TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

„Predictive Process Management for Aircraft MRO“

Authors	Beck, Janina; Bernhardt, Lisa-Marie; Chauhan, Hemraj; Kirschstein, Tobias; Maier-Borst, Moritz
Mentor(s)	Grindemann, Philipp and Hoffmann, Maximilian (Lufthansa CityLine GmbH); Nakladal, Janina (Celonis SE)
Co-Mentor	Maximilian Fiedler
Project lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Abstract

Having successfully introduced Celonis as a process mining software to improve internal processes, the Lufthansa CityLine GmbH (CLH) aims at further improving their Maintenance-Repair-Overhaul-Process (MRO process) by applying machine learning (ML) prediction algorithms. The following report describes the project of the TUM DI-LAB dealing with this task. As the MRO process is highly complex, we conduct several interviews and analyze the process to understand CLH's problem in-depth and derive the overall goal: "develop a predictive process management tool to predict and proactively steer critical cases in the aircraft MRO processes of Lufthansa CityLine until July 2020 by implementing and evaluating three different prediction models". More precisely, we aim at predicting variables for three concrete use cases (incomplete maintenance, deassignments and work order success) that can help CLH to recognize possible issues with a maintenance case in time and then try to prevent them. Based on a literature research on approaches and algorithms in process prediction, we select three ML models for implementation: Decision Tree, Neural Network and a Relational Graph Convolutional Network (R-GCN). Furthermore, we implement a Logistic Regression model as a baseline to evaluate the performance of the other three models.

As the four models require different inputs, we set up a dataset for the R-GCN as well as a dataset with handcrafted features for the other three approaches. For the sake of consistency, both datasets contain the same cases as well as the exact same split into training, validation and test partitions.
After implementing the four different models, we find that the R-GCN achieves the best results in predicting all of the three use cases. This is probably the case since the R-GCN is able to take the hierarchical structure of the data into account, as opposed to the other approaches. However, predicting the use cases shows to be relatively difficult – especially because of the highly imbalanced distribution of the variables for incomplete maintenance and deassignments. Nevertheless, the business cases for predicting incomplete maintenance as well as for deassignments yields a positive business impact by reducing the costs. The use case of predicting work order success can additionally be used for tracking the success of the implementation of the models.
Lastly, the interpretation of the model shows that some features such as a particular station to conduct the work order at are very important for predicting the outcome variables.

All in all, the project showcases the potential of predicting the process flow of cases for a business. In the future, the models can be further improved by including more cases to train the model as well as by validating and interpreting it with experts at CLH.

Table of Contents

1 Introduction

Artificial Intelligence (AI) and Machine Learning (ML) have received a lot of attention in recent years. While there are many unresolved public discussion about what these developments mean for humankind or society as a whole, companies are already focusing on applying ML to either develop completely new products or to improve current processes.

The Lufthansa CityLine GmbH (CLH), the regional carrier of the Lufthansa Group which mainly performs hub-feeder-flights, has quite a long and successful history with respect to the latter. The key element to this success is the close collaboration between CLH and the Technical University of Munich's (TUM) first unicorn start-up Celonis SE. The value created by Celonis' software is derived from the academic discipline of process mining which was introduced by Wil van der Alst in 2011 [1]. Contrary to the previously quite popular business process modelling task, the fundamental idea of process mining is the analysis of logs provided by any kind of IT system. Thereby, the actual process flow of the process' matter of subject can be visualized, compared to the desired process flow and actions can be taken to prevent deviations from the optimal process. Through the application of process mining, CLH already achieved various improvements within their internal processes, in particular, through carefully analyzing the ground operations processes (e.g. catering, cargo loading and fueling). In consequence, punctuality and customer satisfaction improved tremendously.

In order to continue this success story, going beyond the ex-post process discovery stage and advancing into the area of process prediction is next on the agenda of CLH. Thus, being part of the TUM Data Innovation Lab, we aim at supporting Lufthansa CityLine to improve their Maintenance-Repair-Overhaul-Process (MRO process) by applying ML prediction algorithms. This report describes the whole course of the project:

The first chapter comprehensively describes the problem that is to be solved within the scope of the project. Thereby, we give a detailed overview of the most important steps and characteristics of the MRO process as well as the description of the project goal, project setup and the deliverables. The subsequent subchapter offers information on state-of-the-art approaches in the area of ML for process prediction. By outlining their most important characteristics and comparing them with each other, we select three final approaches (namely Neural Networks, Decision Trees and Relational Graph Convolutional Networks (R-GCN)) to be implemented on the data of CLH and be compared to the baseline model of Logistic Regression. Thus, we implement four models in total.

In the second chapter, we detail the process of data analysis: the general approach and the individual adaptation we made due to the project setting. This encompasses all parts of the data analysis: set-up, data preprocessing and feature engineering, evaluation, implementation as well as improvements of the ML models.

The third chapter includes the best results for all the selected algorithms for the three different use cases derived from interviewing employees of CLH: predicting incomplete maintenance, predicting deassignments and predicting work order success. It also discusses the results and feature interpretations and explains the overall outcome of the project. To determine the final business impact of the project the results chapter additionally presents the expected return on prediction of the three use cases based on the current performance of the models.

Finally, we draw an overall conclusion in the fourth chapter by summarizing the project and its outcome followed by drafting ideas on how CLH could continue the project in the future.

1.1 Problem Definition and Goals of the Project

This section describes the concrete problem to be solved throughout the project. Before summarizing the problem and deriving the concrete goal form it, we start by describing the MRO process at CLH before summarizing the problem and deriving the concrete goal.

Any process consists of several stages which are modelled as so called “activities” in Celonis. The analyzed log files of the underlying IT system further introduce a timestamp (= when an activity happened) and a unique case identifier (= which particular process subject has been affected). In consequence, a particular case consists of multiple activities which in turn depict a process flow based on the respective timestamps.

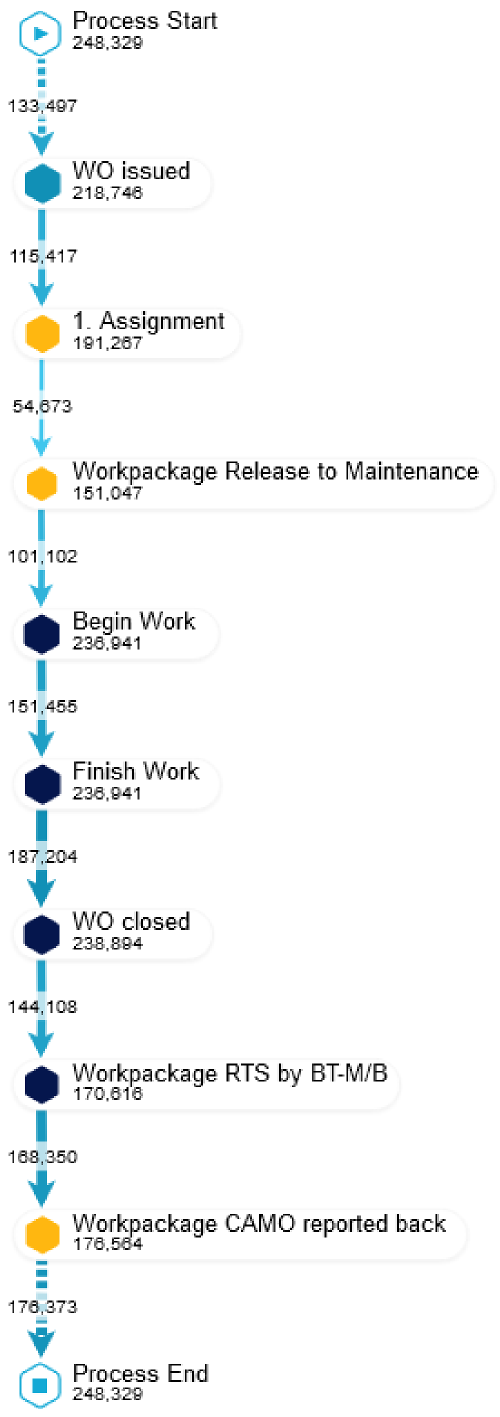


Illustration 1: most common variant of the MRO process flow at CLH

As it can be seen in pretty much every application of Celonis, the actual (as-is) process flow of many cases does not conform with the desired (to-be) process flow. Thus, a lot and often very unique process schemas exist which in turn makes it challenging to visualize all of them in their entirety. However, uniqueness often comes along with rarity. Therefore, analyzing and optimizing the process flow explaining as many cases as possible (= most common variant) is likely to yield the biggest impact on business.

The MRO process at Lufthansa CityLine contains over 30 distinct activities. Thereby, the most common alternative of the process flow consists of eight activities, starting with the issuing of a work order (WO) as depicted in Illustration 1. A work order contains information about e.g. what maintenance needs to be conducted, how long the aircraft is estimated to be grounded due to this maintenance event and general information about the aircraft itself which is subject to the work order.

Each work order is further linked to six or seven work steps on average which sequentially describe what needs to be done in order to complete the required maintenance. However, depending on the scope of the maintenance to be conducted, the amount of work steps can scale to an arbitrary number. A work step also serves as an imaginary bucket for materials, tooling and other resource requirements in order to perform the concrete step. On the other hand, each work order is part of a work package (WP) which contains multiple work orders to be performed by the line maintenance department. A work package usually describes the scope of either a night or a day shift and quite often the same aircraft is subject to an entire work package. The nightshift is the preferred time to perform the maintenance on the aircraft, because short/medium haul aircrafts in Europe stay on the ground due to night curfew anyhow.

In order to grant a high security level, the MRO process itself is heavily influenced by the regulations and directives issued by external stakeholders such as the European Union Aviation Safety Agency (EASA), the aircraft's manufacturer or the Federal Aviation Office (Luftfahrt Bundesamt, LBA). An exemplary directive could demand the inspection of a certain part of the engine after a certain amount of flight hours. These general but also aircraft type tailored directives are collected by the engineering department which for instance keeps track of how many flight hours per aircraft have been completed since the last check. Apart from such part-specific checks there are also so-called A-, B- and C-checks, which are more comprehensive, general inspections an aircraft needs to undergo on a regular basis. Based on these requirements from the external authorities and manufacturers, the engineers issue the respective work order including its major work steps, set a due date and forward it to the planning department. While the engineering department's work orders are of a scheduled nature, the so-called Maintenance Operation Control Center (MOCC) schedules work orders based on rather unforeseen events. An example here could be a broken switch in the cockpit reported by the pilot. The engineers at the MOCC then decide whether the aircraft can continue to operate and if so, for how long until the maintenance needs to be conducted. The thereby created work orders are then also sent to the planning department of CLH for further processing.

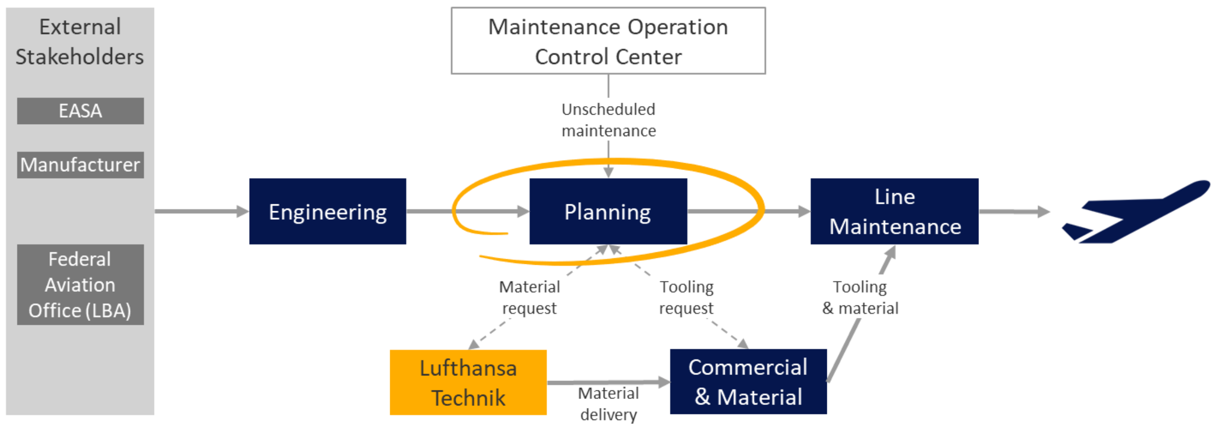


Illustration 2: Organizational Flow Chart for MRO process at CLH

Through conducting various interviews with all different departments being actively involved in the MRO process at CLH, we identified the planning department's central role to the entire process, which is depicted in Illustration 2. Thereby, its main objective is to assign work orders to work packages in the most efficient way. In other words: it is about which work order will be performed at which point in time and where, considering the availability of materials, tools, people and other resources. This task requires close coordination with other departments such as Lufthansa Technik as CLH's external material supplier as well as with the CLH department "Commercial & Material". The latter handles the delivered materials, CLH-owned and rented tools and provides all the resources to the line maintenance department.

It is worth stating here that the line maintenance department is the least critical to the project goal as it just carries out the work based on the availability of the above of material, tooling, people and infrastructure. In consequence any deviation from the "happy" path (optimal path) during the previous stages of the MRO process are likely to create negative impact on CLH's business. The negative consequences vary from rework during the process over delays to grounded aircrafts and thereby cancelled flights. As three other departments are dependent on its actions, the planning department was identified to be the part of the MRO process which has the most impact on the future process flow and would in turn benefit the most from intelligent predictions.

