Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Situation Prediction for Situation-Aware Workflows on Customer Order Settlements

Sascha Sprott

Course of Study:	Informatik
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Kálmán Képes, M.Sc.
Commenced:	June 17, 2019
Completed:	December 17, 2019

Acknowledgements

I would like to thank Prof. Dr. Dr. h. c. Frank Leymann for making this thesis possible. Further special thanks to my supervisor Kálmán Képes for enabling me to write my thesis and for the exceptional good support. Further I would like to thank Dr. Nicole Ondrusch of the msg systems ag for her support and willingness to cooperate, to make this thesis possible. Special thanks to Dragan Sunjka and Ghenadie Rosca from the msg systems ag for their professional and technical consultation. Thanks goes also to Deborah Freudigmann for proofreading.

Abstract

These days, companies and manufacturers experience a need for faster and automated adaptions or re-configurations of their workflows and business processes. By means of context- and situation-aware systems, these desires are partially achievable, however only in a reactive manner. This situation based reactive behavior leads to expensive, non-efficient and time-consuming delays within workflows and business processes and hence asks for more research. Current state of the art in the topic of machine learning enables new approaches to investigate in, for workflow adaptions by predicting situations and allow proactive measurements for workflow adaptions and re-configurations. In this thesis an abstract concept is developed, that allows to turn reactive workflow adaptions into proactive adaptions within situation-aware workflow management systems, by predicting situations. This is achievable through computation of situation confidence scores by means of machine learning models and algorithms. Further, this concept allows using these algorithms without expert knowledge in the topic of machine learning, by hiding the implementation details from the user. The concept of predicting situation confidence is tested on a use case scenario for real orders of a manufacturing company and compares the classification approaches Support Vector Machine, Multilayer Perceptron and Random Forest for the prediction of orders. Results show only good performance for the Random Forest classifier, but also a concomitant possible applicability of the concept. More algorithms need to be tested and the tested algorithms need improvements, to fortify the applicability of the developed concept.

Contents

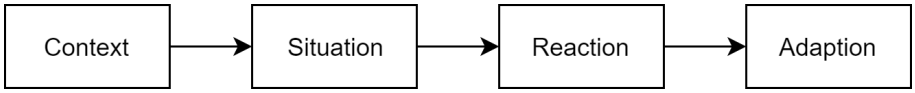
List of Figures

List of Tables

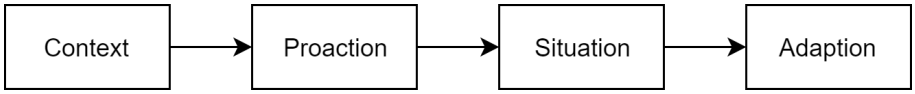
List of Listings

1 Introduction

Predicting situations is no novel demand of humanity. It has always been desired by humans to know situations and events before someone else. With this intention, the ability to know things, even if they have not happened yet, is the intuitive next step. In latter case, the scientific phrase is *predictions*. Nowadays, companies and cyber-physical systems run complex workflows and processes. These workflows are the basic machinery for companies and systems to offer certain services, in form of products or components. Manufacturing fabrics or cyber-physical service systems often have to deal with re-configurations or restarts of components within their workflows. Re-configurations are thereby needed to change the workflows behavior, allowing the systems to provide different services or produce different products. This re-configuration and adaption of workflows or business processes can be very cumbersome, time consuming and expensive, due to additional effort related to such adaptions. Recent work [KBL19; SBLW16; WSBL15] does only investigate into reactive workflow adaptions within context aware systems, like *Smart Homes* or *Smart Factories*. However, reactive design of workflows, as it is shown in Figure 1.1a, induces an additional delay to the adaption of a process, after the occurrence of specific workflow influencing situations. Hence the questions *if* and *when* such situations occur, have a high impact on the adaptions of workflows. The possibility to predict such situations in advance seems to bring a solution to this problem and to overcome the induced delay. While there exist some research [LM15; YZWM19] that aims for overall situation prediction, only few research [KHSY15; PHG13] try to predict situations to allow proactive workflow adaptions, by using machine learning approaches. If systems know about a given situation before it occurs, adaption to corresponding workflows and processes can be initiated accordingly in advance and at the right time. This minimizes expense, delay and effort induced by the implementation of re-configurations and adaptions. Predicting situations allows to change the sequence of reactive workflows as shown in Figure 1.1a to the proactive more efficient sequence in Figure 1.1b, which enables to overcome the delay between an occurring situation and its induced adaptions and hence allows to save expenses and resources.



(a) Reactive treatment of situation-aware workflow scenarios.



(b) Proactive treatment of situation-aware workflow scenarios.

Figure 1.1: Context-aware system workflows for reactive and proactive situation-awareness, to enable adaption to an application, workflow or business process.

As companies are growing and with the great diversity of workflows and environments, cyber-physical systems become more complex. This complexity accompanies the pertinent need to reduce costs, effort and workflows to act more effective. Hence, the need for smarter workflows for companies and in cyber-physical systems is settled. The topic of machine learning is already a part of our everyday life and its usage for workflow improvements is not devious. Some existing research [DMJS19; Her00] already uses machine learning to improve the overall workflow management or workflows themselves. While some investigation [LM15; YZWM19] for the task of situation prediction for network security is present, the current literature lacks the general use and investigation of machine learning algorithms for situation prediction in context- and situation-aware systems.

The aim of this thesis is to validate the hypothesis, if situation predictions within cyber-physical systems like context-aware systems, is applicable to real world problems like order settlement predictions. Therefore, this thesis develops and validates a framework that allows predicting confidence scores for requested situations, which affect workflows and business processes by demanding adaptions and re-configurations of such. Eventually confidence scores shall allow a system to enable proactive adaption to changes and allow faster and more efficient reconfiguration to occurring situations. The framework provides confidence scores and allow a reformatting of reactive behavior of workflows into a proactive workflow behavior, and thus enable further saving of resources and time. To achieve such situation prediction and hence a proactive behavior of workflows, this thesis uses machine learning approaches to compute the desired confidence scores. Three different classification machine learning models are trained and evaluated, to provide situation predictions for business processes and workflows. These machine learning classification models include a *Support Vector Machine* (SVM), a *Multilayer Perceptron* (MLP) and a Random Forest (RF). For validation and evaluation of situation predictions, a use case of real historical order settlements of manufacturing sales data is used. The machine learning models are trained on this use case data and compute confidence scores for situations, in form of new orders. Regarding this use case, some research exist for sales [GXF+14] or time-series [SS09] order predictions in form of regression machine learning algorithms, while others researches investigate in the explanation of machine learning algorithms used for sales predictions [BBR17]. These investigations show some of the applicable appropriated machine learning algorithms for order predictions, which could also perform for the joined task of situation and order prediction. Another task of this thesis is to integrate the functionality of situation confidence computation into a situation-aware workflow management system. This system, called SitOPT [WSBL15] can receive situations and automatically adapt and re-configure workflows and applications and with the above integration is able to request situation confidence scores for corresponding situations.

This thesis starts with an introduction to the use case for testing the concept in Chapter 2, where a overview of the composition of the data is given. In Chapter 3 fundamental knowledge, mandatory needed to understand the concepts of this thesis are provided. Further in this chapter are some mathematical backgrounds introduced to help understanding the machine learning approaches and evaluation metrics. Afterwards, Chapter 4 presents related scientific work according to the concept of situation prediction and workflow adaptions of context aware systems. It will show some of the major machine learning algorithms used in the topic of order prediction and situation prediction in situation-aware systems. Chapter 5 covers the development of the framework, which enables the calculation of confidence scores for situations. It will introduce to the concept of situation prediction and present the used machine learning approaches. Afterwards, Chapter 6 will present the evaluation results of the confidence score computing machine learning models, based on the given use case to predict actual order settlements. This chapter will introduce to visualization metrics that are build

on the in Chapter 3 presented basic evaluation metrics. Chapter 7 presents detailed information about the overall implementations and the integrated development environments, used for the programming tasks of data preprocessing, training of the different machine learning algorithms and implementation and integration of the situation confidence predictions into the situation-aware workflow management system SitOPT. In Chapter 8 the accomplished results and concomitant problems of the framework design and implementation are discussed. Finally Chapter 9 summarizes the findings of this thesis, and gives some remarks of encountered problems and outlooks of what further can be done in research in situation prediction for situation-aware workflows.

2 Use Case

The developed concept in this thesis is tested on a use case scenario for order predictions of a manufacturing company. Incoming new orders are treated as situations, whereas the historical data set of past orders serves as the context in which orders are predicted. Figure 2.1 shows a simplified reactive production process, how new orders are treated. A customer experiences a need for a specific product, which may forces him to break or stop a specific workflow to order the product at a manufacturer to continue the workflow. The manufacturer starts the production of the product, which induces some delay by means of the manufacturers own internal processes. After finishing production, the product can be delivered back to the customer, who is then able to continue his workflow. With the developed framework, this reactive order settlement workflow is to be redesigned into a proactive workflow, shown in Figure 2.2. The workflow for the customer is the same as before, but without a delay for a workflow. For the manufacturer, some components change. He precautionary sends an order prediction request to a prediction service, which replies with a confidence score for the requested order. Based on this confidence score, the manufacturer starts production in advance of receiving the actual order, to allow the product delivery immediately after ordering. The confidence scores to compute are for predicting certainty about orders, which are received as new requests from customers. Such requests include information about the type and quantity of product ordered the date of order inflow, the type of order and information about the customer who requested or ordered the product. A small extract of the data set is given in Table 2.1. With these confidence scores, production of certain products for corresponding orders could be proactively triggered. Therefore, a responsible person would have to check the produced score for a request and need to decide if it is enough to trigger the preproduction. Another way would be to define a threshold, and if the confidence crosses it, the production is automatically initiated. With the proactive enhancement, warehousing costs for productions on the off chance and unnecessary shipments could be saved, and further delivery times could be shortened. Hence, both parties, customer and manufacturer would profit from the implementation of such framework.

kdgrp	kunnr	matnr	comnr	mtart	matkl	prdha	mstae	farbe	hoehe	durch	lschn	auart	erdat_y	erdat_m	limng
6	250057	1004236	M19	ZFER	106	PE1	99.0	12	16	17,1	_	TA	2009	8	84000
2	250025	1001540	VF2	ZFER	104	PE2	80.0	10	51	37	_	TA	2009	8	14381
2	250025	1001540	VF2	ZFER	104	PE2	80.0	10	51	37	_	TA	2009	8	9580
2	250025	1001540	VF2	ZFER	104	PE2	80.0	10	51	37	_	TA	2009	8	14400
2	250025	1001540	VF2	ZFER	104	PE2	80.0	10	51	37	_	TA	2009	8	7280
6	250057	1001395	M15	ZFER	104	SC2	99.0	11	38,85	24,29	_	TA	2009	8	43200
6	250057	1005308	M18	ZFER	106	PE1	99.0	12	18	22	_	TA	2009	8	90170
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2.1: Extraction of the use case data set provided by a manufacturing company to train the machine learning algorithms addressed in this thesis.

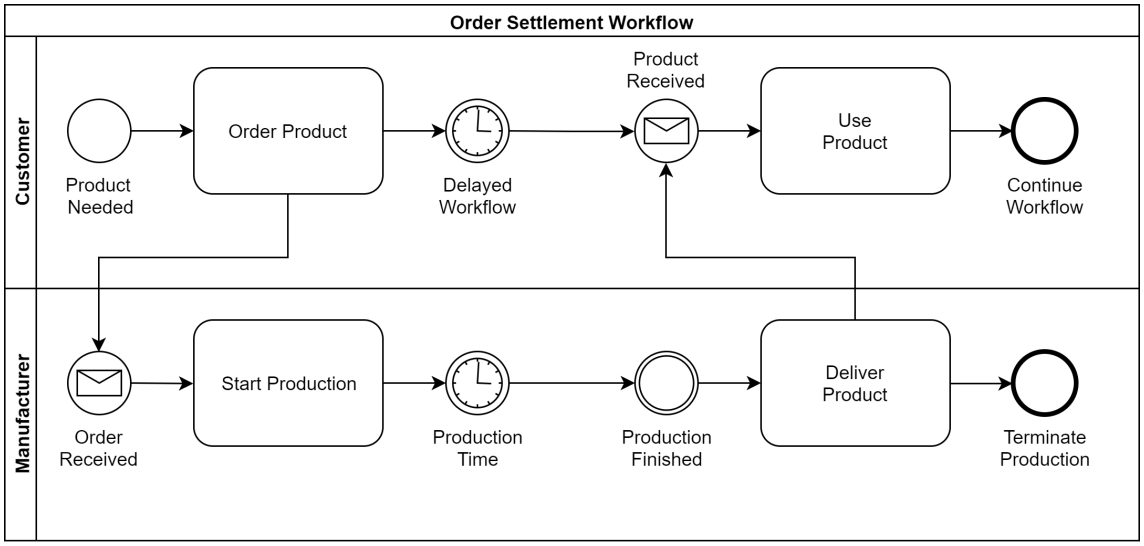


Figure 2.1: Simplified visualization of a common *reactive* handling of order settlements.

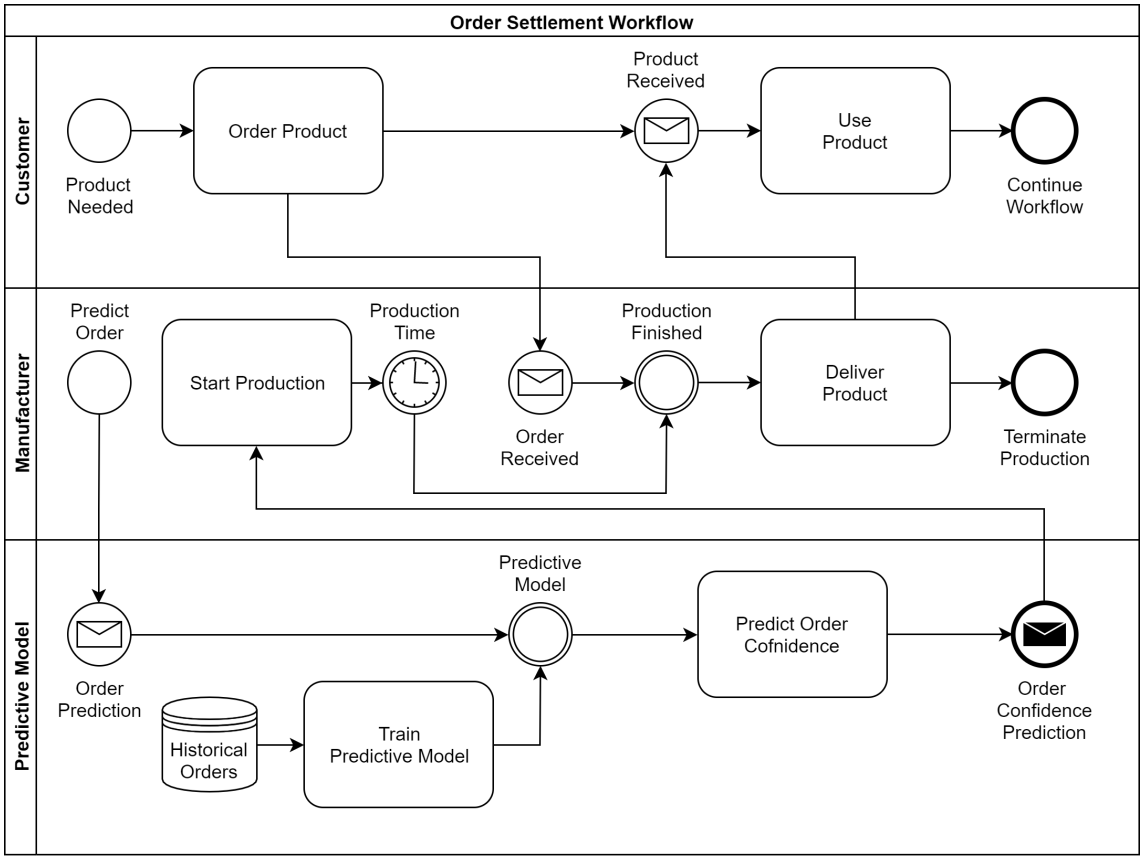


Figure 2.2: Simplified visualization of a desired *proactive* handling of order settlements.

The historic order settlements, used as a basis for this thesis, are compositions of three different single data sets. A manufacturer provided these data sets, by exporting them from their Enterprise Resource Planning system. Each data set contains different types of information about customers, materials and orders. The first data set includes actual customer settlements, starting from mid 2009 until mid 2019, scattered around different customers, materials, and sales organization representing different core business branches. Each settlement can involve multiple rows. Whereas each rows meaning is defined by its sales and position number. If in each row of a common sales number, the position is the same, then the different columns describe different states for the quantity of a settlement: requested, confirmed and eventually shipped. Else, if the position number differs, the settlement is a cumulative settlement within the same settlement but with different requested delivery dates. For each row of a settlement, there are further information given about the product itself, the customer and the category. Further given are a date of shipment, a commission number, the cumulative quantities for each state and some flags, indicating non-zeroes. At last, a counter for days between the requested and confirmed date of shipment. The second file holds extensive information for each of the order settlements in the first file. For each sales number there exists a row, contributing a settlement generation date, an ordering date, a sales type, a settlement type and a higher order sales number. In the last and third file, information about every product within the companies product catalogue is selling is provided. It includes for example the material class, its color, size, shape and the hierarchical position of the item.

Not all of the parameters of the data files are needed for the purpose of this thesis. Actually some parameters are recurrent, and thus the files can be cropped immensely, eventually leading to the 16 parameters presented in Table B.1, with a total number of $57,094$ orders.

3 Fundamentals

This chapter gathers some of the general fundamentals, which are required to understand and needed to develop the concepts and functionalities presented in this thesis. The first part of this chapter, Section 3.1, focuses on presenting some fundamental knwoledge of machine learning approaches, continuing with a focus on the overall understanding of machine learning tasks and some basic evaluation methods. The machine learning models used to build the desired framework in this thesis, are not covered in this chapter, but a deeper and also mathematical explanation is given later in the corresponding Chapter 5. After the machine learning part, Section 3.2 introduces to context aware systems and their architectural design methodologies, since the provided confidence scores for situation prediction are a contribution to improvements of situation-aware workflows, which themselves are part of context aware systems. Section 3.3 presents SitOPT, a design model for adaptive workflows, implemented with the TOSCA specification language. Section 3.4 covers an introduction to the *Open Topology Orchestration and Specification for Cloud Applications* (OpenTOSCA) an extension runtime to manage dynamic application service designs.

3.1 Machine Learning

This section will introduce to some of the core concepts of machine learning, required in this thesis. Machine learning is an approach to problem solving if the size of data sets or the immense amount of relations within a data set, become overwhelming for humans. Machines are supportive tools for pattern recognition and further simplify and finish tasks within a short period of time, where humans would need significantly longer to accomplish the same goal. With the use case data set of $57,094$ entries, each with 16 parameters, it is nearly impossible to detect all relations and patterns by hand. Hence, machine learning is a reasonable approach to solve the problem of finding confidence scores for future orders or situations. In the following, some approaches to different problem settings and different evaluation metrics for machine learning models are presented. Overall the knowledge about some machine learning fundamentals and the partial content of this chapter are taken from the Scikit-Learn website [PVG+11], Grus's book "Data Science From Scratch" [Gru15] and [JWHT14] by James et al., which are excellent elaborations and guidelines for getting into machine learning.

Machine learning is the topic of programmed machines, which learn patterns from existing observations and try to reproduce these onto new unseen data. A machine observes some given data, where each observed instance is parametrized by a specific set of values. Patterns within this data set depend on these parametric values, which themselves can be in continuous or categorical form. Data or observations used by a machine are commonly given in table format. The machine reads the observations from the table, and processes them with a specific algorithm. This algorithm discovers and learns patterns and relations from the data and further tries to reproduce or approximate them as precisely as possible onto new unseen data. After learning and discovering, such an algorithm is ready to make the desired predictions expected from a machine learning approach.

In machine learning, there are different methodologies to approach a problem. Two of these methodologies include *Supervised* and *Unsupervised* learning. In this thesis, a supervised approach is chosen. To understand, why this type of models are used, the previously mentioned two methodologies need to be explained.

Supervised learning Machine learning models of this type are identified by containing an extra feature, called "target". They have a "supervisor" telling the model what target a specific observations implies, based on the given input. Models of such type can further be separated into *Classification* and *Regression* problems. Depending on latter separation, the target value of a measurement of the input data set has to be interpreted differently. On the one hand it can be a qualitative value (*Classification Algorithm*), meaning the value is not interpreted numericaly but as a categorical class. An example for such a target value datatype could be the prediction of species or weather conditions. If there exist only two classes, it is a binary classification problem (e.g. predicting rain, while it either rains (1) or it does not (0)). The other possible interpretation considers target values to be quantitative (*Regression Algorithm*). In this case, targets are numerical values that are to be predicted. An example for these type of predictions are stock prices or the number of ants within an anthill, based on its size. For both of these types of supervised learning models exist further classifications of models. Some example for linear models are Ridge- Lasso- or Bayesian-Regression, which use a linear combination of the input features to predict target values. Ensemble models like Random Forest use the principles of bagging and boosting, by instantiating multiple weak learners and combine them to get the average best or a single strong learner. Another type of algorithms are neural networks, which are compositions of layers of neurons that are used to learn patterns. By using more layers of neurons, patterns that are more complex can be discovered.

Unsupervised learning These type of machine learning models do not have a target feature indicating the output for a certain input. As the name suggests, there is no "supervisor" helping the model to learn something. It is rather a self-organized analysis, to discover correlations and patterns within the unlabeled data. A simple example for this type of learning is a model trying to distinguish between cats and dogs, without knowledge about how a dog or a cat looks like. Unsupervised learning algorithms are typically clustering algorithms and neural networks.

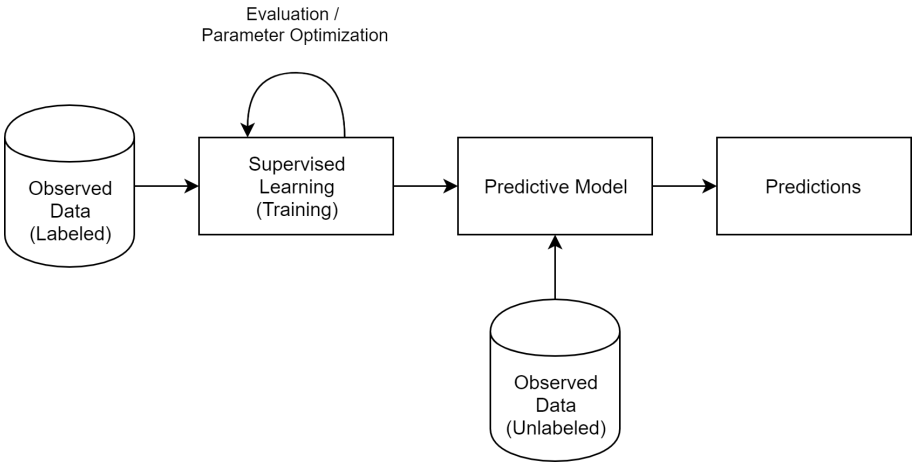
Figure 3.1 shows typical workflows for supervised (see Figure 3.1a) and unsupervised (see Figure 3.1b) learning models. Both types include observations and a learning procedure (called training), to generate a predictive model. The main difference can be found in the observations, as they are labeled or unlabeled.

