Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Computational Engineering and Technical Physics
Computer Vision and Pattern Recognition

Juho-Pekka Koponen

PREDICTING LEAD TIMES OF PURCHASE ORDERS USING GRADIENT BOOSTING MACHINE

Master's Thesis

Examiners: Professor Lasse Lensu
M.Sc. Tal Katzav

Supervisors: Professor Lasse Lensu
M.Sc. Tal Katzav

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Computational Engineering and Technical Physics
Computer Vision and Pattern Recognition

Juho-Pekka Koponen

Predicting lead times of purchase orders using gradient boosting machine

Master's Thesis

2020

59 pages, 19 figures, 9 tables, 1 appendix.

Examiners: Professor Lasse Lensu
M.Sc. Tal Katzav

Keywords: gradient boosting, ensemble learning, supply chain management, prediction, hyper-parameter optimisation, machine learning, data science

A company can make more accurate predictions of its internal processes and sales lead times when it has accurate predictions of the lead times of purchase orders. It results in more efficient processes as well as improved business performance and customer satisfaction. This thesis focuses on defining and implementing a method for predicting the lead times of purchase orders. Gradient boosting machine was used as the prediction model. The model uses purchase orders from the past to find patterns that can be used to estimate future lead times. Each data sample is related to a specific purchase order and consists of data about the supplier, item purchased, purchasing unit and purchase contract. Tree-based Parzen estimator was used for quantitative feature selection and hyper-parameter optimisation. Three objective functions for the Tree-based Parzen estimator were experimented with. Each of them produced, on average, more accurate predictions than the supplier estimates. The differences between the objective functions resulted in a trade-off between the model complexity and prediction accuracy.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Laskennallinen tekniikka ja teknillinen fysiikka
Konenäkö ja hahmontunnistus

Juho-Pekka Koponen

Ostotilausten toimitusaikojen ennustaminen gradien boosting -metodia käyttäen

Diplomityö

2020

59 sivua, 19 kuvaajaa, 9 taulukkoa, 1 liite.

Tarkastajat: Professori Lasse Lensu
M.Sc. Tal Katzav

Hakusanat: gradient boosting, ensemble-oppiminen, toimitusketju, ennustus, hyperparametrien optimointi, koneoppiminen, datatiede

Yritys voi tehdä tarkempia ennusteita sisäisistä prosesseistaan ja myynnin toimitusajoista, jos sillä on tarkat ennusteet ostotilausten toimitusajoista. Se johtaa tehokkaampiin prosesseihin sekä kohentaa asiakastyytyväisyyttä. Tämä diplomityö keskittyy määrittelemään ja toteuttamaan menetelmän, jolla voi ennustaa ostotilausten toimitusaikoja. Gradient boosting -menetelmää käytettiin ennustemallina. Malli käyttää vanhoja ostotilauksia löytääkseen rakenteita, joiden avulla voi ennustaa tulevia toimitusaikoja. Kukin otos liittyy tiettyyn ostotilaukseen ja koostuu toimittajaan, ostettuun tuotteeseen, ostoyksikköön sekä ostosopimukseen liittyvistä tiedoista. Tree structured Parzen estimator -menetelmää käytettiin kvantitatiiviseen piirrevalintaan ja hyperparametrien optimointiin. Kolmea kohdefunktiota testattiin kyseiselle estimaattorille. Kukin niistä tuotti keskimäärin tarkempia ennusteita kuin toimittajien arviot. Erot kohdefunktioiden välillä johtivat vaihtokauppatilanteeseen mallin monimutkaisuuden ja ennusteiden tarkkuuden välillä.

PREFACE

I would like to thank my supervisors Lasse Lensu and Tal Katzav. After supervising my bachelor's thesis, Lasse was my first choice for the master's thesis due to his expertise, helpfulness and punctuality. He showed the same qualities in the making of this thesis. During my time in Wärtsilä, Tal has always been eager to help and make things go as smoothly as possible. I would also like to thank my manager Jarkko Pukkila for making it possible to work with such an interesting problem.

Thank you to my colleagues in LUT for the entertainment, company and support in studies and free-time. Especially with Teemu Härkönen we had many interesting conversations including the thesis and life in general. I would like to thank him for having the patience to explain me many mathematical concepts and methods.

I have the utmost gratitude towards my Mom and Dad, who have always encouraged and supported me in my studies and life decisions. I would also like to thank my big brother who has been an excellent brother and friend. He has also been an example of what can be achieved if one puts his mind into it. I would like to thank my whole family and relatives for teaching me healthy life values, taking care of me, and for showing support through the ups and downs.

Lappeenranta, November 11, 2020

Juho-Pekka Koponen

CONTENTS

LIST OF ABBREVIATIONS **6**

1 INTRODUCTION **8**
1.1 Background 8
1.2 Objectives and delimitations 9
1.3 Structure of the thesis 10

2 PREDICTING LEAD TIMES OF PURCHASES **11**
2.1 Supply chain management 11
2.2 Machine learning 12
2.3 Methods used for predicting lead times of purchases 13
2.3.1 Binomial regression model 13
2.3.2 Hybrid model of stepwise regression and gamma distribution . . . 15
2.3.3 Random forest with dimension reduction 16

3 METHODS USED IN THIS STUDY **18**
3.1 Ensemble learning 18
3.2 Decision trees 18
3.3 Gradient boosting 20
3.4 LightGBM 23
3.5 Tree-structured Parzen estimator 24
3.6 Shapley additive explanations 27

4 EXPERIMENTS **29**
4.1 Data 29
4.2 Evaluation criteria 32
4.3 Description of experiments 32
4.4 Results 35
4.4.1 Hyper-parameter optimisation and feature relevance 35
4.4.2 Training and complexity of the models 44
4.4.3 Prediction 44

5 DISCUSSION **49**
5.1 Current study 49
5.2 Future work 51

6 CONCLUSION **52**

REFERENCES **53**

APPENDICES

Appendix 1: SHAP values

LIST OF ABBREVIATIONS

CART	Classification And Regression Trees
EFB	Exclusive Feature Bundling
EI	Expected Improvement
ERP	Enterprise Resource Planning
GBM	Gradient Boosting Machine
GMM	Gaussian Mixture Model
GOSS	Gradient-based One-side Sampling
LAD	Least Absolute Deviation
MAE	Mean Absolute Error
MSE	Mean Squared Error
PCA	Principal Component Analysis
PO	Purchase Order
QRF	Quantile Regression Forest
RF	Random Forest
SCM	Supply Chain Management
SHAP	Shapley Additive Explanations
SMBO	Sequential Model-based Global Optimisation
TPE	Tree-structured Parzen Estimator

1 INTRODUCTION

1.1 Background

Many large companies are currently interested in supply chain management (SCM). Its ultimate goal is to make every part of a supply chain run as efficiently as possible in order to create savings, make better products and create a competitive edge. [1]

A major part of the supply chain is the lead time of purchased items or materials, which are to be refined to produce the final product [1]. The lead time is the time between the making of a purchase order (PO) and the item reaching the purchaser. In some cases, it contains only the delivery of the product, and sometimes also the manufacturing of it. The lead times of these intermediate products are a part of the supply chain that is out of control of the purchasing company, as the items are bought from external vendors. The suppliers usually make some estimates of the lead times, but the estimates can vary widely. The products are often delivered later than promised, and this can have negative consequences in the supply chain. In the worst-case scenario, a late delivery can block the whole manufacturing process and cause the delivery of the final product to be late. This naturally causes dissatisfied customers and extra expenses. [2]

One approach for making the estimates of lead times more accurate is to enhance the communication between the suppliers and the buyer. It would be beneficial for the buyer if the suppliers would share their production schedules. However, in order to maintain competitive advantages, the suppliers are often unwilling to do this. Also, for a large company that orders items from thousands of suppliers, it is challenging to maintain excellent communication with every supplier. [3]

An interesting question is if the lead times of purchased items can be estimated more accurately with only the data available to the purchaser. As data is gathered from similar purchases over some time, maybe some patterns emerge in the lead times. Perhaps there are differences between the supplier' given estimates. One supplier could always make precise estimates while another makes regularly promises it cannot keep. Maybe there is so much demand for certain types of items that their lead times are highly unpredictable. There may be some times of the year, month or week that cause a particular item or supplier to have longer lead times. There is a vast amount of different hidden variables that can affect the lead time of an item, and it is almost an impossible task to make a model by hand that could properly estimate the delivery times. However, there is a high possibility

that these hidden variables affect the data in predictable ways, and that machine learning may prove useful in solving this problem.

In machine learning, the data is passed to a model, and it tries to learn the patterns in the data. One does not have to - and in most cases cannot - know precisely what kind of patterns the model found. The task at hand is an example of supervised learning; the input data includes information about the suppliers, items and dates, and the lead times are the target data. The task is supervised because the model can be trained with the input and target data from the past, and it will use new, unseen input data to predict the new values for the target data in the future. A few recent studies have been made in an attempt to predict lead times from the buyer's perspective [2–4]. The study by Liu et al. [3] is the most recent and produced the most promising results via usage of an ensemble learning model called random forest (RF).

Ensemble learning is a field of machine learning where the goal is to combine multiple weak models into one strong ensemble model. The individual models have low bias and high variance, corresponding to high precision and low accuracy, on average. The motivation for combining the models is to minimize the variance, which naturally happens when taking averages. This is analogous to many voting systems people use for making decisions including democratic elections and the jury of peers or panel of judges in a judicial system. [5]

RF [6] is an ensemble learning algorithm where multiple decision trees [7] are trained independently and combined. Gradient boosting machine (GBM) [8] is a method that can be used for producing an ensemble model by sequentially training weaker models, each correcting the errors of the previous ones. Decision trees are commonly used as the weak models in GBM because of their simplicity and ability to approximate arbitrary functions. LightGBM [9] is a state-of-the-art implementation of GBM which optimises speed, memory usage and accuracy. It is also able to deal with categorical features without the need for complicated pre-processing.

1.2 Objectives and delimitations

The main objective of this thesis is to find a suitable method for creating point predictions for lead times of purchases for large companies. The case study is for a global company which manufactures and services power solutions for the marine and energy markets. The parameters used in the methods are optimised for the dataset from this company, but the

methods are applicable for any company with large numbers of purchases from different suppliers.

To achieve the main objective, GBM is used as the predictive model and its hyper-parameters are optimised via tree-structured Parzen estimator (TPE) with three distinct objective functions, and the results of these are compared against each other. The workflow of implementing the practical part of the thesis is shown in Figure 1.

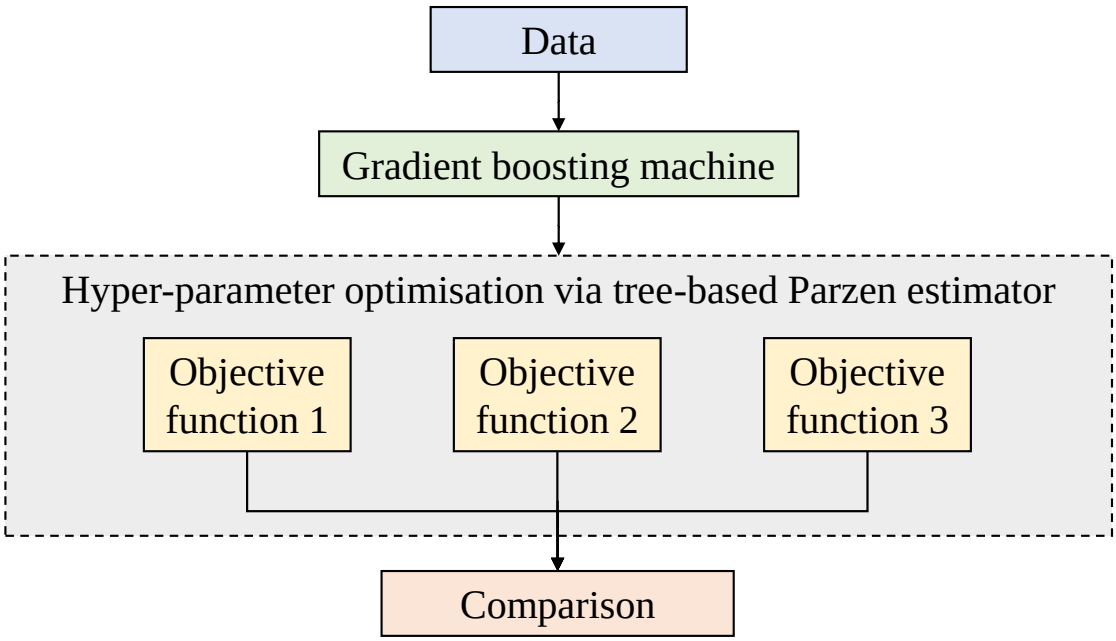


Figure 1. Workflow of implementing the practical part of the thesis.

1.3 Structure of the thesis

Chapter 2 contains a brief overview of SCM and machine learning, as well as existing solutions for the task of predicting lead times of purchases. Chapter 3 explains some theory behind the methods used in the thesis. The experiments, data and results are described in Chapter 4. The results are discussed in further detail in Chapter 5, and Chapter 6 contains the conclusions made from the results.

2 PREDICTING LEAD TIMES OF PURCHASES

2.1 Supply chain management

SCM is a set of approaches for integrating every part of a supply chain in order to make it as efficient as possible. This includes the suppliers, manufacturers, stores and warehouses. The goal is to manufacture and distribute the correct amount of products, at the right time, and to the correct places. From the perspective of a multinational company, purchasing and inbound transportation are significant parts of the supply chain, and the optimisation of these through negotiations and waste elimination can lead to savings worth millions of euros. [1]

Even if a company has the best product and service, it might not be enough in today's highly competitive global marketplace. It is crucial to be able to produce and deliver the products to the customer as fast as possible. A traditional way of achieving fast lead times is to keep sufficient inventories. However, high inventories mean significant amounts of capital invested, limiting liquidity. A good information network can provide a complete view of the inventories both at the moment and in the future. It can be the decisive factor between a market leader and follower. As the amount of data and computational power have increased, artificial intelligence methods have become a more significant part of translating the data into useful information. [10, 11]

In Figure 2, a basic structure of a supply chain is shown. This thesis focuses on the information flow between the producer and the supplier, from the producer's point of view. In order to have a complete view of the inventory in the future, a producer must have good communication with the suppliers. However, in large companies with lots of suppliers around the world, it is increasingly challenging to maintain proper communication with all of them [3]. The suppliers may knowingly or unknowingly hide some information, and they do not necessarily even have all of the information available. Therefore, enhancing the lead time predictions made by the suppliers can be beneficial for the producer.

In their recent review of research trends, Ni et al. [12] conclude that machine learning in SCM is still in the developmental stage. An earlier review by Bousqaoui et al. [13] states that machine learning is used in all stages of the supply chain, but limited mostly to planning and transportation. In both reviews, the studies in question deal mostly with demand estimation, supplier selection, inventory management, transportation routes and customer feedback. No studies referenced by them involved predicting lead times of purchases.

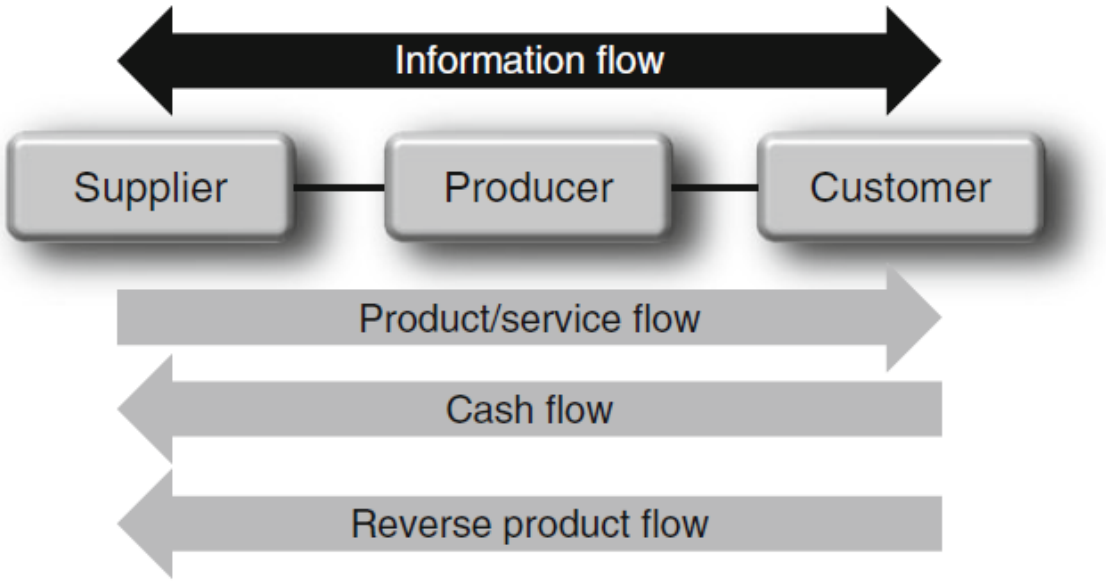
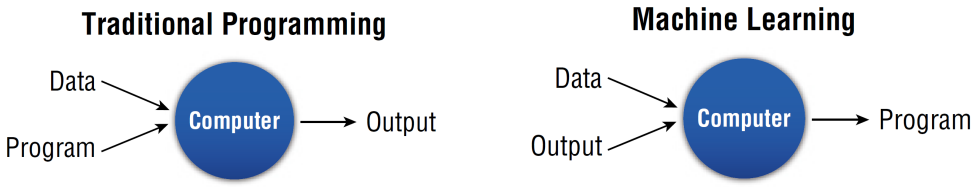


Figure 2. Basic supply chain structure [10].

