Master Thesis

A machine learning approach to predict service level for call centers

Author: Hatim Akidas (2596883)

1st supervisor:	Rene Bekker
daily supervisor:	Siqiao Li (CCMath)
2nd reader:	Sandjai Bhulai

A thesis submitted in fulfillment of the requirements for the VU Master of Science degree in Business Analytics

August 1, 2022

Abstract

One of the main challenges for call centers is to utilize the appropriate type and amount of workforce to handle stochastic demand throughout time. The process which is concerned with tackling this challenge is named workforce management (WFM). One of several phases within the WFM process is the agent scheduling phase, in which the service level is predicted to determine whether the schedule meets the service demands. Currently, service level prediction is done by the use of Erlang models (Erlang C, Erlang A) as well as simulation. However, the Erlang models make several assumptions that do not always hold in reality, which can lead to poor performance. Whereas simulation may potentially perform better than Erlang models, it is less convenient to use in a business environment due to its large time complexity and challenging implementation. In addition, setting up a simulation model is a challenging task which requires advanced skills and a lot of time. Furthermore, a call center may show different behaviour depending on different attributes like its location or agents. Based on this behavior different correlations and patterns may be found, which cannot be captured by the Erlang models. Capturing these patterns by a simulation model is a challenging task, since extensive data analysis and implementation expertise is required to do so. We wonder whether a machine learning model is able to capture these type of hidden patterns and thus provide a more accurate prediction of the service level. This paper proposes a machine learning method to predict the service level in a realistic call center environment, alternative to the traditional Erlang models and simulation. The results show that the proposed machine learning method is able to have a more accurate prediction of the service level than Erlang C and Erlang A with negligible computing time. We were able to implement a machine learning method that has a prediction error of 0.195 in terms of *WAE* in our experiments, which is a reduction of 47.9% and 26.6% with respect to the Erlang C and Erlang A model, respectively.

Contents

CONTENTS

List of Figures

LIST OF FIGURES

List of Tables

1

Introduction

Call centers play an important role in providing good customer service for a wide range of businesses. One may argue that due to the increasing popularity of offering automated customer service, this role may become less significant in the future. However, market growth statistics [12] show that the call center market size is increasing and is expected to increase in the years to come. The general goal of call centers is to offer high quality customer service while minimizing labor costs. A well-known metric to measure the quality of customer service within call centers is the service level. The service level (SL) describes the percentage of calls that are responded to within a certain time threshold τ. Call centers are required to meet the so-called service level agreement (SLA), determined over a certain time period. An often used SLA is that 80% of incoming calls (including abandoned calls) must be answered within 20 seconds, measured per working day. A formal definition of the service level is given in chapter 2. It is important to note that although the service level makes a great tool to measure quality of service provided by a call center, it must be put in perspective. Although responding to as many calls as possible within the specified threshold will yield a higher service level, the actual quality of service provided during these calls matters at least as much.

Within call centers service may only be provided shortly after demand has occurred. The short time in which a call must be responded to, combined with the randomness in demand and service time, creates challenges regarding workforce planning. An attempted solution that may naturally address this challenge is predicting the required workforce throughout time, based on the forecasted demand. The process of utilizing the appropriate workforce at the right moment is called workforce management (WFM). The WFM process consists of multiple phases, which we further elaborate on in chapter 2. This paper mainly focuses on the agent scheduling phase.

An important part of the agent scheduling phase is predicting what the service level will be over a certain period of time (e.g. a working day), given an agent schedule and a forecast of the calls arriving. Based on this predicted service level, the agent shifts may be rearranged, such that the service level of the considered day meets the SLA. The SL estimation is often accomplished using Erlang models (Erlang C, Erlang A), which are easy to apply in practice by only requiring a small number of relatively simple inputs. However, the Erlang models have certain assumptions which may not always hold in reality. These

assumptions include that the calls arrive according to a Poisson process and that the service times follow an exponential distribution. Another method to predict the service level is by simulation. Whereas simulation may potentially achieve more accurate prediction of service levels than the Erlang models, it is less convenient to use in a business environment due to its large time complexity and challenging implementation. Additionally, a call center may show different behavior depending on several characteristics like its location, its agents or the nature of the underlying business. In this behavior, different correlations and patterns may be found, which cannot be captured by the Erlang models. Capturing these patterns by simulation is a challenging task since extensive data analysis and implementation expertise is required to do so. Moreover, modelling can be very challenging since many different factors and patterns have impact on the service level.

One example of these hidden patterns in agent behavior may be that specific agents take considerably more time to handle calls due to various reasons, which in turn may impact the service level. This example illustrates a hidden pattern which may be found by analyzing the data of a call center, but which cannot be captured by Erlang models. However, a machine learning model is able to capture patterns of such kind and, in doing so, may provide a more accurate prediction of the service level. Moreover, in recent years call center data has become more accessible than ever before. This allows for data-driven approaches of optimizing business processes within this industry. Thus, this paper proposes a framework of how to build a machine learning model specifically for a call center using its historical data set so as to provide a more accurate prediction of the service level than the traditional Erlang models while having limited computational time.

In chapter 2 we describe the process of WFM in a call center context. Additionally, we explain how and in which phases of the WFM process the Erlang models are applied. In chapter 3 we describe our data set, explain processing steps and present data analysis to have a better understanding of the call center characteristics as well as potential patterns. In chapter 4 we explain our methodology, including the choice of model, feature selection and performance measure choices. In chapter 5 we obtain several features. This initial feature selection is based on additional feature analysis as well as data analysis that is presented in chapter 3. In chapter 6 we present the results in terms of model performance on the conducted experiments and provide analysis to interpret these performances. In chapter 7 we conclude our research by briefly summarizing our findings, describing our contributions and giving recommendations for practitioners. Finally, in chapter 8 we discuss the limitations of this paper and give recommendations for future research.

1.1 Related work

This paper aims to provide an approach to predict the service level for multi-skill call centers using machine learning. Call centers are usually modelled by queuing models. The most widely used queueing model for modelling call centers is the Erlang C model (M/M/c queue), which was introduced by A.K. Erlang in [8]. [3] analyzes the fit of the Erlang C model in realistic call center situations. They show that when Erlang C is used in a realistic call center environment the predicted performance measures contain large error. This error tends to be pessimistic, the Erlang C model tends to predict a lower service level than in reality [3], which can lead to overstaffing. C. Palm proposed the Erlang A model (M/M/N+M) in [18], which extends the Erlang C model by considering abandonment. The Erlang A model assumes that every caller has a patience time. If the waiting time for an agent to respond exceeds the patience time of the caller, the call is abandoned. The patience time is modelled as an exponential distribution with constant mean [11, 17]. The Erlang A model seems to make a decent prediction of steady-state behavior of high traffic call centers. However, in realistic call center situations several assumptions are violated. This causes the Erlang A model to suffer significant error in predicting performance measures. The error seems to have an optimistic bias in low volume scenarios and a pessimistic bias in high volume scenarios [19]. Erlang C and Erlang A both assume that the calls are homogeneous and that all agents are able to handle these calls. In a multi-skill situation, which is often the case in a realistic call center environment, this assumption is violated [16].

Simulation models are proposed as a method to model multi-skill call centers in order to predict performance measures [1, 6]. [13] provides an overview of simulation models to model call centers. They discuss required inputs to model a call center by the use of simulation. Simulation models allow for complex modelling and are able to make accurate predictions of performance measures [4, 2]. However, one of the major drawbacks of simulation models is that they may have a large computational time. This makes simulation models less appealing for schedulers that require multiple iterations of performance measure predictions within a short time frame. Due to the need for faster methods that allow for real-time optimization of call center schedules, literature has recently started exploring machine learning methods for prediction. [21] proposed a machine learning framework to predict performance measures for a multi-skill call center, given a staffing level obtained with simulated data. The paper concluded that a machine learning approach to predict performance measures is promising for both small-to-mid-sized and large complex call centers. [5] proposed a data-driven method to predict the service level based on real-world data. They discuss several input features based on data analysis and propose ensemble tree methods to predict the service level. The paper conducts performance experiments in which ground-truth values of input features were used instead of forecasted values, which are used in reality. It concludes a reduction of 6% (MAE) in prediction error compared to Erlang A based on the experiments.

2

Preliminaries

In this chapter we provide relevant information that is needed to have a better understanding of this paper. This includes the context in which the research is conducted. In section 2.1 we give a general description of the call center process and WFM. In section 2.2 the Erlang models are briefly discussed and a description is given of their application within call centers. In section 2.3 we provide a list of definitions of terms and formula's that are frequently mentioned in this paper.

2.1 Call center process & WFM

The data set considered in this paper originates from Vanad Laboratories, which uses a first come first served (FCFS) routing policy. In practice this means that the first arriving call will be served first In figure 2.1 we show a simplified diagram which illustrates the call center process schematically. Customers may abandon the queue if the waiting time is too long. After abandonment the call may either be lost, or the customer may redial to try again, which is counted as a new call arrival. After customers are served they may either be satisfied or call again for any reason related to the previous call, which again is considered as a new call arrival.

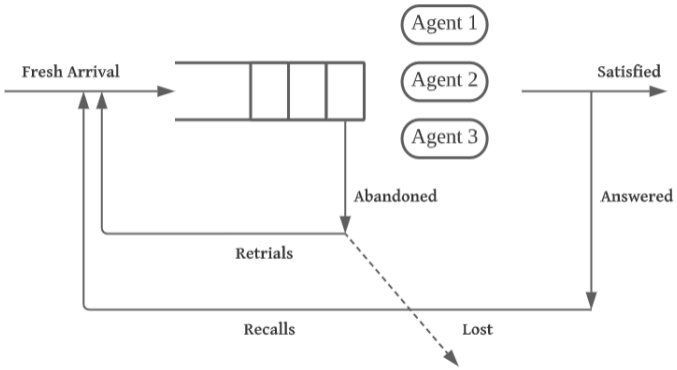

Figure 2.1: Simplified overview of call center call process with three agents

This paper focuses on a multi-skill call center situation. In a multi-skill call center dif-

ferent calls can require different types of service. Only agents that are trained to offer a specific service are then able to handle an incoming call that requires the aforementioned service. This constraint introduces a new layer of complexity when it comes to forecasting and agent scheduling compared to a single-skilled call center as sketched in figure 2.1.

The WFM process aims to utilize the appropriate workforce at the right moment. In figure 2.2 we present a schematic overview of the WFM process [10]. The process can be split in five steps, one row for each step. In the *Time* column we observe an approximation of how much time in advance the considered step should be taken. The *Frequency* column indicates how often the considered step should be taken. *Steps* describes the steps that are taken. Note that prior to each step there is a forecasting step. This forecast is made based on historical data and gives an indication of how much call volume to expect within the desired time scale. Capacity planning is an important step which aims to estimate how many agents are required based on a forecast and on the required output. It can also be noted that capacity planning occurs several times for different steps and processes. The distinction between those steps is made based on their time scale and desired output. This brings us to the next column *Output* which specifies what the desired outcome is after taking the specified steps. To clarify how the overview should be read and how the steps can differ from each other based on the time scales we give a description of the first two rows of the overview.

We see that the budget planning process and capacity planning process both have forecasting and capacity planning as steps to take with different desired output. Within the budget planning process, which is described in the first row, it is noted that approximately a year in advance a forecast of the expected call volume should be made. Based on this forecast the call center performs capacity planning. Based on this capacity plan the call center then creates a budget plan. During the capacity planning process, described in the second row, the call center makes a forecast of the call volume approximately three months in advance. This allows a more precise forecast than the forecast made in the budget planning phase. Based on this forecast the number of agents needed weekly can be estimated. Combined with the budget plan, the hire/fire agenda is developed so as to meet the required demand with sufficient service level as much as possible.

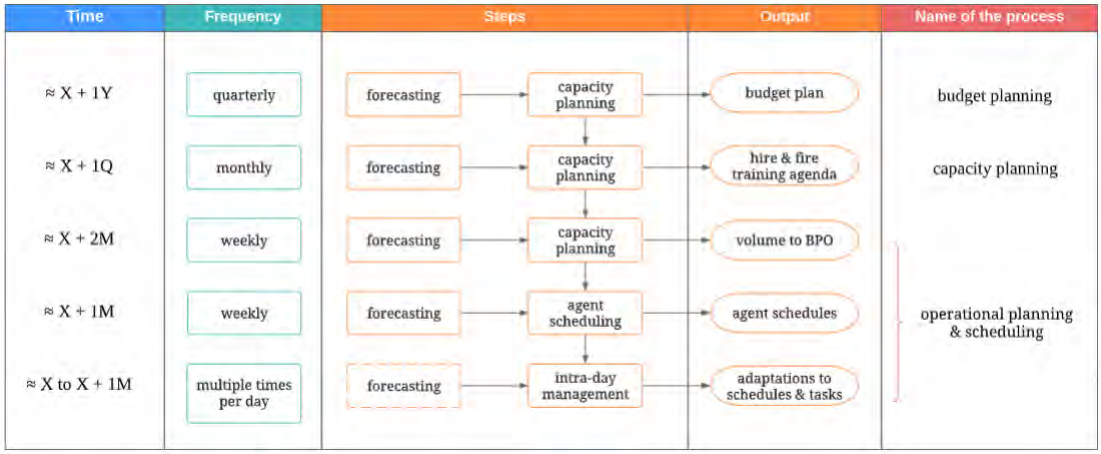

Figure 2.2: Schematic overview of WFM process

As mentioned in section 1, our research locates in the agent scheduling phase. This phase in practice often consists of multiple steps. The process starts with staffing, which requires a forecast of the call volume, the outsourced call volume and the available (hired and trained) agents as input. During this step the schedulers estimate how many and which agents are required for every day. This estimation is generally done by the use of queueing models (e.g. Erlang C) or simulation. Service performance measures (service level, average speed of answer) as well as employee workload measures (occupancy) are taken into account during staffing. This means that the number of agents staffed for a certain day must meet several requirements, for example the service level agreement (SLA).

After staffing, schedulers can convert staffing levels to shifts and assign them to agents. This phase of scheduling shifts and shift assignment based on staffing levels can be compared to a feedback loop, which is illustrated in figure 2.3 [10]. During this phase a certain schedule is created, after which the performance measures (e.g. SL) are predicted throughout the day and based on the result the schedule may be altered by re-arranging the shifts and the shift assignment to the agents. In this phase, agents' contract hours, unavailability and scheduling rules should be considered. This is repeated until a sufficient schedule is found to meet their different kinds of SL agreements (e.g., meet a SL target throughout the day or meet a SL target every interval). The prediction of the performance measures is again done by queuing models or simulation. In practice, many call centers or scheduling tools will apply the Erlang C or Erlang A model instead of simulation during scheduling to predict the service level per interval because of its calculation speed. The focus of our research is on this scheduling & shift assignment phase. A better SL prediction considering agents' behavior can lead to a more reliable schedule which gives less adjustment in the last phase: intra-day management process.

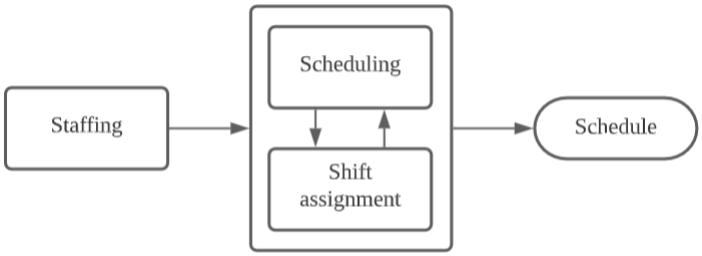

Figure 2.3: Diagram of agent scheduling process

2.2 Erlang models

To give a better understanding of how the Erlang models are exactly used we give formal formulations of each queuing model and further elaborate on their utility within the scheduling phase.

The Erlang C model (also known as M/M/c queue) is the most simple and popular queuing model used within call centers. It requires several assumptions, some of which can also be its main drawbacks when used for call centers. These assumptions within a call center environment are:

- Poisson arrival process of calls,
- Exponentially distributed service times,
- no abandonments,
- calls are handled according to FIFO,
- calls are directed to the first available agent,
- queue sizes are unlimited.

The probability that a calling customer has to wait to get service is given by the Erlang C formula $P_w = \frac{\frac{A^N}{N!}\frac{N}{N-A}}{(\sum_{i=0}^{N-1}\frac{A^i}{i!})+\frac{A^N}{N!}\frac{N}{N-A}}$, where

- P_w is the probability that an arriving call has to wait,
- λ is the number of call arrivals per hour,
- T is the average handling time in minutes,
- $A = \lambda\frac{T}{60}$ is the total traffic intensity in Erlangs (total call hours per hour),
- N is the number of agents available.

Based on the probability P_w the service level can be calculated according to the model. The formula is given by $SL = 1 - P_w * e^{-\frac{\tau(N-A)}{60T}}$, where τ is the response threshold in seconds. For the system to be stable the number of available agents must be larger than the total traffic intensity ($N > A$). Given a call volume forecast, average handling time and number of agents the Erlang C model can be used to predict the service level. This service level prediction is done at several phases of the WFM process. Since we are concerned with the scheduling phase, we will give a description of how the Erlang C model is used during this phase. In practice, call centers may have different types of SLA's that they must meet. Contracts may contain agreements about service levels that must be met throughout the day (e.g. meeting service levels over consecutive intervals) as well as over each day separately. This is an important reason for call centers to predict service levels per interval when scheduling. Partially guided by the predicted service levels, the schedule-makers rearrange the schedule and again apply the Erlang C model to predict new service levels after

changes are made. This is repeated until a schedule is found that meets the concerned SLA.

