Technische Universität Braunschweig

Bachelorarbeit

Comparison of supervised learning algorithms for arrival time estimation in meal delivery

an der Technischen Universität
Carolo-Wilhelmina zu Braunschweig
Carl-Friedrich-Gauß-Fakultät
Institut für Wirtschaftsinformatik, Abteilung Decision Support.

Decision Support

Eingereicht von:	Emre Gezer
Matrikelnummer:	4901507
Studiengang:	Wirtschaftsinformatik
Referent:	Jun.-Prof. Dr. Marlin Ulmer
Betreuer:	M. Sc. Florentin D. Hildebrandt
Eingereicht:	Braunschweig, den 10.02.2021

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Braunschweig, 10.02.2021

_____

Contents

List of Figures

1 Introduction

Nobody likes waiting - be it when you order your next book on Amazon for class or the pizzas at UberEATS for your birthday. In fact, when doing research on this topic, psychologists found out that increased waiting times generally have a significant negative impact on customer satisfaction and loyalty [1,2]. If we take a glance at the online food delivery market, there are more than 700 million people globally that used food delivery services in 2017 with twice as many users being expected in 2024 [3]. According to forecasts for the same time frame, the userbase of eCommerce platforms in general will grow from 2.480 billion to roughly 4.6 billion users [4]. Thus, waiting becomes a large scaled economic issue. What one might not expect is that the customers' own perceptions regarding their waiting time negatively affect the perceived service quality stronger than actual waiting times do [5,6]. Given these points, we can conclude that a key challenge lies in the communication of accurate arrival time estimations to customers.

Current research tackles this task commonly by means of machine learning techniques. Exemplary, while Hildebrandt and Ulmer used Gradient Boosting Decision Trees in their offline approach to map state features directly to expected arrival times in food delivery [7], Zhu et al. implemented a Multilayer Perceptron to predict accurate Order Fulfillment Cycle Times [8]. Hardly any kind of machine learning model comparison for arrival time estimation purposes in vehicle routing settings can be found. Therefore, we intend to conduct further research in this area.

This paper examines the forecast quality and performance of different supervised machine learning models for arrival time estimation problems (ETA) in meal delivery. To accomplish this task, the algorithms will be trained on historical meal delivery data collected within Iowa City.

This paper is organized as follows. Firstly, we will review and discuss literature that focuses on arrival time estimation via offline supervised learning. We will then proceed with our model, where we describe the ETA problem generally and define the model with respect to the underlying problem of our experimental design. Several steps are taken in the solution approach that follows upon our model definition.

In the beginning, we give an introduction to fundamental concepts in supervised learning and present techniques used at different stages in the machine learning pipeline generally for the sake of understanding the way each of them functions. We will then advance to the computational study, where we present our experimental design and discuss the results. Finally, we will create variations of our experimental design in order to analyze each algorithm's robustness and impact in these different settings.

2 Literature Review

This chapter gives an overview of research related to this work. The goal of this review is to motivate our work by showing that researchers indeed use different offline supervised learning approaches for quite similar problems. We first present research that is related to our work the most, namely papers in the field of arrival time estimation for meal delivery. Then, we broaden the scope to research in arrival time estimation for vehicle routes in general.

2.1 Most related work

This section includes research on offline arrival time estimation via supervised learning in dynamic pick-up and delivery settings. To the best of the authors knowledge, there are only three papers that fit this description. Amongst them, the most closely related work to this paper is that of Hildebrandt and Ulmer (2020), who contributed a offline supervised learning approach to predict arrival times for the Restaurant Meal Delivery problem, a dynamic pick up and delivery problem with uncertainty in travel times, processing times and requests originally presented in M. Ulmer, Thomas, Campbell, and Woyak (2020). In their offline approach, Hildebrandt and Ulmer (2020) map spatial, temporal, routing, and processing features based on the RMDPEAT to expected arrival times by means of a gradient boosting decision tree (GBDT) model. This paper is inspired by them and can be seen as complementary to their paper since we aim to estimate arrival times offline based on the same underlying problem setting via several supervised learning algorithms, including GBDTs.

Zhu et al. (2020) predict arrival times by means of deep learning with uncertainty being present in requests, courier travel times, courier waiting times at restaurants and cooking times. Besides using temporal, spatial and processing features for travel time prediction, they additionally include dish specific features and information about weather conditions. In contrast to Hildebrandt and Ulmer (2020), they include no routing information. They instead introduce a separate component that ranks courier assignments w.r.t. logistics cost and customer inconvenience. According to their

analysis, the proposed deep learning architecture produces, inter alia, more accurate results than a GBDT approach.

Liu, He, and Max Shen (2018) compare linear models, support vector regression and ensemble learning methods for travel time estimation based on spatial, temporal and order-related features, and integrate travel time prediction into the order assignment problem with uncertainty in requests, travel times and service times. The order assignment problem aims to assign orders in a way that the assignments minimize the total delivery delay over all driver routes. Analyzing the prediction models w.r.t to their accuracy, feasibility and interpretability, they found that random forests (RF) and support vector regression yield slightly more accurate results but are computationally less feasible due to exponential runtime and less interpretable than linear models. For the latter two reasons, Liu et al. (2018) prefer linear models. Amongst the linear models, lasso regression obtained the best accuracy.

2.2 Arrival Time Estimation

In this section, we broaden the scope from offline arrival time estimations via supervised learning for dynamic pick-up and delivery problems to offline arrival time estimations via supervised learning for vehicle routes in general. Tab. 1 classifies the literature on arrival time estimation for vehicle routes with regards to the problem setting and the solution. With *Route type*, the table distinguishes arrival time estimation research that has been applied to vehicle route sequences consisting of single origin-destination (OD) pairs (referred to as *OD* in the table) from those that have been applied to route sequences consisting of multiple OD pairs (referred to as *trips* in the table) each. By *Uncertainty*, the table refers to uncertain elements in the underlying problem settings. Sources of Uncertainty considered here are requests, travel and service times, and processing times. Uncertainty in requests indicates that customer requests are not certainly known at the start of the problem and arrive dynamically over time. Uncertainty in travel and service times expresses itself through uncertain weather conditions, traffic congestion or individual challenges when serving customers (e.g. parking or waiting times). Uncertainty in processing occurs when two stochastic processes are synchronized (e.g. the synchronization of bus ride and bus boarding, or meal preparation and delivery). From the solution view, we classify the literature based on features and supervised learning algorithms used for travel time prediction. The *features* column distinguishes between temporal,

Literature & Application	**Routing**	**Uncertainty**	**Features**				**Offline Approaches**
			Temporal	Spatial	Routing	Processing	
Hildebrandt et al. (2020) (Restaurant Meal Delivery)	Trips	Requests, Travel/Service times, Processing Times	Y	Y	Y	Y	Ensemble Learning (GBDT)
Zhu et al. (2020) (Order Fulfillment Cycle Time Prediction)	Trips	Requests, Travel/Service times, Processing times	Y	Y		Y	Deep Learning (ANN)
Liu et al. (2018) (Restaurant Meal Delivery)	Trips	Requests, Travel/Service times	Y	Y			Linear models, SVR, Ensemble Learning (RF)
Vanajakshi et al. (2007) (Traffic Information Systems)	OD	Travel/Service times	Y				Support Vector Regression, Deep Learning (ANN)
Siripanporchana et al. (2016) (Traffic Information Systems)	OD	Travel/Service times	Y				Deep Learning (ANN)
Jindal et al. (2017) (Taxi travel time prediction)	OD	Travel/Service times	Y	Y			Deep Learning (ANN)
Huang et al. (2018) (Taxi travel time prediction)	OD	Travel/Service times	Y	Y			Ensemble Learning (GBDT)
Cheng (2019) (Traffic Information Systems)	OD	Travel/Service times	Y	Y			Ensemble learning (GBDT)
Huang et al. (2020) (Taxi travel time prediction)	OD	Travel/Service times	Y	Y			Ensemble Learning (GBDT)
This paper (Restaurant Meal Delivery)	**Trips**	**Requests, Travel/Service Times, Processing Times**	**Y**	**Y**	**Y**	**Y**	**Ensemble Learning (GBDT & RF), Linear Models**

Figure 2.1: Classification of Literature on Arrival Time Estimations with Supervised Learning

spatial, routing and processing features. Temporal and spatial features include time and space related variables respectively. Examples for former are time stamps at a start/end point or historical travel times, and examples for latter are GPS coordinates or distances between locations. Processing and routing features give information about the uncertainty in processing times (e.g. meal preparation or bus boarding time) and the properties of a trip (e.g. number of stops in a trip) respectively. The column *Offline Approaches* presents all offline supervised learning approaches used in the respective literature, regardless of whether they were used as the primary approach to predict arrival times or solely for comparison purposes.

Significant research where arrival times are estimated via supervised learning was conducted for origin-destination problems in different subfields of intelligent transportation systems. In the following, we will show how researchers used different supervised learning approaches for problem settings of the same field.

To predict travel times on freeways for different short-term forecasting horizons, Vanajakshi and Rilett (2007) use support vector regression (SVR) based on estimated route travel times from prior research and conclude that SVR performs comparably well to artificial neural networks (ANN). Siripanpornchana, Panichpapiboon, and Chaovalit (2016) propose a deep learning architecture consisting of a deep belief network and a sigmoid regression layer. Former learns features in an unsupervised fashion based on historical route travel times as inputs, latter then estimates travel times based on these learned features. Cheng, Li, and Chen (2019) make use of GBDTs using manually selected travel time features and traffic state related variables. They report that the ensemble learning approach with GBDTs outperforms feed-forward neural networks and support vector machines.

For taxi travel time prediction, Jindal, Chen, Nokleby, Ye, et al. (2017) propose a unified approach based on raw NYC taxi data. They concatenate two neural networks, where the first one uses spatial features to predict travel distances, and the second one uses these predicted distances and additional temporal information to predict travel times. They solely compared their approach to other deep learning architectures. In contrast to them, Huang and Xu (2018) and Huang, Pouls, Meyer, and Pauly (2020) compare several tree-based learning methods to predict travel times on different horizons each based on NYC taxi data as well, among them random forests (both), GBDTs (both), and CART (only Huang et al. (2020)). While Huang and Xu (2018) selected features by means of principal component analysis, Huang et al. (2020) engineered them manually. Both ended up using spatial and temporal

features mainly. Their results indicate that all tree-based ensemble methods are able to predict travel times more accurately than the respective benchmark algorithms (CART and naive approach in Huang et al. (2020); linear and logistic regression in Huang and Xu (2018)).

2.3 Summary

In this chapter, we presented related research in the field of arrival time estimation. The goal of this review was to demonstrate how current research approaches arrival time estimations via supervised learning for quite similar problems differently. Generally, deep learning and tree-based ensemble learning were the most frequently used approaches for the arrival time estimation problems presented in our review. In the field of meal delivery, the limited amount of research involves different approaches as well: Tree-based ensemble learning in Hildebrandt and Ulmer (2020), the use of linear models in Liu et al. (2018), and deep learning in Zhu et al. (2020). Therefore, we intend to analyze different models from different viewpoints.