2.2 Machine learning

In traditional programming, a program is written as a sequence of instructions that transforms the input into an output. To model real-world processes this way, one has to have a deep understanding of the problem at hand in order to make a model that approximates the real world. For example, if the goal is to predict the trajectory of a ball in the physical space, one would make a physics model based on Newtonian physics and give the ball's speed, position and forces affecting it as input. In some problems, the underlying model is difficult or even impossible to figure out. Separating spam emails from legitimate emails is an example where the underlying model is so complicated that it would be almost impossible to know how to transform the input into the output. Fortunately, since the computing power of modern computers and the amount of data have grown significantly during the past decades, a class of new methods of making algorithms has become viable. Machine learning is a field of study where the goal is for the machine to generate an algorithm that gives certain outputs from certain inputs. The machine learns the essential patterns from the data, hence the name ”machine learning”. Figure 3 visualizes the difference between traditional programming and machine learning. [14, 15]



(a) (b)

Figure 3. Inputs and outputs of algorithms [15]: (a) Traditional programming; (b) Machine learning.

2.3 Methods used for predicting lead times of purchases

Only a few studies proposing methods for predicting lead times of purchases were found, and the proposed methods vary. The common aspect is that all of the methods utilize supervised machine learning.

2.3.1 Binomial regression model

In their study, Walls et al. [4] make a broad analysis of the risk in inbound parts of engineering systems. As seen in Figure 4, a part of the analysis is the late delivery model based on historical delivery records. Negative binomial regression modelling is used to predict how late an order will be.

In the study, the authors do not tell the exact amount of data, but they do tell that it is from a span of one year of a manufacturing company. The data consists of entries of inbound deliveries with standard details such as supplier, item type, expected delivery date and the lead time. A separate model was made for each item type. Because they used a count model, they had to specify a time delay interval for the prediction.

At a daily level, there are artificial cyclical patterns in the data due to data recording practices, which caused them to settle on an interval of one week. They do not present any numbers of the results, but the histograms in Figure 5 show that at least the distribution of the predictions is close to the observations.

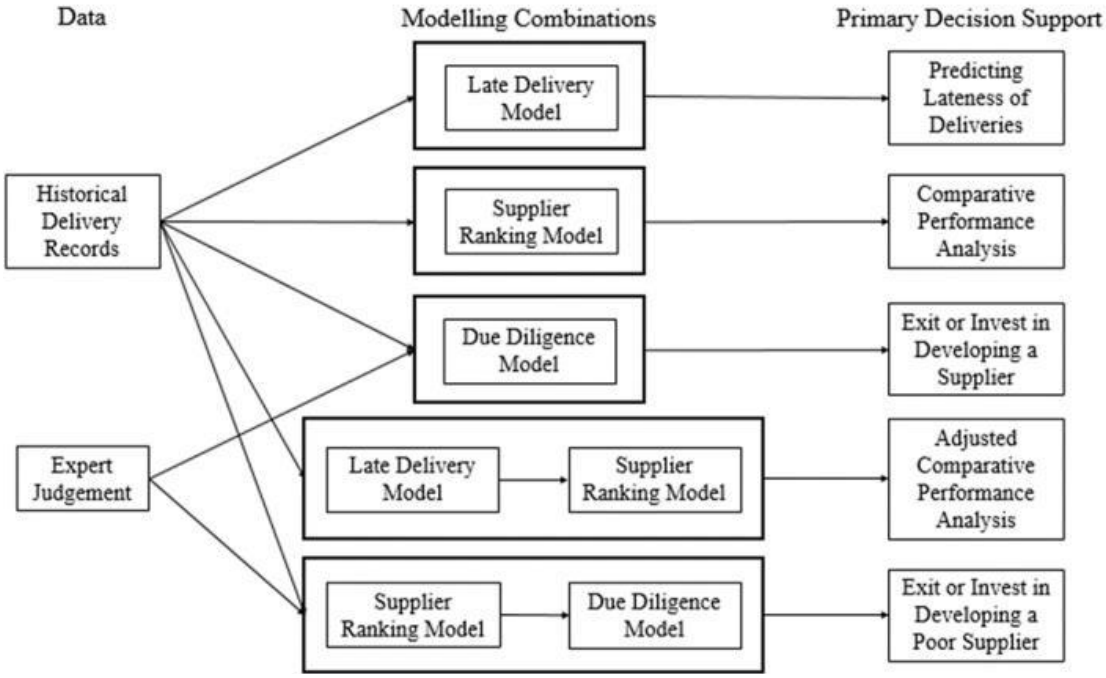


Figure 4. A framework for supply risk analysis by Walls et al. [4]

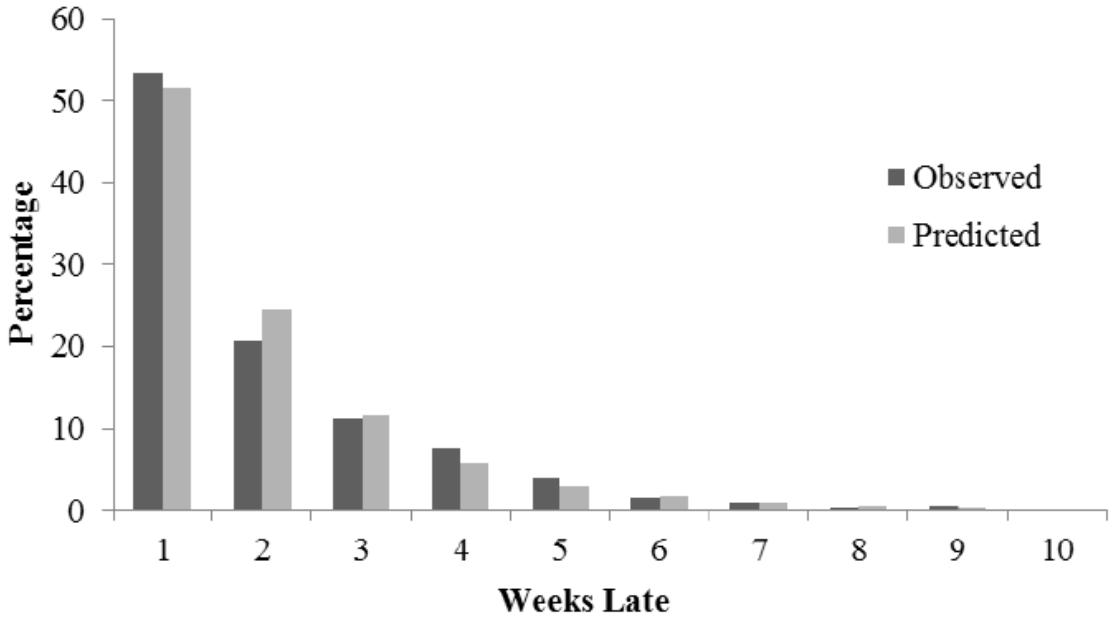


Figure 5. Comparison of observed late deliveries and predictions made by the late delivery model by Walls et al. [4]

2.3.2 Hybrid model of stepwise regression and gamma distribution

Banerjee et al. [2] studied the prediction of supplier delivery times of aircraft engine parts based on closed POs. The continuous features in the data consisted of the dates associated with the POs and surrogate measures of supplier capabilities. The categorical features consisted of part attributes such as material types and qualities.

The authors separated the data into groups based on suppliers and pre-processed the data in each group as follows:

1. Weight the samples of the training data so that newer samples are more important than older ones.
2. Impute negative and missing values with the most similar observation in the training set, using only continuous features.
3. Calculate Pearson correlation coefficients for every continuous feature for every supplier-part combination. If the coefficient is smaller than a user-defined threshold, discard it.
4. Separate the data into clusters according to delivery times.

After obtaining the clusters, two distinct models are trained for each cluster separately. First, they fit a linear regression model with linear quadratic and bilinear terms and eliminate the terms one by one based on the predictive power of the terms. Then, they use maximum likelihood and maximum a posteriori methods to fit a gamma distribution to the training data. Gamma distribution was chosen because it has been useful in modelling waiting times in econometrics. When making predictions, they select the model, which predicts a longer delivery time for each sample in order to obtain conservative predictions. There is a diagram of the whole prediction system in Figure 6.

They used two separate datasets with 589 130 and 534 764 samples for training and tested the model with 13 952 and 16 849 samples. The delivery times were in the range of 200-700 days. The results are promising as the median prediction errors for the test sets were 6.78 and 5.39 days with standard deviations of 14.43 and 7.02 days. Their model beat the linear regression and autoregressive integrated moving average models. Unfortunately, they did not disclose how the predictions compared to the estimates made by the suppliers. A downside of their approach is long training time. The training took in total 14 hours of CPU time on a 1000 core Unix HPC cluster.

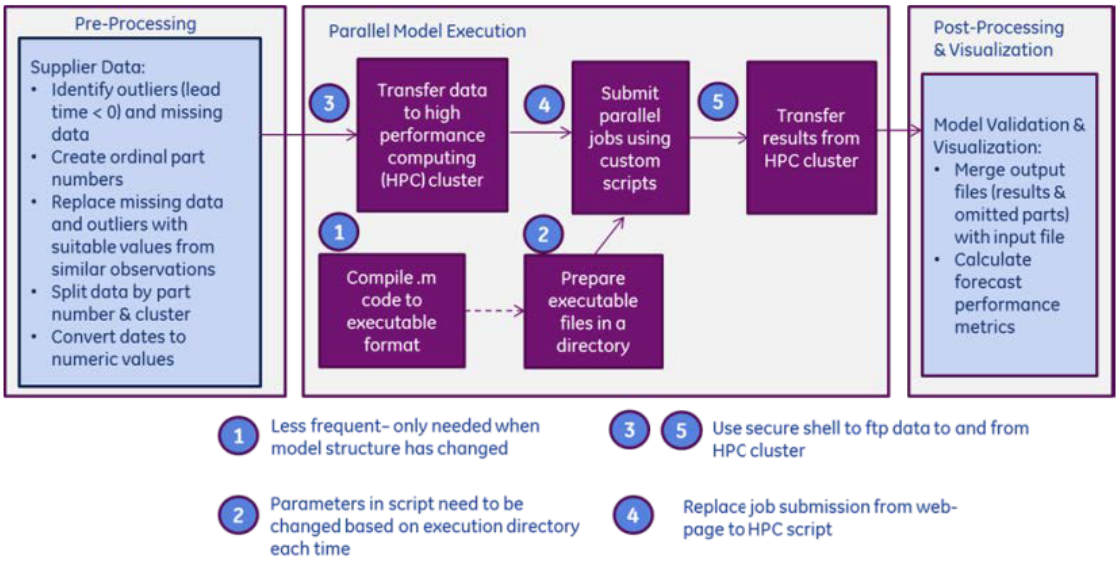


Figure 6. The prediction system by Banerjee et al. [2]

2.3.3 Random forest with dimension reduction

A study by Liu et al. [3] is the latest of the relevant studies found. They were also interested in predicting delivery times of aircraft engine parts, and some of the authors are the same as in the study by Banerjee et al. [2]. Therefore this can be seen as a continuation of the previous study. Liu et al. took a different approach to the problem and utilized ensembles of decision trees. In particular, they compared RFs to quantile regression forest (QRF)s [16]. They also used the whole dataset for training the models instead of training separate models for each supplier.

In order to include categorical features, they expanded them into dummy variables. Because a large number of categories can cause the curse of dimensionality, they used principal component analysis (PCA) to reduce the dimensionality. However, because of computational limitations and the complexity of PCA, they needed to first reduce the number of categories by combining the least frequent categories into one. After applying PCA, they kept the m features that had the most contributions in the top-n principal components and combined the rest to one. Figure 7 contains a diagram of the prediction system as a whole.

The data consisted of four ordinal and four categorical features. The categorical data were reduced with the above method so that each feature contained 15 categories. The training set had 53 105 samples, which is much smaller than the previous study. 10-fold cross-validation was used to select the models. The models provided slightly better predictions that the estimates made by the suppliers. The median error of the suppliers was 6 days

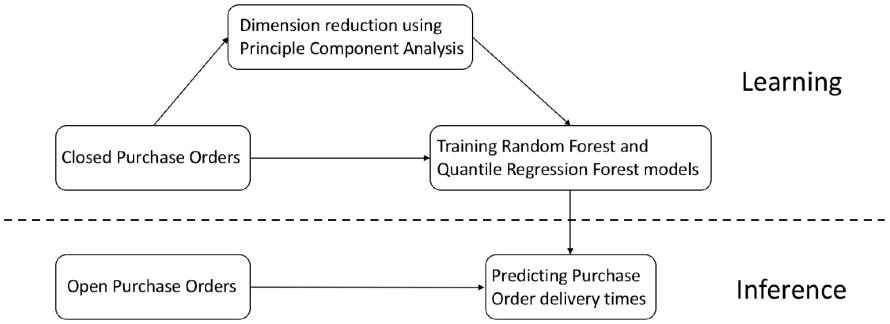


Figure 7. The prediction system by Liu et al. [3]

while the median errors for RF and QRF were 5.07 and 5.04 days, respectively. The benefit of the models becomes more evident when looking at larger percentiles. 95 per cent of the supplier estimates have an error less than 41 days while the same value for RF and QRF is 26.95 and 27.95 days. A comparison of the errors of supplier estimates and the predictions by the RF are shown in Figure 8. It contains only the errors below 60 days.

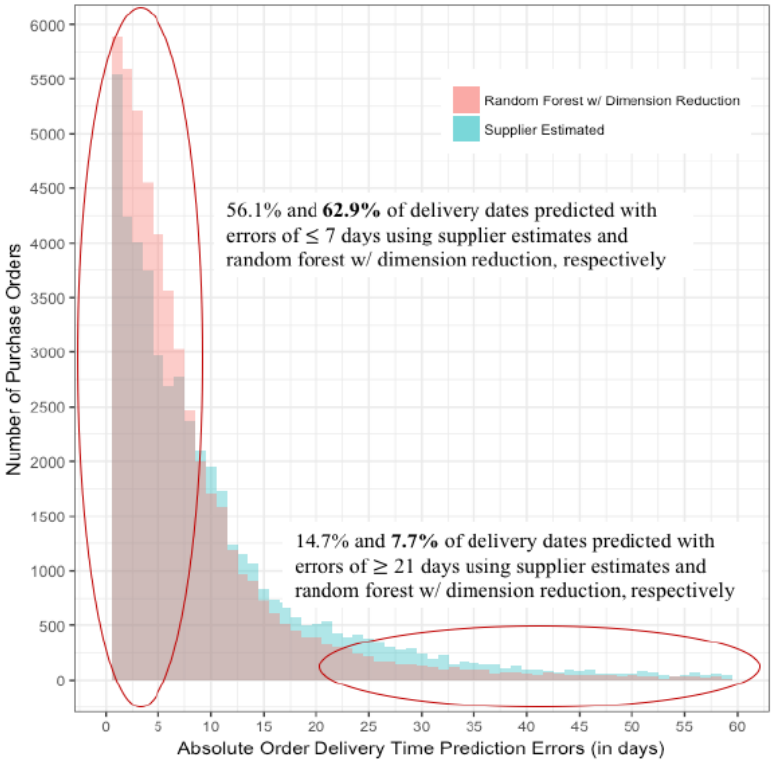


Figure 8. Comparison of the errors of supplier estimates and the predictions by a random forest by Liu et al. Only the errors below 60 days are shown. [3]

3 METHODS USED IN THIS STUDY

3.1 Ensemble learning

In the recent decades, ensemble learning has become a popular method in machine learning. The goal of ensemble learning is to make the model more general by combining multiple base-models. The base-models are often simple, so called weak models, with high variance, and the goal is to combine them into one strong model with low variance. [5]

Two popular ensemble-based algorithm families are bagging and boosting. In bagging, a number of independent models are trained on separate datasets, which are formed by sampling with replacement from the original training dataset. In boosting, the weak models are trained iteratively so that each subsequent model focuses on correcting the errors made by the previous models. [5, 17]

3.2 Decision trees

Decision trees are often used as weak models in ensemble learning due to their simplicity and interpretability. A decision tree consists of nodes that each split the dataset into two subsets. Each node considers typically one feature and consists of a threshold value. The dataset is split so that the values of the feature are above the threshold in one subset and below it in the other. The goal is for the two subsets to be as homogeneous, or pure, as possible with respect to the target variable. [18]