One of the main drawbacks of the Erlang C model is that it assumes that the queue can be infinitely long which means that there are no abandonments. It is evident that this assumption does not hold in reality, since customers will hang-up after a certain amount of time waiting in the queue. The Erlang A model (also known as M/M/N+M queue) extends the Erlang C model by taking abandonments into account. It models that each caller has an exponentially distributed patience time with mean $\frac{1}{\theta}$. If the waiting time exceeds the patience time, the caller will abandon the queue and hang-up the call. Even a small amount of abandonments can have a large impact on the performance of the queuing system. Extending the model from assuming no abandonment to considering abandonment is therefore a fundamental change which can have great impact on the model performance measures. In contrast to the Erlang C model, the Erlang A model tends to overestimate the service level [20], which can lead to understaffing.

2.3 Definitions and terminology

Definitions of frequently used terminology are given below.

- Call volume: The number of calls that arrive to the call center within a specified time interval is referred to as call volume.
- Abandonment: The number of calls that are hanged up before being picked up by an agent within a specified time interval is known as abandonment.
- Call time: The time that an agent spends on call with the caller.
- Wrap-up time: Time that is spent on finishing work related to a call after hanging up is referred to as wrap-up time.
- Shrinkage: The time (or fraction of time) that agents spend on activities that do not contribute to handling calls (e.g. breaks or meetings). We distinguish between scheduled and unscheduled shrinkage.
- Service level (SL): The fraction of calls that are answered within a time threshold τ over a time interval. There are several methods to define the service level. The definition that is considered in this paper is given by $SL = \frac{\#answered \leq \tau}{\#offered}$, where τ equals 20 seconds.
- Service level agreement (SLA): The service level that the call center aims to minimally offer. The industry standard is that 80% of calls are answered within 20 seconds.
- Handling time (HT): The sum of call time and wrap-up time.
- Average Handling Time (AHT): The average handling time for a specified interval.

- Weighted Average Error (WAE): The average prediction error weighted by the call volume. WAE is defined as:

$$WAE = \frac{\sum_{i=1}^{n} C_i |y_i - \hat{y}_i|}{\sum_{i=1}^{n} C_i} \tag{2.1}$$

  , where

  - C_i is the call volume of interval i,
  - y_i is the realised service level over interval i,
  - $\hat{y}_i$ is the predicted service level over interval i
  - and n is the number of intervals.

- Mean Absolute Error (MAE): is defined by

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|. \tag{2.2}$$

- Gradient Boosted Decision Trees (GDBT): A tree-based ensemble learning method that uses gradient boosting as its ensemble technique.

3

Data analysis

In this chapter we describe the data and perform analysis to get a better understanding of the call center and its attributes. In section 3.1 we give a thorough description of the data as well as relevant data processing steps. In sections 3.2 to 3.5 data analysis is performed to find potential patterns and correlations, which are used to determine the machine learning model features.

3.1 Data description and processing

The data that is used in this paper is from a multi-skill call center named Vanad Laboratories. Two data sets of the year 2014 are considered. The first data set consists of call records whereas the second data set consists of agent activity records. Both data sets consist of one year of records. In table 3.1 we illustrate example records of the call record data set. The column descriptions are summarized below.

queue_time	queue_id	agent	answered	consult	transfer	hangup
2014-01-03 14:45:44	*30175*	*6945*	*2014-01-03 14:46:25*	*NaN*	*NaN*	*2014-01-03 14:48:54*
2014-01-03 14:45:48	*30175*	*9427*	*2014-01-03 14:46:29*	*2014-01-03 14:49:57*	*2014-01-03 14:50:59*	*2014-01-03 14:57:54*
2014-01-03 14:45:56	*30175*	*NaN*	*NaN*	*NaN*	*NaN*	*2014-01-03 14:46:28*

Table 3.1: Example of call record data

- *queue_time* describes the moment that the call has arrived.
- *queue_id* denotes what queue type the contact record is admitted to.
- *answered* describes whether a call was answered, and if so, the moment it was answered. Missing values mean that the call was never answered and thus abandoned.
- *consult* describes if and when a consultation (internal) call was made to help serve the customer. Missing values show that there was never a consultation call for the considered record.
- *transfer* describes if and when the call was transferred to a colleague to help serve the customer. Missing values indicate that there was no transfer made for this call record.

- *hangup* describes the moment the call was ended.

This data set contains 1.543.164 rows of contact records, each assigned to one of 27 queue types. We observe that the eight most common queue types make up approximately 99% of all contact records. We decided to drop the records which are assigned to the less relevant queues and only consider these eight queue types. The call center is closed on Sundays and has minimal activity on Saturdays compared to the rest of the week. The call volume on Saturdays makes up 1% of the total number of calls. Additionally, we observe that on working days, the majority of calls arrive between 08:00 - 20:00. We thus decide to focus on calls that occur between 08:00 - 20:00 on working days. Calls that occur during weekends, national holidays or beyond the specified time frame are beyond the scope of this paper. After dropping the mentioned records we are left with 1.503.127 call rows, which is a decrease of 2.6% of the total number of records.

We immediately add several columns which contain relevant information about the records. An updated illustration of the call record data is given in table 3.2. These columns are summarized below.

queue_time	queue_id	agent	answered	consult	transfer	hangup	wait_time	talk_time	after_call	handling_time
2014-01-03 14:45:44	*30175*	*6945*	*2014-01-03 14:46:25*	*NaN*	*NaN*	*2014-01-03 14:48:54*	*41*	*149*	*0*	*149*
2014-01-03 14:45:48	*30175*	*9427*	*2014-01-03 14:46:29*	*2014-01-03 14:49:57*	*2014-01-03 14:50:59*	*2014-01-03 14:57:54*	*41*	*685*	*37*	*722*
2014-01-03 14:45:56	*30175*	*NaN*	*NaN*	*NaN*	*NaN*	*2014-01-03 14:46:28*	*32*	*0*	*NaN*	*NaN*

Table 3.2: Example of call record data after modification

- *wait_time* denotes the total time in seconds the customer has waited in the original queue before receiving response or hanging up. It is defined by $answered - queue\_time$ if an agent answers the call and by $hangup - queue\_time$ otherwise.
- *talk_time* denotes the total time in seconds the call has lasted. It is defined by $hangup - answered$ if an agent picks up, and by 0 otherwise.
- *after_call* describes the time spent by the agent on finishing up work related to the considered call, denoted in seconds. The values are obtained by matching the agent activity records with the call records.
- *handling_time* denotes the total time spent by the agent on handling the call, including any relevant work. It is defined by $talk\_time + after\_call$.

In table 3.3 we illustrate an example of the agent activity data that is used, followed by the column descriptions.

id	user_id	activity_id	start	end	shrinkage	agent
20032893	*5088*	*3*	*2014-01-02 15:09:07*	*2014-01-02 15:10:15*	*0*	6929
20032898	*6664*	*3*	*2014-01-02 15:09:09*	*2014-01-02 15:10:37*	*0*	8494
20032905	*6666*	*16*	*2014-01-02 15:09:17*	*2014-01-02 15:09:18*	*0*	8496

Table 3.3: Example of agent activity records

- *id* denotes the id unique to the activity log record in consideration.
- *user_id* denotes the id unique to the user account that is used by the agent for the activity log record in consideration.
- *activity_id* denotes the id of the following logged activities:
  - *2* logging off
  - *3* taking calls
  - *7* paid break
  - *8* unpaid lunch break
  - *16* wrapping up the call after hang-up
  - *35* logging in
  - *39* meeting with senior
  - *40* meeting with supervisor
  - *41* additional work
  - *42* additional work
  - *43* unpaid break
  - *44* logged off by system
  - *61* outbound calling
  - *71* administrative activity, consider logged off
- *start* denotes the date and time that the considered activity has started.
- *end* denotes the date and time that the considered activity has ended.
- *shrinkage* describes whether the activity can be considered productive to incoming calls, where 1 denotes that the activity is labeled as shrinkage and 0 denotes that the activity is indeed productive to incoming calls.
- *agent* denotes the agent id that performed the activity in consideration.

This data set consists of 1.639.803 rows. Again, we strictly consider records of working days between 08:00 - 20:00 and are left with a total of 1.619.587 rows, which is a decrease of 1.2% of the total number of activity records.

Next, we take necessary data processing steps to aggregate the data. This is done such that we have aggregated information of the number of incoming calls, abandonments, number of agents, which agents are working, average handling time and service level per interval of 15 minutes. We describe how we go from the data sets given in tables 3.2 and 3.3 to an aggregated data set. The steps are described per column of the aggregated data frame.

- *number of calls* is obtained by applying the *pandas* functions $groupby()$ and $Grouper()$ on the call records data frame. By counting every record and specifying the frequency as 15 minutes we obtain the number of calls per 15 minute interval.

- *number of abandoned calls* is obtained by initially creating a column in the call record data frame which denotes 1 if the call is abandoned and 0 if the call is not abandoned. The records of abandoned calls are identified by locating the missing values in the *answered* column. Next, we are able to group the call records per 15 minutes and sum over the newly created *abandoned* column. This gives us the total number of abandonments per interval.

- *average handling time* is obtained by using the created column *handling_time* shown in 3.2. We again group per 15 minutes and calculate the mean of this column, which results in the average handling time per interval.

- *number of agents* is calculated by the following procedure. For every agent we iterate through every day. For each day we identify the shift by the earliest start time and latest end time. Next, we locate the 15 minute intervals that fall within that considered shift. We name these intervals *core intervals* and use $core_s$ to denote the *core intervals* of shift s. For all *core intervals* that are located, we add one agent to the column n_agents. Next, we want to calculate the fraction of the first and last intervals that the agent is working in. We do this by selecting one interval previous to and one interval after $core_s$, we name these intervals f and l, respectively. For every shift s we calculate $head_s = \frac{f_s^{end} - start_s}{d_f}$ and $tail_s = \frac{end_s - l_s^{start}}{d_l}$, where

  - the numerators are expressed in minutes,
  - f_s^{end} denotes the ending time of the first interval that comes prior to $core_s$,
  - l_s^{start} denotes the starting time of the first interval that comes after $core_s$,
  - $start_s$ describes the starting time of shift s,
  - end_s describes the ending time of shift s,
  - d_f describes the duration of interval f in minutes,
  - d_l describes the duration of interval l in minutes.

  For every shift s we add fractions $head_s$ and $tail_s$ to column n_agents on intervals f_s and l_s, respectively.

- *agents* are determined in a similar fashion as the number of agents. We create an empty list for every interval in the *agents* column. Next, we append the agent id of the considered agent to all records located by the *core intervals*. After this we append the agent id to the intervals f_s and l_s, in which the considered agent is working for a fraction of the intervals.

- *service level* is calculated by first creating a new column *meets_threshold* which denotes 1 when the considered call was answered within the threshold of 20 seconds and 0 if not. The calls which are picked up within 20 seconds are identified by locating the records for which *answered* - *queue_time* $\leq$ 20 seconds. Next, we are able to group by 15 minutes and sum the number of times this threshold is met. After obtaining the number of times the threshold is met per interval we calculate the realised service level per interval. We create the service level column by calculating $SL_i = \frac{n\_meets\_threshold_i}{n\_calls_i}$ for each interval i.

In table 3.4 an illustrative example of the aggregated data table is given.

start	end	n_calls	n_abandoned	aht_min	n_agents	agents	sl
2014-01-02 14:30:00	*2014-01-02 14:45:00*	*145*	*11*	*6.011940*	*45.570000*	*[9425, 8189, 8723, 8493, 6940, 9424, 8393, 934...*	*0.289655*
2014-01-02 14:45:00	*2014-01-02 15:00:00*	*120*	*2*	*4.916384*	*34.222222*	*[9425, 8189, 8723, 8391, 8493, 6940, 9424, 839...*	*0.741667*
2014-01-02 15:00:00	*2014-01-02 15:15:00*	*152*	*0*	*4.936075*	*40.007778*	*[9425, 8189, 8588, 8723, 8391, 8077, 8493, 694...*	*0.796053*

Table 3.4: Example of aggregated data per 15 minutes

3.2 Arrivals

After data processing steps we first analyze the call arrivals to see whether we can find any patterns on different time scales.

To get an idea of how the call volume behaves on a larger timescale we illustrate the weekly call volume for the entire year in figure 3.1. We only want to consider entire weeks for this illustration, so we leave out week 1 and week 52. We see significant declines in call volume for several weeks (e.g. weeks 7, 16 and 43). These declines may be coincidental or due to an underlying seasonality pattern. Take for example weeks 30 - 34, in which we observe a decrease of approximately 3500 calls (∼12%). This decrease may be explained by the large number of people that yearly go on holiday during the month of August. However, since we conduct this research based on one year worth of data we are not able to confirm that this is indeed a seasonality pattern. We focus on short time scale prediction but are aware that this potential seasonality pattern may introduce extra variability in the intra-day patterns.

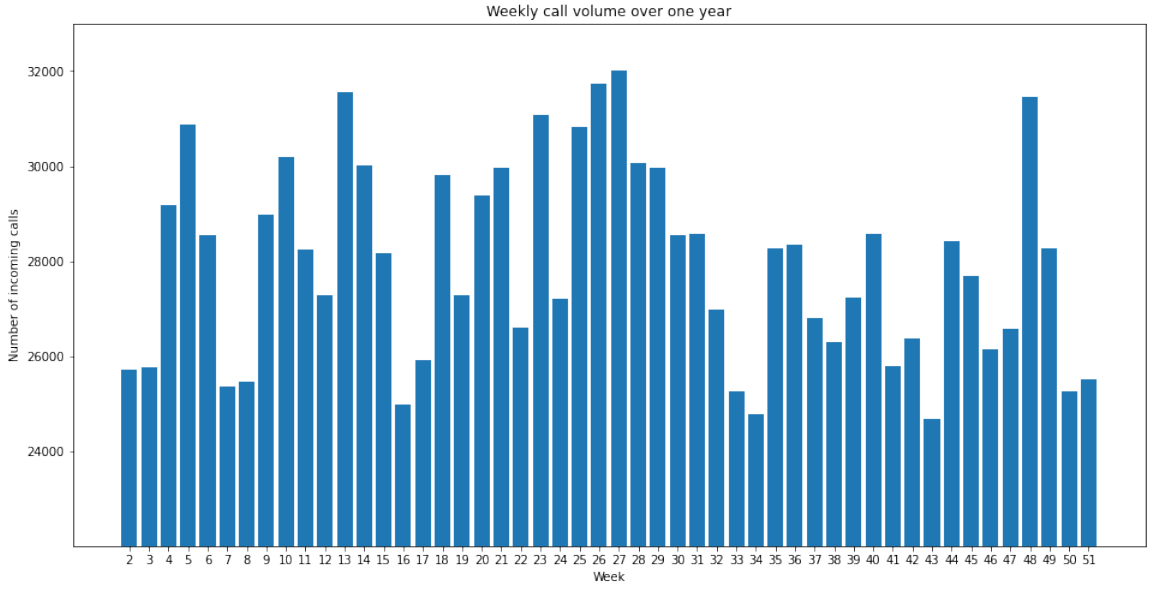

Figure 3.1: Weekly call volume

In figure 3.2 a clear intra-day trend can be observed for the call arrivals. The number of incoming calls seems to drastically increase between 08:00 - 09:00, after which we can see a clear dip and quick recovery in the incoming call volume around lunch time (12:00 - 13:00), followed by a slow decline which eventually turns into a drastic decrease between 16:00 - 17:00. There seems to be relatively little difference between the trends of the different days. However, on Mondays there are significantly more calls compared to the rest of the week. This may be explained by customers that may have encountered problems or intended calling during the weekend. These customers intent on calling as soon as possible, which is on Monday.