Mathematically, a model can be described as below. Given a perfect algorithm f which observes some data X and predict Y, a machine learning approach tries to find an approximation f' of f as accurate as possible, such that it is able to predict targets Y' based on new observations:

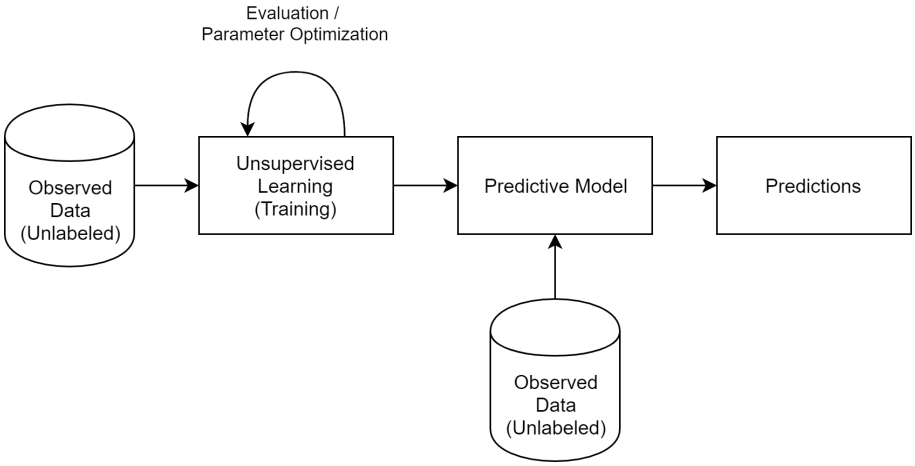
$$Y + \epsilon = Y' \tag{3.1}$$

$$f(X) + \epsilon = f'(X) \tag{3.2}$$

The true (Y) and predicted (Y') targets differ slightly by an error term ϵ. This error arises due to the imperfectness of an estimated algorithm, as it is only an *approximation*.



(a) Procedure for supervised machine learning algorithms, that observes some labeled data, trains a model based on discovered patters and uses the resulting predictive model to predict labels for new unlabeled observations.



(b) Procedure for unsupervised machine learning algorithms, that observe some unlabeled data, trains a model and label the observed data based on discovered patterns and uses the resulting predictive model to predict further labels of new observations.

Figure 3.1: Machine Learning Workflow for supervised (a) and unsupervised (b) machine workflow algorithms.

Since the aim of this thesis is to predict the occurrence of situations, which can be interpreted to classify if a situation occurs or if it does not, further discussion only covers topics associated with classification algorithms of machine learning.

In classification, a prediction of a class involves a probability indicating how certain the classification is. Such a probability for a predicted label y of a sample X is mathematically described with:

$$P(Y = y|X) \tag{3.3}$$

Considering a probability for a specific sample being classified to class y, it is assigned to that class only if its probability exceeds a certain threshold. This threshold can be varied from being extremely high (e.g. 99%) to extremely low (e.g. 1%). Predicting infectious disease by considering data from patients, may need a a very low threshold, to minimize any risk. Whereas predicting stock prices need a very high threshold, since uncertainty need to be minimized. The challenges of how and what threshold to choose is out of the scope of this thesis. The probability thresholds for the classification of the labels in thesis thesis is set to 0.5, but since this thesis considers probabilities as results, this threshold is only used for some evaluation metrics.

After building a model, predicting targets is easy, but if these predictions are usable is unsure. A model can be useless, due to predicting wrong classes in almost every case, even if trained for days. As previously mentioned, machine learning models try to imitate a perfect pattern recognition function f with an approximate function f'. To get an estimation of how good a machine learning model f' can approximate a perfect model f, there is some kind of evaluation needed. In classification, evaluation is simple. A model is bad if it is inaccurate, meaning if it predicts the wrong value. The overall classification of a model with true values $y_i = f(x_i) + \epsilon$ for observation i and $y'_i = f'(x_i)$ as the predicted value, with n observations, is given by the formula:

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{I}(y_i \neq y'_i) \tag{3.4}$$

with $\mathcal{I}$ as the indicator function, indicating 1 if the statement within it is true and 0 if it is wrong. Equation (3.4) gives a mathematical representation of how accurate a model predicts labels or classes, and can be called error rate. Nevertheless, the error rate of a classification model can sometimes be an inappropriate measure. For example, if a given data set has a high class imbalance, meaning many samples of one class (99%) and only few of another (1%), a model that always predicts the first class, would have an error rate of only 1%. This error rate appears to be very good at first sight, since only 1% of data are not from the first class. This would be a very misleading interpretation for the evaluation, since even though the error rate is correct and very low, the model itself is very bad, and hence asks for further investigation.

For imbalanced classification problems, another useful evaluation method can be used: *confusion matrices*. Within a confusion matrix, each distinct label of the target set of the training data gets a row and a column. Columns indicate the predicted class, rows the true class of samples. In a binary classification case, there exist only two classes. The first one is labeled with 1 and is called "positive" class, the second one is labeled with 0 and called "negative" class. Hence, the confusion matrix of a binary classifier is a 2×2 matrix, with four cells, like shown in Table 3.1. Each cell represents an error: *True Positives* (TP), indicating the prediction of class 1 for a sample with true

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

Table 3.1: General confusion matrix with a predicted label axis and true label axis for a binary classification problem of negative and positive samples.

value 1; *False Positives* (FP), indicating the prediction of class 1 for a sample with true value 0; *False Negatives* (FN), indicating the prediction of class 0 for a sample with true value 1; *True Negatives* (TN), indicating prediction of class 0 for a sample with true value 0.

These four metrics serve as the basis of many more different evaluation metrics for classification. Some further basic metrics, which are calculated based on Table 3.1, are defined below with their corresponding equations.

Definition 3.1.1 (Precision)
Precision is a metric about how accurate the correct positive predictions are made over all truly positive samples within the set of samples.

$$Precision = \frac{TP}{TP + FN} \tag{3.5}$$

Precision gives information about the percentage of samples of positives that can correctly be judged. The closer precision is to 1, the more accurate it can classify, until 1 where it does always classify.

Definition 3.1.2 (Recall (Sensitivity))
Recall is a metric about how accurate the positive predictions are made over all positive predicted samples within the set of samples.

$$Recall = \frac{TP}{TP + FP} \tag{3.6}$$

Recall tells the user if the model missed some positive classifications, or how many of all truly positive samples did the model identify. If recall is 1, then every truly positive sample was classified positive, meaning all positives could be correctly classified.

Definition 3.1.3 (Support)
Support is a metric about the percentage of total samples that are targeted so far.

$$Support = \frac{TP + FP}{TP + TN + FP + FN} \tag{3.7}$$

Support is just a measure of how many of the samples for the total data set are classified so far in the evaluation procedure.

Definition 3.1.4 (Accuracy)
Accuracy is a metric about how accurate the overall classifications over the whole sample set is.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.8}$$

Accuracy can be considered as the counter part to the error rate defined in Equation (3.4).

Definition 3.1.5 (Fall-Out)
Fall-out is a metric about the negative classification rate of truly positive samples.

$$Fall - Out = \frac{FP}{FP + TN} \tag{3.9}$$

Fall-Out describes the false negative rate of a classifier.

These metrics form the basis of the evaluation process in Chapter 6 for the in Chapter 5 introduced and trained machine learning models.

3.1.1 Over- & Underfitting

Sometimes, machine learning algorithms are only able to reproduce the discovered data set without the ability to generalize the patterns onto new unseen data, or are not able to learn or discover any patterns within the data sets. These phenomena are called over- & underfitting. To overcome these problems, machine learning algorithms provide adjustable parameters that try to correct the models approximation attempts. To avoid overfitting, a regularization term is added to model approximations. Generally, two different regularization terms are used: *Lasso Regression* (L1) and *Ridge Regression* (L2). Former, uses the absolute value of weights to training parameters, to penalize overfitting. The formula is given with:

$$\lambda \sum_{i=1}^{n} |w_i| \tag{3.10}$$

The second regularization term on the other hand, uses squared weights for penalization and is given with the formula:

$$\lambda \sum_{i=1}^{n} w_i^2 \tag{3.11}$$

In both Equations (3.10) and (3.11), λ gives the weight to the regularization. If it is chosen high, the weights are heavily rated, leading to parameters having a strong impact and eventually to overfitting. Reducing λ close to 0 leads to underfitting, since the parameters have only a small impact.

Most machine learning approaches do use a learning rate parameter, that also allows how fast and accurate a model is trained. It is a value that weights the amount of information carried into the next learning iteration, by weighting the gradient decent to find the minimum loss for the model.

If it is chosen too high, the model cannot find the minimum because it overshoots the minimum loss, hence resulting in a high loss value. If chosen too low, finding the loss can take extremely long time, since every information has to be processed.

The procedure of finding optimal parameters for a model is called hyperparameter tuning. Commonly used methodologies to examine good parameters to train a machine learning model, are *grid search* and *cross validation*. Former allows generating a grid of parameters the model has to be tested on, allowing to compare the results of different parametrizations of a model and eventually to find optimal parameters to train the model efficiently and with minimum loss. *Cross validation* does repeat the training with a specific parametrization multiple times, to make sure that the result of a model is not just a good or bad lucky guess. In combination, those to methodologies allow a powerful hyperparameter tuning procedure. The Scikit-Learn library [PVG+11] offers a class to combine these two methodologies for hyperparameter tuning (`model_selection.GridSearchCV`).

3.1.2 Feature Encoding

Features, parameters or input values for machine learning models can be continuous or categorical, as it was mentioned in the precious sections. If they are continuous, they can be used as they are, or maybe scaled between 0 and 1 for an easier handling. However, if they are categorical, their usage is not as simple as that. The sum of two strings is not defined, and hence there is no use of such interpretation of categorical features. Labeling is the procedure of replacing categorical values with numbers. Nevertheless, simply replacing strings with numerical values does not help either, since this would imply an unwanted weighting to classes. E.g. if one class is replaced with value 2 and another with value 4, the latter would have double the impact than the first one. Therefore, a little trick is used, called *one-hot-encoding*. Instead of simply replacing values with numbers, each distinct category within a parameter gets its own parametrization. That means, for each category within each existing feature, a new column within the data table is generated, assigned a 1 if the parameter matches the specific value of a sample, and 0 if it does not. Again, the SCikit-Learn library [PVG+11] offers a class (`preprocessing.OneHotEncoder`) to accomplish this task.

3.2 Context Aware Systems (CAS)

The prediction of confidence scores for future situations enables CAS to adapt their workflows proactively. CAS do not only benefit from situation prediction, but they also control the amount of data that is used to train machine learning models, that allow proactive situation-awareness. A detailed and comprehensive introduction to fundamental concepts of CAS can be found in the survey [BDR07] of Baldauf et al., which will be the major contribution for this section of the thesis.

3.2.1 CAS Architecture

CAS can be defined by a simple model of 5 layers, depicted in Figure 3.2. A core concept of CAS architectures is the separation between detecting or receiving context information and of using a detected or received context. In that sense, the Sensor layer L1 does cover more than just sensors, but includes every possible source of a context. Context data can be achieved via hard-wired sensors,

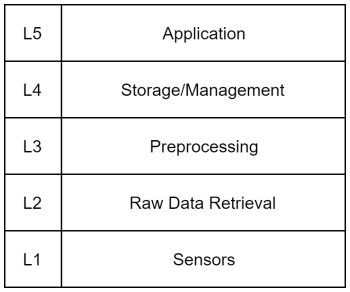


Figure 3.2: General framework concept for CAS, introduced by Baldauf et al. in [BDR07].

software knowledge or a combination of both. Examples for these two categories would be a movement sensor or a software that tracks user behavior on his computer. The Raw-Data-Retrieval layer L2 represents application programming interfaces (APIs) and software instances for connected hardware devices. With this type of layer, instances from L1 can be exchanged without loss of functionality. The next layer in the figure is the Preprocessing layer L3, which is rather optional. In fact, it is only necessary if the achieved raw data is not in an appropriate shape, to convert, aggregate, extend or extract information. At this point, in the presented framework, the field of machine learning can participate. Based on the different processing types presented, preprocessing of context data retrieved from sensors, can be augmented by machine learning approaches to be trained from context data, generating new insights into existing contexts, or to gain insights about context patterns. With the Storage/Management layer L4, an obligatory layer is the next on the list. Context data received and maybe preprocessed, need to be stored. This can be approached, centralized or decentralized and further synchronous or asynchronous. While former means, data is stored on a single machine or on multiple distributed machines, latter means data is achieved via request and response or via publish and subscribe. Either the requesting application awaits the context data while stopping other program tasks, or it reacts to the data when it is published. The last layer, the application layer L5, represents the application interacting with the context data. With the reception of certain context data, an application triggers processes and workflows.

3.2.2 Context Models

Contexts are the basis of CAS, they form the foundation on what data is collected and how it is modelled. A detailed overview of different modelling approaches for contexts is presented in [SL04] by Strang and Linnhoff-Popien. They give an overview of different assessable properties for data management approaches. These properties play an important role, since they reflect the usability of the individual models.

1. Distributed Composition: The missing of a central managing instance, which allows high dynamic behavior in time as well as source management and network topologies.
2. Partial Validation: The validation of context knowledge and context model instances, even if the system is a distributed composition. When only shares of knowledge are accessible, validation still need to take place.
3. Quality of Information: Quality and availability of sensor data within a context. Rates how reliable data sensors are.

	1	2	3	4	5	6
Key-Value	−	−	−−	−−	−−	+
Markup Scheme	+	++	−	−	+	++
Graphical	−−	−	+	−	+	+
Object Oriented	++	+	+	+	+	+
Logic Based	++	−	−	−	++	−−
Ontology Based	++	++	+	+	++	+

Table 3.2: Properties for different data models used for context representation by Strang and Linnhoff-Popien in [SL04].

4. Incompleteness & Ambiguity: An assessment of contextual data, if it is available at any timestamp.
5. Level of Formality: Indicates and ensure the same interpretability for all participating parties, such that no misinterpretation of data is possible.
6. Applicability to Existing Environments: A measure of how good a context model is applicable to a given application environment.

These different requirements or properties allow an assessment of context model designs. Strang and Linnhoff-Popien present six different types of such context model types, which will be briefly introduced in the following list.

Key-Value: Is a pair like data storing model. It is known for its easy usability, but tumbles regarding complex and efficient data representations.

Markup Scheme: An hierarchical data structure, such as XML. The main components are tags with corresponding attributes and values. Tags can recursively occur within another tag, forming a scheme. Markup Schemes are limited by their contextual aspects.

Graphical: Generic modeling constructs, like UML diagrams. They leverage easy transformation into entity relationship models.

Object Oriented: Data is stored in objects to benefit from the object-oriented property of reusability. Access to objects is granted via specific interfaces.

Logic Based: A context is defined by rules and facts, whereas facts are derived from other facts by conditions and rules.

Ontology Based: An Ontology represent facts and concepts and further relations between them. They can also be seen as knowledge bases, hence a representation of the "world" as collection of facts.

Table 3.2 shows the assessment of the presented context models, where − and + show how strong the corresponding context model supports the specific property.

Layer L4 in Figure 3.2 has the duty to manage and store different contexts. Besides the different modelling approaches presented above, in [KM03], Korpipää and Mäntyjärvi describe several principals, desirable for a context representation: *simplicity*, *practical access*, *flexibility*, *expendability*,

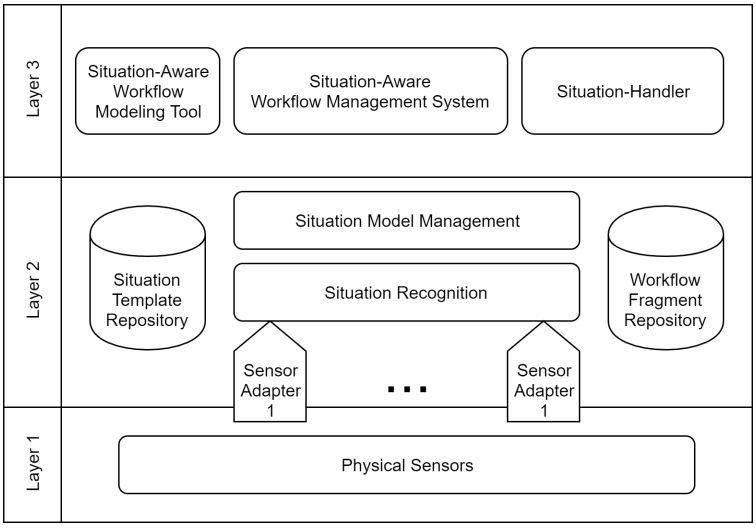


Figure 3.3: Three layer architectural design of the SitOPT project, consisting of a *Sensor Layer*, a *Situation Recognition Layer* and a *Situation-Aware Workflow Layer* [WSBL15].

facilitate inference, *genericity*, *efficiency* and *expressiveness*. Furthermore, they introduce to the term context atom, as a single instance of a context. Additionally, they present basic properties for a context atom. The *context type* identifies and categorizes contexts, and it is thus restricted to one context at a time step. Next, the *context* itself is the symbolic representation of the instance, while the *context value* represents the data a context receives from sensors. Further optional parameters are a *timestamp*, a *source* indicating where the information comes from, some *free attributes*, and a *confidence*, indicating certainty for the context.

Some systems, which use different types of data management approaches, are introduced in [BDR07], and thus will no further be discussed here.

3.3 Situation-Aware Workflow Management System (SitOPT)

SitOPT is a project by the University of Stuttgart, whose main purpose is depicted in [HWS+16], the work of Hirmer et al. and by Wieland et al. in [WSBL15]. It is conceptualized to split workflow design, workflow adaption and situation recognition, and thus to support situation-awareness and automated dynamic adaption within smart environments. It allows designing an interaction-like behavior of sensors within a smart environment, which can automatically react to contextual changes with proper adaptions. SitOPT is conceptualized as a three layered CAS, shown in Figure 3.3, and implemented in the TOSCA language for specification of cloud applications [OAS13].

The first layer has the duty to capture raw sensor data, and is called *Sensing Layer*. It does only sense the environmental changes and capture them by hardware sensors. In the second layer, *Situation Recognition Layer*, the captured data is processed, aggregated and filtered to allow situation recognition, where situations are classified by *Situation Templates*. The last layer, the *Situation-Aware Workflow Layer* is able to design situation-aware workflows, based on situations from the second layer, and handled by a *Situation Handler*. Within this last layer, a situation-aware workflow management system is responsible for dynamic workflow adaptions.

The basis of SitOPT are situations and workflows. Workflows represent standard procedures within a system. Such workflows can deviate from their normal behavior, whereby this behavior is referred as exceptions and the workflow changes into alternative workflows. These exceptions need to be modelled manually, and are triggered by situations. Hence, situations need to be observed, recognized and labeled with different *Situation Templates*, which is the task of the second layer in the SitOPT architecture in Figure 3.3. If a situation is detected, the *Situation Model Management* announces the occurrence of that situation. The *Situation Handler* processes the situation occurrence, and adapt the workflow behaviour accordingly. Therefore, the handler needs to consider every current occurring situation and compose compatible workflows, which are started, or adapted by the *Situation-Aware Workflow Management System*.

While the SitOPT project is an enrichment for workflow management, it is also capable to manage workflows as applications, like it is done by Képes et al. in [KBL19]. However, in both cases its approach conducts adaptions only in a *reactive* manner. At this point, the contribution of this thesis is manifested. This thesis aims to generate a property for situations or orders, considering the given use case scenario, called confidence scores. These confidence scores contribute a probability of a situation occurrence. Using a prediction probability, when a specific situation can happen, the situation model management is enabled to announce the occurrence of a situation in a *proactive* manner. Thus, the situation handler can trigger workflow adaptions, and if the situation finally occurs, the adaptions are immediately alive. This thesis validates, if it is possible to predict situations in a reliable way, to allow quicker workflow adaptions.

3.4 OpenTOSCA

Binz et al. present in [BBH+13] an extensive runtime for the Topology Orchestration and Specification for Cloud Applications (TOSCA) [BBLS12; OAS13], called OpenTOSCA. It is a supportive runtime for imperative processing in the TOSCA environment. The system uses workflows (called plans in [BBH+13]) to enable dynamic configurations and management of applications and cyber physical systems. OpenTOSCA can deploy cloud service architecture (CSAR) files and instantiate applications. Its main purpose is to run plans, manage states and operate management operations in a completely automated fashion. How this automation is enabled, is show in Figure 3.4. At the start, CSAR files are extracted and their content is stored. The OpenTOSCA architecture consists of two main components, a *Control* unit and a *Core* unit. Whereas the first component gets requests via the *Container API* and loads the TOSCA XML files, and processes them. Latter component provides services for other components. Furthermore the *Control* component calls the *Implementation Artifact* and *Plan* engine. Implementation artifacts from a CSAR file are run and provided to plans by the eponymous engine. Plugins to this engine, take care of how and where to run and deploy such artifacts, and further store deployed management operations and services into the *Endpoint* database. Plans or application workflows from the CSAR file are processed by the *Plan Engine*. Plugins of this engine provide access to different model languages and take care of binding services from the endpoint database to the corresponding plans. After binding services to plans, a plan is deployed. Via the *Plan Portability API* management plans from the CSAR file are able to read the topology and instance information of nodes and relationships.

Finally a deployed application is instantiated by running it's corresponding build plan, started via a *Admin UI* of a *Self-Service Portal*, or by a SOAP (Simple Object Access Protocol) message.

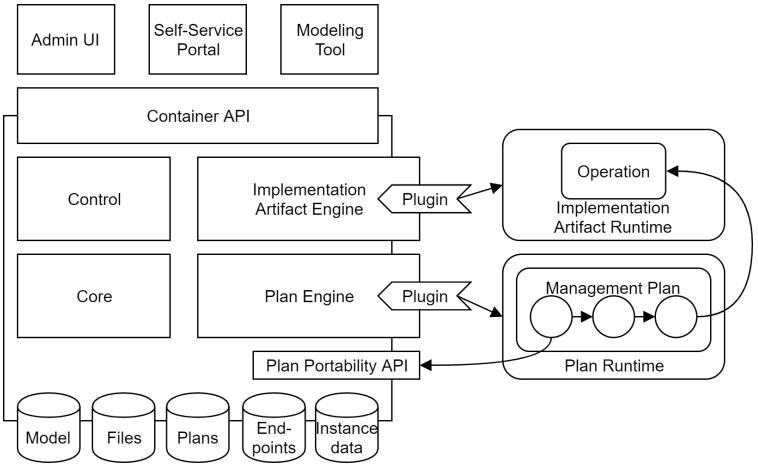


Figure 3.4: Overview of the in [BBH+13] presented architecture of the OpenTOSCA package.

OpenTOSCA allows deploying applications in a dynamic fashion and is thus able to manage cyber physical systems, like the SitOPT project needs. TOSCA/OpenTOSCA and SitOPT is powerful combination to allow the automated and dynamic management of workflows and applications within a cyber-physical system.

4 Related Work

This chapter presents some scientific records about CAS, situation prediction and machine learning approaches within these topics. Different papers are briefly introduced and their reference to this thesis is shortly emphasized.

4.1 Proactive Hard Disk Failure Detection

Pitakrat et al. compare in their work [PHG13] different machine learning models for a proactive predicting approach for hard drive failures. They appeal to the problem of reactive actions to occurring problems, since it means that if an actual malfunction occurs, the system can only take measurements afterwards. Reactive behavior causes many problems. If for example a user forgot to do a backup, reacting to a crash with backing up the hard drive is not possible anymore. Further, if a hard drive crash can be predicted in advance, measurements like buying a new one can be initiated, to ensure faster resuming of work, depended on that hard drive. They assume that malfunctions of hard drives are loomed by anomalies in system parameters like CPU or memory usage, network traffic and system load and counter measurements can thus be initiated preemptively. Detecting these anomalies enable immediate counter measurements to take place, without any delay.