In order to handle the additional heterogeneity among the work orders with respect to their predictability and consequently urgency to minimize the time of the aircraft being grounded, the planning department is further split into three teams. While the long-term planning team focuses on handling work orders with a lead time of more than three months, the mid-term planning team performs the assignments within a timeframe of three months to three days in advance. Lastly, the short-term planning team is in charge of assigning work orders within the last 72 hours before the line maintenance department puts its hands on the aircraft. It is important to state that the short-term planning's lead time is already very scarce and efforts are mainly focused on avoiding the grounding of the aircraft by all means. Therefore, predictions regarding the further process flow would not be of much help here. Contrary to this, active steering of a work order, e.g. through the re-assignment to a different work package, is possible within the scope of the long- and mid-term planning. In order to leverage this potential, we decided to tailor the project towards supporting these two teams of the planning department, in particular.

After outlining the complexity of the MRO process and the problems which arise from it, we want to clearly formulate the prediction targets and the overall goal of this TUM Data Innovation Lab project in the following.

The overall smart goal is to improve the MRO process e.g. by predicting certain types of events and process paths of a work order. To facilitate the project planning, we phrased the following SMART goal for the project:

> **"We develop a predictive process management tool to predict and proactively steer critical cases in the aircraft MRO processes of Lufthansa CityLine until July 2020 by implementing and evaluating three different prediction models."**

With this goal formulation in mind, we analyzed the MRO process flow in detail in Celonis while in particular focusing on the tasks being performed by the planning department. Thereby, we identified not only inefficiencies within this focus department but also activities affecting the overall process and thus business success. The prediction targets derived from this analysis are "incomplete maintenance", "deassignments" and "work order success". Each of them yields different benefits which are summarized in Illustration 3 and outlined in detail in the following.

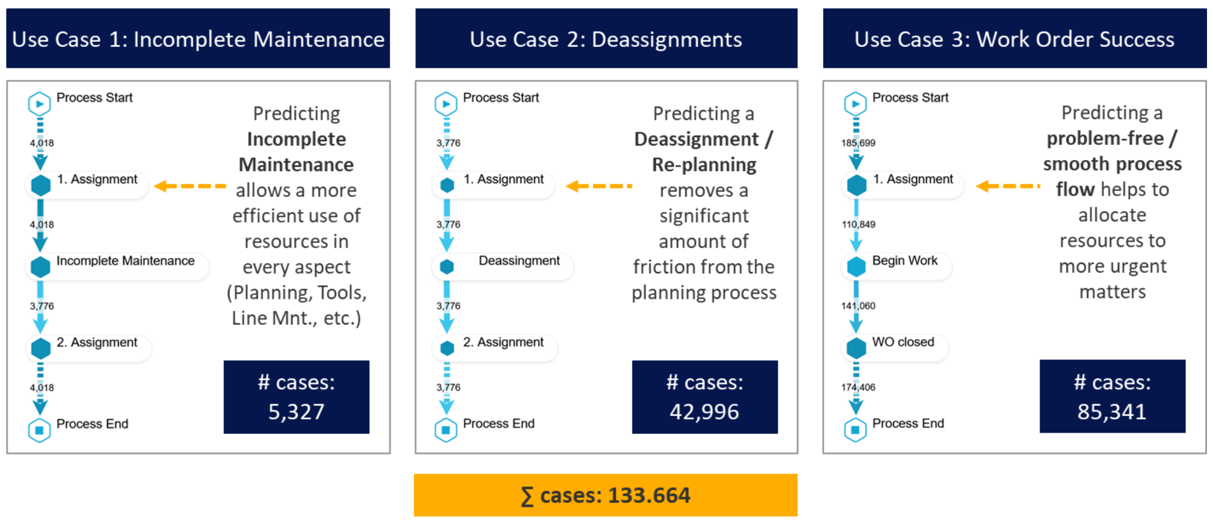

Use Case 1: Incomplete Maintenance
Process Start
4,018
1. Assignment
4,018
Incomplete Maintenance
3,776
2. Assignment
4,018
Process End
Predicting **Incomplete Maintenance** allows a more efficient use of resources in every aspect (Planning, Tools, Line Mnt., etc.)
# cases: 5,327
Use Case 2: Deassignments
Process Start
3,776
1. Assignment
3,776
Deassingment
3,776
2. Assignment
3,776
Process End
Predicting a **Deassignment / Re-planning** removes a significant amount of friction from the planning process
# cases: 42,996
Use Case 3: Work Order Success
Process Start
185,699
1. Assignment
110,849
Begin Work
141,060
WO closed
174,406
Process End
Predicting a **problem-free / smooth process flow** helps to allocate resources to more urgent matters
# cases: 85,341
∑ cases: 133.664

Illustration 3: The three use cases to predict and their share within the dataset

Firstly, we aim at predicting the occurrence of an incomplete maintenance event. If this state of a work order is reported back by the line maintenance department, this means that the required maintenance could not be conducted in time or not conducted at all, e.g. due to missing materials. The worst-case consequence of this scenario can be even an “aircraft on ground” (AOG) event. In other words, the airplane is no longer allowed to fly. This in turn results in costs due to short-notice aircraft replacements, re-planning efforts and bears other negative effects such as customer dissatisfaction, e.g. if a flight does not take place. Therefore, predicting under which circumstances such an incomplete maintenance event occurs allows for a better use of resources throughout the entire process. More precisely, if a case is predicted to end up in incomplete maintenance, one could increase the effort to manage this case and thus prevent further issues, e.g. by assigning additional workers or scheduling more time for the completion of the task.

The second use case is the deassignment of a work order from a work package. In other words, the particular circumstances of the situation require a replanning of the respective work order to another work package. For instance, this is the case if a part which is required to perform the task cannot be delivered or any other unforeseen issues happen (e.g. a mechanic with the required qualification is not available at the currently planned time slot). Naturally, each time a work order is replanned this requires additional time spent by the planner and therefore is costly – especially if a work order is deassigned (= has to be replanned) multiple times. Hence, predicting a deassignment prior to the actual deassignment could reduce this additional effort as e.g. the case could then be planned very carefully at the first time to avoid replanning under potentially unforeseen circumstances at a later stage of the process.

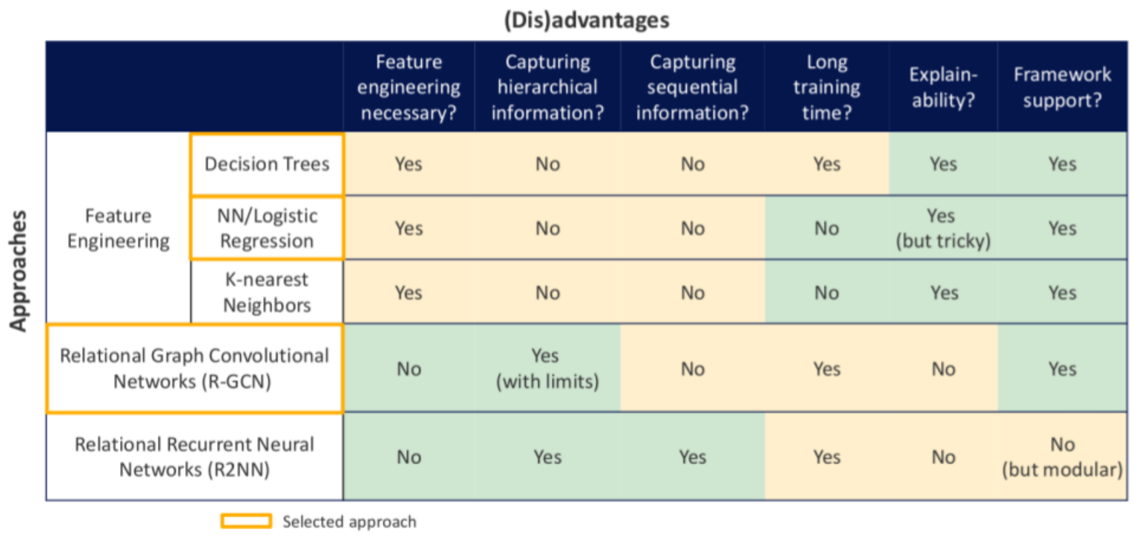
Lastly, it might also be helpful to predict if a work order will be completed successfully. In this case, this means that there are no major issues during the process. For the course of this project, we defined a work order as being successful if no incomplete maintenance or deassignment occurred during the process. Having this information can support CLH’s employees to separate critical from non-critical work orders which allows them to spend their time more efficiently e.g. by focusing on the more critical work orders and work packages.

In the following, we analyze different Machine Learning approaches that could be used to reach our SMART goal.

1.2 State-of-the art approaches and algorithms in Process Prediction

Typically, predictive process monitoring is done by mostly considering the process graph, i.e., all the past activities of a running case. For the aircraft maintenance process at CLH, this graph is rather simple, while the objective of predicting problematic cases is very complex. As such, solely relying on the process graph will not be sufficient, especially since predictions early in the process are favored. Therefore, we need detailed information about everything that is involved with a maintenance case, which can be obtained by exploiting the database of CLH’s maintenance operation tool. On a high level, our task is thus to classify whether a maintenance case will be problematic based on data from a relational database.

The main challenge of this task is the heterogenous nature of such data. Machine Learning models usually expect a feature vector of a *fixed size* as input, which is not the case for relational data from databases. Relevant data points can be scattered across different tables in the database, connected via 1:1, 1:N or N:M cardinalities, which makes it difficult to represent every case with a vector of the same fixed size. Furthermore, important information can be stored multiple “hops” away from the table containing maintenance cases. As a result, the amount of datapoints available for different cases varies drastically. To address the issues arising from relational data, we can either manually aggregate 1:N cardinalities by possibly loosing information (Feature Engineering) or find neural architectures that are suited for relational data (end-to-end learning). In the following, we will describe suitable approaches for both streams, compare them and decide on a subset that we will implement. The results of this comparison are depicted in Illustration 4.



(Dis)advantages

Approaches		Feature engineering necessary?	Capturing hierarchical information?	Capturing sequential information?	Long training time?	Explain-ability?	Framework support?
Feature Engineering	▭ Decision Trees	Yes	No	No	Yes	Yes	Yes
	▭ NN/Logistic Regression	Yes	No	No	No	Yes (but tricky)	Yes
	K-nearest Neighbors	Yes	No	No	No	Yes	Yes
▭ Relational Graph Convolutional Networks (R-GCN)		No	Yes (with limits)	No	Yes	No	Yes
Relational Recurrent Neural Networks (R2NN)		No	Yes	Yes	Yes	No	No (but modular)

▭ Selected approach

Illustration 4: Overview of Different Approaches for Process Prediction, their (Dis)Advantages and the Final Selection

To decide on a set of suitable approaches, we identified a set of criteria that are important for our use case:

- *Expressiveness*: can the model capture hierarchical/sequential information? The process graph is inherently sequential and the data in the database is structured in a hierarchical manner. A model that is designed for this kind of data is likely to show better results.
- *Effort*: Do we have to manually craft features? Does it take long to train or fit a model? Is there already a framework that speeds up implementation?
- *Explainability*: For real business cases, considering the end user is very important. A user who does not understand the reasoning behind a model’s decision will be reluctant to use it. Thus, considering models that can answer *why* the model predicted a certain class are crucial. We present some exemplary explanations for the predictions of our final models later in section 4.2

For the Feature Engineering stream, we focused on Decision Trees, Logistic Regression / Neural Networks and K-Nearest Neighbors. All these approaches are straight forward to implement and are either interpretable by design (Decision Trees, K-Nearest Neighbors) or can be explained by pointing at input features that were crucial for the prediction (Neural Networks). This is feasible, as the feature vector consists of hand-crafted features which are human-understandable.

Despite the need for manual feature engineering, we decided to use Decision Trees as well as Neural Networks due their easy implementation and interpretation. Additionally, we employ Logistic Regression as a baseline. K-Nearest Neighbors requires defining suitable distance metrics for every dimension of the feature vector. As this is not straight-forward for all our handcrafted features, we decided to not further pursue this approach. More details on how we craft the features via aggregations of 1:N cardinalities can be found in section 2.2.1.

For the end-to-end learning stream, a literature research revealed two experimental neural architectures that are capable of operating directly on the data from the database in an end-to-end manner.

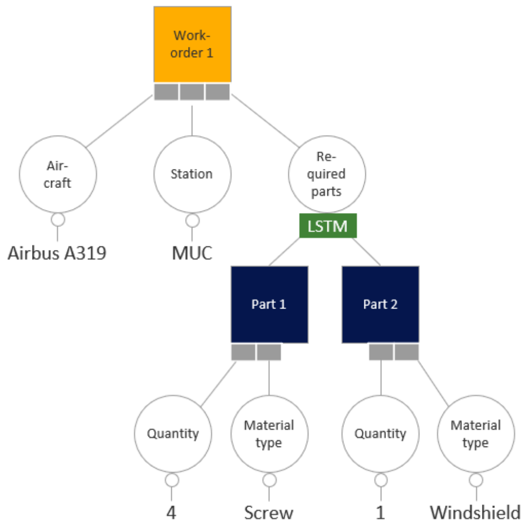


Illustration 6: Relational Recurrent Neural Network (R2NN)

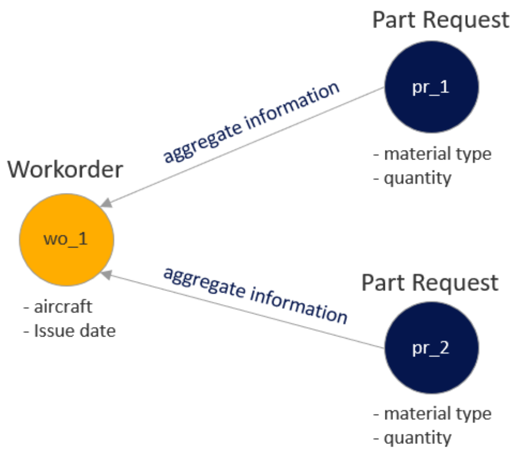


Illustration 5: Relational Graph Convolutional Network (R-GCN)

Relational Recurrent Neural Networks (R2NN) [2] interpret the data from the database as a tree by considering the case row in the main table as the tree root and all linked rows as its children. Information is then aggregated in a bottom-to-top fashion using *Recurrent Neural Networks* (RNNs) as depicted in Illustration 6. After the aggregation, the tree root contains information from all children which can then be used to make a prediction.