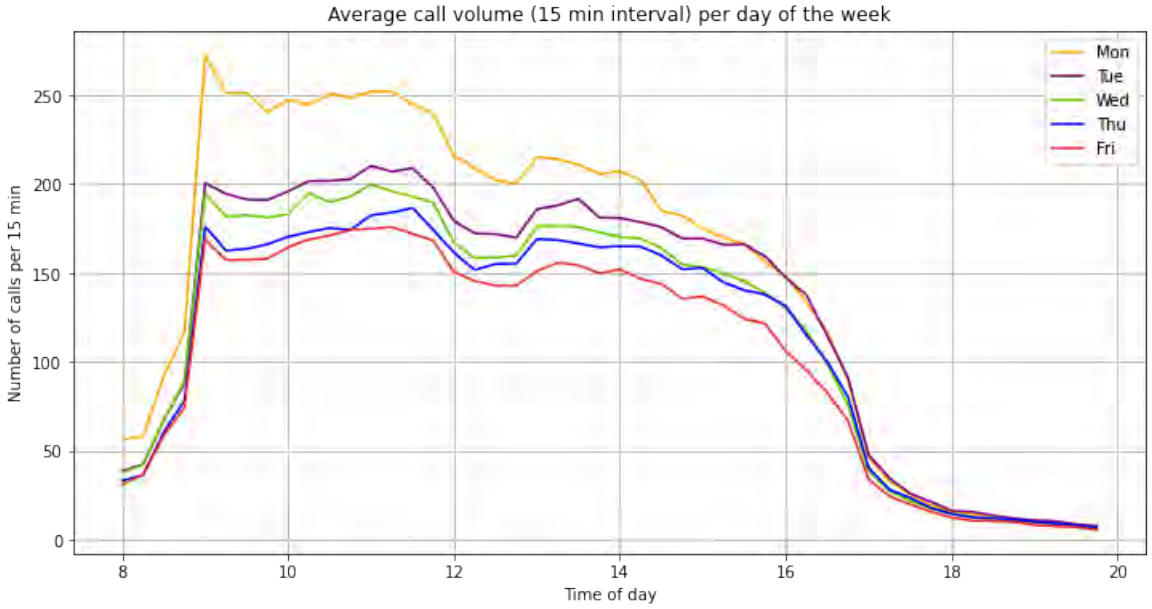

Figure 3.2: 15 min interval call volume per day of the week

We see that the average call volume on the other weekdays (Tuesday to Friday) is quite similar. Because the average call volume is calculated over a full year of data, we illustrate the variability in terms of standard deviation. In figure 3.3 we plot the call volume including the standard deviation for Mondays and the average of the other days. We see that the variability is larger on Mondays compared to the rest of the week. Additionally, we observe that the variability increases in bandwidth when the call volume increases and that the bandwidth becomes smaller when the call volume decreases.

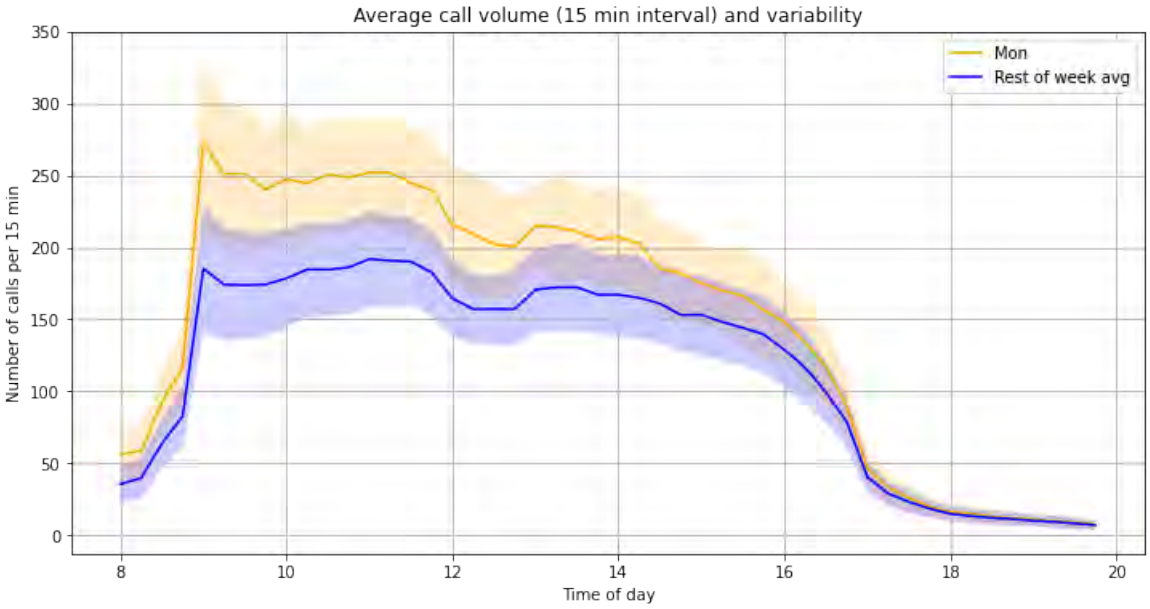

Figure 3.3: 15 min interval call volume with variability

We want to see how the variance increases and decreases relative to the average value of call arrivals. We do this by plotting the *variance-to-mean ratio* over all intervals in figure 3.4. We can see clear fluctuations in the variance-to-mean ratio, which means that the variance does not move in proportion to the mean. It is interesting to see that the variance-to-mean ratio is large during the early hours and small during the later hours of the day, while the average call volume is relatively low during both moments. This means that there is larger variance during the early hours, which can also be seen in figure 3.3. Furthermore, it is observed that for most intervals the variance-to-mean ratio is larger than 2. This tells us that the variance is often much larger than the mean. We know that for a Poisson distribution holds that $E(X) = Var(X) = \lambda$, which clearly is not the case for our data. This gives us the expectation that the Erlang C and Erlang A models may not model this call center data properly, since Poisson distributed call arrivals are one of the assumptions of the Erlang models.

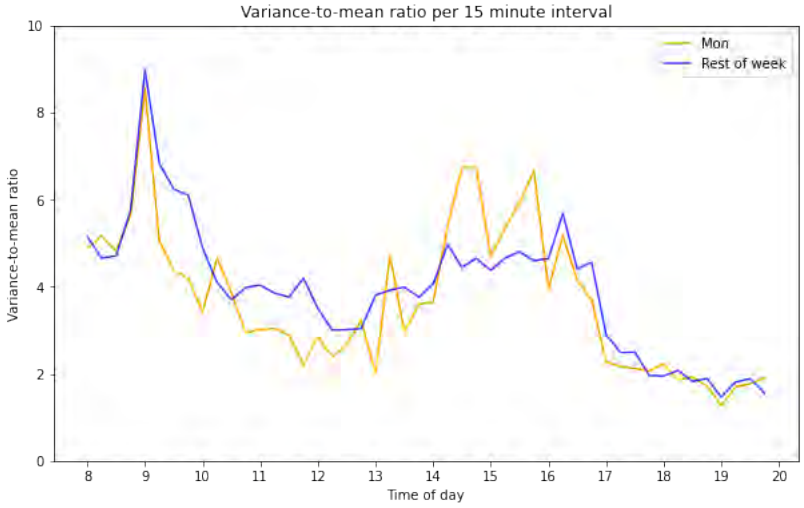

Figure 3.4: Variance-to-mean ratio per interval over entire year

3.3 Agents

The number of agents available to handle calls is an important factor when it comes to the service level. As one can imagine, if more agents are available to take a call, the call may be picked up quicker. Not all agent activities are considered to be productive when we want to focus on handling calls. Activities that do not contribute to picking up or handling calls are named shrinkage activities. Examples of shrinkage activities are e-mailing, receiving training and taking breaks. Initially, we have determined the number of agents per interval including shrinkage activities. This means that we look at the number of agents, without taking non-productive activities into account (e.g. breaks). This number will be specified as the scheduled number of agents. We are aware that it is not equal to the actual number of scheduled agents because of last-minute changes and unexpected absence that we cannot retrieve from the data. We do however consider this a decent approximation of the number of scheduled agents. To first see whether there is any difference in the number of agents on different weekdays, a plot of the distribution of the number of agents is given in figure 3.4.

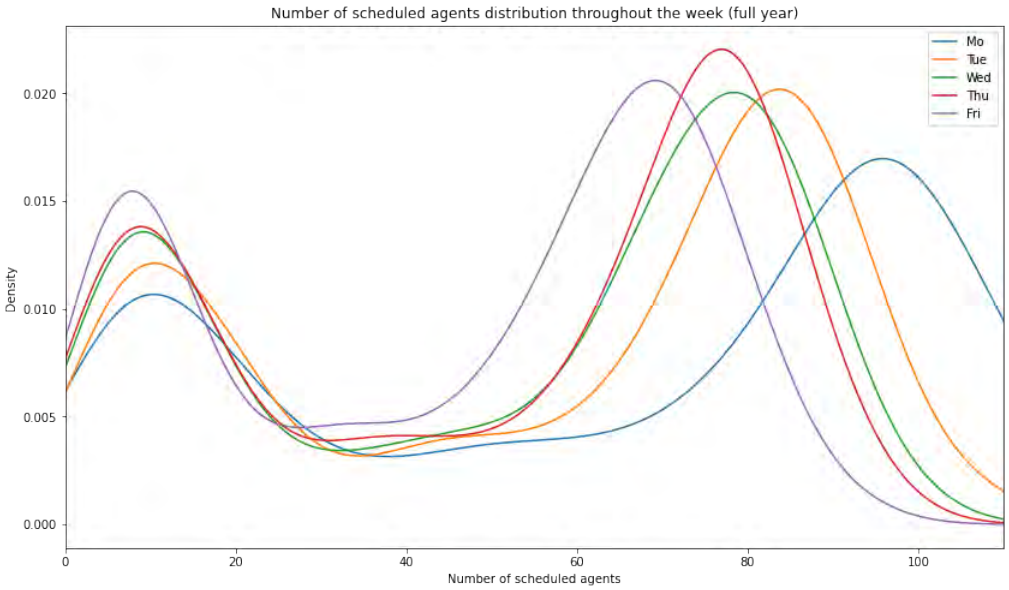

Figure 3.5: Scheduled number of agents distribution

We can observe that for a significant fraction of time the number of agents scheduled is around 10 and 80. It is interesting to note that generally less agents are scheduled as the week passes. The largest difference seems to be on Mondays compared to the rest of the week. We see significantly more agents scheduled on Mondays. This may be because of the schedulers anticipating a larger call volume observed on Mondays. We want to take a closer look at the scheduled number of agents and zoom in throughout the day.

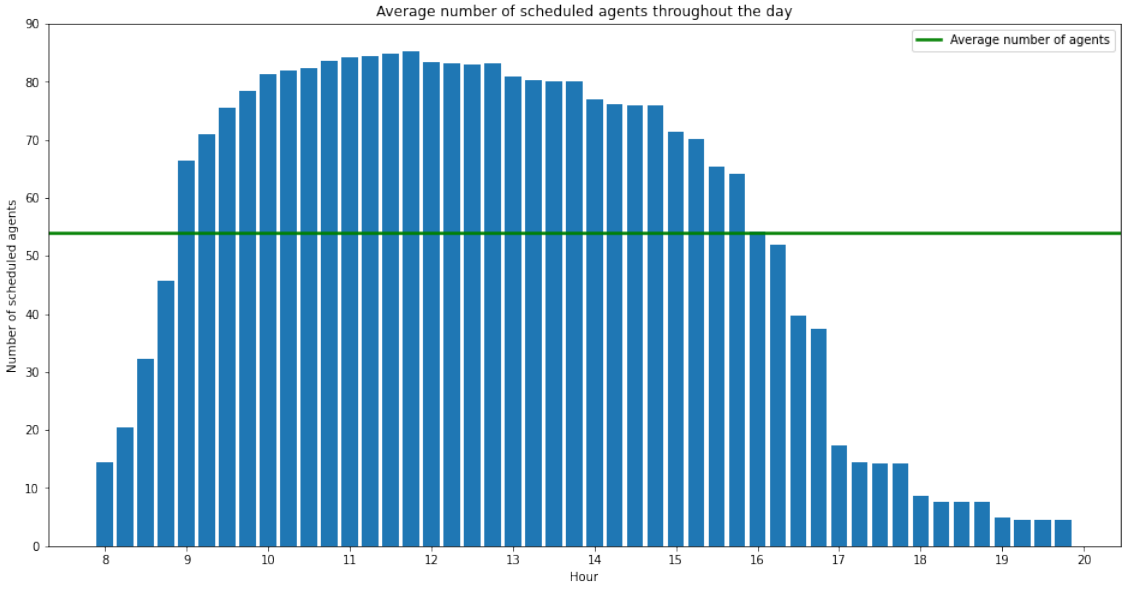

Figure 3.6: Average number of scheduled agents throughout the day (full year)

In figure 3.6 we see the average number of scheduled agents throughout the day. The green line indicates the average number of scheduled agents for all intervals, which is approximately 54. We can see clearly that during the first hour of the day a small number of agents are scheduled. The same can be observed for hours after 17:00, during which we see a gradual decrease in the number of scheduled agents for each hour until 20:00. We see that on average a relatively large number of agents is scheduled during hours 09:00 - 17:00. This is to be expected since we have seen in figure 3.2 these are the most busy hours regarding call volume. It seems that the peak around 10 agents observed in figure 3.6 can be explained by the relatively small number of scheduled agents during the hours 08:00 - 09:00 and 17:00 - 20:00. Similarly, the peak around 80 agents can be explained by the number of agents scheduled during 09:00 - 17:00. During 12:00 - 16:00 we see a gradually decreasing pattern similar to the one observed after 17:00. It is noticeable that even though we clearly see the average number of agents vary per interval, the decision makers seem to schedule agents per hour for certain parts of the day.

Now it is interesting to see how the number of agents compares when we take the shrinkage activities into consideration. We determine this number by counting the number of agents per interval that are either *taking calls* or *wrapping up after hang-up*, which are the only non-shrinkage activities. This means that agents that are doing any other activities (e.g. meetings) are not counted. We shall refer to this number as the *effective number of agents*. In figure 3.7 we see that compared to the distribution in figure 3.5 there are significantly less number of agents per day of the week. We see relatively small difference between the peak around 10 in figure 3.7 compared to the peak around 10 in figure 3.5. However, we see a clear shift to the left for all days for the second peak. We now see that for a significant fraction of time there are roughly 50 agents rather than the original 80 agents that we have observed before. Although it can be observed that the differences between the days have become smaller, we can still see a similar pattern compared to the number of scheduled agents. The number of effective agents decreases throughout the week and on Mondays there are significantly more effective agents compared to the rest of the week.

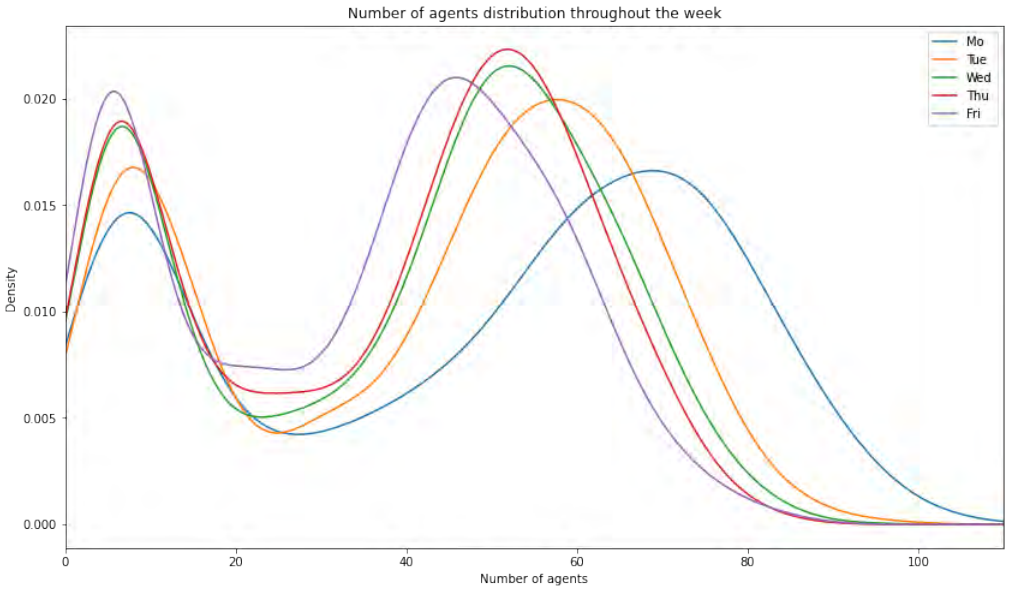

Figure 3.7: Effective number of agents distribution

Now we again zoom in and take a closer look at the effective number of agents throughout the day in figure 3.8. We keep the distinction of the days to see whether there are any differences between the patterns and how the average number of agents compare.