3 Problem formulation

This chapter gives an overview of the classical RMDP from M. Ulmer et al. (2020) and the RMDPEAT introduced in Hildebrandt and Ulmer (2020). Here, we focus on showing the differences between them and emphasize the need for the incorporation of arrival time estimation in dynamic vehicle routing. Then, we give a formal, but concise description of the problem of estimating arrival times for delivery routing (EAT) introduced in Hildebrandt and Ulmer (2020) as well.

3.1 Restaurant Meal Delivery Problem

According to the taxonomy for dynamic vehicle routing problems introduced in Psaraftis, Wen, and Kontovas (2016), the classical RMDP belongs to the category of (1) dynamic and (2) stochastic problems since (1) drivers routes need to be reoptimized over time by means of a routing policy and (2) only the probability distributions of inputs are known at the start of the problem. In the RMDP, customers are distributed across a known service area and request service during a finite service time window. Drivers are distributed in the service area and dispatched by the meal delivery platform that provides meal delivery service for participating restaurants.

The RMDP process is triggered with the arrival of customer requests. When a customer requests service on the platform of the meal delivery service, the assignment policy proposed in M. Ulmer et al. (2020) assigns the customer an available driver. When this is done, the dispatcher updates the routes of the drivers. It is therefore assumed that the routing policy is exogenously given. The assigned driver then drives to the restaurant, picks the meal up and eventually waits at the restaurant if meals are not ready yet, and then delivers it to the customer. This whole process is impacted by uncertainty in the arrival of customer requests, in pick-up and delivery times of drivers, and in meal preparation times of restaurants. The RMDP allows order bundling. Order bundling refers to the possibility of assigning a vehicle more than one order at a time which results in greater flexibility for assignment decisions.

In addition, to potentially bundle orders better, the RMDP allows for postponement in order assignment. Overall, the RMDP seeks maximization in the amount of served customers and minimization in delivery delays.

M. Ulmer et al. (2020) formulated the RMDP formally by means of the route-based markov decision process (MDP) notation introduced in M. W. Ulmer, Goodson, Mattfeld, and Thomas (2017), which will aid in understanding fundamental differences between the RMDPEAT and the classical RMDP. For the full formulation of the RMDP as a route-based MDP, the reader is referred to M. Ulmer and Thomas (2018). We begin by describing basic terminology in route-based MDP notation for the RMDP:

- **Decision epochs**: A decision epoch k is a point of time which triggers a decision x. In the RMDP, decision epochs occur when a customer requests delivery or an order gets postponed since orders must be assigned and drivers routes must be updated in both cases.

- **Decision states**: Decision states contain any informations needed to make a decision at their corresponding decision epoch. In the RMDP, a decision state S_k contains the set of requested orders $\mathcal{D}_k$ including the new order D_k, and the set of planned routes Θ_k at decision epoch k. With planned routes, the RMDP refers to potential routing decisions that can be made when a new decision is made.

- **Decisions and deterministic costs**: A decision in the RMDP is the assignment of unassigned orders to vehicles at the corresponding decision epoch. This leads to a transition from the pre-decision state to the post-decision state S_k^x. S_k^x is the consequence of a decision point being triggered and represents the state with the updated set of orders $\mathcal{D}_{\|}$ and new, updated planned routes Θ_k. Note that in post-decision state S_k^x, the decision still has to be carried out and hence is still not realized. Since planned arrival times are known, M. Ulmer and Thomas (2018) define the deterministic costs as the marginal change indelay, which in turn is the difference of currently planend delay for routes Θ_k at decision epoch k and updated planned delays for updated routes Θ_k^x associated with the post-decision state S_k^x as deterministic costs.

- **Transitions and Stochastic Costs**: A transition to the next decision epoch $k+1$ with the next pre-decision state $S_{k+1} = (S_k, x, \omega)$ is induced when the decision x following state S_k is realized. ω denotes exogenous, random information that is

realized during the transition from post-decision state S_x^k to the next pre-decision state S_{k+1} and is finally known when the subsequent decision epoch $k+1$ is triggered. In the RMDP, uncertain delivery times impacted by uncertainty in meal preparation times is considered as random information. Given the planned arrival times of the updated planned routes Θ_k^x in post-decision state S_k^x and the actual arrival times resulting from a realization of ω, M. Ulmer and Thomas (2018) define the stochastic costs as the realized delay over all customers served during the transition from k to $k+1$.

Now that we are familiar with the route-based MDP notation for the RMDP, we will point out how the RMDPEAT differs from the classical RMDP in the subsequent section.

3.2 Restaurant Meal Delivery Problem with Arrival Time Estimation

When a request comes in, a new decision point in the RMDP is triggered. Consequently, an assignment decision has to be made. After that, the RMDP assigns the request to the driver that is regarded as the optimal choice by the assignment policy. When this is done, the random information occuring during the transition is realized when the next decision epoch $k+1$ is triggered. However, customers' restaurant choices are not modeled in the RMDP process since random information in the RMDP only includes uncertainty in meal preparation and delivery. The RMDPEAT however additionally considers the customer's restaurant choice as random information and models it explicitly by assuming that customer preferences are composed of customers' static restaurant and dynamic arrival time preferences. Hence, the RMDPEAT accounts for the customer order process.

The RMDPEAT process is triggered when a customer requests delivery. This leads to the calculation of estimated times of arrival for each restaurant the customer can order from that are then communicated to the customer. Based on his restaurant and arrival time preferences, the customer decides either to order from a restaurant or to not order at all. When the customer decides not to order, the process is terminated. When the customer orders, the dispatcher informs the chosen restaurant and updates driver routes. The process terminates when the request has been served. M. Ulmer and Thomas (2018) assume that planned arrival times are assumptions made by

the dispatcher, whereas Hildebrandt and Ulmer (2020) explicitly account for the arrival time estimations communicated to the customer. As we already pointed out in chapter 1 of this paper, this is crucial as there is evidence of arrival delay having a negative impact on customer satisfaction.

The changes wrt. to the route-based MDP are the following:

- **Modified Decision States**: As described, the RMDP state space contains the set of orders that are yet to serve, and the set of planned routes. Additionally, the RMDPEAT state space is extended by the set of restaurants R_k and their respective workloads. Further, they extend the information available for routes by not only including sequences of pick up and delivery locations to be visited, which is just spatial information, but also incorporating temporal quantities for each sequences as well.

- **Modified Decisions**: In the RMDP, a decision is triggered if a customer requests service or if the assignment of a request is postponed. This is also the case for the RMDPEAT. However, decisions made in the RMDPEAT have two stages. As in the RMDP, the first stage of RMDPEAT updates the set of orders and the driver routes when a decision is triggered. In the second stage, the RMDPEAT decision estimates arrival times based on the current RMDPEAT state and the updated routes. These estimations are then communicated to the customer.

- **Modified Transitions**: As in the RMDP, when decision epoch k is triggered and the transition to epoch $k + 1$ is therefore finished, random information is realized. In the RMDPEAT however, the restaurant choice of the customer and stochastic parking and meal preparation times of the delivery actions happening during a transition are now additionally considered as random information.

Since the RMDPEAT is a composite of two problems, namely the RMDP which accounts for driver assignment and routing, and the problem of estimating arrival times for delivery routing (EAT), it seeks to maximize the expected number of served customers and communicate accurate arrival times to the customer. The latter is the focus of the EAT, which is covered in the next section in detail.

3.3 Estimating arrival times for delivery routing

Hildebrandt and Ulmer (2020) define the RMDPEAT as a dynamic decision process. Based on that, they further formulate the problem of estimating arrival times for

delivery routing (EAT), which is just the EAT part of the RMDPEAT process, formally and in detail as well. In the following, we recap their EAT formulation formally since we operate on the same underlying DVR problem setting and share a the same task in predicting arrival times, but keep our explanations concise. For a in-depth explanation of the EAT, the interested reader is referred to section 3.2 in Hildebrandt and Ulmer (2020).

Let c_k denote the k-th customer requesting service at time point t_k, $\mathcal{N}_k$ the set of customers that have requested service but have not been served yet, R_k the set of restaurants and their workloads, and Θ_k the set of planned routes for each driver at decision epoch k. The decision state for the RMDPEAT is therefore given as $S_k = (t_k, \mathcal{N}_k \cup c_k, \mathcal{R}_k, \Theta_k)$. Decisions are triggered when customers request service and happen in two stages: First, potential routing decisions Θ_{ik} for every restaurant $i = 1, \ldots, |\mathcal{R}_k|$ the customer can order from are computed. The union of the RMDPEAT state S_k with these potential routing decisions Θ_{ik} represents the EAT state $S_k^{EAT} := (S_k \cup \Theta_{ik})$. Based on S_k^{EAT}, we estimate arrival times for each potential routes Θ_{ik} respectively and denote them as $X(S_k^{EAT}) := (X_{ik})_{i=1,\ldots,|\mathcal{R}_k|}$. Customers then choose a prefered restaurant based on their individual preference functions $\Phi_k((X_{ik})_{i \in \{1,\ldots,R_k\}})$. The objective of the EAT is defined as

$$\min_{X(S_k^{EAT})(j) \in \mathbb{R}_+} \mathbb{E}_{S_k^{EAT}}(||A(S_k^{EAT})(j) - X(S_k^{EAT})(j)||_2^2). \tag{3.1}$$

Equation 3.1 seeks to minimize the expected deviance of the estimated arrival time $X(S_k^{EAT})(j)$ from the actual arrival time $A(S_k^{EAT})(j)$ where j represents the restaurant that the customer will chose. The deviance is measured with the mean squared error.

4 Methodology

This chapter presents our methodogolical approach and describes its elements in detail. Section 4.1 gives an overview of the methodological approach used to evaluate and compare the models. Section 4.2 gives an introduction to supervised learning and presents the different models considered in the comparison in detail. Section 4.3 focusses on feature selection techniques applied in our computational study.

4.1 Design of Experiments

The primary goal of this study is to find the most suitable model for the given arrival time estimation problem. To attain the best possible solution for this problem, it is necessary to analyze the presented mdoels from different points of view since it is not always clear which model does the best job in predicting arrival times as our literature review shows. While Zhu et al. (2020) use deep learning as a means of estimating arrival respectively delivery times for restaurant meal delivery, Hildebrandt and Ulmer (2020) turn to GBDTs on a quite similar problem setting, whereas Liu et al. (2018) conclude that linear models are the better choice overall. For that reason, we will conduct experiments that will allow us to analyze the behaviour of the considered models and their parameters.