Relational Graph Convolutional Networks (R-GCNs) [3] are an extension of regular *Graph Convolutional Networks* (GCNs) for relational data. The idea is to interpret the data from the database as a graph where table rows correspond to nodes and links between these rows correspond to edges. Illustration 5 exemplifies this idea. Tables usually contain very different information. Thus, treating every node equally (as it is done in regular GCNs) would not account for this fact. R-GCNs solve this shortcoming of regular GCNs by introducing relation-specific transformations and are thus interesting for our use-case. A more detailed description of R-GCNs can be found in section 3.2.

Both approaches can directly operate on data from the database and account for its hierarchical nature. Additionally, R2NNs are well suited for sequential data. However, as both approaches constitute complex neural architectures, they have long training times, their predictions are hard to interpret and, in case of R2NNs, do not have an existing easy-to-use implementation. Since many of the relevant datapoints in the database are not sequential, we decided on the R-GCN due to the better framework support.

To summarize, we approach the task with Decision Trees and Neural Networks using hand-crafted features and furthermore employ Relational Graph Convolutional Networks (R-GCNs) that can work on the raw data from the database in an end-to-end manner. The next chapter describes the preprocessing procedure including the setup of the infrastructure and the data inputs.

2 Data Analysis

To capture the whole process of the data analysis conducted throughout the project, this chapter starts by outlining the two main data sources – Celonis and AMOS – that were used. Afterwards, we describe the feature engineering required to implement the baseline model of Logistic Regression, the Neural Networks and Decision Trees as well as the preprocessing for the R-GCNs. Before describing the implementation of the different models we lay the groundwork for this by presenting the results of an exploratory data analysis on the final input dataset and describing the metrics used to compare the different algorithms.

2.1 Data Sources

As a result of the different departments, the multiple stakeholders and elements of the MRO process, there are two main data sources that are relevant to this project: the first and major one is AMOS (Airline Maintenance & Operational Systems), which is a software solution developed and maintained by SWISS Aviation Software Ltd.. The engineers, planners and technicians use it on a daily basis to carry out and document their work. Hence, the information being stored here is very detailed and also sensitive as it includes, among other data, personal information on the employees. To summarize, AMOS can be regarded as an MRO process tailored ERP system. As in other systems of such kind, the data is thereby stored in a relational manner, linking different tables with each other via foreign keys.

The second system, Celonis, builds upon AMOS and creates value by analyzing the process flow of a case (= work order) based on its electronic footprints in the system. The three major things Celonis requires to build analyses are an unique identifier of the case, an activity / action applied to it and the respective timestamp when this happened. Each case can be further enriched by an unlimited amount of reference data. This could consist of information about the material required to perform the work, the technician's qualification to do so as well as information about the aircraft's type, manufacturer or flight hours. Thereby, Celonis provides the user with a quite holistic view while analyzing the entire process flow in a very user-friendly manner. Building these analyses is supported by Celonis' Process Query Language (PQL) – a SQL dialect with additional process-specific functionalities.

2.2 Preprocessing and Feature Engineering

As described in chapter 1.1, the goal of this project is to support the planning department by predicting the future process flow in order to prevent incomplete maintenance events and reduce process friction in the sense of touching a work order again. In the case of an incomplete maintenance event, another assignment is inevitably coupled to it, as depicted in the upper process flow of Illustration 7. If this information would have been available to the

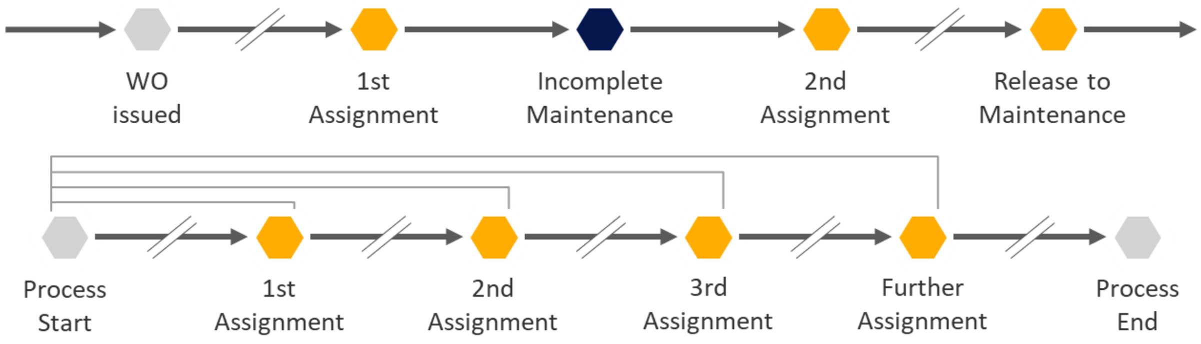


Illustration 7: schematic representation of process flows

planner in the moment of the first assignment, he could have steered against it, e.g. by assigning the work order to another work package on a later date.

But also at other stages of the process, the planner faces a very similar if not identical decision when assigning a work order to a work package. However, the planner (as well as the models later on) can only refer to the information being available at the moment of the assignment. Therefore, and also in order to increase the amount of data to train on, we split each work order based on the amount of its assignments into subcases. Thereby, the assignment activity's timestamp is selected as the cut-off criteria in order to avoid data leakage. In other words, only information being entered into the system no later than the assignment itself is taken into account. This constraint is applied to process flow as well as to reference data which also contains respective timestamps, e.g. the creation timestamp of a work step. The lower process flow of Illustration 7 depicts how a single case is split up into four subcases and which scope of information (including reference data) is considered.

This approach of processing the same work order multiple times over its process lifecycle in different ways breaks with the fundamental assumption of Celonis that every case is unique to the retrospective analysis of the process. Two additional things which make it challenging to use Celonis as the primary data source are the following: first, Celonis' PQL does not allow subqueries which makes it quite difficult to achieve the desired result in many cases. Second, Celonis' ease of use for the majority of its users is based on a fixed data model with predefined joins among the underlying tables. This circumstance in particular makes it impractical to achieve the above-mentioned result. Consequently, we transformed the AMOS raw data directly in SQL Server and used Celonis mainly for data and process visualization purposes.

	Feature Engineering	R-GCN
Dataset Size	4 MB	582 MB
Scope	70 DB columns, 167 features	133 DB columns
Datatype	Handcrafted features	Raw data as graphs
# Cases	133,664	

Table 1: Dataset statistics

Split	# Cases	# Incomplete Maintenance	# Deassignments
Train (80%)	106,931	4,239 (3.96%)	34,397 (32.17%)
Validation (10%)	13,366	575 (4.30%)	4,253 (31.82%)
Test (10%)	13,367	513 (3.84%)	4,346 (32.51%)
∑ Whole dataset	133,664	5,327 (3.99%)	42,996 (32.17%)

Table 2: Both datasets use the exact same train/validation/test splits. Only around 4% of the cases lead to Incomplete Maintenance, while the amount of Deassignments with 32% is quite high

To account for the data requirements of the different Machine Learning models, we crafted two different datasets that contain the exact same work order cases, but in different format. The feature engineering dataset, that contains manual aggregations of datapoints into fixed sized feature vectors, and the R-GCN dataset that contains each work order case as a graph. From

Table 1 it can be seen that the R-GCN dataset is much bigger as it contains the raw data from the database without any aggregations. To ensure comparability across the approaches it is also crucial that the same train/validation/test splits are used. Table 2 shows that the training set contains around 106k cases and that only very few of these are labelled as Incomplete Maintenance.

2.2.1 Feature Engineering

As already touched upon in chapter 1.2, two of our three selected baseline and prediction models, namely Logistic Regression, Neural Networks and the Decision Trees, require feature engineering. Thereby, the models expect a fixed size feature vector to train on. The starting point of this vector is logically the table which stores the work orders. In a relational database environment with many 1:N cardinalities between different tables, this requires aggregating information to achieve such a fixed size vector. A quite simple example from section 1.1 is the 1:N cardinality between work orders and work steps. As there exists another 1:N cardinality between a work step and e.g. a resource, one could speak of a “1:N:M” relationship between the work order table and the resource table.

Any aggregation comes along with a loss of information. Thus, retaining as much information as possible needs to be achieved in a different way. For instance, encoding the material groups and counting the occurrence of the material group XYZ within a work order solves this problem to a certain degree. For numerical data, e.g. the required amount of a particular part, other forms of aggregation (such as SUM, AVG, MIN, MAX, etc.) are obviously also suitable.

In the end, through applying the domain, process and data knowledge acquired in the first phase of this project, we manually crafted a feature vector consisting of 167 features in total. To outline some of these features (without attaching any importance to them), the information provided to the machine learning models covers the following, among others: the time passed since the process started (timestamp based), how many incomplete maintenance events occurred in the past and due to which reason, the last activity before such an event, the (average) amount of parts within a work order, the categorical encoding of the aforementioned material groups and in the similar way the required qualifications of the line workers.

2.2.2 Preprocessing for R-GCN

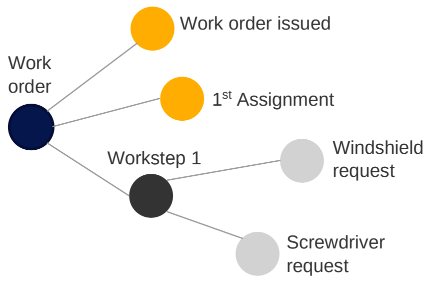


Illustration 8: Examplary graph input from the R-GCN dataset

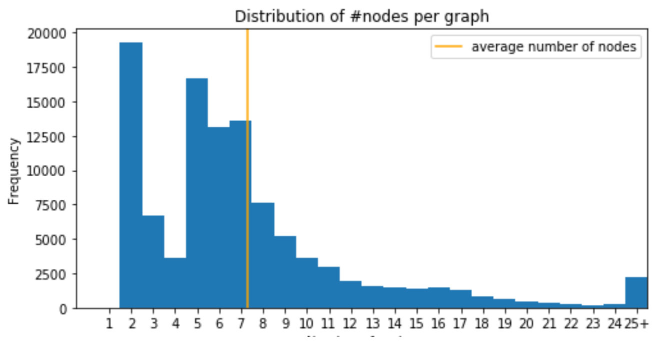


Illustration 9: Distribution of graph sizes in the R-GCN dataset

As opposed to Decision Trees, Logistic Regression and Neural Networks, the R-GCN can directly operate on the relational data from the database. It expects as input a graph comprised of nodes and edges. In our case, each node represents a row from a table and each edge represents a link between two table rows. Additionally, edge types correspond to the table that a linked row belongs to. Illustration 8 shows an example. Since there is no manual feature engineering involved in this process, the R-GCN follows an *end-to-end learning* paradigm.

To process all relevant information for a work order case, the database tables are first mapped to Python objects via *SQLAlchemy*[1] and then transformed into graphs with the *DeepGraphLibrary*[2]. Additionally, we encode categorical variables and calculate timestamp features. In the end, each work order case is represented as a graph with the datapoints attached to the nodes.

Illustration 9 shows that most of the graphs are rather small with 7 nodes on average. The high number of graphs with 2 nodes can be attributed to the fact that many work orders do not have much information available at the time of their 1st Assignment. Therefore, these cases are hard to predict as only rudimentary information is available, such as the airport at which the aircraft will be maintained, while details about the required parts and planned resources are missing.

2.2.3 Exploratory Analysis of the Dataset

As logistic regression and neural network requires a feature vector to represent numeric characteristics of an aspect of an object, exploratory analysis of the dataset is carried out. We compared the distribution of positive and negative examples over certain features of the training datasets. This helped us to identify whether the distribution makes sense and to analyze whether the positive examples contain a higher rate of extreme values. The joint plot for distinct resources (aircraft, staff, hangar, tooling) vs. total planned time (manhours needed) for the work order, generated over the distribution of incomplete maintenance labels, is depicted below.

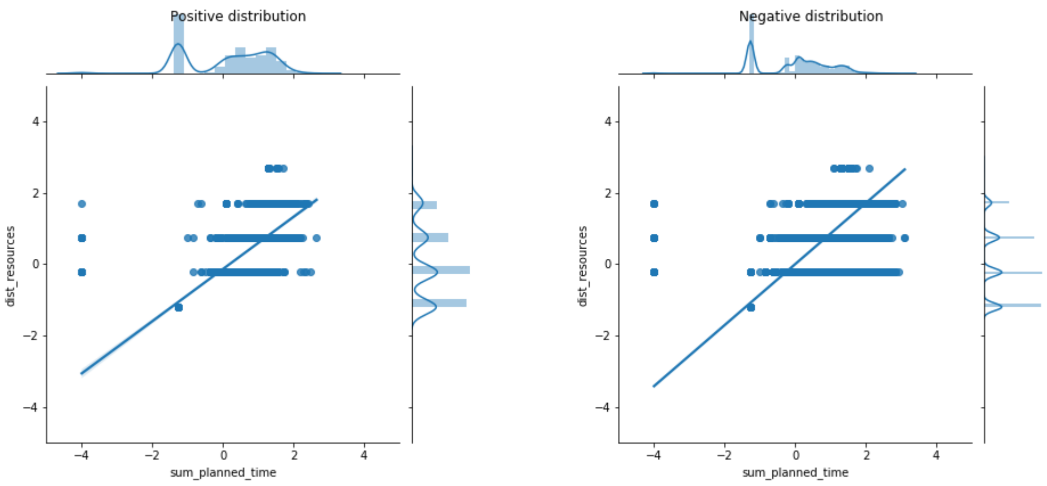


Illustration 10: Joint plot for distinct resources vs. planned time for incomplete maintenance

In the case of predicting incomplete maintenance, most of the features for positive and negative samples are very similar. In other words, the positive samples do not have extreme values or a clear separate region, making it difficult to predict incomplete maintenance with a higher accuracy.