With enough nodes, one can ideally make accurate predictions of the target variable based on the features. The prediction is made by following the tree from top to down. A path of conditional statements based on the splitting thresholds and the features is being followed and eventually, a terminal node is reached. The terminal nodes are called leaves. In a classification task, the prediction is the label that is the most frequent in the leaf for the training data. In regression, the average of the training samples in the leaf is used as the predicted value. [7]

Mathematically, a regression tree h with J leaves with the input $\mathbf{x}$, is defined as

$$h(\mathbf{x}; \{b_j, R_j\}_1^J) = \sum_{j=1}^{J} b_j 1(\mathbf{x} \in R_j), \tag{1}$$

where R_j are the regions of $\mathbf{x}$ that correspond to each leaf. The indicator function $1(\cdot)$ takes value 1 when its arguments are true and zero otherwise. Each coefficient b_j is the average of the responses that fall into leaf j. [8]

The tree could be grown until the data cannot be split anymore, but this would most likely lead to overfitting the training data and the tree would fail in predicting for an unseen dataset. The tree can be pruned to prevent it from growing too large. To prune the tree during growing, one can define one or more of the following limitations: a minimum number of points in a node, an error threshold of a node, or a maximum depth of the tree. As soon as a node or the tree meets one of the limitations, the growth is halted. Figure 9 contains an example of a pruned decision tree and the predictions made by it. [18]

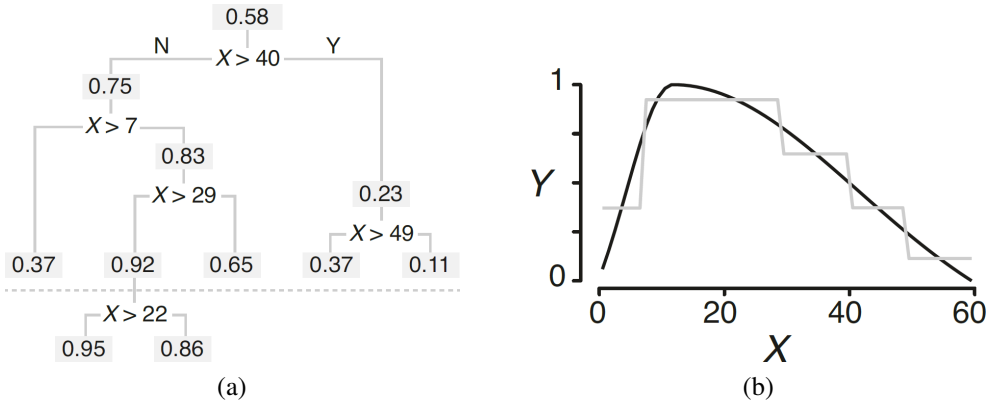


Figure 9. Example of a decision tree with one feature X and target variable Y [18]; (a) Structure of the tree. The relative decrease of the error in the split below the dashed line did not meet a user-specified threshold, so that split was declined and the growth was stopped. (b) The black line is the underlying function $Y(X)$ and the grey line represents the predictions made for each value of X.

In the conventional classification and regression trees (CART), the best split is found by going through each possible split in each feature and selecting the purest one. This can often be made faster by utilising algorithmic shortcuts such as ordering the splits by the size of the conditional mean or proportion. The tree is grown greedily, meaning that the algorithm focuses on the current split without looking to past or future splits. Many extensions to the original CART have been proposed in the past 30 years. The details of the original CART can be found in [19]. [20]

In CART, the tree is grown depth-wise, meaning that each branch is grown to its full depth before a new branch is made higher up in the tree. In the best-first method, the tree is always grown from the node with the best split. The two methods produce identical trees if there are no limitations, but if the depth of the tree is limited by a fixed number,

the best-first method produces better results. Figure 10 shows the difference between depth-first and best-first methods for growing a tree. [21]

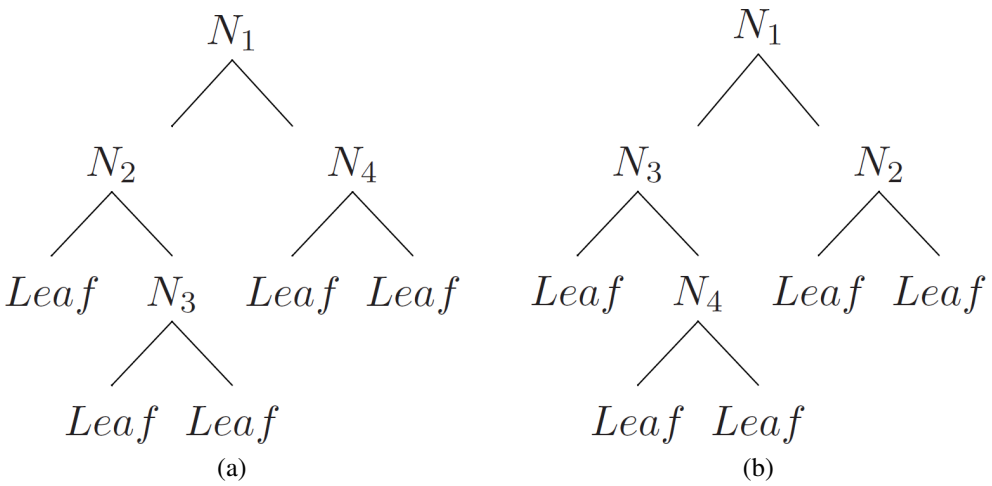


Figure 10. Different ways of growing a decision tree; (a) Depth-first. (b) Best-first. The subscripts of N represent the order of growing the nodes. [21]

3.3 Gradient boosting

GBM is an ensemble algorithm belonging to the family of boosting methods. It has been used in many recent prediction and classification applications across fields such as energy distribution [22, 23], medicine [24–26], civil engineering [27], geology [28, 29], education [30] and economics [31].

In boosting, each base-model is added to the ensemble sequentially. The distinctive feature of gradient boosting is the use of gradient descent optimisation method in the function space. Often, like in the case of neural networks, gradient descent is used in the parameter space in order to tweak the parameters to optimise the loss function. In GBM, by operating in function space, the outputs of the base-models are tweaked sequentially. After adding the base-model to the ensemble, its parameters are not tweaked anymore and the optimisation will happen on the subsequent base-models. Because GBM optimises the weak models as they are being built, it can achieve better results than a RF with the same computing costs. This comes at a cost of a need for hyper-parameter tuning, most importantly the learning rate ν. [32]

Below, the principles of GBM are explained by following mostly the same notation as the developer of the method, Jerome H. Friedman [8], albeit in a simplified manner. A similar

method was developed simultaneously by Mason et al. [33], and both methods were built on the work by Leo Breiman [34]. If one wants to get a more thorough understanding of GBM, it is recommended to read the articles cited above.

The goal of any machine learning prediction task is to estimate function $F(\mathbf{x})$, which maps the input variables $\mathbf{x} = x_1, \ldots, x_n$ to a target variable y, using a training dataset $\{\mathbf{x}_i, y_i\}_1^N$ with N samples. It is usually achieved by minimising a loss function $L\big(y, F(\mathbf{x})\big)$. In boosting, the estimate function $F(\mathbf{x})$ is grown iteratively using base-models. In GBM with regression trees, each regression tree $h(\mathbf{x}, \mathbf{a}_k)$ is multiplied by a coefficient ρ. Note that, to simplify the notation, a substitution $\mathbf{a} = \{b_j, R_j\}_1^J$ is used here. With K iterations,

$$F(\mathbf{x}; \{\rho_k, \mathbf{a}_k\}_1^K) = \sum_{k=1}^{K} \rho_k h(\mathbf{x}; \mathbf{a}_k). \tag{2}$$

The parameters ρ and $\mathbf{a}$ are obtained by using gradient descent in the function space. The base-models h are fit to so-called pseudo-responses $\tilde{y} = \{\tilde{y}_i\}_1^N$, which are defined as

$$\tilde{y}_i = -\left[\frac{\partial L\big(y_i, F(\mathbf{x}_i)\big)}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{k-1}(\mathbf{x})}. \tag{3}$$

In other words, the pseudo-responses are the negative gradient of the loss function $L(y, F)$ over $F(\mathbf{x})$ in the previous iteration. Coefficients ρ_k are obtained by line search

$$\rho_k = \arg\min_{\rho} \sum_{i=1}^{N} L\big(y_i, F_{k-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_k)\big) \tag{4}$$

and the approximation is updated by

$$F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \rho_k h(\mathbf{x}; \mathbf{a}_k). \tag{5}$$

When using regression trees as base-models, (5) can be expressed as

$$F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \rho_k \sum_{j=1}^{J} b_{jk} 1(\mathbf{x} \in R_{jk}). \tag{6}$$

The coefficients can be combined as $\gamma_{jk} = \rho_k b_{jk}$, so that

$$F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jk} 1(\mathbf{x} \in R_{jk}). \tag{7}$$

Finally, the learning rate ν can be implemented to work as a regularizer. The final form of the update step is

$$F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \nu \sum_{j=1}^{J} \gamma_{jk} 1(\mathbf{x} \in R_{jk}). \tag{8}$$

Many implementations of GBM introduce additional regularization terms. One of those implementations is LightGBM, which was used in this study.

In this study, least absolute deviation (LAD) was used as the loss function L so that $L(y, F) = |y - F|$. In LightGBM, the usage of LAD is reflected as the user selecting its aggregate mean absolute error (MAE) as the loss function. The exact definition of MAE and the reasons for selecting it are presented in Chapter 4. When using LAD, (3) becomes

$$\tilde{y}_i = \text{sign}\left(y_i - F_{k-1}(\mathbf{x}_i)\right) \tag{9}$$

and

$$\gamma_{jk} = \underset{\mathbf{x}_i \in R_{jk}}{\text{median}} \{y_i - F_{k-1}(\mathbf{x}_i)\}, \tag{10}$$

which is the median of the current residuals in the jth terminal node at the kth iteration. The goal of the regression trees at each iteration is to predict the sign of the current residuals $y_i - F_{k-1}(\mathbf{x})$. The approximation $F(\mathbf{x})$ is then updated by adding the median of the residuals in each of the leaves. Algorithm 1 describes the whole process of GBM algorithm with regression trees and LAD in pseudo-code.

Algorithm 1 Gradient boosting algorithm with regression trees as base-models h and LAD as the loss function [8].

function LAD_TREEBOOST($\mathbf{x}$, y, K, ν)
  $F_0(\mathbf{x}) \leftarrow \text{median}\{y_i\}_1^N$
  for $k \leftarrow 1$ **to** K **do**
    $\tilde{y}_i \leftarrow \text{sign}\left(y_i - F_{k-1}(\mathbf{x}_i)\right), \quad i = 1, \ldots, N$
    $\{R_{jk}\}_1^J \leftarrow J$-terminal node *tree*$(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$
    $\gamma_{jk} \leftarrow \text{median}_{\mathbf{x}_i \in R_{jk}}\{y_i - F_{k-1}(\mathbf{x}_i)\}, \quad j = 1, \ldots, J$
    $F_k(\mathbf{x}) \leftarrow F_{k-1}(\mathbf{x}) + \nu \sum_{j=1}^{J} \gamma_{jk} 1(\mathbf{x} \in R_{jk})$
  end for
  return $F_k(\mathbf{x})$
end function

3.4 LightGBM

At the time of writing, LightGBM [9], XGBoost [35] and CatBoost [36] are the competing state-of-the-art implementations of GBM, with XGBoost being the earliest of them. LightGBM introduced gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB) to decrease the computing cost, claiming to produce almost the same accuracy with up to nine times faster training times than XGBoost. The goal of CatBoost was to improve the accuracy in applications with categorical features, and its computing cost is said to be similar to LightGBM. The implementation used for this study was LightGBM because of a large amount of data, limited computing power, and the author's previous experience with it.

When using GBM with decision trees, the part that takes the most resources in terms of computing time and memory is fitting the decision trees, and especially finding the best split points. To reduce the computing cost, LightGBM uses a histogram-based algorithm for finding the split points. The algorithm buckets continuous feature values into discrete bins and uses those to construct feature histograms during training. The maximum number of bins per feature and the minimum number of samples in each bin are considered hyper-parameters. They are named *max_bin* and *min_data_in_bin*, respectively.

GOSS is a way of sampling the data during training with the goal of keeping the samples that produce the most information gain. The idea is that the samples which produce the largest gradients of the loss-function are the most under-trained, and therefore will produce the most information gain. Therefore, in each iteration m of GBM, with the dataset $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, GOSS keeps the samples with the largest gradient $g_m(x_i)$ and performs random sampling on the rest of the samples. Therefore, the sampled dataset contains two subsets, one made of large-gradient samples and the other of small-gradient samples. The sizes of the two subsets are considered hyper-parameters. The algorithms of the histogram-based decision tree splitting and GOSS are presented in pseudo-code in [9], as well as more detailed explanations.

EFB is a nearly lossless method of reducing the dimensionality of a sparse dataset by bundling mutually exclusive features together. The term "mutually exclusive" refers to features that never take nonzero values simultaneously. The most typical occurrences of mutually exclusive features are when categorical features are one-hot encoded. In one-hot encoding, a categorical feature with n categories is expanded to n features so that each feature corresponds to a category c. Each encoded sample is valued 1 if the original sample belongs to c and 0 otherwise.

Because some of the categorical features in this study had a large number of categories, i.e. high cardinality, it would have been impractical to use one-hot encoding due to memory limitations. Therefore, the categorical features were encoded ordinally, so that the most frequent category was set as 1, and the most infrequent category as n. LightGBM's documentation [37] also notes that one-hot encoding is a suboptimal approach for tree learners. Especially, if a categorical feature has a high cardinality, the tree tends to be unbalanced and needs to grow deep for good accuracy. LighGBM includes an approach for obtaining optimal splits for ordinally encoded categorical features in regression trees, based on the discoveries of Walter D. Fisher [38].

LightGBM contains several additional hyper-parameters such as regularization terms that are not discussed in depth in this study. The details of all of the hyper-parameters can be found in LightGBM's documentation [39]. The optimisation process used for all of the hyper-parameters is described in Chapter 4.

3.5 Tree-structured Parzen estimator

According to initial experiments, GBM can be sensitive to hyper-parameters. Because manually tuning the hyper-parameters can be laborious and slow, an automated hyper-parameter optimisation method was used. TPE was selected as the method because of its easy-to-use and well-documented implementation in Optuna [40]. Below, the principles of TPE are described similarly to the article where it was first presented [41]. The notation is different than in the original article so that the notation stays consistent throughout this study.

TPE belongs to a class of optimisation algorithms called sequential model-based global optimisation (SMBO). The goal of these algorithms is to use a computationally simple function S as a surrogate for a more expensive function f when minimising it. The surrogate S minimises a cheap model M, which starts as a guess, but often gets more accurate with respect to f after re-fitting through many iterations. SMBO can suffer from the problem of converging to some local optimum. This can be mitigated by using a probability density function as M and drawing samples from it when doing the minimisation task.

In this study, the evaluation of $f(\theta)$ is composed of two steps. First, GBM is trained with the training set $\{\mathbf{x_i}, y_i\}_1^N$ and hyper-parameters θ. Then, the fitness of the output $\hat{y}$ against the test set is evaluated by the objective function O. The output of O is the output of $f(\theta)$. The choice of O for this specific case, as well as the split between train and test sets, are

described in Chapter 4.

The surrogate function S is minimising a surrogate model M with respect to the hyper-parameters θ. In essence, the costly and often impractical minimisation task

$$\theta^* = \arg\min_{\theta} f(\theta) \tag{11}$$

is replaced by a simpler task

$$\theta^* = \arg\min_{\theta} S(\theta, M). \tag{12}$$

The task (11) is approximated by updating M sequentially and evaluating (12) after each update. In each iteration of the algorithm, $f(\theta^*)$ is evaluated with θ^* from the previous iteration, and M is fit to the whole observation history $\mathcal{H}$ of $\big(\theta^*, f(\theta^*)\big)$. The steps of the generic SMBO are listed in Algorithm 2.

Algorithm 2 Generic SMBO [41].

```
function SMBO(f, M_0, T, S)
    H ← ∅
    for t ← 1 to T do
        θ* ← arg min_θ S(θ, M_{t−1})
        Evaluate f(θ*)                          ▷ Expensive step
        H ← H ∪ (θ*, f(θ*))
    end for
    Fit a new model M_t to H.
    return H
end function
```

In TPE, the expected improvement (EI) [42] is used as the optimisation criterion. It is defined as

$$\mathrm{EI}_{z^*}(\theta) = \int_{-\infty}^{\infty} \max(z^* - z, 0) p_M(z \mid \theta)\, dz, \tag{13}$$

where z^* is a predefined threshold and $p_M(z \mid \theta)$ is the probability of z given θ, based on model M. In TPE, M is not explicitly defined, and $p_M(\theta \mid z)$ is defined instead. It is then used to calculate $p_M(z \mid \theta)$ according to the Bayes' theorem. Two gaussian mixture models (GMMs), called $l(\theta)$ and $g(\theta)$, are used to define

$$p_M(\theta \mid z) = \begin{cases} l(\theta), & \text{if } \theta \leq z^* \\ g(\theta), & \text{if } \theta \geq z^* \end{cases}. \tag{14}$$

In each iteration, the models $l(\theta)$ and $g(\theta)$ are fitted by using the hyper-parameter combinations θ_l and θ_g, respectively. The combinations are defined as

$$\theta_l = \{\theta_i \mid f(\theta_i) \leq z^*\}_1^t \quad \text{and} \tag{15}$$

$$\theta_g = \{\theta_i \mid f(\theta_i) \geq z^*\}_1^t, \tag{16}$$

where t is the current iteration.