Further challenge also lies in the selection of a well suited data model for the given problem which is highly non-trivial. Almost every related work shown in the literature review crafts features manually with the help of human domain expertise, and rightfully so: Raw data is usually characterized by high dimensionality and sparse information. We aim to find the right balance for the two conflicting objectives of minimizing the feature space and thus reducing the dimensionality on the one hand, while maximizing the informative value each considered feature contributes to the objective on the other hand. To demonstrate how models perform on different datasets and to find out which data model delivers more promising results, we decide to conduct our experiments upon the two data models presented in section 4.3, with one data model being high in dimensionality and the other one having a significantly

smaller feature space. In real-world scenarios, data can be corrupted by external influences, either caused by human or machine error. To see how those corruptions affect the model performances, we analyze supervised learning models also wrt. their robustness.

Our experimental pipeline is designed as follows:

1. **Finding Sufficient Sample Sizes:** The more data is available for training, the better the generalization will be. However, this conflicts with training times since there are more calculations to make when there is more data. Therefore, we seek a good balance for the tradeoff between sample size and accuracy. We do this by defining a number of training runs and assigning each run sample sizes from an interval with evenly spaced values. This experiment returns the sufficient sample size for each model, and additionally gives us information on how the models perform on solely manually set hyperparameters.

2. **Hyperparameter Analysis:** Tree-based ensembles are complex in their structure and can achieve great performance when their parameters are tuned properly. For that reason, we intend to conduct hyperparameter optimization (HPO) for GBDT and Random Forest. We will first specify the hyperparameters we consider to optimize and then specify their respective search space. The values from these evenly-spaced search space intervals are sampled via the **C**ovariance **M**atrix **A**daption **E**volution **S**trategy, or in short **CMA-ES**. *CMA-ES* follows a simple principle: The probability of samples from previously succesful optimization steps being drawn again is positively correlated to the contribution of those samples to the objective. For further information on *CMA-ES*, the reader is referred to Hansen (2016). To speed up the optimization process, we will use **hyperband pruning**, a bandit-based optimization approach presented in Li, Jamieson, DeSalvo, Rostamizadeh, and Talwalkar (2018) that prunes unpromising trials early. Having done the HPO, we now can detect which parameter configurations deliver the best results, and analyze the importances of the optimized hyperparameters for each GBDT and Random Forest optimization on both datasets via **fANOVA**, a hyperparameter evaluation algorithm presented in Hutter, Hoos, and Leyton-Brown (2014). *fANOVA* calculates feature importances by fitting a random forest regression model to the optimal parameter configuration that results of the HPO with which it aims to predict the corresponding objective value. The relative importances provide information about the parameter

variances. A higher value is associated with a higher variance, meaning that a change of the parameter setting will have a greater impact on the prediction performance compared to parameters of low relative importance. Thereby, we can assess which parameters impact the model significantly and which parameters are less significant, and especially how the importances differ for the two data sets. By that, we hope to get a fine-tuned model that maximizes prediction quality on one hand, and intuition for suitable parametrizations on the other hand.

3. **Noise Induction**: To test the robustness of our models, we induce noise on a selected group of temporal features present in both datasets. We further vary the levels of noise to see how an increase of noise impacts the model performances. The smaller a examined model varies in accuracy on different noise levels, the more robust it is. The noise we add to each feature is sampled from a normal gaussian distribution.

We will specify the configuration details for each experiment in the respective sections in chapter 5.

4.2 Algorithms

This section shortly introduces the statistical and conceptual framework of supervised learning, and then proceeds with the detailed examination of how each chosen algorithm included in our comparison, namely linear regression, and the tree-based ensembles, works. Since our comparison involves the conduction of several experiments that are quite time consuming due to limitations in hardware, we do not consider a deep learning approach in our comparison.

4.2.1 Framework of supervised learning

As inputs, supervised learning algorithms receive **training data** in form of a finite set $S = \{(x_2, y_2), (x_2, y_2), \ldots, (x_n, y_n)\}$ of N pairs from $\mathcal{X} \times \mathcal{Y}$ where x_i is a **sample** described by p features, and is associated to it's corresponding **label** y_i. They return a **hypothesis** $h : \mathcal{X} \to \mathcal{Y}$ that aims to predict the corresponding label $y \in Y$ for any $x \in X$, but especially for unseen samples or i.e. **test data** $x \notin S$. We further assume a joint probability distribution P over $\mathcal{X}$ and $\mathcal{Y}$ each pair is identically and independently distributed according to. Thus, $h(x)$ can be treated as a random

variable being conditionally distributed with $P(y|x)$ for a given x, and not as a deterministic function of x. The prediction accuracy of h is measured with a **loss function** $L : Y \times Y \to \mathbb{R}^{\geq 0}$. For a given pair (x_i, y_i), the loss function $L(y_i, \hat{y}_i)$ represents how far a prediction $\hat{y}_i$ for the i-th sample x_i is away from the actual corresponding label y_i. The average loss across all $(x, y) \in S$ is called **empirical risk**. Thus, the goal of a supervised learning algorithm is to find the optimal hypothesis h^* for which the overall empirical risk is minimal:

$$h^* = \arg\min_{h \in H} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i). \tag{4.1}$$

Predictions can generally be made for either **regression** or **classification** problems which differ in the nature of their target variable. While regression is used to predict continuous targets $Y = \mathbb{R}$, classification predicts discrete Y. The arrival time estimation problem is a regression task since we are estimating (continuous) arrival times. Although various supervised learning models use different mathematical procedures to predict targets, all of them follow the principle of induction, meaning that general rules are inductively inferred from training data. This is also called **generalization**. In order for a model to generalize well, two phenomena have to be avoided: **Underfitting** and **Overfitting**. Poor predictions on both the training and test data indicate that a model is underfitted. Very good prediction accuracy for training samples $x \in S$, but poor accuracy on test samples $x \notin S$ indicate that a model is overfitted. In supervised learning, this dilemma is known as the **bias-variance tradeoff**. An overfitted model is characterized by rather being low in bias and high in variance, whereas the same goes vice versa for underfitted models. Models high in bias and low in variance are rather simple, but cannot capture complex mappings due to high bias, e.g. linear models. On the other hand, models low in bias and high in variance are able to capture complex patterns in the data which is the reason why they fit given training data S very well. The problem with models high in variance arises when predictions are performed on independent samples $x \notin S$, because the better the model is fitted to S, the higher becomes the variability of prediction performance on independent samples.

A good mechanism that counters overfitting is the practice of **early stopping**. Early stopping is activated when an algorithm performs worse at the current iteration $k \in \mathbb{N}$ than it did at the previous one $k-1$. The mechanism tracks the loss for the next $s \in \mathbb{N}$ iterations. If a better loss compared to the loss attained at $k-1$ improves at

any iteration in the interval $[k, k+s]$, early stopping terminates and the training goes on. When the loss, again compared to the loss attained at $k-1$, never improves until iteration $k+s$, the training is stopped early. All in all, the key challenge lies in finding a model balancing this tradeoff well and thus having good and consistent prediction quality especially on test data. For that reason, we consider diverse models covering the wide spectrum of model complexities to discover the most suitable options for the EAT.

4.2.2 Linear Regression

Linear regression models assume that every y can be predicted with a linear combination of inputs, i.e. the training data S. In the following, we will concisely explain how the classic linear regression works. For further explanations, the reader is refered to Friedman, Hastie, and Tibshirani (2001).

Let $\mathbf{X}^T = (X_1, X_2, \ldots, X_p)$ denote the input matrix containing p column vectors with N components each, and $\hat{Y} = (y_1, y_2, \ldots, y_n)$ the column vector with the corresponding real-valued outputs. A simple linear regression model is a linear combination of coefficients $\mathbf{w} = (w_0, \ldots, w_p)$ and column vectors X:

$$h(X) = w_0 + \sum_{j=1}^{p} X_j w_j \tag{4.2}$$

with coefficient w_0 representing the bias term. The feature vectors $X \in \mathbf{X}^T$ can be of any form and be arbitrarily transformed since linear models only assume linearity in their coefficients, and not in their inputs. A common method of approximating the hypothesis in linear regression is *least squares*, which seeks to find parameters that minimize the squared differences across all N pairs the training set S:

$$\arg\min_{w} \sum_{i=1}^{N} (y_i - w_0 - \sum_{j=1}^{p} x_{ij} w_j)^2. \tag{4.3}$$

The coefficients resulting from the least squares minimization in 4.3 are then employed in 4.2, which can be used to predict labels for given samples. The larger a coefficient, the more does its corresponding feature impact the model outcome.

Linear models are the better choice when it is known that the ground truth mapping between the independent and dependent variables of our model is linear since the model assumes linearity, is easy to implement and fast to train. The flipside

of the coin is that more complex patterns can not be fully captured by linear models since it is always assumed that the underlying mapping of the given problem is linear which might not be the case for more complex scenarios. Nevertheless, we consider the presented linear regression model in our comparison for the former reasons. As we will show later on in section 5.4, the performance of linear models is competitive under certain circumstances.

We will use the linear regression algorithm implemented in *Scikit-Learn*, a library providing numerous methods from the field of data science and machine learning written in Python and proposed by Pedregosa et al. (2011).

4.2.3 Tree-based Ensembles

In the following, we will present the tree-based ensembles we consider in our comparison. We again refer to the explanations done in Friedman et al. (2001). As shown in the literature review, tree-based ensembles like GBDTs and Random Forests are popular in the field of arrival time estimation. Ensemble learning is motivated by the idea that *base learners* (synonym to *model* or *hypothesis*) produce rather inaccurate results on their own, but form an accurate prediction model when they are combined. For tree-based ensembles, decision trees are used as base learners. Generally speaking, decision trees partition the feature space into distinct regions and then, in the case of regression, fit a real-valued prediction to each region. A single decision tree is given by

$$h(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{4.4}$$

where c_m is the constant representing the prediction for all samples belonging to the m-th terminal region R_m. Adopting the mean squared error as the loss metric, the predicted constant $\hat{c}_m$ for each terminal region R_m is the average of all y for each $x \in R_m$:

$$\hat{c}_m = \frac{1}{N_{R_m}} \sum_{x_i \in R_m} y_i \tag{4.5}$$

with N_{Rm} being the number of samples in terminal region R_m. Given this, the decision tree algorithm needs to find optimal binary splits. Solving this analytically is computationally infeasible. Therefore, decision trees are constructed in a greedy fashion. Let j be the split variable, representing the feature based on which the split of the feature space is done on, and s its associated split threshold. The initial

partition creates two disjoint regions

$$R_1(j,s) = \{X|X_j < s\}, R_2 = \{X|X_j > s\}. \tag{4.6}$$

The algorithm seeks optimal j and s by finding constants that minimize the overall loss, which in our case is measured with the mean squared error, for both terminal regions R_1 and R_2:

$$\min_{j,s}\left[\min_{\hat{c}_1}\sum_{x_i\in R_1(j,s)}\frac{1}{2}(y_i-\hat{c}_2)^2+\min_{\hat{c}_2}\sum_{x_i\in R_2(j,s)}\frac{1}{2}(y_i-\hat{c}_2)^2\right] \tag{4.7}$$

Having found the optimal split variable and split threshold, this procedure - also known as the classification and regression tree (CART) algorithm - is repeated for each terminal region until a termination condition holds, e.g. a fixed maximal depth for the tree or a lower bound wrt. number of samples in each terminal region. When trees are build according to the procedure we just explained, decision trees grow level-wise. Note that other ways to build decision trees such as the *best-first* approach presented in Shi (2007) where trees grow leaf wise do exist.

