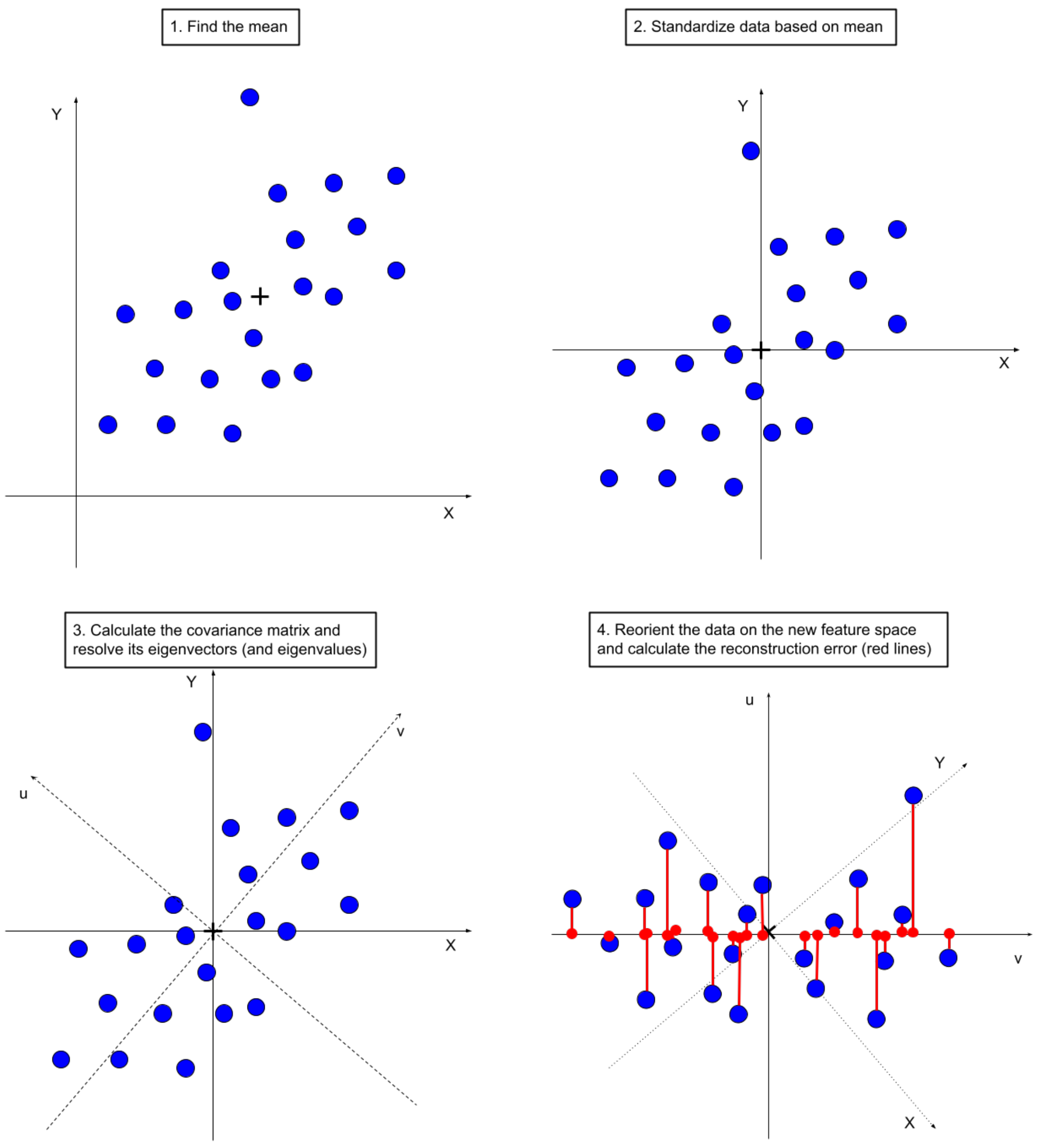

1. Find the mean
Y
X
2. Standardize data based on mean
Y
X
3. Calculate the covariance matrix and resolve its eigenvectors (and eigenvalues)
Y
X
u
v
4. Reorient the data on the new feature space and calculate the reconstruction error (red lines)
u
Y
v
X

Figure 7: PCA anomaly detection explained step by step. The normalized reconstruction error is the final anomaly probability score.

2.5 Other anomaly detection algorithms

Azure ML Studio has also another anomaly detection algorithm to use besides the PCA-based one. This module is called One-Class Support Vector Machine. However, it was not usable in the renewed ML Studio environment, but was only available in the *classic* Azure ML Studio. In addition, this module was not deemed suitable in our case, as the documentation mentioned that "The dataset that you use for training can contain all or mostly normal cases." Because the content of the data used did not meet this requirement, the usage possibilities of this component were not decided to investigate. [26] To limit the scope of this study to reasonable proportions, we will not discuss other anomaly detection algorithms (ADAs) further.

2.6 N-gram features and feature hashing

As discussed before, features are the key elements in ML algorithm training. As textual input does not have any meaning to computers by itself, it is necessary to create a connection between words and features for algorithm. In ML training, one typical approach is to convert textual input into numerical features. For example, by creating a dictionary of words used in the input and assigning each word an identification number, we can express sentences as a count of certain words used. In addition, words include meanings not only individually but also with relation to each other and with their order. For example, words "success without errors" has the opposite meaning than "errors without success" where same words exist in a different order. We can add more information for the algorithm by creating word pairs and groups in the dictionary. These groups are referred to as word grams, where **n** in n-gram refers to the maximum number of words in a group of consecutive words in the input sentence. [27]

Example of n-gram composition of a sentence can be seen in the table 1. Usually, the unnecessary stop words are removed from the text (such as "the"), as they don't bring any additional meaning to words or sentence. One word grams are called *unigrams*, two word grams *bigrams*, etc. . To compose the sentence *"Lorem ipsum dolor sit amet, quisquam est, qui dolor ipsum qui dolor sit amet"* with different n-gram dictionaries, we could make, for example, the following representation with unigram dictionary:

$$1, 2, 3, 4, 5, 6, 7, 8, 3, 2, 8, 3, 4, 5 \tag{17}$$

where each number represents the index of a word in a unigram dictionary from table 1.

N-gram representation of the text streamlines the algorithm processing of textual features. However, each new word creates a new feature when converting text to n-grams. This happens, because in order to present text as numerical features, we need to create a new feature space out of the n-gram dictionary. Consider our unigram dictionary from the previous example. This would convert one text column into 8 unigram features, as seen in table 2.

As the number of word grams in a dictionary can increase significantly in complex input cases, it is necessary to limit the resource usage by decreasing the features

Index	**Unigrams**	**Bigrams**	**Trigrams**
1	lorem	lorem ipsum	lorem ipsum dolor
2	ipsum	ipsum dolor	ipsum dolor sit
3	dolor	dolor sit	dolor sit amet
4	sit	sit amet	sit amet quisquam
5	amet	amet quisquam	amet quisquam est
6	quisquam	quisquam est	quisquam est qui
7	est	est qui	est qui dolor
8	qui	qui dolor	qui dolor ipsum
9		dolor ipsum	dolor ipsum qui
10		ipsum qui	ipsum qui dolor
11			qui dolor sit

Table 1: N-gram feature extraction from a sentence “Lorem ipsum dolor sit amet, quisquam est, qui dolor ipsum qui dolor sit amet”.

row	**lorem**	**ipsum**	**dolor**	**sit**	**amet**	**quisquam**	**est**	**qui**
1	1	2	3	1	2	1	1	2

Table 2: One row with a textual feature presented as n-gram feature transformation. Each new word in the n-gram dictionary adds a new feature and thus a dimension to the data.

that are analyzed. One way to compress the dimensions is to use feature hashing for n-gram features. [28] This means that instead of pure n-gram features representing a single n-gram instance, we use hashed value of several n-grams thus reducing the amount of features. Hashing can be done in different ways, for example by multiplying each original feature together, or calculating a weight value based on the frequency of each feature. As a drawback, the amount of information might also get reduced as the data is “compressed”. The more dimensions are compressed into hashing values, the more information is bound to be lost, but this way we can include more features for algorithm training without significant resource demands. [29, 30]

2.7 Robotic process automation

Robotic process automation, or RPA, is used to automate mechanical tasks executed on computer software. Usually it operates on the UI level and can be used to repeat meaningful functions instead of mechanical actions. For example, with screen recording macros, only mouse position at the screen and actual pressing of the keyboard is recorded and repeated. RPA automation, however, is able to repeat the functionalities those actions trigger, such as inputting text to a certain named field on the UI, or logging in with a given username and password regardless of the location of those fields on the layout. [31]

In Samlink, an RPA technology called UiPath is used as a base of RPA operations. A central coordinating system called *UiPath Orchestrator* supervises the RPA processes. RPA process is executed by following an *RPA automation*, a predefined operation instruction build by RPA developer. The software responsible for the automation execution is called *RPA agent*. Agents are located in the *workstations*, where a single agent is running in one workstation. Each agent executes one automation at a time, given by the Orchestrator, and communicates with Orchestrator during the execution. This hierarchy is visualized in figure 8.

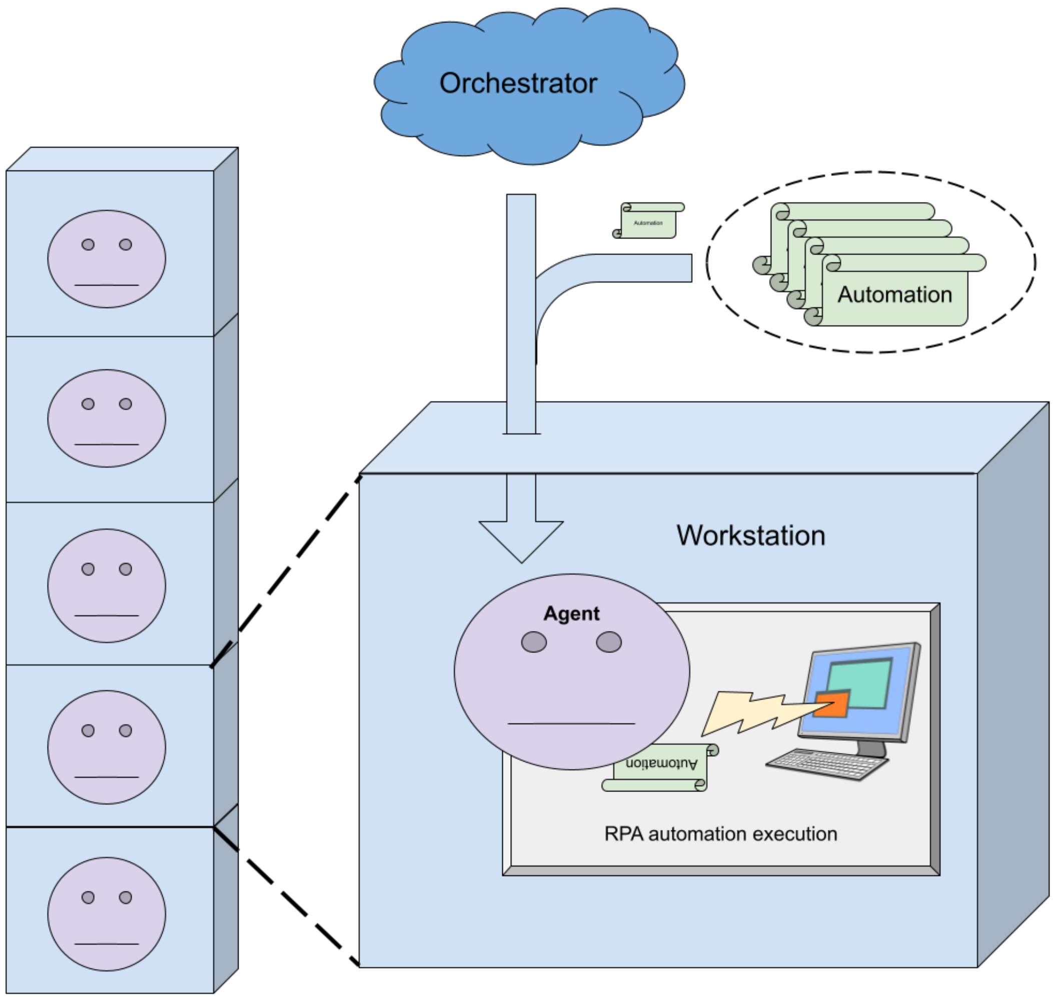


Figure 8: Hierarchy of RPA components explaining the terms and their relations.

2.8 Data sensitivity

During this study, it was necessary to make sure no sensitive data was moved out of the production environment. This was mostly due to restrictions imposed by GDPR. In order to maintain the data security, the data had to be anonymized before it could be exported to the cloud environment. After anonymization, the data would not include any information that can be connected to real individuals. Three different anonymization methods were considered, which were pseudonymization, k-anonymization and full anonymization.

Pseudonymization refers to a method where sensitive information is de-identified. This means, that each sensitive piece of information is replaced with an encrypted value so that no information is lost but a human cannot identify individuals when reading the data. Encryption and de-identification could be reversed, i. e. data could be re-identified, with a decryption key which tells a computer how to convert the replaced value back to the original form. As machine learning algorithms do not care about the meanings behind personal identification information, such as phone numbers or addresses, pseudonymization would preserve the information in the data unchanged for ML algorithms to use so that no information would be lost. [32]

As pseudonymization is a reversible operation with an encryption key, it is not the safest way to anonymize the data because the encryption key leaking is always a risk. K-anonymization is the next step in securing the data sensitivity. Excluding all unique identifiers such as full name or social security number, information like home street, age, workplace or last name are not on their own enough to identify a certain individual, but combined they can single out a person. K-anonymization is an unreversable anonymization approach where identifying information is generalized to mask individuals into a crowd. With k-anonymization, an algorithm replaces single informative details with more general variants, for instance, address to hometown or age to age range. K-anonymization loses information as it cannot be reversed. If personal information is essential for the use case of the ML algorithm, this method weakens the algorithm results. [33]

Eventually, due to high customer data sensitivity and strict data safety policies, it was determined that individual information in the log data used in this study was not relevant for connecting the log events to technical support ticket timestamps. Thus full anonymization was decided to execute on the log data. This way, each personal information was replaced with a general token disclosing only what type of information (phone number, email address etc.) was anonymized. Frankly, it is not certain if identifiable personal data would have improved the algorithm results, but because corresponding support ticket data was stripped from all other information except timestamps, any possible connections between personal data in logs and in tickets were lost nonetheless.

Data anonymization was executed in the production environment with PowerShell script. Several predefined identification features were searched with regular expression (or regex) patterns and replaced with default keys. Anonymization scripts and the data format is described in more detail in the section 3.3.