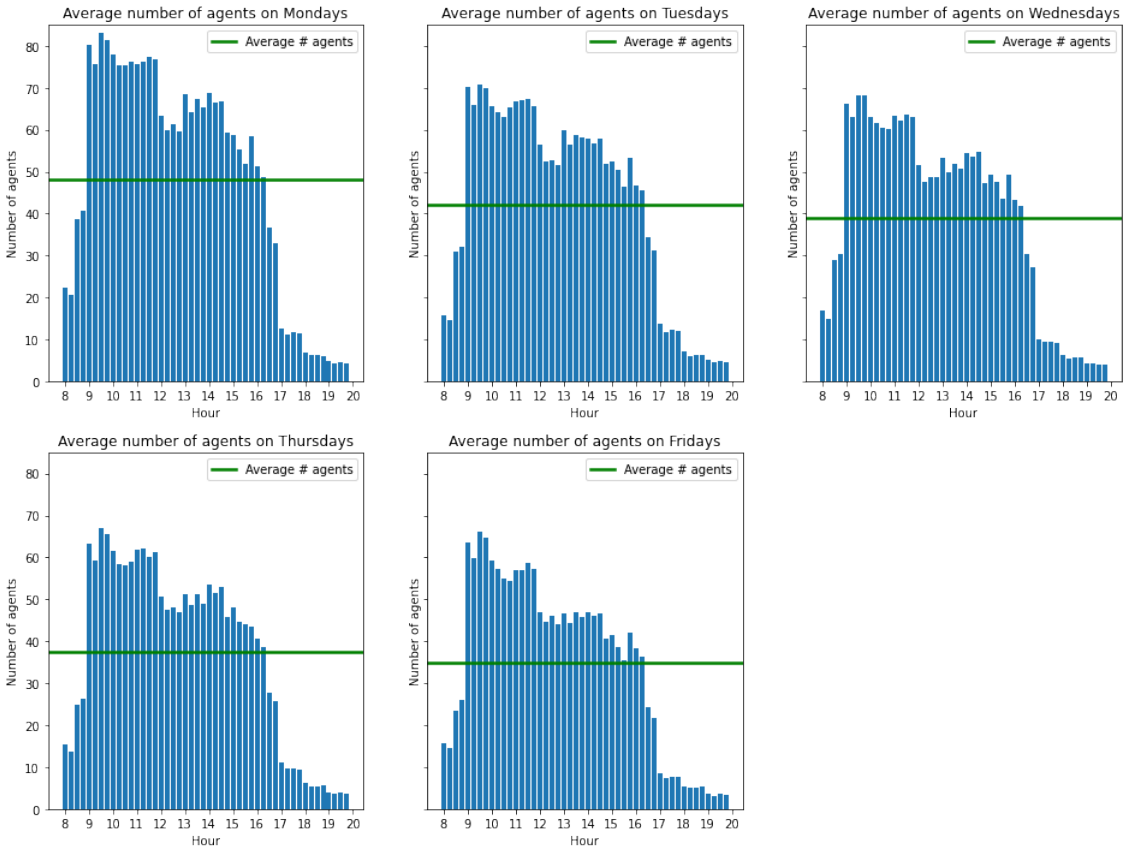

Figure 3.8: Average number of agents across the day per day of the week

In figure 3.8 we see five plots representing the average number of effective agents throughout the day, for each day of the week. The average number of effective agents for the considered day is represented by a green horizontal line, which is below 50 consistently and even below 40 for the second part of the week. This is a clear decrease compared to the average of 54 scheduled number of agents over all days which we have observed in figure 3.6. Next, we see a clear pattern where early in the day the number of agents is relatively low and drastically increases between 08:00 and 09:00, which is similar to the pattern that we have seen before. However, after 09:00 we see a different pattern which we have not observed for the scheduled agents. The number of effective agents declines slowly until a larger decrease occurs around lunchtime (11:30-12:30). After lunchtime we see a small recovery in the number of agents, after which a drastic decrease occurs from 16:00 on. Every day of the week seems to follow a similar pattern. We can also see significantly higher number of agents on Mondays compared to the rest of the week, which was observed before in figure 3.7. Lastly, similar to our observation for the scheduled agents, we see a lower number of agents in the early and late hours of the day. For the larger part of the day we see an higher amount of agents. This explains the peaks of approximately 10 and 50 agents in the distribution shown in figure 3.7. The general shape of the average number of agents in figure 3.8 is similar to the shape shown in figure 3.6 except for the mentioned

moments in which we see clear drops in number of agents.

It is clear that shrinkage activities play an important role in the number of agents and the intra-day patterns. We want to understand the shrinkage activities to determine whether they are known in advance during scheduling or whether they are unknown prior to the considered working day. For this reason shrinkage activities are split in two categories, *scheduled shrinkage* and *unscheduled shrinkage*. In figure 3.9 we illustrate break activity records of two different activity types, *unpaid lunch break* and *paid break*. There is a clear difference between the two break types. We see that the lunch breaks have a consistent length of around 30 minutes and most of the time take place at 12:00, 12:30, 13:00 or 13:30. For the paid breaks however, we see much variability in the duration. Most of these breaks seem to be outside of the hours 12:00 - 14:00. This can be explained by the lunch breaks that take place at this moment. It seems reasonable that if an agent takes a break during hours 12:00 - 14:00, it will generally be a lunch break rather than a paid break. During hours 08:00 - 10:00 and 17:00 - 20:00 we see very little breaks since there are significantly less agents working. Additionally, it is expected that agents will not take breaks shortly after their shift starts or shortly before their shift ends. Outside of these hours the timing of paid breaks seems to be rather random. Based on this analysis it seems reasonable to classify that lunch breaks are a form of scheduled shrinkage activities whereas paid breaks are a form of unscheduled shrinkage activities.

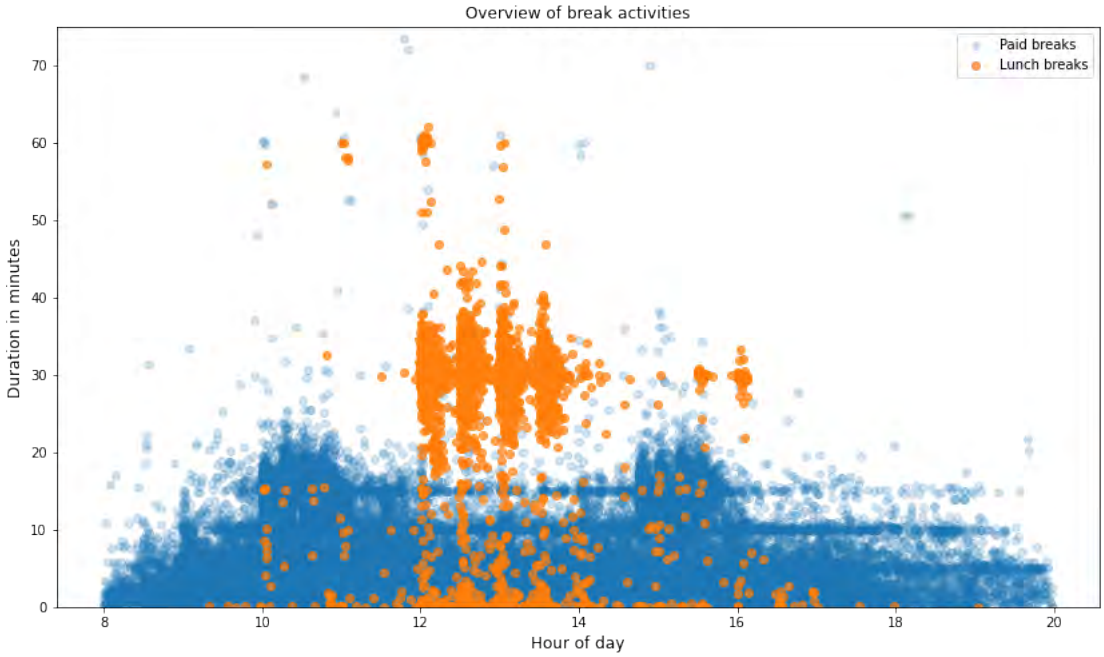

Figure 3.9: Overview break activities throughout the day

In figure 3.10 we show a similar plot but now of the activity *meeting with supervisor*. We see that the meetings are held at various times but not later than 17:00. This can be explained by the supervisor being absent as well as the low number of agents after 17:00. Intuitively one would expect that meetings are a form of scheduled shrinkage. However, in

figure 3.10 it can be observed that that the duration of these meetings varies substantially, ranging approximately from one minute to one hour. It can be argued that meetings of very short duration are not scheduled beforehand but rather occur in a more natural way depending on several factors. Moreover, in practice it is not usual to schedule activities of very short duration. We think that it is reasonable to classify activity *meeting with supervisor* with a duration smaller than 5 minutes as unscheduled shrinkage. Whereas *meeting with supervisor* with a duration larger than or equal to 5 minutes is considered scheduled shrinkage. We analyze all shrinkage activities in similar fashion to classify them accordingly.

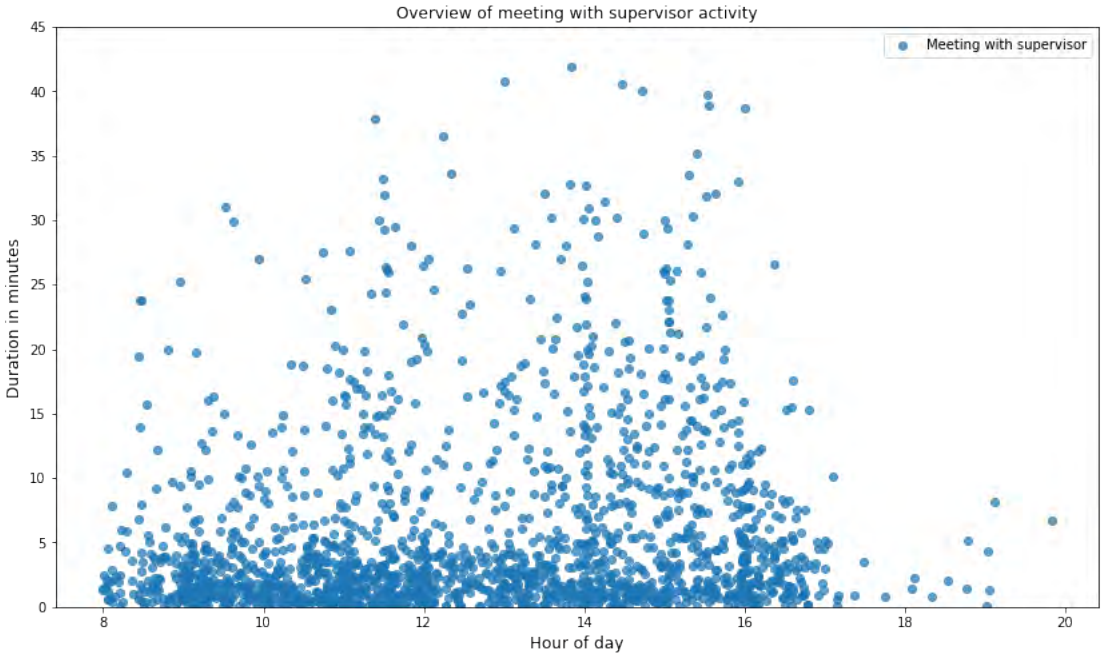

Figure 3.10: Overview meeting with supervisor activity

3.4 Average handling time

The average handling time (AHT) is an important metric which indicates how long, on average, it takes for an agent to handle a call. This time includes the so-called wrap-up time, which contains any work that is necessary to finish up the considered call after breaking contact with the customer. As one can imagine, the average handling time largely impacts the availability of agents since it measures how efficient the agents are when it comes to handling calls. The availability of agents may in turn impact the service level significantly. We analyze whether there are any intra-day patterns when it comes to the average handling time of agents per interval of 15 minutes. For example, increasing average handling times throughout the day may indicate possible shift-fatigue patterns. In figure 3.11 we illustrate a box plot that summarizes the average handling time per 15 minutes throughout the day for calls.

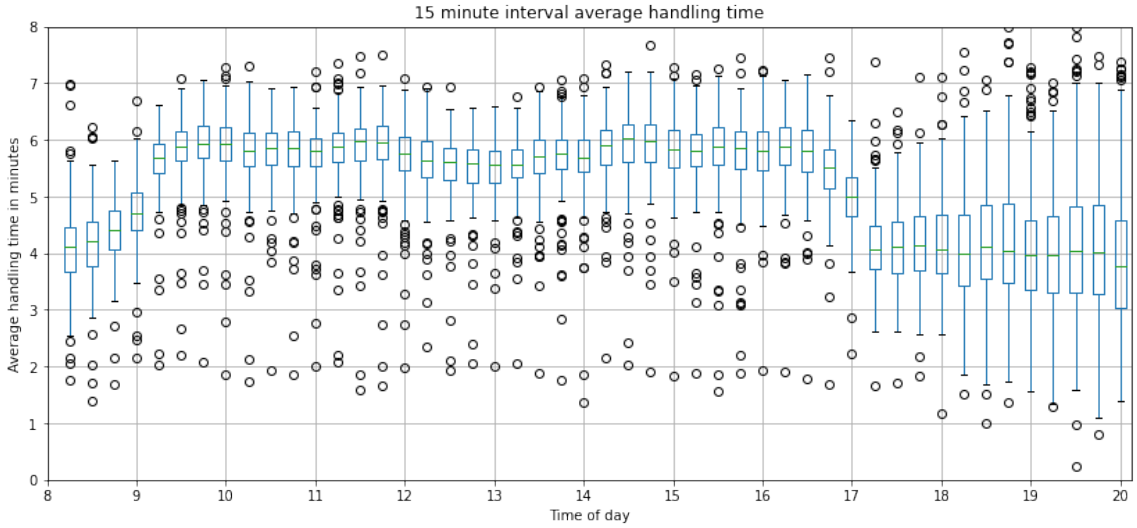

Figure 3.11: Average handling time per 15 min interval

We can observe that the AHT is relatively short in the early part of the day but quickly becomes longer and stabilizes after 09:00. Next, we see a small decrease and recovery around lunchtime. Between 16:30 and 17:00 we see the AHT quickly become shorter again, after which it stabilizes. The AHT seems to vary considerably more towards the end of the day compared to the rest of the day.

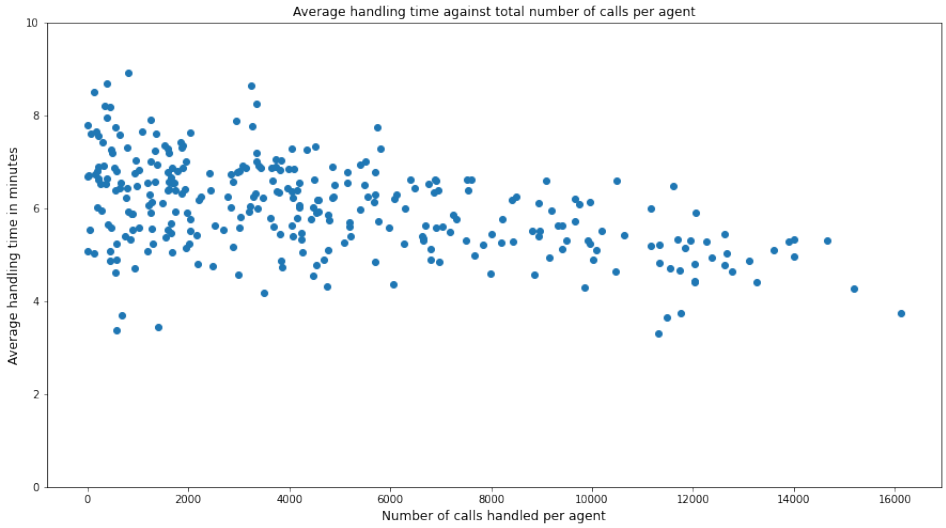

Figure 3.12: Average handling time against number of calls handled per agent over entire year

We want to compare the AHT per agent to understand whether the availability of different agents largely impacts the speed at which the calls are handled. We do this by the

use of a scatter plot where each data point represents one agent, this plot can be seen in figure 3.12. We can observe large differences in AHT between the agents, varying between approximately 3 and 9 minutes. It seems that the agents with more calls handled have slightly lower AHT than agents with fewer calls handled.

3.5 Service level

In this subsection we perform analysis of the service level (SL) so that we can find potential patterns in the service level behaviour. After this analysis we can have a better understanding of how we are able to predict the service level.

As mentioned in subsection 3.1 we focus on the eight most common queue types. Each queue requires a certain type of service, which is specified by an identification number. In figure 3.13 we illustrate the service levels per service type, we see that the differences in service level between these service types are rather small, varying from 0.61 to 0.68. The service level does not vary drastically per service type. We believe this is a result of scheduling since decision-makers would likely aim to schedule agents such that the team is able to handle each queue, which results in a consistent service level across different queues.

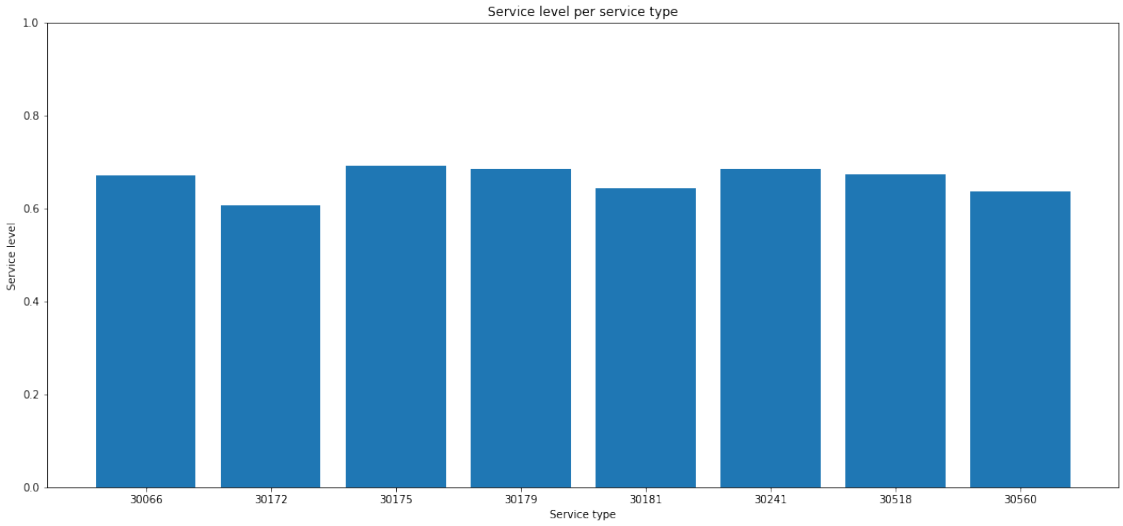

Figure 3.13: Service level per service type

Next, we examine the SL on a daily basis to understand if there are any intra-day patterns and to get an idea of the variability. In figure 3.14 we see a box plot of the SL per 15 minute interval. It is observed that that there is large variability in SL throughout the day. The SL seems to be quite often close or equal to 1.0 during the early hours of the day, after which we see the SL decline. Especially after 12:00 we see the SL decline more frequently, where it seems to be below 0.8 for the larger part of the time. We previously observed in subsections 3.2 and 3.4 that the call volume and average handling time decrease around

12:00. Intuitively, one would expect that decreasing call volume and average handling time would lead to an increase in SL. Less incoming calls and less time spent per call should result in more idle agents, which allows for more quick responses to calls. However, in subsection 3.3 we saw that the number of agents declines at the same time due to the lunch break, which may explain the decrease in SL.