However, decision trees have several shortcomings. First of all, decision trees tend to do a good job of predicting labels for samples used as training data. When it comes to independent samples, i.e. test data, they are performing poorly, which indicates high variance and overfitting. We thus turn to decision tree ensembles, which basically combine several regression trees and aim to reduce the variance of the decision trees in their respective own ways. The two ensemble learning approaches relevant for our work are **bagging**, shorthand for *bootstrap aggregating*, and **boosting**. Algorithms belonging to bagging first create bootstrap datasets by randomly drawing samples with replacement from training data, fit each bootstrapped set a hypothesis and take the average across all hypotheses as the bagging estimate.

Since decision trees are low in bias and high in variance due to their ability to capture complex mappings, bagging is motivated by the idea that averaging across all base learners benefits from the strong law of large numbers and therefore leads to a reduction in variance. Random Forests belong to the class of bagging algorithms and work as follows: For an arbitrarily chosen and fixed amount of estimators $B \in \mathbb{N}$, we create bootstrap data sets S^{*b} with $b = 1, \ldots, B$ of size N from the training data S. The b-th regression tree T_b is grown to S^{*b} by first selecting a subset of m features with $m \leq p$, and then growing a regression tree according to CART as described

above. Having grown each bootstrap data set S^{*b} its own regression tree T_b, random forest for regression averages across all hypotheses of each regression tree:

$$h_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x) \tag{4.8}$$

as the bagging estimate.

TODO: BOOSTING EINLEITEN BEVOR ES IN DIE DETAILS FÜR GBDT GEHT

The first step in GBDT is to initialize a model with a constant $\hat{c}$ that minimizes the overall loss across all training samples.

$$h_0(x) = \arg\min_{\hat{c}} \sum_{i=1}^{n} L(y_i, \hat{c}) \tag{4.9}$$

Using mean squared error as the loss metric, the predictor minimizing the overall loss is simply the average of all y. Given the initial model $h_0(x)$, M regression trees can now be build sequentially as shown in the following. The first step computes pseudo-residuals r_{im} between the prediction value of the former learner for the i-th sample $h_{m-1}(x)$ and the actual i-th target for the m-th tree y_i by deriving the loss function w.r.t to $h_{m-1}(x)$:

$$r_{im} = -\left[\frac{\delta L(y_i, h(x_i))}{\delta h(x_i)}\right]_{h(x)=h_{m-1}(x)} \tag{4.10}$$

for every i-th example. In the second step, the algorithm fits a regression tree to every r_{im} and creates disjoint regions R_{jm} for $j = 1, ..., J_m$. In the third step, we compute:

$$\hat{c}_{jm} = \arg\min_{\hat{c}} \sum_{x_i \in R_{ij}} L(y_i, h_{m-1}(x_i) + \hat{c}) \tag{4.11}$$

for every terminal region j in the m-th tree. Given this, we now update the former weak learner with the new one:

$$h_m(x) = h_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \hat{c}_m I(x \in R_{jm}) \tag{4.12}$$

where α is the learning rate, and $I(\,\cdot\,)$ is the indicator function. Hence, we use the GBDT implemetation of *lightGBM*.

4.3 Feature Selection

A significant role for the success of machine learning models in general plays the data we gather and how we make sense of it - i.e. how we turn raw data into informative features - since machine learning models can only learn from the data they receive for the learning process. Feature selection seeks two conflicting objectives: Minimize the amount of used features while maximizing the accuracy. Balancing this trade-off is crucial, since high dimensional data sets cause a significant increase in training times and are hard to scale and rahter inefficient. Further, the more features there are, the higher is the probability of having features in the feature space that actually decrease the performance of models. But on the other hand, a decrease in features can at the same time mean that possibly strong predictors are excluded from the feature space although they would significantly contribute to the objective. Therefore, we intend to analyze how our models work on data sets with different features and especially of different dimensionality.

Since we operate on the same problem and the same raw data as Hildebrandt and Ulmer (2020), we employ the data model resulting from their feature selection done for the offline approach also in our work as the low-dimensional data model we have spoken of in section 4.1. As we have seen in the literature review, their data model includes temporal, spatial, routing and processing features and is of low dimensionality with a feature space of 12 features. In their data model, they use the customer order time as well as the sum of expected durations for each action in the customer route as temporal features. Restaurant and customer locations belong to the class of spatial features Hildebrandt and Ulmer (2020) use. They further include the number of total stops, pick-up stops and delivery stops to capture the structure of the delivery route to the customer as well as temporal information on time shifts that can occur due to bundling or past insertions. Last but not least, Hildebrandt and Ulmer (2020) include the meal preparation time as a feature that captures the stochastic processing times the arrival time is impacted by.

As we have stated in section 4.1, high-dimensional datasets are mainly not desirable due to increase in training time, scalability, and potentially decreasing accuracy. However, allowing high dimensionality enables us to include information that represent the four feature classes and thus the state of the RMDPEAT very detailled. For the high-dimensional data model, we therefore omit the routing features that count the number of stops, keep all the other 9 features Hildebrandt and Ulmer (2020) used,

and add the following features that are already given in the original raw data model instead:

- **vehicle_route_to_customer_pos_{x,y}_i**: These two features present the target position for action $i \in \{0, 1, \ldots, 23\}$ given by x- and y-coordinates (in kilometers) respectively.

- **vehicle_route_to_customer_action_i**: This feature represents the type of action i. The driver can either drive to either a customer (=1) or restaurant location (=3), or wait at either a customer (=2) or restaurant location (=4). Further, we observe a value of 0 for this feature when the driver finishes the delivery to the targeted customer. When this happens, the target positions presented previously and the duration of the action presented next are also no longer tracked.

- **vehicle_route_to_customer_time_action_i**: This feature denotes the duration of action i.

By that, we get additional temporal, spatial, routing and processing information since all actions and their properties are known. Altogether, the high-dimensional feature space contains 109 features. The original raw data model has more than 1000 features that gave additional information about the actions that allowed a sample to capture the whole RMDPEAT state. Our feature selection is motivated by the idea that by narrowing down the selection to features that are directly related to the customer request that - technically speaking - corresponds to a certain sample, we get all the information that might be somehow relevant for this request. Since the data model of Hildebrandt and Ulmer (2020) can be engineered with the data model we just presented, we will from now on refer to former as the *crafted* data model, and to our propose as the raw data model.

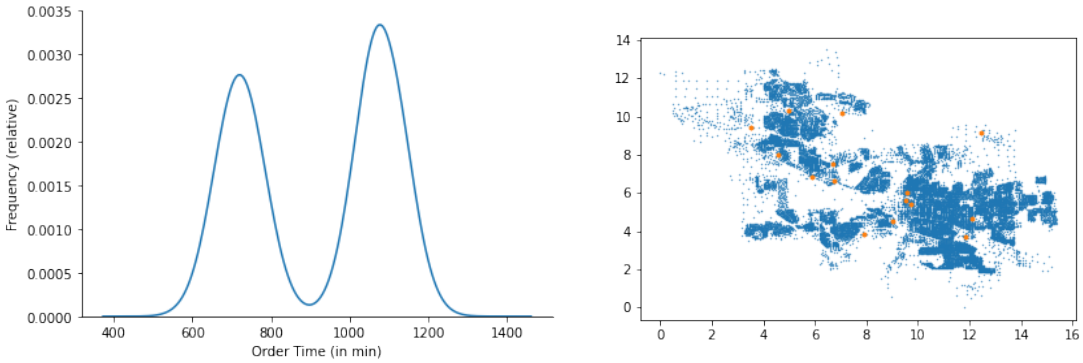
5 Computational Study

This chapter presents our computational study and is structured as follows: Section 5.1 describes the data our experiments are conducted on. Section 5.2 examines how different samples sizes used for training affect the test error. In Section 5.3, we optimize model hyperparameters and seek to get better intuition when for parameter configuration by analyzing parameters considered for our optimization. Last but not least, section 5.4 demonstrates how noise affects predictive performance of our models.

5.1 Data exploration

The available data set is provided by Hildebrandt and Ulmer (2020) who create a dataset using the RMDP instances originally used in M. Ulmer et al. (2020). It comprises 850.469 samples with 23.341 unique customer locations, a delivery fleet of 15 vehicles and 15 unique restaurant locations. The temporal and spatial distribution of the orders is depicted in figure 5.1. Panel (a) of figure 5.1 shows the order behaviour of customers. The x-axis denotes the day time in minutes, the y-axis denotes the relative frequency of incoming customer orders for a given day time on the y-axis. We can observe that the order time behaviour across all customers follows a bimodal gaussian distribution. The order frequency peaks at around 12:00 a.m (roughly 700 minutes of day time) and again around 6.00 p.m (roughly 1100 minutes of day time). This indicates that the probability of an order taking place during lunch or dinner time is relatively high. Panel (b) of figure 5.1 shows the spatial distribution of customer and restaurant locations. The x- and y-axis show the location longitudes and the latitudes respectively. The blue points represent customers, the orange points depict the restaurant locations. There we can see that the majority of restaurants is located where the density of customer locations is rather high.

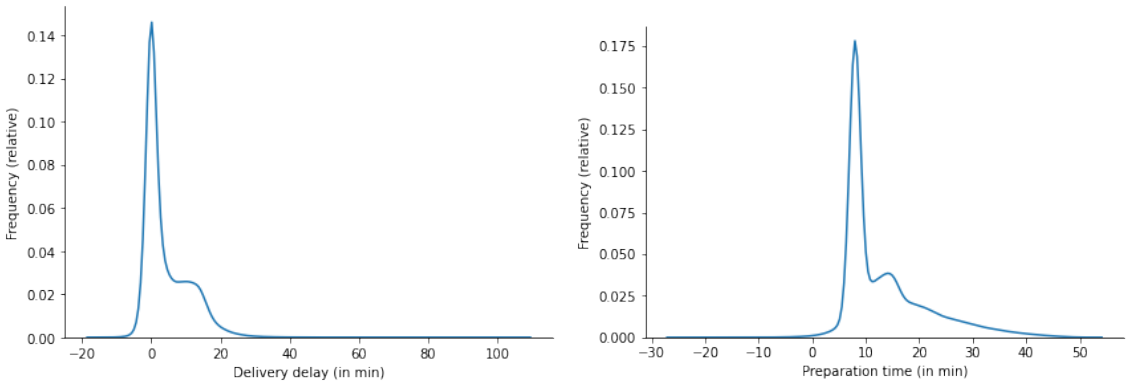
Panel (a) of figure 5.2 shows the distribution of delivery delay times in minutes for all requests in the data. The delivery dela is here defined as the difference between actual and the expected time of arrival based on PoM. First, we observe that roughly



(a) Request arrival time distribution (b) Spatial distributions

Figure 5.1: Spatial and temporal distributions

14-15% of the requests in our data are delivered on time. Negative values for the delivery delay on the plot indicate that a rather tiny amount of orders happens to be delivered earlier than expected. However, we are most interested in the orders that are actually delivered later than expected. We observe that a non-trivial amount of orders is delivered with a delay of up to 20 minutes. As we will later show, arrival time estimation via supervised learning is a vastly better option compared to planning on means. Since arrival times are impacted by uncertainty in meal preparation times, we take a look at their distribution in our data as well. In panel (b) of figure 5.2, we observe that roughly a fifth of the orders had a preparation time of around 10 minutes. However, a non-trivial amount of orders seem to have quite longer preparation times ranging from 15 to less than 40 minutes.