2.9 Log data analysis and anomaly detection with ML

Using machine learning for log data analysis is not a new field of study. [34, 35, 36, 37] The key issue tends to be the format of the log data which shifts the dilemma to natural language processing. Some studies also combine anomaly detection using machine learning to log data analysis. [38, 39] Comparing existing studies to our case raises at least one major suggestion for improvement: log data refining.

When log data has a consistent format, multiple different algorithms can be utilized for anomaly detection and log analysis. Log events can be clustered and different types of events can be counted if the amount of types is finite and known. [38]

If training data already have information we wish to teach the algorithm to forecast (i. e. data is labeled), combining the results of the log data analysis to external features is more feasible. With labeled data, supervised learning methods can improve the results of the algorithm forecast abilities. [34]

As explained in the section 3.1, data features connected to anomaly detection results are pure datetime values. With more insight into ticket data properties than just timestamps, ML algorithms could be able to extract more valuable information from the log data.

3 Research material and methods

In this section we explain in more detail what the data used in the study consists of and what methods were used in an attempt to answer the research goals. The content of the section is briefly described below, after which the steps of the research are explained.

The data in the research is mainly made of two parts. The most important part is the log data produced by the numerous RPA processes. The second data part, complementing the study, is the support ticket data written by clerks of customer banks. In order to use the data safely in the cloud environment it was necessary to sanitize the data from any sensitive information. This was done by anonymizing the log data and using only timestamps from the support tickets.

After confirming the results of anonymization, the data was preprocessed into a better format to make it more usable by algorithms. More processing was done inside the pipeline as ML Studio offered several usable components for this but the main cleaning was easier to execute in a local environment. This was also done with PowerShell scripting.

The actual ML pipeline structure is discussed in the next section.

3.1 Support ticket data

Like all other software, RPA components fail from time to time. As described before, RPA logs are verbose making error identification difficult. Due to that, it is not feasible to create log parsers that would be able to identify critical errors from within thousands of lines of log. When a critical error occurs causing the RPA process to fail, the banking clerks need to finish the job that was left by the RPA robot manually. Every time this happens, the clerks then send a support request ticket to Samlink technical help desk and request a fix for the issue.

When clerks send the ticket to technical support, a verbose description of the situation is written to help developers to identify the problem. This description often contains sensitive end customer information like bank account details and social security numbers. To avoid privacy issues when processing this data, it was decided to use only timestamps of the tickets. The resulting data was practically a list of date and time values. The issue is described further from a privacy point of view in section 3.3.

3.2 RPA log data

Robotic process algorithms used in Samlink are designed to ease the workload of bank clerks. RPA robots work on behalf of bank clerks executing routine tasks that require mostly manual labor.

Like other software, RPA also produces log data during its runtime. As dozens of RPA automations are running in several bank environments the amount of log entries produced is also significant, up to over a million lines per week. This log data is not in consistent structure as it is formed out of typical CSV data and injected with even more inconsistent JSON data that varies in contents vastly. An example of log data after the anonymization phase can be seen in the appendix B.

RPA log data is stored in an SQL database. The database is split in live production log that is gathered for two weeks and then moved to an archive that has several years worth of log. In this study we used archived data as it was easier to acquire in one run without the need to merge different parts together. The archive also had an amount of data that was considered sufficient for machine learning algorithm training, with data entries spanning almost two and a half years and rowcount exceeding 80 million.

Samlink RPA logs have few standard fields. These are listed in table 3. The most notable aspects here are the fields named message and rawmessage.

Message holds the short log message written by the RPA agent during automation execution. This includes details about the issue, what part of the process failed, and possible stack trace of the error.

Rawmessage is JSON-formatted representation of all the default features, including the message and multiple other additional features that the RPA agent is able to output regarding the log event. These additional fields are what vary from log entry to log entry. Some of the possible fields are shown in table 3, but several other field types exist. The JSON data in them can be nested in multiple layers, as seen in appendix B.

Without rawmessage, the data was in pure CSV-format and could have been more easily processable from the start. However, it was not certain that rawmessage would not hold usable data for ML as in some cases the plain message-field did not include all the most interesting keywords that were present in the extra fields of rawmessage. Nevertheless, using rawmessage in anomaly detection posed another problem.

When feeding the log data to the anomaly detection algorithm, it was crucial that all the rows were as minimally unique as possible in order to use the pattern finding abilities of the algorithm. Too unique data points would have made all of them anomalies compared to each other. Because rawmessage included other column data in a long text format, certain unique information such as fingerprint and timestamp were necessary to remove from within the data manually, as they could not be extracted easily with tools in Azure ML Studio during ML pipeline execution. Thus, it deemed easier to preprocess the data with another script in a local environment before exporting it to the cloud. The final script is presented in appendix D. When training ML algorithm with rawmessage, timestamp and job ID values were included retrospectively from their corresponding columns outside rawmessage.

3.3 Data anonymization

Support ticket data privacy

Samlink handles highly sensitive banking customer data in its processes, such as personal identification numbers, home addresses, email addresses and bank account

Field	**Contents**	**Examples**
organizationUnitId	Samlink organization unit	5
level	Log level	Information | Warning | Error | etc.
logType	Type of log entry	Default | User
timeStamp	Date and time for log entry	2019-09-10T03:00:01.6278373+03:00
fingerprint	Unique identifier for log entry	bcd51984-agdc-40a6-b571-y6a97f98a4e3
machineName	Workstation name of the RPA agent	W2490N101
processName	Name of the RPA automation	RPA-bank-task-application_Samlink Production
jobId	Identifier for current RPA automation execution	24a84531-010b-457f-90t1-5ayc98d7b557
robotName	Name of the agent executing automation	RPA-BANK-1-1234
machineId	Workstation ID number	5
message	Short message regarding the entry	Throw exception: Saldo ei riitä | Tarkistettiin A:n nimi
rawmessage	JSON formatted message including most of the columns above and several more fields regarding the log entry	{ **“message”**: “Siirryttiin Varallisuus-sivulle.”, **“level”**: “Information”, **“logType”**: “User”, **“timeStamp”**: “2019-09-10T03:00:50.6103121+03:00”, **“fingerprint”**: “70f44345-22bb-46df-885e-75f180fc4d48”, **“windowsIdentity”**: “LOCAL\\\\T123456”, **“machineName”**: “W2490N101”, **“processName”**: “RPA-bank-task-application_Samlink Production”, **“processVersion”**: “1.0.7111.31245”, **“jobId”**: “24a84531-010b-457f-90t1-5ayc98d7b557”, **“robotName”**: “RPA-BANK-1-1234”, **“machineId”**: 11, **“fileName”**: “LisaaTiedot_Talous”, **“logF_BusinessProcessName”**: “rpa-bank-011-JOB-valmistelu” }

Table 3: Log fields in RPA log data

numbers. All possibly sensitive data had to be removed before data could be transferred out of the production environment and into cloud. Due to bureaucratic reasons, technical support tickets were under more strict policies. Because of this, they were allowed to be used in the research on the condition that no business critical nor customer sensitive information was processed in the first place. The only way to assure this, was to select solely timestamp fields from ticket data. Thus, no sanitation for ticket data was needed as ticket data consisted of only a list of datetime values.

RPA log data sanitization

Information privacy is one of the key values in Samlink's business promise as the company develops high security banking applications and processes sensitive customer data. Thus, several aspects were needed to take into consideration before log data could be authorized for thesis study usage. To improve privacy, it was decided to assume that personal customer details are not critical information for ML algorithm training if the goal is to find possible problems in RPA runtime and not detect individual customer related problems. Also, as mentioned in section 2.8, all customer and user related information was excluded from the support ticket data, making individual customer information redundant in the log data. Thus, it was not necessary to achieve just adequate security by less safe and more effort consuming ways such as pseudonymization or k-anonymization (explained in the section 2.8), which would have also required strict inspections before the data could have been approved for cloud processing.

During the beginning of this thesis study, several sensitive information types were recognized from the log data, and all possible types and their different forms were considered before anonymization script was approved to be used in the production environment. Anonymization was executed by replacing sensitive information with general pattern describing the replaced information type. With this, it was at least possible to keep the information whether a log row had included customer sensitive data, and what type of data was involved. Thus, it was theoretically still possible to recognize repeating anomalies that had, for example, social security number included in the log event. A list of considered sensitive info types, their examples, as well as the values replacing the sensitive data can be seen in table 4.

A script searched for repeating patterns related to sensitive information types. This was done with regular expression, or regex, searching. With regular expressions, different repeating patterns can be extracted and replaced from the data. Each sensitive data type has some sort of unifying feature, such as string length, number of digits, or location of certain character.[40] Some types are clear and standardised like social security number (6 numbers, '-', '+' or 'a', three numbers and a number or a letter), or credit card number (15-16 numbers), while with some other types it may be hard to take all possibilities into consideration, like address (one or more words, at least one number, possibly one letter or more in case of 'apartment' or 'apt', more numbers etc.). Still, most of the cases could be considered to a degree that was deemed satisfying from a security point of view. All regex search clauses are listed in table 5

Info type	**Example**	**Replaced value**	**Comment**
Social security number	010190-0123	10105051470101	Includes '-','+' and 'a' format
Email	author@thesis.fi	EmailAddress0101	
IBAN number	FI8612345600000123	1010IBANnumber0101	Only Finnish format
BBAN number with dash	123456-123	1010BBANnumber0101	
Phonenumber, international	+358501234567	1010PhoneNumberInt0101	With or without whitespaces
Phonenumber, local	050-1234567	1010980230101	With or without whitespaces or dashes
Business ID	1234567-8	1010BusinessID0101	Finnish format
Business ID, international	FI12345678	101086512350101	
Business ID, int. zero form	0012345678	1010865123500101	
Credit card number	4920191061682346	1010664900101	
Windows Identity	K123456	1010WinID0101	Used in company processes
Address, common	Teekkarikuja 1 a 42	1010AddressCommon0101	Common street name endings
Address, ZIP	Bulevardi 2 B 69, 00100	1010AddressZip0101	Disregards city name after ZIP
Bank ID	12345678	10108426100101	Bank user ID
BBAN without dash	12345600000123	101088420101	
Artificial business ID	8123456789	101086512354970101	Used in RPA processes

Table 4: Information replaced with Regex search from log data. Data values are replaced with patterns with numbers or numbers and letters depending on the original format in the data. Patterns are formatted uniquely so that they can be recognized amongst the anonymized data, each starting with 1010 and ending with 0101, and having a typewise identifier in the middle. With numeric patters, numbers are selected as letter representations, like business ID = 8651235 (BUSINES)

Info type	**Regex**
SSN	?<![a-zA-Z0-9])[\d]{6}[-a+]?[\d]{3}[\w]{1} (?:0{0}|0{3})(?![a-zA-Z0-9])
Email	[^\‘"\s]+@[\.\w -]*[\w]
IBAN	(?:(?<![a-zA-Z0-9])|(?<=\\\D))(?:FI|fi) (?: ?\d){16}(?![a-zA-Z0-9])
BBANwith-Dash	(?<![a-zA-Z0-9])[\d]{6}-[\d]{2,8}(?![a-zA-Z0-9])
PhoneInt	(?<![a-zA-Z0-9])\+358(?: ?\d){8,10}(?![a-zA-Z0-9])
PhoneLoc	(?<![a-zA-Z0-9-])[0][\d]{2,3}[-]? (?: ?\d){6,8}(?![a-zA-Z0-9-])
BusinessId	(?<![a-zA-Z0-9])[\d]{7}-[\d]{1}(?![a-zA-Z0-9])
BusinessIdInt	(?<![a-zA-Z0-9])[a-zA-Z]{2}[\d]{8}(?![a-zA-Z0-9])
BusinessId-IntZero	(?<![a-zA-Z0-9])[0]{2}[\d]{8}(?![a-zA-Z0-9])
CreditCard	(?<![a-zA-Z0-9-.])[\d]{1}(?: ?\d){14,15} (?![a-zA-Z0-9-])
WinId	(?<![a-zA-Z0-9])[a-zA-Z]{1,2}[\d]{6}(?![a-zA-Z0-9])
AddressCom	[^\s ""’,.]* ?(katu|tie|kuja|polku|kaari|linja|raitti |rinne|penger|ranta|väylä|taival|tanhua|portti |veräjä|laita|reuna|syrjä|aukio|tori|laituri|tunneli) [\d]{1,3}(?[a-zA-Z.]{1,4} ?[\d]{0,3})?(?!\w)
AddressZip	(?<=\s)[\S]* [\d]{1,3}(?[a-zA-Z.]{1,4} ? [\d]{0,3})?(\s |,\s)[\d]{5}(?!\w)
BankId	(?<![a-zA-Z0-9-])[\d]{8}(?![a-zA-Z0-9-])
BBANnoDash	(?<![a-zA-Z0-9])[\d]{14}(?![a-zA-Z0-9])
ArtifBusines-sId	(?<![a-zA-Z0-9])[89]{1}[\d]{9}(?![a-zA-Z0-9])

Table 5: Regex search patterns for sensitive info finding. Most of the regex patterns start with negative lookbehind and end with negative lookahead so that found pattern is not part of another string. Order of the regex patterns as listed on the table is important as some patterns give overlapping matches so we wish to recognize certain patterns before others.

As the production environment is built on a Microsoft Server based solution, and because it was highly unrecommended to install additional software to the production server, data acquiring and anonymization tools were chosen based on what was already usable in the RPA production environment. Microsoft PowerShell offers sufficient tools for database SQL querying and stream editing. The amount of data was significant which made straight file editing impossible due to the local machine memory limitations. Thus, stream editing was necessary for finding and replacing sensitive information from the data.

Anonymization took a good portion of the research time as processes were slow, the amount of data was huge and multiple re-runs were needed before the results were deemed adequate. The final anonymization script is introduced in appendix C.

3.4 Azure cloud resources

Azure provides a vast set of tools and resources for different kinds of cloud projects. Resources needed for Azure ML Studio usage depend on the subscription used and the security restrictions set by the subscription manager. When starting this study, due to these restrictions, all resources used for ML training in this project needed to be inside the same virtual network. The Azure ML Studio environment could be opened from any network, but most of the features were unavailable if the computer browsing web UI of the studio was outside this virtual network. Thus, a virtual machine had to be acquired as Azure resource from within the same network as other resources and ML Studio UI had to be opened with the browser on this machine. Later on it was found, that Azure Machine Learning Workspace networking feature could be configured to allow public access making it possible to access ML Studio UI from all networks.

Resources needed for Azure ML Studio usage and their relations are shown in figure 9. This is the configuration required with the existing subscription models of Samlink, albeit different configuration combinations could be possible depending on the subscription and network restrictions. The most crucial parts were the storage account, private endpoint and virtual network.