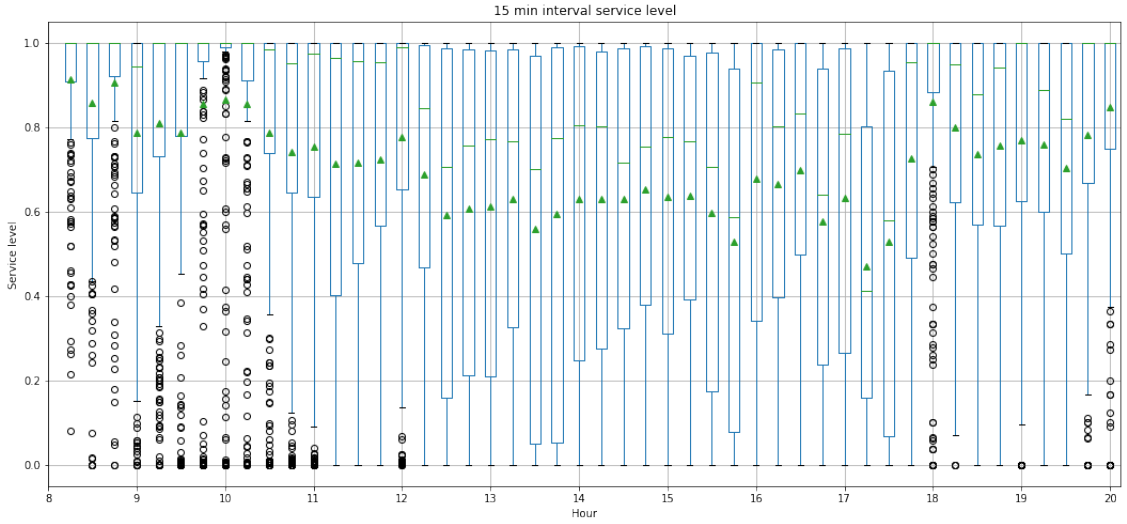

Figure 3.14: Boxplot of 15 min interval service level including average (green triangles)

During the final part of the day the SL seems to increase again whereas its variability becomes relatively limited, yet larger compared to the variability during the early part of the day. In subsections 3.2 and 3.4 we have seen significantly smaller call volumes and shorter average handling times during the early and late hours of the day, which explains the high SL values during these intervals. It is interesting to note that in figure 3.8 we have observed drastically lower number of agents before 09:00 and after 17:00. Furthermore, we see the variability in average handling time increase after 17:00, which explains the larger variability in SL after 17:00 compared to the early hours.

Now that we have a better understanding of the intra-day service level behaviour we take a look on a larger timescale. In figure 3.15 we see the weekly service level throughout the entire year. Note that the weekly service level is below 0.8 for most weeks. We see a clear decline during weeks 20-25 (below 0.4). Furthermore, weeks 12, 17, 18 and 48 stand out as well due to their low service level (below 0.5). As mentioned in section 3.2 we are aware that underlying seasonality patterns may have an effect on the service level. It can be seen that the service level indeed is different for specific weeks. We are not able to confirm whether these declines are due to an underlying yearly seasonality pattern since we only have one year worth of data. We do however keep this possible effect in mind when evaluating our prediction accuracy.

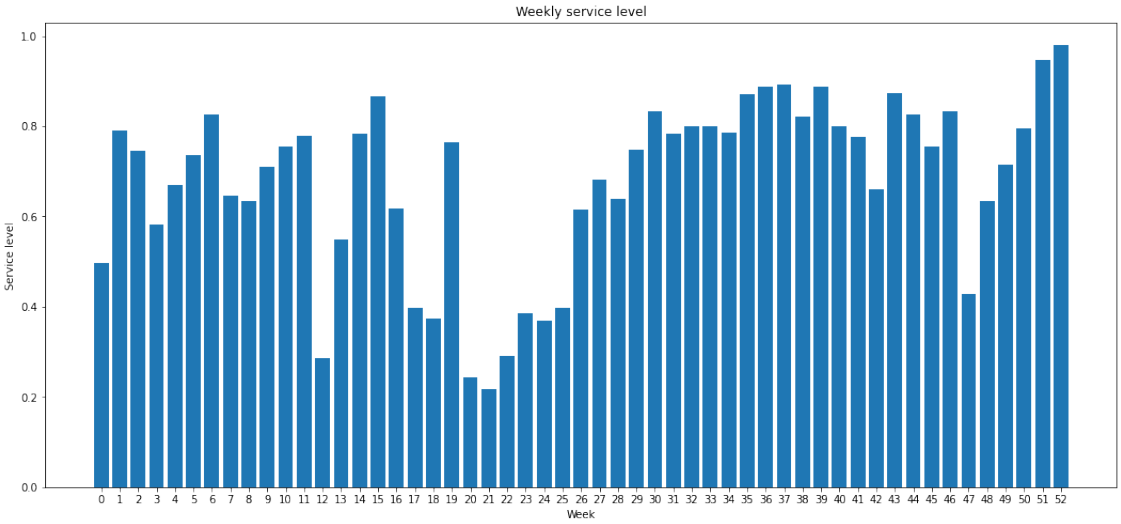

Figure 3.15: Weekly service level

4

Methodology

In this chapter we describe the different steps taken in developing a machine learning method to predict the service level. In section 4.1 we provide an overview of the process and give a brief description of each step. In section 4.2 we discuss the feature analysis process and our initial selection of features. In section 4.3, we explain the machine learning model that we consider and its implementation. In section 4.4 we discuss our methods for model validation, feature selection as well as hyperparameter tuning. At last, we explain the choices made for the performance measures in section 4.5.

4.1 Machine learning pipeline

Our machine learning pipeline is visualized in figure 4.1. We initiate the process with *data extraction & cleaning*. This step is concerned with collecting the necessary data, replicating it to the desired environment and cleaning it so it may be ready for analysis. Since we have received a clean dataset in the form of *csv* files, which are conveniently imported to a *Python* environment, this step is rather insignificant in our situation. However, for the sake of completeness we have included this step in our process. During *data analysis* we analyse the data such that we get a general understanding of how the call center operates (e.g. opening hours, total number of agents etc.). In addition, we may find certain patterns or interesting behaviour during this phase. Based on this understanding we are able to make decisions during the *data preparation* step, which is concerned with preparing the data in such a manner that it is cleaned and ready to be used for features. Next, we perform *feature extraction*, during this phase we extract several features from the data which serve as potential input for our model. This step is based on prior analysis and understanding of the data as well as additional feature analysis. During *model evaluation and validation* we evaluate how our model performs and get an understanding which features to drop and select. Besides feature selection, we are looking to tune the hyperparameters of the model, such to optimize the performance while controlling overfitting. This is done by the use of 5-fold cross validation. We now have a final, trained model which is ready to be used. We perform a *final evaluation* by the use of an unseen test set to get an idea of how well the model performs.

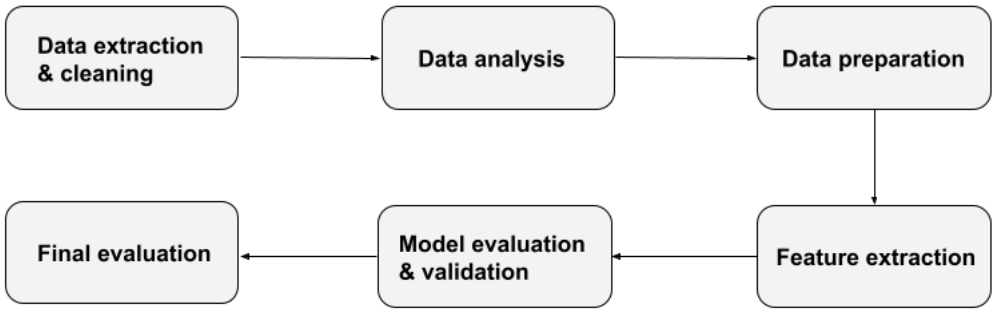

Figure 4.1: Schematic overview of the machine learning pipeline

4.2 Feature analysis & extraction

To select an initial group of features we analyse relations between potential features and the service level. In addition, we look for relations that are not necessarily obvious when directly compared to the service level. This is done by analysing interaction between the features and using our understanding of the dataset gained by prior analysis. The features are expressed per interval, and can generally be extracted by aggregating the data per interval of 15 minutes. For time-sensitive features we ensure to create a forecast or estimator based on historical data. The extracted features are categorized in three groups, *basic features*, *intermediate features* and *advanced features*. This categorization is made for several reasons. The distinction between the feature groups is made based on the complexity of their extraction combined with the anticipated predicting value the feature will bring to the machine learning model. Creating three groups of features allows us to understand how performance is impacted by the number and type of features used as input for the machine learning model. Based on interim results of the models, which are based on the feature groups, the best performing model is selected. The features considered in the selected model will endure a feature selection process which is based on the importance of the feature for making predictions. We elaborate on the feature selection process in section 4.4.

4.3 Machine learning model

4.3.1 Ensemble learning methods

Ensemble learning methods are methods that aim to achieve better prediction performance by combining predictions of multiple models. There are different approaches to ensemble methods. We will explain the three most popular techniques, *bagging*, *stacking* and *boosting*.

Bagging stands for bootstrap aggregation. This technique aims to obtain diverse ensemble members by varying the training data. It often consists of multiple models based on a decision tree algorithm. Each model is trained on a different sample of the same training dataset. The predictions of these models are then combined. This is usually done by simple methods, like averaging or voting. The key of bagging is in the unique training

data samples for every ensemble member. This variation results in distinctive and capable models.

Stacking (stacked generalization) is an ensemble technique which aims to have a diverse group of ensemble members by training different model types on the same training dataset. The predictions of the models are then combined by the use of a model. The distinctive models are named *first-level learners* whereas *second-level learner* refers to the combiner model. It is possible to add more layers of models and have a third-level or fourth-level learner for example. However, it is most common in stacking to have two layers of models. It is common to use a linear model to combine the predictions and to have more complexity in the first layer of models.

Boosting is an ensemble technique that focuses on improving previous prediction errors. The models are fit on the training data and added to the ensemble sequentially. This is done in such a way that every model tries to improve particularly on the worst predictions of the previous model. The models are usually based on a simple decision tree algorithm, which are referred to as *weak learners*. The predictions of these weak learners are combined by simple methods, similar to bagging, like averaging or voting. However, the combining method may be weighted by assigning a larger weight to each model according to their performance. The idea is to combine the weak learners such to create a so-called *strong learner*. There are several techniques when it comes to boosting. One of the widely-used techniques is *gradient boosting* [9]. Gradient boosting can be used for classification as well as regression tasks. It looks to optimize the boosting process by minimizing the loss function of the model. This is done by adding weak learners using gradient descent, hence the name 'gradient' boosting.

4.3.2 eXtreme Gradient Boosting

The machine learning technique that we propose is eXtreme Gradient Boosting (XGBoost), which is implemented by [7]. XGBoost is a particular implementation of Gradient Boosted Decision Trees (GBDT), which is a tree-based ensemble learning method that uses gradient boosting as its ensemble technique. The general process of tree-based gradient boosting is illustrated in figure 4.2 [15].

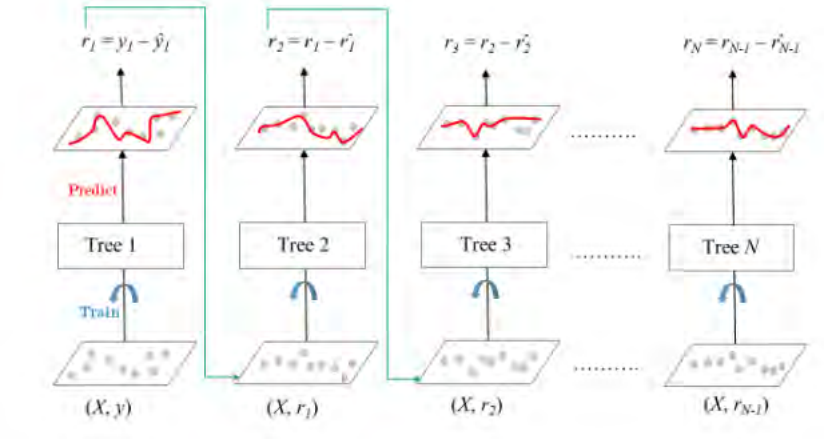

Figure 4.2: Schematic overview of tree-based gradient boosting

The weak learners are decision trees with the same structure, each usually consisting of 8 to 32 leaves. The ensemble learning method creates an initial prediction of the output variable, usually by calculating the sample mean. For each sample prediction the residual is calculated. The first iteration residuals are then used as leaves in the first decision tree. The leaves that contain multiple residuals are replaced by the mean value of those residuals. The learning algorithm then iterates and creates a new prediction by adding the residuals of the previous decision tree to the previous predicted value. To scale the result of the decision tree and counter over-fitting, the predicted residuals of each tree are weighted by a learning rate. Based on the second prediction made by the first decision tree, new residuals are calculated. These residuals are placed in the leaves of a second decision tree that is constructed. The same process proceeds, such that a new decision tree is constructed until a maximum number of trees is reached, or that new decision trees do not reduce the residuals. This results in a final prediction of the output variable. In algorithm 1 the general outline for GBDT is given.

Algorithm 1 GBDT

`Initialize` $f_0(x) = argmin_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1, ..., M$ **do**
  for $i = 1, 2, ...N$ **do**
    $r_{im} = -\left[\frac{\delta L(y_i, f(x_i))}{\delta f(x_i)}\right]_{f=f_{m-1}}$
  end for
  `Fit a regression tree to` r_{im} `obtaining` $R_{jm}, j = 1, 2, ..., J_m$
  for $j = 1, 2, ..., J_m$ **do**
    $\gamma_{jm} = argmin - \gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$
  end for
  `Update` $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
end for
`Output` $\hat{f}(x) = f_M(x)$

XGBoost provides parallel tree boosting which allows for more efficient utilization of computer power compared to sequential tree boosting. Furthermore, XGBoost constructs the trees in a unique manner. It aims to choose the best decision nodes based on the largest gain in similarity score compared to the initially selected decision node. This gain is influenced by a regularization parameter λ which controls overfitting. In this research all XGBoost models are implemented by the use of the *XGBoost Python Package*.

4.4 Model evaluation & validation

The input features are expressed per intervals of 15 minutes. The dataset will be split in 70% for training and 30% for testing. All three models will be trained and tested on the same training and dataset. Based on interim results of the three models, we select the best performing model for feature selection and hyperparameter tuning. Gradient boosting allows for a relatively straightforward feature selection method. During the construction of the boosted trees, it stores how many times each feature is used for important decision nodes. The more a feature is used for key decision nodes, the higher its relative importance will be. To calculate feature importance over a single tree, each decision node is considered. For each decision node the improvement in impurity is calculated. Weighing this improvement by the number of observations gives us the importance. For regression, which is what we are concerned with, the impurity metric used is the *mean squared error*. The final feature importance is then determined by calculating the average for each feature, over all boosted trees within the model.

With the use of the feature importance scores, several subsets of the input features can be selected. This is done based on several thresholds. By taking multiple thresholds that correspond well to the feature importance, we are able to iteratively evaluate model performance with each subset of input features. We validate the performances of several models which are characterized by their subset of input features. This is done by the use of 5-fold cross validation. Based on these performances we can choose which subset of features to include in our final model. After the feature selection process we perform hyperparameter tuning. XGBoost has many hyperparameters which can be tuned. We select a subset of hyperparameters, namely *n_ estimators, max_ depth, max_ child_ weight, subsample, colsample_ bytree* and *eta*. We tune the hyperparameters using 5-fold cross validated *Grid Search*. After feature selection and tuning we again test our selected model and analyse its final performance.