The problem of predicting failure is transferred into a binary classification problem. With a classifier trained by labeled observations of the system, indicated as failed or not failed, they aim to predict failures preemptively. They used several different machine learning approaches to tackle this problem. The approaches include probabilistic models Simple- and Multinomial-Naïve Bayes Classifier and a Bayesian Network, decision tree based models C4.5, REPTree and Random Forest, rule-based models ZeroR, OneR, Decision Table, RIPPER and PART, hyperplane separation models Support Vector Machine, Sequential Minimal Optimization and Stochastic Gradient Descent, function approximation models Simple and Normal logistic regression, Multilayer Perceptron and Voted Perceptron, and finally instance-based learning models Nearest Neighbor Classifier, K-Star and Locally Weighted Learning. These different models are compared against several evaluation metrics, further against their prediction quality produced by a ten-fold cross validation and lastly against their training and prediction time.

Overall, they reveal good results, leading to the assumption; machine learning is capable of proactive failure detection in a binary classification problem. However, they admit that the approach to choose highly depends on application constraints and need to consider evaluation metrics as well.

4.2 Machine Learning Based Adaptive Context-Aware System for Smart Home Environment

Smart home environments are a special type of CAS, if a specific context occurs, several specific workflows are triggered. The work Kabir et al. in [KHSY15] approaches a very similar problem, like considered in this thesis, by using a specific machine learning approach to learn contexts and improve adaptions within a CAS. This CAS is a smart home environment, which is aimed to be turned it into a rational agent, meaning an agent that allows choosing the best workflow for a given context. As a goal, this work aims to predict user needs based on a smart home context, while users returns feedback information in form of agreement or disagreement of the changes made in the context. This feedback is then dynamically used to adapt predicted context information. They design the CAS based on a three-layer approach with a sensor layer, a middleware layer and an application layer. Like in CAS mentioned in Section 3.2, the sensor layer gathers information of the context. Further, the middleware layer controls sensors and models contexts. The application layer holds a machine learning based service selection module, which learns user behavior from specific context configurations. Within the latter layer, a context receiver module selects services, based on a context, which is given by a set of parameters measured by various sensors. Afterwards, a supervised Neural Network within a service selection module is used to learn the correct services from measured sensor data, and to predict user needs proactively. The learned model is stored as a knowledge base within the application layer. Users can then change their requirements and give according feedback to the system. The system then needs to adapt its workflows in real-time, using a adaption module. For example if a user happens to adopt a child, the smart home should nor raise shutters as early as it had to before the presence of the child. To enable these adaptions, a reinforcement learning model called temporal differential is used to update the knowledge base, based on the users given feedback. The system overall is a proactive system, due to learning contexts from users, predicting them and use it to support the user in its everyday life proactively. However, the adaption with a reinforcement algorithm is a reactive behavior, since this algorithm needs to wait on user's reactive feedback to the environmental changes made by the CAS.

4.3 Situation-Aware Management of Cyber-Physical Systems

Képes et al. address in their work [KBL19] the processing of structural changes of application within cyber-physical systems (CPS). A CPS is a rough description of complex computer systems, where hard- and software components interact closely. They uses a smart home scenario for validation of their findings. Their implementation is based on the TOSCA specification language, and further updates the basic SitOPT concept, presented in Section 3.4. The basic component supporting this approach, is formed by so-called *Application Packages*, which are collections of different models describing the application. Particularly, *Deployment Models*, *Situation Detection Deployment Models* and *Situation-Aware Management Process Models*. Latter equips tasks, the application can make, with process triggers formed by situations, which enable that certain process. These models are uploaded onto a *Deployment & Situation-Aware Management System*, where they are instantiated by installing and uploading all necessary files and packages. After instantiating the models, a user can create an application, by requesting the corresponding packages and files to install. An application then uses its *Situation Detection Model* instance, to observe situations and forward

their stats to the *Situation Detection Layer*, where it is matched against situation-aware processing triggers. If a situation hence triggers, the corresponding management process is instantiated, and interacts with the running application.

Changes to applications and their configurations are made in reactive manner, based on specific situation triggers. At this point, enabling proactive situation detection, would allow faster adaptions of application configurations and overall cyber-physical systems, to save costs and resources.

4.4 Context-Sensitive Adaptive Production Processes

Sungur et al. [SBLW16] investigate in the adaption of processes within cyber-physical smart factories. They aim to automate the adaption of workflows to changes in production processes due to alternative products to be produced. By analyzing a manufacturing production process, they are able contribute new ideas to enable this contribution within a smart factory as CAS. These ideas include *Context-Sensitive Execution Steps* (CES) and *Context-Sensitive Adaptive Production Processes*. CES are process modeling structures, that allow workflow changes based on contexts or *Context Definitions*, with the aim to fulfill a *Main Realization Process*. Latter is defined by a *Main-Intention*, representing the desired outcome of a specific process. A CES receives input data and starves until the input matches its context definition. Afterwards, the system evaluates context definitions of all present realization processes, and as long is the corresponding context definition for the current CES is alive, the CES stays alive too. If realization processes with the same main-intentions exist, all but one processes are dropped. After optimizing the CES, it is eventually executed. To use CES, the authors introduce to context-sensitive adaptive production processes, which serve as an enhancement for regular production processes.

With this approach, Sungur et al. allow reactive adaption of production processes to contextual changes. This thesis ties to the presented work, by enabling the prediction of the contextual changes and hence allow proactive adaption.

4.5 Bayesian Games for Threat Prediction and Situation Analysis

A rather abstract and completely different approach to situation prediction, is conducted by Bryniels-son and Arnborg in [BA04]. They examine situation awareness of *Command and Control* scenarios, through usage of Bayesian Games. Bayesian Games are thereby game scenarios with incomplete information about the context. In such scenarios, there exist actors who can make decisions leading to so called consequences. A consequence induces a utility score, as a contribution to a specific goal. Gaming in this context means, different agents pursuing different strategies to win a game and hence working against each other. This type of context and situation treatment can easily be transferred into economics and industry, where companies try to use the best strategy to reach a certain goal, with other contenders trying to do the same.

They aim to predict decisions of agents in a certain situation, by using extended inference diagrams and a tree look-ahead algorithm. A scenario or game starts with a so-called *historical chance node*, representing the past, which is partially known to all contending agents. Parameters are represented by random variables, which are independent of decisions took by agents. The root node is followed

by a set of possible decisions the agent can take, followed by uncertain decisions made by other agents, and again followed by the former agents' decisions, and so on. The constructed tree is afterwards solved bottom-up, to find the *Nash-Equilibrium*, a solution, contributing strategies for each agent, from which no agent can foresee, without losing some kind of benefit.

This type of situation-awareness and prediction is a more competitive approach, and should rather be considered as future work to this thesis. As after considering the adaption of production workflows proactively depend on different actors, contributing decisions that may change order settlements.

4.6 A Machine Learning Approach to Workflow Management

In [Her00], Herbst present an approach to overcome the time consuming tasks of workflow adaption and also acquisition. They focus on applying machine learning techniques, to overcome the problem that commonly experts are needed, for the workflow adaption and acquisition. They distinguish between *workflow models* and *workflow instances*, to formulate their problem as induction of workflow models. Therefore, given a set of workflow instances, they use machine learning to find a workflow model structure for the given workflow instances. Further, they differentiate between sequential and concurrent workflow models to find their solution.

Their approach does not exactly adapt workflows itself, but provide workflow models, which can replace current workflows and serve as adaptions.

4.7 The Role of Machine Learning in Scientific Workflows

In contrast to this thesis by enhancing workflow procedures, the work of Deelman et al. in [DMJS19] tackles the design and selection of workflows with the help of machine learning approaches. They state, that managing workflows such as deal with components, provisioning and execution is conducted by users so far, but can be enhanced with appropriate machine learning algorithms. With a set of knowledge bases for an application, so far users search fitting workflows within these knowledge bases. This task may be relieved by machine learning, since it is possible to learn relevant workflows, based on users' preferences and the data the workflow is used on. They suggest that machine learning approaches may be able to offer a variety of different fitting workflows and are able to compare them. Problems within scientific workflows, such as the structural workflow design, resource selection and scheduling workflow tasks could be leveraged by machine learning. Therefore four different tasks have to be tackled, to allow machine learning models to analysis of workflow behaviors. First, the *workflow-level analysis* discovers the behavior of jobs within a workflow, by using unsupervised clustering algorithms to learn about similarity of workflows. The second tasks, *task-level analysis* investigates in conspicuous workflow behaviors. Third, *infrastructure-level analysis* deals with inconsistencies of resources used by workflows. *Cross-level analysis* investigates in using applications as well as whole systems to detect performance issues. Lastly, *on-/off-line analysis* investigates in performance of workflows for online and offline performance behavior. The authors state, that with the help of machine learning for scientific workflow improvement, productivity would increase. However, an induced problem with the usage of machine learning to discover workflows would be the aggravation of reproducibility.

4.8 Time Series Prediction Using Support Vector Machines

In [SS09], Sapankevych and Sankar investigate in *Time Series Forecasting* to predict orders, with the application of regression Support Vector Machines. Therefore, they review the application of SVMs on many different use cases such as general business applications, environmental parameter estimation, electric utility load forecasting, machine reliability forecasting a control system and signal processing applications. Time series are defined by a set of past and current data samples, where a prediction depends on the whole data set. A prediction is the next value in the series driven by the past. They allow the usage of SVM regression for *Time Series Forecasting*, the authors depict what changes has to be made to accomplish this. With their review, they find that SVMs for *Time Series Forecasting* do allow overall better results than neural networks within this topic. Further, they address some challenges coming with the usage of SVMs, for example selecting the correct Kernel function, the correct optimization technique and metrics for evaluation are some of the problems difficult to solve.

Regarding situations as a time series, where the occurrence of a situation may depend on the whole past, seems also like a reasonable approach to solve the problem of situation prediction. SVMs promise a very powerful application for this type of problem setting and hence may perform similarly well in the classification case.

4.9 Network Security Situation Prediction

Another closely related investigation is conducted by Leau and Manickam in [LM15]. They research and compare different approaches to predict the situation or status of the network security for a system. They state, that with the growing size of information sharing and resource access, the vulnerability within networks also increases and that security defence mechanisms of the current state of the art are insufficient. Therefore, they suggest that networks should act like CAS, and use system context data to interpret the situation of a network to predict future network security states, based on current and historical data of the network state. They compare the prediction of network security states of a Neural Network, a Markov Model and a Grey Theory approach. Their results show good results with the Neural Network approach, but bad results for the Markov Model. However, the Grey Theory in contrast to the other two methods does not need large data sets to be trained and computes faster better results than the Neural Network.

According to their work, could the Grey Theory be the next step in situation prediction for system workflow management systems, in which this thesis investigates.

4.10 Situation Prediction of Large-Scale Internet of Things Network Security

The work in [YZWM19] of Yang et al. follows a similar investigation than [LM15]. They also investigate in the improvement of network security but for the *Internet of Things*. Looking at situation awareness by means of randomness, time-sequence and complexity, they propose a time series forecasting approach for situation prediction of network security. The time series analysis

is conducted with a modified Support Vector Machine model with a sequence kernel and particle swarm optimization. With their results, they find that the modified SVM is feasible for the situation prediction approach and even more effective with higher accuracy than the original SVM without particle swarm optimization.

With their findings, they support not only the situation prediction by means of SVMs, but also the formulation of situation prediction as a time series approach.

5 Concept & Algorithms

This thesis aims to develop a system, which eases the usage of machine learning prediction algorithms within cyber-physical systems. The developed concept allows abstracting the procedure of situation prediction for business processes and workflows, and hence enables proactive workflow adaption and reconfiguration for non-specialists of machine learning.

5.1 Concept Overview

In the following, this section will describe the abstract architecture, which allows the usage of different machine learning algorithms to predict situations in form of confidence scores. This architecture allows users to make predictions, without expertise within machine learning, by completely hiding any details about the underlying algorithms. Hence, other systems can integrate this developed system, without any machine learning related implementation. The only prerequisite is a trained machine learning model for the according use case, which although can be developed and implemented independently by a third party. In the further sections different machine learning approaches are discussed, used to build predictive algorithms for the given use case scenario introduced in Chapter 2, where situations are represented as order settlements. These algorithms serve as exemplary models to predict confidence scores, but can be replaced with other applicable machine learning algorithms.

Figure 5.1 describes the abstract architecture of the framework design. An external service provider represented by a *Situation Service* is able to generate new situations, which are independently stored within a data set, representing a historic set of situations. A *Situation-Aware Workflow Management System*, like it is presented in [KBL19], holds workflows and is able to observe new generated situations by the *Situation Service* and triggers workflow adaptions and reconfiguration based on these situations. Apart from receiving new situations, the *Situation-Aware Workflow Management System* can also receive notifications for changes within another already existing situation. The *Situation-Aware Workflow Management System* is aware of the existence of situations and is thus able to model situations based on given situation templates. Templates define the basic shape and parametrization options for different situation. The workflow management system is then able to ask the *Situation Confidence Predictor* to compute a confidence score for such parametrized situations. The predictor does compute the confidence by means of a machine learning algorithm. Therefore it collects a fix set of parametrized situations of the historical situation set provided by the *Situation Service*, to learn patterns from the historical data. After discovering patterns (training) from the data, a model is able to assign confidence scores for the certainty of an unseen situation, which the workflow management system asked for. This confidence score is then assigned to that situation within the workflow management system, and allows further processing with this information. If the

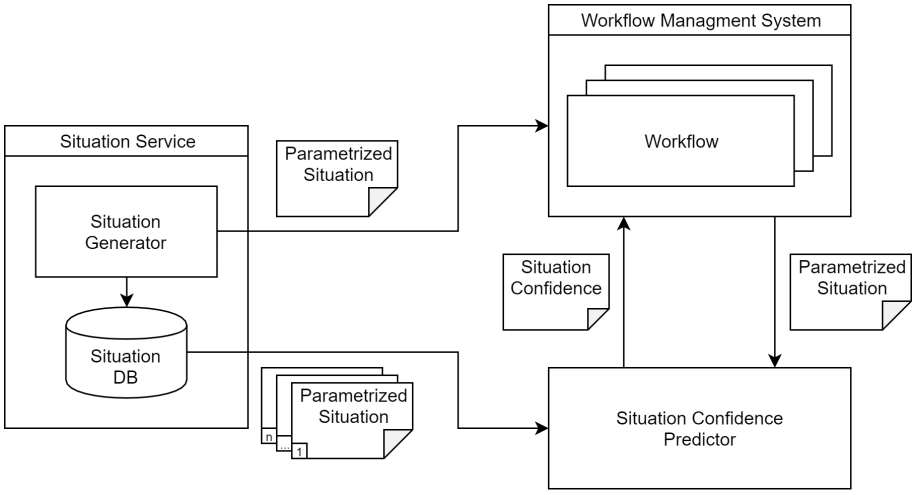


Figure 5.1: Abstract description of the framework integration into a situation-aware workflow management system, consiting of three comunicating components, *Situation Service*, *Workflow Management System* and *Situation Confidence Predictor*.

system considers the confidence sufficiently high, it is able to trigger measurements to reconfigure and adapt workflows, according to the predicted situation occurrence. However, it is not obligated to make use of this additional information.

To test the applicability of machine learning algorithms, with the ability of predicting situation confidence scores, the developed concept is validated against a use case scenario for predicting order settlements as situations to adapt production workflows, as it was presented in Chapter 2. Transferring this presented architecture to the use case, the *Situation Service* represents a customer, generating orders from time to time, which can be treated as new situations. Generated orders are stored within some kind of database, for example any Enterprise-Resource-Planning (ERP) system and can be accessed by any appropriate API or service. New orders are sent to a manufacturer, representing the *Situation-Aware Workflow Management System*, which then adapts its workflows and production processes accordingly to the ordered product. The manufacturer is also able to provide a manually parametrized order to the *Situation Confidence Predictor*, which in turn responds with a confidence score for that exact order. For the use case, the predictor can be chosen out of the following three different machine learning models: *Support Vector Machine*, *Multilayer Perceptron* and *Random Forest*. These predictors were trained with a set of historic order settlements. Table B.1 depicts how orders within this data set, as well as new orders are parametrized. This parametrization of an order includes information about a product (What?), from a certain customer (Who?), on a specific date (When?) and the quantity of products ordered (How much?). Details about the data set used for training and the predictions, can be found in Chapter 7. After the *Situation Confidence Predictor* answers with a corresponding score to the manufacturing workflow, it can be decided if the computed confidence score exceeds a certain threshold and the production process is to be initiated or adapted, even if there is yet no order settlement received. This allows to overcome the delay a reactive production process would arrange in order to process the received order. An important benefit to mention is, customers or the *Situation Service* won't recognize the usage of the *Situation Confidence Predictor*. If the *Situation Service* does not expect a response, then there is indeed no recognition. However, if a response is expected, like in the use case the delivery of the product, the

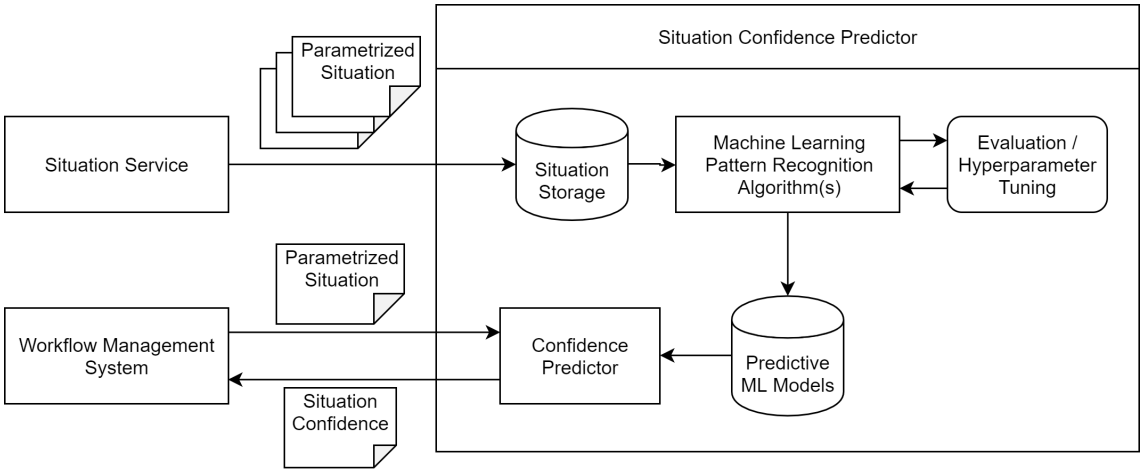


Figure 5.2: Detailed overview of the *Situation Confidence Predictor* architecture with a situation storage, ML model training and evaluation, and confidence prediction components.

only thing changing for the customer is an experience of faster receptions of ordered products or general service adaption to situations, due to the overcoming of the adaption and re-configuration delay.

In the following sections, the conceptual details about the *Situation Confidence Predictor* and the different machine learning models are presented. To make use of the presented concept, it is added to the TOSCA based implementation SitOPT for cyber-physical systems by Képes et al. developed in [KBL19]. Details about this integration are presented in Chapter 7 and are not discussed here.

5.2 Situation Confidence Predictor

This section focuses on the prediction API design, which allows generating confidence scores for requested situations or orders in this use case. Figure 5.2 shows the abstract internal structure, where parametrized situations are collected into a *Situation Storage* or database. After that, one or more machine learning algorithms use this data to train and learn patterns from it. The procedure is refined and repeated with hyperparameter tuning and evaluation of the models, until they produce acceptable results on a test data set. Once a model has finished its training, it is stored and available for future usage. The *Situation Confidence Predictor* then receives a request of a new order, with a selected model to use for predictions. It uses that model to generate the confidence score, which in turn is then sent back to the requestor, allowing him to take advantage of the confidence score.

The machine learning approaches that were chosen for this work, are based on supervised classification algorithms, which allows to classify requesting orders into “yes it is an order” or “no it is not an order”. As in the use case in Chapter 2 described, the current data only contains positive entries, indicating a yes or simply a 1. To overcome this problem, one big preprocessing step is to generate so called “negative samples”. They enable the machine learning model, to learn not only positives, but also distinguish between positive and negative orders, and hence classify requests into 1 or 0 respectively. Details about how this negative sampling is realized are also described in the implementation details in Chapter 7. Combining positives and negatives, a resulting total number

of $56,544,000$ samples is hold by the data set. The next preprocessing step includes the encoding of parameters, leading to a total number of $4,484$ parameters for each sample. More details about how this encoding process took place, can be found in Chapter 7. The size of this data explodes with this enhancement, and due to memory restrictions, it has to be partitioned into chunks of data from only two years. Further processing of data is thereby done with data from years 2010 and 2011 only. In order to train and evaluate a machine learning model correctly, the data has to be split into two parts by a 80/20 rate to serve the training and test set respectively.

With the property of positive and negative samples, the next step is to look for classification algorithms that are able to compute confidence scores for such order samples. Since confidence is a probability value, probabilistic approaches are the algorithms to choose here. In the fundamentals in Chapter 3, Scikit-learn [PVG+11] was introduced as the go-to library for machine learning algorithm implementations in python. This package holds several good applicable algorithms, which are used in this thesis. Some of the classification algorithms offer a probabilistic version for predictions.

To generate the desired predictive model, this work compares three machine learning approaches, which comply with the conditions stated. These three algorithms are explained in detail in the following sections of this chapter. Section 5.3 presents the *Support Vector Machine* [PVG+11] (SVM) combined with *Stochastic Gradient Descent* (SGD) as optimization, promises to be a fast solution for large scaled [Bot10] and high dimensional data. In Section 5.4, the *Multilayer Perceptron* [PVG+11] (MLP), a neural network approach, as a type of machine learning model which shows overall good results in pattern recognition [RIGE17], with the shortcoming of long training times. The last of the three model, the *Random Forest* [PVG+11] (RF), is presented in Section 5.5. It is a decision tree based ensemble approach, which combines multiple learners to achieve better results [CCS12]. Further it promises overall good results in recent research [PHG13] and as a very fast trainable model, due to the usage of binary decision trees.

Each model computes different probability values and results, since they differ in performance, which is reported later in the results in Chapter 6.