The storage account *worspacemlrpa* was linked to the Azure Machine Learning workspace. This storage is the main disk that stores all the input data used for algorithm training as well as the result data from said algorithms. Virtual network *vnet-test-machine-learning* is the mentioned network that includes both ML workspace and the virtual machine used to access the ML Studio web UI. Private endpoint *endpoint-ml-nic01* acted as a final link between ML workspace and the virtual network.

3.5 Azure ML Studio

The actual machine learning pipeline design and algorithm training is done with ML Studio portal, which is a graphic web user interface. It is a separate tool usable outside the usual Azure environment after configuring the workspace in Azure resources.

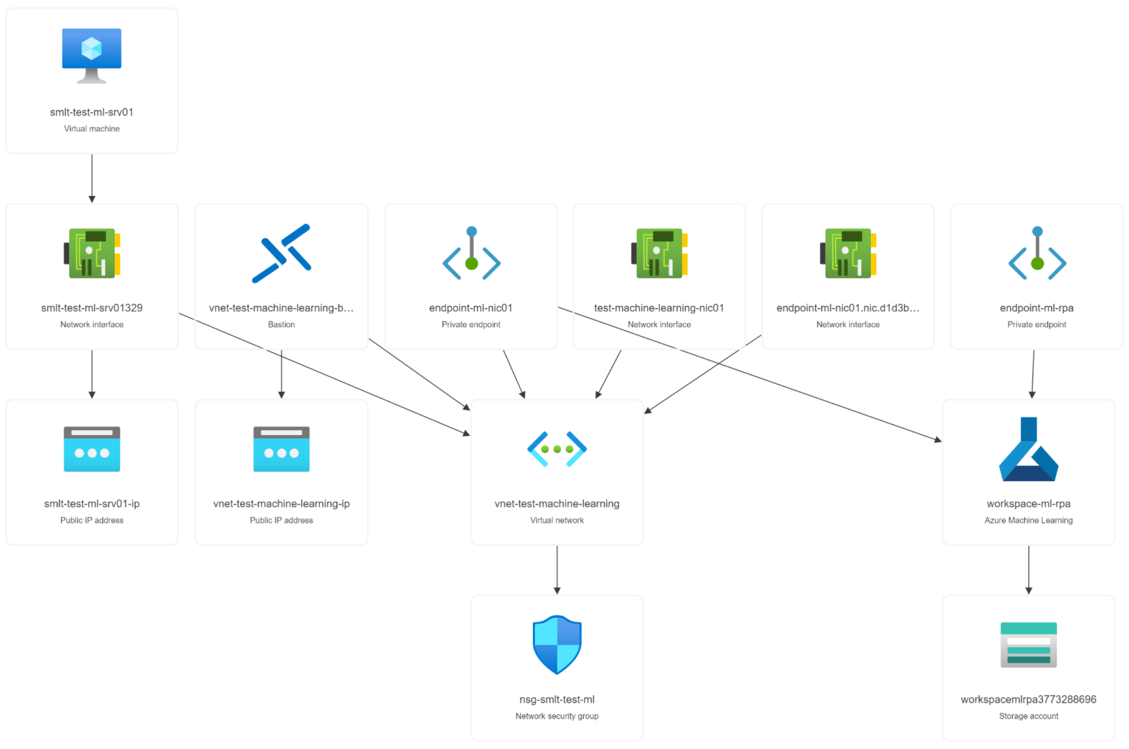


Figure 9: Azure cloud resources needed and their relations

Inside ML Studio, there are two important resources needed to set up before pipeline designing can start.

First, the data have to be registered as an usable asset. Multiple pre-existing data sources are available to choose from online sources, and the designer can also choose to use other accessible web file sources. In addition to these, datasets can be imported from Azure datastore accounts. Data can be uploaded from a local machine through the Studio UI so it is saved to the chosen datastore. As datastores are usable with all Azure service resources, it is possible to use data gathered from any other Azure service for machine learning training. In this study, the data that was anonymized and preprocessed in local machine was then imported with a separate Azure software to the storage account and registered as an ML dataset from Studio UI.

In addition to the data we used to train the algorithm, we needed to set up a computing instance in Azure ML studio. Some predefined resource limitations affected the computing instance choosing. After encountering some memory related issues in pipeline training, we were encouraged to pick memory prioritized instances. Single computing instance did not work, but we needed to choose a computing cluster instead to be able to run an ML training pipeline. After choosing suitable virtual machine properties for compute cluster instances, it is possible to set a number of nodes. Each of these nodes are a copy of the virtual machine chosen in the previous step. If more than one node is chosen, the computing cluster scales to use more nodes depending on the ML training demands.

For the purposes of this study, we chose to use maximum two nodes. The virtual machine properties chosen were 4 core machine with 28 GB of RAM and 200 GB of disk space.

4 Machine learning pipeline structure

The full component chain from input to output with algorithm training and result validating is called a machine learning pipeline. In this section, we discuss how the ML pipeline was created in Azure ML Studio. Several ML algorithms were compared in order to find the most feasible set for our goal in mind. ML training was organized in two different phases in order to find the relation between log anomalies and technical tickets. Although, the Azure environment and ML Studio requirements were the objectives of the study and therefore part of the outcome of the results, these results were also a prerequisite for solving the final objective considering the possibilities of the ML algorithm. Therefore, the resulting Azure resources and ML Studio pipeline components are demonstrated in this section.

Results of the trained algorithms were validated against newly acquired production data in order to estimate how well the initial goals of the study were fulfilled. These results are presented later in the section 5.

Azure ML Studio makes ML pipeline creation easy and comparing different methods and algorithms effortless. Nevertheless, with a hybrid approach having two different phases, and result comparison being done against the anomaly hypothesis, the pipeline drafts started to accumulate in content.

When starting the ML pipeline testing, the initial plan was to feed the log data to the anomaly detection algorithm and try to get some sort of estimate of possible anomaly count. This plan had several problems. First, as stated, logging is very abundant and several thousands of rows is logged during a single day. Some encountered errors are not critical and RPA agent is able to recover from them and finalize the initial task. This means, that errors which could be deemed anomalous may not result to a ticket in the end.

In addition, one single error case noticed by bank clerks may be linked to several problems in runtime, meaning that one ticket is might be linked to multiple, dozens, or even hundreds of log rows.

Two different algorithms were needed. In phase 1, the algorithm defines how likely one datapoint, or log row, is to be considered an anomaly. In phase 2, another algorithm aims to predict how many tickets are expected to be received within a time frame. This dual algorithm utilization is referred to as a hybrid machine learning approach. [41]

4.1 Hybrid machine learning

Hybrid machine learning (HML) refers to an ML technique where two or more ML methods are combined to overcome the limitations of or to boost the estimation capabilities of a single method alone. [42] Hybrid machine learning is not a rare technique in the ML field. [43, 41, 44, 45, 46, 47, 48, 49] In this study, we combine a PCA-based anomaly detection algorithm with a regression algorithm in order to amplify the prediction powers of our ML algorithm when trying to determine the possible ticket count based on log events.

We use two different algorithms and two particular data sets in two separate phases as visualized in figure 10. The results of the first ML algorithm are combined with the second set of data, and this combination is used to train the ML algorithm in the second phase. In order to clarify whether a hybrid approach is suitable for the current study problem we will compare the results of the hybrid ML technique with a single ML algorithm usage.

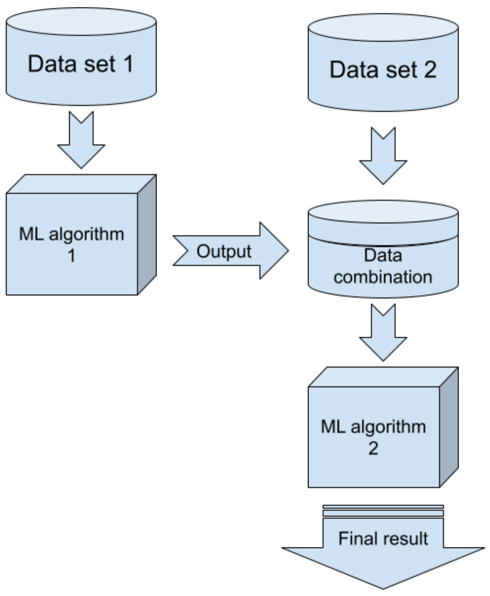


Figure 10: Simplistic example of a hybrid machine learning model. The first algorithm learns from the initial data, and the results are used with a second data set to train another algorithm.

It is not feasible to use anomaly detection on its own to estimate ticket amounts as plain sum of anomalies detected is not correlating with tickets received. Using pure statistical values of the log data such as log rows per day would possibly give some results with a single algorithm, but only if the amount of log events correlates strongly with tickets received. We can, however, amplify our ticket estimating algorithm with anomaly feature values. As we first count the anomaly numbers with the anomaly detection algorithm and use the statistical features of the results in another algorithm, like regression algorithm, we get more relative information to use when creating the final ticket number estimations. This is explained in more detail later in section 4.3.

4.2 HML phase 1: PCA-based anomaly detection

PCA-ADA input feature formatting

In Azure ML Studio, the only selectable module for anomaly detection is the PCA-based anomaly detection algorithm (PCA-ADA), which is explained in section 2.4. However, with textual input like logs it can be used in at least two ways.

First, input data can be fed to the algorithm trainer as is, letting the PCA-ADA component do the work without further modifying the log rows. This way, the component tries to recognize the anomalies based on all the information included in the row. Practically this means, that the component processes data in textual format making each row in the input a feature as a whole to consider. As we discussed in the section 3.3, unique values should be removed from the log rows and the data should be somewhat clean. PCA should be able to find similar values and create anomaly score with purely textual features. However, it is probable that with a single word change on a message feature, PCA defines two rows as completely different.

Second option is to convert the textual features into numerical features. This can be done either with the "Extract N-Gram Features from Text" component, or the "Feature Hashing" component which uses n-gram feature extraction behind the scenes. With n-gram feature extracting, as explained in the section 2.6, each word or n-gram is converted to a number of said instance found on the row being processed, and each row can be presented as a sequence of numbers indicating the number of those features.

An N-gram feature can in addition have a weight based on the frequency the n-grams appear in the entire data. Different weights usable in Azure ML component are listed in table 6. During this study, only binary weight was used, so it is possible that improved results could have been acquired with a different weight method.

N-gram weight	**Explanation**
Binary Weight	Assigns a binary presence value to the extracted n-grams. The value for each n-gram is 1 when it exists in the document, and 0 otherwise.
TF Weight	Assigns a term frequency (TF) score to the extracted n-grams. The value for each n-gram is its occurrence frequency in the document.
IDF Weight	Assigns an inverse document frequency (IDF) score to the extracted n-grams. The value for each n-gram is the log of corpus size divided by its occurrence frequency in the whole corpus. `IDF = log of corpus_size / document_frequency`
TF-IDF Weight	Assigns a term frequency/inverse document frequency (TF/IDF) score to the extracted n-grams. The value for each n-gram is its TF score multiplied by its IDF score.

Table 6: Statistic metrics of the time frame compression that are considered possibly useful for ML algorithm. [50]

As stated, a vast dictionary of n-grams demands resources from the ML computing instances as every new n-gram in the dictionary adds another column to the training dataset. To reduce the amount of memory needed, the "Feature Hashing" component can be used. By hashing the n-gram features, the amount of resources needed by the pipeline can be significantly reduced. Feature hashing allows us to use the entire amount of data as an input if feature hashing parameters are tuned enough. However, as discussed in the section 2.6, the greater the compression is, the more information is lost to reduce the need for resources.

Before the n-gram operations, the textual data can be preformatted in Azure ML Studio with “Preprocess text” -component. This component includes several options to choose from in order to clean text to more processable form. Most useful options to select from are *stop word removal* (which removes uninformative words such as “the”, “is” and “and”), *lemmatization* (which converts words to their canonical form, for example “bigger mice eating” to “big mouse eat”), *detect sentence* (which inserts a sentence boundary symbol to help algorithm text analysis), *text case normalization* (which normalizes all characters to lower case to reduce the number of different words when first letter is capitalized), and various *character removals* (which range from number, special character, and duplicate character, to url and email address removal possibilities). Text preprocessing usually helps to reduce the number of features when text is used in n-gram feature extraction and feature hashing. Most of the options, however, do not work well with other languages than English, which might create issues with this study case as logs contain a mixed amount of English and Finnish words. [51]

In the end, the need for memory proved problematic and pure n-gram feature extracting forced us to reduce the data size to only 2% in order to finish the algorithm training pipeline. This amount was considered to be too low for reliable algorithm training results. Still, all variations of input feature formatting were tested to see how much possible shortcomings would affect the results. More about the memory issue is discussed in the section 5.1.

PCA output and anomaly probability

The output values of the PCA-ADA component, as explained in the section 2.4, are normalized so the values range between 0 and 1. This anomaly probability value is the main output of hybrid ML phase 1. Based on our initial hypothesis, that each anomalous event in the log is linked to a real life support ticket received, the bigger a single anomaly probability value is for a log row, the more likely is that the row is related to a ticket inducing event. Further processing steps of the output values are discussed later in the section 4.3.

Unconventional training approach

As stated in section 2.1, the approach we attempt in this study is, if expression is allowed, unorthodox. Typically, the data points used in ML algorithm training and validating should always be different. Acting otherwise leads to algorithm processing with same data it was trained with, thus creating a situation where algorithm already knows what to do with the current data point. If the results were validated after this the algorithm would get an unreliably good score as it had the validation data already in the training phase. This could be compared to giving some right answers to students during a test and scoring test results as if no help was given. However, due to the nature of the study problem and contents of the data, it was decided to test whether bending this rule would provide better results in algorithm training.

The large amount of data was enough to cause issues with memory. Although this problem was succesfully circumvented, the hybrid approach and the time frame compression (discussed later in section 4.3) resulted in a significant data loss in phase 2. As a general rule of thumb in ML training, only 20–30% of the data is used to validate the algorithm. With the hybrid ML approach, the validation results in phase 1 are what actually form the data used in phase 2. This data is further compressed to time frame groups leading to only a few dozen data points in phase 2 ML training compared to millions of rows in phase 1.

Because of the way the PCA-based anomaly detection algorithm works, the over-lapping data points are not as big of an issue as it would be with other types of algorithms like regression algorithms. This is why we could use part of the data for training the anomaly detection algorithm as usual and then use all the data available for validation without overfitting the algorithm, which happens when algorithm fits to the training data well, but cannot generalize with new data [52]. Also, because the main forecasting functionality comes in the phase 2, overfitting in phase 1 may not cause issues.

To verify if this unconventional training method gives good results without issues, the trained algorithms were tested with new production data that had zero overlapping data points with training and validation data. This training method was also compared to a traditionally trained algorithm to see the differences in results of both training styles. This way we were able to compare different training approaches to determine the best overall pipeline structure.