4.5 Performance measures

There are several ways to measure the prediction performance. We need to select the performance measures most suitable for our situation. Since we predict the SL per interval of 15 minutes we need to realize that not every predicted SL is equally important. This is due to the fact that the call volume can vary drastically between intervals. It is common practice for call centers to monitor the service level throughout the day (per interval) as well as over an entire day. Since an interval with high call volume will have larger impact on the service level over the entire day than an interval with low call volume, we want to weight the interval service levels according to their call volumes. Additionally, looking at the interval level, in a trade-off between better accuracy in SL prediction for high call volume intervals and less accurate SL prediction for low call volume intervals or vice-versa, one would naturally select the prior. This is because accurately predicting the SL during high call volume intervals would lead to more customers receiving better service, which is eventually what matters.

Keeping this in mind, we introduce a weighted performance measure that we are going to use to evaluate our baseline models and machine learning model performance. *WAE*, which stands for *weighted average error*, gives the mean absolute prediction error weighted by the call volume. WAE is defined as

$$WAE = \frac{\sum_{i=1}^{n} C_i |y_i - \hat{y}_i|}{\sum_{i=1}^{n} C_i} \tag{4.1}$$

where,

- C_i is the call volume of interval i,
- y_i is the realised service level over interval i,
- $\hat{y}_i$ is the predicted service level over interval i
- and n is the number of intervals.

Additionally, we consider a second performance measure that is not weighted. The reason for adding this performance measure is because of its intuitive and straightforward nature as well as the additional insight that may be gained by comparing it to the errors measured by the weighted performance measure. The second performance measure that we use is *Mean Absolute Error (MAE)*, defined by

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|. \tag{4.2}$$

5

Feature analysis

In this section we analyse potential features and their relation to the service level. We categorize the input features for the models in three groups. *Basic features* are straightforward and relatively simple to obtain. *Intermediate features* are features that are relatively easy to obtain but may capture valuable effects in addition to the basic features. *Advanced features* contain all features that may not be as straightforward as basic and intermediate features and are more difficult to obtain. For each feature category it is described how the features are defined, how they are obtained and why they are selected.

5.1 Basic features

We introduce five basic features which are based on the input variables for the Erlang A model. To capture the call volume we introduce the feature C which is specified per 15 minute interval. It is obtained by counting the number of call records with an arrival time within the considered 15 minute interval. Since the ground truth values of the call volume are not known at the moment of scheduling, a forecast should be implemented to anticipate the expected call volume. We assume that there must be at least a full week in between the moment of scheduling and the week to be scheduled. Therefore, we will use a one-week margin for calculating historical metrics that will be used for our input features. However, in a realistic situation, at the moment of training the model historical data is available. This means that for model training the actual call volume is known. Therefore, we implement a simple forecasting method specifically for the call volume input feature of the test set.

First we define intervals to be *similar* when they share the moment they take place within a day and the day of the week. Then the forecast is determined by calculating the Moving Average of the call volume over four *similar* intervals prior to the interval to be forecasted. Using a one-week margin in this case, means that to predict the SL of intervals in week w, the forecast will be calculated over weeks $w-5$ to $w-2$, such that $w-1$ is used as a one-week margin for scheduling.

Secondly, we want to capture how long it takes to handle calls. We do this by introducing the feature H, which denotes the average handling time per 15 minute interval. Similar

to the call volume, we calculate an average handling time estimate for the test set input feature which is implemented in the same manner as the call volume forecast. The average handling time for interval i is defined as

$$H_i = \frac{\sum_{j \in \mathbb{C}_i} t_j + u_j}{d_i} \tag{5.1}$$

where,

- t_j denotes the call time in seconds of call j,
- u_j denotes the wrap-up time in seconds of call j,
- $\mathbb{C}_i$ is the set of calls that arrived in interval i,
- d_i describes the duration in seconds of interval i.

The third basic input feature that we introduce is A, which represents the number of agents working per 15 minute interval. Note that at the moment of scheduling it is not known exactly how many agents will end up working during every interval. This is due to unexpected absences (e.g. sick calls) and unscheduled shrinkage (e.g. short meeting with supervisor) during the shift. Thus, the scheduled number of agents should be used as an input feature rather than the actual number of agents. Although we do not have access to the original schedule that corresponds to the data set, we are able to approximate the scheduled number of agents per interval based on the agent activity data set. By the use of the analysis in section 3.3 we are able to identify what shrinkage activities are scheduled or unscheduled. We define A for interval i as $A_i = |\mathbb{A}_i| = |\mathbb{A}_i^s| - \frac{S_i}{d_i}$, where

- $\mathbb{A}_i$ is the set of scheduled agents to effectively work in interval i after removing scheduled shrinkage,
- $\mathbb{A}_i^s$ is the set of all scheduled agents to work in interval i without including scheduled shrinkage,
- S_i^s is the total duration of scheduled shrinkage activities in seconds within interval i,
- d_i describes the duration in seconds of interval i.

Additionally, we want to capture the workload per interval and see whether it may impact the service level. We introduce basic feature L which estimates the total duration of call(related) work that needs to be handled by the agents within the specified time frame. We define L over each interval i by $L_i = \frac{C_i * H_i}{d_i}$ where d_i denotes the duration of interval i expressed in seconds. Note that L is an estimated input feature if C and H are estimated.

The last basic input feature, based on the input variables for the Erlang A model, is the estimated patience of the customers denoted as P. Patience can be defined as the time a caller is willing to wait before abandoning the queue. Customers that have less patience than 20 seconds will abandon the call early and are counted as calls that have not met the 20 second threshold. This means that little patience has a direct negative effect on the SL. Additionally, customers that have large patience will stay in the queue for a longer

time, which impacts the queuing system. This can be of value in predicting the SL. We want to estimate this patience and use it not only as an input variable for Erlang A but as an input feature for our ML model as well. We estimate the patience by the use of the *Kaplan-Meier estimator*, which is a technique of estimating the survival function S_t by $\hat{S}(t) = \prod_{i:t_i \leq t}(1 - \frac{d_i}{n_i})$, where $S(t)$ is the probability of surviving longer than t, t_i is a time at which at least one call was abandoned, d_i is the number of abandoned calls at time t_i, n_i the number of calls until time t_i. Using this probability we calculate the density between each abandon moment, after which the patience is calculated by determining the expectation. The patience input feature P is calculated for each day of the week over the entire data set. Note that in reality the patience for interval i cannot be known prior to interval i. However, since we estimate the patience per day of the week over the entire dataset and not over each interval, we find it reasonable to believe that the 'unfair' advantage for the benefit of our model is negligible.

In figure 5.1 we observe a correlation heatmap of the basic input features mentioned. We see that SL is slightly negatively correlated with H, C and L. One may intuitively anticipate lower SL with larger call volumes as well with longer AHT. We see minimal negative correlation between SL and A. Perhaps at first sight, one would expect the service level to be positively correlated with the number of agents, since more availability of agents should allow the call center to respond to calls more quickly. However, this small negative correlation can be explained by the schedulers anticipating incoming call volumes. This means that more agents are scheduled when larger call volumes are expected. This idea can be confirmed since we indeed observe that C and A are highly positively correlated.

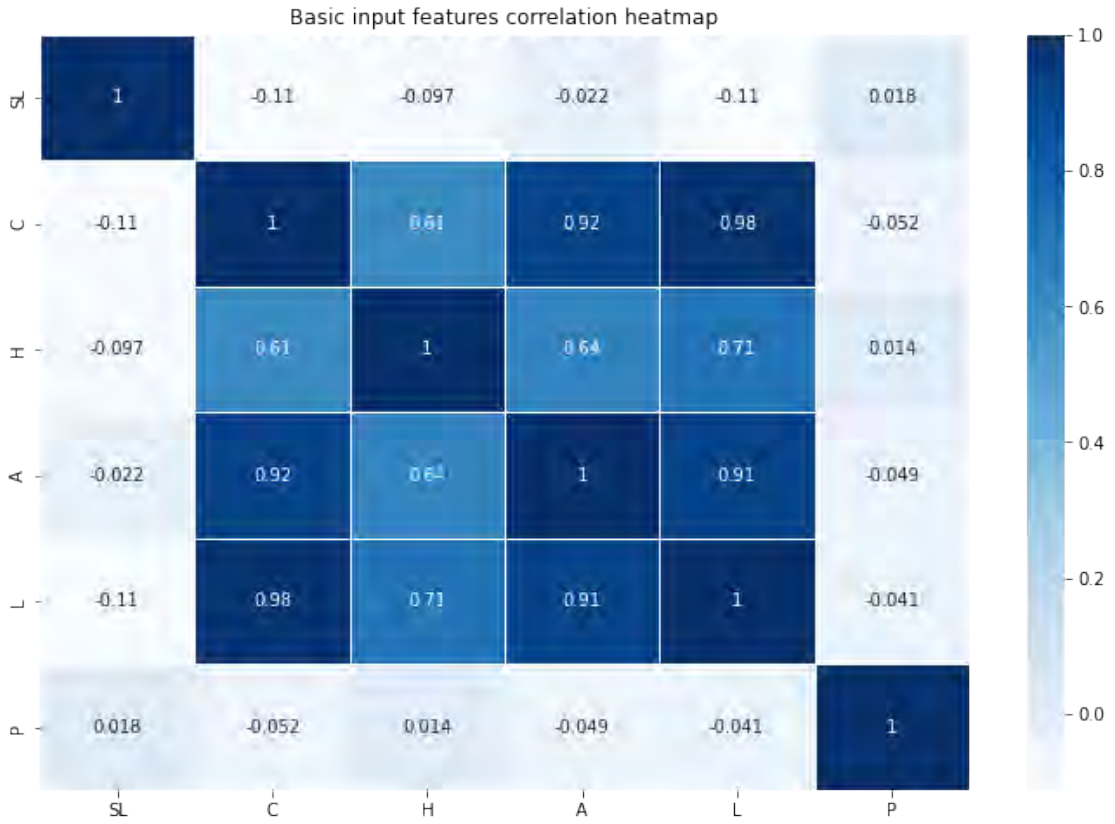

Figure 5.1: Correlation heatmap of basic input features

Furthermore, we see that P has very little correlation with all other features. There is positive correlation between C and H. More incoming calls will include more complicated calls, which may lead to more transfer and consulting calls as well. These internal lines increase the total handling time significantly, which may explain the positive correlation. Additionally, we notice that A and H are positively correlated as well. This means that when more agents are actively working the average handling times become longer, which may be due to the prominent correlation between number of agents and call volume. Finally, we note that L is highly positively correlated with the other three basic input features. This is to be expected since L is derived from the product of C and H.

In section 3.5 we have observed that around lunchtime (12:00) there is a decrease in call volume, average handling time, number of agents while there is a decrease in SL as well. Similarly we have seen during the early and late intervals small call volumes, short average handling times and a small number of agents but now with increasing SL. This may be an explanation for the small negative correlation between A and SL. Smaller C and H increase the SL while smaller A would drive the SL down. When the input features take smaller or larger values simultaneously due to interactions which cause the SL to decrease or increase, the correlations between the basic input features and the SL may be compensated. Especially for A since this input feature is highly dependent on C.

The correlations and corresponding patterns give us reason to believe that the basic input features may contain meaningful information to predict the service level. The input features H, A, C, L and P are initially selected. However, we keep in mind the large interactions between the features, especially the large correlation between C,A and L.

5.2 Intermediate features

Next, we want to capture the impact of time on the service level. We do this by creating two time related input features. I denotes the interval of the considered day. The interval duration is 15 minutes and a full working day is from 08:00 to 20:00, this means there are 12 hours and four intervals within each hour, which gives 48 intervals per day. Thus, I can take on values $\{1, 2, ..., 47, 48\}$. To capture the impact of time on a weekly basis we introduce the time related feature D, which specifies what day of the week it is. D can take on values $\{0, 1, 2, 3, 4\}$ which represent Monday, Tuesday, Wednesday, Thursday and Friday, respectively. With the possible value combinations of these two features the machine learning model can interpret information as 'the third interval (08:30 - 08:45) on a Monday. Since the focus of this paper is on short time scale prediction, large time scale indications (e.g. week number, month) are not considered as input features.

In figure 3.12 we see that agents can have very different average handling times, varying between 3 and 9 minutes. These differences can be due to several factors like generally slow or fast agents, inexperienced agents that are slow and need training, specific agents that have a lot of unscheduled shrinkage activities (e.g. sudden breaks) etc.

To capture this type of agent behaviour we introduce the third and last intermediate feature a_x, which specifies whether agent x is working (and available) per interval. However, as described in subsection 5.1, we do not know exactly which agents are working during each interval at the moment of scheduling due to unexpected absence and unscheduled shrinkage activities. We want to use an estimation of the scheduled agents for each interval. Therefore, a_{xi} takes on value 1 for each interval i if agent $x \in \mathbb{A}_i = \mathbb{A}_i^s \setminus \mathbb{A}_i^a$ and 0 otherwise, where

- $\mathbb{A}_i$ is the set of scheduled agents to effectively work in interval i after removing scheduled shrinkage,
- $\mathbb{A}_i^s$ is the set of all scheduled agents to work in interval i without including scheduled shrinkage,
- $\mathbb{A}_i^a$ is the set of agents that have scheduled shrinkage activities during entire interval i and thus are absent.

For a_x we choose to use a one-hot encoded representation. This means that we create 307 columns, one for every unique agent id. For convenience and tracking purposes we re-define the agent ids from 1 to 307 based on the ascending order of their original id.

5.3 Advanced features

Finally, we introduce four advanced input features. These features are less straightforward and can give the ML models an edge compared to the Erlang models.

In section 3.4 we have seen that agents which transfer calls more often to colleagues and make more consulting calls have larger average handling times. We want to see whether the number of transfers and consulting calls can be used as a predicting factor for the service level. When using predictive input features we must be aware to use information that is known at the moment of scheduling. Similar to the forecast, we will use a one-week margin for calculating historical metrics that will be used for our input features. To create a new feature that incorporates the number of transfers and consults per agent, we first introduce a new metric named r (consulting & transfer ratio). r calculates the 4-week Moving Average of the number of consults and transfers made divided by the total number of calls handled by the considered agent.

We define r for agent x and interval i as $r_{xi} = \frac{X_x^o + X_x^t}{X_x}$, where

- X_x is the number of calls that were routed to agent x over four weeks prior to the week of interval i,
- X_x^o is the number of consults agent x has made over four weeks prior to the week of interval i,
- X_x^t is the number of transfers agent x has made over four weeks prior to the week of interval i

By the use of r we create a new input feature R that is calculated per interval. We define R for interval i as $R_i = \frac{\sum_{x \in \mathbb{A}_i} r_{xi}}{|\mathbb{A}_i|}$, where $\mathbb{A}_i$ is the set of scheduled agents to effectively work in interval i. For the intervals that miss R values (e.g. for the first week of the data), we insert the average over the entire data set.

Because of the basic features, we have all necessary input variables to use both Erlang C and Erlang A as a prediction method of the SL. This means that the result of the Erlang models can serve as an estimated SL input feature. We introduce input features E^c and E^a for interval i by the use of input features C_i, H_i, A_i and P_i.

Since we are considering a multi-skill call center, it is of importance what type of service is required for each call and whether agents capable of offering that service are available. To capture this effect we create an input feature that measures the variety and depth of skills that the effectively scheduled team offers. We re-define the different queue id's from 1 to 8, where 1 is the queue with least call volume and 8 is the queue with the largest call volume. This input feature is denoted as Q_s for every required service s. Q_{si} describes the total capacity of agents able to offer service s and are present in interval i. The capacity of an agent is distributed over the queues that the agent is able to handle, weighted by the call volume. In table 5.1 we illustrate an example of the Q feature. Finally, we provide an overview of all input features in table 5.2

interval	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
40	*0.5*	*1.4*	*1.5*	*3.6*	*8.6*	*13.7*	*17.3*	*33.4*
41	*0.5*	*1.3*	*1.2*	*3.3*	*7.6*	*12*	*15.4*	*29.7*
42	*0.5*	*1.2*	*1.3*	*3.3*	*7.6*	*12*	*15.4*	*29.8*

Table 5.1: Example of Q input feature representation

Description	**Symbol**
Basic features	
Call volume	C
Average handling time	H
Number of agents	A
Load	L
Patience	P
Intermediate features	
Interval	I
Day of the week	D
Agent presence features	a_x
Advanced features	
Consult & transfer ratio	R
Team skill	Q_s
Erlang C SL prediction	E_c
Erlang A SL prediction	E_a

Table 5.2: Overview of input features

6

Results

In this chapter we describe the experiments that we have conducted and evaluate the performance of the machine learning models, which are compared to the performance of the Erlang models. Finally, we provide analysis and give our interpretation of these performances.

6.1 Experiments and performance

We create three different XGBoost models which are characterized by their input features. The *basic model* is a minimal model that considers the *basic input features*, which includes features *C, H, A, L* and *P*. We create this model to get an idea of how the machine learning model performs with input features comparable to the input variables of the Erlang models. The *intermediate model* takes the *intermediate features* as input in addition to the basic features. Features *I, D, P* and a_x are labeled as intermediate features. We create this model to understand how large the impact is of using relatively easy obtainable input features that contain information that is not directly captured by the Erlang models. The *advanced model* extends the *intermediate model* by also considering the *advanced features*, which are input features that are not necessarily straightforward and may be more difficult to obtain. We consider *Q,* E_c*,* E_a and *R* as advanced features. Additionally, the basic features *C, H* and *L* are split per queue.