5.3 Support Vector Machine

The first algorithm, used in this thesis to predict confidence scores, is a *Support Vector Machine* (SVM) [PVG+11; Vap98], invented by Vapnik. Originally SVMs were used for classification only, but nowadays there exist also applications for regression problems (e.g. for Time Series Analysis [SS09]). The SVM algorithms goal is to find a linear surface (called hyperplane), that can split the data set of two different classes into two distinct parts. Generally data points or samples are placed within a D-dimensional space, where D is the number of parameters of a sample. A hyperplane is then placed between the two sets of data points, such that as much points as possible of one set are on the one side and points from the other set are on the other side of the hyperplane. Further, to support generalization of the model, the margin between these points and the hyperplane is to be maximized. If the given data sets are not separable by a hyperplane and data points are allowed to violate the margin, it is called a soft margin; otherwise, it is a hard margin. One way to overcome the problem of inseparable data sets is the use of the kernel trick. This trick increases the dimensions of the source data, such that the data set is separable within the new generated dimension. To generate that new dimension, different functions can be used such as radial basis functions and polynomial functions. However, the parametrization of the used SVM uses a linear kernel, due to the high

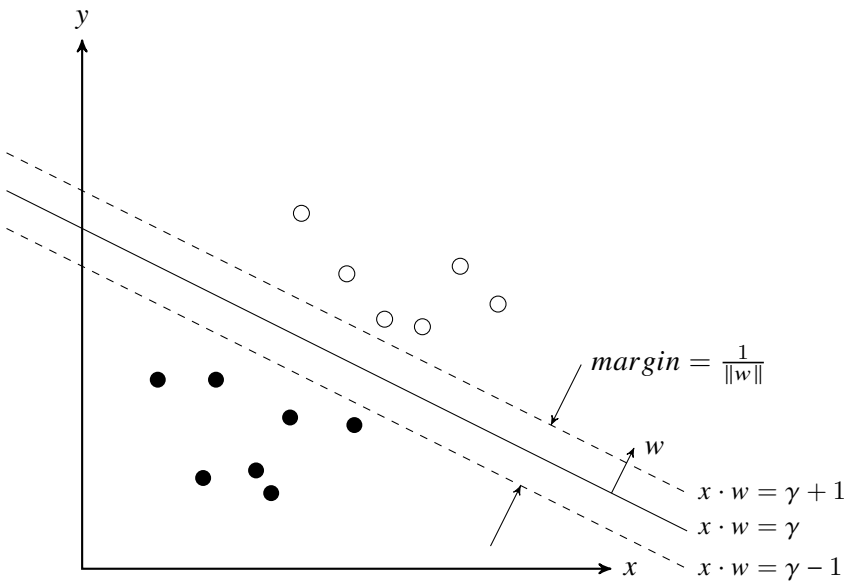


Figure 5.3: Two dimensional Support Vector Machine with a data separating One dimensional hyperplane $xw = \gamma$.

dimensionality of the data and the massive time consumption of training SVMs with non-linear kernels. Figure 5.3 shows the SVM for a two dimensional space. A hyperplane is separating two sets of data points, where each point lies outside the dictated margin.

Within a SVM [BGV92], input data is represented by D-dimensional vectors x, with output classes y. The closest sample points to the separating hyperplane, but outside the margin, are called *support vectors*. Labels of the two classes indicate on which side of the hyperplane the samples lie. The hyperplane represents a decision function which labels input samples x_i with output y_i. Reformulating the margin functions in Figure 5.3 with labels able to take values 1 and -1, the SVM problem can be formulated as follows:

$$w \cdot x_i + \gamma \geq 1 \quad \text{for } y_i = 1 \tag{5.1}$$

$$w \cdot x_i + \gamma \leq -1 \quad \text{for } y_i = -1 \tag{5.2}$$

$$\Rightarrow y_i(w \cdot x_i + \gamma) - 1 \geq 0 \quad \text{for } y_i \in 1, -1 \tag{5.3}$$

With x representing a parametrization of a sample, weighted with w and γ, where w weights each sample parameter and γ determines the position of the margin function relatively to the origin. This leads to the distance between both margins $M = \frac{2}{\|w\|}$. With the aim of the SVM to maximize the distance between a margin and the support a support vector, the problem to solve is stated as:

$$max\frac{1}{\|w\|} = min\|w\| \tag{5.4}$$

It can thus be reduced to find minimal weights w. This formula is further reformulated, due to using the l2 regularization term:

$$min\frac{\|w\|^2}{2} \tag{5.5}$$

Since there are possible margin violating samples, which are rather unclear to classify, the so called error on how strong a sample violates the margin, as slack variable ξ, is introduced. They are subtracted or added as a weight for an error to Equation (5.1) and Equation (5.2) respectively:

$$w \cdot x_i + \gamma \geq 1 \qquad \text{for } y_i = 1 - \xi_i \tag{5.6}$$

$$w \cdot x_i + \gamma \leq -1 \qquad \text{for } y_i = -1 + \xi_i \tag{5.7}$$

$$\Rightarrow y_i(w \cdot x_i + \gamma) - 1 + \xi_i \geq 0 \qquad \text{for } y_i \in 1, -1 \tag{5.8}$$

This leads to a modified optimization problem, where incorrect classifications are penalized with some constant C weighted for each violation of sample x_i, with weight ξ_i. With the reformulation of the constraint, the following equation is used:

$$min\frac{\|w\|^2}{2} + C \cdot \sum_{i=1}^{n} \xi_i \qquad \text{s.t. } \xi_i = max(0, 1 - y_i(w \cdot x_i + \gamma)) \tag{5.9}$$

The constant C represents the regularization parameter and if it is chosen to be 0, errors are not penalized and hence the hyperplane can be placed anywhere. On the other hand, if $C \to \infty$, errors are not allowed and the hard margin problem is induced.

To find the maximum margin now, this problem has to be solved by finding optimal parameters w and γ. This can be achieved with various optimization methodologies. A solution for large scaled problems is provided with the optimization method SGD [Bot10] as already mentioned. Normal Gradient descent is an optimization method to find optimal parameters by searching a minimum within a function, based on a learning rate that suggests how big the step size is to reach the minimum. If the step size is too large, the minimum will be overshoot and never be found, if it is too small, it takes very long to find the minimum. The SGD differs from the normal gradient descent method, by not computing the gradient at every optimization iteration. Instead, it is estimated by choosing a random sample (x_i, y_i). The formula for the SGD is as follows:

$$w_{t+1} = w_t - \alpha_t \nabla_w loss((x_i, y_i), w_t, \gamma) \tag{5.10}$$

Where for each step, w_t is dependent on the random chosen sample and ∇ represents the gradient operator as the derivative and α as the learning rate for each step. The SGD expects a faster evaluation of the optimal parameters, than the simple gradient descent methodology as it is depicted in [Bot10]. However, other optimization methods are suitable too.

As depicted below in the chosen parameters for the SVM classifier (see Table 5.1), the loss function for the SVM is the *modified huber loss*.

Parameter	Description	Value
`loss`	Loss function	modified_huber
`penalty`	Regularization term	l2
`alpha`	Multiplier for regularization term, impacts learning rate	0.00001
`max_iter`	Maximum numbers of iterations over data	1
`learning_rate`	Learning rate schedule (α)	optimal
`early_stopping`	Terminate training if validation score does not improve	True

Table 5.1: SVM classifier training parameters, found with grid search and cross validation for the use case data set.

$$loss(y_i, x_i, w, \gamma) = \begin{cases} max(0, 1 - y_i(w \cdot x_i + \gamma)) & \text{if } y_i(w \cdot x_i + \gamma) \geq -1 \\ -4y_i(w \cdot x_i + \gamma) & \text{else} \end{cases} \tag{5.11}$$

Transferring the use case to the SVM, the SVM is trained by a set of labeled positive or negative orders, taking places in the $4,484$-dimensional space. The SVM calculates the optimal weights for a hyperplane that separates those two classes as good as possible. This calculation may take a very long time. After the optimal hyperplane is found, new orders are thrown into the $4,484$-dimensional space and it is easily observable to which class they belong. The probabilities for each sample are computed by the distance of each sample to the hyperplane. The further away from a hyperplane, the closer to probability 1.

The parametrization of the used SVM (see Table 5.1) takes several arguments, whereby the best parameters are found by a grid search and cross-validation. As described above, the `loss` represents the function for which the problem is optimized. The `penalty` and `alpha` parameters define the regularization term for the model. The number of iterations over the data set, to find the optimal solution, is set by the `max_iter` parameter. With an optimal `learning_rate`, the resulting value of the formula $\mu_0/(1 + \texttt{alpha} \cdot \mu_0 \cdot t)$ is used as learning rate, with `alpha` as described in Table 5.1. The computation of μ_0 is determined by investigations in some samples of the data. In this work, not all parameters were evaluated, only parameters that are depicted in Table 5.1. More details about how they were found is described in Chapter 7.

5.4 Multilayer Perceptron

The MLP [PVG+11; RIGE17] is a neural network approach, which consist of a single input layer, some hidden layers and an output layer. All layers are composed by so-called neurons, which take input values, multiply a certain weight to them, combine them and pass them to the next layer. The number of parameters of the prediction problem dictates the number of neurons in the input layer, whereas the size of the output layer matches the number of targets to predict. In a neural network, multiple hidden layers can exist. The user is able to define the size of each hidden layer manually. A neuron has a link from each neuron of the previous layer to itself, and passes its output to all neurons of the next layer. Neurons within the same layer are not interconnected.

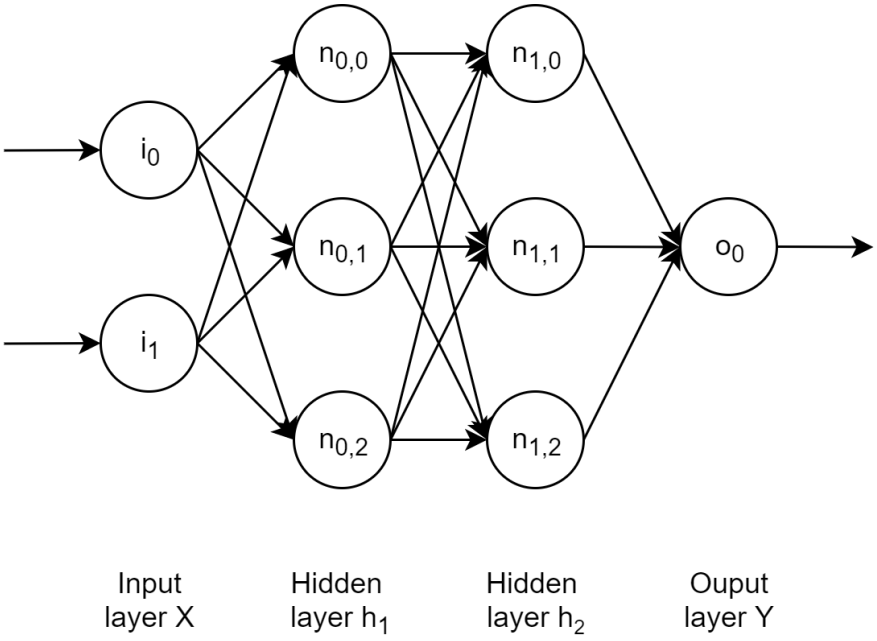


Figure 5.4: Structure of a simple neural network with a two dimensional input feature space, two hidden layers composed of three neurons each, and a single output target value.

Generally, the procedure dictates each neuron from each layer to calculate its output value and to propagate it to the next layer, to compute an output of the network. This is repeated until the output layer Y is reached. Figure 5.4 shows how such a neural network is structured.

Another main part of neurons is the activation function $f(X)$, which can be chosen on initiation, and which defines if a neuron propagates its computed value. Further, each connection between neurons is labeled with a weight, indicating how the value is changed by a neuron. Given hidden layers h_i with $i = 1, ..., N$, and n_i indicating the number of neurons within a hidden layer h_i, the output of a neuron for a single hidden layer is defined by the following term:

$$h_j^i = f(\sum_{k=1}^{n=i-1} w_{k,j}^{i-1}, h_{i-1}^k) \qquad i = 2, ..., N \text{ and } j = 1, ..., n_i \tag{5.12}$$

With $w_{k,j}^{i-1}$ indicating the weight (connection) between the neuron k of the previous hidden layer h_{i-1} and neuron j of the current hidden layer h_i. Thereby the output of a complete hidden layer is defined by:

$$h_i = \begin{pmatrix} h_i^1 \\ \vdots \\ h_i^{n_i} \end{pmatrix} \tag{5.13}$$

The input layer replaces h_{i-1}^k in Equation (5.12) with the input values x_k, where $X = (x_0, ..., x_{n_0})$ is the feature vector of the prediction problem.

Parameter	Description	Value
`hidden_layer_sizes`	Tuple indication number of layers (length) and neurons on each layer	(10,)
`solver`	Optimizer for weights w	adam
`alpha`	Multiplier for regularization term, impacts learning rate	0.1
`learning_rate_init`	Initial learning rate, impacts step-size of weight updates	0.001
`max_iter`	Maximum number of iterations for convergence (number of epochs)	5
`early_stopping`	Terminate training if validation score does not improve	True

Table 5.2: MLP classifier training parameters, found with grid search and cross validation for the use case data set.

The output layer differs also slightly form normal hidden layers, where a neuron y_i of that output layer is defined by:

$$y_i = f(\sum_{k=1}^{n_N} w_{k,j}^N h_N^k) \tag{5.14}$$

Which leads to the target:

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_{N+1} \end{pmatrix} \tag{5.15}$$

The MLP is trained with the labeled order settlements, where the labels define the output of the neural network. Its goal is to find optimal weights for each neuron, such that at the end of the network, the computed output is as close as possible to the true output. To do so, the network is traversed in multiple iterations, to adapt the weights until it cannot reduce the error anymore. Applying a rectified linear unit function to that output layer, the output generates a value between 0 and 1, representing the probability for the classification of an input sample. The amount of neurons to be trained is very high and hence induces a massive computation time.

The use case dictates the number of $4,484$ input neurons converging their information into a hidden layer network of 10 neurons, and further converges them to 1 single output neuron, delivering the classification probabilities. Those closer to 0 represent the complementary probability for negative classified samples, closer to 1 the probability for positive classified samples.

Using grid search and cross validation, the parameters in Table 5.2 deliver the best performance. The `solver` parameter can be regarded as the optimization method, like the SGD for the SVM previously described. With parameters `alpha`, `learning_rate` and `max_iter` also having the same meaning, as for the SVM. The MLP classifier model used for predictions is trained with these parameters.

(a) Decision tree for continuous feature. **(b)** Decision tree for categorical feature.

Figure 5.5: General split procedure of decision trees within a RF, for feature space X_i with descendent partitioned data sets Q_L and Q_R, that fulfills or violates the condition repsectively.

5.5 Random Forest

Breiman as the author of the original paper about Random Forests [Bre01] (RF), gives a very clear definition of what a RF exactly is:

> "A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, ...\}$ where the $\{\Theta k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\mathbf{x}$." [Bre01]

Where x represents the parametrized input vector, and Θ_k represents a random selection of data points for the split of the decision tree and further a random selection of a split.

Generally, a RF [CCS12; PVG+11] is an ensemble algorithm, where many weak learners, represented by binary decision trees, are combined to form one single strong learner. These decision trees partition the input feature space X_i recursively, by so called "splits". A split partitions all features by a condition into two descendant nodes Q_L and Q_R. Split-points define how a split is computed. If the feature, the split is based on is continuous, then the procedure follows the pattern like in Figure 5.5a. If it is categorical, it follows the pattern like it is shown in Figure 5.5b.

A split is chosen from all possible splits of the feature space and is based on some criterion. The criterion used for classification is commonly the *Gini Impurity*:

$$I(p) = \sum_{i=1}^{N} p_i \sum_{k \neq i} p_k \tag{5.16}$$

With N being the number of possible labels (classes) and p_i the probability of an item to be labeled with class i:

$$p_i = \frac{1}{n} \sum_{i=1}^{n} \mathcal{I}(y_k = i) \tag{5.17}$$

Where $\mathcal{I}$ describes the indicator function. The *Gini Impurity* gives a measure for the amount of a random item being incorrectly labeled, by choosing a label from the label set randomly but with distribution according to the given data set. The closer the *Gini Impurity* is to 0 the better the split, leading to the optimization formula:

$$f(x) = \underset{y}{argmin} \sum_{j=1}^{J} \mathcal{I}(h_j(x) = y) \tag{5.18}$$

With $h_j(x)$ as the target for input x, decided by decision tree j.

A RF assumes the feature space to be unknown joint distributed. With the zero-one loss function:

$$loss(y_i, f(x_i)) = \mathcal{I}(y_i \neq f(x_i) = \begin{cases} 0 \text{ if} & y_i = f(x_i) \\ 1 \text{ else} \end{cases} \quad \text{for } y_i \in Y, x_i \in X \tag{5.19}$$

the prediction function can be determined, since the algorithm minimizes the expected loss:

$$E_{XY}(loss(Y, f(X))) \tag{5.20}$$

leading to the prediction function:

$$f(x) = \underset{y \in \mathcal{Y}}{argmax}\, P(X = y | X = x) \tag{5.21}$$

with $\mathcal{Y}$ as the set of different classes ($|\mathcal{Y}| = N$).

A decision tree of a RF takes two arguments. First, a randomly selected subset of orders. Second, a randomly selected subset of features to compute the split on. With those arguments, a split is computed by finding the feature that delivers the best *Gini Impurity* according to the random subset of samples. The *Gini Impurity* thereby tries to minimize the number of incorrect classified orders, for each of the specified features, and eventually picks the smallest. This procedure is repeated in the subsequent splits, until there is only a single sample left within a leaf node.

For the given use case, experiments came up with an optimal number of 15 decision trees as depicted in Table 5.3. An order is sent to each of the 15 decision trees and propagate through it. Within each tree the order is classified by the 67 ($\approx \sqrt{4484}$) features the split nodes had selected in the training, due to the `max_features` parameter. Each tree then votes for a specific class. The probability for that class is then estimated by the fraction of orders with the same class, from the training data set within that leaf. To compute a confidence score for a requested order, the RF takes the mean value of the probability values of each decision tree.

The resulting parametrization from cross validation and grid search for the RF, used here to predict the confidence scores, is given in Table 5.3. The parameter `max_depth` set to `None` splits set sizes of minimum number of `min_samples_split` samples within the set to split, until `min_samples_leaf` number of samples are left within a leaf.

Parameter	Description	Value
`n_estimators`	Number of decision trees within the forest	15
`max_depth`	Maximum depth for a decision tree within the forest	*None*
`min_samples_split`	Minimum number of samples needed to compute split	2
`min_samples_leaf`	Minimum number of samples a terminal node must hold	1
`max_features`	Number of features considered by splitting	*sqrt*

Table 5.3: RF classifier training parameters, found with grid search and cross validation for the use case data set.

6 Evaluation Results

In the previous chapter, the *Situation Confidence Predictor* was introduced, which allowed to generate confidence scores for parametrized situations. The generation of the confidence scores was based on three different classification machine learning algorithms. To evaluate the applicability of the classification algorithms, they have to be evaluated to generate reference of their performance. This chapter focuses on evaluation and performance measures of the predictive models, computed for the confidence scores within the framework. Basic formulas of evaluation for classification algorithms, such as accuracy, precision, recall and support were introduce in the fundamentals in Chapter 3. Here, some on these fundamental evaluation methods based more advanced metrics and visualizations are used to evaluate the results and performance of the proposed machine learning models, *Support Vector Machine* (SVM), *Multilayer Perceptron* (MLP) and *Random Forest* (RF). The different advanced metrics are presented in stand-alone sections, with a brief introduction to their informative value. Further, in each section for a metric the results of all three models are presented. The visualized plots are based on the Scikit-Plot library [Nak17] and the Scikit-Learn library [PVG+11], which simplify the visualization results by offering a wide diversity of metrics. For evaluation, a set of size 20% of the total size of samples is randomly drawn and represents the test set, as mentioned in Chapter 5. All metrics but the learning curve in Section 6.7 use the test data set for computation of their results.

6.1 Accuracy

The first metric, which was already introduced in the fundamentals in Chapter 3, is the accuracy score for classification algorithms. This metric provides knowledge about how accurate a classifier is able to classify given samples into a binary or multiclass classification. For the calculations of the accuracy in this section, the in Equation (3.4) introduced error rate, with y_i as the orders true label and y_i' as the predicted label for order x_i is used. However, the accuracy can also be calculated based on the confusion matrix, with the formula from Equation (3.8). Given this formula, the three trained machine learning models deliver the following accuracy scores:

SVM: 0.9985623080012899

MLP: 0.9996637370657222

RF: 0.9999126635701718

All values are close to 1, meaning almost every sample was correctly classified. RF achieved the highest value, followed by the MLP and the SVM at last.

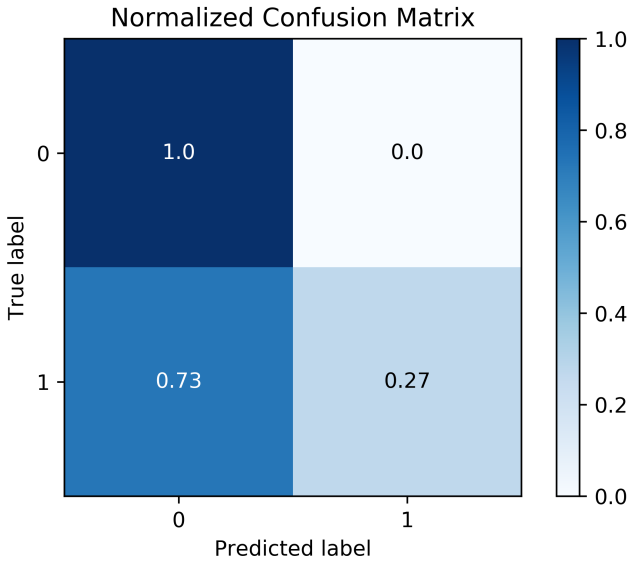


Figure 6.1: The confusion matrix for the Stochastic Gradient Descent classifier, with probability threshold 0.5.

6.2 Confusion Matrix

The confusion matrix is a score that measures performance of a classification model. It is an easy to use and very powerful evaluation methodology for classification problems, that visualizes the in Chapter 3 introduced metrics true negative (TN), true positive (TP), false negative (FN), false positive (FP). With the insights it provides into what errors are made by each classifier, it is a pleasant metric to start evaluation of classification algorithms. Given a binary classification problem, a confusion matrix consists of two dimensions, namely *predicted label* and *true label*, and delivers four different scores. Accordingly, to this, the values for each quarter are calculated based on the presented formulas for TN, TP, FN and FP. Here, class 1 is considered as positive label, and 0 as negative label for the trained classification models, where the positive class represent “order” and the negative class presents “no order”.

The computed values for the confusion matrices for the three different trained predictive models SVM, MLP and RF respectively, are normalized to hold only values between 0 and 1. They are computed by label predictions with classification threshold 0.5. If the probability is larger than 0.5, the order is classified positive, else negative. The SVM classifier in Figure 6.1 labeled 73% of samples with true positive labels as negatives and 27% of true positives labels correctly. The MLP classifier in Figure 6.2 labeled 100% of the samples as negatives, regardless if the true label is positive or negative. The RF classifier in Figure 6.3, as well labeled all true negatives correctly. Further 76% of true positives were correctly labeled and 24% false labeled. For all three models, truly negative samples were in all cases correctly labeled.

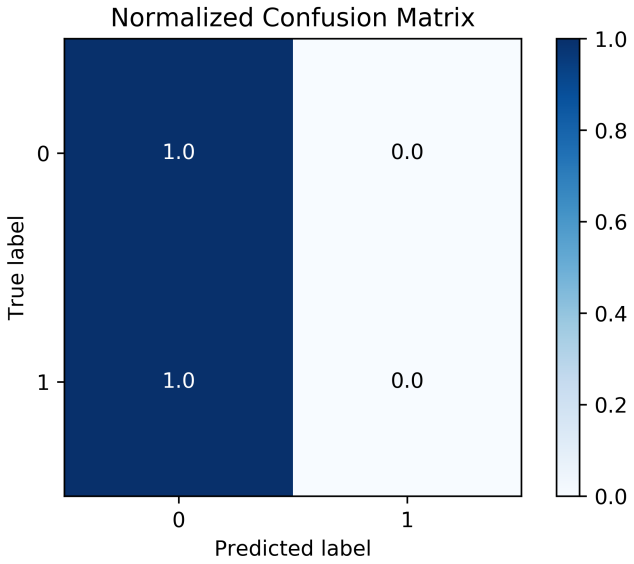


Figure 6.2: The confusion matrix for the Multilayer Perceptron classifier, with probability threshold 0.5.