(a) Delivery delay distribution (b) Meal preparation time distribution

Figure 5.2: Spatial and temporal distributions

5.2 Experiment: Sufficient Sample Sizes

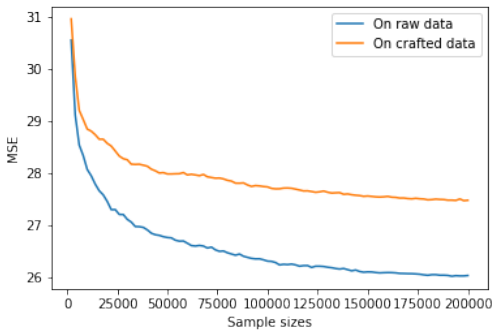
This section compares how a difference in sample size impacts the test error for each possible combination of our models and datasets. Since less samples in the training data correspond to lower training times, we use the results of this experiment for the subsequent hyperparameter optimization and the noise analysis to speed our experimental procedure up. For our linear model and the considered ensembles, we consider 100 training runs with every run corresponding to a sample subset size $\{2000, 4000, \ldots, 200.000\}$. For each model, we set the parameters manually found by trial-and-error since we did not conduct the hyperparameter optimization yet. For GBDT, we set the learning rate to 0.02, the number of estimators to 1000 and enable early stopping with a patience of 20. For Random Forest, we set the number of estimators again to 1000 and subsample 80% of the samples from only 80% of the features in every iteration. For Random Forest, we enable early stopping with a patience of 20 as well. Every iteration for every test instance is validated on the same validation set which contains 20% of our total amount samples.

Figure 5.3 shows the the model performances on both datasets with increasing sample sizes. In all panels, the x-axis depicts the sample size in a training run, and the y-axis represents the mean squared test error (MSE) of that run. GBDT, Random Forest and Linear Regresson belong to their own panels (a), (b), and (c) respectively. For GBDT and Random Forest, we observe that a sample size of 200.000 seems to be sufficient since the models, whether they are trained on the raw or the crafted data, begin to converge there. For GBDT, the MSE at a sample size of 200.000 is approximately 26 when trained on the raw data, and between 27-28 when trained on crafted data. Random Forest performs slightly worse on both datasets when compared to GBDT. However, when less are used for training, In contrast to GBDT and Random Forest, the linear model has a poorer performance on both datasets with a MSE that converges around slightly more than 30 on the raw set and roughly at a little bit more than 31 on the crafted set. However, the linear model converges almost immediately when trained on the crafted dataset while slightly less than 25.000 samples are needed in order for the linear model to converge when we train it on the raw dataset. Moreover, there is only a marginal difference between the performances of the linear model on the different datasets which is not the case for GBDT and Random Forest. All in all, we conclude the following:

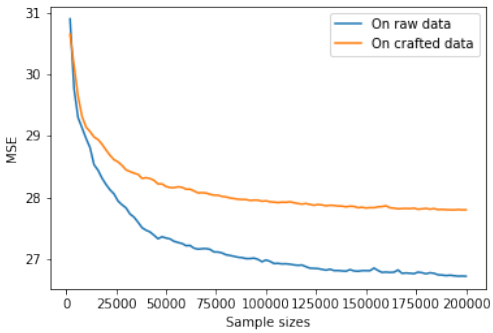
- All models perform consistently better when trained on the raw dataset

- GBDT returns the best results for both datasets

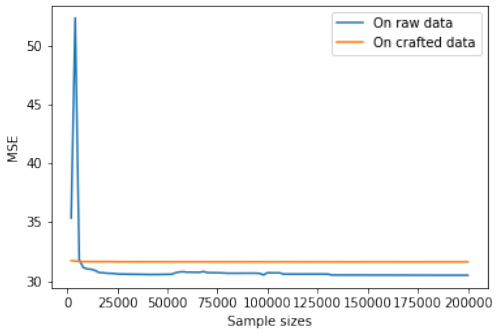
- Regardless of the datasets, the linear model needs the least samples in order to converge. However, GBDT and Random Forest significantly outperform the linear model consistently.



(a) For GBDT



(b) For Random Forest



(c) For Linear Regression

Figure 5.3: MSE for different sample sizes

5.3 Experiment: Hyperparameter Analysis

This section presents the hyperparameter optimization (HPO) experiments conducted for GBDT and Random Forest. We use *Optuna*, a popular HPO framework proposed and designed by Akiba, Sano, Yanase, Ohta, and Koyama (2019), to conduct our HPO experiments. *Optuna* provides a simple implementation design that allows us to analyze several parameters of choice from different points of view. The optimizations were conducted on both the crafted data set and the raw data set. Concretely, we

will proceed as follows: First, we will present the hyperparameters we decided to include in the HPO, then present the optimal hyperparameter values resulting from the experiment and explore the predictive power of different parametrizations. For every GBDT and RF experiment instance, we will use the *Optuna*-implementation of *CMA-ES* to sample values from the respective search spaces of the parameters considered for optimization. To speed up the optimization process, we use the *Optuna*-implementation of the *Hyperband Pruner* presented in Li et al. (2018) and furthermore use only 200.000 of our samples since the models begin to converge at this subset size as shown in section 5.2. We will use the *optuna* implementation of *fANOVA* presented in Hutter et al. (2014) to determine hyperparameter importances. For each parameter importance prediction, we set the number of estimators in *fANOVA* to 1000. For both datasets respectively, we will use 200.000 samples for trainnig, and 20% of all samples for validation to train and validate our models.

5.3.1 GBDT

In this subsection, we will conduct HPO experiments for GBDTs on both datasets.
´

- learning_rate $\in [0.01, 0.05]$ in steps of 0.001.
- max_depth $\in [5, 100]$ in steps of 0.001.
- feature_fraction $\in [0.1, 1.0]$ in steps of 0.01.
- feature_fraction_bynode $\in [0.3, 1.0]$ in steps of 0.01
- num_leaves $\in [20, 300]$ in steps of 1
- min_child_samples $\in [10, 400]$ in steps of 1.
- subsample_freq $\in [1, 10]$ in steps of 1.
- subsample $\in [0.3, 1.0]$ in steps of 0.01.

Besides the parameters considered for optimization, we set the number of estimators for one iteration in GBDT to 1000 and enable early stopping with a patience of 20. Our selection of parameter search spaces is the result of trial-and-error since machine learning problems are highly individual and, as we have already demonstrated with our literature review, different solution approaches for quite similar problems are

possible. The determination of parameter values and parameter search spaces therefore has to happen at least partly in a manual fashion, which is why we regard the trial-and-error heuristic as a suitable approach for the configuration of hyperparameter values and hyperparameter search spaces for our HPO. Detailed descriptions of the considered parameters for the *lightgbm*-GBDT implementation can be found under `https://lightgbm.readthedocs.io/en/latest/Parameters.html`.

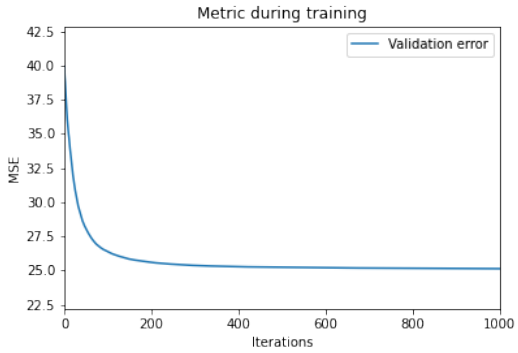
The exact optimal parameter configuration for GBDT on both datasets is depicted in figure 5.4. Training GBDT with the optimal parameter configuration using the full available amount of samples results in a mean squared error of approximately 25.1384 on the raw data, and 27.0852 on the crafted data. Early Stopping does not set in when training GBDT on raw data, whereas it does set in for the crafted data at around 600.

Figure 5.5 depicts the parameter configurations of each optimization epoch in form of a parallel coordinate plot for GBDT on raw data in panel (a) and on crafted data in panel (b). Except the vertical gray line one the very left in the plots, each vertical gray axis represents the values ranging within the defined intervall of its corresponding parameter, whereas the very left line represents the axis for the objective values. The lines connecting all the parameter axes represent the parameter configurations of the optimization epochs. The bluer a line, the better the objective value - in our case the mean squared error. At first glance, it can be seen directly that the parameter configurations of both plots for good predictive performance are quite similar. For both sets, the mean squared error is minimized when we roughly use two-thirds of the features for each GBDT iteration (feature_fraction, feature_fraction_bynode). Less test error is more likely achieved when we allow the algorithm to build rather deep trees up to a maximal depth of 50 while constraining the tree splitting process by specifying a lower bound of around 200 for the minimal amount of samples a node must contain in order to split it further (min_child_samples). We can furthermore see that a better objective value is attained when using nearly all samples considered for a GBDT iteration rather than applying (subsampling) and learning rates ranging between 0.03 to 0.05 (learning_rate).