We use the feature data expressed per interval to create a 70%/30% split for training and testing. Since the output of our model should be between 0 and 1, we have specified the learning task and corresponding objective to be logistic regression. We found that the default parameters of the XGBoost library would cause the models to overfit, especially the intermediate and advanced models. We believe this is due to the complexity introduced by the large number of input features. After several experiments considering all hyperparameters that control overfitting in XGBoost, we found that hyperparameters *n_estimators, eta, max_depth* and *min_child_weight* had the largest impact. Therefore, we choose the initial values for the hyperparameters such that we solve the overfitting problem for the intermediate and advanced models. The values set for the hyperparameters are $n\_estimators = 1000$, $eta = 0.01$, $max\_depth = 4$ and $min\_child\_weight = 1$. The remaining hyperparameters are set as default values. Since the basic model has less

complexity, we choose to set $eta = 0.05$ so that the model may become less conservative. The remaining hyperparameters are the same as for the other models. The in-sample performances of the three models are shown in table 6.1.

	MAE	**WAE**	**Min SL**	**Max SL**
XGBoost Basic	0.255	0.269	0.033	0.992
XGBoost Intermediate	0.206	0.195	0.050	0.992
XGBoost Advanced	0.203	0.195	0.033	0.992

Table 6.1: XGBoost models in-sample performance

Next, we test the performance of our models on the unseen test set. In addition, we test two benchmarks models to compare the performances. The Erlang C and Erlang A model are applied per interval, based on the forecasted values. The performance of the benchmark and machine learning models are shown in table 6.2.

	MAE	**WAE**	**Min SL**	**Max SL**
Test Set	//	//	0	1
Erlang C	0.301	0.374	0	1
Erlang A	0.227	0.266	0	1
XGBoost Basic	0.272	0.303	0.172	0.988
XGBoost Intermediate	0.240	0.239	0.181	0.966
XGBoost Advanced	0.201	0.209	0.337	0.993

Table 6.2: Benchmark and machine learning models test set performance

We see that for all models the prediction errors are larger than 20% in terms of *MAE* as well as *WAE*, which is quite a significant error. As mentioned in section 4.5, we mainly focus on the weighted performance measure *WAE* for performance. We see that even though the basic model performs poorly due to its limited input features, it still has a smaller weighted prediction error than Erlang C. It is interesting to see that the Erlang A model significantly outperforms the basic model, which has input features that contain similar information to the input variables of Erlang A. The intermediate model notably improves the basic model and outperforms the Erlang A model in terms of *WAE*. The difference in error is perhaps significant but not necessarily large by any means. Furthermore, it is interesting to note that although the basic and intermediate model have a smaller *WAE* than the Erlang C and Erlang A model, respectively, the error measured in *MAE* is larger. With the advanced model, which introduces the advanced features, we again see a significant improvement in performance compared to its predecessor. The advanced model performs the best out of the three models and reduces the prediction error of Erlang A with 24% in terms of *WAE*.

After obtaining initial results we select our best performing model XGBoost advanced for feature selection and hyperparameter tuning. First we perform feature selection, which is based on the built-in XGBoost feature importance metric described in section 4.4. Since there are many input features we visualize the top and bottom ten features regarding feature importance in figures 6.1 and 6.2, respectively. There are several features with importance equal to nil, these are not considered for visualization. It can be observed that the most important features as well as the least important features largely consist of agent features, which describe whether the considered agent is working or not.

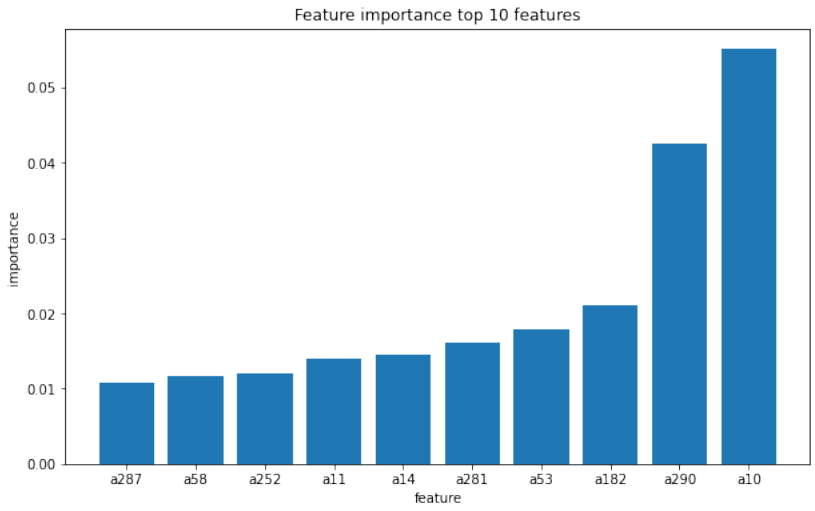

Figure 6.1: Advanced model top ten features based on importance

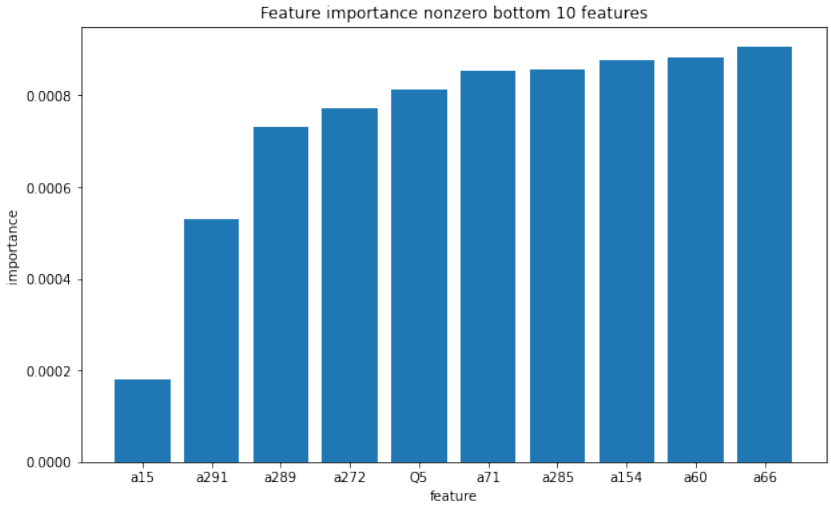

Figure 6.2: Advanced model bottom ten features based on importance

We determine ten thresholds based on the feature importance distribution which are going to be used in our feature selection process. These thresholds are given in table 8.1

in the appendix. The feature selection process proceeds as follows:

1. Iterate over every threshold.

2. For every iteration select a subset of features with feature importance larger than threshold.

3. Perform 5-fold cross validation on the selected feature subset.

4. After all iterations select final feature subset based on the smallest cross validation error.

We find that *threshold 4* gives the smallest cross validation prediction error. We select the features with importance larger than 0.0026 for our final model. We drop 147 features that have an importance lower or equal to the threshold and are left with 146 features, which reduces model complexity significantly. The subset of features for our final model is summarized in table 6.3. The features that were dropped during the feature selection process are shown in table 8.2 in the appendix.

Description	**Symbol**	**Queues**
Call volume	C	$5, 7, 8$
Load	L	$5, 6, 7$
Team skill	Q	$1, 2, 3, 6, 7, 8$
Erlang C prediction	E_c	//
Erlang A prediction	E_a	//
Day of the week	D	//
Interval	I	//
127 agent features	a_x	//

Table 6.3: Features selected for final model

In section 6.2 we take a more thorough look at why certain features are dropped and are considered not sufficiently important for the final model. With these selected features we tune the hyperparameters to obtain our final model. We perform tuning by the use of 5-fold cross validated *Grid Search*, the considered and final parameters are shown in table 6.4.

Parameter	**Range**	**Stepsize**	**Final setting**
n_ estimators	$100, ..., 1500$	1	1500
max_ depth	$3, ..., 6$	1	5
min_ child_ weight	$1, ..., 4$	1	1
subsample	$0.7, ..., 1$	0.1	0.7
colsample_ bytree	$0.7, ..., 1$	0.1	1
eta	$0.001, 0.01, 0.1, 0.3$	//	0.01

Table 6.4: Parameter tuning

We train and test the final model on the same sets used for table 6.2. In table 6.5 we show the performance of our final model including the relative error reduction compared to the performances in table 6.2.

In-sample	**MAE**	**WAE**	**Min SL**	**Max SL**
XGBoost Final	0.154	0.141	0.016	0.996
Test set				
XGBoost Final	0.184	0.195	0.388	0.998
Relative error reduction				
Erlang C	38.9%	47.9%	//	//
Erlang A	18.9%	26.6%	//	//
XGBoost Basic	32.4%	35.6%	//	//
XGBoost Intermediate	23.5%	18.3%	//	//
XGBoost Advanced	8.6%	6.7%	//	//

Table 6.5: Final model performance

We see a significant reduction in prediction error after feature selection and hyperparameter tuning. The final model clearly outperforms the other XGBoost models. It is worth noting that since we have seen an improvement of 6.7% in *WAE* after feature selection and tuning of the advanced model, we can expect the basic and intermediate model to significantly improve in performance after feature selection and tuning as well. Finally, we see a large reduction in *WAE* of 47.9% and 26.6% compared to Erlang C and Erlang A, respectively.

6.2 Performance analysis

We take a closer look at the selected features and features that are dropped during the selection process. We find that the total importance of the 127 agent features is approximately 75%. Therefore, we split the selected features in agent features and non agent features. In figures 6.3 we visualize the importance of the top agent features, which are agent features with larger importance than all non-agent features.

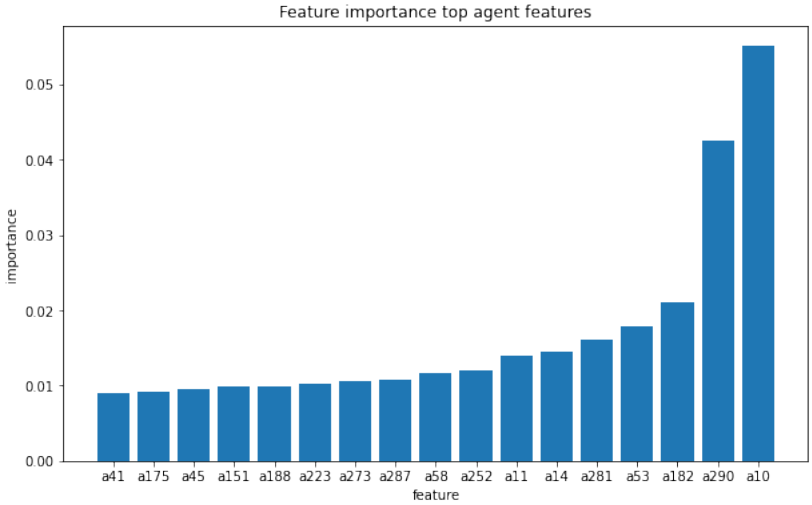

Figure 6.3: Final model top agent features

We want to get a better understanding of which agents are dropped, selected and considered as highly important agents for the final model. We do this by determining the service level per agent over the entire dataset. The service level per agent is determined by calculating the average service level over all intervals in which the agent was present. We create a scatter plot in figure 6.4 that visualizes the service level per agent against the number of intervals the agent has worked in. We distinguish between agent features dropped after feature selection, agent features selected and agent features considered most important for the final model, which are *a290* and *a10*.

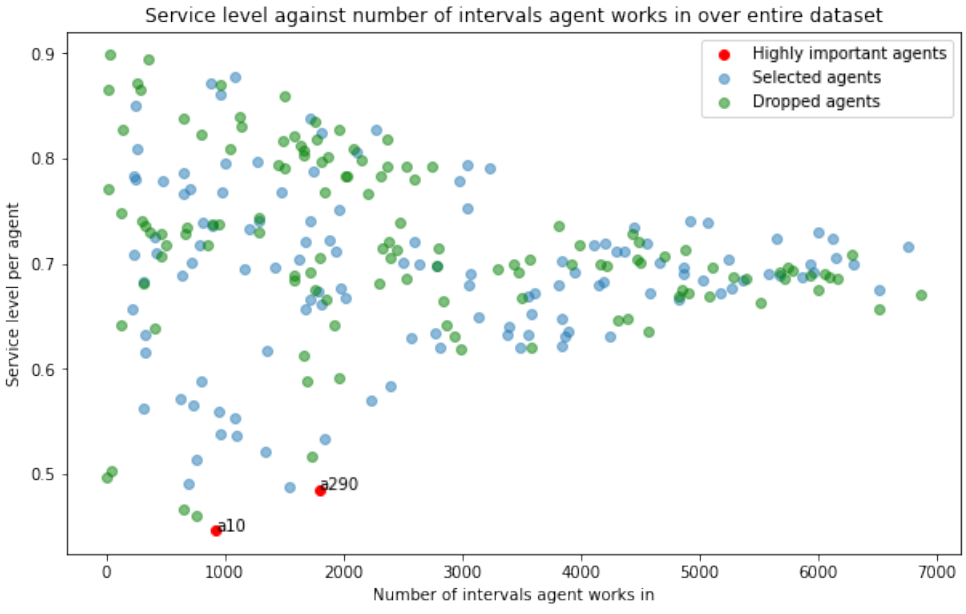

Figure 6.4: Agent features analysis

We see that the most important agents for the final model have a low service level and work in a limited number of intervals. The importance of a feature indicates the relative amount of splits made based on that feature. This means that the highly important agent features are used frequently in the decision trees to base the prediction on. It seems that the model has found these two agents to be good predictors of the service level. The two agents work in enough intervals such that the model has sufficient data to base the predictions on, while having a service level that is sufficiently discriminating. Furthermore, there seems to be no obvious difference between the selected and dropped agents. We believe that the model learns for many agents whether they are working, or not working, based on the presence of other agents. This can be supported by the fact that there is large correlation between the presence of agents, which is shown in figure 8.1 in the appendix. This is due to the typical consistency in schedules throughout time (e.g. largely similar team of agents that always work on Monday mornings). Additionally, a scheduler would generally aim to make balanced teams of agents in terms of experience and performance. This explains the variety of the selected and dropped agents in figure 6.4. In figure 6.5 we visualize the importance of all non-agent features.

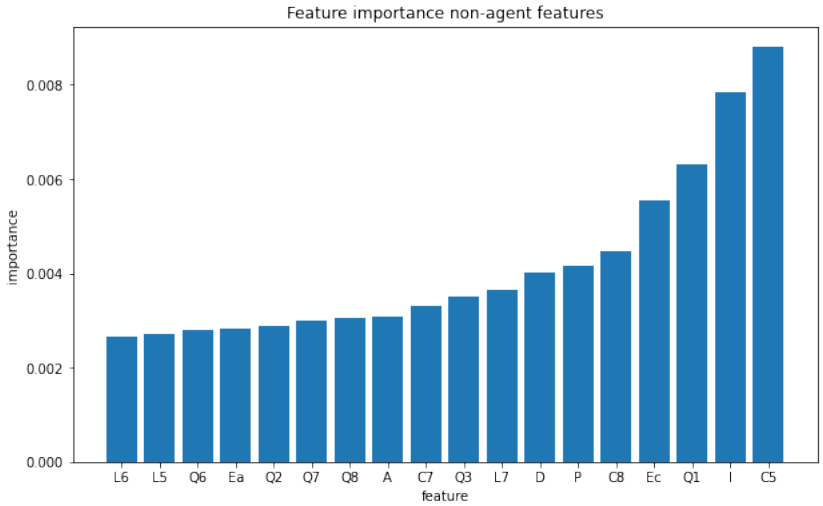

Figure 6.5: Final model feature importance non-agent features

There numerous observations that seem interesting. We see that all average handling time features were dropped during feature selection, which is remarkable. In addition, we see that several call volume, load and queue features were selected while others were dropped. It seems that the model captures the effect of average handling time by the other features. Furthermore, it is interesting to note that all features related to queue seven, except for the average handling time were selected. It seems that the skill capacity to handle each queue is a sufficiently important feature that six out of eight queue features where selected. Lastly, we find it particularly interesting that the Erlang C feature is considered as more important than the Erlang A feature, while the Erlang A model achieves significantly better performance in predicting the service level. We take a closer look at the prediction errors of the Erlang models as well as the Final model. In figure 6.6 we visualize scatter plots of the test set prediction errors of the Erlang C, Erlang A and Final model against

the call volume of the corresponding interval.