The threshold z^* is chosen to be some value of the observed z values, so that $p(z < z^*) = \Gamma$, but a model for $p(y)$ does not need to be explicitly defined. Ultimately, by utilising $p(\theta \mid z)$, $l(\theta)$, $g(\theta)$ and Γ defined above, (13) can be restructured to a format that satisfies

$$\mathrm{EI}_{z*}(\theta) \propto \left(\Gamma + \frac{g(\theta)}{l(\theta)}(1 - \Gamma)\right)^{-1}. \tag{17}$$

The details of how (17) can be derived have been explained by Bergstra et al. [41].

During each iteration, a number N_θ of candidates for θ^* are drawn from the distribution defined by l. They are evaluated according to (17) and the candidate with the highest evaluation is returned. If there is some prior knowledge about the possible hyper-parameters θ, prior distributions l_{prior} can be set independently for each of them. The hyper-parameters and their prior distributions used for this study are described in Chapter 4.

When the definitions above are substituted for those in Algorithm 2, a process described in Algorithm 3 emerges. Note that, in [41], TPE was described only in written form instead of laying out the algorithm in pseudocode. Therefore, Algorithm 3 is derived from their written description and the documentation of the implementation in Optuna [43].

Algorithm 3 TPE, based on [41].

function TPE(f, T, N_θ, l_{prior}, Γ)
  $\mathcal{H} \leftarrow \emptyset$
  $l_0 \leftarrow l_{\text{prior}}$
  $g_0 \leftarrow l_{\text{prior}}$
  for $t \leftarrow 1$ **to** T **do**
    Draw samples $\{\theta_i\}_1^N$ from l
    $\theta^* \leftarrow \arg\max_{\theta \in \{\theta_i\}_1^N} \left(\Gamma + \frac{g_{t-1}(\theta)}{l_{t-1}(\theta)}(1-\Gamma)\right)^{-1}$
    Evaluate $f(\theta^*)$ ▷ Expensive step
    $\mathcal{H} \leftarrow \mathcal{H} \cup \left(\theta^*, f(\theta^*)\right)$
    Fit l_t and g_t to $\mathcal{H}$
  end for
  return $\mathcal{H}$
end function

3.6 Shapley additive explanations

If one wants to analyse the importance of individual features for a model, it would be useful to have methods for explaining how the features affected the model's output. These explanations can be either local or global. Local explanations are made individually for each input sample and can be used to analyse how a feature affected a single prediction. Global explanations tell a big picture of the model's performance and can be used to analyse the importance and effect of each feature as a whole.

There exist several different methods for global and local explanations for tree-based machine learning models, but only recently a method that combines both worlds has been developed [44]. The method is called TreeExplainer and it makes use of the structure of tree-based models to generate Shapley additive explanations (SHAP) [45] in low-order polynomial time. Below is a summarised description of TreeExplainer [44].

Shapley values originate from game theory. They can be used to represent the importance of features in a model when the model is treated as a cooperative game and the features are treated as players. Shapley values represent the only solution that satisfies the three following properties:

1. local accuracy,
2. consistency and
3. missingness.

The detailed descriptions of the above properties can be found in the articles by Lundberg et al. [44, 45]. Theorem 1, which follows results from cooperative game theory, was also proposed in those articles. Ultimately, they provided pseudocode for an algorithm that generates the Shapley values defined in the theorem in low-order polynomial runtime. They named the values generated by the algorithm SHAP values.

Theorem 1 *Only one possible feature attribution method based on F_x satisfies properties 1, 2 and 3:*

$$\phi_i(F, x) = \sum_{R \in \mathcal{R}} \frac{1}{n!} \Big(F_x(P_i^R \cup i) - F_x(P_i^R) \Big) \tag{18}$$

where $\mathcal{R}$ is the set of all feature orderings, P_i^R is the set of all features that come before feature i in ordering R, and n is the number of input features for the model. [44]

4 EXPERIMENTS

4.1 Data

The dataset used for the experiments consists of past closed purchase orders of a factory producing ship engines. The exact quantities of the dataset are not disclosed due to confidentiality to the business. The dataset contains a couple of hundred thousand samples from a couple of years. It was randomly sampled to train, validation and test sets containing 80 %, 10 % and 10 % of the samples, respectively. The initial set of independent variables was determined with a domain expert, and their descriptions are presented in Tables 1 and 2. The number of categories in the categorical variables varies from dozens to tens of thousands. All of the features are either equivalent to or derived from fields in the company's global enterprise resource planning (ERP) system. Because of the obvious correlation between the suppliers' estimates and the actual lead times, the error of these estimates was used as the target variable for the prediction models, named as *promise_error*. When displaying the results of the experiments, the suppliers' estimates were summed to the target variable. This made it easier to compare the predictions with the suppliers' estimates.

Figure 11 shows the Pearson correlation coefficients for each of the numerical features and the target variable. Most of the variables do not have a significant correlation with each other. Naturally, *purch_ord_date_month* and *stat_relv_del_date_month* have some positive correlation. Since 0.6 % of the suppliers' estimates are longer than one year, a later *purch_ord_date_month* generally implies a later *stat_relv_del_date_month*, except when the latter is in the following year. Variables *m_po_value_per_unit* and *net_weight* have a positive correlation, meaning that a heavy order generally implies an expensive order. Variable *m_po_value_per_unit* also has some positive correlation with *stat_relv_del_date_delta*, which means that the more expensive orders tend to take more time to deliver. The only independent variable that has a significant correlation with the target variable is *m_gr_process_time*. The positive correlation between these variables implies that the purchase orders with a long receipt processing time are delayed more than those with a short receipt processing time. Variable *m_gr_process_time* also correlates positively with *stat_relv_del_date_delta* and *m_po_value_per_unit*, meaning that orders with long lead times and high monetary value correspond to long receipt processing times.

Table 1. Categorical features before quantitative feature selection.

Feature name	Description
mat_group_key	Item group
mat_key	Specific type of item(s)
mrp_prof	Profit center
outline_agreement	Agreement for price and terms of delivery
payment_term_key	Payment terms
purch_group_key	Purchasing group
storage_loc	Location of the storage
vend_key	Supplier
wbs_element_key	Project related to the PO

Table 2. Numerical features before quantitative feature selection.

Feature name	Description
days_of_conf	Number of days it took for the vendor to confirm the PO
m_gr_process_time	Estimate of the goods receipt processing time
m_po_value_per_unit	Monetary value of the PO in euros
net_weight	Weight of the item(s)
po_quantity	Quantity of items
purch_ord_date_day	Day of month when the PO was made
purch_ord_date_month	Month when the PO was made
purch_ord_date_weekday	Weekday when the PO was made
purch_ord_date_year	Year when the PO was made
stat_relv_del_date_day	Day of month of the supplier's delivery date esimate
stat_relv_del_date_delta	Days between making the PO and the supplier's delivery date estimate
stat_relv_del_date_month	Month of the supplier's delivery date estimate
stat_relv_del_date_weekday	Weekday of the supplier's delivery date estimate

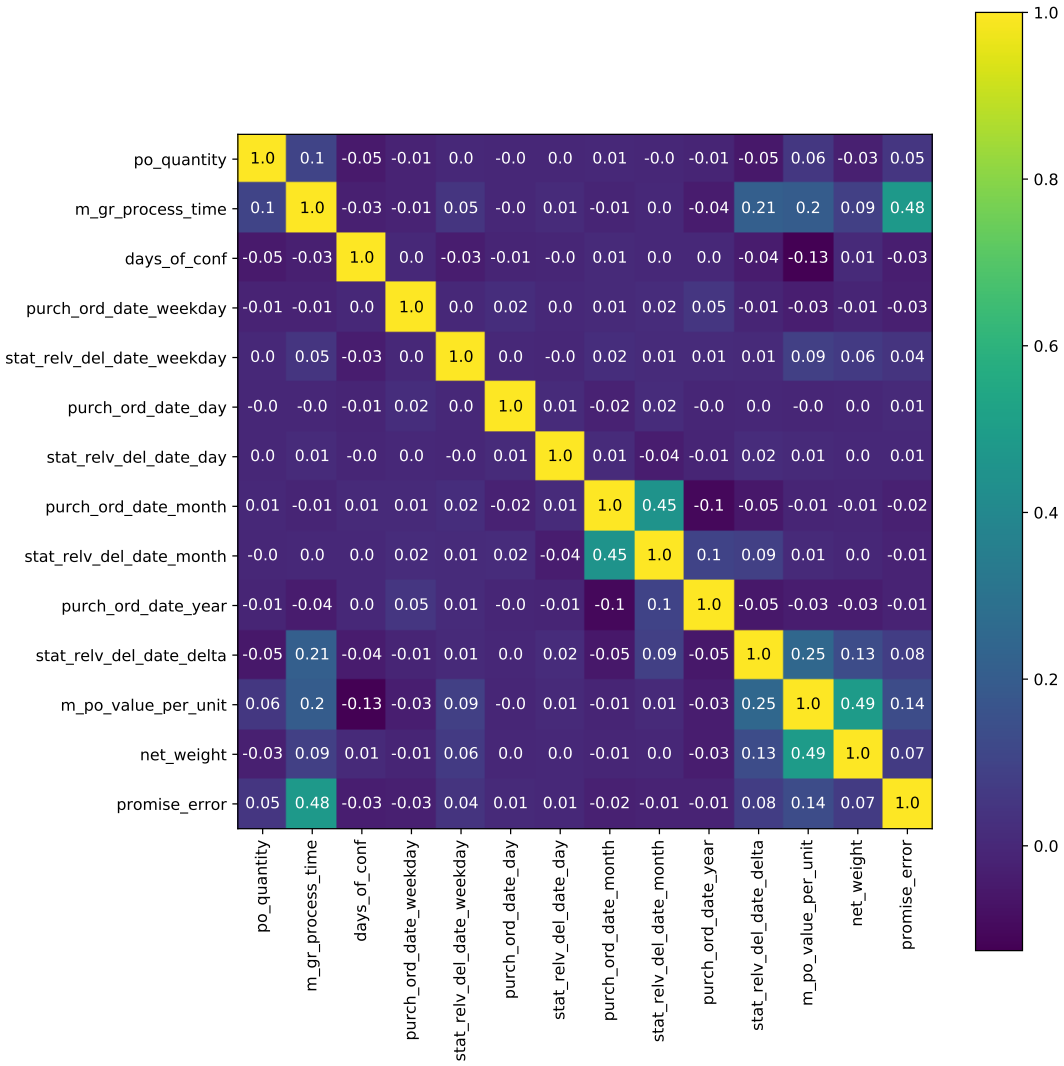


Figure 11. Pearson correlation coefficients for the numerical features and the target variable *promise_error*.

4.2 Evaluation criteria

The models were evaluated by comparing the model predictions to the observed lead times in the test set. Three different error metrics were used for the evaluation. The metrics are MAE, mean squared error (MSE) and coefficient of determination (R^2). They are defined as follows:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{19}$$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{20}$$

$$R^2 = \frac{\sum_{i=1}^{N}(y_i - \bar{y})^2}{\sum_{i=1}^{N}(\hat{y}_i - \bar{y})^2} \tag{21}$$

where N is the number of samples, y_i is the observed value of the target variable for the ith sample, $\hat{y}_i$ is the model's output for the ith sample and $\bar{y}$ is the arithmetic mean of the observed values, defined as

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i. \tag{22}$$

4.3 Description of experiments

When designing the experiments, there were three goals in mind. The first goal was to find the optimal hyper-parameters for LightGBM so that the purchase order lead times can be predicted as accurately as possible. The second goal was to find the features from the data that have the best predictive power. Since the two goals are dependent on each other, they were combined into one process, using TPE. The third goal was to produce a model that can make the predictions with minimal complexity, making it more probable to be general and saving resources such as memory and computing time. The first and third goals contradicted each other, and the aim was to find a suitable balance.

The hyper-parameters of LightGBM and the prior distributions chosen for them are shown in Table 3. According to preliminary experiments, GOSS and random sampling produced similar accuracy, and because the inclusion of GOSS would introduce two additional hyper-parameters, random sampling was used instead. One of the main hyper-parameters producing more complex models is *num_leaves*, indicating the maximum number of leaves in each tree. Since one of the goals was to generate a simple model, a logarithmic prior distribution was chosen for it. According to preliminary experiments, if a uniform

distribution was used, there was a possibility that TPE started from higher values, taking much more computing time per iteration. With the log-uniform distribution, most of the early iterations are performed with a small value for *num_leaves*. Larger values are explored rarely, and only if they produce a much better result than than the small values, they start to be explored more often. This way, at least the beginning of the hyper-parameter optimisation process is performed significantly faster. The log-uniform distribution was also chosen for the hyper-parameter *subsample_freq*, that indicates the frequency of subsampling the training dataset. In this case, the reasoning behind the log-uniform distribution was that the difference between the frequency of one has a more significant effect in the small values than the larger ones. A uniform distribution was used in all of the other hyper-parameters. For *colsample_bytree* and *subsample*, the limits are the lowest and highest possible bounds. For the rest, the limits were chosen empirically so that they extend beyond the ”reasonable” limits.

For each feature, the inclusion of the feature was added as a hyper-parameter because it was not clear which features are useful and which features only add noise. Since rare categories can cause overfitting, the categorical features were pre-processed so that the rare categories were combined to form a single category. Liu et al. [3] used a similar method in their study, and they chose to keep the most frequently occurring categories that maintain 80 % of the rows. Because the optimal value for the proportion of the categories to keep is not self-evident, and it can vary between features, it was added as a hyper-parameter for each categorical feature separately. A uniform distribution from 0.5 to 1.0 was used as the prior distribution.

Also, LightGBM has an option to handle categorical variables as categorical or numerical. According to the documentation [37], the latter can produce better results in some cases. Therefore, this option was also added as a boolean hyper-parameter for each categorical feature. All in all, there were 55 hyper-parameters to optimise, and they were optimised using TPE with 5-fold cross-validation in each trial.

MAE was used as the cost function in LightGBM because it was used in the earlier study by Liu et al. [3]. In the case of lead times to punish the errors proportionally. MSE was also used in initial experiments, but it caused an undesirable effect of dispersing the errors around zero. MAE provided more predictions with zero error and more errors with a large amplitude. It focuses more on making the small errors be near zero. This is a desirable effect because it is more effective to focus on enhancing the suppliers' predictions that are already closer to the truth than trying to enhance the predictions that are more unreliable. The latter can be thought of as lost causes while the former ones are those that have the

highest possibility of adding value.

Table 3. Hyper-parameters being optimised for LightGBM, named as in LightGBM's documentation [37] and the prior distribution chosen for each parameter. $\mathcal{U}(l, u)$, $\mathcal{U}\{l, u\}$ and $\mathcal{U}_{\log}\{l, u\}$ represent continuous uniform, discrete uniform, and discrete log-uniform distributions, respectively, with a lower bound l and upper bound u.

Hyper-parameter	Prior distribution
cat_l2	$\mathcal{U}(0.0, 100.0)$
cat_smooth	$\mathcal{U}(0.0, 100.0)$
colsample_bytree	$\mathcal{U}(0.0, 1.0)$
early_stopping_rounds	$\mathcal{U}\{1, 50\}$
learning_rate	$\mathcal{U}(0.001, 2.0)$
max_bin	$\mathcal{U}\{2, 10\,000\}$
min_child_samples	$\mathcal{U}\{1, 1\,000\}$
min_data_in_bin	$\mathcal{U}\{1, 100\}$
min_data_per_group	$\mathcal{U}\{1, 1\,000\}$
min_gain_to_split	$\mathcal{U}(0.0, 2.0)$
num_leaves	$\mathcal{U}_{\log}\{2, 2\,000\}$
reg_alpha	$\mathcal{U}(0.0, 1.0)$
reg_lambda	$\mathcal{U}(0.0, 1.0)$
subsample	$\mathcal{U}(0.0, 1.0)$
subsample_freq	$\mathcal{U}_{\log}\{1, 5\,000\}$

The choice of the objective function to minimize in TPE was an important decision. The natural choice is to use the mean of the values of LightGBM's cost function in the validation sets, and this was experimented first. Formally, the objective function is defined as

$$O_1 = \frac{1}{N} \sum_{i=1}^{N} \text{MAE}_{\text{val}}^{i} \tag{23}$$

where N is the number of folds in cross-validation, and $\text{MAE}_{\text{val}}^{i}$ is the MAE of the validation set in the ith fold.