[1] https://www.sqlalchemy.org/
[2] https://www.dgl.ai/

This joint plot provides a good insight of data but only two features can be compared at a time. To understand which features can be generally helpful in predicting Incomplete Maintenance we performed statistical independence tests. In detail, we employed Chi-squared tests for all categorical features and Analysis of Variance (ANOVA) for all numerical features to find input variables that are highly dependent on the prediction target. The outcome of these tests is a p-value for every variable that indicates the probability of observing the dataset instances under the assumption that the variable is independent of the target variable. As such, the p-value can be used to select good features by filtering out variables above the significance level. The detailed analysis of all features can be found in the appendix.

3 Models

Before outlining the different model's implementation and how we improved them, we will define a set of metrics in order to measure and compare the results of the four different models deployed.

3.1 Metrics Used to Compare Algorithms

As we aim at solving a binary classification problem, we only considered metrics that are commonly used for these types of models.

Firstly, it is important to note that the outcome variables for the dataset are quite imbalanced as explained before (see section 2.2.3). This has several implications for the evaluation metrics used as some metrics are highly sensitive to imbalanced datasets. For instance, a metric that is commonly used is the accuracy ($\frac{TP+TN}{TP+TN+FP+FN}$). However, in a highly imbalanced dataset, the accuracy metric will naturally be quite high e.g. if the dataset consists of 90% negative values and 10% of positive values. In such a case, a model that does not learn from the data but classifies all cases as negative thereby reaches an accuracy of 90% which seems to be a good performance – although it is not. Hence, we decided to use metrics that are not affected by the imbalance of the data or at least take the imbalance into account.

Secondly, we have to take into account that there might be different implications and (negative) consequences depending on which error the classification model has. In a classification problem, there are always two basic errors for the classification: False Positives (FP, also: Type I error, alpha-error) and False Negatives (FN, also: Type II error, beta-error). A false positive error means that the model classified a case as positive although, in reality, the case is negative. In the case of CLH, this translates to classifying a case as incomplete maintenance – although there will not be any issues during the process would constitute an FP. As opposed to this, a false negative describes a case that is wrongfully classified as negative – actually, this case is supposed to be positive. At CLH this would mean to predict a case that is not going to work out well as complete maintenance. Thus, this case will be handled as usual and the negative consequences arising from an incomplete maintenance event cannot be prevented.

Based on these considerations, we decided to use the confusion matrix as one of the metrics for evaluation. To measure the precision ($\frac{TP}{TP+FP}$) as well as the recall ($\frac{TP}{TP+FN}$) in one metric, we decided to use the F1-score ($\frac{2*Recall*Precision}{Recall+Precision}$) as a leading metric. This allows us to make models that have high recall and low precision comparable to models with a low recall and a high precision as both cases might occur in the models. It is important to note that the F1-score takes FNs and FPs equally into account. However, from a business perspective, these might not be equally important. Therefore, we decided in accordance with CLH to use the F2-score

($\frac{5*Recall*Precision}{Recall+4*Precision}$) as a leading metric as it allows us to compare the models based one score that takes recall as well as precision into account and weighs recall and therefore the false negatives higher than precision. This is important as the business impact of false negatives for CLH is higher than the impact of cases that are false positives. However, the third use case (predicting work order success) is reversed to the others logically (the planner has to intervene if “0” and not “1” is predicted). Therefore, also the business impact of the FNs and FPs is reverses and we use the F0.5-score ($\frac{1.25*Recall*Precision}{Recall+0.25*Precision}$) as a leading metric as it is the logical counterpart to the F2-score.

Lastly, we used the area under the precision-recall curve (AUC PR) as an additional metric for the logistic regression, the neural network as well as the R-GCN model. This metric is - opposed to the often used area under the receiver operating characteristic curve (AUC ROC) - well suited for imbalanced datasets. The AUC PR allows us to measure the performance of the model across different classification thresholds – more precisely how well the model is able to distinguish between the two classes. As thresholds are only relevant for models predicting probabilities, i.e., Logistic Regression, Neural Networks and in the R-GCN, we do not report this metric for the decision tree models.

3.2 Selected algorithms and Methodology

In the following section, we explain the algorithms we defined, the general method we used for implementation as well as how we tried to improve each algorithm.

3.2.1 Logistic Regression

Logistic regression is a well-known basic approach in machine learning. Therefore, we decided to implement this simple algorithm to get a fast and profound understanding of the overall algorithm pipeline. This includes the data preparation to make the data appropriate for input criteria of the algorithm, the setup and training of the algorithm itself, as well as creating the predictions and evaluating them on certain metrics, mentioned in the section 3.1. Furthermore, logistic regression helps us to set a baseline for performance comparison with the other algorithms we have implemented.

The structure of logistic regression is very simple and is equivalent to a one layer neural network. Illustration 11 depicts the structure of the network as well as its underlying principle – the sigmoid function.

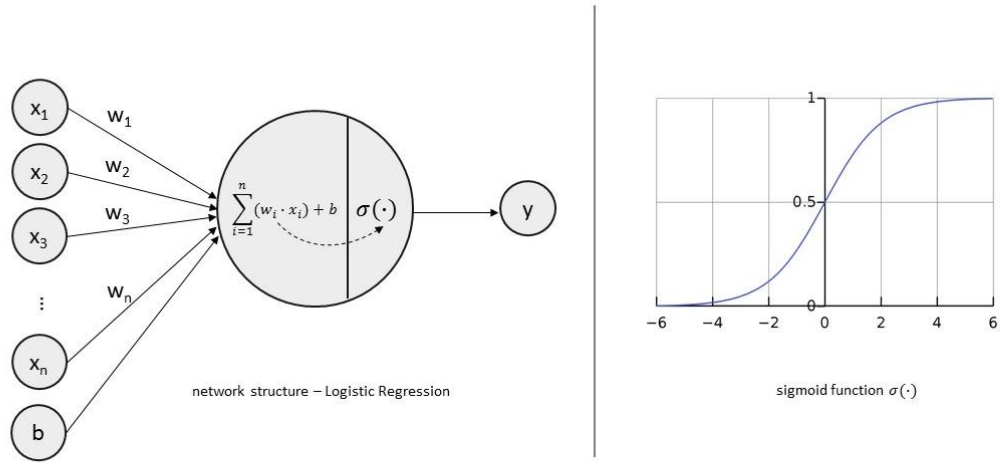

x_1
x_2
x_3
⋮
x_n
b
w_1
w_2
w_3
w_n
$\sum_{i=1}^{n}(w_i \cdot x_i) + b$
$\sigma(\cdot)$
y
network structure – Logistic Regression
1
0.5
0
−6
−4
−2
0
2
4
6
sigmoid function $\sigma(\cdot)$

Illustration 11: network structure of Logistic Regression and the sigmoid function.

Logistic regression typically consists of x_n input variables each with an assigned weight value w_n. The input variables are multiplied with its respective weight, are summed up and a bias is added. This result is passed on into the sigmoid function, which outputs a value ranging from 0 to 1. The output can be interpreted as a probability. Depending on which threshold is used on the probability, it is assigned either to class 0 equal to false or to class 1 equal to true. Usually one uses a threshold of 0.5. However, in the following paragraphs we will point out why this was not suitable for our use cases.

In our case, we have designed the network with around 336 learnable parameters. During training, the network computes its learnable parameters to be able to predict the target variable y of the desired use case. To deliver tailored results for each of the three use cases of this project, including incomplete maintenance, deassignments or work order success, we trained three different models. Since the use case of incomplete maintenance turned out to be of most interest, the following paragraphs describe the procedure for its application.

For the application of logistic regression for incomplete maintenance, we have witnessed various difficulties due to the imbalanced dataset. The dataset for incomplete maintenance includes 3.96 % positive cases (incomplete maintenance = true) and 96.04 % negative cases (incomplete maintenance = false). To overcome this issue we implemented four different methods: the bias, the class weighting, the method of oversampling and the threshold interpretation.

Bias

In this method, we set the output layer's bias to reflect the imbalance in the dataset. The default bias initialization without any regularization has a loss of about $ln(2)$. The correct bias b_0 can be derived from the following equations:

$$p_0 = \frac{pos}{pos + neg} = \frac{1}{1 + e^{-b_0}} \tag{1}$$

$$b_0 = -log_e(\frac{1}{p_0} - 1) \tag{2}$$

$$b_0 = log_e(\frac{pos}{neg}) \tag{3}$$

The initial loss with this initialization should be: $-p_0 \log(p_0) - (1 - p_0)\log(1 - p_0)$

The initial loss is found to be significantly less compared to naïve initialization. This way the model doesn't need to spend the first few epochs just learning that positive examples (True Positives)

Class Weighting

Since we want to predict incomplete maintenance and we do not have many positive samples to work with, we require the classifier to weight that few available positive examples heavily. It can be achieved by passing weights for each class through a parameter in the framework Keras. The model will be more attentive towards the examples from an under-represented class. Applying class weighting to our use case of incomplete maintenance, we compute the class weights as follows:

$$Weight\ for\ class\ 0 = \frac{1}{all\ negative\ samples} \times \frac{total\ number\ of\ samples}{2.0} \quad (4)$$

$$Weight\ for\ class\ 1 = \frac{1}{all\ postive\ samples} \times \frac{total\ number\ of\ samples}{2.0} \quad (5)$$

We receive the following results:

```
Weight for class 0: 0.52
Weight for class 1: 12.61
```

Class weighting can affect the stability of the training depending on the optimizer. Optimizers like Stochastic Gradient Descent may fail because its step size is dependent on the magnitude of the gradient. Adam is however unaffected by scaling change and found to be optimum in this case. As a result of the scaling, the class weight changes the range of loss, making it incomparable with same model not using class weights.

Oversampling

this method, we generate a balanced dataset by replicating the minority class until an even distribution is reached. ThThe total dataset size is larger, and each epoch runs for more training steps. Instead of showing each positive example with a large weight (as in case of class weight), they are shown with a small weight in many different batches each time. The oversampled data provides a smoother gradient signal, making it easier to train the model.

To use this dataset, we need the number of steps per epoch, which defines the number of batches of samples to be used in one epoch. The value assigned will trigger the completion of one epoch and starting of the next epoch. Oversampling often leads to overfitting due to the duplication of samples, which is why we implemented another hyperparameter “steps per epoch” which allows a final control over training by breaking up the epoch earlier. For example, if we have a “steps per epoch” equal to 5 with a batch size of 10 per epoch, the epoch will terminate after 5 batches due to the steps per epoch.

To reduce the issue of imbalance in the dataset for incomplete maintenance, we identified that the method of class weighting and oversampling were significantly close in terms of scores. Oversampling has a little higher accuracy compared to class weights.

Threshold Interpretation

An additional method to increase the performance of the logistic regression as our baseline model is the implementation of a threshold interpretation. In the above section, it is already said that for a sigmoid function one usually uses a threshold of 0.5, to classify everything below the threshold as 0 or false and everything above as 1 or true. However, due to the imbalance of the dataset our threshold is shifted. Plotting the F2 classification performance of our model with regard to the threshold ranging from 0 to 1, we see, that the peak of the threshold does not lie at 0.5. It is shifted towards a smaller value. In our example in Illustration 12 the optimal threshold lies at 0.2. Considering this insight, we were able to improve logistic regression as a baseline model.

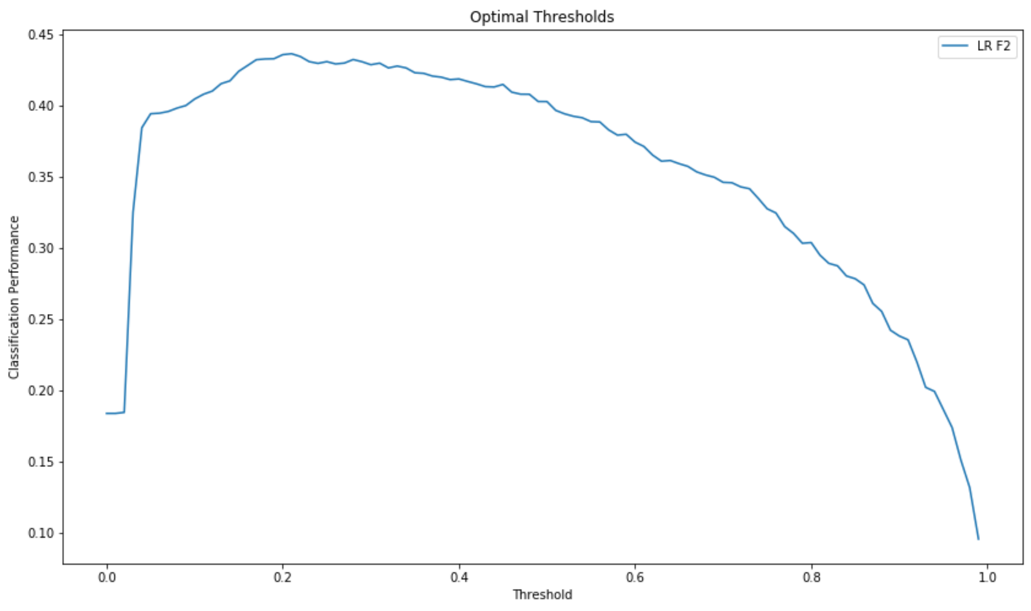


Illustration 12: Threshold of logistic regression over F2 classification performance.