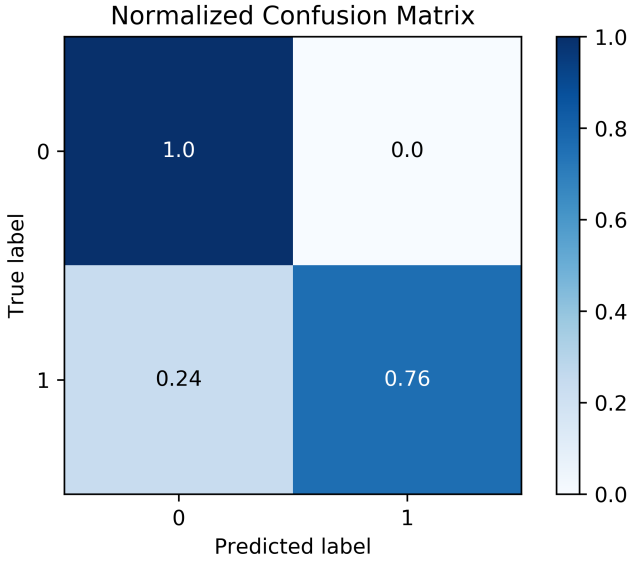


Figure 6.3: The confusion matrix for the Random Forest classifier, with probability threshold 0.5.

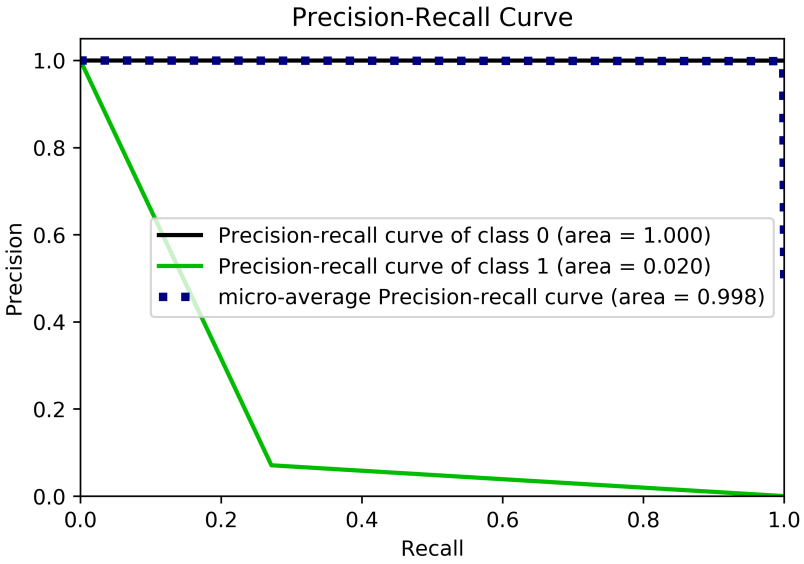


Figure 6.4: Precision-Recall curve for class 1, class 0 and micro-average of the Support Vector Machine classifier.

6.3 Precision-Recall

Precision and Recall are scores defined previously in Chapter 3, that work as an enhancement to the raw accuracy metric. In the Precision-Recall curve, the trade-off between precision and recall of a single class is visualized. This metric allows clarification for results of imbalanced classification problems. For every probability threshold the precision and recall values are calculated and plotted against each other. The perfect curve for a classifier is represented by 100% precision with 100% recall, and hence form a straight line on the top of the plot. The micro-average curve, shown in plot, is calculated by summing up the TP, TN, FP, FN of all classes and calculate precision and recall with the summed versions. It characterizes the equal contribution of all samples to the metric. The endpoint of the micro average however, should not be in any of the right corners, since that would assume all samples as positives (upper right) or negatives (lower right) respectively.

For all Precision-Recall curves of the different used predictive models, the value for class 0 (negative) is 1.0, highlighting that each classifier does always predict negatives correctly. Class 1 for SVM in figure 6.4, shows an immediate loss of precision when increasing recall until the ~ 0.28 mark, where it makes a cut and shows a slower decrease. Figure 6.5 shows a similar behavior as the SVM classifier for class 1 of the MLP classifier. Precision decreases fast until recall ~ 1.8, where it makes a cut and decreases slower. In case of the RF classifier 6.6, the graph for class 1 shows almost until recall ~ 0.6 a precision of 1.0, and than a fast decrease of precision when recall crosses the ~ 0.9 mark. For class 0 and the micro-average however, for all three classifiers, the curves show 100% precision for any recall threshold, and hence almost perfect classifiers.

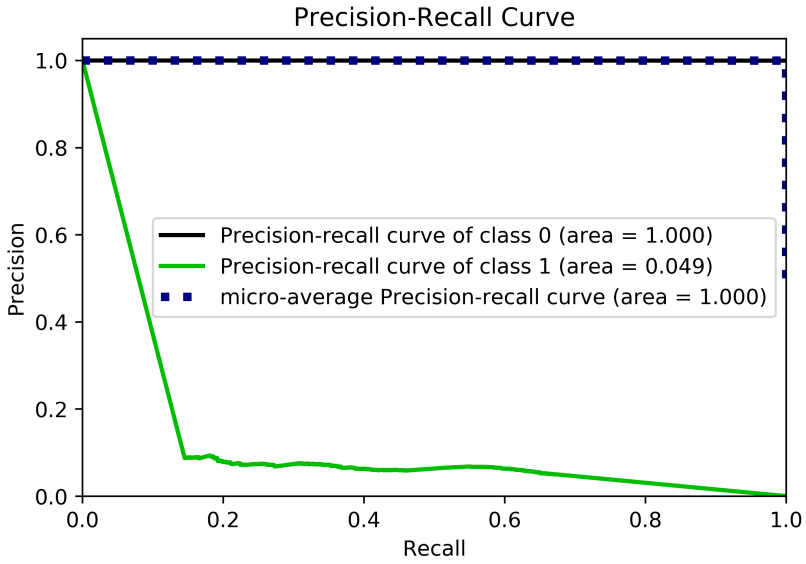


Figure 6.5: Precision-Recall curve for class 1, class 0 and micro-average of the Multilayer Perceptron classifier.

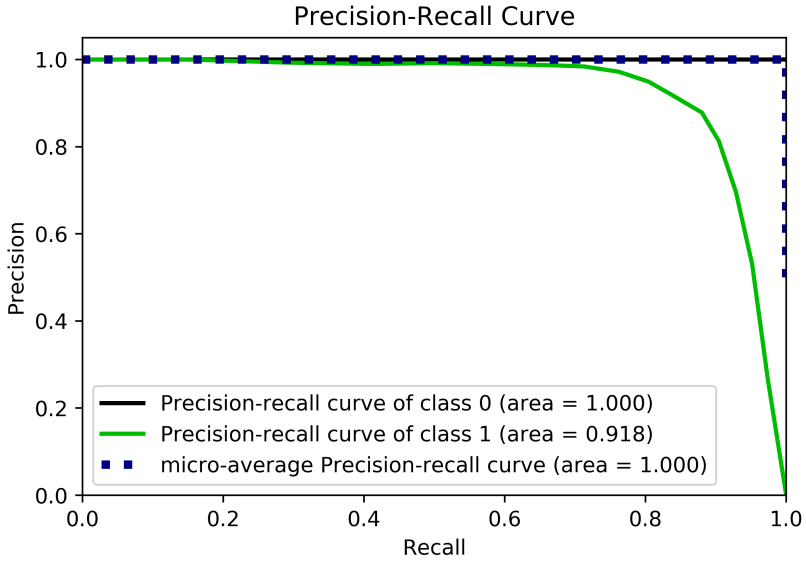


Figure 6.6: Precision-Recall curve for class 1, class 0 and micro-average of the Random Forest classifier.

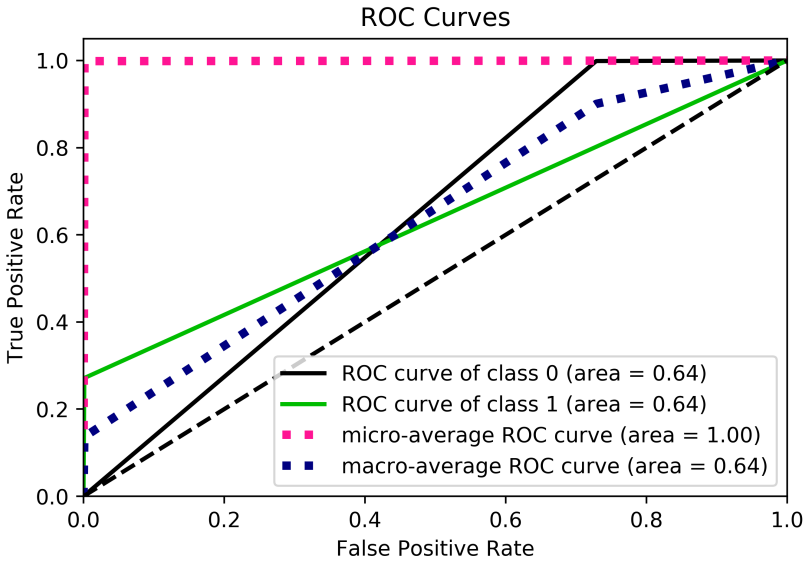


Figure 6.7: ROC curves for class 1, class 0, micro- and macro-average of the Support Vector Machine classifier.

6.4 Receiver Operating Characteristic

The ROC curve is a metric for binary classification, to show their performance across multiple classification thresholds. Precision and recall on their own, are metrics for the case of a fixed chosen classification threshold. The ROC metric does not necessary need such a fix threshold, rather it evaluates the performance of a classifier, by considering many different classification thresholds. A ROC curve visualizes the TP-rate (recall) and FP-rate (fall-out) for any given probability threshold. A diagonal indicates, that both recall and fall-out increase with the same rate. The ROC curves are computed for both classes independently. The further shown macro-average ROC curve gives the averaged value of recall vs the averaged value of fall-out over both classes and hence gives a metric how both classifications combined perform. It characterizes equal contribution of all classes to the metric. Similarly as already described previously in Section 6.3, does the micro-average provide the sum of recall and fall-out over both classes.

The SVM classifier in Figure 6.7 shows a ROC curve for class 1 with a short straight increase of recall with 0 fall-out until recall ~ 0.28, where it cuts and starts increasing linearly with a gradient lower than 1. The ROC curve for the class 0 shows a linear gradient until 100% recall is reached at ~ 0.7 fall-out. For the MLP classifier in Figure 6.8, the ROC curves for class 1 and class 0 are both similar to the ROC curves of the SVM classifier, except for class 1 the ROC curves increases to ~ 0.7 and for class 0 it reaches 100% recall at ~ 0.82 fall-out. The micro-average shows for both classifiers a rectangular curve, whereas the macro-average shows the mean between class 1 and 0 ROC curves of both classifiers. In contrast, the ROC curves for class 1 and 0 as well as micro- and macro-average of the RF classifier in Figure 6.9 show an almost instant increase of recall to 100% with low fall-out. In case of the ROC curve of class 1 the recall slightly increases with growing fall-out, whereas for the ROC curve of class 0 it has slightly more fall-out but hits the 100% recall without any further cut.

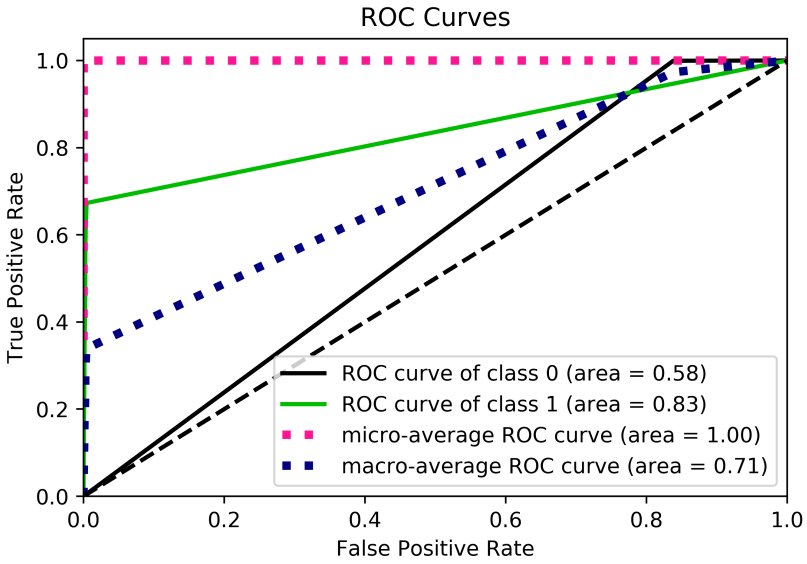


Figure 6.8: ROC curves for class 1, class 0, micro- and macro-average of the Multilayer Perceptron classifier.

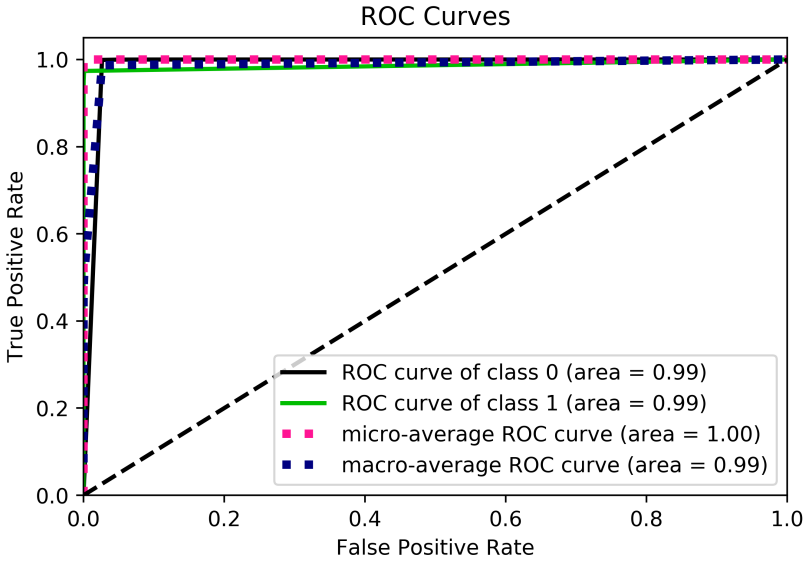


Figure 6.9: ROC curves for class 1, class 0, micro- and macro-average of the Random Forest classifier.

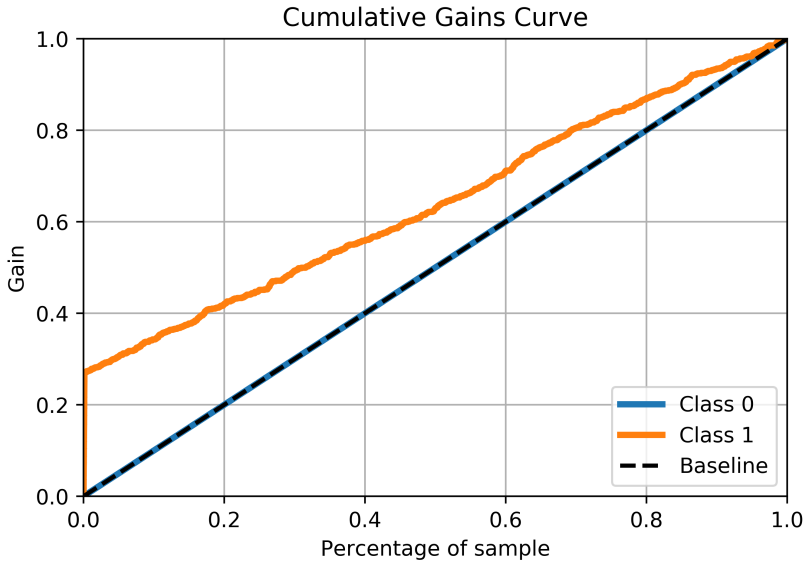


Figure 6.10: The cumulative gains curves for class 1 and class 0 of the Support Vector Machine classifier and the baseline for a random classifier.

6.5 Cumulative Gains

The cumulative gains (CG) is a metric, which shows how effective a classifier works for classifying a specific label, by comparing how much a classifier gains on considering more samples of the data set. A CG plot shows the gains of a classifier against its support (see Chapter 3). The CG therefore uses the training data as well as the test data combined. To calculate the gain, the predicted data is sorted by probability for that specific class, in descending order. The top of the data represents samples with high classification probability, the bottom samples with low classification probability for that class. To generate the cumulative gains curve, a step-wise portion from the top of the data (support) is taken and the according percentage of samples of the total number of samples of this class is generated. This percentage represents the gain for a specific percentage of samples (support value). A diagonal curve again suggests a random classifier, since it assumes an even distribution of both classes over the probabilities. An ideal classifier would instantly grow steep, until 100% gains is reached. That is, because every sample of class 1 is classified with high probability, and there are no false classifications. The sorted data set would include first all positive classified samples, followed by all negative classified.

In case of all three classifiers, the labeling of class 0 shows a random behavior, as it matches the diagonal baseline. For the classification of class 1, the CG curves for SVM (see Figure 6.10), MLP (see Figure 6.11) and RF (see Figure 6.12) show almost the same behavior. The CG curves for three classifiers increase almost straight to a certain gain, at which they make a cut and start increasing almost linearly to 100% gain at 1.0 support. The SVM shows the cut with lowest value at ~ 0.3, the MLP as second with cut at ~ 0.7 and the RF with the cut at the highest value ~ 0.99.

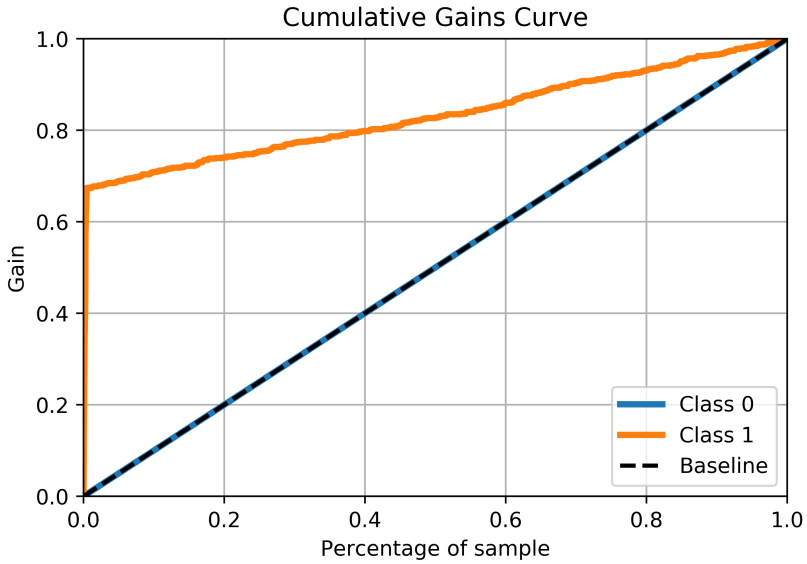


Figure 6.11: The cumulative gains curves for class 1 and class 0 of the Multilayer Perceptron classifier and the baseline for a random classifier.

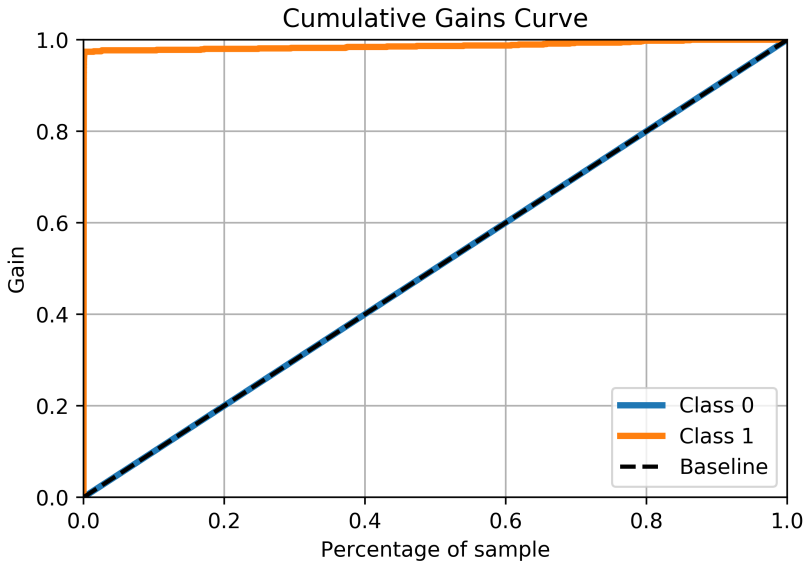


Figure 6.12: The cumulative gains curves for class 1 and class 0 of the Random Forest classifier and the baseline for a random classifier.

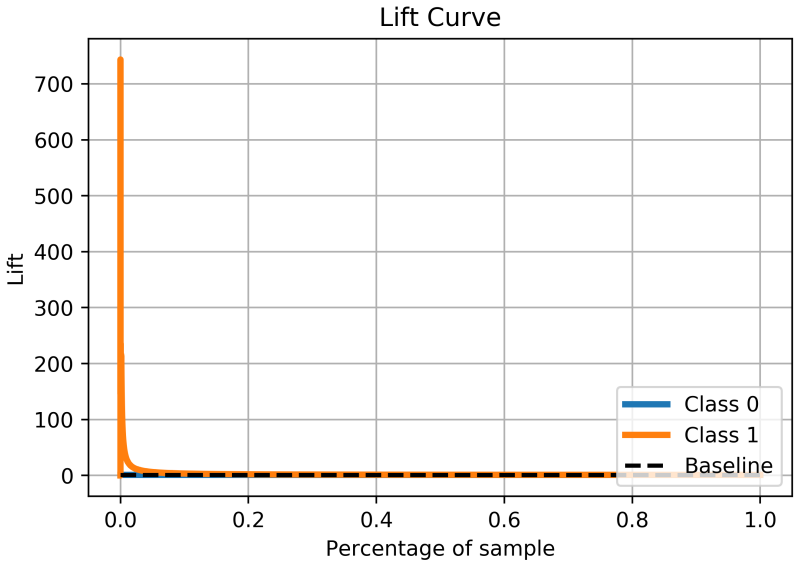


Figure 6.13: Lift curves for class 1 and class 0 of the Support Vector Machine classifier and a baseline at lift value 1 indicating a random classifier.

6.6 Lift

Lift, similar to the cumulative gains, does also measures how beneficial the usage of a certain predictive machine learning model is against a random classifier. For visualization, the lift score is plotted against the support. To calculate the lift score, it is derived from the cumulative gains. Again, the samples are sorted by their probability values for the corresponding class in descending order and the cumulative gains is calculated. The lift now takes the gains value of the corresponding cumulative gains curve of a classifier and divides the value by the corresponding support. In fact, the Lift curve is the derivative of the cumulative gains. Hence, the random classifier is represented as a constant line at lift score 1. If the Lift curve falls below the dotted line (Baseline/Random classifier) at $y = 1$, the model performs worse than a random classifier.

Applying the lift to the used machine learning models, the results show only a different visualization than the cumulative gains curves. According to the CG curves of all three classifiers for class 0, the lift curves (see Figures 6.13 to 6.15) are set to the constant $y = 1$, representing the derivative of the CG curves. The lift curves of class 1 for all three classifiers, show a straight increase to a high lift value according to the high gradient of the classifiers for the corresponding CG curves. In case of the SVM classifier in Figure 6.13, the lift increases to ~ 750 after which it drops back to a value close to 1. Whereas for the MLP classifier in Figure 6.14, the model shows a slightly lower lift value of ~ 600 at its peak. The RF classifier for classification of labels of 1 on the other hand (see in Figure 6.15), achieves more than four times the lift value than the SVM or MLP classifier lift curves. However, for all three models, the lift does shrink rapidly within a support value below 10%.