GBM is prone to overfitting, and there is lots of noise in the data. Therefore, the method favoured overly complex models. A symptom of this is that the value of the cost function differs significantly between the training and validation sets. Because of the overly complex models, the training also took a long time making the hyper-parameter optimisation slow.

To tackle this problem, a custom cost function was used for TPE. The goal was to make the estimator favour simpler models where the training and validation errors are close to each other. This was achieved by adding a second term that consists of the absolute difference

of the MAE in the validation and training sets. Formally it is defined as

$$O_2 = \frac{1}{N}\sum_{i=1}^{N} \mathrm{MAE}^i_{\mathrm{val}} + \left|\mathrm{MAE}^i_{\mathrm{val}} - \mathrm{MAE}^i_{\mathrm{train}}\right| \tag{24}$$

where $\mathrm{MAE}^i_{\mathrm{train}}$ is the MAE of the training set in the ith fold.

With the above cost function, another minor problem remained. If one of the folds in the cross-validation vastly outperforms the others, it can cause the mean of the MAEs to decrease considerably. Therefore, to keep the model as stable as possible, the standard deviation of the validation MAEs, σ_{val}, was added as an additional term. Formally the new cost function is defined as

$$O_3 = \frac{1}{N}\sum_{i=1}^{N} \mathrm{MAE}^i_{\mathrm{val}} + \left|\mathrm{MAE}^i_{\mathrm{val}} - \mathrm{MAE}^i_{\mathrm{train}}\right| + \sigma_{\mathrm{val}} \tag{25}$$

where σ_{val} is the standard deviation of MAE of the validation sets.

The hyper-parameter optimisation was carried out using an implementation called Optuna [40]. To make the optimisation faster, the trials were pruned using hyperband method [46] using Optuna's default options.

Three experiments were performed, one for each objective function of the hyper-parameter optimisation. The experiments were named as Exp. 1, 2 and 3, numbered according to the subscript of the corresponding objective function. In each experiment, the hyper-parameter optimisation was run for 24 hours on an 8-core AMD EPYC 7571 processor with 64GB RAM.

4.4 Results

4.4.1 Hyper-parameter optimisation and feature relevance

The hyper-parameters were optimised and the features to be used were selected in each experiment via TPE. Figures 12 and 13 show the training and validation error of each experiment during the hyper-parameter optimisation, as the function of time in hours and number of trials, respectively. The cost functions O_2 and O_3 produce models with a smaller difference between the training and validation errors, compared to O_1. The different scale

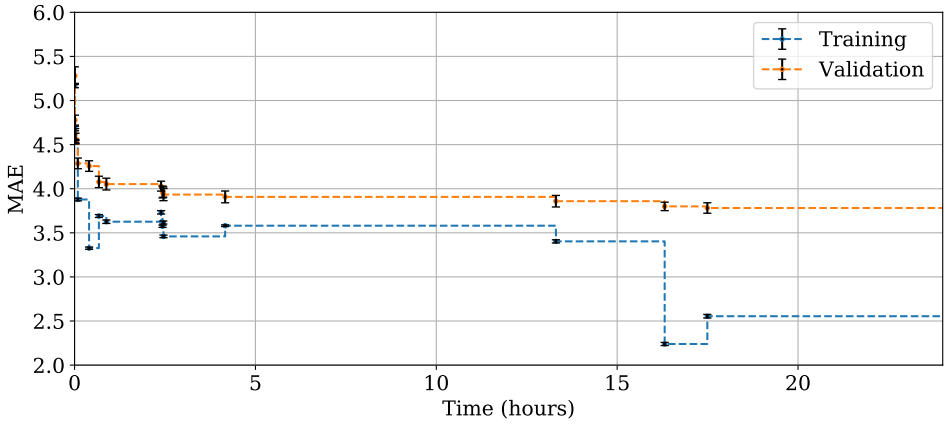
of the horizontal axis in Figure 13 indicates that in Exp. 2 and 3, four to six times more hyper-parameter combinations were processed than in Exp. 1. This is because the cost functions O_2 and O_3 prefer simpler models than O_1, causing the training times to be shorter. Another notable difference is that in Exp. 1, the validation error of the best model always decreases. In Exp. 2 and 3, it can increase if the difference between the training and validation error decreases. In Exp. 3, the validation error can also increase if the standard deviation of it decreases. Nonetheless, Exp. 1 produced the smallest validation error.

The final hyper-parameters for LighGBM chosen in each experiment are shown in Table 4. The hyper-parameters used by the model with the least validation error were chosen. Some hyper-parameter choices were close to each other in each experiment, but most of them varied considerably. This indicates that many different combinations of the hyper-parameters can produce adequate results. Noticeably, *colsample_bytree*, which refers to the fraction of features randomly selected for each tree, was almost the same in every experiment. The values correspond to 9, 9 and 10 features selected in Exp. 1, 2 and 3, respectively. The hyper-parameter *min_gain_to_split*, which means the minimal gain to perform a split, has a large effect on the complexity of the model, with small values producing more complex models and vice versa. The aim of producing less complex models in Exp. 2 and 3 is reflected by the larger value of *min_gain_to_split* in them, compared to Exp. 1.

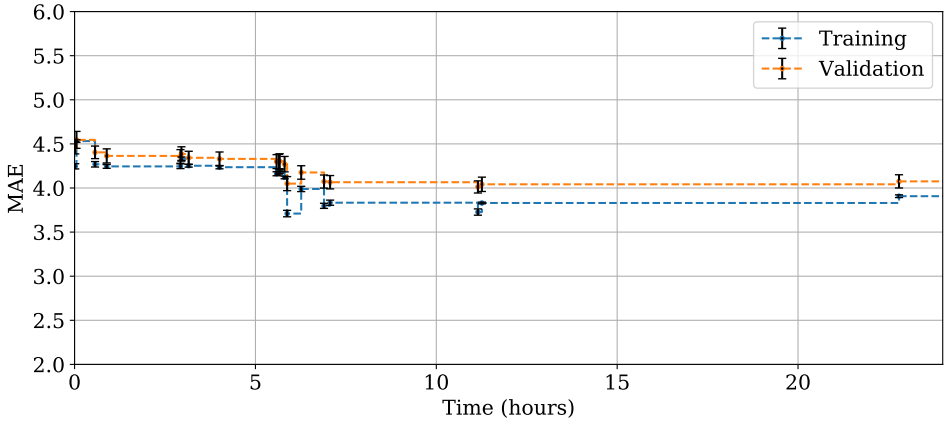
Table 4. The hyper-parameters chosen for LightGBM in each experiment.

Hyper-parameter	Exp. 1	Exp. 2	Exp. 3
cat_l2	78.9	26.2	75.6
cat_smooth	93.4	5.12	44.3
colsample_bytree	0.63	0.68	0.74
early_stopping_rounds	43	4	49
learning_rate	0.070	0.058	0.18
max_bin	9 447	1 834	4 060
min_child_samples	30	46	20
min_data_in_bin	74	64	91
min_data_per_group	727	599	869
min_gain_to_split	0.057	0.95	1.2
num_leaves	370	191	845
reg_alpha	0.69	0.76	0.50
reg_lambda	0.49	0.098	0.91
subsample	0.89	0.98	1.00
subsample_freq	3 315	2	37

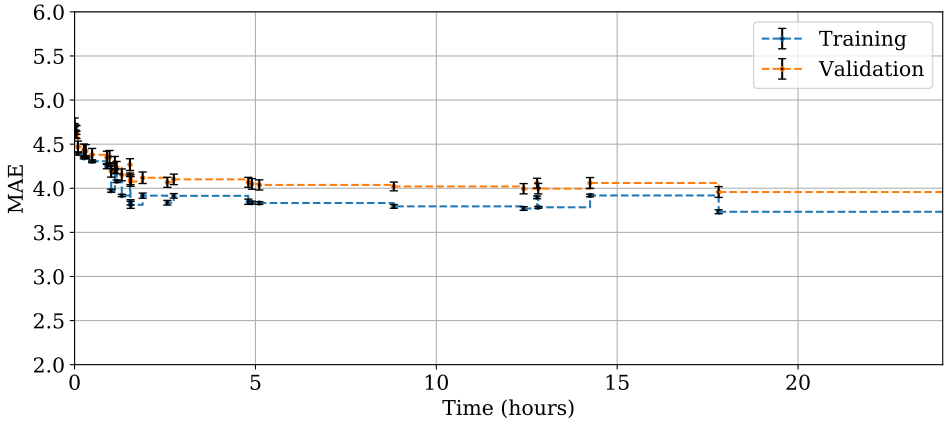
Table 5 shows the categorical features chosen in each experiment and the splitting rule



(a)

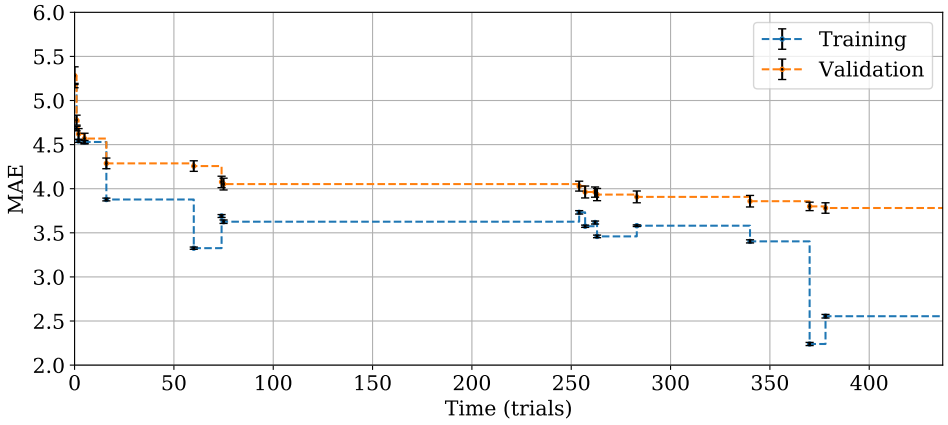


(b)

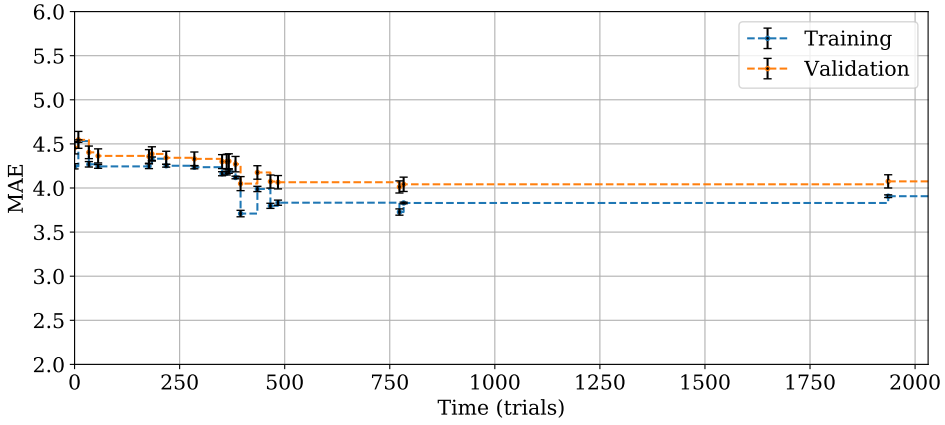


(c)

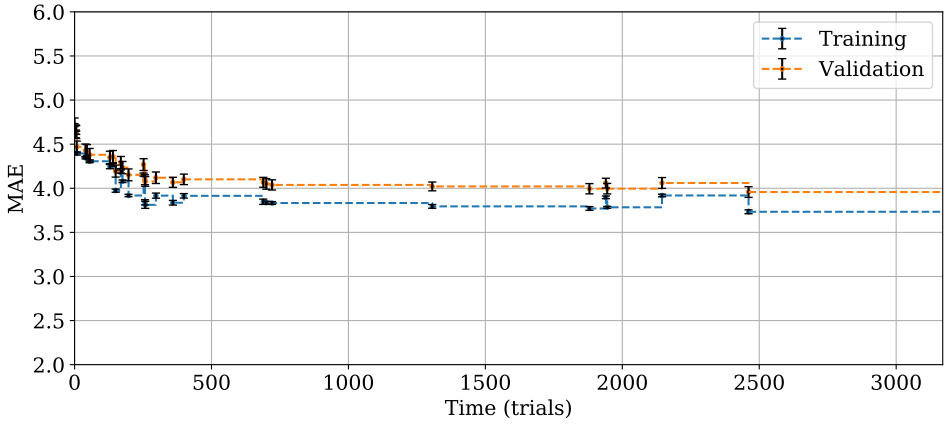
Figure 12. The development of the loss function of the models during the hyper-parameter optimisation, with time as hours. The dashed lines represent the best parameter combination found before a point in time, according to each cost function. The vertical position of the lines refers to the mean of MAE over the 5-fold cross-validation and the height of the black bars represent the standard deviation σ below and above the mean: (a) Exp. 1; (b) Exp. 2; (c) Exp. 3.



(a)



(b)



(c)

Figure 13. The development of the loss function of the models during the hyper-parameter optimisation, with time as trials. The dashed lines represent the best parameter combination found before a point in time, according to each cost function. The vertical position of the lines refers to the mean of MAE over the 5-fold cross-validation and the height of the black bars represent the standard deviation σ below and above the mean: (a) Exp. 1; (b) Exp. 2; (c) Exp. 3.

used for the features. The features *mrp_prof*, *purch_group_key* and *vend_key* were selected in every experiment while the others were chosen in one or two experiments. No feature was discarded in every experiment. This indicates that all of the features include some relevant information for the prediction, but they have some correlation in a way that they make each other useless. The only features that had the same splitting rule across all of the experiments were *mrp_prof*, *purch_group_key* and *wbs_element_key*, with all of these with the splitting rule designed for numerical features. It can be concluded that the choice of the splitting rule does not have a significant effect on the results in most cases. The portions of categories kept in each experiment are shown in Table 6. Most of the portions are close to each other across the experiments, with the most difference in *mrp_prof* and *wbs_element_key*.

Table 5. Splitting rules chosen for the categorical parameters in each experiment. A dash means that the feature was not used at all by the model.

Categorical feature	Exp. 1	Exp. 2	Exp. 3
mat_group_key	Categorical	-	-
mat_key	Numerical	Categorical	-
mrp_prof	Numerical	Numerical	Numerical
outline_agreement	-	-	Categorical
payment_term_key	-	Numerical	Categorical
purch_group_key	Numerical	Numerical	Numerical
storage_loc	Categorical	-	-
vend_key	Categorical	Categorical	Numerical
wbs_element_key	Numerical	-	Numerical

Table 6. The portions of categories kept chosen for the categorical parameters in each experiment. A dash means the feature was not used at all by the model.

Categorical feature	Exp. 1 (%)	Exp. 2 (%)	Exp. 3 (%)
mat_group_key	67.4	-	-
mat_key	79.9	74.0	-
mrp_prof	55.7	80.8	57.9
outline_agreement	-	-	96.1
payment_term_key	-	95.8	71.1
purch_group_key	73.4	92.4	86.4
storage_loc	69.3	-	-
vend_key	94.4	93.7	87.3
wbs_element_key	63.7	-	94.3

In Table 7, the choices of numerical features are presented. Five features were chosen in

each experiment, and those can be considered as the indisputably useful features. The features *po_quantity* and *purch_ord_date_day* were not chosen in any of the experiments and can be thought of as irrelevant. The rest were chosen in one or two experiments, indicating either vague importance or some correlation between the features.

Table 7. Whether or not each numerical feature was chosen in each experiment.

Numerical feature	Exp. 1	Exp. 2	Exp. 3
days_of_conf	Not chosen	Chosen	Chosen
m_gr_process_time	Chosen	Chosen	Chosen
m_po_value_per_unit	Chosen	Not chosen	Not chosen
net_weight	Chosen	Not chosen	Not chosen
po_quantity	Not chosen	Not chosen	Not chosen
purch_ord_date_day	Not chosen	Not chosen	Not chosen
purch_ord_date_month	Chosen	Chosen	Not chosen
purch_ord_date_weekday	Chosen	Not chosen	Chosen
purch_ord_date_year	Chosen	Chosen	Chosen
stat_relv_del_date_day	Chosen	Chosen	Chosen
stat_relv_del_date_delta	Not chosen	Chosen	Not chosen
stat_relv_del_date_month	Not chosen	Chosen	Chosen
stat_relv_del_date_weekday	Chosen	Chosen	Chosen

Figures 14, 15 and 16 show the impacts that each of the chosen features had to the prediction errors in the test set for each experiment. The impacts were determined by SHAP values [44]. The feature *m_gr_process_time* was one of the top three influencers in every metric in every experiment. Two other important features were *vend_key* in Exp. 1 and 2, and *outline_agreement* in Exp. 3. This indicates a possible correlation between the two features. It is highly likely because the former refers to the supplier of an item and the latter to the price and delivery agreement, and these often go hand in hand. The feature *days_of_conf* had zero effect in Exp. 2 and 3, meaning that it could be discarded without any effect on the predictions. During the hyper-parameter optimisation, the predictions were handled as continuous numbers, whereas in the prediction phase they were rounded to the nearest integer. This caused an effect where *days_of_conf* had a minor effect during the hyper-parameter optimisation phase, which got mitigated by rounding during the prediction. The SHAP values are presented in more detail in Appendix 1.