With this pipeline, we successfully built our baseline for comparison and could then focus on the more advanced algorithmic approaches.

3.2.2 Neural Networks

The Neural Network builds upon the baseline of the logistic regression including the implemented methods of the bias, the shared weights, oversampling and the threshold interpretation.

For the best neural network structure, we designed the network with an optimal depth of 13 layers. The network includes L2 regularization in the first layer and dropout implemented in every uneven hidden layer of the network. We chose the number of neurons per layer, such that we gradually decrease the parameter number the closer we reach the output layer. We identified that big steps between the parameter numbers due to the number of neurons of two following layers, has a negative effect on the model performance. The optimal model shape is shown in the appendix.

After we find the optimal model structure, we perform random hyperparameter tuning to increase the performance of the network. Here we focus on parameters such as the learning rate, the batch size, strength and penalty of regularization and sensitivity of early stopping. Furthermore, we identify number of thresholds for auc influences the performance of the algorithm. This parameter (number of thresholds for auc) helps in approximating the true auc more accurately by controlling the degree of discretization with large number of data samples. We see that tuning this number increases the quality of the approximation and therefore our overall model performance in all three use cases.

In the final step of tuning the neural network, we have implemented the SHAP library [4] for feature interpretation of our predictions. This tool enables us to backtrack what importance which feature has to compute a certain prediction. Considering this insight, we double-check the relevance of each feature with our insights we have from the CLH expert interviews. It appears that some features are irrelevant. We identified that our model achieved a higher performance on all three use cases after we remove the irrelevant features from the training set and therefore put further research in the implementation of the SHAP library for feature interpretation.

For the interpretation of the features, SHAP lists all important features which led to the prediction of that class label. However, the features don't point to the specific root cause of the problem, namely, incomplete maintenance. Hence, we crafted synthetic features for our model called as cross features or feature crosses. In Cross features, two or more features can be combined to provide predictive abilities beyond the range of normal features. It can increase the expressivity of the model and is an efficient strategy to learning highly complex models. We create cross feature x_3 by multiplying x_1 and x_2, $x_3 = x_1 x_2$ and treat this new feature like any other feature. Thus, x_3 encodes non-linear information and the model trains to learn the weights of this new synthetic feature by: $y = b + w_1 x_1 + w_2 x_2 + w_3 x_3$

After creating several cross features based on our domain knowledge, the length of the input vector was doubled and hence a new neural network model needs to be introduced. The new neural network model was trained with close to 2.8 million parameters. This increases the precision and accuracy of the neural network model compared with our initial neural network approach we explained in the above section. It also led to a better insight into core issues for incomplete maintenance and deassignments using SHAP interpretability plots. The results of the new model are in section 4.1. The results of the SHAP interpretability plots are in section 0. Further explanations about SHAP can be found in section 4.2.

3.2.3 Decision Trees

Decision Tree models are widely used in machine learning as they can be used to solve regression as well as classification tasks. Furthermore, their intuitive structure (see Illustration 13) with decision nodes step by step leading to the leaves is useful as they can be interpreted as opposed to other approaches in machine learning, e.g. neural networks. Hence, this approach is one of the selected approaches used to predict the three different use cases. In general, we used sklearn as a framework for the decision tree model as it is one of the most commonly used frameworks and offers various possibilities to adapt and thus improve the decision tree model.

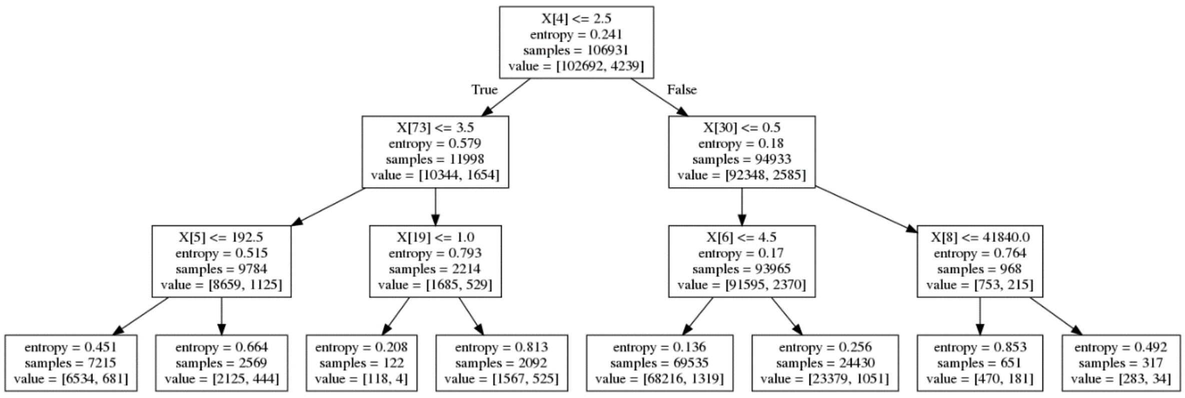


Illustration 13: Exemplary decision tree – the data set is split at each node until it reaches the lowest level and is translated to a classification

The input for the decision tree model is the dataset that resulted from the feature engineering stream. Hence, it contains 176 features and 133,664 observations (106,931 for training, 13,366 for validation, 13,367 for testing) in total. The input and the output (variables describing the three use cases) are separated into two data frames before preprocessing the data by using a label encoder. To do so, we use the label encoder included in sklearn to transform non-numerical labels to numerical labels and thus facilitate the decision tree model. Besides that, we fill the NA values with a numerical input to prepare the data for the decision tree model.

When implementing a simple decision tree as a common baseline for all the decision tree models using sklearn we find that the model shows perfect results (F2-Score of ~1.00) on the training data set and a rather weak performance on the validation dataset (F2-score of ~0.17). This indicates that the model is able to learn to predict the outcome to some extent from the input data but currently heavily overfits. In the following, we describe different approaches that we used to improve the model performance and to prevent overfitting: hyperparameter optimization, random forest, and feature selection. These approaches are exemplarily described using "Incomplete Maintenance" as the target variable as the process for the other two target variables was identical.

Hyperparameter optimization:

The framework sklearn allows the adaption of different parameters of a decision tree model. In the following, we focused on the most important ones: the criterion for splitting and the selected split, the maximal depth of the decision tree and the number of minimal nodes per leave. In general, we adapted the hyperparameters step by step, starting with one and then applying the best setting to the model we used to determine the next parameter. The criterion for splitting in sklearn can be set to "entropy" or "gini". Based on this setting, the model uses different calculations to evaluate the different splitting options. Hence, this parameter can highly impact model performance. When comparing the two options, we find that using "entropy" yields slightly better results.

Secondly, also the splitter can be set to "best" or "random". If this option is set to "best", sklearn uses the most relevant feature to split the tree on a certain node. Else, the attribute to split on is selected randomly. However, we either find no difference between the two models differing only regarding the splitter or only a slight tendency towards "best". Therefore, we decided to keep the option "best" which is also the default option for this parameter in sklearn. The maximal depth of a tree is one of the most influential variables on a decision tree as it determines among other parameters how granular the model is, more precisely, how many nodes it uses. Theoretically, the tree can have more than hundreds of nodes. Yet, a higher depth of the tree might also result in overfitting as the tree tries for each level to grasp the information of the training data set as good as possible. Hence, it is reasonable to set a maximum depth for the tree. In order to still ensure the interpretability of the final model, we ran the model with a depth ranging from 3 to 30. The high interval size regarding the F2-score of >0.2 for the models with varying maximal depth illustrates the importance of this parameter for the decision tree model. The best results are achieved when aiming for a maximal depth of about 14.

Lastly, we optimize the parameter that determines the minimum number of observations per leave. In doing so, we aim at preventing overfitting as by default this parameter is set to 1 – meaning that there can be leaves that only exist to classify one case. If there are some leaves that have such a low number of observations, this can easily lead to overfitting as the final decision tree model then resembles the structure of the training data too well. We run different models for a minimum number of observations on intervals ranging from 1 to 2,000. We find that the best performing model regarding the F2 score only has a minimum of 5 observations per leave. Besides that, we find that the F2 score highly drops if the number of minimum observations per leave exceeds.

All in all, optimizing these parameters increases the performance of the decision tree model on the validation dataset by ~0.07.

Random forest

A common approach to prevent overfitting in machine learning is to run a so-called random forest model instead of a single decision tree. A random forest model is usually more robust to overfitting as it does not only run a singular decision tree model but instead a large number of decision trees. When the model has to classify a case, the predictions of all these different decision trees for the respective case are calculated and compared. The final prediction is the class that was predicted by the majority of the decision trees in the random forest model. When fitting the random forest model without hyperparameter optimization to the training data and then predict the validation data, we find that the results as compared to the single decision tree are only slightly better (~ +0.02 F2-score). Hence, the random forest model is not better with regards to the performance as compared to a hyperparameter optimized decision tree.

Feature selection

Feature selection is a powerful approach to boost the performance of decision tree models as it allows the exclusion of unimportant features. This can especially improve the performance if the dataset contains many different variables that might not be really linked to the target variable in reality and, therefore, induce randomness or confusing random correlations in the decision tree model. To select the features, we first determined the feature importance with sklearn as well as by implementing the decision tree model again on another framework (catboost) as this framework offers additional functions to determine feature importance. In sklearn as well as in catboost we selected the most important for the decision tree and then excluded with additional information on the process and the interpretation of the variables the features that are unimportant (importance <1) and then decided to run the model with the top features starting at 5 at gradually increasing.

However, in sklearn nor running a simple decision tree model neither running a decision tree model with hyperparameter optimization showed clearly better results than the model that took all features into account. In catboost, the results even worsened a bit (~ -0.05 F2-score). Therefore, we decided to also approach the feature selection from a process view and exclude features that are not important from a process view or cannot be changed (as these features limit the options for process steering). Doing so also did not result in any major changes, which might hint at the complexity of the data in the model. Yet, we discuss the results and implications of the feature importances not here - but in chapter 4.

Summary

We conducted all these optimizations for the three different use cases to receive a solid performance of the decision tree models. However, the issue of the overfitting of the decision tree model could not be solved completely.

3.2.4 Relational Graph Convolutional Networks (R-GCNs)

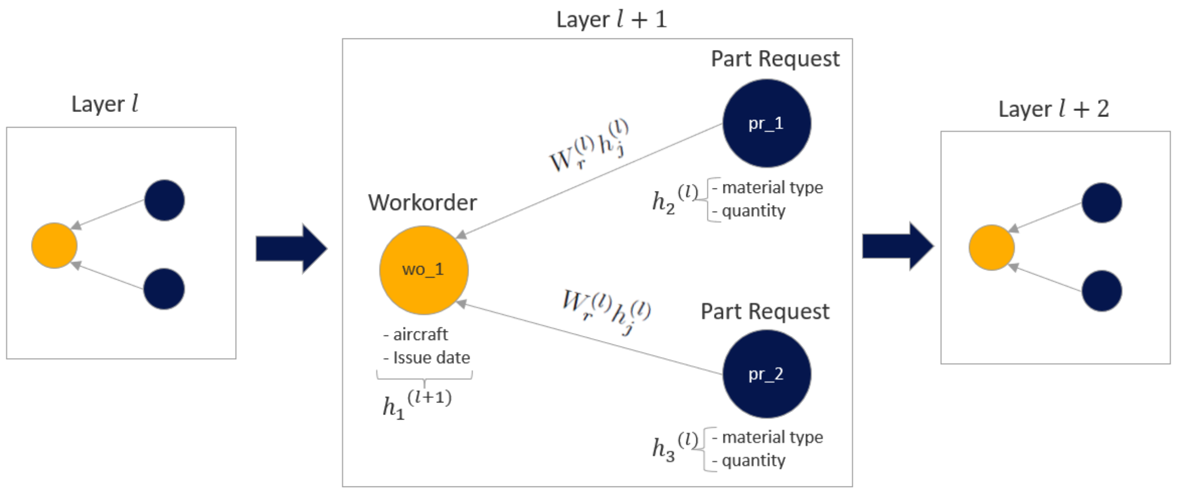


Illustration 14: Agglomeration procedure of the R-GCN. Information from neighboring nodes is pulled and weighted by the learnable matrices $W_r^{(l)}$. Using multiple layers increases the receptive field of each node.

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \qquad (6)$$

Equation 6: Layer-wise update rule of the R-GCN

The Relational Graph Convolutional Network (R-GCN) is an extension of regular Graph Convolutional Networks (GCNs) for relational data. Processing is done by *passing messages* between connected nodes as depicted in Illustration 14. In detail, a node i agglomerates information from all neighboring nodes $\mathcal{N}_i^r$ by considering the neighboring node's relation type r. In our case, the relation type corresponds to the table of the linked row, e.g., whether it is a part request or a past activity etc. The information from each neighboring node $h_j^{(l)}$ is pulled, multiplied with a relation-specific matrix $W_r^{(l)}$ and accumulated in a normalized sum to create a representation $h_i^{(l+1)}$ that contains information from all neighboring nodes. This procedure essentially mimics how Feature Engineering works but differs in that the agglomerations do not have to be specified manually but instead are learned by the network in terms of the weight matrices $W_r^{(l)}$. Furthermore, to ensure that a node does not "forget" its own information self-loops are added via $W_0^{(l)}$. In the end, a nonlinearity σ is applied and the procedure is repeated in a layer-wise fashion to enable aggregating information from non-adjacent nodes. In our case, a 2-layer network usually is sufficient as the input graphs are small and contain at most 2 hops between any pair of nodes.

To feed the data from the database into the network, additional preprocessing is necessary. Table rows contain more than one relevant datapoint and to combine these into a single vector representation per node we simply concatenate all data points of a relation. For categorical values, we additionally use embeddings. The resulting vectors vary in size and since R-GCNs require the data representations $h_j^{(l)}$ to be of equal length, we apply an additional linear transformation per relation to project the concatenated vectors into a common input space, which comprises the inputs $h_j^{(1)}$ for the first layer. In total, the R-GCN then has roughly 285k parameters.