Kind of informal mention considering the traditional training approach in our study case is the data splitting of the log data for algorithm training. Because of the hybrid model, and the data consisting of log rows, we cannot make a random split for the training and validation data as is usually done. Purely for training the random splitting can be done, but if data in phase 1 is split randomly for validation, some anomalous rows could be skipped from a time frame that would be crucial information for the estimations in phase 2. This is why we must make sure the possible data splitting for validation data is chronological in phase 1.

4.3 HML phase 2: Ticket count estimation with regression

Input data random delay and time frame compression

As mentioned previously in section 1, it takes time for a bank clerk to notice the error in the RPA process, send a technical support ticket considering the issue, and for the support team to redirect the request to the corresponding developer team. As several steps of human interaction and workday schedules are in between the event of logging and ticket receiving, the random delay of such may span from hours to days. Random delay in input data features is not an unusual aspect in time-series forecasting. Time-series in the context of ML refers to data features that vary over time and can be affected by past values. [53] As an example, an ML algorithm could try to predict future weather based on measured temperature and air pressure. Both these features change over time and also affect their own future values.

This study, however, is not about time-series because the majority of the log rows are not affected by previously logged events. As random delay of such does not seem to be trivial to take into account with ML algorithms, a simple method to solve this was used where log rows were grouped by time stamp into certain time frame groups. We call this method “time frame compression method”.

Time frame compression means, that in order to eliminate the effects of random delay we compress some features into a certain time frame at least as long as the longest estimated delay. Simply put, if we count possible anomalies during one hour of log, we cannot compare this number to actual tickets received at the same hour or the next. What we can do, with time frame compression, is that we count some statistical values of anomaly estimates, for example, the mean and median values of a week, and then compare these numbers with the tickets received during the same week. Statistically important and thus compressible features of a time frame were determined to be the log row count, amount of unique job IDs, and anomaly probability metrics.

The amount of log rows describes how many logged issues occurred in a time frame. Alone, this feature may not give much insight as the amount of logs is possibly not linearly comparable to the amount of tickets received. Combined to other statistical metrics it may, however, provide additional value for ticket forecasting. Job ID is the identification information of a specific RPA automation execution run. It means, that the job ID is not unique for each log row, but it binds together all the log entries on the same RPA automation execution. By counting the amount of unique job IDs in a time frame we can get insight about the amount of automation jobs executed during the time frame. This metric is important with the total row count: low number of unique jobs with high number of total rows indicates that plenty of loggable events and possible errors happened during the time frame, whereas high number of unique jobs combined to low number of rows imply that not much happened (or perhaps executions were completely crashed).

By the original hypothesis, anomalies in the logs are linked closely to the support tickets received. The anomaly detection algorithm produces a probability value of how strongly a row is considered to be an anomaly, thus, the mean and median values of the anomaly probabilities in a time frame indicate how anomalous all the executions within a time frame have statistically been. However, when compressing the anomaly probability metrics in a time frame, some information is bound to be lost. The mean and median values do not provide information about the anomaly probability value distribution. There may be few very high values and a lot of low probabilities, and it would lead to the same mean value as if there were a lot of slightly higher probabilities and just few very low values. By adding more statistical values we can reduce the loss of information caused by time frame compression. If the original hypothesis is correct, the most relevant anomaly values are the highest anomaly probabilities. Thus, calculating higher quantiles and the amount of values exceeding them should improve the estimation abilities of the algorithm.

In table 7, all the statistic metrics considered interesting and valuable from time frame compression of the log rows are listed with short explanation. In the pipeline, the statistical values are calculated using the R-script executing component. An example of such script executed by the “Execute R Script” component is presented in appendix E.

Statistic feature	**Explanation (in time frame)**
LogRowCount	Number of rows/instances overall
UniqueJobIDs	Amount of unique job IDs
AnomalyProbabilityMean	Mean value of anomaly probabilities
AnomalyProbabilityMedian	Median value of anomaly probabilities
AnomalyQuantile90	90-quantile value of anomaly probabilities
AnomalyCountOverQ90	Number of instances with anomaly probability over 90-quantile

Table 7: Statistic metrics of the time frame compression that are considered possibly useful for ML algorithm.

The result values of the anomaly detection algorithm are time-frame-compressed along with the technical ticket timestamps, which form our second part of the data. The timestamp data, however, gets compressed simply into a count of tickets received in the defined time frame. Now we have a comparable feature that is usable by regression algorithms.

Regression algorithm options

Azure ML Studio has six regression algorithms usable. One of them, *Fast Forest Quantile Regression*, is used to explain the distribution of value being predicted.[54] In our case, however, this is does not provide any use for us as we are predicting the ticket count per time frame. The distribution of tickets on each time frame is irrelevant.

The remaining five algorithms are presented below:

1. Linear regression [55]
2. Boosted decision tree regression [56]
3. Decision forest regression [57]
4. Neural network regression [58]
5. Poisson regression [59]

Linear regression is the simplest one, and even though it can be easily explained with the least square principle explained in the section 2.3, linear regression component is capable of more advanced methods, such as gradient descent. Poisson regression works only for Poisson distributed data. As the distribution of the anomalies or ticket inducing log events is not certain, we cannot rule this method out before seeing the results.

Other components with a *tree* or *forest* in their name are based on the same algorithm model called *decision trees*. Decision trees are models which run a series of simple tests for all data points in each branching node. The data is moved on like water along the tree from trunk to branches until it reaches the leaf node which marks the decision-making point. When multiple trees are placed in serial, the decision tree becomes a forest. [57]

When decision searching tree branches do not trickle from few to many but the flow can contract and widen again, or multiple nodes can reach same child nodes, the model becomes a neural network.

All of these regression algorithm models were tested and their results validated with new data in order to find the one with the most promising results.

4.4 Pipeline branching

In order to find the best possible combination of components, all different combinations must be compared. As the pipeline is structured in a tree-like flow, each node which includes more than one possible choices diverges the pipeline into branches. Several diverging points have been mentioned before in this study, and in this section we join the information together.

First, the error message used to calculate the anomaly probability of a log row had two options. We could either use simple *message*, or more verbose *rawmessage*. This textual data could be fed to the ADA-component in several forms. Most straightforward way was using textual data without any preformatting or modification. Text could also be run through "Preprocess text" -component. N-gram features could have been extracted from the original or preprocessed text and these features could have been used instead. Instead of n-gram features, textual data could be converted to numeric also with "Feature Hashing" -component. After getting the ADA-component results, the anomaly probabilities were compressed with R-code or SQL. This concludes the phase 1.

In the next phase, branching of the pipeline was due to comparing results without the anomaly probability values calculated in phase 1, and then utilizing different regression algorithms in phase 2. In practice this means that in order to validate the results against our initial hypothesis, we used pure statistical log data, such as row count and unique job ID count without anomaly probabilities, to determine whether anomaly metrics provided any insight regarding the ticket data. When using n-gram features or hashed features, these additional column features were also included for this comparison as forming them does not depend on the anomaly properties of the log instance.

Each branching step, or layer, multiplies the amount of comparable values used in final comparison that would determine the best possible pipeline combination. These layers are simplified in the table 8.

The divergent count implies the number of branches diverging from the previous component. The total count of branch ends, or leaves, would then be the multiplication of all divergent counts, totaling to 240 comparable pipeline combinations. Moreover, n-gram feature extraction and feature hashing have several tunable parameters that strongly influence the end results of the algorithm training. To reduce this amount when considering the best possible pipeline, we simplified this by narrowing down the options based on initial test run results of some of the divergent options.

For example, n-gram feature component suffered greatly from the memory problem

Branching node	**Options**	**Divergent count**
Input text column	message rawmessage	2
Text preprocess	Yes No	2
Numeric conversion	No N-gram Feature Feature Hashing	3
ADA training	Unconventional Proper	2
Validation without anomaly metrics	Yes No	2
Regression algorithms	Linear regression Boosted decision tree regression Decision forest regression Neural Network Regression Poisson regression	5

Table 8: Pipeline divergent layers

(which we discuss more in section 5.1), and the data amount that the *Extract N-Gram Features from Text* -component was able to handle comprised of only 2% of the original data. This was deemed as too small amount for training an ML algorithm as it would be extremely likely with 98% of the data skipped that also possible rows relevant to the ticket anomalies would get trimmed out too.

The flowchart of pipeline with branching options visualized is illustrated in figure 11. The final pipeline structure used for result acquiring can be seen in the appendix A.

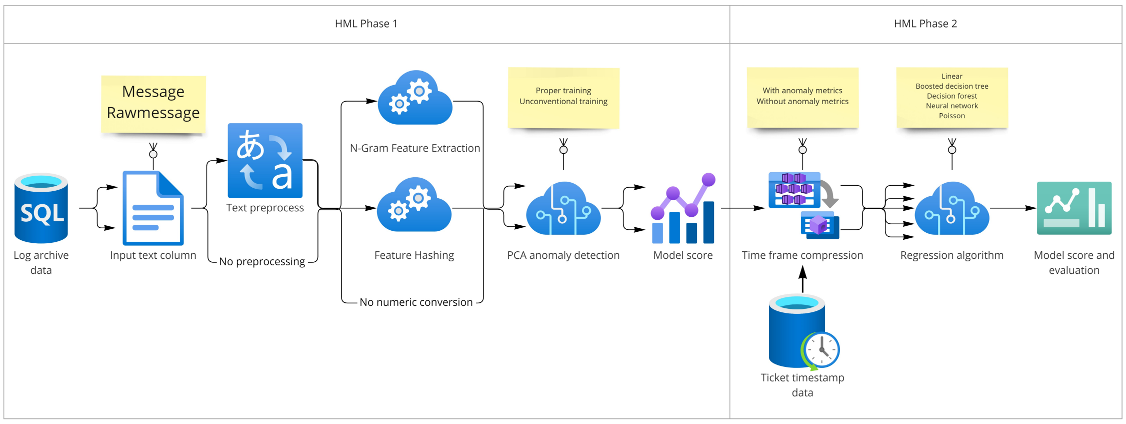


Figure 11: Pipeline flowchart with branching steps illustrated. HML phase boundaries are made visible for clarity.

4.5 Comparable metrics

In previous sections we have discussed of different approaches that could be used to get the best results from the ML training. In order to find the best possible combination of components, we must compare different results, and for this we need comparable metrics. After training, Azure regression algorithms are scored with a validation data. The final output from this is a numeric *Scored Labels* feature. In our study case, this value is what algorithm estimates the ticket count on the time frame to be. Next, the *Evaluate Model* component calculates a few evaluation metrics to be used for result comparison. These metrics for regression algorithm are listed in table 9.

Name of the metric	**Explanation**
Mean Absolute Error (MAE)	Measures how close the predictions are to the actual outcomes; thus, a lower score is better.
Root Mean Squared Error (RMSE)	Creates a single value that summarizes the error in the model. By squaring the difference, the metric disregards the difference between over-prediction and under-prediction.
Relative Squared Error (RSE)	Normalizes the total squared error of the predicted values by dividing by the total squared error of the actual values.
Relative Absolute Error (RAE)	The relative absolute difference between expected and actual values; relative because the mean difference is divided by the arithmetic mean.
Coefficient of Determination (CoD)	Often referred to as R^2, represents the predictive power of the model as a value between 0 and 1. Zero means the model is random (explains nothing); 1 means there is a perfect fit. However, caution should be used in interpreting R^2 values, as low values can be entirely normal and high values can be suspect.

Table 9: Comparable metrics provided by *Evaluate Model* component for regression algorithm.[60]

Our goal is to find a connection between the log anomalies and the ticket timestamps. Our initial hypothesis suggested that the most anomalous events in the logs would be more related to the ticket events than other rows. Thus, by comparing the algorithm evaluation metrics to the metrics produced by the neighboring pipeline branch where anomaly probabilities has been removed, we should find the best algorithm where evaluation values are significantly better than the same algorithm without those anomaly values. This would prove, that the anomaly probability values improve the algorithm estimations indicating that the hybrid algorithm combination has indeed found the connection between anomalies and ticket timestamps. This will also be our logical basis when selecting certain algorithms for further tests.

5 Results

This section presents the results of HML pipeline combinations. The values of algorithm evaluations are compared and the results are judged. Before that, we discuss about the memory issue encountered that affected our pipeline component choices.

After presenting the results, the sensibility of the outcome is evaluated and suggestions for improvement are made.

5.1 Memory issues and limitations

Memory is crucial resource in ML training. Algorithms take multiple steps while iterating the data and intermediate results are stored in the RAM rather than on the disk. While building ML pipeline in Azure ML Studio, a memory issue emerged that affected several components and caused serious limitations in terms of usable components and data size. Due to the time limits of this study, this issue was not resolved and the problem causing it was not found. As several conditions considering the environment costs were already issued by the company, the issue was supposed to be linked with compute instance property limitations and thus could not be resolved without cost overruns. However, this was not certain.

In order to complete some of the ML pipeline runs the data amount had to be reduced to 600 megabytes. This was considerably less than what was assumed to be sufficient for reliable results, as this size of data would cover for just a few months of logging. To increase the time span of log data, it was decided to trim info-type log messages from the data. This way we were able to reduce the data to 8.6 million log rows which was about 10% of the original data size. Before final cleaning operations, the data took 8.1GB of disk space, and after cleaning the rawmessage-field with the script in appendix D, the final size in disk was 6.6GB. Even with this data size, some Azure ML components still faced this memory issue and forced us to choose such components that were able to handle these data amounts.

One of the most sensitive components for this issue was n-gram feature extraction module. As we explained in section 2.6, each new n-gram or word creates a new column in the data. This module suffered greatly from the memory exhaustion, especially while processing rawmessage which it included more data in the JSON structure than pure message-field. With rawmessage, n-gram feature extraction was able to handle less than 1% of the data amount, resulting to more than 1000 columns of n-gram feature columns. With message data, this situation was just slightly better, as the limit of data amount was in 2%.

Feature hashing component was able to process more data than pure n-gram feature extracting module, but this required the compression level to be quite high. The main parameter to tune in feature hashing was the bitsize of the hashing. The smaller the hashing bitsize was, the greater the level of compression was thus resulting to more lost information. In order to utilize this module at all, the hashing bitsize had to be set to 7 which produced approximately 130 hashed feature columns.

With some component combinations, additional data split modules were needed in order to finish the ML pipeline. For example, with rawmessage-data, feature hashing and unconventional training, the validation data had to be trimmed to 70% so that the model scoring could be executed.