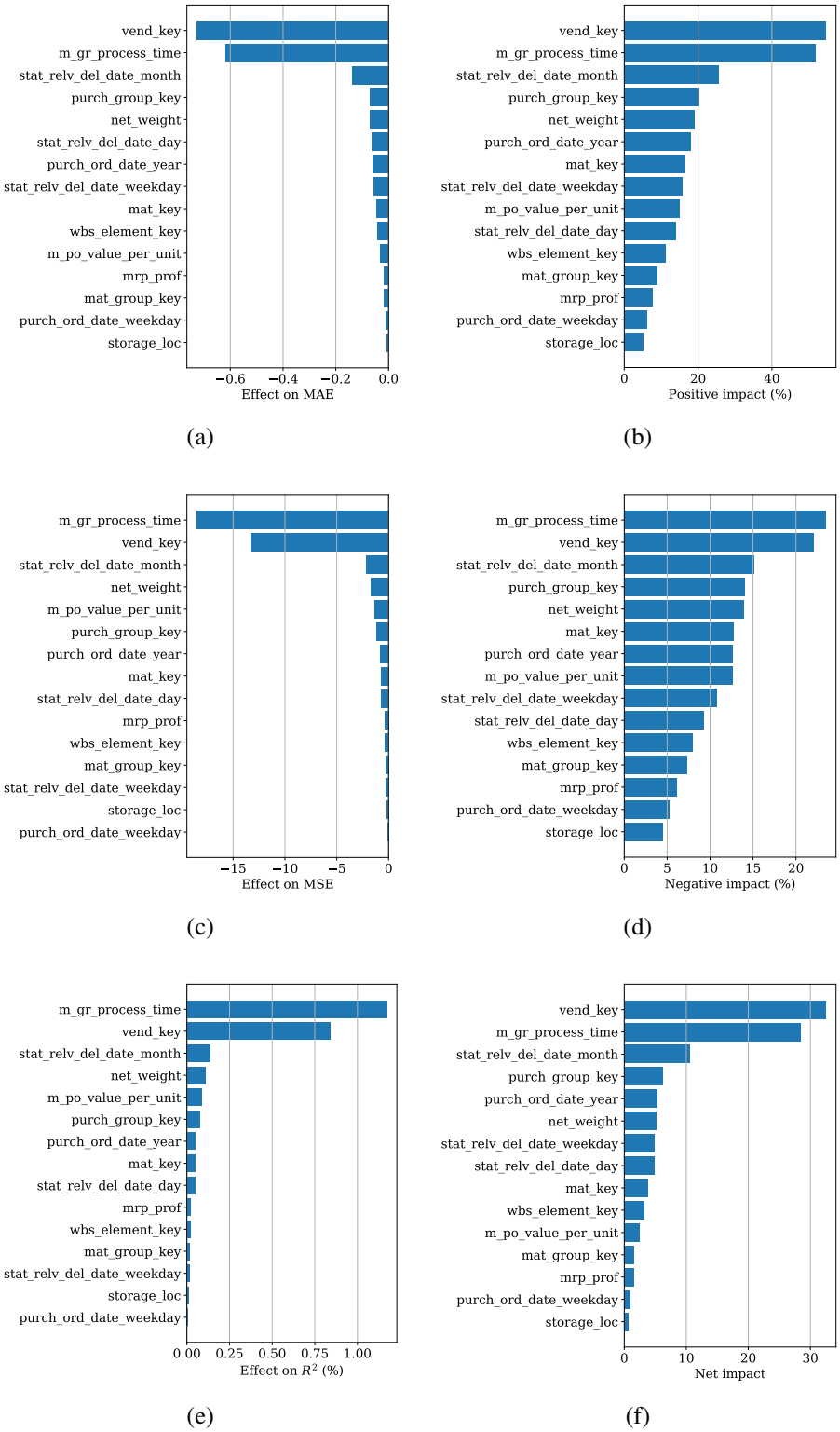


Figure 14. The impact that each of the features had to the errors in the test set in Exp. 1: (a)(c)(e) Effect on the error metrics; (b)(d) The proportion of samples where each feature had a positive or negative impact; (f) The difference between the proportion of samples with positive and negative impacts.

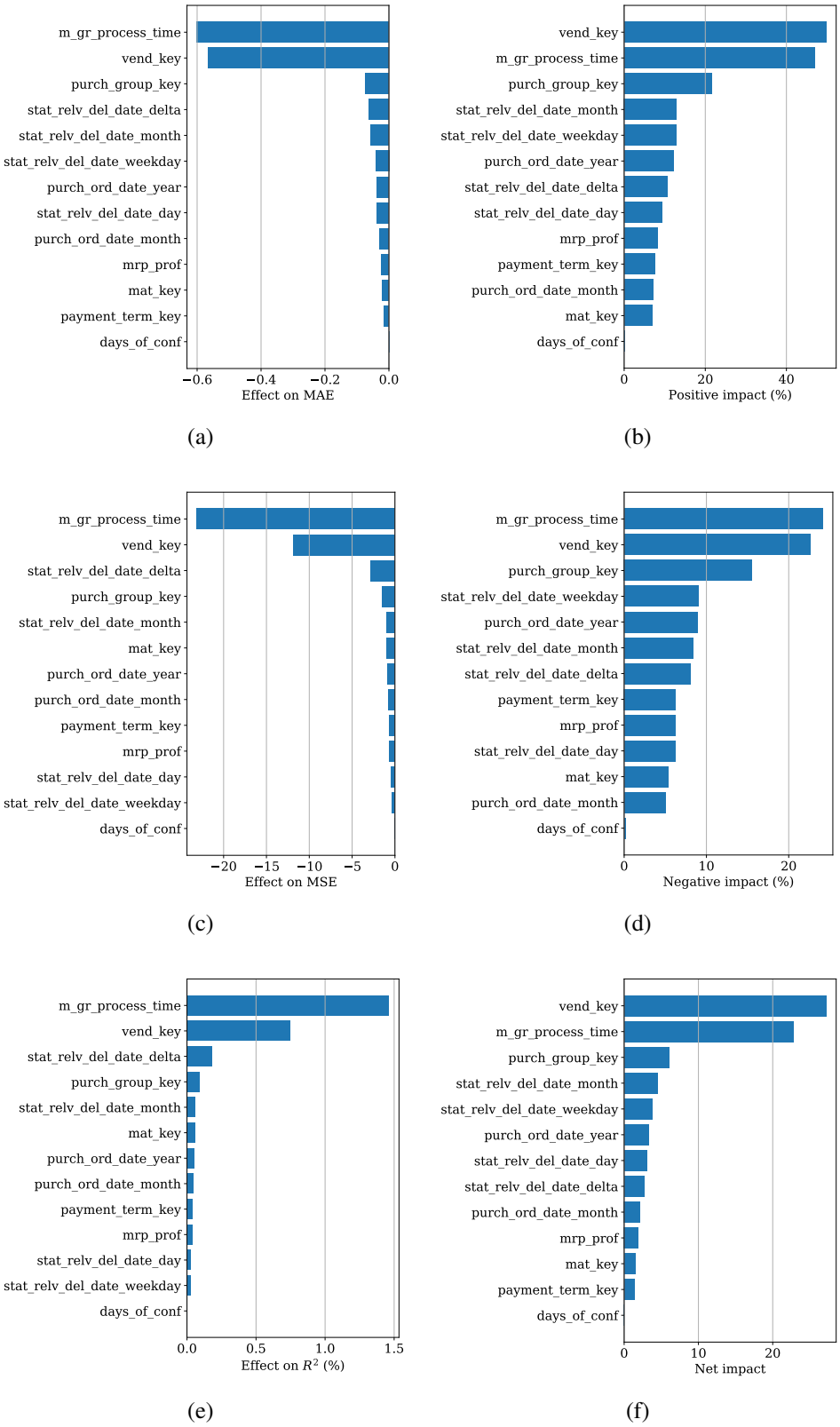


Figure 15. The impact that each of the features had to the errors in the test set in Exp. 2: (a)(c)(e) Effect on the error metrics; (b)(d) The proportion of samples where each feature had a positive or negative impact; (f) The difference between the proportion of samples with positive and negative impacts.

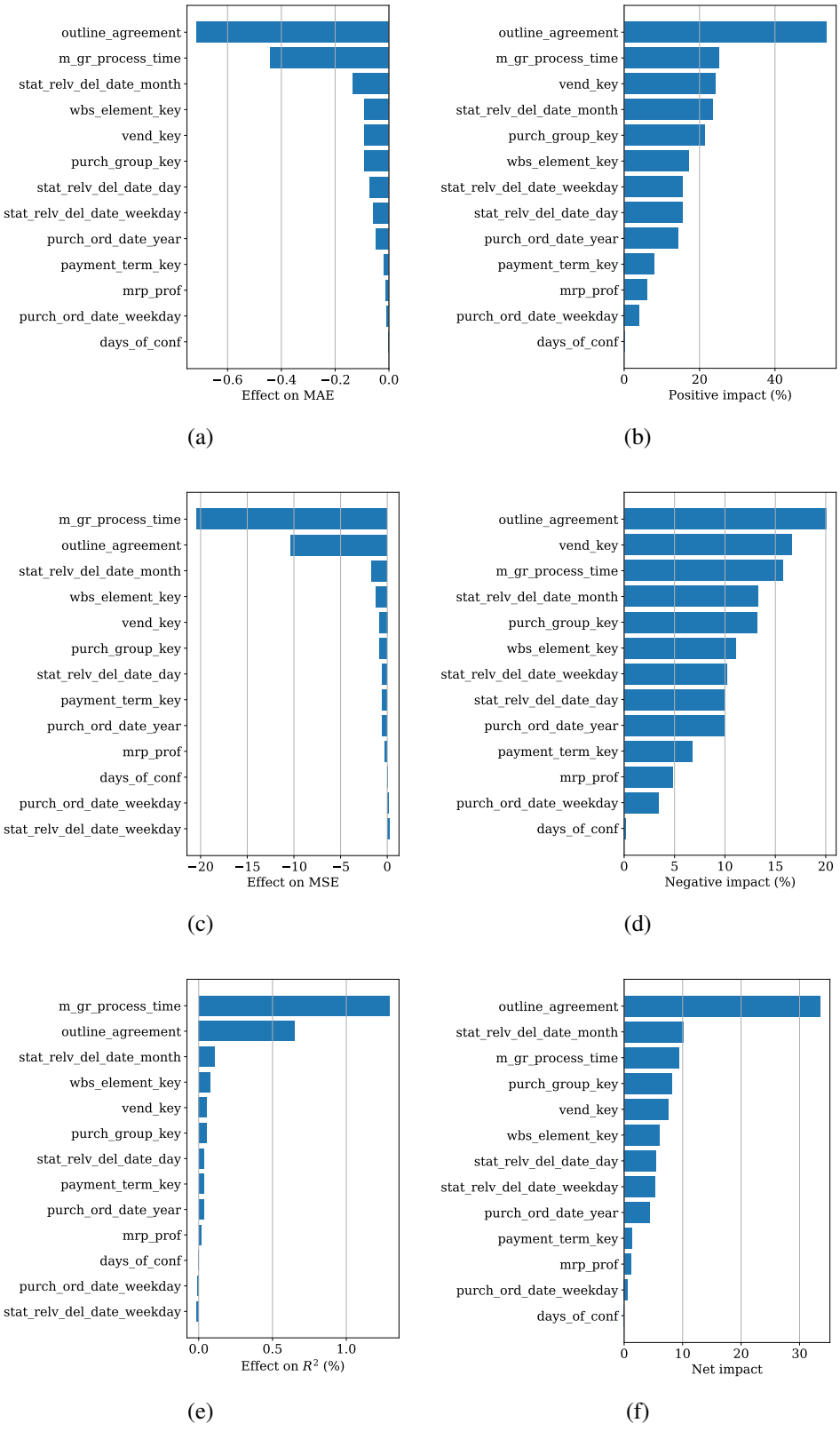


Figure 16. The impact that each of the features had to the errors in the test set in Exp. 3: (a)(c)(e) Effect on the error metrics; (b)(d) The proportion of samples where each feature had a positive or negative impact; (f) The difference between the proportion of samples with positive and negative impacts.

4.4.2 Training and complexity of the models

Figure 17 shows the training and validation loss of the selected model in each experiment as a function of training time. The training time is depicted as iterations, and in the case of GBM, it means trees. Again, Exp. 1 ended up with the smallest validation error, but the largest difference between the training and validation error. Furthermore, the model selected by Exp. 3 converged faster than the other models. The model selected by Exp. 1 had five times more trees than the other experiments, which often, but not necessarily, indicates a more complex model.

The larger complexity of the model in Exp. 1 is confirmed in Table 8. The model in Exp. 1 contained five times more trees and ten times more leaves and nodes than the model in Exp. 2. The difference was tenfold in the size of the model in memory as well as the training time. The models chosen in Exp. 2 and 3 were closer to each other in complexity.

Table 8. Metrics about the complexity of the models in each experiment. The first three columns represent the number of trees, leaves and nodes. The last columns depict the size of the model in memory in megabytes and the time it took to train the model with the specific dataset and hardware.

	Trees	Leaves	Nodes	Size (MB)	Training time (s)
Exp. 1	3 140	1 161 800	2 320 460	97.1	88.3
Exp. 2	593	100 033	199 473	9.0	8.5
Exp. 3	553	191 591	382 629	16.0	9.6

4.4.3 Prediction

Figure 19 shows histograms of the prediction errors in each experiment, compared to the promises made by the suppliers. The test set was used for the predictions. All of the experiments shifted some of the suppliers' errors of five or more days towards zero. The histograms of Exp. 1 and 2 are almost identical, whereas Exp. 3 produced more predictions with zero error and fewer predictions with an error of two or three days. The effect is more noticeable in Figure 18, which depicts the difference between the two histograms in each subfigure of Figure 19.

Table 9 shows the error metrics for the suppliers' estimates and the models chosen by each experiment, as well as the prediction times. Each of the experiments produced more accurate predictions than the estimates made by suppliers, with Exp. 1 being the most