As such, the R-GCN is able to learn which aggregations from neighboring nodes are useful via the matrices $W_r^{(l)}$ and can account for the heterogeneity of datapoints from different tables, which renders it well-suited for the task at hand.

4 Results

In this section, we present the results for all three use cases and additionally outline the use in practice of these cases. This includes an estimation of the value of this use case for the business - whenever possible.

4.1 Model comparison and business impact

We compare our models with respect to F1- and F2-Score as well as the Area under the Precision-Recall Curve (AUC) on the test partition. For all 3 metrics, higher numbers indicate better performance. The F2-Score is a variant of the F1-Score that weights FPs less, which makes sense in the case of predicting Incomplete Maintenance or Deassignments where the positive class represents undesired events and raising a flag too many times is preferred over missing an instance. In contrast, the AUC measures how well a model addresses the trade-off between Precision and Recall, which only applies to models that predict probabilities (Logistic Regression, Neural Networks, R-GCN) and thus can be calibrated by choosing a threshold. It might be that model A has a higher AUC but a lower F2-Score than model B, if model B only performs well considering a specific weighting of Precision and Recall, while the predictions of model A are more often correct for different calibrations.

4.1.1 Incomplete Maintenance

Model	F1	F2	AUC	TP	TN	FP	FN
Logistic Regression	19.25	34.22	14.58	270	11166	1688	243
Neural Network	28.71	41.73	31.53	307	11535	1319	206
Decision Tree	12.08	9.07	-	39	12796	58	474
R-GCN	37.24	46.45	39.67	340	11586	1268	173
+ comb. train/valid	43.56	49.53	45.71	303	12150	704	210

Table 3: Results of the models for predicting Incomplete Maintenance. AUC reflects the area under the Precision-Recall curve. For models that predict probabilities, the confusion matrix is obtained by choosing a decision threshold that optimizes the F2-Score, where the positive class corresponds to Incomplete Maintenance.

Incomplete Maintenance is hard to predict as relationships that cause it are intricate and subtle, sometimes maybe even random (cf. section 2.2.3). Additionally, we do not have detailed data about part ordering and shipping, which was often mentioned by the experts to be related to Incomplete Maintenance. Furthermore, the use case poses an extremely imbalanced classification problem with only 4% positive cases. While dataset imbalance is challenging in general, in our case it also leads to a situation where the models have to detect general patterns for Incomplete Maintenance from only ~4200 positive examples, which is a marginally small example base to learn from. Having a bigger dataset would certainly simplify the task.

The aforementioned challenges are reflected by the comparably low performances of all models on this use case, as seen in Table 3. Decision Trees struggle to predict Incomplete Maintenance and perform even worse than the Logistic Regression baseline. The R-GCN performs best on all 3 metrics, with a small, yet significant margin to Neural Networks. This indicates that the manually crafted features for Logistic Regression, Decision Trees and Neural Networks are indeed useful, but only the Neural Network is powerful enough to utilize them for the task. However, the aggregations learned by the R-GCN seem to be more viable to predict Incomplete Maintenance than the manually crafted features. Furthermore, it can be seen that the models would perform better with a bigger dataset. For example, the R-GCN can improve by a few points on all metrics, when it is trained on the combined data from the train and validation partition. Given the overall low performance scores however, none of the approaches are good enough to apply their predictions as they are. Even the R-GCN cannot perform better than detecting 60% of all Incomplete Maintenance Cases with a precision of 30%. Should the model be used in practice, an expert would have to double-check predicted cases to filter out false positives.

All in all, predicting incomplete maintenance, has a very high impact on the business (e.g. compared to deassignments) as predicting and thus possibly preventing incomplete maintenance has mainly three direct benefits: reducing the number of deassignments, delays as well as cancellations of flights. To calculate the final business impact of incomplete maintenance cases, we take all of these three levers into account. The number of deassignments reduced is a reduction of process costs and is detailed below in Use Case 2 (Deassignments). As opposed to this, the benefits of the reduction of delays and cancellations are a reduction of costs generated by the process results. We calculate these reductions by estimating the number of delay minutes and cancellations caused by technical issues resulting from the MRO process in 2019. We then transferred the MRO process's share of total delays and cancellations as well as the average costs caused per cancellation and delay/minute to an estimated reduction that could be realized by preventing some of these events as a result of the prediction model.

However, we did not only consider the benefits of the predictions into account. To calculate the full business impact, we included a cost perspective as well – e.g. by the extra work that is required to actually turn the information of the prediction into action and to prevent incomplete maintenance. Here, we assumed that this requires 50% of the time that a planner needs to schedule a whole new work order. Based on these considerations, we calculated the overall impact of all four fields of the confusion matrix: TP, TN, FP and FN. Especially, the FNs have a high impact on the overall return of this use case as they cause extra work – that is actually not necessary. All in all, our calculations show that with the current performance, this use case is able to save process costs as compared to today.

Furthermore, a positive impact of this use case is the cost reduction on the long term. This reduction could, for instance, be realized by a reduction of the spare fleet size as there are less delays and cancellations and, therefore, fewer spare airplanes are required.

4.1.2 Deassignments

Model	F1	F2	AUC	TP	TN	FP	FN
Logistic Regression	55	59	56.73	4160	1953	7068	186
Neural Network	59.07	75.84	73.47	4065	3669	5352	281
Decision Tree	55.69	55.21	-	2386	7185	1836	1960
R-GCN	70.65	78.39	81.00	3904	5407	3614	442

Table 4: Results of the models for predicting Deassignments. The confusion matrix is obtained by choosing a decision threshold that optimizes the F2-Score, where the positive class corresponds to Deassignments.

For the use case of deassignments all models perform better then on the previous use case of incomplete maintenance, as the dataset for deassignments is less imbalanced. Furthermore, the numbers show, when comparing the three models with regard to the baseline model of logistic regression, that the neural network (NN) and the R-GCN strongly increase their performance whereas the decision tree even stays below the performance baseline of logistic regression with regard to the F2 and AUC score. The decision tree cannot represent the complexity of the task and therefore performs almost as good as the baseline model of logistic regression, which is also a simple model and does not have the power to represent high data complexity with its model structure. On the other side, the neural network as well as the R-GCN outperform the baseline model for this use case, because both deep models have a high number of learnable parameters, which are able to represent the interconnections and relations within the data. The R-GCN outperforms the NN, because it has the advantage to use data as a graph and is therefore not relying on handcrafted features. However, the performance of the NN highly depends on the handcrafted features it uses for training. Spending more time on the feature preparation for the NN could increase its performance.

The business impact of the use case of deassignments is the time a planner needs to reschedule a case. The time is varying due to the complexity and the reason for the deassignment of a particular case. To calculate the concrete business impact for a deassignment we, therefore, used 50% of the mean of minutes a planner needs to assign a new case as an approximation. To calculate the actual costs of assigning a case we used the costs per manhour and the number of work orders assigned in the planning department as well as a factor for time the planners spend on activities other than planning (e.g. meetings, strategic planning, communication). Due to an AI based monitoring approach as it is introduced with this project, there is potential to lower the workload for each planner because the appearance of deassignments can be forecasted and thus these events can be prevented, for instance, by carefully reviewing the case again. This leads to a steadier and reduced workload for each planner and saves costs in the long run.

4.1.3 Work Order Success

Model	F1	F0.5	AUC	TP	TN	FP	FN
Logistic Regression	78.9	73.31	81.82	7692	1562	3297	816
Neural Network	77.61	82.23	89.14	6038	3845	1014	2470
Decision Tree	83.52	83.49	-	7110	3451	1408	1398
R-GCN	85.94	84.91	91.72	6544	3896	963	1964

Table 5: Results of the models for predicting Work Order Success. The confusion matrix is obtained by choosing a decision threshold that optimizes the F0.5-Score, where the positive class corresponds to a success.

In predicting work order success, the performance of all three approaches namely, neural network (NN), decision tress and R-GCN is found to be similar comparing the F0.5 score and area under the curve. The decision tree has performed better than the baseline model of logistic regression in this case. All models show very high scores, because the class of "work order success" is overrepresented in the data. However, this use case is of least importance to CLH because it does not help in identifying the bottlenecks in the process and has no significance for process optimization. Though this model has higher accuracy, one should be skeptical to implement it in a real-world scenario because predicting false work order success (FP) can cause potential disruption in planning process. This can possibly lead to more incomplete maintenance cases because the count of false positives samples is higher than the count of actual incomplete maintenance cases.

Predicting whether a work order will be finished successfully, has a smaller business impact as compared to the other two use cases. As it is easier to steer cases if one knows how big the impact of the issue that might occur is (e.g. will a de-assignment or incomplete maintenance occur), it is better to use case 1 and 2 to actively steer the cases instead of the prediction of work order success. Nevertheless, this metric is important as it can be used to track the overall improvement resulting from deploying the prediction model.

4.1.4 Summary

R-GCN performs comparatively better in all three business cases, closely followed by the neural network (NN). In this analysis, the decision tree could not surpass the accuracy of logistic regression for the two most impactful business cases of incomplete maintenance and deassignments. Decision trees show better performance when the targeted class is balanced or over-represented in the dataset. The improvement of neural network directly depends on the domain knowledge of the data to create helpful features to predict a certain class. Incomplete maintenance has the highest impact on the business, but it is very hard to predict, as it is a highly under-represented dataset. Further, due to the complex non-linear relationship of the data there is no clear boundary, which separates the incomplete maintenance class from other classes. However, R-GCN and the neural network can be used for feature interpretation to identify features that affect incomplete maintenance. The predictions for deassignments can reduce rework for the planners in assigning the work order again to a different work package. The predictions of work order success offer no business value to CLH.

4.2 Interpretation and Discussion of Results

To further improve our models and to get an understanding for the models' predictions we implemented several explainability methods. These can help to identify how important each feature is for the prediction which in turn allows to improve the models by dropping spurious features or to draw conclusions about the underlying process. For example, features that are important for predicting Incomplete Maintenance may hint at frictions in aircraft maintenance and are thus good starting points for CLH to improve the process.

We experimented with several explainability methods and in the end decided to use *Integrated Gradients* [5] for explaining single predictions and *SHAP* [4] for explaining what features the model deems important in general. Both methods assign a weight to every input feature which constitutes the explanation. SHAP has a strong theoretical foundation and calculates feature importances by decomposing the model's prediction $f(x)$ into a linear combination of the features:

$$f(x) = \phi_0 + \sum_i \phi_i x_i \tag{7}$$

where x_i is the i-th feature of input x and ϕ_i its importance for the prediction. These ϕ_i can be obtained by leveraging results from Cooperative Game Theory. In this interpretation, the x_i are players that all contribute to a common goal and we wish to find a fair allocation of the common profit $f(x)$ among them. This fair allocation is computed by measuring each player's usefulness in any combination with other players, i.e., how much the player's presence increases the common profit in a particular coalition. Averaging the player's contribution over all coalitions then constitutes their *Shapley Value* ϕ_i which is the fairest possible allocation. SHAP interprets these allocations as feature importances and thus provides a truthful explanation of the model's decision. The biggest disadvantage is the computational cost of calculating a feature's usefulness for all possible coalitions, which can pose a challenge if the model is very complex and the number of features is high. As this is the case for R-GCNs, we employ a different explainability method, Integrated Gradients, which provides the same output as SHAP, but does so in a slightly less theoretically grounded and less accurate way.

In the following, we analyze the explanations of a single R-GCN prediction. Making the model's decision for a single case more transparent helps the user to build trust in the system. For them, it can be a powerful tool to ultimately decide whether to accept a prediction and to figure out which levers to pull in order to steer against an incomplete maintenance. Furthermore, we analyze the predictions of the Neural Network on the whole dataset to get an impression of how the model operates globally which can hint at general inefficiencies in the maintenance process.

4.2.1 Explanation of single predictions

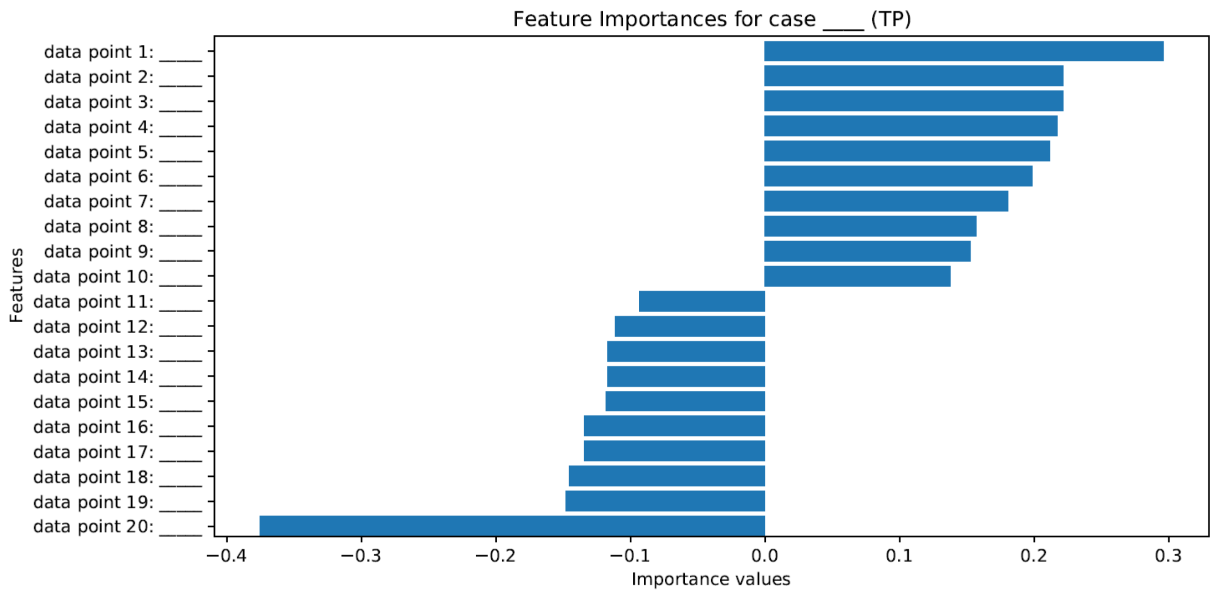


Illustration 15: The 20 data points that the R-GCN drew most attention to when correctly predicting Incomplete Maintenance for a particular case.