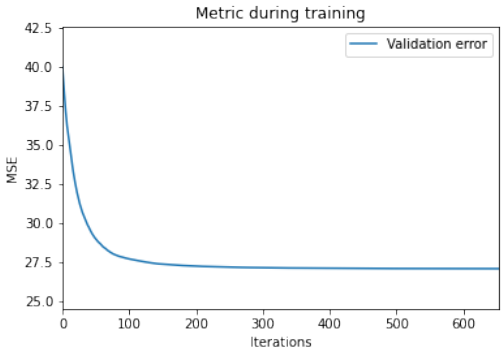
Up next are the hyperparameter importances of GBDT depicted in figure 5.6. Panel (a) shows the importances for GBDT when trained on raw data Panel (b) shows the importances for GBDT when trained on crafted data. The x-axis of the importance plots depicts the relative parameter importances. On the y-axis is categorical and shows the parameters we considered to optimize. We can observe that

	Raw data	Crafted data
GBDT	**Result of optimization** max_depth: 53 learning_rate: 0.031 feature_fraction: 0.65 feature_fraction_bynode: 0.94 num_leaves: 160 min_child_samples: 205 subsample_freq: 6 subsample: 0.95 **Manually set** n_estimators: 1000 early_stopping: 20 max_bin: 1000 MSE: 25.1384	**Result of optimization** max_depth: 52 learning_rate: 0.03 feature_fraction: 0.65 feature_fraction_bynode: 0.94 num_leaves: 160 min_child_samples: 205 subsample_freq: 5 subsample: 0.94 **Manually set** n_estimators: 1000 early_stopping: 20 max_bin: 1000 MSE: 27.0852

(a) Optimal configurations



(b) Training history on crafted data



(c) Training history on crafted data

Figure 5.4: Optimal configurations and training histories for GBDT

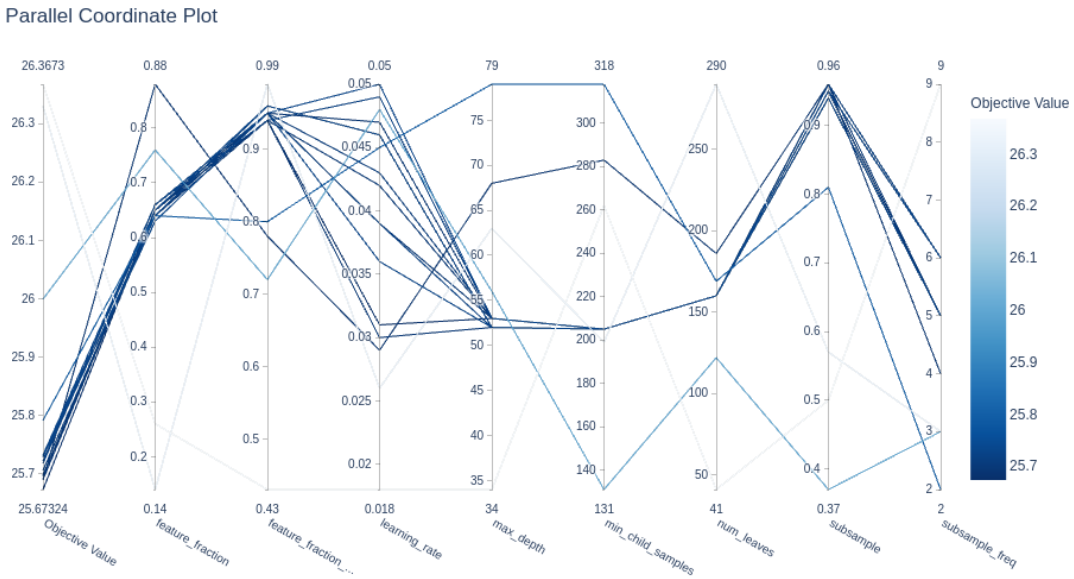
the results for the datasets GBDT was trained on differ significantly. For GBDT on the raw data, we observe that choosing the right amount for subsampling, a suitable learning rate for calculating the gradient in GBDT, and the right feature fraction are of highest importance. For GBDT on the crafted data, we observe that the learning rate and the feature fractio are of even more relative importance, whereas the parameter for subsampling has far less significant impact on the objective value as compared to GBDT on raw data. We observe further non-trivial differences in relative importance for the subsampling frequency showing a difference 7%, and for *min_child_samples* with a difference of 5%. For both datasets, the number of leaves, the percentage of features fractioned per node, and the maximal depth constraint for the estimators remain nearly equally important.

Although there are significant differences between the hyperparameter importances for both GBDT experiment instances, the resulting optimal configuration of both does hardly differ. Thereby, we suggest that the difference in relative parameter importances does not affect the outcome of the optimization. We also suggest that the importances depend on the features that are used. GBDT consistently performs better when trained on the raw data model. This could be due to the reason that, besides the features both sets have in common, our raw set includes exact routing information of every action on the customer's route, whereas the crafted set in contrast only includes only the amount of total stops, pick-up stops, and delivery stops.

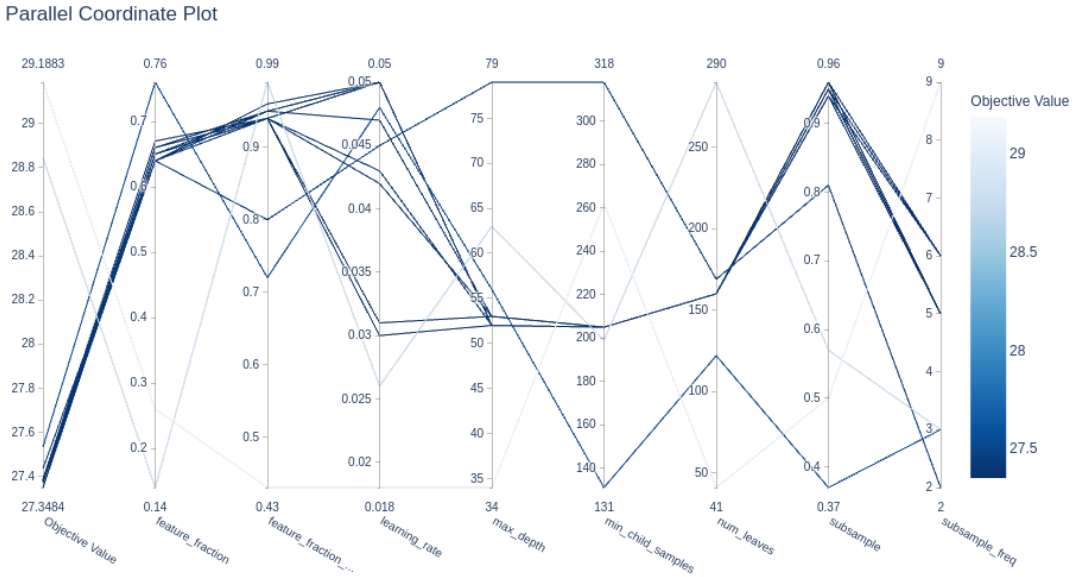
We conclude the following:

- Although there are significant differences between the hyperparameter importances for both GBDT experiment instances, the resulting optimal configuration of both does hardly differ. Thereby, we suggest that the difference in relative parameter importances does not affect the outcome of the optimization. We also suggest that the importances depend on the features that are used.

- GBDT consistently performs better when trained on the raw data model. This could be due to the reason that, besides the features both sets have in common, our raw set includes exact routing information of every action on the customer's route, whereas the crafted set in contrast only includes only the amount of total stops, pick-up stops, and delivery stops.

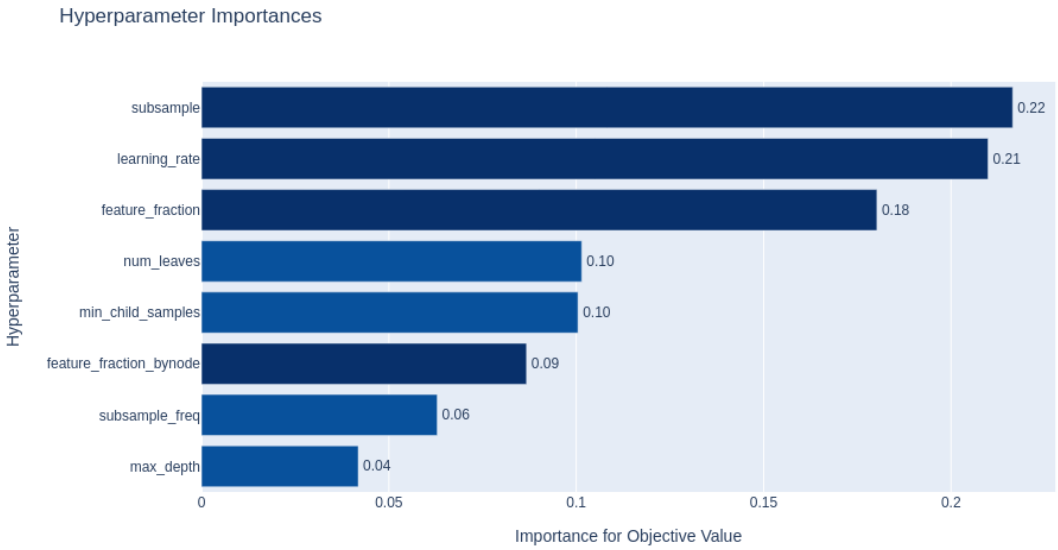


(a) On raw data

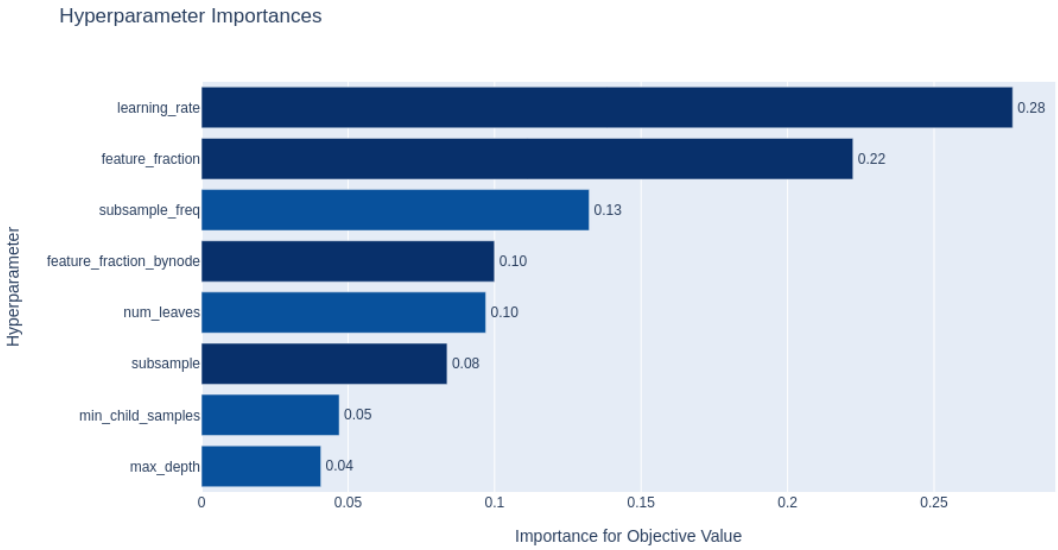


(b) On crafted data

Figure 5.5: Parallel Coordinate Plots for GBDT



(a) On raw data



(b) On crafted data

Figure 5.6: Hyperparameter Importances for GBDT

5.3.2 Random Forest

In this subsection, we will conduct HPO experiments for Random Forest on both datasets. For Random Forest, we decided to optimize following parameters and set the search spaces for each of them as follows:

- max_depth $\in [5, 100]$ in steps of 1.
- feature_fraction $\in [0.3, 1.0]$ in steps of 0.01.
- feature_fraction_bynode $\in [0.3, 1.0]$ in steps of 0.01
- num_leaves $\in [900, 1200]$ in steps of 1
- subsample_freq $\in [1, 10]$ in steps of 1.
- subsample $\in [0.3, 0.99]$ in steps of 0.01.