5.2 Algorithm estimation results

Based on the first pipeline experiments, the Decision Forest Regression algorithm gave the most promising results, so the values listed in this section are based on the use of this algorithm unless otherwise stated. However, this first impression was a misinterpretation, as the final results show. For unknown reasons, when time frame compression was carried out with SQL queries instead of R-script, the results were considerably better. However, these results could not be relied upon, as further investigation showed that the time frame compression was incorrect. This was verified by calculating the total number of log rows and comparing it to the sum of the line counts in all time frames. As something did not work as intended with SQL queries, SQL-based time frame compression was decided to reject. It was also much easier to include more statistical metrics with R-scripting than with SQL queries.

As explained in the section 4.4, the amount of pipeline component combinations to compare was significant. It was decided to run initial experiments with a few basic component combinations and select the most promising branches to continue with. For this reason, it was decided to reject pure n-gram feature extraction in the final analysis because the memory issue affected so much its reliability in terms of the amount of data. Feature hashing, however, could be included. The amount of data was seen sufficient to consider the results reliable, although, to make this possible, a lot of information was lost due to the level of compression used.

In the table 10, the final results of the pipelines using Decision Forest Regression algorithm in phase 2 are listed for comparison. When interpreting the comparison metrics, it was decided to focus on the coefficient of determination (CoD-value), mean absolute error (MAE-value), and their difference with the corresponding metrics in cases where the anomaly probability value was omitted from phase 2. Like we explained in the section 4.4, we assumed that anomaly probability values calculated by the anomaly detection algorithm provides valuable information for ticket count estimating. Thus, if these values are removed before the phase 2 algorithm training, the estimation capabilities of the algorithm should weaken. Hence, we decided to experiment more with those pipeline component combinations which resulted in noticeably better results than their anomaly-probability-omitted counterparts. As mentioned in the previous section, with MAE smaller is better, whereas with CoD bigger is better.

As highlighted in the table10, the most promising results have been received by using:

- message data, feature hashing and proper training
- rawmessage data and unconventional training
- rawmessage data and proper training
- rawmessage data, preprocessed, proper training, and

- rawmessage data, preprocessed, unconventional training.

Next, these chosen combinations are tested with different regression algorithms in the phase 2.

In the table 11, we can see the results of different regression algorithms with message data and feature hashing. However, as the results indicate, comparison metrics do not improve with different algorithms. Thus, Decision Forest Regression is selected for the final validation with fresh test data.

With rawmessage, we were unable to execute tests with fresh data because the model scoring module failed with an error, stating that *"Number of unique values in column: 'rawmessage' is greater than allowed"*. This happened even though the rawmessage field was cleaned from the unique data values such as timestamp and fingerprint. For this reason, we could not acquire test results of pipeline component combinations that did not include rawmessage preprocessing. With text preprocessing, this issue did not occur.

As seen in tables 12 and 13, Decision Forest regression gives the most promising results when using rawmessage-column, with or without text preprocessing. Thus, the initial component combinations were selected to further experimentation.

The final testing was executed with freshly acquired production data. In the end, only three most promising pipeline component combinations got tested in this phase. The results are listed in the table 14. As seen from the final comparison metrics, the results were not convincing. Coefficient of determination was a negative number in each case, meaning that our ML model did not find any sensible relationship between log data and ticket timestamps. The negative value means also, that even a random model would have performed better in ticket count estimation than our trained algorithms. Further examination of the score values behind these results reveal a possible reason for this. In case of preprocessed rawmessage, proper training and Decision Forest algorithm, the MAE-value was 0.5277 and CoD-value 0.79976. Without context, these values would have been excellent, meaning that an average error in ticket count estimations would have been just a little over 0.5 in time frame. However, because the memory issue forced us to trim the data amount, and the time frame compression condensed the training data in phase 2 even more, the final evaluation was executed with just 3 rows of data. Obviously, this is not a sufficient amount of data to get reliable results.

After considering the results of all the results, it was determined that our initial goal to find a connection between log anomalies and technical ticket timestamps did not succeed. Even though we were not able to find the connection with the methods used in this study, it should not be ruled out that this connection does not exist. Our final verdict is that further studying is needed.

5.3 Improvement discussion

This research encountered several problems that could not be anticipated beforehand. Most notable of the problems was the memory issue with Azure ML modules. This, along with the bureaucratic challenges regarding the data security, forced us to limit the study scope in regards to the schedule of the thesis. Thus, several possibilities could not be examined thoroughly.

In order to use the results of this study in production, the ML pipelines with trained algorithms should be published as online endpoints. The data in production needed a great deal of processing before it could be exported to the cloud environment. To be able to use live production data for real time ticket estimation, a solution for data processing and exporting would have been needed.

The data format proved challenging, not only because of the mixed type of data that included both CSV and JSON, but also because of multiple different types of sensitive information that had to be taken into account. With more simple and consistent log formatting, further insight of the data could have been acquired. One major improvement could be the possibility to utilize One-Class Support Vector Machine. This, however, would have needed a manually constructed dataset including only “normal” log events, or such events that can be reasonably assured not to be related to the ticket inducing issues. With better preformatted data, more informative log metrics could be found which may prove useful for algorithm estimations.

Time frame compression was inevitably losing some information. We attempted to minimize the drawbacks by calculating multiple statistical values along with the compression. Different approaches could still have been experienced with. One way would be calculating anomaly probability metrics per job ID to increase the understanding of anomaly distribution inside time frames. Due to the working week effects on the tickets, shifting the time frames from Monday-Sunday to Saturday-Friday could also change the results, as it is presumable that tickets received early on Monday are related to issues on weekend.

The original hypothesis was that anomalous events in the logs were clearly linked to the tickets received. However, as memory errors due to the size of data forced us to skip info-typed rows, it is possible the data anomalies did not reflect to the tickets. As stated before, multiple error lines in the log may be linked to a single issue, which could make the ticket inducing events more common log feature. Thus, it could have been possible to get better results by tuning the statistical values used, so that more common error messages would have been taken into account.

In this study, we decided to skip some corners and selected pipeline component combinations to continue with based on the initial results of Decision Forest regression algorithm. Other algorithms were tested with the best combinations that worked on Decision Forest regression. It is possible that with different algorithms, other component combinations would have worked better. Also, all of the algorithms have multiple tunable parameters that affect greatly on the results. Most of the algorithms, however, were used with default parameters to save time and computing costs. Model hyperparameter tuning and parameter combination testing could still provide further results.

Finally, without the memory issue limiting our data amount and component selection, n-gram feature extraction could have been utilized in full extent. Other data preprocessing methods were bound to lose valuable information, and anomaly detection from the textual features in this study case was not possible with the algorithms available.

Component combinations with Decision Forest regression					
Combination	**MAE**	**RMSE**	**RSE**	**RAE**	**CoD**
Msg PropT	3.6875	4.2886	1.6966	1.3169	-0.6966
Msg PropT NA	3.5125	4.2718	1.6834	1.2544	-0.6834
Msg UnconT	3.5236	4.3578	1.3493	1.1313	-0.3493
Msg UnconT NA	4.0405	4.8958	1.7030	1.2972	-0.7030
PreP.Msg PropT	3.25	3.7720	1.3125	1.1607	-0.3125
PreP.Msg PropT NA	3.5125	4.2718	1.6834	1.2544	-0.6834
PreP.Msg UnconT	3.7905	4.5707	1.4843	1.2169	-0.4843
PreP.Msg UnconT NA	4.0405	4.8958	1.7030	1.2972	-0.7030
Msg FHash PropT	2.4875	2.8611	0.7551	0.8883	**0.24484**
Msg FHash PropT NA	2.85	3.1922	0.9400	1.0178	**0.05990**
Msg FHash UnconT	2.6071	3.2484	0.7234	0.8488	0.27657
Msg FHash UnconT NA	2.6562	3.2593	0.7282	0.8648	0.27171
PreP.Msg FHash PropT	3.05	3.4058	1.0701	1.0892	-0.0701
PreP.Msg FHash PropT NA	2.6	2.8858	0.7682	0.9285	0.23172
PreP.Msg FHash UnconT	2.7901	3.4467	0.8144	0.908	0.18556
PreP.Msg FHash UnconT NA	2.8616	3.4125	0.7983	0.9316	0.20162
RawMsg PropT	**1.3611**	1.8263	1.6678	1.0208	-0.6678
RawMsg PropT NA	**2.5833**	2.8694	4.1168	1.9375	-3.1168
RawMsg UnconT	2.9	3.8200	0.7859	0.7928	**0.21402**
RawMsg UnconT NA	3.0733	3.9588	0.8441	0.8402	**0.15588**
PreP.RawMsg PropT	**0.5277**	0.6328	0.2002	0.3958	**0.79976**
PreP.RawMsg PropT NA	**2.5833**	2.8694	4.1168	1.9375	**-3.1168**
PreP.RawMsg UnconT	2.9466	3.7159	0.7437	0.8056	**0.25627**
PreP.RawMsg UnconT NA	3.0733	3.9588	0.8441	0.8402	**0.15588**
RawMsg FHash PropT	3.7916	5.3078	2.6361	1.3787	-1.6361
RawMsg FHash PropT NA	4.0625	5.2026	2.5326	1.4772	-1.5326
RawMsg FHash UnconT	2.875	3.3994	1.0458	1.0267	-0.0458
RawMsg FHash UnconT NA	2.9	3.8249	1.3240	1.0357	-0.3240
PreP.RawMsg FHash PropT	2.8194	2.8993	1.4834	1.4097	-0.4834
PreP.RawMsg FHash PropT NA	2.5694	3.0503	1.6419	1.2847	-0.6419
PreP.RawMsg FHash UnconT	2.6153	3.4303	0.8327	0.9324	0.16723
PreP.RawMsg FHash UnconT NA	2.5384	3.1784	0.7149	0.9050	0.28503

Table 10: Results of HML pipeline with Decision Forest regression algorithm in the phase 2. **FHash** means *Feature Hashing*, **PropT** indicates *proper training*, **UnconT** that *unconventional training* is done in phase 1, **PreP.** means that *text preprocessing* has been used, and **NA** means that *anomaly values has been removed* for comparison (NoAnomalies). The most promising comparison metrics and their component combinations are bolded.

Message with feature hashing					
Algorithm	**MAE**	**RMSE**	**RSE**	**RAE**	**CoD**
Poisson PropT	3.1045	3.4232	1.0810	1.1087	-0.0810
Poisson PropT NA	3.0647	3.3359	1.0266	1.0945	-0.0266
Poisson UnconT	2.7530	3.4857	0.8329	0.8963	0.16704
Poisson UnconT NA	2.7412	3.3489	0.7688	0.8924	0.23113
NeuralNet PropT	2.9549	3.5045	1.1330	1.0553	-0.1330
NeuralNet PropT NA	2.9368	3.4668	1.1087	1.0488	-0.1087
NeuralNet UnconT	3.629	4.6548	1.4854	1.1817	-0.4854
NeuralNet UnconT NA	3.7330	4.7398	1.5401	1.2154	-0.5401
Boosted PropT	2.2013	2.8158	0.7314	0.7861	0.26852
Boosted PropT NA	2.1983	2.7899	0.7180	0.7851	0.28193
Boosted UnconT	3.0881	3.7033	0.9402	1.0054	0.05976
Boosted UnconT NA	2.9448	3.4691	0.8250	0.9587	0.17494
Linear PropT	2.9491	3.1988	0.9439	1.0532	0.05602
Linear PropT NA	2.8327	3.0956	0.8840	1.0117	0.11597
Linear UnconT	2.8672	3.5982	0.8876	0.9335	0.11238
Linear UnconT NA	2.6735	3.3376	0.7637	0.8704	0.23628
DecFor PropT	2.4875	2.8611	0.7551	0.8883	**0.24484**
DecFor PropT NA	2.85	3.1922	0.9400	1.0178	**0.05990**
DecFor UnconT	3.5236	4.3578	1.3493	1.1313	-0.3493
DecFor UnconT NA	4.0405	4.8958	1.7030	1.2972	-0.7030

Table 11: Results of HML pipeline with different algorithms used in phase 2. Feature hashing has been used with **message**-column in each case. **Poisson** means *Poisson regression*, **NeuralNet** indicates *Neural Network regression*, **Boosted** means *Boosted Decision Tree regression*, **Linear** means *Linear regression*, and **DecFor** means *Decision Forest regression*. Each algorithm is tested with unconventional (**UnconT**) vs. proper training (**PropT**), and with or without anomaly probability values from phase 1 (**NA** means NoAnomalies). The most promising results are bolded.

Pure rawmessage					
Algorithm	**MAE**	**RMSE**	**RSE**	**RAE**	**CoD**
Poisson PropT	2.7528	2.9170	1.0348	1.1261	-0.0348
Poisson PropT NA	2.7540	2.9185	1.0359	1.1266	-0.0359
Poisson UnconT	2.2626	2.9650	0.8661	0.8927	0.13385
Poisson UnconT NA	2.3380	2.9977	0.8853	0.9225	0.11464
NeuralNet PropT	2.3926	2.5209	0.7729	0.9788	0.22704
NeuralNet PropT NA	2.3468	2.3981	0.6994	0.9600	0.30056
NeuralNet UnconT	2.8173	3.5439	1.2373	1.1116	-0.2373
NeuralNet UnconT NA	2.3270	2.9708	0.8694	0.9181	0.13050
Boosted PropT	2.75	3.0103	1.1021	1.125	-0.1021
Boosted PropT NA	2.75	3.0103	1.1021	1.125	-0.1021
Boosted UnconT	2.8929	3.6821	1.3357	1.1414	-0.3357
Boosted UnconT NA	2.7651	3.5996	1.2765	1.0910	-0.2765
Linear PropT	2.6982	2.8526	0.9896	1.1038	0.01030
Linear PropT NA	2.6796	2.8188	0.9663	1.0962	0.03361
Linear UnconT	2.4777	3.0516	0.9174	0.9776	0.08254
Linear UnconT NA	2.3310	2.9720	0.8702	0.9197	0.12976
DecFor UnconT	2.9	3.8200	0.7859	0.7928	**0.21402**
DecFor UnconT NA	3.0733	3.9588	0.8441	0.8402	**0.15588**
DecFor PropT	**1.3611**	1.8263	1.6678	1.0208	-0.6678
DecFor PropT NA	**2.5833**	2.8694	4.1168	1.9375	-3.1168

Table 12: Results of HML pipeline with different algorithms used in phase 2. Each case uses pure **rawmessage**-column without preprocessing. **Poisson** means *Poisson regression*, **NeuralNet** indicates *Neural Network regression*, **Boosted** means *Boosted Decision Tree regression*, **Linear** means *Linear regression*, and **DecFor** means *Decision Forest regression*. Each algorithm is tested with unconventional (**UnconT**) vs. proper training (**PropT**), and with or without anomaly probability values from phase 1 (**NA** means NoAnomalies). The most promising results are bolded.