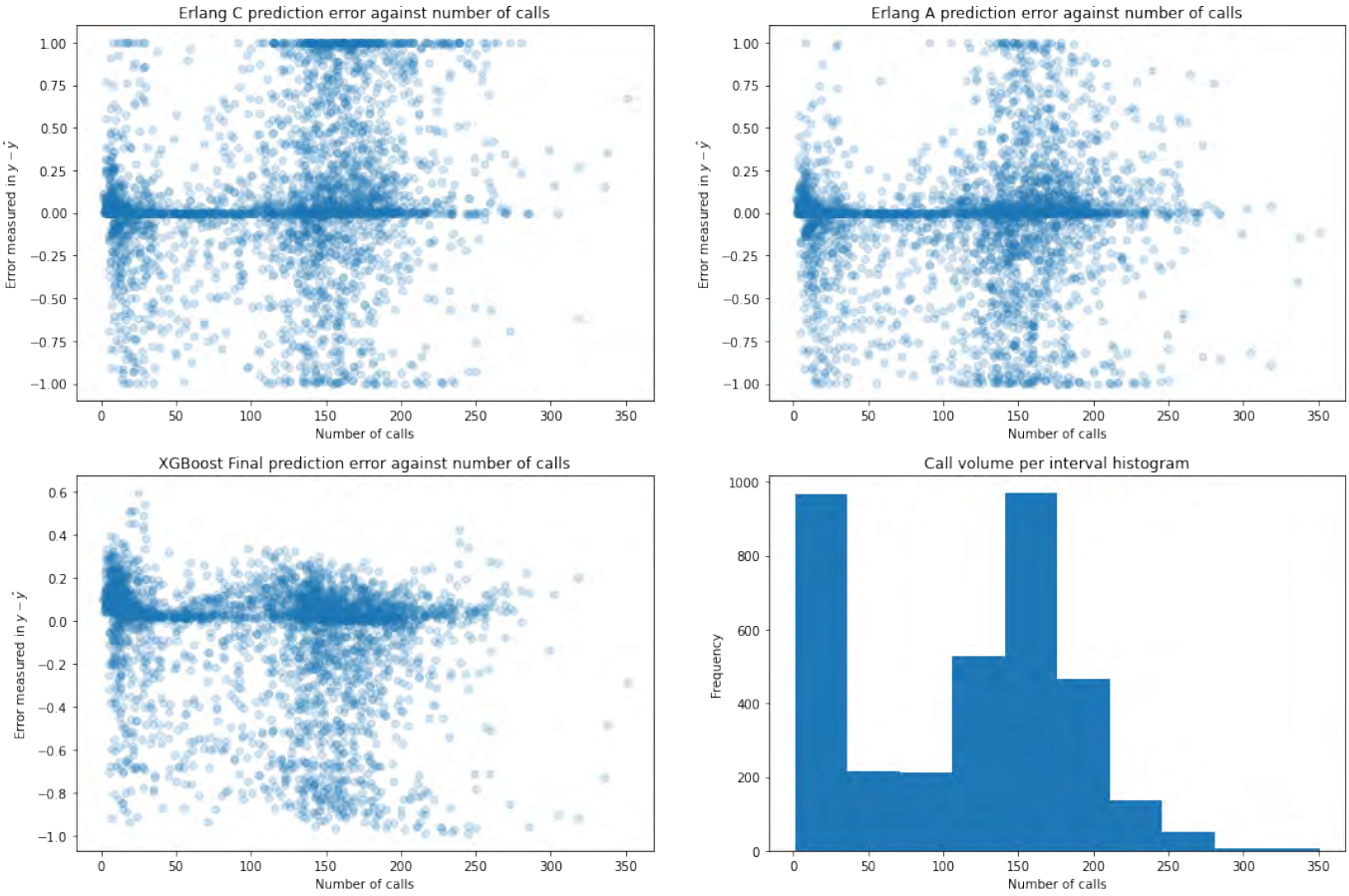

Figure 6.6: Erlang C, Erlang A and XGBoost Final prediction errors against number of calls

We see for all three models that there are many error measurements around 10 and 150 calls. This is due to the large number of intervals that have a call volume of those magnitudes which is visualized in the histogram. Erlang C seems to make very large errors that are close to 100% for intervals that have between 100 and 250 calls. We found that these are cases at which the forecasted load exceeds the number of agents. In these cases Erlang C predicts a service level of 0, while in reality a high service level is realised. This show cases that the model is pessimistic. The large prediction errors during intervals with a substantial amount of call volume explain the large WAE, since the errors are weighted by the call volume. We do not see similar behaviour for Erlang A because of its consideration of abandonments. We have found that the performance of the Erlang C model improves drastically (to 24.7% in WAE) when disregarding the cases for which it predicts a service level of 0%. This may however explain the larger importance of the Erlang C feature. Our XGBoost model may have picked up that Erlang C is probably not reliable when its prediction equals 0%. Therefore it makes relatively more splits on Erlang C which means it has a larger importance.

Furthermore, it can be observed that the Erlang models have more of a symmetric error compared to the XGBoost model. All models have a hard time predicting the intervals with low call volume. We see that the models often underestimate the service level when there are a small number of calls. Little call volume introduces larger variability. Since we are considering short time intervals of 15 minutes, which introduces variability as well, we have many intervals with large call volume. This combined with the extra forecasting error that is introduced explains the large prediction error for all models. We see that although the XGBoost model very often underestimates the service level. It does a better job than the Erlang models in limiting the magnitude of its underestimation. The model found that limiting its underestimation is rewarding, which explains our earlier observation of the lowest predicted service level of 38.8% in table 6.5.

7

Conclusion

The goal of this paper was to propose a machine learning method to predict the service level in call centers, alternative to the traditional queueing models and simulation. The context in which the predictions are made is within the scheduling & shift assignment phase of the WFM process. This means that the predictions have to be short-term and are determined over short time intervals. We assumed that the predictions are made at least one week prior to the predicted moment and that the intervals are 15 minutes long. To assess whether the proposed machine learning method can be of value in a realistic environment, we train the model on a real world dataset and evaluate the performance on an unseen test set. We strictly consider input features that can be obtained at the moment of scheduling and compare the model performance to that of the traditional Erlang C and Erlang A models.

The machine learning technique that we proposed is XGBoost, an implementation of GBDT, which is a tree-based ensemble learning method that uses gradient boosting as ensemble technique. Three different XGBoost models, distinguished by their input features, were considered. This is to understand how well the model performs with several levels of input features. The basic model considers input features that are comparable to the input variables of the Erlang A model. The basic model performed better than Erlang C and worse than Erlang A. The intermediate model performs slightly better than Erlang A in terms of *WAE*, while the advanced model improves all prior models. However, the hyperparameters of these three models are not tuned nor have they endured the feature selection process. Based on these interim results we selected our final model for feature selection and hyperparameter tuning. We have found that our final model has a prediction error of 0.195 in terms of *WAE*, which is a reduction of 47.9% and 26.6% with respect to the Erlang C and Erlang A model, respectively.

The prediction errors are large in general for all models. This is partially due to the extra variability introduced by the short time intervals. We can conclude that the Erlang C and Erlang A model perform poorly in the environment considered within this research. This is because several assumptions of the models were violated. The dataset considered within this paper originates from a multi-skill call center. This means that the assumption that the calls are homogeneous and the agents are identical clearly does not hold. We have found that the arrivals do not follow a Poisson process. Furthermore, Erlang C and Erlang

A have been applied on small intervals of 15 minutes. This assumes that the arrival and service rate are constant and that a steady state is quickly reached within each interval [14], which is not the case for this dataset. In section 6.2 we have shown that the Erlang models poorly predict intervals with small call volumes and that Erlang C in particular makes very large prediction errors due to underestimation.

The Final XGBoost model often underestimates the service level but with limited magnitude. However, we have seen that the service level over short time intervals still remains hard to predict. We found that input features have large impact on the performance of the machine learning models. Furthermore, we concluded that the effect of the average handling time on the service level was captured by the model due to the other features. We believe that the load largely replaces the average handling time as input feature. In section 6.2 we have shown that the model considers many agent features more important for predicting the service level than features based on traditional input variables. This is an interesting insight since it implies that the model captures the impact of the presence of specific agents on the service level.

We can state that within a realistic call center environment a machine learning method can provide more accurate short-term prediction of the service level than Erlang C and Erlang A while having negligible computational time, which can be valuable for practitioners during the scheduling & shift assignment phase. Although the model provides more accurate prediction than the Erlang models, its prediction error is still quite large. Properly predicting the service level of short time intervals remains a challenge due to large variability. In conclusion, applying a machine learning method to predict the service level for call centers in a realistic environment can be valuable and is a promising field for future research. Based on this conclusion we recommend call centers and WFM practitioners to gather more quantitative and qualitative data such that enhanced future machine learning methods may be researched and implemented.

8

Discussion

In this research we have proposed a machine learning framework to predict the service level for call centers. In chapter 7 we have concluded that the proposed machine learning method is able to provide more accurate prediction of the service level than Erlang C and Erlang A and can be valuable for practitioners. However, there are several limitations to this paper to be mentioned. The research is conducted base on one year worth of data which means that yearly seasonality patterns could not be captured. Although we have focused on short-term prediction, we believe that a yearly seasonality effect could have impact on predicting the service level. In addition, even though the call center operates on Saturdays and holidays, we have left those out of consideration. Furthermore, we have applied both Erlang models as if we had a one-skill call center environment by considering the different queues as one. An alternative to this method used by practitioners is to apply the Erlang model per queue separately. The working capacity of one agent is then distributed over the queues that the agent is able to handle. This is done according to the call volume of the corresponding queues. This may potentially result in better performance of the Erlang models. Due to time and computation related limitations we were not able to explore alternative machine learning methods and implementations (e.g. extensive hyperparameter tuning, additional features).

There are several aspects that are interesting for future research within this field. As mentioned in chapter 7 we have found that predicting 15 minute intervals remains a challenge due to large variability, even with a machine learning model that considers many input features. A natural continuation would be to implement the proposed framework for longer intervals (e.g. 30, 60 minutes). We believe that the overall error of the machine learning models as well as the Erlang models would be significantly smaller due to less variability.

We believe the machine learning method to be less sensitive than the Erlang models to the forecast error introduced by the forecast implemented in this paper. An interesting topic for future work would be to research several forecast implementations and compare the introduced performance errors of the Erlang models with the machine learning model. Furthermore, we have explored an alternative method of training the machine learning method which showed promising interim results. We adjusted the training data such that we duplicate the interval rows as many times as the number of calls within that interval.

By doing this we add more weight to intervals with larger call volume. Since for the training data holds that each row represents one call, we are able to train on whether the call is picked up within 20 seconds or not. We did this by adding a new column *success*, in which we denote 1 if the corresponding call is picked up within 20 seconds and 0 otherwise. The model learns the relations between the input features and the new output feature. The model is able to output a predictive probability that this interval is 'picked up' within 20 seconds or not. This probability is essentially the service level prediction of the considered interval. The gain of this method alternative to the method implemented in this paper is that more value is assigned to intervals that have large call volume.

The machine learning method may be improved in several aspects. More input features could be considered, as for example the remaining shift duration per agent or individual performance metrics of agents. Alternative machine learning techniques may be explored with the use of real world datasets of different call centers. Lastly, more extensive experiments may be conducted, for example comparing the model performance to simulation models that are constrained by limited computational time.

Appendix

threshold 1	threshold 2	threshold 3	threshold 4	threshold 5
0	$6 \cdot 10^{-4}$	$1.2 \cdot 10^{-3}$	$2.6 \cdot 10^{-3}$	$3.4 \cdot 10^{-3}$
threshold 6	**threshold 7**	**threshold 8**	**threshold 9**	**threshold 10**
$4.2 \cdot 10^{-3}$	$8.3 \cdot 10^{-3}$	$1.25 \cdot 10^{-2}$	$1.67 \cdot 10^{-2}$	$3.33 \cdot 10^{-2}$

Table 8.1: Feature selection thresholds

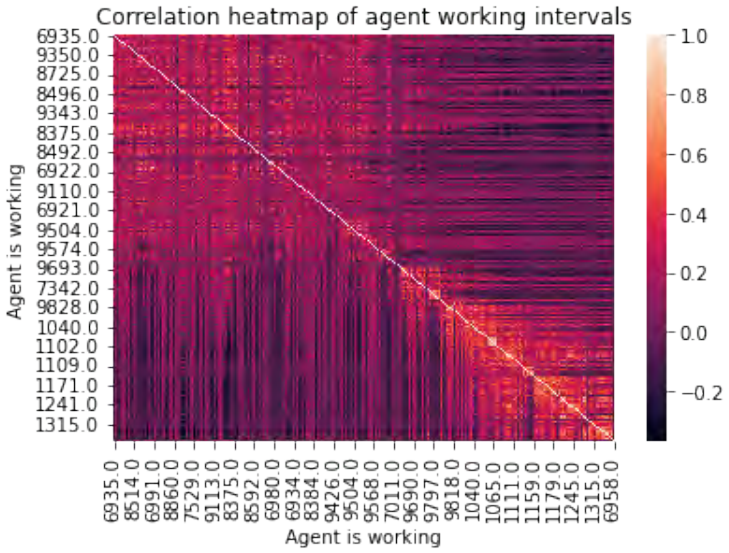

Figure 8.1: Correlation heatmap between agent features (whether agents are working or not)

Description	Symbol	Queues
Call volume	C	$1, 2, 3, 4, 6$
Load	L	$1, 2, 3, 4, 8$
Team skill	Q	$4, 5$
Average handling time	H	$1, 2, 3, 4, 5, 6, 7, 8$
Consult & transfer ratio	R	//
124 agent features	a_x	//

Table 8.2: Features dropped during feature selection

References

[1] A. Avramidis et al. "Optimizing daily agent scheduling in a multiskill call center". *European Journal of Operational Research* 200 (Feb. 2010), pp. 822–832. DOI: 10.1016/j.ejor.2009.01.042.

[2] A. Ingolfsson et al. "Combining integer programming and the randomization method to schedule employees". *European Journal of Operational Research* 202 (Apr. 2010), pp. 153–163. DOI: 10.1016/j.ejor.2009.04.026.

[3] Thomas Robbins et al. "Does the Erlang C model fit in real call centers?" *Proceedings - Winter Simulation Conference* 10 (2010), pp. 2853–2864. DOI: 10.1109/WSC.2010.5678980.

[4] M. Bodur and J. Luedtke. "Mixed-Integer Rounding Enhanced Benders Decomposition for Multiclass Service-System Staffing and Scheduling with Arrival Rate Uncertainty". *Management Science* 63 (June 2016). DOI: 10.1287/mnsc.2016.2455.

[5] B. Cao C. Hou and J. Fan. "A data-driven method to predict service level for call centers". *IET Communications* 16 (June 2022). DOI: 10.1049/cmu2.12192.

[6] M. Çezik and P. L'Ecuyer. "Staffing Multiskill Call Centers via Linear Programming and Simulation". *Management Science* 54 (Feb. 2008), pp. 310–323. DOI: 10.1287/mnsc.1070.0824.

[7] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". *Association for Computing Machinery* (2016), pp. 785–794. DOI: 10.1145/2939672.2939785.

[8] A.K. Erlang. "Solution of Some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges." *Post Office Electrical Engineer's Journal* 10 (1917), pp. 189–197.

[9] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." *Institute of Mathematical Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451.

[10] G. Koole and S. Li. "A practice-oriented overview of call center workforce planning" (Jan. 2021).

[11] A. Mandelbaum and S. Zeltyn. "Service Engineering in Action: The Palm/Erlang-A Queue, with Applications to Call Centers". *Advances in Services Innovations* 20 (Jan. 2007), pp. 17–45. DOI: 10.1007/978-3-540-29860-1_2.

[12] D. Mazareanu. "Call center market size region". *statista.com* (2017).

[13] V. Mehrotra and J. Fama. "Call center simulations: call center simulation modeling: methods, challenges, and opportunities". *Proceedings of the 35th Winter Simulation Conference: Driving Innovation* (Dec. 2003), pp. 135–143.

[14] G. Koole N. Gans and A. Mandelbaum. "Telephone call centers: A tutorial and literature review". *European Journal of Operational Research - EJOR* 5 (Oct. 2002).

[15] GeeksforGeeks Nikki. "https://www.geeksforgeeks.org/ml-gradient-boosting/" (Sept. 2020).

[16] M. Armony O. Aksin and V. Mehrotram. "The Modern Call-Center: A Multi-Disciplinary Perspective on Operations Management Research, Production and Operations Management". *Special Issue on Service Operations in Honor of John Buzacott* 16 (Jan. 2007), pp. 655–688.

[17] A. Mandelbaum O. Garnet and M. Reiman. "Designing a Call Center with Impatient Customers". *Manufacturing Service Operations Management* 4 (July 2002), pp. 208–227. DOI: 10.1287/msom.4.3.208.7753.

[18] C. Palm. "Research on telephone traffic carried by full availability groups". *Tekniska Meddelanden fran Kungl. Telegrafstyrelsen* 1 (1957), pp. 1–107.

[19] T. Robbins. "Evaluating the fit of the Erlang A model in high traffic call centers". *2016 Winter Simulation Conference* (Dec. 2016), pp. 1790–1801. DOI: 10.1109/WSC.2016.7822226.

[20] T. Robbins. "Evaluating the Performance of the Erlang Models for Call Centers". *International Journal of Applied Science and Technology* 9 (Mar. 2019). DOI: 10.30845/ijast.v9n1p1.

[21] Qingchen Wang Siqiao Li and Ger Koole. "Predicting Call Center Performance with Machine Learning: Proceedings of the 2018 INFORMS International Conference on Service Science" (Jan. 2019), pp. 193–199. DOI: 10.1007/978-3-030-04726-9_19.