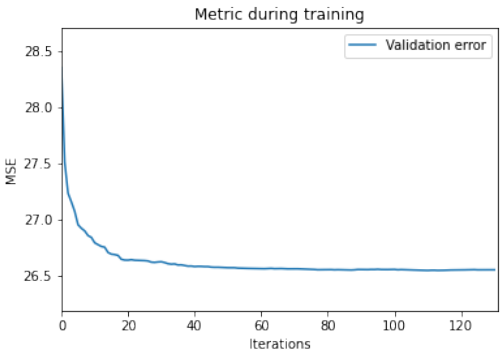
Besides the parameters considered for optimization, we set the number of estimators for one iteration in Random Forest to 1000 and enable early stopping with a patience of 20 as we did for the GBDT optimization. Additionally, we set *min_child_samples* to 1 and thus exclude it from the set of parameters considered for optimization, because we found by trial-and-error that *lightgbm-RF* performs best when it is set very low. For the same reason, we set *min_data_in_bin* to 1. Again, we are going to analyze optimal configurations with the help of the corresponding parallel plots first and then examine the hyperparameter importances for Random Forest trained on each dataset.

Figure 5.7 depicts the optimal parameter configurations resulting of the HPO for the considered parameters and the respective training histories. We observe that Random Forest on the raw data returns a lower test error of 26.5538 compared to the crafted features with a mean squared error of 27.7938 when trained on their respective optimal parameter configurations and with the full amount of samples available. Early Stopping sets in at around 170 iterations for the raw data instance, and at roughly 70 for the crafted data experiment instance.

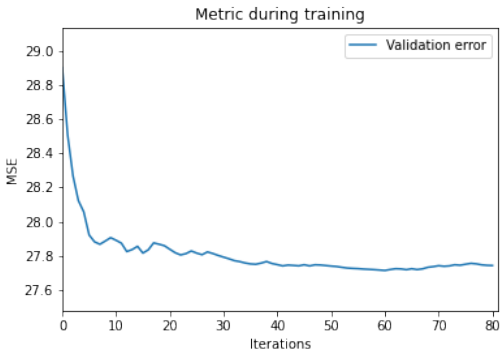
Figure 5.8 depicts the parallel coordinate plots for Random Forest on the raw and crafted features in panel (a) and (b) respectively. We observe that the distribution of parameter configurations achieving a comparably well objective value is quite similar. A minimized objective value is achieved when the feature fraction and the feature fraction by node used for each estimator range between 0.8 and 1. Further,

	Raw data	Crafted data
Random Forest	**Result of optimization** max_depth: 52 feature_fraction: 0.89 feature_fraction_bynode: 0.69 subsample: 0.99 subsample_freq: 5 num_leaves: 1050 **Manually set** n_estimators: 1000 early_stopping: 20 max_bin: 1000 min_child_samples : 1 min_data_in_bin : 1 MSE: 26.5538	**Result of optimization** max_depth: 53 feature_fraction: 0.82 feature_fraction_bynode: 0.82 subsample: 0.93 subsample_freq: 6 num_leaves: 1050 **Manually set** n_estimators: 1000 early_stopping: 20 max_bin: 1000 min_child_samples: 1 min_data_in_bin : 1 MSE: 27.7398

(a) Optimal configurations



(b) Training history on raw data



(c) Training history on crafted data

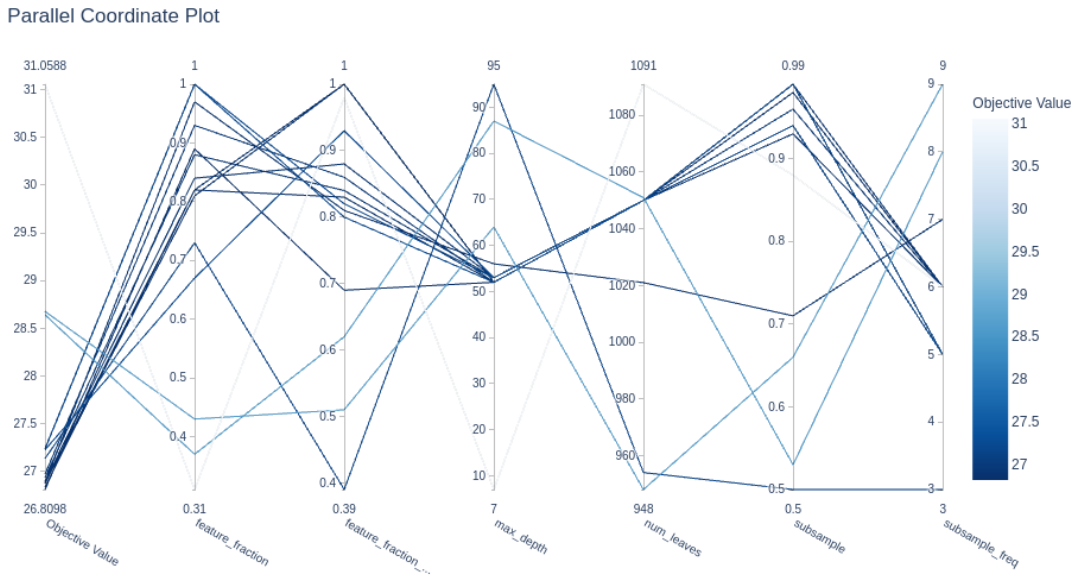
Figure 5.7: Optimal configurations and training histories for Random Forest

the maximal depth constraint set at a rather high value ranging between 50 and 60 seems to be optimal according to the HPO. In the majority of optimal cases, the number of leaves each estimator has at the end of the iteration is at roughly 1050. For subsampling, we observe that setting subsampling fraction greater than 90% in a frequency of 6 is associated with a minimized test error.

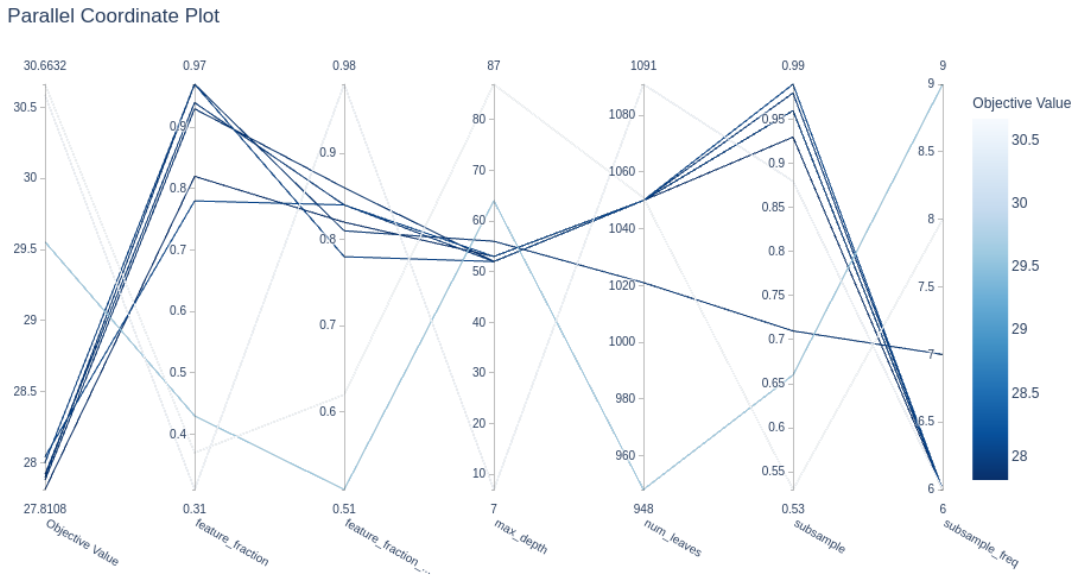
The hyperparameter importances for Random Forest on the raw and the crafted data set are depicted in 5.9. For both instances, we observe that the chosen feature fraction has by far the most impact with a relative importance of 0.43 for the raw data instance, and 0.48 for the crafted data instance. Combining this information with the information given on the parallel plot, we conclude that it is crucial for the

performance to set the feature fractions higher than 80%. The instances significantly differ when it comes to subsampling and the subsampling frequence with differences of 10% and 11% respectively. When it comes to the maximum permitted depth and the number of leaves, the instances also show noticeable percentual difference of 6% and 3% respectively. We conclude the following:

- As it was the case for the GBDT instances, the model performs noticeably better when trained on the raw data model.
- Again, as it was the case for GBDT instances, the parameter configurations promising the best results for both Random Forest instances are quite similar while their respective hyperparameter importances do differ.
- We observe that one parameter is impacting the outcome of the model by far the most compared to the other parameters considered for optimization. Such a dominant impact was not observed for the GBDT instances.