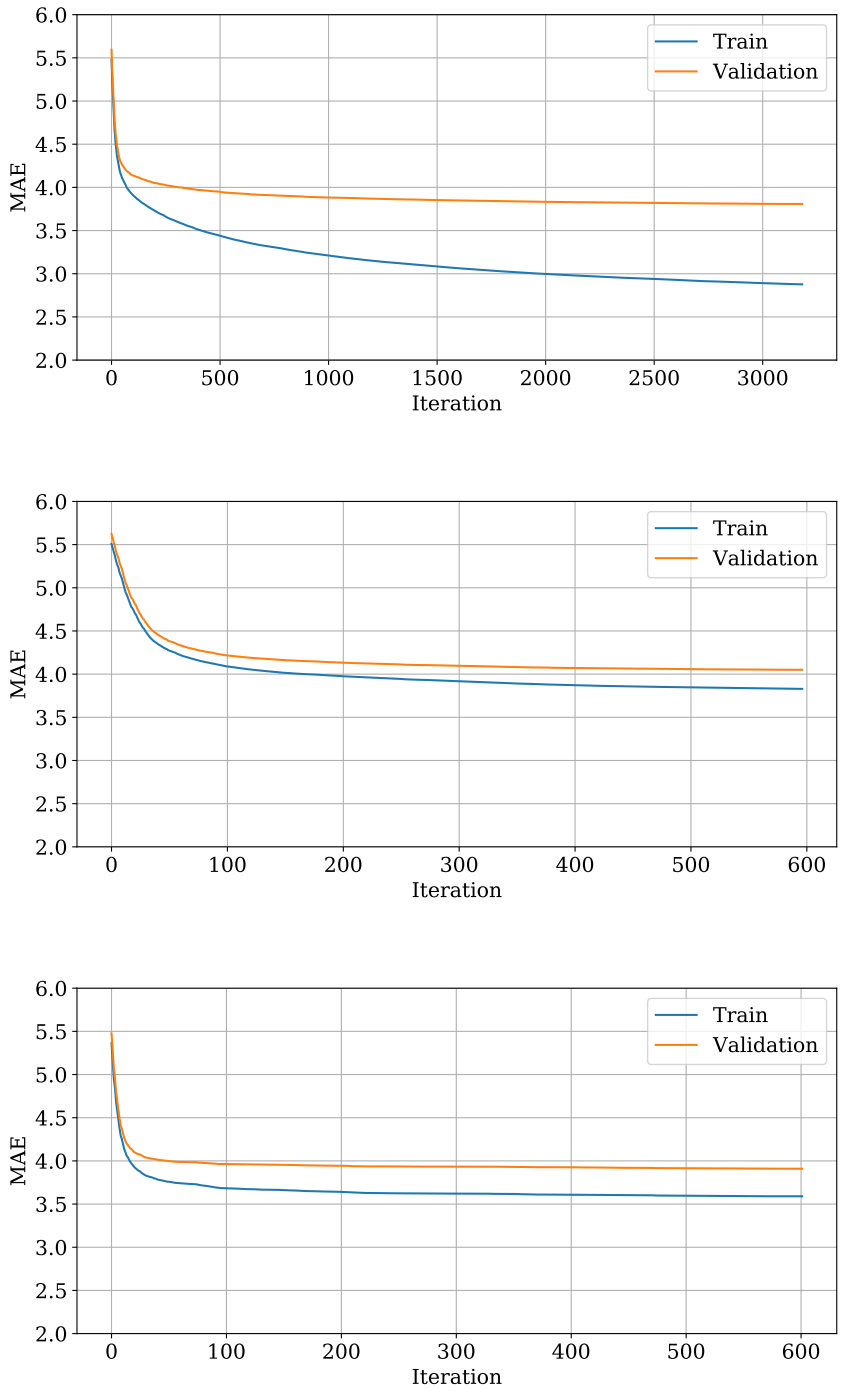
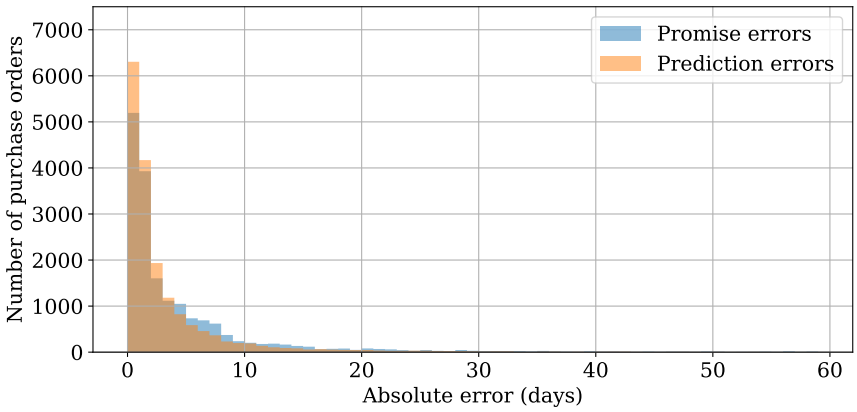
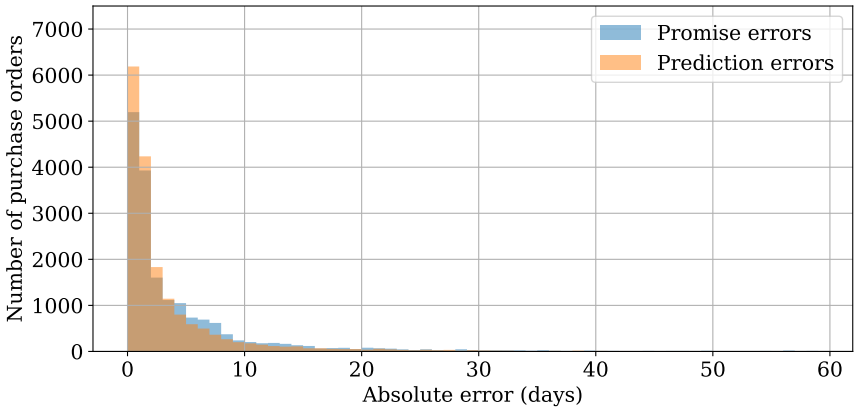


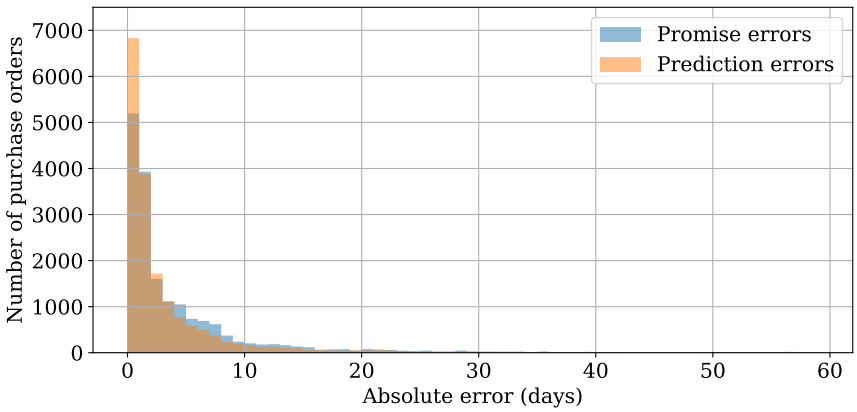
Figure 17. The curves depicting the train and validation errors during training the selected model: (a) Exp. 1; (b) Exp. 2; (c) Exp. 3.



(a)



(b)



(c)

Figure 18. Histograms of absolute prediction errors of each experiment and the errors of the promises made by the suppliers for the test set. For clarity, only the absolute errors less than 60 days are included, excluding 1.6 % and 0.6 % of the promise and prediction errors, respectively: (a) Exp. 1; (b) Exp. 2; (c) Exp. 3.

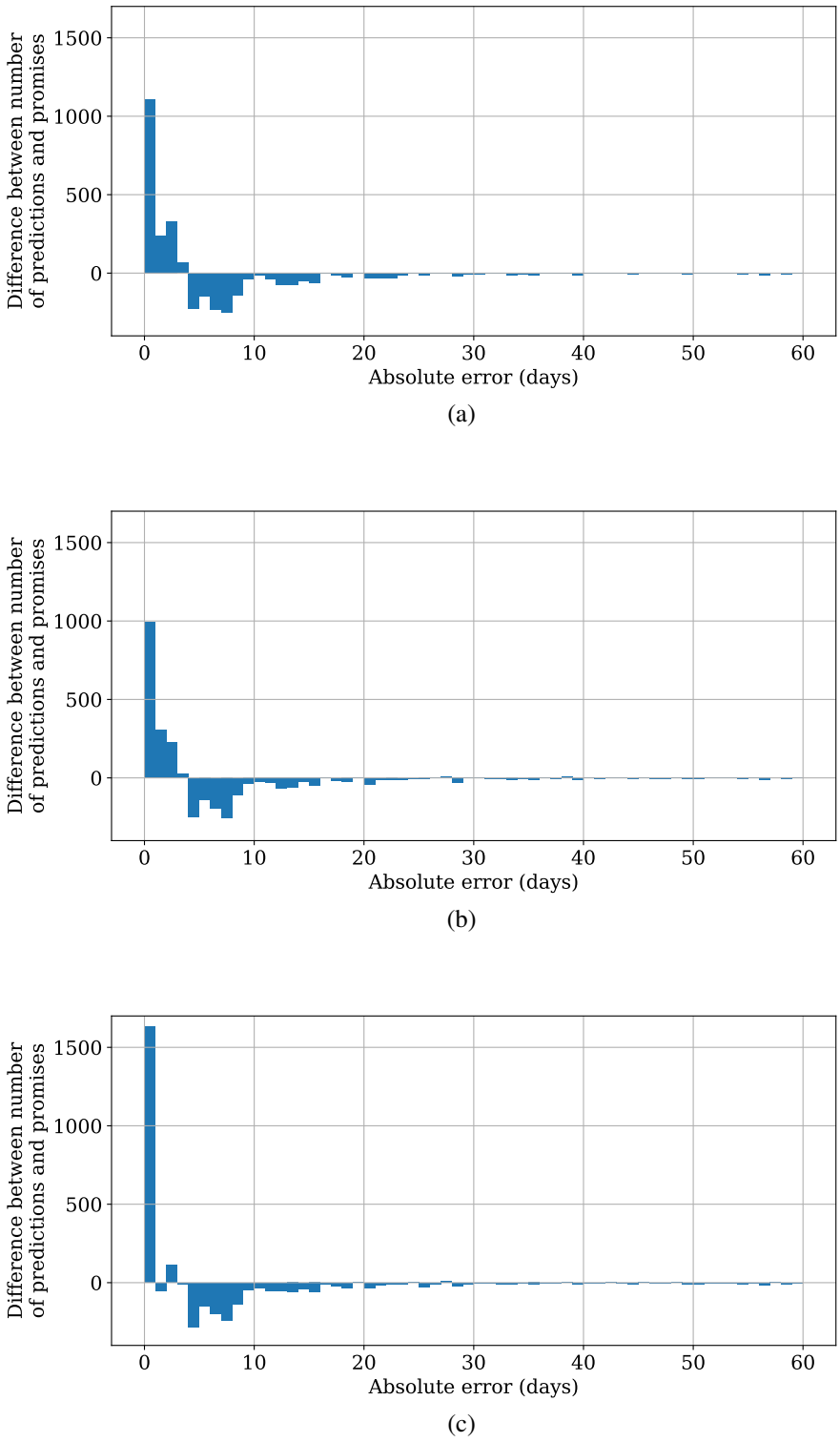


Figure 19. Difference between the histograms of absolute prediction errors and the errors of the promises made by the suppliers for the test set. For clarity, only the absolute errors less than 60 days are included, excluding 1.6 % and 0.6 % of the promise and prediction errors, respectively: (a) Exp. 1; (b) Exp. 2; (c) Exp. 3.

accurate. The accuracy came in the cost of prediction time, which was ten times longer for Exp. 1 compared to the others. Exp. 3 produced more accurate predictions than Exp. 2, even though its prediction time was slightly shorter.

Table 9. Error metrics for the promises made by suppliers and the predictions made by the models of each experiment for the test set. For the experiments, the mean and standard deviation of prediction time for 10 runs is shown.

	MAE	MSE	R^2 (%)	Prediction time (s)
Supplier estimates	5.64	172.6	89.09	
Exp. 1 predictions	3.74	130.8	91.73	4.48 ± 0.06
Exp. 2 predictions	3.97	140.9	91.09	0.48 ± 0.01
Exp. 3 predictions	3.84	138.0	91.28	0.44 ± 0.02

5 DISCUSSION

5.1 Current study

The aim of this thesis was to find a suitable method for predicting the lead times of purchase orders of a company. The method used to obtain the predictions is GBM, and its hyper-parameters were optimised via TPE.

Only a few studies related to this specific problem were found, and the methods used in them are different from each other. The methods include a binomial regression model [4], a hybrid model composed of stepwise regression and matrix gamma distribution fitting [2], RFs and quantile random forests [3]. Banerjee et al. [2] separated the data samples into groups based on suppliers and fit a distinct model for each group. Liu et al. [3] included categorical features and apply PCA to reduce the number of categories. Even though Banerjee et al. and Liu et al. both used purchase orders of aircraft engine parts and some of the authors in the studies are the same, they used different datasets, and their results were not compared. Therefore it is unclear whether the results are comparable to each other. The only study that compared the predictions to the estimates made by suppliers is the one by Liu et al. Their predictions surpassed the supplier estimates by some margin. Because of the promising results, small number of studies, and differing methods, it seems that the task at hand needs more research to find the most suitable methods. Providing more accurate estimates of the lead times can make the processes of many companies more efficient, leading to optimised utilisation of resources, improved customer satisfaction and overall performance.

Because the ensemble of trees as a model structure has been proven viable for solving the task, it was logical to study GBM. In theory, it should provide better results with fewer trees than RF, because it builds the trees sequentially instead of at random. Also, an implementation called LightGBM [9] can handle categorical data without expanding them to dummy variables by using a splitting rule designed for categorical data [38]. This is a significant benefit compared to the lossy dimensionality reduction method by Liu et al. The primary deficiencies of GBM compared to RF are the larger number of hyper-parameters and the higher sensitivity to their values. An optimisation method called TPE was used to obtain the optimal hyper-parameters. Three different objective functions were used, and the results of them were compared. Quantitative feature selection was performed simultaneously with hyper-parameter optimisation via TPE. The three objective functions, O_1, O_2 and O_3, were implemented for Exp. 1, 2 and 3, respectively.

The selected hyper-parameters, as well as features, differed widely between the three experiments. This indicates that the pre-selected features correlate with each other, and there exist multiple combinations of hyper-parameters that produce more accurate predictions than the suppliers' estimates. It was not a surprise that *m_gr_process_time* was selected as the most important feature in every experiment, since it was the numeric feature with the largest correlation with the target variable, as shown in Figure 11 on page 31. This implies that the internal processes of the company are a major factor in the delays of the purchase orders.

The reason for experimenting with two additional objective functions instead of using just the most simple one, O_1, is that it produced overly complex models, as can be seen from Figures 12 and 17 on pages 37 and 45, as well as Table 8 on page 44. The goal of reducing the complexity was achieved, reducing the training time to one tenth of the original time, as well as bringing the training and validation scores closer to each other. However, it was achieved with the cost of accuracy, as can be seen from Table 9 on page 48. It depends on the application whether the reduced complexity is worth the decrease in accuracy.

The original goal of Exp. 3 was to produce a model that is more general than in Exp. 2. Figure 17 on page 45 shows that the errors during training converged faster, but the difference between the training and validation error was larger than in the one produced by O_2. The distribution of errors was significantly different in Exp. 3 compared to the other two, as is shown in Figure 18 on page 46. There were one and a half times more errors of zero in Exp. 3, and almost zero errors of two and three days. Table 9 on page 48 shows that Exp. 3 produced smaller errors with slightly shorter prediction times than Exp. 2. However, the differences between Exp. 2 and 3 are so minor that it is not clear if they are caused by the different cost function or merely a symptom of the randomness in the hyper-parameter optimisation process.

The experiments made in this study cannot be directly compared to those of Liu et al., because of a different domain, different dataset and different features. As can be seen when comparing Figures 8 and 19 on pages 17 and 47, the predictions made by the suppliers in Liu et al.'s studies are more inaccurate than those in the dataset of this study. The fact that they did not provide the supplier errors in a numerical form makes quantitative comparison impossible. However, some qualitative comparisons can be made.

A major difference between the methods proposed here and those of Liu et al. is that they have multiple steps where the person applying the method has to make choices dependent on the dataset. The choices include the number of categories to keep before and after

PCA, the number of principal components to consider, as well as the hyper-parameters of RF. Additionally, the method needs tweaking if the features available are not the same as in their study. This is often the case as the data collected can vary between companies in the real world. An example is the dataset provided for this study, which had many additional features compared to Liu et al.'s data but lacked the *ABC Indicator*, that was used in their study. The methods proposed in this study do not need any arbitrary choices made by a human since the automated hyper-parameter optimisation attempts to find the optimal values. The only choice left for the user is to select the objective function based on the preference between complexity and accuracy. The feature selection is also mostly automated, so the user can experiment with different features and see if they are essential, based on if they get selected by TPE. These attributes make the methods applicable across domains and datasets with minimal effort needed by the user.

5.2 Future work

There might exist an objective function that provides a better balance between accuracy and complexity than the ones used in this study. Different objective functions should be experimented with. The methods should also be tested with different datasets from different domains to confirm the assumption of the methods' generality. Further experimentation with the same dataset would also be beneficial in determining if the differences between Exp. 2 and 3 were caused by the cost functions or the randomness of the hyper-parameter optimisation.

The predictions made by the models can vary by a significant margin, and there is no information about the reliability of a prediction. Therefore, it would be beneficial to make prediction intervals, that would, for example, predict that there is a 95 % probability of a lead time being between two values. This way, the uncertainty of the point predictions could be evaluated, and further analysis could be performed to conclude the reasons behind the inaccuracies. A possible method would be to train two separate GBMs using quantile regression as a cost function. The hyper-parameters could be optimised via TPE using the cost function defined by Pearce et al. [47].

6 CONCLUSION

The objective of this study was to study methods for predicting the delivery times of purchase orders for a company by enhancing the predictions made by the suppliers. The goal was achieved by using GBM as predictive model. The hyper-parameter optimisation and quantitative feature selection were implemented via TPE. Three different objective functions for TPE were compared.

All of the models with different objective functions produced an enhancement in the predictions, with one of them producing more accurate results with a more complex model than the other two. The methods used in this study need minimal input from the user and can be applied across domains and datasets.

REFERENCES

[1] S. A. Rehman Khan and Z. Yu, "Introduction to supply chain management", in *Strategic Supply Chain Management*, ser. EAI/Springer Innovations in Communication and Computing, Springer International Publishing, 2019, pp. 1–22.

[2] A. G. Banerjee, W. Yund, D. Yang, P. Koudal, J. Carbone, and J. Salvo, "A Hybrid Statistical Method for Accurate Prediction of Supplier Delivery Times of Aircraft Engine Parts", in *Proceedings of the ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 1B, American Society of Mechanical Engineers, 2015.

[3] J. Liu, S. Hwang, W. Yund, L. N. Boyle, and A. G. Banerjee, "Predicting Purchase Orders Delivery Times Using Regression Models With Dimension Reduction", in *Proceedings of the ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 1B, American Society of Mechanical Engineers, 2018.