Despite the black-box nature of the R-GCN which we initially thought would make it inherently uninterpretable, we were able to extract feature importances for the model. Illustration 15 shows what inputs the R-GCN put attention to for a successful Incomplete Maintenance prediction. In this case, the model heavily used the knowledge that this case ran into incomplete maintenance several times already which makes it more likely that it will happen again. Apart from this statistical insight, the R-GCN also found that the particular airport at which the maintenance should be performed is important, as well as other datapoints such as the type of work order or specific requested parts. In particular, we could identify data leakage with this analysis, as *data point 20* in Illustration 15 appeared in almost all explanations and would always decrease the model's confidence in predicting Incomplete Maintenance. After consulting the technical documentation of the database, we found that this particular data point is only set after the work order has been completed. This is problematic in two ways: First, it gives away that the case will be successful. Second, it does not generalize as this data point would not be set in a live prediction setting but often had a value during training. By excluding this data point, we could thus improve the generalization capabilities of the R-GCN.

All in all, explaining single predictions of the R-GCN provides valuable insights. However, as the R-GCN operates on graphs, certain information such as whether the model detects patterns in the way how nodes are connected cannot be captured with this explainability method. Furthermore, as the input graphs vary in size, it is impossible to explain the model's decision in the general case. Therefore, we performed further analysis on the Neural Network which in terms of explainability profits from the features being hand-crafted and of fixed length.

4.2.2 Feature importance for incomplete maintenance and deassignments

The charts in the subsection explain which features are on average relevant for the use cases of incomplete maintenance and deassignments. For this evaluation, we use the neural network model with cross features. In order to protect the confidentiality of the results, we have anonymized the features in this section. A detailed description about the functionality of cross features is in section 3.2.2.

Deassignments

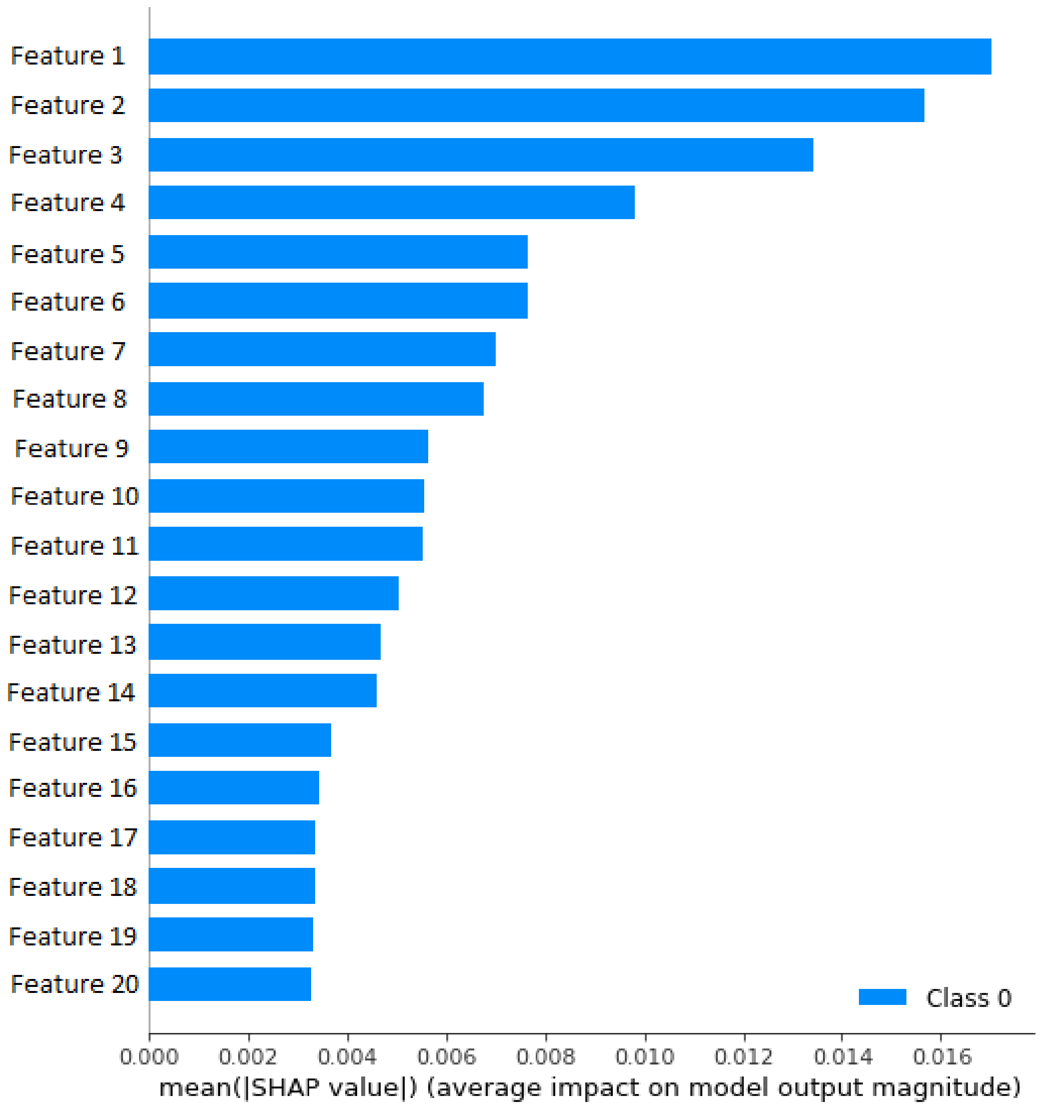


Illustration 16: Overview of relevant feature on average for the detection of deassignments.

For the use case of deassignments we can see from Illustration 16 that there are quite a few relevant features that cause a deassignment. This analysis was performed on 1000 cases. It was found that the deassignments happen very often in the context with a particular station.

Incomplete Maintenance

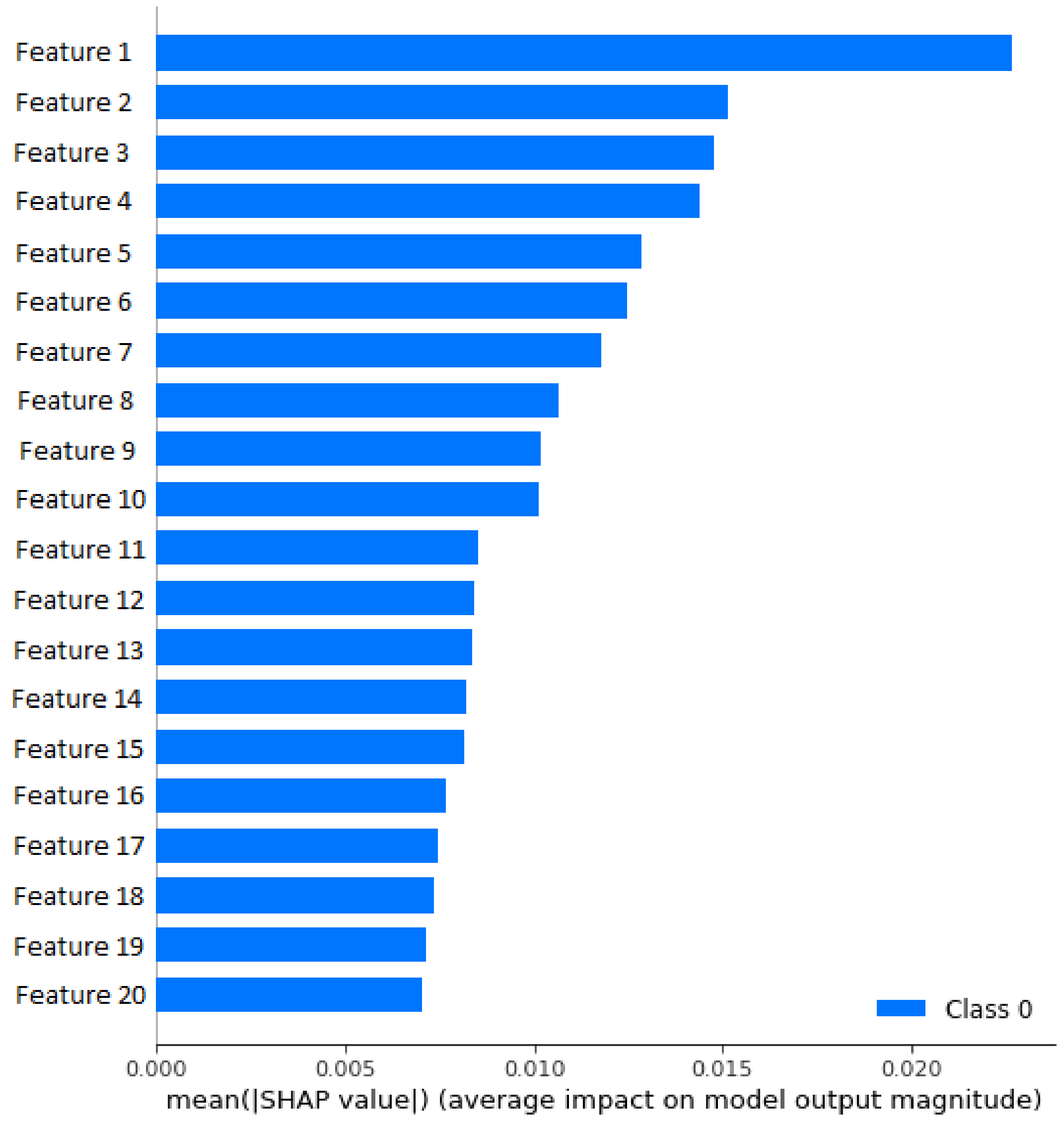


Illustration 17: Overview of relevant feature on average for the detection of incomplete maintenance.

From Illustration 17, we derive that there are several reasons why a particular work order is detected as incomplete maintenance. This analysis was performed on 575 cases. Initially, we noticed that one station was of higher relevance for incomplete maintenance. Rerunning the interpretation again with cross features, we found that the major factor affecting that particular station was the time allotted to perform the maintenance task.

5 Conclusion

All in all, we reached the overall goal which we defined based on the insights in the process as well as the business understanding and developed a predictive process management model to predict critical cases in the aircraft MRO processes. We achieved this by implementing four different approaches for ML models based on previous research on predictive process management: Logistic Regression, a Neural Network approach, a Decision Tree model and a Relational Graph Convolutional Network. To find and steer the critical cases in the MRO process we focused on three major use cases.

First, we predicted incomplete maintenance which posed major difficulties due to the highly imbalanced distribution of the variable. Hence, even the best performing model (R-GCN) shows a rather average performance (46.45 F2-score). Nevertheless, the model achieves the highest business impact as compared to the current process as it reduces replanning as well as delays and cancellations. The second use case is to predict a deassignment of a work order – more precisely a replanning. Here, the R-GCN as the best model shows good results (77.88 F2-score) and also allows for a cost reduction. Lastly, a prediction of the use case “work order success” can allow as a metric indicating the improvement of the process – meaning more work orders are completed without any deassignment or incomplete maintenance. For this use case, the best results were achieved (91.60 F2-score, R-GCN).

To allow for deeper insights on what impacts critical cases and to suggest variables that allow an active steering of the process, we also conducted an analysis of feature importances. Here, we find that some features such as a particular station to conduct the work order at are very important for predicting the outcome variables.

In the future, the models could be further improved by adding additional data (more precisely: more work orders to learn from as well as more information on part shippings and the ordering process) or by data augmentation i.e. creating artificially more cases to reduce the imbalance in the datasets. The R-GCN as the best performing model could be advanced by integrating textual information such as descriptions of the work steps to enrich it with more information on the particular work order. Furthermore, the feature importances could be evaluated in-depth with process experts to gain more insights in relevant features and features that could be included.

With regards to the business perspective that combines the initial process discovery stage with the final models, we can conclude that the project results are highly relevant. Firstly, the results prove that it is feasible to predict certain outcomes in the MRO process and thus to actively steer and thereby prevent outcomes that are not desirable from a business perspective. Secondly, the calculations of the business cases and the overall positive value of the three use cases – despite of the currently wrongly classified cases – hints at the high potential impact the implementation and further improvement of the models could have for Lufthansa CityLine.

6 Bibliography

[1] W. van der Alst, A. Adriansyah, A. K. Alves de Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. van den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B. F. van Dongen, M. Dumas, S. Dustdar, D. Fahland and D. R. Ferreira, "Process Mining Manifesto," in *International Conference on Business Process Management*, Clermont-Ferrand, France, 2011.

[2] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber and T. Lillicrap, "Relational recurrent neural networks," in *Advances in neural information processing systems*, 2018.