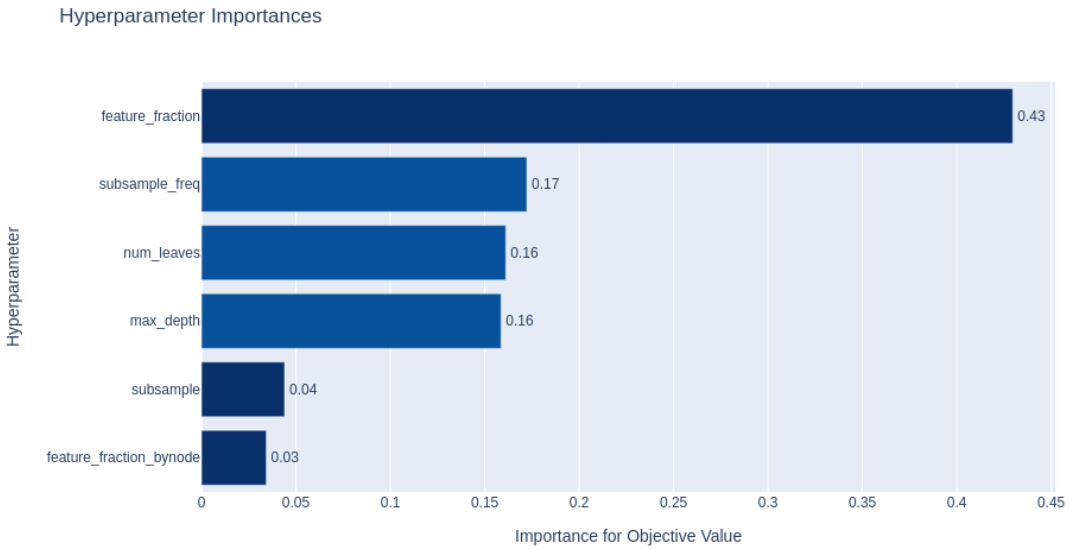


(a) On raw data

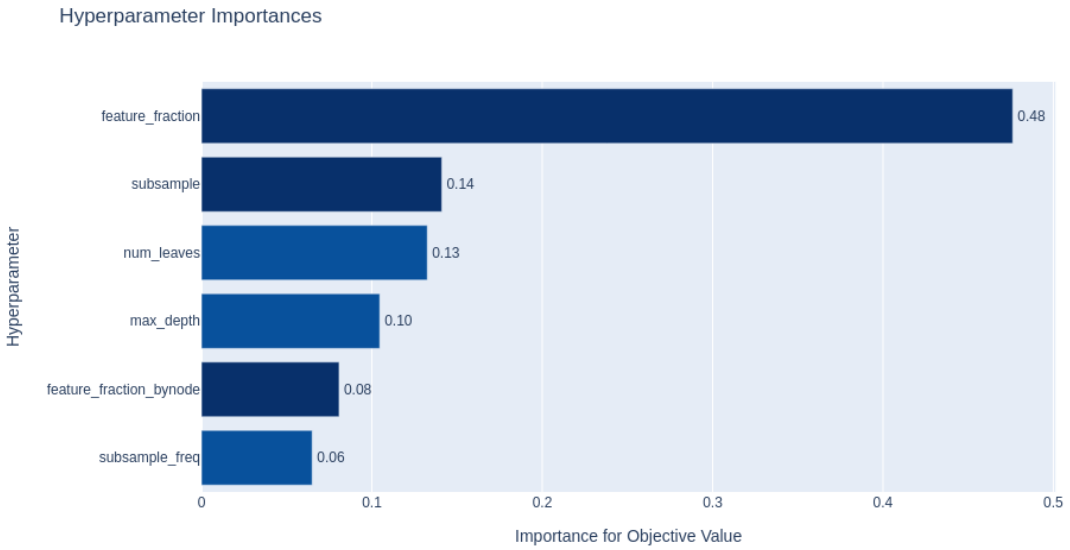


(b) On crafted data

Figure 5.8: Parallel Coordinate Plots for Random Forest



(a) On raw data



(b) On crafted data

Figure 5.9: Hyperparameter Importances for Random Forest

5.4 Experiment: Noise Induction

In this experiment, we test the robustness of our models. We do this by randomly sampling values from the gaussian distribution

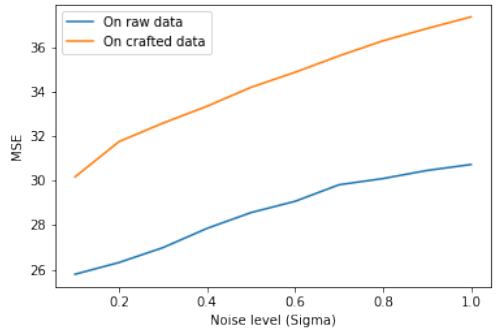
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.1}$$

with a mean μ of 1 and standard deviations $\sigma \in \Sigma$ in the set of $\{0.1, 0.2, \ldots, 1.0\}$ and adding the resulting value, which is the noise, to the order time, the meal preparation time and the PoM-based expected time of arrival. A higher standard deviation is understood as a wider spread of the distribution, and vice versa.

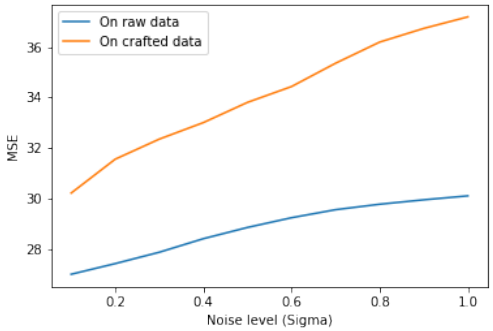
The GBDTs and Random Forests were trained on with the optimal parameter configurations resulting from the HPO and on the full size of available samples. Figure 5.10 depicts the model performances for different levels of noise on both datasets. The x-axis represents the standard deviation, the y-axis shows the mean squared test error. We decided to plot the performances of linear regression separately for both datasets for visualization purposes since the y-axis is scaled vastly different. At first glance, we observe that a higher test error is associated with higher levels of noise for every test instance. GBDT and Random Forest trained on the raw data produce the best results. For them, we observe quite similar behaviour when trained on the different datasets: While both start out fairly close to the value attained in figure 5.4 and 5.7 with a mean squared error of approximately less than 26 and 28 respectively on the raw set, both show a monotonous increase in test error up to approximately 30 for the highest noise level considered. However, we get significantly different results when we train GBDT and Random Forest on the crafted set. For both models, the MSE starts out with a mean squared error around 30, which is noticeably less than the mean squared error achieved with the configurations for the crafted set depicted in 5.4 and 5.7, and rise up to slightly more than 36. We thus conclude that the models trained upon the raw set are significantly more robust when noise is induced. Our linear model produced results that are highly interesting: Despite both (c) and (d) visualizing quite similar looking exponential increases, the resulting MSEs for the different noise levels could not be further away from each other. When trained on raw data, linear regression levels off between 30-31 and is therefore the most robust model in our comparison since its variability in results is by far the lowest. When trained on crafted data however, a consideration of linear

regression as a prediction model is no longer an option as the MSE skyrockets to 1200 at a noise level of 1.0. We further note that linear regression on the raw data consistently produces better results than GBDT and Random Forest when trained on the crafted data. Wrt. to the robustness of our models, we conclude the following:

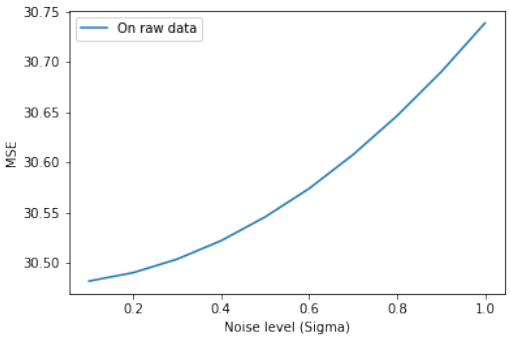
- Whether we use the raw or the crafted data set for training, GBDT and Random Forest produces the best results. On raw data, GBDT is ahead of random forest, whereas both perform nearly equally good when trained on the crafted data set.
- Linear Regression is robust to noise the most on the raw data. Surprisingly, the complete opposite is true when we train linear regression on the crafted data.



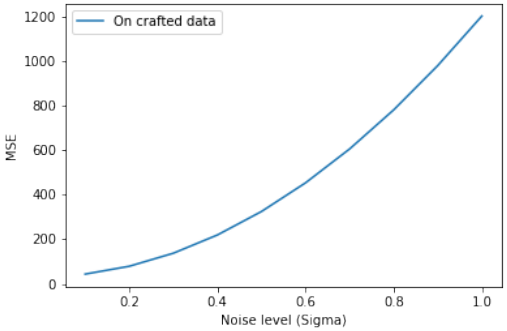
(a) For GBDT



(b) For Random Forest



(c) For Linear Regression (Raw)



(d) For Linear Regression (Crafted)

Figure 5.10: Performance of models for different noise levels on both datasets

6 Conclusion

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A next-generation hyperparameter optimization framework.*
26

Cheng, J., Li, G., & Chen, X. (2019). Research on travel time prediction model of freeway based on gradient boosting decision tree. *IEEE Access*, *7*, 7466-7480. doi: 10.1109/ACCESS.2018.2886549
6

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1) (No. 10). Springer series in statistics New York.
17, 18

Hansen, N. (2016). *The cma evolution strategy: A tutorial.*
14

Hildebrandt, F., & Ulmer, M. (2020, 08). *Supervised learning for arrival time estimations in restaurant meal delivery.*
3, 7, 8, 11, 12, 13, 21, 22, 23

Huang, H., Pouls, M., Meyer, A., & Pauly, M. (2020). *Travel time prediction using tree-based ensembles.*
6, 7

Huang, L., & Xu, L. (2018). Research on taxi travel time prediction based on gbdt machine learning method. In *2018 eighth international conference on instrumentation measurement, computer, communication and control (imccc)* (p. 718-722). doi: 10.1109/IMCCC.2018.00155
6, 7

Hutter, F., Hoos, H., & Leyton-Brown, K. (2014, 22–24 Jun). An efficient approach for assessing hyperparameter importance. In E. P. Xing & T. Jebara (Eds.), *Proceedings of the 31st international conference on machine learning* (Vol. 32, pp. 754–762). Bejing, China: PMLR. Retrieved from `http://proceedings.mlr.press/v32/hutter14.html`
14, 27

Jindal, I., Chen, X., Nokleby, M., Ye, J., et al. (2017). A unified neural network approach for estimating travel time and distance for a taxi trip. *arXiv preprint arXiv:1710.04350*.
6

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). *Hyperband: A novel bandit-based approach to hyperparameter optimization.*
14, 27

Liu, S., He, L., & Max Shen, Z.-J. (2018, 05 15). On-time last-mile delivery: Order assignment with travel-time predictors. *Management Science*. Retrieved from `https://doi.org/10.1287/mnsc.2020.3741` doi: 10.1287/mnsc.2020.3741
4, 7, 13

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.
18

Psaraftis, H. N., Wen, M., & Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, *67*(1), 3-31. Retrieved from `https://onlinelibrary.wiley.com/doi/abs/10.1002/net.21628` doi: https://doi.org/10.1002/net.21628
8

Shi, H. (2007, 01). Best-first decision tree learning. *Master Thesis*.
19

Siripanpornchana, C., Panichpapiboon, S., & Chaovalit, P. (2016). Travel-time prediction with deep learning. In *2016 ieee region 10 conference (tencon)* (p. 1859-1862). doi: 10.1109/TENCON.2016.7848343
6

Ulmer, M., & Thomas, B. (2018, 03). Enough waiting for the cable guy - estimating arrival times for service vehicle routing. *Transportation Science*.
9, 10

Ulmer, M., Thomas, B., Campbell, A., & Woyak, N. (2020, 08). The restaurant meal delivery problem: Dynamic pick-up and delivery with deadlines and random ready times. *Transportation Science*. doi: 10.1287/trsc.2020.1000
3, 8, 9, 23

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., & Thomas, B. W. (2017). *Route-based markov decision processes for dynamic vehicle routing problems* (Tech.

Rep.). Technical report, Braunschweig.
9

Vanajakshi, L., & Rilett, L. R. (2007). Support vector machine technique for the short term prediction of travel time. In *2007 ieee intelligent vehicles symposium* (p. 600-605). doi: 10.1109/IVS.2007.4290181
6

Zhu, L., Yu, W., Zhou, K., Wang, X., Feng, W., Wang, P., ... Lee, P. (2020). Order fulfillment cycle time estimation for on-demand food delivery. New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3394486.3403307`
3, 7, 13