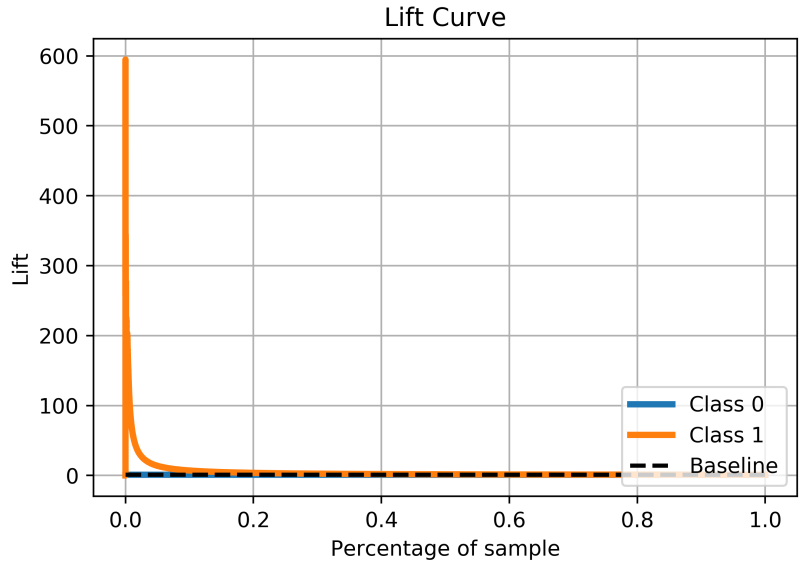


Figure 6.14: Lift curves for class 1 and class 0 of the Multilayer Perceptron classifier and a baseline at lift value 1 indicating a random classifier.

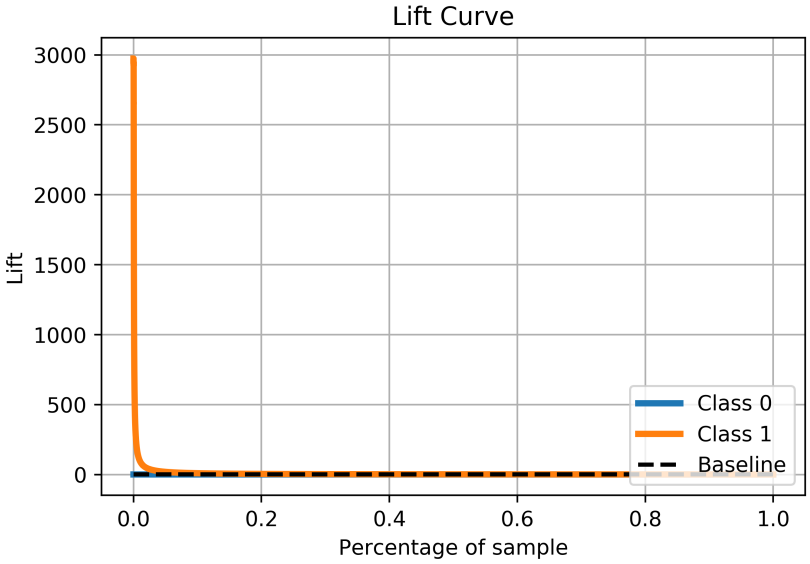


Figure 6.15: Lift curves for class 1 and class 0 of the Random Forest classifier and a baseline at lift value 1 indicating a random classifier.

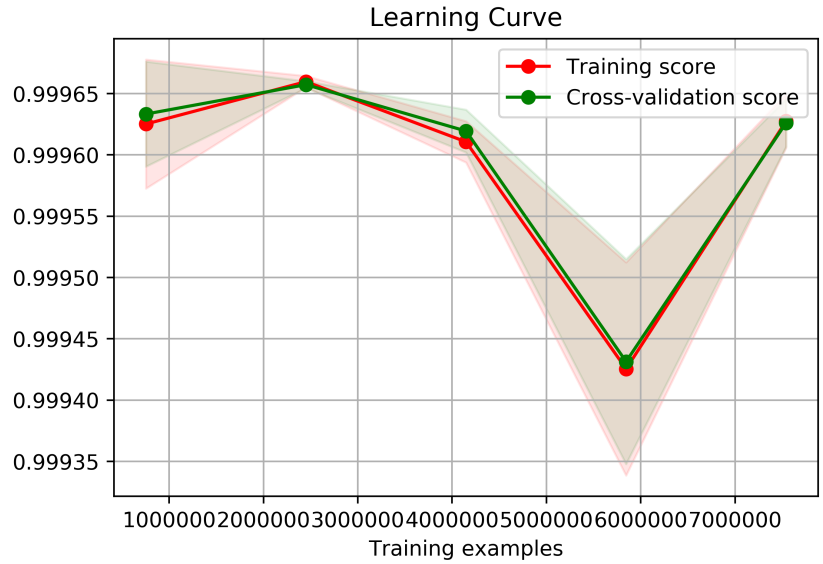


Figure 6.16: The learning curves of the Support Vector Machine classifier for a training data set as well as for a test data set.

6.7 Learning Curve

A learning curve visualizes the benefit of using more training samples, to train a specific machine learning model against a certain score that measures the corresponding models performance. The learning curves in the following plots (see Figures 6.16 to 6.18), are based on the mean accuracy of the training data set, for each model respectively. The plots include two different curves, one for training and one for testing. The training and test data is recomputed for the sake of this visualization and for each curve, a three-fold cross validation is used to compute the mean accuracy and its standard deviation. Five different set sizes of the data are generated to compute the metrics and are linear interpolated within the graphics. The learning curve indicates how the accuracy of both training and test data behaves in course of applying more samples.

Considering the SVM classifier in Figure 6.16, the learning curves for the training cross validation as well as for the testing cross validation, show a short increase of mean accuracy with decreasing standard deviation, between the first to data set sizes. For the next data set sizes the mean acurracy drops with increase in standard deviation. For the last data set size the mean accuracy increases again with decreasing standard deviation again. The learning curves for both training and test data of the MLP classifier in Figure 6.17 show a large standard deviation which decreases with the size of data sets and a settling mean accuracy at the same value. The RF classifiers learning curves in Figure 6.18, show an optimal mean accuracy for the training data cross validation, and a rising mean accuracy for the test data cross validation with increase of the data set sizes. Further almost no standard deviation is present for both curves.

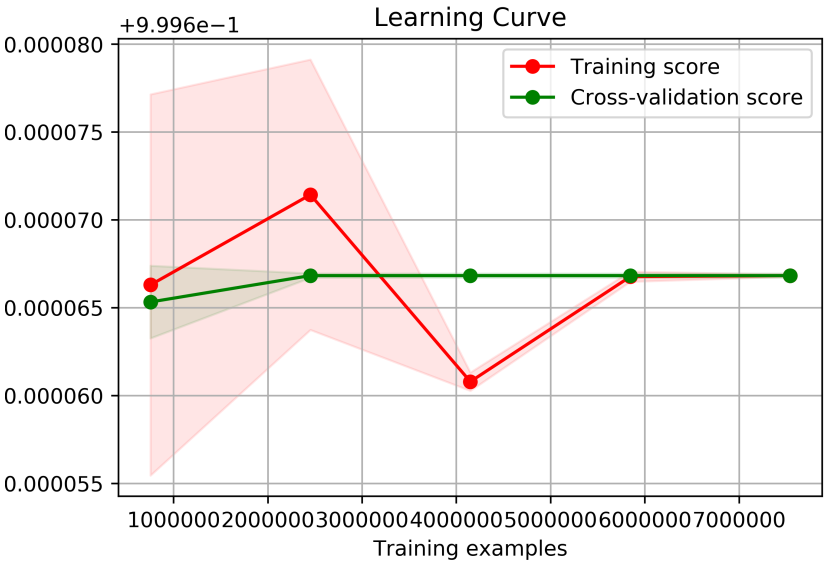


Figure 6.17: The learning curves of the Multilayer Perceptron for a training data set as well as for a test data set.

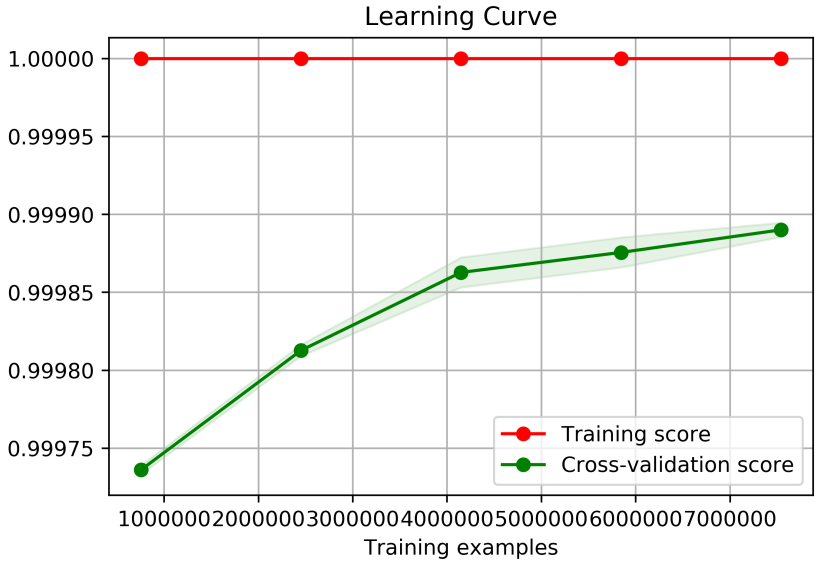


Figure 6.18: The learning curves of the Random Forest classifier for a training data set as well as a the test data set.

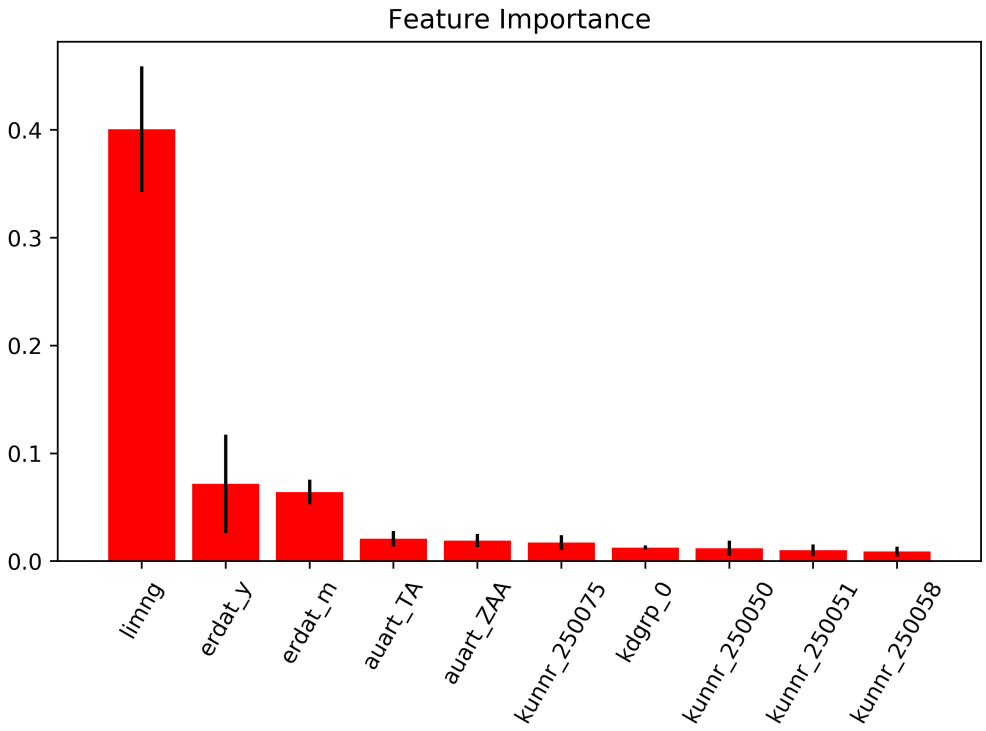


Figure 6.19: Feature importance values for features used for the split operation in decision trees within the Random Forest classifier.

6.8 Feature Importance

Feature importance is subject to the RF classifier only, since the used models from Scikit-Learn do not support a `feature_importance` score for the MLP and SVM classifiers. It is reflecting the influence of certain feature onto the output of a trained machine learning model. In other words, it depicts the features with the highest importance, contribute the most to predict a target value. The feature importance of all parameters sums up to 1. For the RF classifier, the feature importance is calculated based on the averaged decrease of *Gini Impurity* over all decision trees in the forest. The problem with this computation is, that due to measure of impurity decrease, continuous features with large values are favored compared to categorical features, leading to a biased result.

Figure 6.19 shows the importance values of features for the use case classification problem, based on the RF classifier. The feature leftmost representing the most important feature, and the right most, the least important. However, this plot does not include all $4,484$ features due to space constraints, but only the top ten. The feature `limng` shows an extraordinary importance with a score of $\sim 40\%$, followed by the two time dependent features `erdat_y` and `erdat_m` with an importance score around $\sim 5-7\%$. A figure with 20 features and their corresponding importance values, can be found in the appendix in Figure A.1. The features itself are also described in the appendix, in Table B.1.

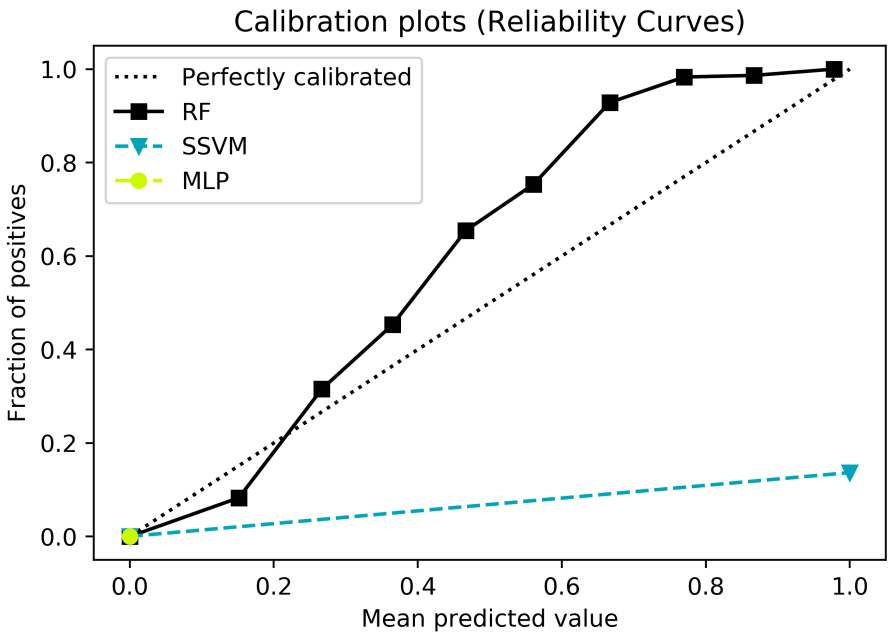


Figure 6.20: Reliability curves for SVM, MLP and RF classifiers with a diagonal indicating a perfectly calibrated probability predictor.

6.9 Reliability Curves

A reliability curve or calibration plot of a specific machine learning model, gives a measure on the confidence of a probability prediction about a certain sample the model made. It visualizes the predicted probability frequency against the fraction of observed positives. The y-axis is a normalization of counts of positive labels relatively to the number of total samples, while the x-axis plots the mean predicted values. A perfect calibrated classifier would lay on the diagonal in Figure 6.20, indicating the exact same probability of mean predicted values against its probabilities in the data set. Values above that diagonal are considered to indicate too high probabilities for the label prediction, and values below too low probabilities. For the negative class, the plot would be mirrored at the asymptote $\texttt{mean predicted value} = 0.5$, and since is not necessary to compute.

The calibration curves shown in 6.20, reference to the three trained machine learning models, tested with a new test set of data, ranging from year 2012 to 2013, instead of the data set of years 2010 to 2011. The SVM classifier does predict 0-labels good, and predicts 1-labels with only a very small probability, resulting in a low fraction of positives with a high predicted probability. The single yellow point for the MLP classifier indicates, that it can only predict 0-labels, with 100% certainty. A more vivid curve is computed for the RF classifier. It is close to the optimal calibrated diagonal, indicating well-calibrated probability predictions.

7 Implementation Details

This chapter gives insights into the used environments and details about the developed models, framework and integration into the TOSCA based SitOPT project, extending the work of Képes et al. in [KBL19]. Further, detailed coding parts are presented, to document the performed tasks as good as possible. Starting with the overall development setup used for the implementation tasks in Section 7.1, and continuing with the implementation of preprocessing the use case data and introduce to the implementation problems and difficulties coming with it in Section 7.2. Next in Section 7.3 implementation details of the machine learning model training and in Section 7.4 evaluation of their corresponding hyperparameters is shown. Afterwards, in Section 7.5 the implementation of the *Situation Confidence Predictor*, a script which is able to send requests to the models and send it to the SitOPT environment, is presented. At last in Section 7.6, the SitOPT integration is shown, which allows the situation-aware workflow management system the usage of the confidence scores.

7.1 Setup

For the most part of the implementation, a Jupyter notebook on a Google Cloud Platform (GCP) instance was used. The GCP offers resources and virtual machines for a wide variety of tasks. It provides a free trial for 12 Months, or resource usage in value of 300$, whereby one can choose from different hard drive storage sizes, CPUs, RAM and grafic cards. Using resources is charged by seconds and the resources can be adapted and configured on demand with resulting changes in costs. The location of the resources can be chosen by continent and several countries. GCP offers an AI Platform, which helps to setup an environment for development of machine learning tasks. For this thesis, a setup was used with an 2 TB hard drive, 8 virtual CPUs and 52 GB of memory, resulting in hourly costs of 0.099€. The AI Platform creates a VM instance with the desired resource parameters on a Linux debian basis, with a preinstalled Jupyter Lab, which can be accessed via a web interface. Implementations were made with Python 3.5.3 in this IDE. The most important python libraries, used for development were on the following verions: `pandas` (0.25.2), `sklearn` (0.21.3), `numpy` (1.17.3), `matplotlib` (3.0.3), `scikitplot` (0.3.7), `scipy` (1.1.0), `pickle` (0.25.2). The TOSCA integration was done with Java SE 1.8 in an Eclipse IDE of version 2019-09 R (4.13.0). The framework, which loads the models and XML requests to send confidence scores for *Situations* to the TOSCA ecosystem, was developed on a laptop with 16 GB of ram and a i7-7600 CPU and with Python 3.6 in PyCharm 2019.2.3 (Professional Edition).

7.2 Use Case - Data Preprocessing

The use case data was briefly introduced in Chapter 2, without too many details on how it was processed to usable for the machine learning models. As CSV files exported from SAP, the python data analysis toolkit *pandas* is an easy way to import the three tables into a pandas `DataFrame`. The `DataFrame`, containing the individual data sets, are filtered by a parameter `vkorg` to be equal to 1000, which stands for a special topic of products. The next step of preprocessing modified the data, by splitting the parameter `erdat`, indicating the date of order creation at the manufacturer, into year, month and day, while dropping latter. Thereby the orders within a month, of the same product and material combination needed to be merged, resulting in the $21,686$ remaining positive samples. Merging was done by summing up the values of parameter `limng` within a month for equal orders, as it depicts the quantity of the ordered product. An equal order is given, if all parameters except `limng` are the same. This is necessary due to memory problems, which are explained below. Another filtering step considered the range of years the data covers. Primal, the data set covered years 2009 to 2019, but the years 2009 and 2019 are incomplete and are hence dropped. Afterwards, the order settlements data consists of a total number of $21,686$ rows, with 16 different parameters describing a single order. Each parameter column is formatted as a categorical string value, except for `limng` which is a continuous value and is formatted as an unsigned 32-bit integer, `erdat_y` as 16-bit unsigned integer and `erdat_m` as 8-bit unsigned integer, resulting in a data set of size $21,686 \times 13 \times 64bit + 21,686 \times 32bit(\texttt{limng}) + 21,686 \times 8bit(\texttt{erdat\_m}) + 21,686 \times 16bit(\texttt{erdat\_y}) = 2.41$ MB. These in the remaining data set included parameters and a detailed explanation of them can be found in Table B.1 in the appendix.

7.2.1 Negative Sampling

The next step in preprocessing included sampling negatives for the data set. This was necessary, as explained in Chapter 5, due to the usage of classification algorithms for the machine learning tasks. Therefore, for each combination of existing materials of the sales topic of interest, filtered earlier, each customer and at each existing date (year and month only), a new negative sample is generated. Another crucial point is that between these parameters, there exist some dependencies, which have to be considered when creating the negative samples. For example for a value of the parameter `kdgrp`, representing a group of specific customers, not every customer number (`kunnr`) can be matched with that `kdgrp`. Hence for each sample, the parameter `kdgrp` is randomly chosen from those customers, for which there is already an existing combination of `kunnr` and `kdgrp`. The same problem and approach holds for the parameter `auart`. While parameters `comnr`, `mtart`, `matkl`, `prdha`, `mstae`, `farbe`, `hoehe`, `durch` and `lschn` depend on `matnr`, the parameter `limng` is calculated as a random value, between the minimum and maximum value discovered from this parameter over the whole data set. The generation of the negative samples was a massively time consuming task and had further to be done in parts, due to memory issues again.

With this method, there are samples of both types (positives and negatives) for each unique date, customer and material combination. The company which provided the data set, has a total number 248 registered customers, with 2,375 different materials to produce, with data ranging over 12 month $\times 8$ years $= 96$ months (years 2010 to 2018). This results in a total number of $56,544,000 - 21,686 = 56,522,314$ negative samples, with a total size of 7.46 GB. After negative sampling, both positive and

negative data were assigned a new target row, indicating if the sample is a positive or a negative, and are then merged together into a single `DataFrame` and sorted by year and month. This preprocessing procedure enables the use of classification algorithms on the data set.

7.2.2 Encoding

After generating and merging all necessary data entries, an important task to do if features are categorical, as they are for 13 of the 16 features in this case, is to encode these features. The python package Scikit-Learn [PVG+11] for machine learning tasks, offers a simple class (`OneHotEncoder`) for that functionality. It takes the categorical features an performs a one-hot-encoding to it, resulting in $4,481$ columns plus the three not encoded columns, to make it $4,484$ columns. The application of the encoding would result in a data set of the size of $56,522,314 \times 4,482 \times 8bit + 56,522,314 \times 16bit + 56,522,314 \times 32bit = 253.67$ GB. This immense size leveraged many problems during the development of the models, and is the reason why only orders by month are considered and not orders by days. Further, this immense size results in the need to split the data into junks of half years in advance of encoding, each with a size of 15.85 GB. However, the presented *OneHotEncoder* provides a configuration where its results are stored within a so called sparse matrix, provided by the `Scipy` library [VGO+19]. This special form of a matrix does only store locations of 1s within a matrix and hence saves a lot of memory. Using these type of matrices does improve some performance issues, but does not work for encoding the total data set, as it might seem at first sight. The sparse matrices get unzipped, when they are used with a Scikit-Learn machine learning model, hence enlarging its size. Therefore, the data still need to be split into the introduced junks, considering the range of two years only.

With the `OneHotEncoder`, only the 13 categorical features (see Table B.1) are encoded. The rest of the parameters is attached to the data again, after encoding is done. Further the target (output) row is stored in a separate variable, since the models provided by Scikit-Learn do need the input and output values separately. Both, features and targets are stored in `.npz` file format.