Preprocessed rawmessage					
Algorithm	**MAE**	**RMSE**	**RSE**	**RAE**	**CoD**
Poisson PropT	2.7540	2.9186	1.0360	1.1266	-0.0360
Poisson PropT NA	2.7540	2.9185	1.0359	1.1266	-0.0359
Poisson UnconT	2.3714	3.0466	0.9144	0.9357	0.08556
Poisson UnconT NA	2.3380	2.9977	0.8853	0.9225	0.11464
NeuralNet PropT	2.5653	2.6747	0.8701	1.0494	0.12986
NeuralNet PropT NA	2.3468	2.3981	0.6994	0.9600	0.30056
NeuralNet UnconT	2.7686	3.4396	1.1656	1.0924	-0.1656
NeuralNet UnconT NA	2.3270	2.9708	0.8694	0.9181	0.13050
Boosted PropT	2.75	3.0103	1.1021	1.125	-0.1021
Boosted PropT NA	2.75	3.0103	1.1021	1.125	-0.1021
Boosted UnconT	2.9566	3.6357	1.3022	1.1666	-0.3022
Boosted UnconT NA	2.7651	3.5996	1.2765	1.0910	-0.2765
Linear PropT	2.6992	2.8546	0.9910	1.1042	0.00890
Linear PropT NA	2.6796	2.8188	0.9663	1.0962	0.03361
Linear UnconT	2.2808	2.9685	0.8681	0.8999	0.13182
Linear UnconT NA	2.3310	2.9720	0.8702	0.9197	0.12976
DecFor PropT	**0.5277**	0.6328	0.2002	0.3958	**0.79976**
DecFor PropT NA	**2.5833**	2.8694	4.1168	1.9375	**-3.1168**
DecFor UnconT	2.9466	3.7159	0.7437	0.8056	**0.25627**
DecFor UnconT NA	3.0733	3.9588	0.8441	0.8402	**0.15588**

Table 13: Results of HML pipeline with different algorithms used in phase 2. Each case uses preprocessed **rawmessage**-column. **Poisson** means *Poisson regression*, **NeuralNet** indicates *Neural Network regression*, **Boosted** means *Boosted Decision Tree regression*, **Linear** means *Linear regression*, and **DecFor** means *Decision Forest regression*. Each algorithm is tested with unconventional (**UnconT**) vs. proper training (**PropT**), and with or without anomaly probability values from phase 1 (**NA** means NoAnomalies). The most promising results are bolded.

Results with fresh test data					
Algorithm	**MAE**	**RMSE**	**RSE**	**RAE**	**CoD**
Decision Forest, message, feature hashing, proper training	1.7333	2.2899	1.2341	1.1470	-0.2341
Decision Forest, preprocessed rawmessage, unconventional training	3.4888	3.8921	3.5654	2.3088	-2.5654
Decision Forest, preprocessed rawmessage, proper training	4.25	4.6840	5.1637	2.8125	-4.1637

Table 14: Final HML results with fresh test data. CoD-value in each case is negative, which means that the estimation power of the trained algorithms is weaker than random. Thus, these component combinations are not able to find the connection we were aiming for.

6 Summary

This thesis study investigated possibilities to utilize machine learning in order to predict technical ticket arrival based on RPA log events and ticket timestamps using anomaly detection. Several steps were required before the actual algorithm training could be initialized.

First, the data had to be anonymized. We compared the possible outcome of full anonymization of sensitive data with pseudonymization and k-anonymization. Data security restrictions and the probable type of log events inducing technical tickets settled us to choose full anonymization, which was carried out with PowerShell script in production environment.

Next, local data preprocessing was needed before the data was deemed suitable for ML training. As data consisted of miscellaneous types of input, we had to create another script to reformat the data in a consistent form.

After this, the proper ML training steps in Azure ML Studio environment could begin. One of the main problems we had to solve was how to combine log events with simple timestamp data of the technical tickets with an unknown span of random delay. We decided to use hybrid machine learning fused with time frame compression method to avoid the random delay issue. This, however, leaned heavily on the hypothesis that most anomalous log events are related to technical tickets received. This hypothesis had to be validated by comparing the results of the final algorithm training with an ML pipeline branch where anomaly values had been removed.

In order to find the best possible combination of ML components for hybrid machine learning both in terms of suitable algorithms and input data features, several combinations were tested and compared with each other. In the end, however, it became clear that either our original hypothesis had to be assumed incorrect and anomalous events are not linked to technical support tickets, or more studying is required to find the connection between the log events and ticket timestamps.

References

[1] P. K. Donepudi, “Machine learning and artificial intelligence in banking,” *Engineering International*, vol. 5, no. 2, pp. 83–86, 2017.

[2] W. M. Van der Aalst, M. Bichler, and A. Heinzl, “Robotic process automation,” pp. 269–272, 2018.

[3] A. DeLaRosa, “Log monitoring: not the ugly sister,” *Pandora FMS*, 2018. [Online]. Available: https://web.archive.org/web/20210901031146/https://pandorafms.com/blog/log-monitoring/

[4] W. K. Ho, B.-S. Tang, and S. W. Wong, “Predicting property prices with machine learning algorithms,” *Journal of Property Research*, vol. 38, no. 1, pp. 48–70, 2021. [Online]. Available: https://doi.org/10.1080/09599916.2020.1832558

[5] K. Ghanem, F. J. Aparicio-Navarro, K. G. Kyriakopoulos, S. Lambotharan, and J. A. Chambers, “Support vector machine for network intrusion and cyber-attack detection,” in *2017 Sensor Signal Processing for Defence Conference (SSPD)*, 2017, pp. 1–5.

[6] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[7] “Definition of algorithm,” accessed: 2022-05-19. [Online]. Available: https://web.archive.org/web/20220510183749/https://www.merriam-webster.com/dictionary/algorithm

[8] T. O. Ayodele, “Types of machine learning algorithms,” *New advances in machine learning*, vol. 3, pp. 19–48, 2010.

[9] B. Mahesh, “Machine learning algorithms-a review,” *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[10] R. Vickery, “Beginners guide to the three types of machine learning,” 2019, accessed: 2022-07-12. [Online]. Available: https://web.archive.org/web/20220712170551/https://towardsdatascience.com/beginners-guide-to-the-three-types-of-machine-learning-3141730ef45d?gi=9db5aaf56001

[11] X. Chen, Y. Fang, M. Yang, F. Nie, Z. Zhao, and J. Z. Huang, “Purtreeclust: A clustering algorithm for customer segmentation from massive customer transaction data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 559–572, 2017.

[12] H. Li, “Which machine learning algorithm should i use?” 2017, accessed: 2022-07-13. [Online]. Available: https://web.archive.org/web/20220402120055/https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/

[13] P. Baheti, "Train test validation split: How to and best practices 2022," 2022, accessed: 2022-07-14. [Online]. Available: https://web.archive.org/web/20220714125714/https://www.v7labs.com/blog/train-validation-test-set

[14] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[15] "Comparing machine learning as a service: Amazon, microsoft azure, google cloud ai, ibm watson," accessed: 2022-04-04. [Online]. Available: https://web.archive.org/web/20220714131635/https://www.altexsoft.com/blog/datascience/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai-ibm-watson/

[16] "Azure machine learning - ml as a service - microsoft azure 2022," 2022, accessed: 2022-07-12. [Online]. Available: https://web.archive.org/web/20220714154625/https://azure.microsoft.com/en-us/services/machine-learning

[17] "Definition of regression," accessed: 2022-07-19. [Online]. Available: https://web.archive.org/web/20220719100911/https://www.merriam-webster.com/dictionary/regression

[18] R. J. Freund, W. J. Wilson, and P. Sa, *Regression analysis.* Elsevier, 2006.

[19] F. Stulp and O. Sigaud, "Many regression algorithms, one unified model: A review," *Neural Networks*, vol. 69, pp. 60–79, 2015.

[20] "Pca-based anomaly detection: component reference - azure machine learning," 2022, accessed: 2022-07-10. [Online]. Available: https://web.archive.org/web/20220710090015/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/pca-based-anomaly-detection

[21] L. I. Smith, "A tutorial on principal components analysis," University of Otago, Tech. Rep. OUCS-2002-12, 2002, available: http://hdl.handle.net/10523/7534.

[22] S. Börm and C. Mehl, "Numerical methods for eigenvalue problems," in *Numerical Methods for Eigenvalue Problems.* de Gruyter, 2012, pp. 9–10.

[23] Z. Jun, "Unsupervised learning (ii) dimension reduction," October 2019. [Online]. Available: https://web.archive.org/web/20220721113413/https://thudm.github.io/Tsinghua-ML-Course/slides/5-Dimension%20reduction.pdf

[24] Z. Jaadi, "A step-by-step explanation of principal component analysis (pca)," 2021. [Online]. Available: https://web.archive.org/web/20220720181640/https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[25] S. M. Holland, "Principal components analysis (pca)," *Department of Geology, University of Georgia, Athens, GA*, pp. 30 602–2501, 2008.

[26] "One-class support vector machine component reference - azure machine learning 2021," 2019, accessed: 2022-07-19. [Online]. Available: https://web.archive.org/web/20220719151913/https://docs.microsoft.com/en-us/previous-versions/azure/machine-learning/studio-module-reference/one-class-support-vector-machine

[27] J. Fürnkranz, "A study using n-gram features for text categorization," *Austrian Research Institute for Artifical Intelligence*, vol. 3, no. 1998, pp. 1–10, 1998.

[28] "Feature hashing component reference - azure machine learning," 2021, accessed: 2022-07-21. [Online]. Available: https://web.archive.org/web/20220721142304/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/feature-hashing

[29] C. Caragea, A. Silvescu, and P. Mitra, "Protein sequence classification using feature hashing," in *Proteome science*, vol. 10, no. 1. Springer, 2012, pp. 1–8.

[30] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan, "Hash kernels for structured data." *Journal of Machine Learning Research*, vol. 10, no. 11, 2009.

[31] A. M. Tripathi, *Learning Robotic Process Automation: Create Software robots and automate business processes with the leading RPA tool–UiPath*. Packt Publishing Ltd, 2018.

[32] R. Noumeir, A. Lemay, and J.-M. Lina, "Pseudonymization of radiology data for research purposes," *Journal of digital imaging*, vol. 20, no. 3, pp. 284–295, 2007.

[33] J.-W. Byun, A. Kamra, E. Bertino, and N. Li, "Efficient k-anonymization using clustering techniques," in *International Conference on Database Systems for Advanced Applications*. Springer, 2007, pp. 188–200.

[34] A. Rantala, "Applying machine learning to automatic incident detection from software log output," Master's thesis, Aalto-yliopisto, Sähkötekniikan korkeakoulu, 2019, available: http://urn.fi/URN:NBN:fi:aalto-201906234018.

[35] S. Allagi and R. Rachh, "Analysis of network log data using machine learning," in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. IEEE, 2019, pp. 1–3.

[36] N. Kondo, M. Okubo, and T. Hatanaka, "Early detection of at-risk students using machine learning based on lms log data," in *2017 6th IIAI international congress on advanced applied informatics (IIAI-AAI)*. IEEE, 2017, pp. 198–201.

[37] Q. Cao, Y. Qiao, and Z. Lyu, "Machine learning to detect anomalies in web log analysis," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2017, pp. 519–523.

[38] D. Liu, “Log analysis for anomaly detection,” 2019, accessed: 2022-07-15. [Online]. Available: https://web.archive.org/web/20220715102721/https://davideliu.com/2019/10/26/log-analysis-for-anomaly-detection/

[39] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, “Robust log-based anomaly detection on unstable log data,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[40] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Jagadish, “Regular expression learning for information extraction,” in *Proceedings of the 2008 conference on empirical methods in natural language processing*, 2008, pp. 21–30.

[41] C.-F. Tsai and M.-L. Chen, “Credit rating by hybrid machine learning techniques,” *Applied soft computing*, vol. 10, no. 2, pp. 374–380, 2010.

[42] F. Anifowose, “Hybrid machine learning explained in nontechnical terms,” *JPT*, 2020. [Online]. Available: https://web.archive.org/web/20220612142143/https://jpt.spe.org/hybrid-machine-learning-explained-nontechnical-terms

[43] T. Shon and J. Moon, “A hybrid machine learning approach to network anomaly detection,” *Information Sciences*, vol. 177, no. 18, pp. 3799–3821, 2007.

[44] S. Mohan, C. Thirumalai, and G. Srivastava, “Effective heart disease prediction using hybrid machine learning techniques,” *IEEE access*, vol. 7, pp. 81 542–81 554, 2019.

[45] N.-C. Hsieh, “Hybrid mining approach in the design of credit scoring models,” *Expert Systems with Applications*, vol. 28, no. 4, pp. 655–665, 2005.

[46] A. Jain and A. M. Kumar, “Hybrid neural network models for hydrologic time series forecasting,” *Applied Soft Computing*, vol. 7, no. 2, pp. 585–592, 2007.

[47] H.-j. Kim and K.-s. Shin, “A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets,” *Applied Soft Computing*, vol. 7, no. 2, pp. 569–576, 2007.

[48] T.-S. Lee, C.-C. Chiu, C.-J. Lu, and I.-F. Chen, “Credit scoring using the hybrid neural discriminant technique,” *Expert Systems with applications*, vol. 23, no. 3, pp. 245–254, 2002.

[49] R. Malhotra and D. Malhotra, “Differentiating between good credits and bad credits using neuro-fuzzy systems,” *European journal of operational research*, vol. 136, no. 1, pp. 190–211, 2002.

[50] "Extract n-gram features from text component reference - azure machine learning 2021," 2021, accessed: 2022-07-10. [Online]. Available: https://web.archive.org/web/20220717165959/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/extract-n-gram-features-from-text

[51] "Preprocess text component - azure machine learning," 2021, accessed: 2022-07-24. [Online]. Available: https://web.archive.org/web/20220724154720/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/preprocess-text

[52] H. Wang, Z. Lei, X. Zhang, B. Zhou, and J. Peng, "Machine learning basics," *Deep learning*, pp. 98–164, 2016.

[53] W. Palma, *Time series analysis.* John Wiley & Sons, 2016, pp. 12o,2i.