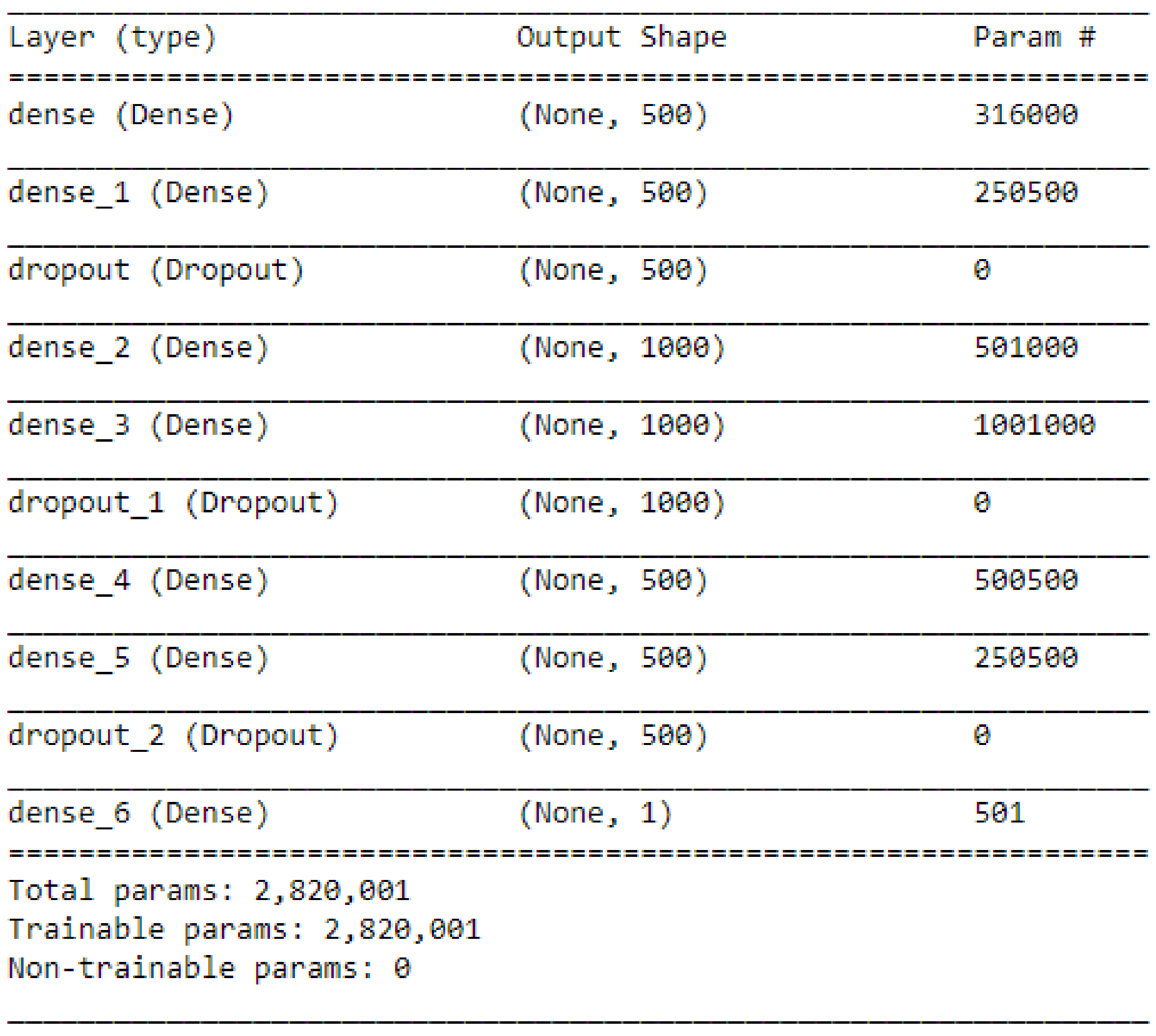
[3] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov and M. Welling, "Modeling relational data with graph convolutional networks.," in *European Semantic Web Conference*, Cham, 2018.

[4] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions.," in *Advances in neural information processing systems*, 2017.

[5] M. Sundararajan, A. Taly and Q. Yan, "Axiomatic attribution for deep networks.," in *International Conference on Machine Learning*, 2017.

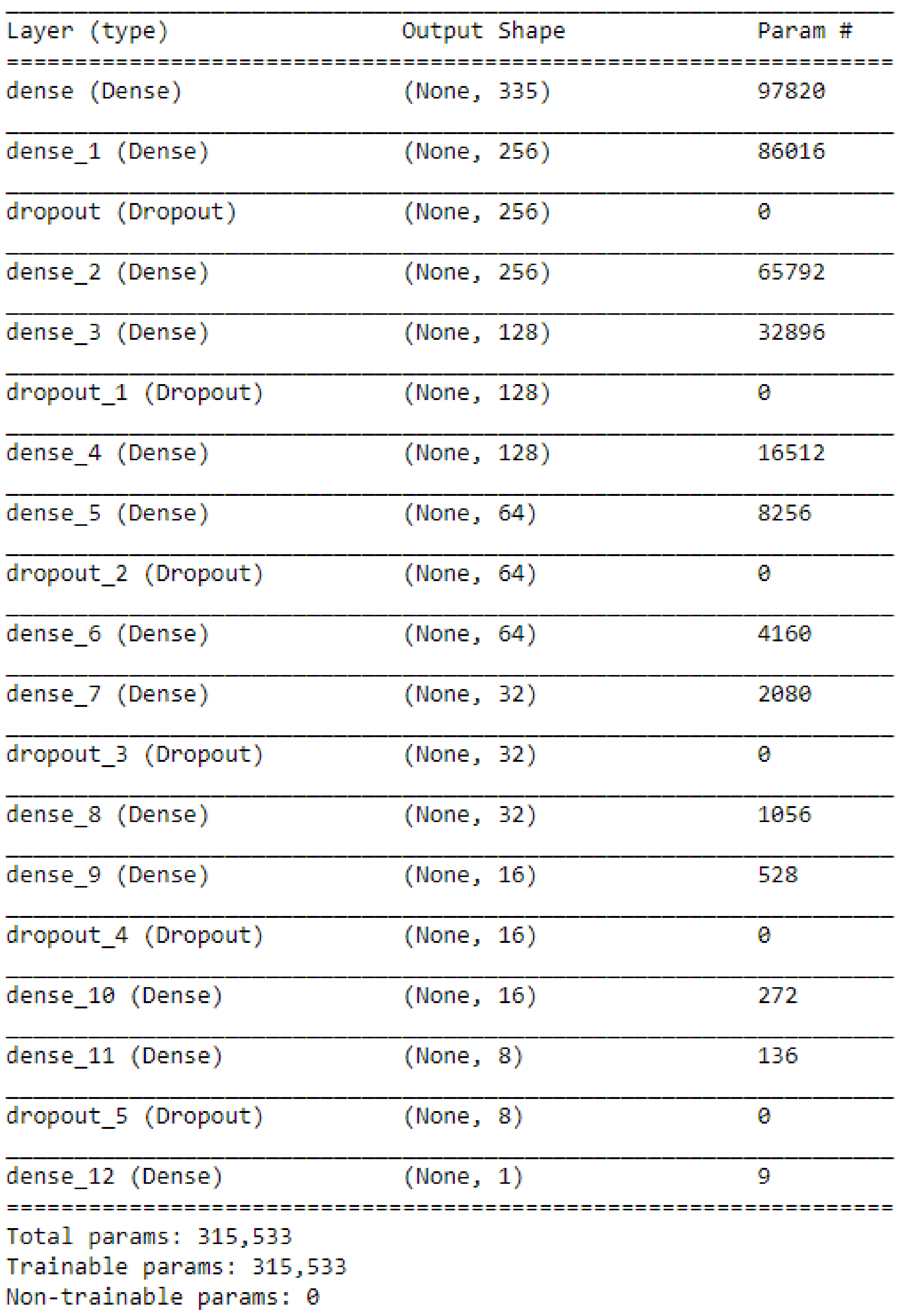
7 Appendix

7.1 Neural Network with cross feature interpretation



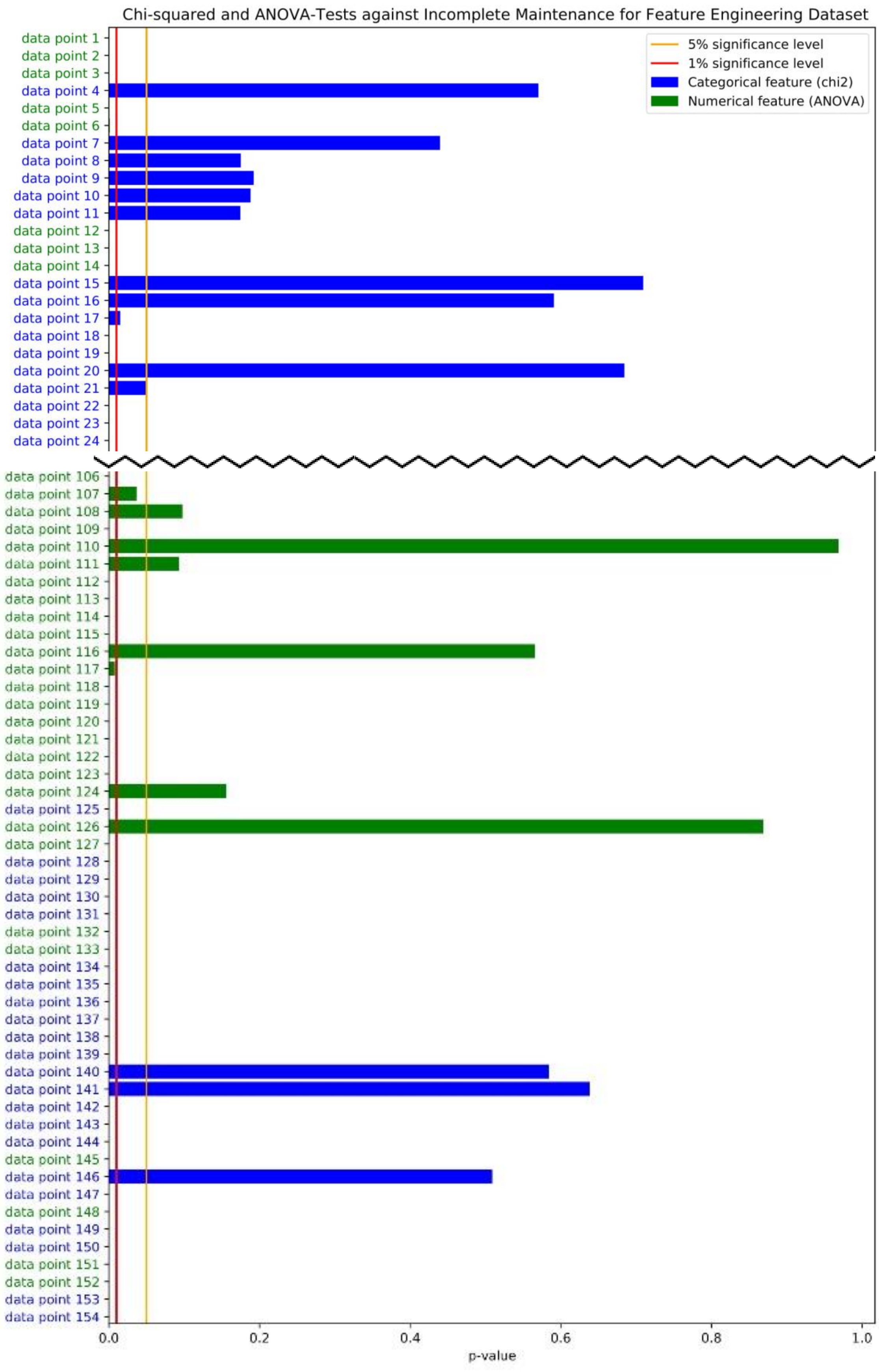
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 500)               316000    
_________________________________________________________________
dense_1 (Dense)              (None, 500)               250500    
_________________________________________________________________
dropout (Dropout)            (None, 500)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              501000    
_________________________________________________________________
dense_3 (Dense)              (None, 1000)              1001000   
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_4 (Dense)              (None, 500)               500500    
_________________________________________________________________
dense_5 (Dense)              (None, 500)               250500    
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 501       
=================================================================
Total params: 2,820,001
Trainable params: 2,820,001
Non-trainable params: 0
_________________________________________________________________
```

7.2 Neural Network without cross feature interpretation – the standard approach



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 335)               97820
_________________________________________________________________
dense_1 (Dense)              (None, 256)               86016
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_8 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_9 (Dense)              (None, 16)                528
_________________________________________________________________
dropout_4 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_10 (Dense)             (None, 16)                272
_________________________________________________________________
dense_11 (Dense)             (None, 8)                 136
_________________________________________________________________
dropout_5 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 9
=================================================================
Total params: 315,533
Trainable params: 315,533
Non-trainable params: 0
```

7.3 Dataset Analysis with statistical Independence Tests


Chi-squared and ANOVA-Tests against Incomplete Maintenance for Feature Engineering Dataset
5% significance level
1% significance level
Categorical feature (chi2)
Numerical feature (ANOVA)
data point 1
data point 2
data point 3
data point 4
data point 5
data point 6
data point 7
data point 8
data point 9
data point 10
data point 11
data point 12
data point 13
data point 14
data point 15
data point 16
data point 17
data point 18
data point 19
data point 20
data point 21
data point 22
data point 23
data point 24
data point 106
data point 107
data point 108
data point 109
data point 110
data point 111
data point 112
data point 113
data point 114
data point 115
data point 116
data point 117
data point 118
data point 119
data point 120
data point 121
data point 122
data point 123
data point 124
data point 125
data point 126
data point 127
data point 128
data point 129
data point 130
data point 131
data point 132
data point 133
data point 134
data point 135
data point 136
data point 137
data point 138
data point 139
data point 140
data point 141
data point 142
data point 143
data point 144
data point 145
data point 146
data point 147
data point 148
data point 149
data point 150
data point 151
data point 152
data point 153
data point 154
p-value