The `OneHotEncoder` needs to be saved, since it is used later to convert samples into readable form for the trained models, which are trained with data in one-hot-encoded form. It is serialized and saved using the `Pickle` library.

Given the data one-hot-encoded within sparse matrices and junks of two years' time span, it can be used to train and test the predictive models which were selected for the purpose of this thesis.

7.3 Model Training

Each of the three selected models is trained separately and individually, but with the same set of data samples preprocessed like in the previous section described. Due to time constraints cause by the time consuming preprocessing of the data and memory constraints due to the large amount of data, models were trained with a single junk of data only, namely the junk for year 2010 and 2011. A mandatory approach in machine learning is to split data into training and test data. Scikit-Learn again provides an easy class `train_test_split`, to get over this task. The data features (X) and targets (y), are passed into this function, which generates four new data structures: `X_train`, `X_test`, `y_train` and `y_test`. Models are then trained based on the `*_train` structures, and later evaluated and tested

with the `*_test` structures. Further, the training of the models precedes a hyperparameter tuning by a *grid search* and *cross validation*, which are described in the fundamentals (see Chapter 3). With these two methodologies for model training, the optimal set of parameters to train the models can be found, which however takes a huge amount of time. Each model contains different hyperparameters to be trained, some need more some less to be trained. However not all parameters could be optimized, again due to time constraints. Some of the hyperparameters were shown in Chapter 5. Documentation for all adjustable parameters of the machine learning model implementations, which are not optimized and hence not explained, can be found on the Scikit-Learn website [PVG+11].

7.3.1 Support Vector Machine

Listing C.1 shows the generation and fitting of a *grid seach* and *cross validation*, and a `SGDClassifier` instance from the Scikit-Learn library as the Support Vector Machine model. The grid search had to evaluate 700 different combinations with a three-fold *cross validation*, resulting in the optimal parameters which were already presented in Table 5.1. Those optimal parameters are serialized and saved. After conducting the *cross validation* and a grid search, the model has to be trained again with the optimal parameters and then serialized and stored into a `joblib` object, illustrated in Listing C.2. Parameters not mentioned in Listing C.1, but displayed in Listing C.2, take default values, since not included in the *grid search*.

7.3.2 Multilayer Perceptron

For a Scikit-Learn `MLPClassifier` instance, the corresponding *grid search* was done with parameters depicted in Listing C.3, resulting in a total number of 69 evaluations to compute. This time only a two-fold *cross validation* was conducted, since the training of an MLP instance took much longer than for an SCM. After the discovery of the optimal parameters, they are saved in the same way as the previous model. Table 5.2 shows the resulting optimal parameters of the cross validation. With these parameters, the model has to be trained and saved again, with the code shown in Listing C.4. Again, the parameters not mentioned in the parameter grid in Listing C.3, but displayed in Listing C.4, are assigned with default values.

7.3.3 Random Forest

Cross validation and *grid search* parameters for the Scikit-Learn `RFClassifier` instance, are shown in Listing C.5 on page 93. For the RF classifier, a three fold *cross validation* was implemented, since its training time were not too large, like they were for the MLP classifier. Combining the amount of parameters, 216 different grids are created and evaluated, resulting in optimal parameters shown in Table 5.3. The optimal parameters are saved afterwards, in the same way as for the previous models. With these parameters the model is also trained again and stored, illustrated by the code in Listing C.6. Just like for the other two models, parameters not mentioned in the grid in Listing C.5 but stated in Listing C.6, take the default values from the Scikit-Learn model instance.

7.4 Model Evaluation

After the models are trained, they are evaluated with different metrics presented in Chapter 6. The basic implementations of the corresponding visualizations are taken from the Scikit-Learn [PVG+11] and Scikit-Plot [Nak17] libraries. therefore the computed `X_test` and `y_test` data structures are used. First, the according predictions are generated as it is shown in Listing C.7. After that, the accuracy for each model can be calculated using the Scikit-Learn `accuracy_score` metric. The different plots for the evaluations in Chapter 6 for the different models, were done with the Scikit-Plot plots and Scikit-Learn evaluation metrics, illustrated in Listing C.8. As last part for the evaluation, the *calibration curve* is generated. Although the Scikit-Plot library provides also a `calibration_curve` class, which is based on the Scikit-Lear `calibration_curve`, there were some changes needed to be made to the source code, leading to the code in Listing C.9. The resulting matplotlib plots can be saved by adding the code line `matplotlib.pyplot.savegig('path.png')` after each line for computing the individual metrics.

7.5 Situation Confidence Predictor

After the models are trained, evaluated and stored, they are loaded into a python script with the instruction `joblib.load('Path')` and can there be used to predict the desired confidence scores for new received orders, by calling the model with the instruction `model.predict_proba('sample_encoded')`. Listing C.10 shows the code to load the models, and their usage. An order or sample need to be in the correct shape and have the correct data types for each parameter. Table B.1 shows the mandatory data types and the order in which they need to be. This new sample or order is then encoded with the `OneHotEncoder` used to encode the training and test data, shown in Listing C.11. To test the validity of the script, a function is defined that randomly draws a sample from the data set of all positives of years 2012 to 2019. However, the user can decide what model to use for prediction, since all three models take the same parametrization for a new request.

For the use case, the confidence score and the date according to the order request are sent back to the manufacturer, which is then able to use the confidence for the requested order. Anyway, this thesis added the functionality for situation confidence prediction to a situation-aware workflow management system, SitOPT [WSBL15]. To update situations within that system, there are some more task to be worked off.

To attach a generated confidence score to a specific situation within that situation-aware system, the situation has to be given by an XML file, and has to be loaded by the python script. For all XML tasks, the `xml.etree` package is used. A XML file can then be loaded by parsing it with the function `xml.etree.ElementTree.parse('xml_file_path.xml')`. After retrieving the XML root with `getroot()`, the tree can be extended with the confidence value for that situation. First a new tree element with the instruction `Element('ElementName')` has to be created for the parameter, and afterwards the parameter can be assigned to the tree element with the instruction `treeELement.text('parameter')`. As the date is also relevant, it can be copied from the original sample and a tree element for the date can be generated. To actually attach the new elements to the XML tree, they have to be inserted with the command `root.insert('position', 'treeElemeent')`.

The general described implementation is shown in Listing C.12. After extending the situation, it is sent back to the situation-aware management system via a HTTP POST request. To create this request, pythons `requests` library does the job. Listing C.13 illustrates this procedure.

7.6 SitOPT Integration

The situation-aware workflow management system SitOPT, where the functionality of situation confidence prediction is to be integrated, was shortly introduced in Chapter 3. However, the project is based on the TOSCA ecosystem, which need to be understood and which will be briefly introduced in the subsequent subsections.

7.6.1 TOSCA

TOSCA [BBLS12] is a standardization that is made to facilitate the design and management of portability of cloud applications within their life cycle. It is designed to close the gap between application developers (Cloud Service Developer), customer authorities (Cloud Service Provider) and end-users (Cloud Service Consumer). The developer is an actor, which develops cloud services and uses TOSCA to model how to instantiate the developed services. A customer or service provider handles requests for services and uses TOSCA to provide developed services. Eventually, the usage of a cloud service is attributed to the service consumer. This actor does not directly benefit from TOSCA, only indirectly via the service provider. In TOSCA, cloud services life cycles are defined by five different phases. In the first one, a service is defined with its topology and managements aspects, relationship types, plans as workflows and nodes as its components, which are all used later for its instantiation. Hence, this phase is called *definition phase*. The service developer conducts this phase. After that, in the *offering phase*, the provider offers the defined service to consumers. The third phase is mainly for the consumer, where he can instantiate services for his demands, thus this phase is called *subscription & instantiation phase*. Instantiating of a service implies acquisition of the corresponding resources for the service. In this phase, a cloud services template has its first appearance. According to the plans and relationships defined with such a template, the required resources are installed and configured. The fourth and last phase is the *production phase*, where a service keeps running, until the customer manually shuts it down, leading to a procedure where all acquired resources are freed.

Figure 7.1 shows an environment, how it is presented by the OASIS in [OAS13]. In the TOSCA specification [OAS13], an application is specified by service templates and packed into a so-called CSAR (Cloud Service Archive) files. Service templates are defined by topology templates and corresponding plans for them. CSAR files are recommended to be created with *TOSCA Modelling Tools*, but alternative modelling tools can be used too. They are passed to a *TOSCA-Container*, which is responsible to deploy, manage and decommission the corresponding cloud application. The primer TOSCA documentation [OAS13] describes two different processing styles of importing CSAR files with application plans and implementation artifacts.

In the first way, called “declarative processing”, it is specified *what* structural cloud application elements are needed. For the first step, after packing a CSAR file, it is imported into a TOSCA-Container. The *CSAR Processor* interacts with a *Model Interpreter*. Further, *Definitions* and

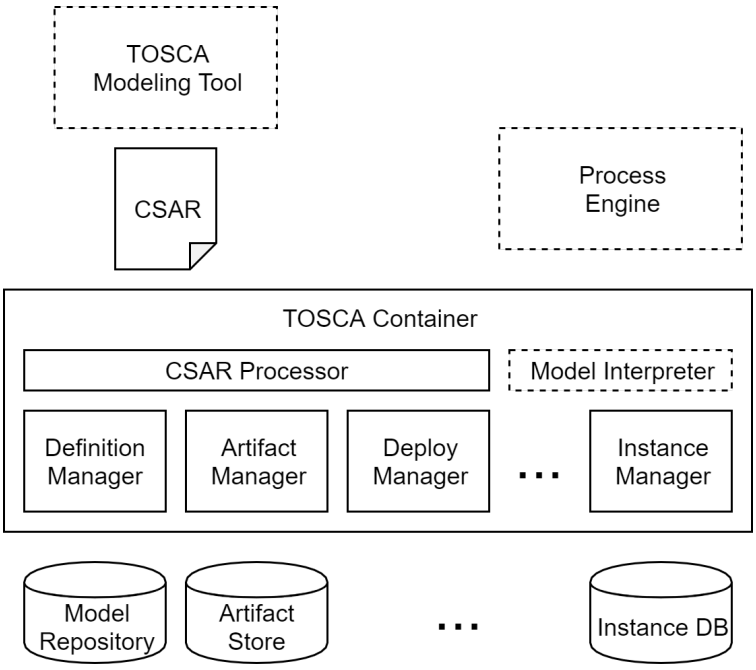


Figure 7.1: Exemplaric TOSCA environment architecture as presented in [OAS13].

Implementation/Deployment Artifacts are extracted from the CSAR file and then passed and stored to their corresponding *Managers* and *Repositories*. Continuing with the *Deploy Manager*, which deploys the *Implementation Artifacts*, and provides availability for all executables for node types and all relationship types from the topology available in the environment.

The second processing style, called "imperative processing", does process a CSAR file differently, and specifies *how* a cloud application is structured/managed, based on a so-called plan. Before packing the CSAR file, the modeling tool interacts directly with a *Model Interpreter* to turn a declarative model into an imperative one. Hence, there is no need for the *TOSCA-Container* to deal with declarative processing of definitions and artifacts.

As described in Sections 3.3 and 3.4, TOSCA is the basis for the situation-aware workflow management system SitOPT, which is build on top of TOSCA. TOSCA allows triggering application management, which is enhanced by an automation tool, OpenTOSCA, which in turn serves as a helper that allows automated situation-aware configurations for workflows.

7.6.2 Integration

The *Situation Confidence Prediction* is eventually integrated into the SitOPT project. Therefore, the *SituationTrigger* templates need to be updated, to accept confidence scores to trigger situation occurrences. However, the integration took place only in an abstract form, such that there is no functionality yet implemented, after the system receives the parameters for proactive triggering. Actually four different classes within the SitOPT system were changed. First, the (SituationTrigger.java) class is extended with the two variables, which will be sent by the *Situation Confidence Predictor*, `eventProbability` and `eventTime` plus their corresponding `getter` & `setter` methods. This

modification is depicted in Listing C.14. Afterwards, the `SituationController` has to be updated, where the `createSituationTrigger` method, responsible for handling the `POST` request for `\triggers`, is changed. The code for this update is shown in Listing C.15. The third class to modify, is the `SituationTriggerDTO` class. It is changed by also adding the two variables `eventProbability` and `eventTime` with their corresponding `getter` & `setter` methods. The corresponding code for this modification is shown in Listing C.16. At last, the `InstanceService` class is changed. Within this class, the `createNewSituationTrigger` method is updated, to accept `eventProbability` and `eventTime` as input parameters. Afterwards, the handling logic is inserted, as it is shown in Listing C.17.

The situation-aware management system is hence able to receive confidence scores and date for situations, which trigger workflow and business process adaptions. The system does not yet has any logic implemented to actually work with those parameters.

The TOSCA ecosystem based SitOPT project with the integrated confidence score receiving functionality, can be cloned from its github-repository at [IAA].

8 Discussion

This chapter will discuss, explain and compare the results examined and found in the previous chapter. It will also answer the question of the applicability of the used machine learning algorithms for specifically order predictions and the usage of the presented concept within situation-aware workflow management systems. To interpret the results, first it has to be clarified what this thesis was looking for, when considering the prediction algorithms. The aim was to predict positive situations (positive orders in the use case). Hence, to build a machine learning model that is able to perform well in the case for predicting class 1 labels with a probability as high as possible.

The different evaluation methods presented in Chapter 6 imply a very clear overview on the performance differences between the three used machine learning models (SVM, MLP and RF). However, for the results to be correctly understood, a close look has to be taken to one major property of the data set. To make the use case work, the data had to be extended by negative samples. This procedure was necessary, since the former data set contained only order settlements, which were already processed and hence there was no doubt these orders were "orders". A binary classifying algorithm does need samples for both classes to be trained correctly, else it would never know when to classify the missing class. To overcome this problem, negative samples were generated. They were mostly generated by considering dependencies between parameters, but the parameter *limng*, describing the amount of ordered products, was randomly generated with bounds as 0 and the maximum value of this parameter in the positive data set. This preprocessing led to a extremely imbalanced data set, with only 21,686 positive orders and 56,522,314 negatives. With this imbalance in mind, the results of Chapter 6 can be discussed and interpreted.

The calculated accuracy scores in Section 6.1 for each model showed values close to 1. However, when taking into account that the overall positives of the data were also only a fraction of the total number of orders, namely 0.03836%, it is clear that the accuracy for SVM (99.8562%), MLP (99.9663%) and RF (99.9912%), are not as accurate as they might appear at first sight.

This observation is further supported by the computed confusion matrices for the classifiers in Figures 6.1 to 6.3, for a probability threshold of 0.5. In all three cases, the confusion matrix shows an accuracy of 100% for true negative values (upper left) and 0% false negative vales (upper right), which represents the majority of orders that needs to be classified. Hence, the negative class does not have to be considered, since it is perfectly classified in every model. For the true positive values (lower right), the models hold different results. The MLP classifier is not able to detect positive values and hence predicts every sample as negative, which makes it useless in predicting orders. Although the SVM classifier does predict positive values, it does it only for 27% of the true positive values and falsely classifies positives as negatives in 73% of the orders. The RF classifier on the other hand shows decent results with a 76% correct true positive prediction. Overall, the latter algorithm seems to be the only one with some usable learned pattern recognition. The Precision-Recall curves (see Figures 6.4 to 6.6) further reveal, that for the classification of label 1, the SVM and MLP classifiers perform very bad in predicting labels for samples of class 1, as their

precision shrinks rapidly for low recall. The RF classifier shows a decent behavior in this metric for the classification of class 1. The ROC curves (see Figures 6.7 to 6.9) as well as the Cumulative Gains (see Figures 6.10 to 6.12) curves for the SVM show a non optimal performance, since both curves are close to the diagonal, indicating an only slightly better classifier for class 1, than a random classifier. Similarly but slightly better, performs the MLP classifier for those metrics. The RF on the other hand, shows almost perfect results for these metrics, supporting that it is a usable predictor for orders. The lift curves (Figures 6.13 to 6.15) for all three classifiers show a very similar shape, with a high lift at the beginning and a rapid loss of lift with the first few percentages of samples of the data set. This occurs, due to the strong imbalance of the data set, since the few samples classified with label 1 are on top of the sorted data set of probabilities for predicted labels. In the learning curves (see Figures 6.16 to 6.18) for the RF, a clear trend of increasing accuracy can be observed, indicating that training the model longer with more samples, may lead to better results. The SVM on the other hand does not show an improvement over training with more samples. For the last and largest data set size, the accuracy is still lower than for the second data set size, with a higher standard deviation. The MLP does not show a performance improvement for the mean accuracy, but its standard deviation seems to settle with increased data set sizes to train. The micro-average curves in the Precision-Recall and ROC plots (see Figures 6.4 to 6.9), fortify the imbalance of the data set, and ask for optimization of the trained models.

The feature importance of the random forest (see Figure 6.19) does show that the parameter `limng` has the most impact onto the results. However, a problem with the used feature importance computation, provided by Scikit-Plot [Nak17], is that it is based on the Scikit-Learn [PVG+11] feature importance, which has the tendency to favor continuous parameters to have a higher impact. Dropping the parameter `limng`, due to its random generation, would likely drastically change the outcome of the RF classifier. Whereas the impact of date values `erdat_y` and `erdat_m` support the detection of seasonality for orders.

The findings, that the SVM and the MLP are unusable and that only the RF is decent is summarized in the calibration curves (see Figure 6.20). Predicting a positive order with a certain percentage with the RF classifier, does not show too large discrepancies of the fractions of positives for that percentage. Only around prediction probability 0.5 to 0.8 the percentage of positives of the tested samples is larger than the probability percentage. Hence, using the designed RF classifier for confidence predictions within the given use case, seems applicable. For classifiers SVM and MLP, there seems some problems in parametrization, as their performance is bad and hence the models are not usable. Maybe they can be enhanced with some further hyperparameter tuning, which in turn is massively time consuming.

The results are not able to show how a random forest performs in different situation prediction problems. It cannot be generalized, since a machine learning model training is based on a fixed set of data and parameters. The data and the parameters may vary for different situation problems. But for this use case, the situation or order prediction is seemingly applicable, since 76% accuracy when predicting true positives and 0% false positives, suppose decent results with room for usage.

One huge difficulty this thesis faced, were the processing power limitations for preprocessing data. As already mentioned this issue can be faced by reducing the parameter space or changing the overall data set size. Nevertheless, the data set will be very large and a lot of memory is needed to train classification based machine learning models for situation prediction.

9 Conclusion & Outlook

This thesis investigated in the application of prediction algorithms, to improve and support situation-aware workflow management systems. Such common systems ([KBL19]) support reactive workflow adaptions and re-configurations, but do not support machine learning integration to predict situations in advance to decrease delays between situation recognition and workflow adaptions. Therefore, this thesis developed a framework, which abstracts the use of machine learning algorithms in the sense, that users are able to use this framework without expertise within machine learning. A *Situation Service* provides a database of situations and generates new situations. These situations are received by a situation-aware workflow management system, which can adapt its workflows accordingly. As an improvement, a *Situation Confidence Predictor* generates confidence scores for certain situations and hence allows to proactively trigger workflow adaptions and re-configurations, which can lead to a decrease in delay between situation occurrence and workflow adaption.

Further, this thesis validated this approach, by means of a use case scenario for order predictions representing situations. The aim was to improve the situation-aware production workflow, which needs to adapt to certain new orders. With the prediction of orders the delay between receiving an order and producing the requested product, can be minimized. To achieve these predictions, three different machine learning algorithms were trained, evaluated and compared by performance. The use case data was contributed by a manufacturing company.

The main occurring problems were connected to a large amount of data to be processed, namely the whole data set consisted of 56,544,000 data entries from two years, leading to a total size of over 250 GB of data. This data could not be processed at once and hence models were only trained and evaluated on a partition of this data. Technical issues, like insufficient memory and down times of the compute engine did also impede the implementation part.

The results show, that the SVM and MLP are either wrong parametrized, or are just not the optimal solution to predict orders. The RF shows decent results but still there is enough room for improvement. Another way to make the SVM and MLP work, could be the redesign of the input. A principle component analysis (PCA) to identify most important dimensions, could lead to a huge dimension reduction, hence shrink the size of the data set and allows the training of algorithms with more than just two years of data. Playing with the amount of parameters and hence, the size of the data set, brings up the next point. More training data could help to improve the MLP or SVM, and definitely would improve the RF, since its learning curve (see Figure 6.18) shows an improvement over more samples. Important to note is, that more negative samples may help to improve the accuracy for positives, however positive samples are much more important and have a higher impact on the distinction between positives and negatives. This is extremely difficult to achieve, since the negative samples show a linear growth over time. Using daily data instead of monthly may also lead to better results, but could also aggravate the results due to the increase of

negative samples. Instead of computing more samples, another way could be to use less by merging the data into quarters of a year or even complete years, which however would result in less freedom to include the date for an order.

Fundamentally, it can be said that powerful machine learning algorithms like the RF classifier shows an applicable result with room to improvement, but allow an application for situation prediction within situation-aware systems, even for extremely imbalanced data sets.

With more time, further models could be assessed. For example, according to [PHG13] the usage of a k-nearest-neighbor algorithm for classification holds very good results for situation prediction. Another example would include the different settings for SVMs by [SS09; YZWM19] or Grey Theory, as suggested by [LM15]. More time would also allow the models to be tested with a wider variety of parameters in their corresponding grid-searches. Another interesting approach would also be the usage of reinforcement learning approaches like it is used in [KHSY15], or even the usage of Bayesian Games for situation awareness [BA04] to simulate a competition between a situation generator and a situation-aware system.

Bibliography

[BA04] J. Brynielsson, S. Arnborg. "Bayesian Games for Threat Prediction and Situation Analysis". In: *Proceedings of the Seventh International Conference on Information Fusion, FUSION 2004* 2 (July 2004) (cit. on pp. 37, 80).

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. "OpenTOSCA – A Runtime for TOSCA-Based Cloud Applications". In: *Service-Oriented Computing*. Ed. by S. Basu, C. Pautasso, L. Zhang, X. Fu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 692–695. ISBN: 978-3-642-45005-1. DOI: 10.1007/978-3-642-45005-1_62 (cit. on pp. 33, 34).

[BBLS12] T. Binz, G. Breiter, F. Leyman, T. Spatzier. "Portable Cloud Services Using TOSCA". In: *IEEE Internet Computing* 16.3 (May 2012), pp. 80–85. DOI: 10.1109/MIC.2012.43 (cit. on pp. 33, 74).

[BBR17] M. Bohanec, M. Borstnar, M. Robnik-Sikonja. "Explaining Machine Learning Models in Sales Predictions". In: *Expert Systems with Applications* 71 (Apr. 2017), pp. 416–428. DOI: 10.1016/j.eswa.2016.11.010 (cit. on p. 16).

[BDR07] M. Baldauf, S. Dustdar, F. Rosenberg. "A Survey on Context-Aware Systems". In: *Int. J. Ad Hoc Ubiquitous Comput.* 2.4 (June 2007), pp. 263–277. ISSN: 1743-8225. DOI: 10.1504/IJAHUC.2007.014070 (cit. on pp. 29, 30, 32).

[BGV92] B. E. Boser, I. M. Guyon, V. N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401 (cit. on p. 45).

[Bot10] L. Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proc. of COMPSTAT* (Jan. 2010). DOI: 10.1007/978-3-7908-2604-3_16 (cit. on pp. 44, 46).

[Bre01] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324 (cit. on p. 50).