[54] "Fast forest quantile regression component - azure machine learning," 2021, accessed: 2022-07-23. [Online]. Available: https://web.archive.org/web/20220723141243/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/fast-forest-quantile-regression

[55] "Linear regression component - azure machine learning," 2021, accessed: 2022-07-23. [Online]. Available: https://web.archive.org/web/20220723141933/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/linear-regression

[56] "Boosted decision tree regression component - azure machine learning," 2022, accessed: 2022-07-23. [Online]. Available: https://web.archive.org/web/20220723141631/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/boosted-decision-tree-regression

[57] "Decision forest regression component - azure machine learning," 2021, accessed: 2022-07-23. [Online]. Available: https://web.archive.org/web/20220723141819/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/decision-forest-regression

[58] "Neural network regression component - azure machine learning," 2021, accessed: 2022-07-23. [Online]. Available: https://web.archive.org/web/20220723142033/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/neural-network-regression

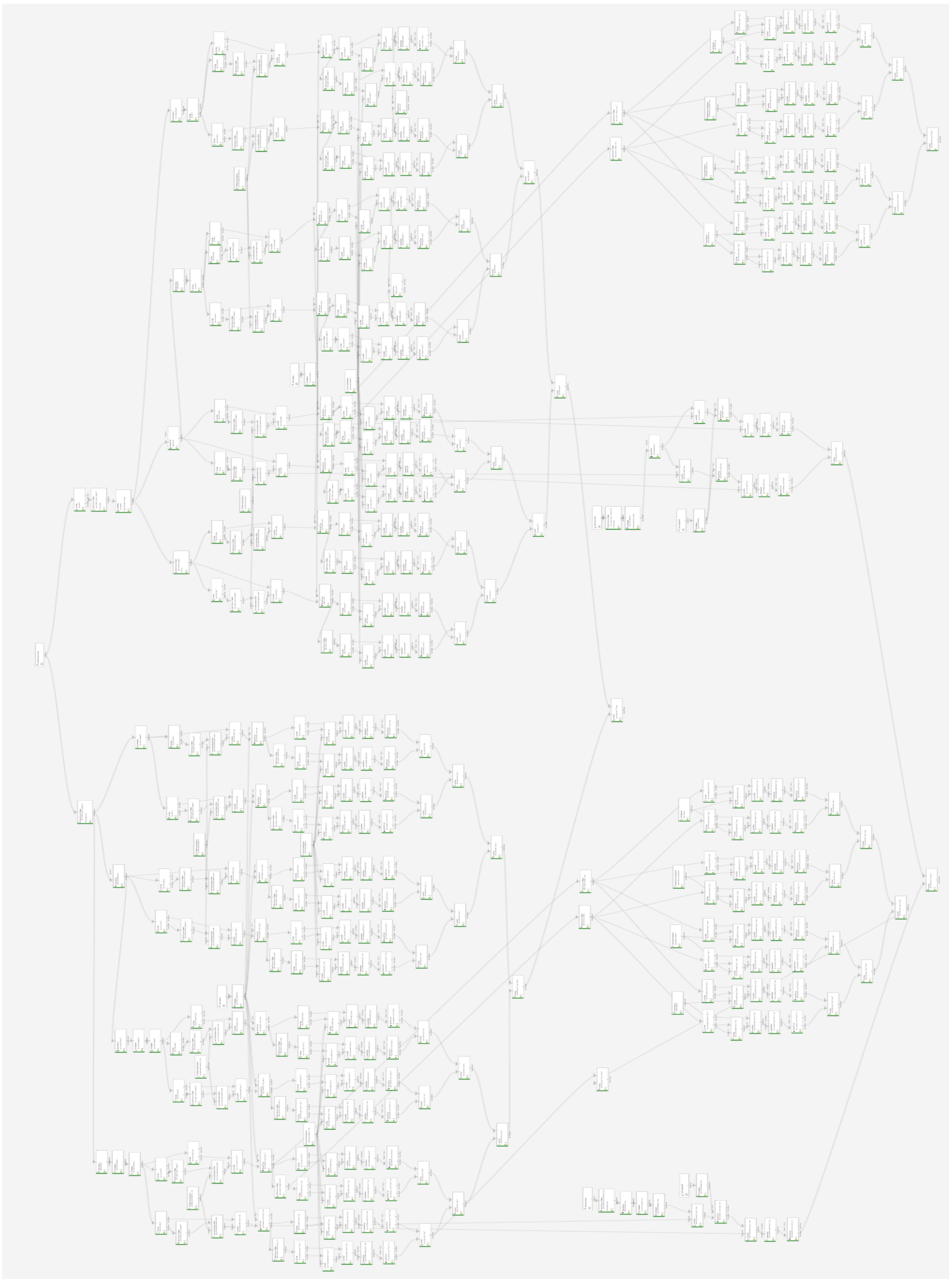
[59] "Poisson regression component - azure machine learning," 2021, accessed: 2022-07-23. [Online]. Available: https://web.archive.org/web/20220723142122/https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/poisson-regression

[60] "Evaluate model component - azure machine learning," 2021, accessed: 2022-07-25. [Online]. Available: https://web.archive.org/web/20220725130626/https:

//docs.microsoft.com/en-us/azure/machine-learning/component-reference/evaluate-model

A Final pipeline structure

Final pipeline includes message and rawmessage branches, their pure, preprocessed and feature hashing combinations, two algorithm comparison blocks, and a three final test branches using fresh data for for model scoring.



B RPA log data example

RPA log data in raw format after anonymizing and exporting to local environment. Here are four lines of log events with different log levels, first line being a header row. First example has been reduced in size with `[...]`-marking. Data is in CSV-format, with each column separated by comma. The rawmessage is injected inside CSV in JSON-format.

This data was fetched from the SQL-database with the following SQL-script:

```
SELECT OrganizationUnitId,TenantId,TimeStamp,Level,
   ProcessName,JobKey,RobotName,Message,RawMessage,
   MachineId FROM [dbo].[logs]
```

```
"OrganizationUnitId","TenantId","TimeStamp","Level","ProcessName","JobKey","RobotName","Message","RawMessage","MachineId"
"5","5","1/2/2020 12:55:24 PM","4","rpa-bank-020-lainahakemuksien-tietojen-siirto_Samlink Production",
    "b03cfd05-1807-45e7-b962-c9d95e49a3fc","RPA-RPP-4","System.IO.IOException: The process cannot access the file
    'W:\RPA\BANK\020 lainahakemuksien tietojen siirto\prod\Config.xlsx' because it is being used by another process.
    at System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
    at System.IO.FileStream.Init(String path, FileMode mode, FileAccess access, Int32 rights, Boolean useRights,
        FileShare share, Int32 bufferSize, FileOptions options, SECURITY_ATTRIBUTES secAttrs, String msgPath, Boolean
        bFromProxy, Boolean useLongPath, Boolean checkHost)
    at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare share, Int32 bufferSize,
        FileOptions options, String msgPath, Boolean bFromProxy)
    at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare share, Int32 bufferSize,
        Boolean useAsync)
        [...]
    at System.Activities.Runtime.ActivityExecutor.ExecuteActivityWorkItem.ExecuteBody(ActivityExecutor executor,
        BookmarkManager bookmarkManager, Location resultLocation)","{
    "message": "System.IO.IOException: The process cannot access the file 'W:\\RPA\\BANK\\020 lainahakemuksien tietojen
        siirto\\prod\\Config.xlsx' because it is being used by another process.\r\n   at System.IO.__Error.WinIOError
        (Int32 errorCode, String maybeFullPath)\r\n   at System.IO.FileStream.Init(String path, FileMode mode,
        FileAccess access, Int32 rights, Boolean useRights, FileShare share, Int32 bufferSize, FileOptions options,
        SECURITY_ATTRIBUTES secAttrs, String msgPath, Boolean bFromProxy, Boolean useLongPath, Boolean checkHost)\r\n
        at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare share, Int32 bufferSize,
        FileOptions options, String msgPath, Boolean bFromProxy)\r\n   at System.IO.FileStream..ctor(String path,
        FileMode mode, FileAccess access, FileShare share, Int32 bufferSize, Boolean useAsync) [...]   at
        System.Activities.Runtime.ActivityExecutor.ExecuteActivityWorkItem.ExecuteBody(ActivityExecutor executor,
        BookmarkManager bookmarkManager, Location resultLocation)",
    "level": "Error",
    "logType": "User",
    "timeStamp": "2020-01-02T14:55:24.6280687+02:00",
    "fingerprint": "5ebb011b-fb3d-402a-8024-79a21e6629c9",
    "windowsIdentity": "LOCAL\\WinID",
    "machineName": "T4690A3018",
    "processName": "rpa-bank-020-lainahakemuksien-tietojen-siirto_Samlink Production",
    "processVersion": "1.0.7244.25507",
    "jobId": "b03cfd05-1807-45e7-b962-c9d95e49a3fc",
    "robotName": "RPA-RPP-4",
    "machineId": 28,
    "fileName": "GlobalHandler"
}","28"
"5","5","1/2/2020 12:55:24 PM","0","rpa-bank-020-lainahakemuksien-tietojen-siirto_Samlink Production",
    "b03cfd05-1807-45e7-b962-c9d95e49a3fc","RPA-RPP-4","Log Error Executing","{
    "message": "Log Error Executing",
    "level": "Verbose",
    "logType": "Default",
    "timeStamp": "2020-01-02T14:55:24.6280687+02:00",
    "fingerprint": "3c68aacf-dd4c-4ad4-8045-043e72b35bb7",
    "windowsIdentity": "LOCAL\\WinID",
    "machineName": "T4690A3018",
    "processName": "rpa-bank-020-lainahakemuksien-tietojen-siirto_Samlink Production",
    "processVersion": "1.0.7244.25507",
    "jobId": "b03cfd05-1807-45e7-b962-c9d95e49a3fc",
    "robotName": "RPA-RPP-4",
    "machineId": 28,
    "fileName": "GlobalHandler",
    "activityInfo": {
        "Activity": "UiPath.Core.Activities.LogMessage",
        "DisplayName": "Log Error",
        "State": "Executing",
        "Variables": {
            "answer": "",
            "retry": "0",
            "HetuFound": "False",
            "FilteredException": ""
        },
        "Arguments": {
            "Message": "System.IO.IOException: The process cannot access the file 'W:\\RPA\\BANK\\020 lainahakemuksien
                tietojen siirto\\prod\\Config.xlsx' because it is being used by another process.\r\n   at System.IO.
                __Error.WinIOError(Int32 errorCode, String maybeFullPath)\r\n   at System.IO.FileStream.Init(String
                path, FileMode mode, FileAccess access, Int32 rights, Boolean useRights, FileShare share, Int32
                bufferSize, FileOptions options, SECURITY_ATTRIBUTES secAttrs, String msgPath, Boolean bFromProxy,
                Boolean useLongPath, Boolean checkHost)\r\n   at System.IO.FileStream..ctor(String path, FileMode mode,
                FileAccess access, FileShare share, Int32 bufferSize, FileOptions options, String msgPath, Boolean
                bFromProxy)\r\n   at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare
                share, Int32 bufferSize, Boolean useAsync)\r\n   at MS.Internal.IO.Zip.ZipArchive.OpenOnFile(String
                path, FileMode mode, FileAccess access, FileShare share, Boolean streaming)\r\n   at System.IO.
                Packaging.ZipPackage..ctor(String path, FileMode mode, FileAccess access, FileShare share, Boolean
                streaming)\r\n   at System.IO.Packaging.Package.Open(String path, FileMode packageMode, FileAccess
                packageAccess, FileShare packageShare, Boolean streaming)\r\n   at DocumentFormat.OpenXml.Packaging.
                OpenXmlPackage.OpenCore(String path, Boolean readWriteMode)\r\n   at DocumentFormat.OpenXml.Packaging.
                SpreadsheetDocument.Open(String path, Boolean isEditable, OpenSettings openSettings)\r\n   at ClosedXML.
                Excel.XLWorkbook.LoadSheets(String fileName) in C:\\Projects\\ClosedXML\\ClosedXML\\Excel\\XLWorkbook
                _Load.cs:line 44\r\n   at UiPath.Excel.WorkbookFile..ctor(String workbookPath, String password, Boolean
                createNew)\r\n   at UiPath.Excel.Activities.WorkbookActivity‘1.ConstructWorkbook(String path, String
                password, Boolean createNew)\r\n   at UiPath.Excel.Activities.WorkbookActivity‘1.BeginExecute(
                AsyncCodeActivityContext context, AsyncCallback callback, Object state)\r\n   at System.Activities.
                AsyncCodeActivity.InternalExecute(ActivityInstance instance, ActivityExecutor executor, BookmarkManager
                bookmarkManager)\r\n   at System.Activities.ActivityInstance.Execute(ActivityExecutor executor,
                BookmarkManager bookmarkManager)\r\n   at System.Activities.Runtime.ActivityExecutor.
                ExecuteActivityWorkItem.ExecuteBody(ActivityExecutor executor, BookmarkManager bookmarkManager, Location
                resultLocation)"
        }
    }
}","28"
"5","5","8/11/2021 10:14:26 PM","3","rpa-bank-011-ott-valmistelu-4503","5186baa6-6f2d-4cc0-8001-a78f518579fb",
    "RPA-RPP-1-4503","Screenshot saved at: G:\Robotiikka\011 OTT Valmis\prod\Screensh\ExceptionScreenshot_.011426.png","{
    "message": "Screenshot saved at: G:\\Robotiikka\\011 OTT Valmis\\prod\\Screensh\\ExceptionScreenshot_.011426.png",
    "level": "Warning",
    "logType": "User",
    "timeStamp": "2021-08-12T01:14:26.3516259+03:00",
    "fingerprint": "3ced557c-7a73-4e0c-bd73-1759fbbd8cca",
    "windowsIdentity": "LOCAL\\WinID",
    "machineName": "T4690K3336",
    "processName": "rpa-bank-011-ott-valmistelu-4503",
    "processVersion": "1.6.7769.24916",
    "jobId": "5186baa6-6f2d-4cc0-8001-a78f518579fb",
    "robotName": "RPA-RPP-1-4503",
    "machineId": 38,
    "organizationUnitId": 5,
    "fileName": "TakeScreenshot",
    "logF_BusinessProcessName": "rpa-bank-011-OTT-valmistelu"
}","38"
"5","5","8/11/2021 9:46:52 PM","2","rpa-bank-011-ott-valmistelu-4260","66d75a97-9f08-401c-8e27-b1d23463a44f",
    "RPA-RPP-2-4260","Kaikkien tilien saldo tÃ¤lle hetulle nyt: 1.05","{
    "message": "Kaikkien tilien saldo tÃ¤lle hetulle nyt: 1.05",
    "level": "Information",
    "logType": "User",
    "timeStamp": "2021-08-12T00:46:52.328433+03:00",
    "fingerprint": "5b4c9287-eb8d-4fb0-9e21-015eee3b3adb",
    "windowsIdentity": "LOCAL\\WinID",
    "machineName": "T4690A3016",
    "processName": "rpa-bank-011-ott-valmistelu-4260",
    "processVersion": "1.6.7769.24916",
    "jobId": "66d75a97-9f08-401c-8e27-b1d23463a44f",
    "robotName": "RPA-RPP-2-4260",
    "machineId": 11,
    "organizationUnitId": 5,
    "fileName": "TilienSaldot",
    "logF_BusinessProcessName": "rpa-bank-011-OTT-valmistelu"
}","11"
```

C Anonymization script

Next here is the PowerShell script used to anonymize the RPA log data. During data fetching with SQL, special string tokens were added at the beginning and the end of each row for anonymizing script to separate each row. This was because of the rawmessage-field which included line breaks and thus span of one log entry could be over dozens of rows in CSV file.

As data size was multiple gigabytes, it was impossible to read it to memory in one go. Instead, stream reading was applied. Script reads one log entry to the memory, makes necessary processing and anonymization, and outputs the entry in one line to output file before reading the next row.

```
# UltimateAnonymizer.ps
# This script converts product data to pure CSV and anonymizes it on the go

$prodLevel = "tuotanto"

# Input file path
$path = "C:\dippa\tuotanto\tuotanto_arkisto_$prodLevel.csv"
$errorLogPath = "C:\dippa\tuotanto\ultimate_error.log"
$logpath = "C:\dippa\tuotanto\ultimate_run.log"

Out-File -Encoding utf8 -FilePath $logpath
Add-Content $logpath "$(Get-Date): Starting powershell script"

Out-File -Encoding utf8 -FilePath $errorLogPath

# Output file path
# Important: specify a *full* path
$outFileStream = [System.IO.StreamWriter] "C:\dippa\tuotanto\tuotanto_arkisto_$prodLevel‘_final.csv"

$lineCounter = 0
$json = ’’

$outFileStream.WriteLine(’"organizationUnitId","level","logType","timeStamp","fingerprint","machineName",
    "processName","jobId","robotName","machineId","message","rawmessage"’)

$SSNregex = "(?<![a-zA-Z0-9])[\d]{6}[-a+]?[\d]{3}[\w]{1}(?:0{0}|0{3})(?![a-zA-Z0-9])"
$EMAILregex = "[^\‘"\s]+@[\.\w-]*[\w]"
$IBANregex = "(?:(?<![a-zA-Z0-9])|(?<=\\\D))(?:FI|fi)(?: ?\d){16}(?![a-zA-Z0-9])"
$BBANDASHregex = "(?<![a-zA-Z0-9])[\d]{6}-[\d]{2,8}(?![a-zA-Z0-9])"
$PHONEINTregex = "(?<![a-zA-Z0-9])\+358(?: ?\d){8,10}(?![a-zA-Z0-9])"
$PHONEregex = "(?<![a-zA-Z0-9-])[0][\d]{2,3}[ -]?(?: ?\d){6,8}(?![a-zA-Z0-9-])"
$BUSINESSregex = "(?<![a-zA-Z0-9])[\d]{7}-[\d]{1}(?![a-zA-Z0-9])"
$BUSINESSINTregex = "(?<![a-zA-Z0-9])[a-zA-Z]{2}[\d]{8}(?![a-zA-Z0-9])"
$BUSINESSINTZEROregex = "(?<![a-zA-Z0-9])[0]{2}[\d]{8}(?![a-zA-Z0-9])"
$CCregex = "(?<![a-zA-Z0-9-.])[\d]{1}(?: ?\d){14,15}(?![a-zA-Z0-9-])"
$WINIDregex = "(?<![a-zA-Z0-9])[a-zA-Z]{1,2}[\d]{6}(?![a-zA-Z0-9])"
$ADDRCOMregex = "[^\s""’,.]* ?(katu|tie|kuja|polku|kaari|linja|raitti|rinne|penger|ranta|väylä|taival|tanhua|portti
    |veräjä|laita|reuna|syrjä|aukio|tori|laituri|tunneli) [\d]{1,3}( ?[a-zA-Z.]{1,4} ?[\d]{0,3})?(?!\w)"
$ADDRZIPregex = "(?<=\s)[\S]* [\d]{1,3}( ?[a-zA-Z.]{1,4} ?[\d]{0,3})?(\s|,\s)[\d]{5}(?!\w)"
$BankIDregex = "(?<![a-zA-Z0-9-])[\d]{8}(?![a-zA-Z0-9-])"
$BBANnoDASHregex = "(?<![a-zA-Z0-9])[\d]{14}(?![a-zA-Z0-9])"
$BUSINESSArtificialregex = "(?<![a-zA-Z0-9])[89]{1}[\d]{9}(?![a-zA-Z0-9])"

function Hide-Sensitive {
    param (
        [Parameter(Mandatory, ValueFromPipeline)]
        [string]$sensitiveLine
    )
    # OBS! If number is replaced with letters,
    # it is possible we break JSON key-value pair where value is (was) number.
    # This is why possibly only number containing strings are replaced to numbers
    $sensitiveLine -replace $SSNregex ,"10105051470101" ‘
            -replace $EMAILregex ,"EmailAddress0101" ‘
            -replace $IBANregex ,"1010IBANnumber0101" ‘
            -replace $BBANDASHregex ,"1010BBANnumber0101" ‘
            -replace $BBANnoDASHregex ,"101088420101" ‘
            -replace $PHONEINTregex ,"1010PhoneNumberInt0101" ‘
            -replace $PHONEregex ,"1010980230101" ‘
            -replace $BUSINESSregex ,"1010BusinessID0101" ‘
            -replace $BUSINESSINTregex ,"101086512350101" ‘
            -replace $BUSINESSINTZEROregex ,"1010865123500101" ‘
            -replace $BUSINESSArtificialregex ,"101086512354970101" ‘
            -replace $CCregex ,"1010664900101" ‘
            -replace $BankIDregex ,"10108426100101" ‘
            -replace $WINIDregex ,"1010WinID0101" ‘
            -replace $ADDRCOMregex ,"1010AddressCommon0101" ‘
            -replace $ADDRZIPregex ,"1010AddressZip0101" ‘
}
```

```
function Write-Json {
    param (
        [Parameter(Mandatory, ValueFromPipeline)]
        [string]$json
    )
    try {
        # From each message/rawmessage field, first filter out all line endings, then remove unnecessary special characters,
        # excess whitespaces, and finally anonymize sensitive information
        ($json | ConvertFrom-Json).SyncRoot |
        Select-Object organizationUnitId,level,logType,timeStamp,fingerprint,machineName,processName,jobId,robotName,machineId,
        @{l=’message’;e={$_.message -replace ’(\r|\n|‘r|‘n)’, ’’ -replace ’[^\p{L}\p{Nd} .,\-_\/(){}\[\]:+]’,
            ’’ -replace ’\s+’, ’ ’ | Hide-Sensitive}},
        @{l=’rawmessage’;e={[System.Text.RegularExpressions.Regex]::Unescape($json) -replace ’(\r|\n|‘r|‘n)’,
            ’’ -replace ’[^\p{L}\p{Nd} .,\-_\/(){}\[\]:+]’, ’’ -replace ’\s+’, ’ ’ | Hide-Sensitive}} |
        ConvertTo-Csv -NoTypeInformation |
        Select-Object -Skip 1 |
        Foreach-Object { $outFileStream.WriteLine($_) }
    }
    catch {
        Add-Content $errorLogPath "$(Get-Date): Found some errors with data during writing JSON to CSV:"
        Add-Content $errorLogPath "$_"
        Add-Content $errorLogPath "*****‘r‘nFull JSON at that moment:"
        Add-Content $errorLogPath "$json"
    }
}

Add-Content $logpath "$(Get-Date): CSVfying data!"

# start of row: "#s#"
# end of row: "#e#"

# Special character remove: https://lazywinadmin.com/2015/08/powershell-remove-special-characters.html

$reader = [System.IO.File]::OpenText($path)
try {
    while($null -ne ($line = $reader.ReadLine())) {
        switch -Regex -casesensitive ($line) {
            ’(?<=^"#s#",)("\d{1,2}")(.*(?>}","#e#"$))?’ {
                $json = ’[{’ + "‘r‘n" + "  ‘"organizationUnitId‘": " + $Matches[1] + ","
                if ($Matches[2]) {
                    $json += $Matches[2] -replace ’,"{’, ’’ -replace ’}","#e#"’, ’’
                    $json += "‘r‘n}]"
                    $json | Write-Json
                    $lineCounter += 1;
                    if ($lineCounter % 500000 -eq 0) {
                        Add-Content $logpath "$(Get-Date): $lineCounter lines passed and counting..."
                    }
                    $json = ’’;
                }
                else {
                    :oneObject while($null -ne ($jsonline = $reader.ReadLine())) {
                        switch -Regex ($jsonline) {
                            ’^\}","#e#"’ {
                                $json += "‘r‘n}]"
                                $json | Write-Json
                                $lineCounter += 1;
                                if ($lineCounter % 500000 -eq 0) {
                                    Add-Content $logpath "$(Get-Date): $lineCounter lines passed and counting..."
                                }
                                $json = ’’;
                                break oneObject
                            }
                            default {
                                if ($json) {
                                    $json += "‘r‘n" + $_
                                }
                                else {
                                    Add-Content $errorLogPath "$(Get-Date): Stumbled to row outside JSON content:"
                                    Add-Content $errorLogPath "$_"
                                }
                            }
                        }
                    }
                }
            }
            default {
                Add-Content $errorLogPath "$(Get-Date): Stumbled to a row outside switch start loop:"
                Add-Content $errorLogPath "$_"
            }
        }
    }
}
finally {
    $reader.Close()
}

$outFileStream.Close()

Add-Content $logpath "$(Get-Date): File CSVfying completed! Counted $lineCounter lines of JSON objects!"

exit
```

D CSV data cleaning script

Next script was mainly used to clean unique data values from rawmessage-field. As rawmessage was JSON formatted text data and included log entry specific identifiers, mainly timestamp, fingerprint, and jobid, such values were necessary to clean from the rawmessage before data could be given to anomaly algorithm to parse.

This script is used for data that has been already anonymized and is thus in proper CSV format, one row per log entry. Script works much like anonymization script working with stream reading and editing in order to avoid unnecessary memory loading.

```
# CsvCleaner.ps
# This script cleans some data out of finished CSV file
# to remove unique values to make anomaly detection possible

$prodLevel = "newprod"
$path = "C:\dippa\tuotanto\tuotanto_arkisto_$prodLevel‘_final.csv"
$errorLogPath = "C:\dippa\tuotanto\csvcleaner_$prodLevel‘_error.log"
$logpath = "C:\dippa\tuotanto\csvcleaner_$prodLevel‘_run.log"

Out-File -Encoding utf8 -FilePath $logpath
Add-Content $logpath "$(Get-Date): Starting powershell script"
Out-File -Encoding utf8 -FilePath $errorLogPath
$outFileStream = [System.IO.StreamWriter] "C:\dippa\tuotanto\
    tuotanto_arkisto_$prodLevel‘_cleaned_final.csv"

$headers = "organizationUnitId","level","logType","timeStamp",
    "fingerprint","machineName","processName","jobId","robotName",
    "machineId","message","rawmessage"

$outFileStream.WriteLine('"organizationUnitId","level","logType",
    "timeStamp","machineName","processName","jobId","robotName","
    machineId","message","rawmessage"')

$timeStampRegex = "timeStamp: .*?,"
$fingerprintRegex = "fingerprint: .*?,"
$jobIdRegex = "jobId: .*?,"

function Remove-Unique {
    param (
        [Parameter(Mandatory, ValueFromPipeline)]
        [string]$lineInProgress
    )
    $lineInProgress -replace $timeStampRegex ,"" ‘
            -replace $fingerprintRegex ,"" ‘
            -replace $jobIdRegex ,"" ‘
}
```

```
Add-Content $logpath "$(Get-Date): Cleaning data!"
$lineCounter = 0
$reader = [System.IO.File]::OpenText($path)
try {
    while($null -ne ($line = $reader.ReadLine())) {
        if($lineCounter -eq 0) {
            $lineCounter += 1;
            continue;
        }
       # $line
        try {
            $line | ConvertFrom-Csv -header $headers |
            Select-Object organizationUnitId,level,logType,
                timeStamp,machineName,processName,jobId,
                robotName,machineId,message,
                @{l='rawmessage';e={$_.rawmessage | Remove-Unique}} |
                ConvertTo-Csv -NoTypeInformation |
                Select-Object -Skip 1 |
                Foreach-Object { $outFileStream.WriteLine($_) }
        }
        catch {
            Add-Content $errorLogPath "$(Get-Date):
                Found some errors with data during reading CSV:"
            Add-Content $errorLogPath "$line"
        }

        $lineCounter += 1;
        if ($lineCounter % 500000 -eq 0) {
            # break;
            Add-Content $logpath "$(Get-Date):
                $lineCounter lines passed and counting..."
        }
    }
}
finally {
    $reader.Close()
}

$outFileStream.Close()

Add-Content $logpath "$(Get-Date): File cleaning completed!
    Counted $lineCounter lines of CSV!"
exit
```

E Example of R-script

R-script is responsible for the time frame compression. It works by summarizing various values in a time frame into new features.

```
azureml_main <- function(dataframe1, dataframe2){
    library(dplyr)
    library(rlang)
    library(lubridate)

    print("R script run.")
    df1 <- dataframe1 %>% mutate(timeStamp= as.Date(timeStamp)) %>%
    mutate(ScoredProbabilities = as.numeric(‘Scored Probabilities‘))
        %>%
    group_by(timeStamp = as.Date(floor_date(timeStamp, "1 week",
        week_start=getOption("lubridate.week.start", 1)))) %>%
    summarize(LogRowCount = n(),
    UniqueJobIDs= n_distinct(jobId),
    AnomalyProbMean = mean(ScoredProbabilities),
    AnomalyProbMedian = median(ScoredProbabilities),
    Quantile90 = quantile(ScoredProbabilities, c(0.90)),
    CountOverQ90 = sum(ScoredProbabilities > Quantile90)
    )

    df12 <- dataframe1 %>% mutate(timeStamp= as.Date(timeStamp)) %>%
    select(timeStamp, contains(’HashingFeature’)) %>%
    group_by(timeStamp = as.Date(floor_date(timeStamp, "1 week",
        week_start=getOption("lubridate.week.start", 1)))) %>%
    summarize_all(sum)

    df13 <- inner_join(df1, df12, by = ’timeStamp’)

    df2 <- dataframe2 %>% mutate(timeStamp= as.Date(timeStamp)) %>%
    group_by(timeStamp = as.Date(floor_date(timeStamp, "1 week",
        week_start=getOption("lubridate.week.start", 1)))) %>%
    summarize(TicketCount = n())

    df3 <- inner_join(df1, df2, by = ’timeStamp’)
    df4 <- df3
    df3$timeStamp <- strftime(df3$timeStamp, format = "%Y-%m-%V")

    # Return datasets as a Named List
    return(list(dataset1=df3, dataset2=df4))
}
```