[4] L. Walls, J. Quigley, M. Parsa, and E. Comrie, "Risk analysis of supply: Comparative performance and short-term prediction", in *Proceedings of the 26th European Safety and Reliability Conference, ESREL 2016*, CRC Press, 2016, pp. 347–354.

[5] R. Polikar, "Ensemble learning", in *Ensemble Machine Learning: Methods and Applications*, Springer US, 2012, pp. 1–34.

[6] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests", in *Ensemble Machine Learning: Methods and Applications*, Springer US, 2012, pp. 157–175.

[7] M. Kubat, "Decision trees", in *An Introduction to Machine Learning*, Springer International Publishing, 2017, pp. 113–135.

[8] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine", *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[9] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree", in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017, pp. 3146–3154.

[10] D. F. Ross, "Introduction to supply chain management", in *Distribution Planning and Control: Managing in the Era of Supply Chain Management*, Springer US, 2015, pp. 3–43.

[11] H. Zijm, M. Klumpp, S. Heragu, and A. Regattieri, ”Operations, logistics and supply chain management: Definitions and objectives”, in *Operations, Logistics and Supply Chain Management*, ser. Lecture Notes in Logistics, Springer International Publishing, 2019, pp. 27–42.

[12] D. Ni, Z. Xiao, and M. K. Lim, ”A systematic review of the research trends of machine learning in supply chain management”, *International Journal of Machine Learning and Cybernetics*, 2019.

[13] H. Bousqaoui, S. Achchab, and K. Tikito, ”Machine learning applications in supply chains: An emphasis on neural network applications”, in *3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*, IEEE, 2017, pp. 1–7.

[14] E. Alpaydin, *Introduction to machine learning*, 2nd ed, ser. Adaptive computation and machine learning. MIT Press, 2010, 537 pp.

[15] W.-M. Lee, ”Introduction to machine learning”, in *Python® Machine Learning*, John Wiley & Sons, Inc., 2019, pp. 1–18.

[16] N. Meinshausen, ”Quantile Regression Forests”, *Journal of Machine Learning Research*, vol. 7, pp. 983–999, Jun 2006.

[17] A. Natekin and A. Knoll, ”Gradient boosting machines, a tutorial”, *Frontiers in Neurorobotics*, vol. 7, 2013.

[18] M. Krzywinski and N. Altman, ”Classification and regression trees”, *Nature Methods*, vol. 14, no. 8, pp. 757–758, 2017.

[19] L. Breiman, *Classification and Regression Trees*. Routledge, 2017.

[20] R. A. Berk, ”Classification and regression trees (CART)”, in *Statistical Learning from a Regression Perspective*, ser. Springer Texts in Statistics, Springer International Publishing, 2016, pp. 129–186.

[21] H. Shi, ”Best-first decision tree learning”, Thesis, The University of Waikato, 2007.

[22] R. Punmiya and S. Choe, ”Energy Theft Detection Using Gradient Boosting Theft Detector With Feature Engineering-Based Preprocessing”, *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 2326–2329, 2019.

[23] S. Touzani, J. Granderson, and S. Fernandes, ”Gradient boosting machine for modeling the energy consumption of commercial buildings”, *Energy and Buildings*, vol. 158, pp. 1533–1543, 2018.

[24] X. Chen, L. Huang, D. Xie, and Q. Zhao, ”EGBMMDA: Extreme gradient boosting machine for MiRNA-disease association prediction”, *Cell Death & Disease*, vol. 9, no. 1, pp. 1–16, 2018.

[25] C. Yan, F.-X. Wu, J. Wang, and G. Duan, "PESM: predicting the essentiality of miRNAs based on gradient boosting machines and sequences", *BMC Bioinformatics*, vol. 21, no. 1, p. 111, 2020.

[26] H. Lu, H. Wang, and S. W. Yoon, "A dynamic gradient boosting machine using genetic optimizer for practical breast cancer prognosis", *Expert Systems with Applications*, vol. 116, pp. 340–350, 2019.

[27] L. Pei, Z. Sun, T. Yu, W. Li, X. Hao, Y. Hu, and C. Yang, "Pavement aggregate shape classification based on extreme gradient boosting", *Construction and Building Materials*, vol. 256, p. 119 356, 2020.

[28] J. Zhou, E. Li, S. Yang, M. Wang, X. Shi, S. Yao, and H. S. Mitri, "Slope stability prediction for circular mode failure using gradient boosting machine approach based on an updated database of case histories", *Safety Science*, vol. 118, pp. 505–518, 2019.

[29] Z. Wei, Y. Meng, W. Zhang, J. Peng, and L. Meng, "Downscaling SMAP soil moisture estimation with gradient boosting decision tree regression over the tibetan plateau", *Remote Sensing of Environment*, vol. 225, pp. 30–44, 2019.

[30] K. F. Hew, X. Hu, C. Qiao, and Y. Tang, "What predicts student satisfaction with MOOCs: A gradient boosting trees supervised machine learning and sentiment analysis approach", *Computers & Education*, vol. 145, p. 103 724, 2020.

[31] F. Climent, A. Momparler, and P. Carmona, "Anticipating bank distress in the eurozone: An extreme gradient boosting approach", *Journal of Business Research*, vol. 101, pp. 885–896, 2019.

[32] T. Parr and J. Howard. (2018). How to explain gradient boosting, [Online]. Available: `http://explained.ai/gradient-boosting/index.html` (visited on 11/05/2020).

[33] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean, "Boosting Algorithms as Gradient Descent", in *Advances in Neural Information Processing Systems 12*, MIT Press, 2000, pp. 512–518.

[34] L. Breiman, "Prediction Games and Arcing Algorithms", *Neural Computation*, vol. 11, no. 7, pp. 1493–1517, 1999.

[35] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system", in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, ACM Press, 2016, pp. 785–794.

[36] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features", in *Advances in Neural Information Processing Systems 31*, Curran Associates, Inc., 2018, pp. 6638–6648.

[37] Microsoft Corporation. (2020). Microsoft/LightGBM, GitHub, [Online]. Available: https://github.com/microsoft/LightGBM (visited on 05/04/2020).

[38] W. D. Fisher, "On Grouping for Maximum Homogeneity", *Journal of the American Statistical Association*, vol. 53, no. 284, pp. 789–798, 1958.

[39] Microsoft Corporation. (2020). Parameters — LightGBM 2.3.2 documentation, [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Parameters.html (visited on 03/30/2020).

[40] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-Generation Hyperparameter Optimization Framework", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, Association for Computing Machinery, 2019, pp. 2623–2631.

[41] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization", in *Advances in Neural Information Processing Systems 24*, Curran Associates, Inc., 2011, pp. 2546–2554.

[42] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces", *Journal of Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[43] Optuna Contributors. (2020). Optuna: A hyperparameter optimization framework — Optuna 2.3.0 documentation, [Online]. Available: https://optuna.readthedocs.io/en/v2.3.0/ (visited on 11/04/2020).

[44] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, "From local explanations to global understanding with explainable AI for trees", *Nature Machine Intelligence*, vol. 2, no. 1, pp. 56–67, 2020.

[45] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions", in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017, pp. 4765–4774.

[46] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization", *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[47] T. Pearce, M. Zaki, A. Brintrup, and A. Neely, "High-Quality Prediction Intervals for Deep Learning: A Distribution-Free, Ensembled Approach", *arXiv:1802.07167 [stat]*, 2018. arXiv: 1802.07167.

Appendix 1. SHAP values

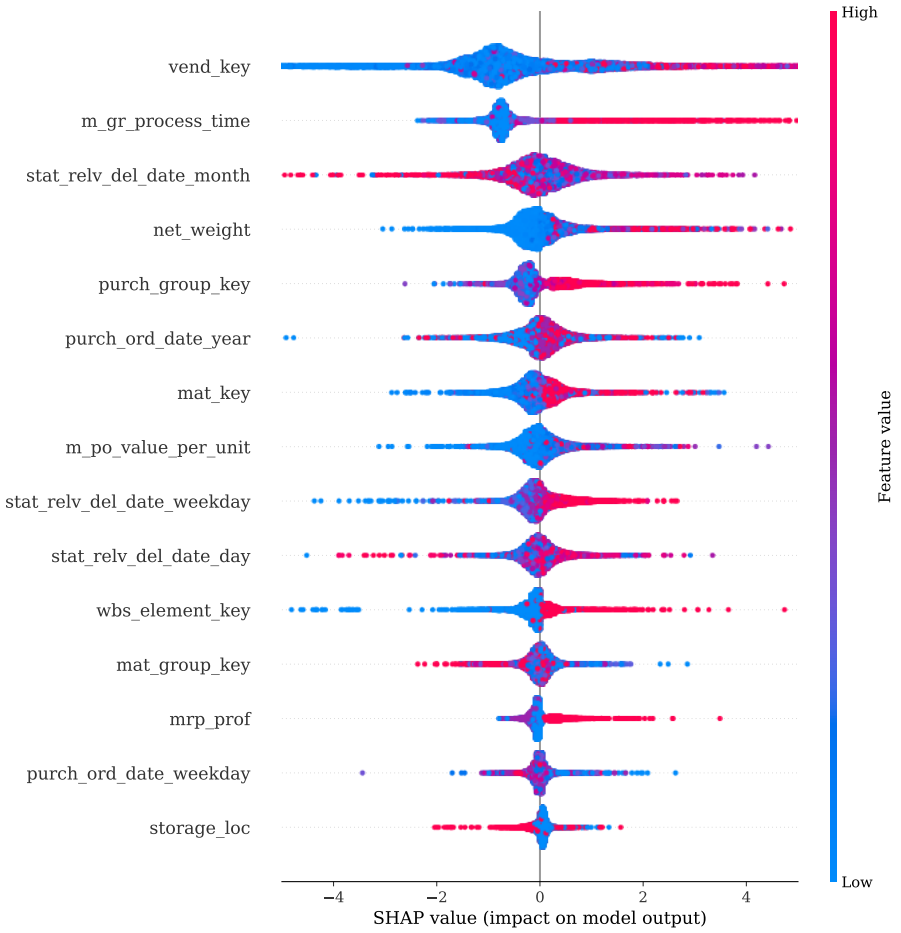


Figure A1.1. SHAP values of the features selected for the final model in Exp. 1, sorted based on the importance of the features. The SHAP values were calculated for the test set. Each point represents one sample. For each sample, position of the point on the horizontal axis tells the direction and amplitude of the impact that the feature had on the model output. The color represents the value of the feature in the sample, with blue being the lowest and red the highest value. Note that for the categorical features, the value is irrelevant. For clarity, the horizontal axis is limited from -5 to 5, excluding 5.3 % of the points.

(continues)

Appendix 1. (continued)

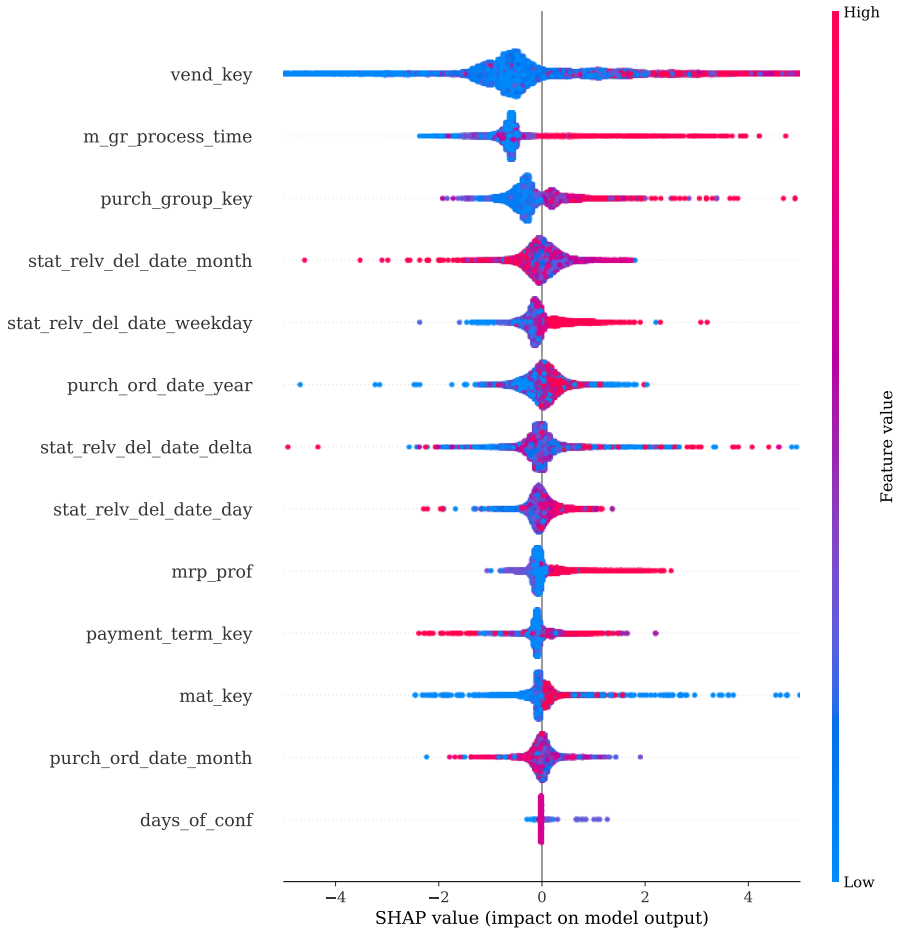


Figure A1.2. SHAP values of the features selected for the final model in Exp. 2, sorted based on the importance of the features. The SHAP values were calculated for the test set. Each point represents one sample. For each sample, position of the point on the horizontal axis tells the direction and amplitude of the impact that the feature had on the model output. The color represents the value of the feature in the sample, with blue being the lowest and red the highest value. Note that for the categorical features, the value is irrelevant. For clarity, the horizontal axis is limited from -5 to 5, excluding 4.4 % of the points.

(continues)

Appendix 1. (continued)

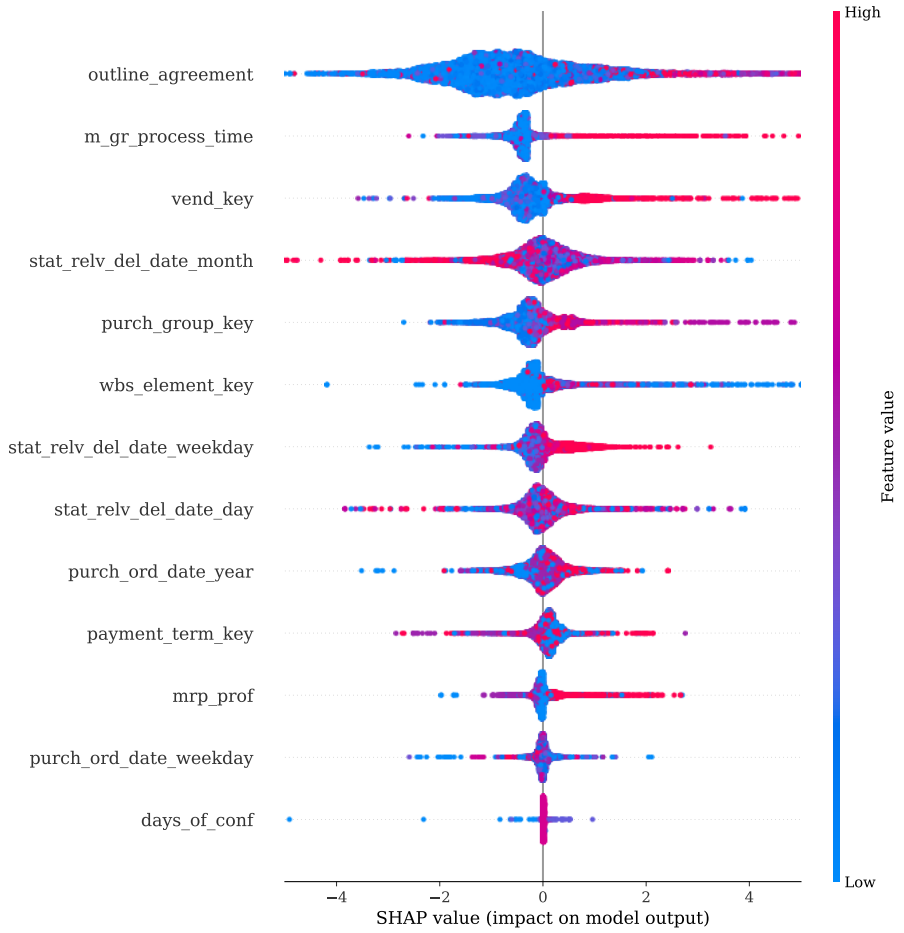


Figure A1.3. SHAP values of the features selected for the final model in Exp. 3, sorted based on the importance of the features. The SHAP values were calculated for the test set. Each point represents one sample. For each sample, position of the point on the horizontal axis tells the direction and amplitude of the impact that the feature had on the model output. The color represents the value of the feature in the sample, with blue being the lowest and red the highest value. Note that for the categorical features, the value is irrelevant. For clarity, the horizontal axis is limited from -5 to 5, excluding 4.6 % of the points.