[CCS12] A. Cutler, D. R. Cutler, J. R. Stevens. "Random Forests". In: *Ensemble Machine Learning: Methods and Applications*. Ed. by C. Zhang, Y. Ma. Boston, MA: Springer US, 2012, pp. 157–175. ISBN: 978-1-4419-9326-7. DOI: 10.1007/978-1-4419-9326-7_5 (cit. on pp. 44, 50).

[DMJS19] E. Deelman, A. Mandal, M. Jiang, R. Sakellariou. "The Role of Machine Learning in Scientific Workflows". In: *The International Journal of High Performance Computing Applications* (May 2019), pp. 1128–1139. DOI: 10.1177/1094342019852127 (cit. on pp. 16, 38).

[Gru15] J. Grus. *Data Science from Scratch: First Principles with Python*. 1st. O'Reilly Media, Inc., 2015. ISBN: 9781491901427 (cit. on p. 23).

[GXF+14] M. Gao, W. Xu, H. Fu, M. Wang, X. Liang. “A Novel Forecasting Method for Large-Scale Sales Prediction Using Extreme Learning Machine”. In: *2014 Seventh International Joint Conference on Computational Sciences and Optimization*. July 2014, pp. 602–606. DOI: 10.1109/CSO.2014.116 (cit. on p. 16).

[Her00] J. Herbst. “A Machine Learning Approach to Workflow Management”. In: *Machine Learning: ECML 2000*. Ed. by R. López de Mántaras, E. Plaza. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 183–194. ISBN: 978-3-540-45164-8 (cit. on pp. 16, 38).

[HWS+16] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. Gómez Sáez, F. Leymann. “Situation Recognition and Handling Based on Executing Situation Templates and Situation-Aware Workflows”. In: *Computing* 99 (Oct. 2016). DOI: 10.1007/s00607-016-0522-9 (cit. on p. 32).

[IAA] IAAS at University of Stuttgart. *OpenTOSCA*. https://github.com/OpenTOSCA/container.git. Accessed: 2019-12-07 (cit. on p. 76).

[JWHT14] G. James, D. Witten, T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN: 9781461471370 (cit. on p. 23).

[KBL19] K. Képes, U. Breitenbücher, F. Leymann. “Situation-Aware Management of Cyber-Physical Systems”. In: *Proceedings of the 9th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,* INSTICC. SciTePress, 2019, pp. 551–560. ISBN: 978-989-758-365-0. DOI: 10.5220/0007799505510560 (cit. on pp. 15, 33, 36, 41, 43, 69, 79).

[KHSY15] M. H. Kabir, M. R. Hoque, H. Seo, S.-H. Yang. “Machine Learning Based Adaptive Context-Aware System for Smart Home Environment”. In: *International Journal of Smart Home* 9 (Nov. 2015), pp. 55–62. DOI: 10.14257/ijsh.2015.9.11.07 (cit. on pp. 15, 36, 80).

[KM03] P. Korpipää, J. Mäntyjärvi. “An Ontology for Mobile Device Sensor-Based Context Awareness”. In: *Modeling and Using Context*. Ed. by P. Blackburn, C. Ghidini, R. M. Turner, F. Giunchiglia. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 451–458. ISBN: 978-3-540-44958-4. DOI: 10.1007/3-540-44958-2_37 (cit. on p. 31).

[LM15] Y.-B. Leau, S. Manickam. “Network Security Situation Prediction: A Review and Discussion”. In: *Intelligence in the Era of Big Data*. Ed. by R. Intan, C.-H. Chi, H. N. Palit, L. W. Santoso. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 424–435. ISBN: 978-3-662-46742-8 (cit. on pp. 15, 16, 39, 80).

[Nak17] R. Nakano. *Scikit-Plot*. Version v0.2.1. 2017. DOI: 10.5281/zenodo.293191 (cit. on pp. 53, 73, 78, 96).

[OAS13] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2013 (cit. on pp. 32, 33, 74, 75).

[PHG13] T. Pitakrat, A. van Hoorn, L. Grunske. “A Comparison of Machine Learning Algorithms for Proactive Hard Disk Drive Failure Detection”. In: *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems*. ISARCS ’13. Vancouver, British Columbia, Canada: ACM, 2013, pp. 1–10. ISBN: 978-1-4503-2123-5. DOI: 10.1145/2465470.2465473 (cit. on pp. 15, 35, 44, 80).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. “Scikit-Learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 23, 29, 44, 47, 50, 53, 71–73, 78, 96).

[RIGE17] H. Ramchoun, M. A. J. Idrissi, Y. Ghanou, M. Ettaouil. “Multilayer Perceptron: Architecture Optimization and Training with Mixed Activation Functions”. In: *Proceedings of the 2Nd International Conference on Big Data, Cloud and Applications*. BDCA’17. Tetouan, Morocco: ACM, 2017, 71:1–71:6. ISBN: 978-1-4503-4852-2. DOI: 10.1145/3090354.3090427 (cit. on pp. 44, 47).

[SBLW16] C. T. Sungur, U. Breitenbücher, F. Leymann, M. Wieland. “Context-Sensitive Adaptive Production Processes”. In: *Procedia CIRP* 41 (2016). Research and Innovation in Manufacturing: Key Enabling Technologies for the Factories of the Future - Proceedings of the 48th CIRP Conference on Manufacturing Systems, pp. 147–152. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2015.12.076 (cit. on pp. 15, 37).

[SL04] T. Strang, C. Linnhoff-Popien. “A Context Modeling Survey”. In: *First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004, Nottingham, England, September 7, 2004*. Sept. 2004. URL: https://elib.dlr.de/7444/ (cit. on pp. 30, 31).

[SS09] N. Sapankevych, R. Sankar. “Time Series Prediction Using Support Vector Machines: A Survey”. In: *Computational Intelligence Magazine, IEEE* 4 (June 2009), pp. 24–38. DOI: 10.1109/MCI.2009.932254 (cit. on pp. 16, 39, 44, 80).

[Vap98] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. URL: https://www.bibsonomy.org/bibtex/21a5aaa75fa8be088b01a7381d2f661be/fluctuator (cit. on p. 44).

[VGO+19] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, S. 1. 0. Contributors. “SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python”. In: *arXiv e-prints*, arXiv:1907.10121 (July 2019), arXiv:1907.10121. arXiv: 1907.10121 [cs.MS] (cit. on p. 71).

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. “Towards Situation-Aware Adaptive Workflows: SitOPT — A General Purpose Situation-Aware Workflow Management System”. In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. Mar. 2015, pp. 32–37. DOI: 10.1109/PERCOMW.2015.7133989 (cit. on pp. 15, 16, 32, 73).

[YZWM19] W. Yang, J. Zhang, C. Wang, X. Mo. “Situation Prediction of Large-Scale Internet of Things Network Security”. In: *EURASIP Journal on Information Security* 2019.1 (Aug. 2019), p. 13. ISSN: 2510-523X. DOI: 10.1186/s13635-019-0097-z (cit. on pp. 15, 16, 39, 80).

All links were last followed on December 16, 2019.

A Figures

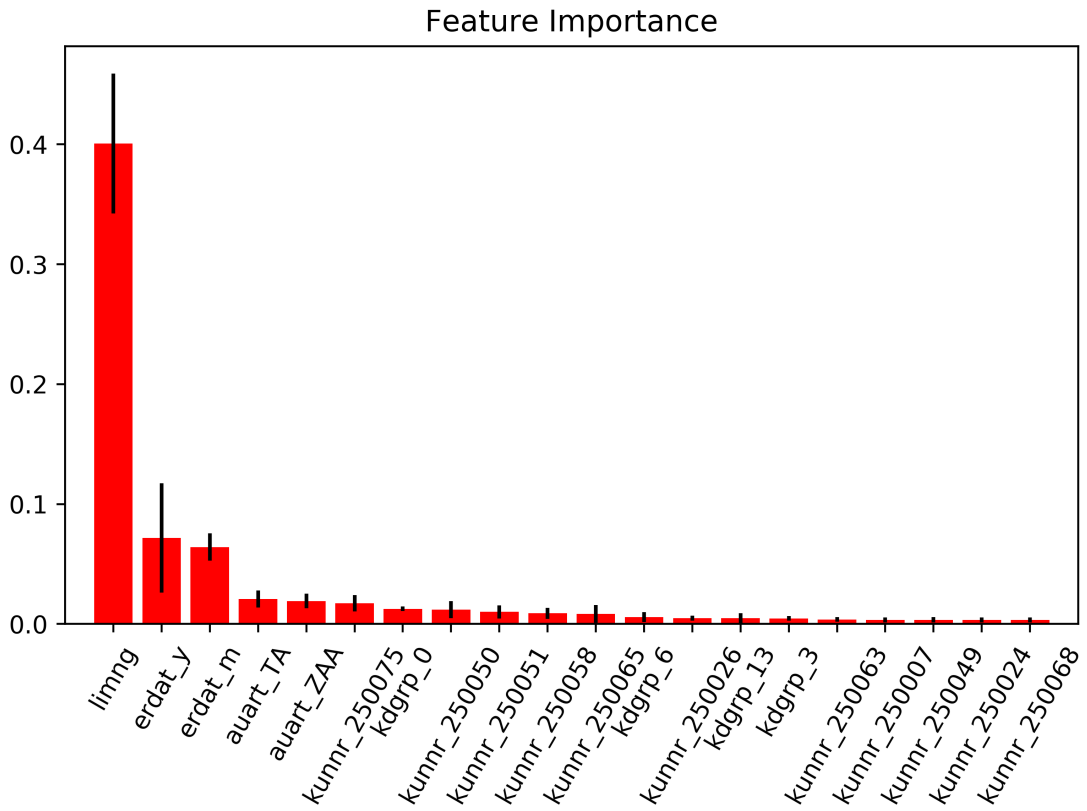


Figure A.1: Feature importance for top 20 features with importance values for the RF classifier.

B Tables

Parameter	Description	Data Type
kdgrp	Customer group. One customer can have multiple customer numbers, and different customers can be grouped into a customer group.	categorical
kunnr	Identifies a customer.	
matnr	Identifies a product.	
comnr	The product family.	categorical
mtart	The type of the products material.	categorical
matkl	Classifies products into groups.	categorical
prdha	Product hierarchy.	categorical
mstae	State of the product.	categorical
farbe	Color of the product.	categorical
hoehe	Height of the product.	numerical
durch	Diameter of the product.	numerical
lschn	Shape of the product.	categorical
auart	Type of settlement. For example a forward order or general order.	categorical
erdat_y	Year of erdat as independent parameter.	numerical
erdat_m	Month of erdat as independent parameter.	numerical
limng	The amount of parts shipped.	numerical

Table B.1: Parameters and their descriptions from the use case data set after filtering, cleaning and extension of the former data set.

C Listings

Listing C.1 SVM *grid search* and *cross validation*

```
    # Scikit-learn Stochastic Gradient Descent Classifier
    sgdc = SGDClassifier()

    # Parameter grid to iterate through
    param_grid = {
        'loss': ['modified_huber'],
        'penalty': ['l2', 'l1'],
        'alpha': [0.01, 0.001, 0.0001, 0.00001, 0.000001],
        'l1_ration': [0.1, 0.15, 0.2, 0.3, 0.5]
        'max_iter': [1, 10, 20, 50, 100],
        'learning_rate': ['optimal'],
        'early_stopping': [True, False]
    }

    # Cross validation and grid search combined
    grid_search = GridSearchCV(
        estimator=sgdc,
        param_grid=param_grid,
        cv=3,
        n_jobs=-1,
        verbose=2
    )

    # compute best parameters
    grid_search.fit(X_train, y_train)

```

Listing C.2 SVM model training with optimal parameters, and dump into file.

```
    # Scikit-learn Stochastic Gradient Descent Classifier
    sgdc = SGDClassifier(alpha=0.00001,
                         average=False,
                         class_weight=None,
                         early_stopping=True,
                         epsilon=0.1,
                         eta0=0.0,
                         fit_intercept=True,
                         l1_ratio=0.15,
                         learning_rate='optimal',
                         loss='modified_huber',
                         max_iter=1,
                         n_iter_no_change=5,
                         n_jobs=None,
                         penalty='l2',
                         power_t=0.5,
                         random_state=None,
                         shuffle=True,
                         tol=0.001,
                         validation_fraction=0.1,
                         verbose=1,
                         warm_start=False)

    # train or fit the model with training data
    sgdc.fit(X_train, y_train)

    # save the model for later usage
    joblib.dump(sgdc, 'sgdc.joblib')

```

Listing C.3 MLP *grid search* and *cross validation*

```
    # Scikit-learn Multilayer Perceptron Classifier
    mlpc = MLPClassifier()

    # Parameter grid to iterate through
    param_grid = {
        'hidden_layer_sizes': [(10,), (100,), (10, 10), (10, 10, 10)],
        'solver': ['lbfgs', 'adam', 'sgd'],
        'alpha': [0.1, 0.01, 0.001],
        'learning_rate_init': [0.1, 0.01, 0.001, 0.0001]
        'max_iter': [5, 10, 20, 50, 100, 200, 1000],
        'early_stopping': [True, False]
    }

    # Cross validation and grid search combined
    grid_search = GridSearchCV(
        estimator=mlpc,
        param_grid=param_grid,
        cv=2,
        n_jobs=-1,
        verbose=2
    )

    # compute best parameters
    grid_search.fit(X_train, y_train)

```

Listing C.4 MLP model training with optimal parameters, and dump into file.

```
    # Scikit-learn Multilayer Perceptron Classifier
    mlpc = MLPClassifier(activation='relu',
                         alpha=0.1,
                         batch_size='auto',
                         beta_1=0.9,
                         beta_2=0.999,
                         early_stopping=True,
                         epsilon=1e-08,
                         hidden_layer_sizes=(10,),
                         learning_rate='constant',
                         learning_rate_init=0.001,
                         max_iter=5,
                         momentum=0.9,
                         n_iter_no_change=10,
                         nesterovs_momentum=True,
                         power_t=0.5,
                         random_state=None,
                         shuffle=True,
                         solver='adam',
                         tol=0.0001,
                         validation_fraction=0.1,
                         verbose=False,
                         warm_start=False)

    # train or fit the model with training data
    mlpc.fit(X_train, y_train)

    # save the model for later usage
    joblib.dump(mlpc, 'mlpc.joblib')

```

Listing C.5 RF *grid search* and *cross validation*

```
    # Scikit-learn Random Forest Classifier
    rfc = RFClassifier()

    # Parameter grid to iterate through
    param_grid = {
        'bootstrap': [True, False],
        'max_depth': [100, None],
        'min_samples_leaf': [1, 2, 4],
        'min_samples_split': [2, 5],
        'n_estimators': [10, 15, 20]
    }

    # Cross validation and grid search combined
    grid_search = GridSearchCV(
        estimator=rfc,
        param_grid=param_grid,
        cv=3,
        n_jobs=-1,
        verbose=2
    )


    # compute best parameters
    grid_search.fit(X_train, y_train)

```

Listing C.6 RF model training with optimal parameters, and dump into file.

```
    # Scikit-learn Random Forest Classifier
    rfc = RFClassifier(activation='relu',
                       alpha=0.1,
                       batch_size='auto',
                       beta_1=0.9,
                       beta_2=0.999,
                       early_stopping=True,
                       epsilon=1e-08,
                       hidden_layer_sizes=(10,),
                       learning_rate='constant',
                       learning_rate_init=0.001,
                       max_iter=5,
                       momentum=0.9,
                       n_iter_no_change=10,
                       nesterovs_momentum=True,
                       power_t=0.5,
                       random_state=None,
                       shuffle=True,
                       solver='adam',
                       tol=0.0001,
                       validation_fraction=0.1,
                       verbose=False,
                       warm_start=False)

    # train or fit the model with training data
    rfc.fit(X_train, y_train)

    # save the model for later usage
    joblib.dump(rfc, 'rfc.joblib')

```

Listing C.7 Predictions for classifers SVM, MLP and RF.

```
    # SVM predictions
    sgdc_predictions = sgdc.predict(X_test)
    sgdc_prob_predictions = sgdc.predict_proba(X_test)

    # MLP predictions
    mlpc_predictions = mlpc.predict(X_test)
    mlpc_prob_predictions = mlpc.predict_proba(X_test)

    # RF predictions
    rfc_predictions = rfc.predict(X_test)
    rfc_prob_predictions = rfc.predict_proba(X_test)

```

Listing C.8 Evaluation for SVM, MLP and RF.

```
    models = [sgdc, mlpc, rfc]
    predictions = [sgdc_predictions, mlpc_predictions, rfc_predictions]
    prob_predictions = [sgdc_prob_predictions, mlpc_prob_predictions, rfc_prob_predictions]

    for i in predictions:

        # Accuracy Score
        print(accuracy_score(y_test, i))

        # Confusion Matrix
        skplt.metrics.plot_confusion_matrix(y_test, i, normalize=True)

    for j in prob_predictions

        # ROC curve
        skplt.metrics.plot_roc_curve(y_test, j)

        # Precision & Recvall cureve
        skplt.metrics.plot_precision_recall(y_test, j)

        # Cumulative gain
        skplt.metrics.plot_cumulative_gain(y_test, j)

        # Lift curve
        skplt.metrics.plot_lift_curve(y_test, j)

        # KS statistic
        skplt.metrics.plot_ks_statistic(y_test, j)

    for k in models:

        # Learning curve
        skplt.estimators.plot_learning_curve(k, X_train, y_train)

        if type(k).__name__ == 'RandomForestClassifier':

            # Feature importance (for RF only)
            skplt.estimators.plot_feature_importances(k, feature_names=feature_names)

```

Listing C.9 Calibration curve, based on the corresponding Scikit-learn class, with modified code of the Scikit-plot `calibration_curve` class. [PVG+11] [Nak17]

```
1
2       clf_names=['SVM', 'MLP', 'RF']
3       n_bins=10
4
5       if ax is None:
6           fig, ax = plt.subplots(1, 1, figsize=figsize)
7
8       ax.plot([0, 1], [0, 1], "k:", label="Perfectly calibrated")
9
10      for i, probas in enumerate(prob_predictions):
11          probas = np.asarray(probas)
12
13          probas = probas[:, 1]
14
15          fraction_of_positives, mean_predicted_value = \
16              calibration_curve(y_test, probas, n_bins=n_bins)
17
18          color = plt.cm.get_cmap(cmap)(float(i) / len(probas_list))
19
20          if i==0:
21              ax.plot(mean_predicted_value, fraction_of_positives, 'v--',
22                      label=clf_names[i], color=color)
23
24          if i==1:
25              ax.plot(mean_predicted_value, fraction_of_positives, 'o-.',
26                      label=clf_names[i], color=color)
27
28          if i==2:
29              ax.plot(mean_predicted_value, fraction_of_positives, 's-',
30                      label=clf_names[i], color=color)
31
32      ax.set_title(title, fontsize=title_fontsize)
33
34      ax.set_xlabel('Mean predicted value', fontsize=text_fontsize)
35      ax.set_ylabel('Fraction of positives', fontsize=text_fontsize)
36
37      ax.set_ylim([-0.05, 1.05])
38      ax.legend(loc='upper left')
39
```

Listing C.10 Code to load the trained machine learning models, serialized in joblib files.

```
    # Load the trained and stored machine learning models
    model_svm = joblib.load('SVM/model.joblib')
    model_rf = joblib.load('RF/model.joblib')
    model_mlp = joblib.load('MLP/model.joblib')

    # Use sample to predict new confidence
    prediction_svm = model_svm.predict_proba(sample)[0][1]
    prediction_mlp = model_mlp.predict_proba(sample)[0][1]
    prediction_rf = model_rf.predict_proba(sample)[0][1]

```

Listing C.11 Encoding an order with the one-hot-encoder used for training.

```
    # Split categorical from continouse values
    sample_to_encode = sample.iloc[:, 0:13].astype('str')
    sample_not_to_encode = csr_matrix(sample.iloc[:, 13:].astype('uint32'))

    # Encode categorical values
    sample_ohe = oneHotEncoder.transform(sample_to_encode)

    # Append non-categorical values to encoded values
    sample_ohe = hstack([sample_ohe, sample_not_to_encode]).astype('uint32')

```

Listing C.12 Load XML file and extend the tree with new parameters, to be send to the SitOPT system via HTTP Post request.

```
    # Load XML file
    tree = ET.parse('XML_Example/xml_request_sample.xml')

    # Get root element to work with xml tree
    root = tree.getroot()

    # Create new xml elements
    eventProbabilityElement = ET.Element("EventProbability")
    eventTimeElement = ET.Element("EventTime")

    eventProbabilityElement.text = str(eventProbability)
    eventTimeElement.text = str(eventTime)

    # Insert new elements to XML tree
    root.insert(-1, eventProbabilityElement)
    root.insert(-1, eventTimeElement)

```

Listing C.13 `HTTP POST` request to the TOSCA ecosystem running in the background.

```
    requests.post('http://localhost:1337/situationsapi/triggers/',
                  data=ET.tostring(root),
                  headers={'Content-Type': 'application/xml'})

```

Listing C.14 Modifications within the `SituationTrigger` class.

```
    ⋮

    @Column(nullable = true)
    private float eventProbability;

    @Column(nullable = true)
    private String eventTime;

    ⋮

    public float getEventProbability() {
        return this.eventProbability;
    }

    public void setEventProbability(final float eventProbability) {
        this.eventProbability = eventProbability;
    }

    public String getEventTime() {
        return this.eventTime;
    }

    public void setEventTime(final String eventTime) {
        this.eventTime = eventTime;
    }

    ⋮

```

Listing C.15 Modifications within the `SituationCotnroller` class.

```
    ⋮

    float eventProbability = -1.0f;
    if (situationTrigger.getEventProbability() != -1.0f) {
        eventProbability = situationTrigger.getEventProbability();
    }

    String eventTime = null;
    if (situationTrigger.getEventTime() != null) {
        eventTime = situationTrigger.getEventTime();
    }

    ⋮

    this.instanceService.createNewSituationTrigger(
        sits, situationTrigger.isOnActivation(),
        situationTrigger.isSingleInstance(),
        serviceInstance, nodeInstance,
        situationTrigger.getInterfaceName(),
        situationTrigger.getOperationName(),
        inputs, eventProbability, eventTime);

    ⋮

```

Listing C.16 Modifications within the `SituationTriggerDTO` class.

```
    ⋮

    @XmlElement(name = "EventProbability", required = false)
    private float eventProbability = -1.0f;

    @XmlElement(name = "EventTime", required = false)
    private String eventTime;

    ⋮

    public float getEventProbability() {
        return this.eventProbability;
    }

    public void setEventProbability(final float eventProbability) {
        this.eventProbability = eventProbability;
    }

    public String getEventTime() {
        return this.eventTime;
    }

    public void setEventTime(final String eventTime) {
        this.eventTime = eventTime;
    }

    ⋮

```

Listing C.17 Modifications within the `InstanceService` class.

```
    ⋮

    public SituationTrigger createNewSituationTrigger(
        final Collection<Situation> situations,
        final boolean triggerOnActivation,
        final boolean isSingleInstance,
        final ServiceTemplateInstance serviceInstance,
        final NodeTemplateInstance nodeInstance,
        final String interfaceName,
        final String operationName,
        final Set<SituationTriggerProperty> inputs,
        final float eventProbability,
        final String eventTime) {

    ⋮

        if (eventProbability != -1.0f) {
            newInstance.setEventProbability(eventProbability);
        }
        if (eventTime != null) {
            newInstance.setEventTime(eventTime);
        }

    ⋮

    }

    ⋮

```

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature