Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Håvard Stene Tryman

Forecasting Cafeteria Visitors With Machine Learning

Master's thesis in Datateknologi (MTDT)
Supervisor: Adj. Assoc. Prof. Hai Thanh Nguyen
June 2019

NTNU
Norwegian University of
Science and Technology

Håvard Stene Tryman

Forecasting Cafeteria Visitors With Machine Learning

Master's thesis in Datateknologi (MTDT)
Supervisor: Adj. Assoc. Prof. Hai Thanh Nguyen
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Preface

Writing this thesis was the final task of the Computer Science (MTDT) program at NTNU. The research was conducted in cooperation with Telenor ASA. My supervisor was Adj. Assoc. Prof. Hai Thanh Nguyen, who originated the idea of forecasting cafeteria demand with various data from Telenor and weather data. The indended audience is anyone with an interest in machine learning and statistical methods for demand forecasting.

Håvard Stene Tryman
Trondheim, June 2019

Abstract

Accurate predictions of demand are necessary to stock and prepare items with low waste and high certainty that the demand is met. Forecasting future visitors can help restaurant owners take action to maximize revenue by planning the labor needed and the amount of raw goods to order.

This thesis will evaluate statistical and supervised learning methods for forecasting the number of daily transactions in three office cafeterias in Norway. The implemented methods are several types of naive forecasts, exponential smoothing (ETS), seasonal autoregressive integrated moving average (SARIMA), gradient boosting, decision tree, random forest, support-vector regression (SVR), multi-layer perceptron (MLP) and an ensemble of ETS, SARIMA, SVR, random forest and two gradient boosting models.

The dataset consisted of entries from October 2016 to February 2019. The supervised learning methods used features from the following categories: historical features (e.g., the number of visitors on the previous weekday), date features (e.g., the day of the week) and parking features (e.g., how many cars were parked in the company's parking garages at 07:00). Weather features (temperature, humidity and precipitation during lunchtime) were considered, but eliminated during feature selection.

Forecasts were made for each cafeteria individually and for an aggregate over all three cafeterias. The forecast label was the number of lunchtime transactions recorded in the cafeteria on a given day. Labels were forecast with three horizons: "zero weekday forecast" (0WDF), "two weekday forecast" (2WDF) and "15 weekday forecast" (15WDF). 0WDF, 2WDF and 15WDF methods make forecasts for the same day, two weekdays ahead and 15 weekdays ahead respectively. The forecast has to be ready before 10:00. How much earlier than 10:00 the forecast is ready depends on the method.

Evaluation was done in an expanding window process where the latest 50% of the dataset was used as evaluation data. The best method for all cafeteria-horizon combinations except four was the ensemble, and for these four combinations, ensemble's score was close to the best score. The best statistical method was ETS. Seasonal naive forecasts with an appropriate season length for the cafeteria and imputation based on the seasonality provided baseline forecasts for the cafeteria-horizon combination. For the aggregate forecast with the 0WDF horizon, the naive forecast gives a baseline mean squared error (MSE) and mean absolute percent-

age error (MAPE) of 8507 and 4.03%, while the ensemble gives an MSE and MAPE of 4332 and 2.86%. For the aggregate 15WDF forecast, the naive forecast gives an MSE and MAPE of 10482 and 4.58%, while the ensemble given an MSE and MAPE of 7508 and 3.91%. ETS and several machine learning methods score close to the ensemble, but with more variation between different cafeteria-horizon combinations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Description

There is currently a lot of uncertainty as to how many people will be eating lunch at each of the office cafeterias at Fornebu. When preparing food or stocking up on food, it is difficult to know exactly how much is required. Difficulties anticipating how many people will come leads to overstocking. The amount of people going to each cafeteria is partially explained by many factors, such as what is on the menu in each cafeteria, what day of the week it is, whether some employees are on holiday and how many people park in the parking garages in the morning, but it is difficult for a human to take all of these factors into account. Appropriate planning is especially important for food because it loses its value if it does not get used in time. The forecast for an individual cafeteria is most useful because when purchasing food for a cafeteria, it is useful to know how many people will be coming to the cafeteria when the food is prepared. It is possible to distribute some resources between cafeterias during the day based on how many people choose each cafeteria.

1.1.1 The Magnitude and Composition of the Food Waste

There is a lack of data about the avoidable food waste of cafeterias in Norway, although there are rough estimates based on Norwegian data and there are more accurate estimates for similar countries.

A 2015 study by Silvennoinen et al. measured food waste in five Finnish workplace and student cafeterias for five days. These cafeterias wasted about 25% of the total mass of originally edible food (OE) on average [2]. OE is divided into kitchen waste, serving waste and customer plate leftovers. These categories are defined in Table 1. The average composition of food waste in workplace and student cafeterias is shown in Table 2. Accurate forecasts can be used to reduce serving waste and the part of kitchen waste that is not due to incorrectly prepared food, while customer leftovers are more difficult to reduce with the use of accurate forecasts. This means that at most 20.8% of food waste in the average workplace or student cafeteria in the study could be avoided by perfectly adjusting production and stocking to the demand. A

	Kitchen waste, preparation and cooking	Serving waste, left from cooked and prepared meals	Customer plate leftovers
Food waste, Originally edible (OE)	Spoiled products, incorrectly prepared food, expired date products	Overproduction, food left from buffet	Food leftovers by customers on plate
Bio waste, Originally inedible (OIE)	Inedible parts of vegetables, coffee grounds and bones	Inedible parts of vegetables, bones	Vegetable peelings, bones

Table 1: Definition of originally edible and originally inedible food waste, and definition how waste was sorted to kitchen waste, serving waste and customer food waste.

Total food waste (OE)	Kitchen waste (OE)	Serving waste (OE)	Customer leftovers (OE)	Bio waste (OIE)
25.3%	3.6%	17.2%	4.5%	6.6%

Table 2: Types and percentages of originally edible food waste (OE) and originally inedible bio waste (OIE) from food produced by workplace and student cafeterias.

2016 report estimated the food waste of large-scale catering establishments in kindergartens, schools, retirement homes, hospitals and prisons in Sweden in 2012 and 2014 [3]. The estimates are shown in Table 3. The numbers take both inedible and edible food into account. The total waste is about 7 kg per capita per year.

	Food waste 2014 (tons)	Food waste 2012 (tons)
Schools and kindergartens	47,000	46,000
Retirement homes	17,000	13,000
Hospitals	4,900	3,700
Remand prisons and prisons	700	1,000
Sum	70,000	64,000

Table 3: Food waste in different large scale catering establishments in Sweden.

The research project KuttMatsvinn2020 has published a somewhat unreliable estimate of how much edible food was wasted by Norwegian workplace cafeterias in 2017. The estimate is 5100 tons, or 0.97 kg per capita, compared to 4500 tons for Norwegian hotels [4]. The number was extrapolated from a small set of cafeterias.

Approximately one third, i.e., 1.3 billion tons, of the food produced in the world for human consumption is wasted [5]. A 2016 study gave a 95% confidence interval of 10.5 ± 1.5 million tons for the food waste of the food service sector in the EU in 2012 (Figure 1) [6]. Both edible and inedible constituents of food are included in this number. This is 12% of all food waste in Europe in the same period and 0.8% of the roughly 1.3 billion tons of food wasted globally. The study's food service definition did not only include catering/cafeterias, but also hotels and restaurants.

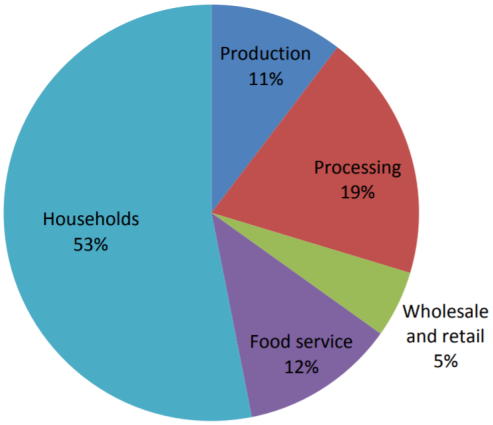


Figure 1: Split of EU-28 food waste in 2012 by sector; includes food and inedible parts associated with food.

1.1.2 Business Inefficiency

When it is difficult to anticipate the number of customers, businesses are forced pay for more labor and more raw ingredients than necessary [7]. Keeping a larger inventory due to difficulties planning also has a cost in storage space. In the Silvennoinen et al. study, the average workplace and student cafeteria wasted 28.24 kg per day. With a cost of 50 NOK per kg, this translates to 1412 NOK per day (the Director of Food and Beverage at Scandic Hotels Norway used an average cost of about 56 NOK per kg food wasted when providing an estimate of the cost of food waste in the hotels in 2017 [8]).

1.1.3 Environmental Impact

Food waste reduction has been an important sustainability goal because of the large magnitude of global food waste. The main environmental impacts mentioned by studies are climate change, energy use, soil and water acidification, adverse biological effects from excessive enrichment of water and continental surfaces and water usage [9]. In addition, there is land usage and material pollution associated with each part of the food supply chain. In 2016, the Norwegian government signed a binding agreement whose purpose is to reduce food waste in Norway by 2030 in line with the UN Sustainable Development Goal 12.3.

1.2 Motivation

This thesis was written to provide research results regarding machine learning for demand forecasting, examine whether cafeteria transactions is a domain where machine learning outperforms statistical models and to see what factors are useful as features for the implemented machine learning methods. This is of interest because machine learning for demand forecasting is in high demand and quickly growing. According to a Juniper Research study, service revenues associated with machine learning in demand forecasting will reach $3 billion by 2023,

up from $760 million in 2019 [10].

1.3 Research Questions

This thesis will answer the following research questions:

Research question 1 What features are useful when forecasting transactions in workplace cafeterias?
Research question 2 Which machine learning methods are best at forecasting the number of transactions in cafeterias?
Research question 3 Do machine learning methods forecast cafeteria transactions better than statistical models?

1.4 Contributions of the Thesis

The thesis has documented a large expanding window experiment where forecasts were made for the transaction count of cafeterias in a large organization with almost 2000 people coming for lunch per day. Performance of several statistical methods and machine learning methods was compared and it was shown that implemented methods can outperform the baseline of seasonal naive forecasts with appropriate imputation. The improvement when compared to naive was especially large for the aggregate forecast with the 0WDF horizon (forecast ready after lunch the previous day or at 08:00 on the same day).

Chapter 2

Theory and Background

This chapter will review related work to give an overview of state-of-the-art solutions for similar forecasting problems. The more specialized subset of theory that is relevant for the thesis will then be explained and terms with several definitions will be defined for the rest of the thesis.

2.1 Related Work

In this section, papers that apply forecasting models in domains similar to cafeteria transactions will be reviewed in order to find out what the state-of-the-art solutions are, whether weather shows potential as a feature, what other features are judged as useful, how models are evaluated and what metrics are used for evaluation. Table 4 lists all papers that will be discussed in this section together with their forecasting domain, methods and which methods were judged as best. Table 5 lists the same papers' features, metrics and evaluation procedures.

Four papers making forecasts for university cafeterias were found: one work forecasting dinners per day, one work forecasting transactions per five minutes and two works forecasting sales. One work forecasting future visitors for a restaurant (Ma et al.) and one work forecasting foot traffic in various places including restaurants were found. Ma et al. was published in 2018 and mentioned that the literature contains few applications of machine learning for restaurant visitor forecasts: "Despite various visitor forecasting techniques for other purposes such as national tourism and hotel demand (e.g., [...]) in the literature, little is known for restaurant owners to estimate the number of future visitors using big data." Other relevant forecasting papers were forecasting sales of specific items in supermarkets, weekly sales of different types of beer, daily supermarket sales and sales of specific fashion items in future seasons.

Common top-performing methods are seasonal autoregressive integrated moving average (SARIMA), random forest, support-vector regression (SVR), seasonal naive and artificial neural networks (ANN). In Ryu et al. which has the most similar domain to this thesis, the best method was multiple linear regression, but a naive forecast that always predicted the same value as one week before was recommended due to being simpler and almost as good as multiple linear

regression. In Weiner et al., the method using a template curve and regression tree was best, but ANN, gradient boosting and a variation of a moving average were among the best methods. The ANN in Weiner et al. was a multilayer perceptron (MLP) with two hidden layers: one with 1000 and one with 300 ReLU units respectively. It was trained with the RMSprop algorithm, learning rate of 0.0005 and batch size of 128. This is a rather different MLP from the one that will be implemented in this thesis. The ANN in Cho et al. was an Elman network. Ensembles were among the top-performing methods for Ma et al., Liu et al., Aburto et al., Gurnani et al. and Weiner et al. Weiner et al. mentions: "While some systems are similar, they do appear to be learning different things: A late fusion of all the winning systems outputs (by taking their mean) results in a new system that outperforms each single system by a large margin."

When it comes to relevant features, it seems like the date features and lagged values for the target are the most useful. Ma et al.: "We found that time-related features (such as week of year) and historical visitor records (such as mean visitors on a day) are the strongest indicators of the future visitor number to a restaurant." The fact that time series forecasting methods have been applied with success is an indication that lagged values are important.

Most of the papers explaining their evaluation method separate the dataset such that the training data contains the earliest observations and the testing data contains the latest observations. 10-fold cross-validation and random selections of training and testing data have been used, but will not be used in this thesis because they do not account for the ordering of the observations.

Mean squared error (MSE) and mean absolute percentage error (MAPE) are common among relevant papers and will be used in this thesis because the former is a useful loss function and is common in machine learning libraries and the latter is simple to interpret.

Regarding weather features, the Abrishami et al. paper which was predicting foot traffic says: "Weather: Before adding this feature to our prediction model, we performed some analysis to explore the potential relationships between the weather features and foot traffic, and no significant correlation observed. Moreover, because weather forecasts are inaccurate and they can cause error propagation, we do not use these features in our prediction model." Xinliang et al. found weather to be a somewhat useful feature and hypothesized that cold, windy or rainy weather would motivate students to buy food in the supermarket or buy take-away food.

One instance of menus being used as a feature was found. The fifth best system in Weiner et al. used the cafeteria menu as a feature by performing automatic textual analysis to group menus, and used this together with time and weather features as input for a gradient boosting model.

Article	Forecast value	Forecasting methods	Best method
Ryu and Sanchez (2003) [11]	Dinners per day in a university dining facility	Naive, seasonal naive with a season length of one week, seasonal naive with a season length of one semester, moving average, double moving average, simple exponential smoothing, double exponential smoothing, Holt's method, Winter's method, linear regression and multiple regression.	Multiple regression was the most accurate forecasting method, but seasonal naive with a season length of one week was selected as the most appropriate forecasting method because of its simplicity and high level of accuracy.
Liu and Sun (2016) [12]	University cafeteria sales revenue	ANN	ANN
Weiner et al. (2017) [13]	Receipts issued per 5-minute slot in a university cafeteria	Template curve + regression tree[a], mean of previous values[b], ANN, gradient boosting	Template curve + regression tree
Xinliang and Dandan (2017) [14]	University cafeteria sales	ANN	ANN
Ma, Tian, Luo and Zhang (2018) [15]	Future visitors of a restaurant	K-NN, random forest, XGBoost, mix of all three	Mix of all three
Arunraj, Ahrens, Fernandes and Müller (2014) [16]	Daily sales of perishable items in a discount store	SARIMA SARIMAX	SARIMAX
Liu, Wei, Wang, Liao and Gao (2011) [17]	Weekly sales of specific beer types in a supermarket	SVR, SVR ensemble (SVRE), SVRE based on sampling equally (SVRESE), Heterogeneous SVRESE (HSVRESE)	HSVRESE
Aburto and Weber (2007) [18]	Demand of four best-selling items in a supermarket	Naive, seasonal naive with season a season length of one week, unconditional average, SARIMAX, MLP, additive SARIMAX-MLP hybrid, sequential SARIMAX-MLP hybrid	Sequential SARIMAX-MLP hybrid
Gurnani, Korkey, Shahz, Udmalex, Sambhe and Bhirud (2017) [19]	Sales by a drug store company	ARIMA, autoregressive Neural Network (ARNN), XGBoost, SVM, hybrid ARIMA-ARNN,hybrid ARIMA-XGBoost, hybrid ARIMA-SVM and STL decomposition using ARIMA, seasonal naive, XGBoost [c]	STL decomposition
Loureiro, Miguéis and da Silva (2018) [20]	Sales of specific fashion products in upcoming seasons	Deep learning, Decision Trees, Random Forest, Support Vector Regression, Artificial Neural Networks and Linear Regression	Deep learning, random forest
Abrishami, Kumar and Nienaber (2017) [21]	Hourly foot traffic in various stores, such as gyms, restaurants and bars.	Random forest, Google Cloud predictor[d], SVR	SVR, but the other methods were close
Yang, Pan and Song (2014) [22]	Hotel nights sold in a destination	ARMAX, threshold autoregressive (TAR) model	ARMAX
Cho (2002) [23]	Arrivals to Hong Kong from other countries	ETS, ARIMA, ANN	ANN

[a] Created template sales curves for different times of the year and used a regression tree to predict the scale factor for the curve.

[b] Used the mean of a certain set of previous 5-minute slots after smoothing with a median filter.

[c] The pure ARIMA model used external regressors and was by definition ARIMAX, and the other ARIMA models were most likely ARIMAX

[d] https://cloud.google.com/prediction/

Table 4: Similar forecasting papers' forecasting domain, methods and best methods.

Article	Features	Metrics[a]	Evaluation procedure
Ryu and Sanchez (2003)		MAD, MSE, MPE, MAPE, RMSE	Trained on 2000 fall semester, test data was 2001 entire spring semester
Liu and Sun (2016)	Current amount of online takeout as indicated by certain Baidu searches, day of the week, holiday, teaching week	MAPE	Before creating the model, a random 70% was selected as training data and a random 30% was selected as test data
Weiner et al. (2017)	Year, month, day, weekday, semester time, weather, menu	RMSE	Training data contained five-minute-slots for 2009 through 2015 and the year 2016 was test data
Xinliang and Dandan (2017)	Weather (temperature, precipitation and maximum wind speed), current amount of online takeout as indicated by certain Baidu searches, week, education week, holiday	MAPE, network accuracy	Random 70% as training set, remaining 30% was test set
Ma et al. (2018)	Year, month, week of year, day of week, holiday, latitude, longitude, genre, and area the restaurant is located in, visitor features (mean, median, minimum, maximum of visitors, and the total number of visitors before a day), reservation features (mean reserved seats, the total number of reserved seats before a day, mean time of the reservation before expected visit, and whether the reservation is within 48 hours)	RMSLE	Split into training and testing set
Arunraj et al. (2014)	Holiday-related factors, yearly demand pattern, and price reduction	MAPE, RMSE	
Liu et al. (2011)	Price of current week, the current point in the season, execute sales promotion, the effect on sales quantity from promotion of other categories, additional demand quantity of current category due to promotion, average sales quantity of previous four weeks, last week's sales quantity.	MAPE	Data was normalized by maximum rule, then divided into train and test sets with a 2:1 ratio
Aburto and Weber (2007)	Sales data previous 14 days, days at the end of the month when people receive their monthly salary, days in the middle of the month where people receive 2-weekly salary, before holiday, holiday, Chilean independence days, days before Easter, summer holidays, summer, new year (the only day supermarkets are closed), day of the week, item's price, "item's price"/Max("Price in micro market"), "item's price"/Min("Price in micro market")	MAPE and NMSE	12 months training data followed by 1 month test data
Gurnani et al. (2017)	School holiday, promotion, open, dayofweek, day, month	MAE, RMSE	Split into training and testing set
Loureiro et al. (2018)	Physical characteristics of the products, price and features representing the domain knowledge	R^2, RMSE, MAPE, MAE, MSE	Bootstrapping [24]
Abrishami et al. (2017)	Regular holidays, festival holidays, happy hour, sport games, local concerts, conferences, other events, close to schools, tourist cities	RMSE, MAE, MAPE	Partition into training and test datasets
Yang et al. (2014)	Web traffic volume data from local destination marketing organizations	MAPE, RMSPE	Several splits into training and testing data
Cho (2002)		RMSE, MAPE	25 years training data, followed by 1 year of testing data

[a]MAD = mean absolute deviation, MSE = mean squared error, MPE = mean percentage error, MAPE = mean absolute percentage error, RMSE = root mean squared error, RMSLE = root mean squared logarithmic error, RAE = relative absolute error, NMSE = normalized mean squared error, MAE = mean absolute error, RMSPE = root mean squared percentage error.

Table 5: Similar forecasting papers' features metrics and evaluation procedures.

2.2 Time Series Forecasting

A time series is a collection of data points ordered by time with a significant temporal dependence. The temporal dependence between points constrains the approaches one should take to make forecasts, but the dependence can also be utilized by forecasting methods. For example, a series of stock prices per day or a series of temperature measurements over time are time series. When making forecasts for a time series, what is interesting is how a model trained on past data would make forecasts for the future. For any of these two series, using every other observation to adjust a forecasting method to make forecasts for the rest of the observations would enable a highly accurate forecast, but would be a poor representation of the models' ability to forecast into the future.

2.2.1 Level, Trend, Seasonality

It can be useful to model a time series as consisting of random noise on top of a pattern, where the pattern consists of level, trend and seasonality. The level can be seen as a baseline value at a certain point. Trend is the increase or decrease in level over time. Seasonality is a regular pattern of increases and decreases in value that repeats at regular intervals. The unexplained variation on top of these components is the noise.

In general, long term forecasts are usually less accurate than short term forecasts. Cafeteria transaction forecasts with longer horizons are expected to be less accurate than shorter horizons.

Aggregate forecasts are usually more accurate than non-aggregate forecasts. With office cafeterias especially, there will be a relatively stable number of employees going to work every day, where most of them will eat lunch in at least one cafeteria. The forecasts for the aggregate number of transactions are expected to be more accurate than forecasts for individual cafeterias.

2.2.2 Differencing

If a time series is stationary, its properties are the same over the entire series. Trend and series makes a time series non-stationary. In order to make a time series stationary, differencing can be applied.

Ordinary differencing can eliminate trend. Ordinary differencing of degree one is to subtract the observation one time step before: $y_i = y_i - y_{i-1}$. Ordinary differencing of degree n is to apply degree one ordinary differencing n times. For example, ordinary differencing of degree two: $y_i = (y_i - y_{i-1}) - (y_{i-1} - y_{i-2})$.

Seasonal differencing can eliminate seasonality. Seasonal differencing is to apply the same transformation but subtracting values n seasons back. For example, if season length is s, seasonal differencing of degree one is: $y_i = y_i - y_{i-s}$.

2.2.3 Naive

One of the simplest forecasting methods for a time series is the naive forecast. The naive forecast is to always predict that the value at time i is the observed value at time $i-1$. For seasonal

time series with repeating seasons of length n, better accuracy can be achieved with a seasonal naive forecast with season length n. The seasonal naive forecast with season length n predicts that the value at time i will be the observed value at time $i - n$.

2.2.4 Exponential Smoothing

Exponential smoothing methods make forecasts that are based on a weighted sum like in SMA, but with exponentially decreasing weights further back in time. Exponential smoothing methods are referred to as ETS, which stands for error, trend, seasonality. Exact calculations made by exponential smoothing models are shown in [25].

Simple exponential smoothing (SES) is similar to simple moving average (SMA). The SMA forecast predicts that the observation at time i will be equal to the average value of the n preceding observations before time i. SMA is therefore like summing the n preceding observations after multiplying each of them with a weight of $\frac{1}{n}$. In SES, forecasts are made like in SMA, but weights decrease exponentially each step further back in time. The result of SES can be viewed as an estimate of the time series' level.

Double exponential smoothing is an extension of SES in the way that it is the weighted sum of the SES forecast and a trend component. The trend component is calculated by summing changes between past points, with exponentially decreasing weights further back in time. The calculation depends on whether the trend is modeled as additive or multiplicative. Additive trend means that the time series level is assumed to change with a constant value over time, e.g., 100 more shoes are sold per season. Multiplicative trend means that the time series is assumed to change by a percentage of itself over time, e.g., 10% more shoes are sold per season.

Triple exponential smoothing is a further extension that forecasts a weighted sum of a level component, trend component and a seasonal component. The seasonal component is the weighted sum of corresponding values from previous seasons, with weights decreasing exponentially.

2.2.5 SARIMA

The exact calculations made by SARIMA models are shown in [25]. This section will only give a high-level summary to guide parameter searches. SARIMA models take seven parameters and are denoted ARIMA$(p,d,q)(P,D,Q)m$. (p,d,q) tunes the non-seasonal part of the model and $(P,D,Q)m$ tune the seasonal part of the model. p and q describe how many lagged values are included in certain parts of the calculation. P and Q describe how many values lagged with n season lengths back are included in certain parts of the calculation. d and D determine the ordinary and seasonal differencing respectively. m is the season length. The number of parameters means that there is a large space of parameter combinations to try and there are no general rules by which to select all parameters. For this reason an automatic parameter search can be appropriate for tuning parameters.

2.3 Machine Learning

2.3.1 Definition

Machine learning is a subcategory of AI. The term AI encompasses many interrelated categories. Among the most important categories of AI that depend heavily on machine learning, we have natural language processing, speech-to-text, text-to-speech, machine vision and image recognition. It is common to classify machine learning methods by one of four learning styles: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. Supervised learning is used in this thesis.

Machine learning is defined as computer systems' automatic learning of tasks by constructing a mathematical model based on training data, instead of explicitly being programmed to solve the task.

2.3.2 Representation, Evaluation and Optimization

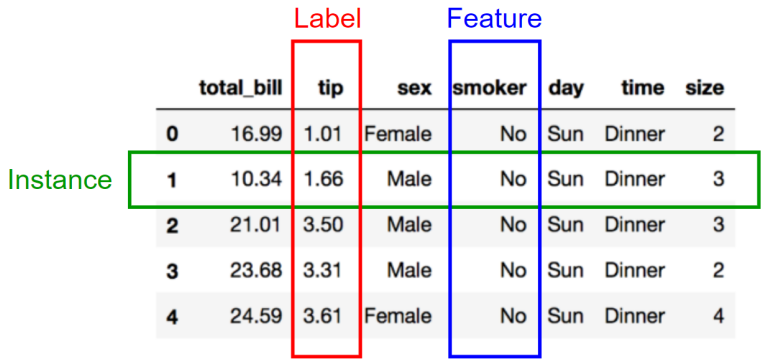
One commonality among thousands of available machine learning algorithms is that they consist of three components: representation, evaluation and optimization [26].

Representation: The machine learning algorithm builds a model that is represented in some kind of formal language, e.g., a set of propositional rules, a decision tree or a neural network.
Evaluation: The model's performance has to be judged with some kind of evaluation function. For instance, if we have a model whose output is a real number, the evaluation function can be the mean squared difference between the model's output and the correct value.
Optimization: The machine learning algorithm has a method of trying out different representations to search for the one that performs best according to the evaluation function.

2.3.3 Supervised learning

Supervised learning is the subset of machine learning where we use labeled data to optimize the model. In other words, the data we use to optimize the model consists of inputs and their desired outputs. Let us first define some terms:

Feature: A feature is a property (i.e., a characteristic or an attribute of something) that can be represented by a numerical or a categorical value. Examples of categorical properties: the day of the week, the color of something and whether or not it is a Monday. Examples of numerical properties: the number of sales on a certain day, the temperature and the humidity. Features are the properties we want the model to take into account when generating outputs.
Input: An input X_i is a collection of values, with each value corresponding to one feature. For example, an input can contain the value 170.2 corresponding to the feature "blood pressure".



	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Figure 2: Sample dataset for the supervised learning problem of predicting the tip in a restaurant transaction.

Target: A target is a property that can be represented by a numerical or a categorical value. The target is the property that the model's output is trying to predict.
Label: A label y_i is the correct value for the target, given the input X_i. It can refer to an entire column of values or to a specific value in the column.
Instance: One row of data in a dataset. It is a collection of values representing one observation, each value belonging to a specific feature or the target.
Dataset: A collection of instances.

The words feature and input, as well as the words label and target, are often used interchangeably, but will follow the aforementioned definitions in this thesis.

A supervised learning problem might for instance be to predict the tip in a restaurant transaction based on other information. A sample dataset for this problem with different parts of the dataset outlined is shown in Figure 2. A machine learning model would be trained on many such instances to hopefully be able to receive a new input with values for the various features and return a good prediction for the value of the tip.

Supervised learning problems with categorical outputs are called classification problems, while supervised learning problems with numerical outputs are called regression problems. Predicting the number of visitors to a cafeteria is best viewed as a regression problem even though there is a discrete set of true outputs. A major problem with using classification algorithms in this context is that their evaluation functions would treat the output 10 as equally bad as 100 if the desired output were 101, and would therefore be too difficult to train.

Some supervised learning methods are parametric. This means that one or more constants have to be specified before the model starts being fitted to the training data. The best parameters vary significantly depending on the dataset. The best parameters can be searched for by performing a grid search where one specifies a set of possible values for each parameter and tries every possible combination of values for each parameter. If we have two parameters whose sets of possible values are $\{1,2,3,4,5\}$ and $\{1,2,3\}$, there are 15 combinations to try. For each combination, the model has to be trained and validated.

2.3.4 Data Preprocessing

Data preprocessing is to make changes to a dataset so that it can be better used by the model. Some relevant preprocessing steps:

- Removing or correcting invalid entries in the dataset, such as an entry where the arrival time is before the exit time.
- Removing entries that contain outlier values.
- Imputing missing values. One way to do this is to fill in the mean value of the non-missing values for the corresponding feature.
- Standardize each feature. For example, subtract the feature mean and then divide with the feature's standard deviation. This makes the values of all features have similar magnitudes. In some machine learning methods such as SVR, having features of different magnitudes gives worse results.

2.3.5 Feature Selection

When a feature is not relevant enough, including it has a high chance of negatively impacting the performance. Redundant features also bear a cost in storage space of the dataset and running time of the algorithm. Feature selection is a process where one tries to find the best set of features. Some methods to help perform a feature search are univariate selection, selection based on feature importance and backward elimination. Weiner et al. had the following comment about all five winning submissions: "One similarity between all the winning entries is that they relied heavily on selecting which data to train their systems on: All of them chose to exclude data, sometimes using only the most recent year and often choosing to train their system only for times during which the cafeteria is known to be open and setting all other times to zero (or modeling them separately)."

Univariate selection evaluates each feature's relationship with the target individually to decide whether or not to keep it. One way to judge the relationship is to look at the Pearson correlation between the feature and the target. The Pearson correlation is a number in the range $[-1, 1]$ and is a measure of the linear relationship between the feature and the target. -1 means a perfect negative linear relationship, 0 means no linear relationship and 1 means a perfect positive linear relationship. The Pearson coefficient is not sufficient for detecting non-linear relationships.

Other forms of feature selection are based on trying to apply the machine learning method with different feature combinations and using the score to select features.

2.4 Performance Measures

It is important to explain the nature of forecasts' errors with error metrics such that the forecasts can be judged and compared.

In this paper, forecasts' performances will be compared by their MSE and MAPE will be given for a simple interpretation.

MSE between a forecast and the actual values is the average of the squared difference be-

tween forecast and the actual value. If y_i is the actual value at time i, the forecast for y_i is $\hat{y}_i$ and the number of observations is n, then MSE is defined as:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

MAPE is a common and intuitive that metric. where there are n data points, A_t is the actual value and F_t is the forecast value. With variables defined as the previous formula, MAPE is defined as:

$$\frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}}{y_i}\right| \tag{2.2}$$

One drawback of MAPE is that it adds more weight to errors where the actual values are small. If the actual value is 2000 and the forecast if 1800, the absolute percent error is 10%. If the actual value is 1600 and the forecast is 1800, the absolute percent error is 12.5%.

2.5 Generalization

The point of training a forecasting model is to make it generalize to new inputs. The fact that a model performs well on the training data does not guarantee that it will perform well on new data. Performing well on training data, but not on unseen data is called overfitting.

While optimizing a model to improve the performance on the training data according to the evaluation function, there is often a point where further optimization consistently worsens performance on unseen data. The model becomes worse at generalizing because it adapts to the noise and specifics of the training data more than it adapts to the general rules of whatever process the data was taken from.

Early stopping methods are methods to stop training before overfitting happens. It is common to train the model on one subset of the dataset, i.e., the training set, while monitoring its performance on a different subset of the dataset, i.e., the validation set. The performance on the validation set is an estimate of the model's ability to generalize. The idealized situation is that the validation error only has one minimum, as shown in Figure 3. In this case, one would stop training as soon as the validation error increased and keep the model in the state it was in before the last training iteration. In reality, the validation error tends to have local minima, as shown in Figure 4. It can therefore pay off to have a more elaborate trigger for when to do early stopping, e.g., stopping when the validation error has not improved for 150 iterations. In the situation displayed in Figure 3, using this trigger would result in the model achieving the lowest validation error close to training epoch 200. If the trigger were to stop training when the validation error had not improved for 10 iterations, it would result in a model achieving the local validation error minimum around epoch 50. The first trigger would lead to a validation error about 1% smaller than the second trigger, at the cost of a running time several times larger. Validation error graphs can have many different shapes and the relative importance of validation error and training time depends on the context. It is therefore a good idea to look at the validation error graphs and find an appropriate early stopping trigger for the specific situation.

Figure 3: Idealized training and validation error curves. Vertical: errors; horizontal: time. Taken from [1].

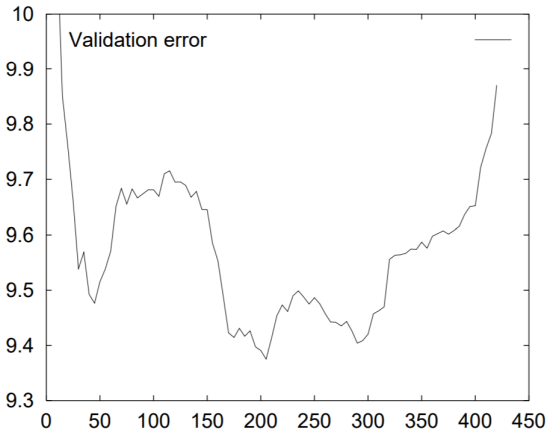


Figure 4: A real validation error curve. Vertical: validation set error; horizontal: time (in training epochs). Taken from [1].

2.6 Evaluation

A very common form of evaluation for machine learning methods is K-fold cross-validation. In this evaluation procedure, the dataset is split into K parts and training and testing is done in K iterations. In each iteration, one part is testing data and $K-1$ parts are training data. Each part is training data exactly once. This evaluation procedure makes good use of the data because every instance is used as testing data once. K-fold cross-validation is not suitable for time series because it does not account for the data's temporal ordering.

For time series, an expanding window evaluation is more appropriate. In an expanding window process, the earliest instances are used as training data and the latest instances are used as testing data. Testing data instances are traversed from early to late in several iterations and forecasts are made using all testing data leading up to the testing data of the iteration. An expanding window process is illustrated in Figure 5.

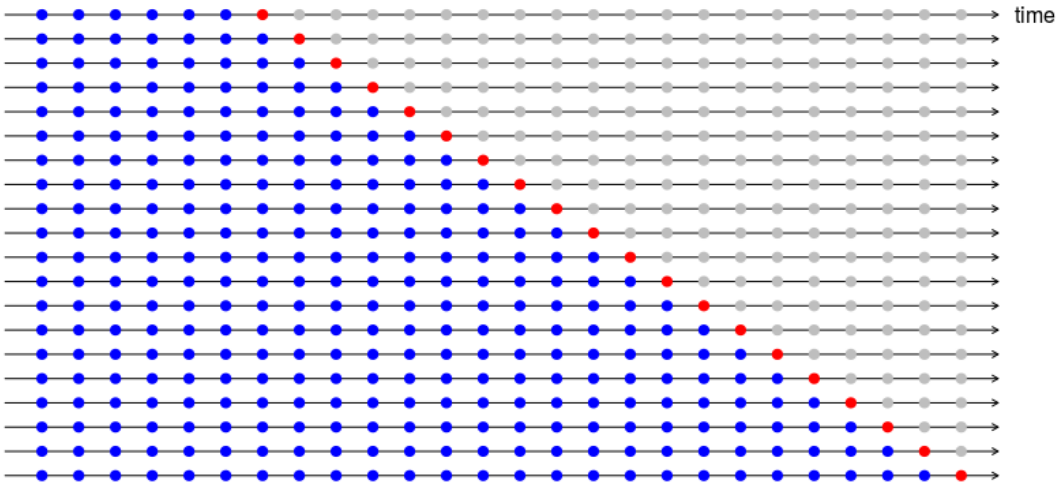


Figure 5: Expanding window process. Each line is one iteration, with later iterations further down. The blue points are the iteration's training set and the red points are testing sets. Taken from https://robjhyndman.com/hyndsight/tscv/.

Chapter 3

Proposed Solution

This chapter will begin with a data section that introduces and analyzes the dataset, defines the label, defines the features and shows how the features were extracted, defines three forecasting horizon categories and their available features and summarizes the features selection process. The next section will introduce and justify the implemented forecasting methods. The final section summarizes the expanding window evaluation process.

3.1 Data

3.1.1 Data Sources

Features and labels were extracted from three datasets:

- $CafeteriaTransactions$: cafeteria transaction data from Telenor.
- $ParkingTransactions$: parking transaction data from Telenor.
- $HourlyWeather$: hourly weather measurement data from Yr, delivered by the Norwegian Meteorological Institute and NRK.

$CafeteriaTransactions$ was created by concatenating a batch of transaction data from swiped cards in the cafeterias. The columns and some sample rows are shown in Figure 6. Values in the "point of sale" column show which machine the transaction was made at. "menu choice" and "price" are not used.

datetime	point of sale	menu choice	price
2016-10-03 11:06:17.000	PC1045FBUUnion HouseSelv	Lunch (fixed price /unit)	42
2016-10-03 11:08:55.000	PC1045FBUUnion HouseSelv	Lunch (fixed price /unit)	42
2016-10-03 11:09:23.000	PC1045FBUUnion HouseSelv	Lunch (fixed price /unit)	42
2016-10-03 11:10:22.000	PC1045FBUUnion HouseSelv	Lunch (fixed price /unit)	42
2016-10-03 11:10:27.000	PC1045FBUUnion HouseSelv	Lunch (fixed price /unit)	42

Figure 6: $CafeteriaTransactions$ example.

ParkingTransactions was created by concatenating a batch of parking data from cards being swiped in three parking garages was received. The columns and some sample rows are shown in Figure 7. "payment satus" is not used.

parking garage	start datetime	end datetime	payment status
Telenor P1 P-hus	2016-12-31 23:39:11	2017-01-01 00:08:46	GRATIS
Telenor P1 P-hus	2017-01-01 00:31:29	2017-01-01 00:32:26	GRATIS
Telenor P1 P-hus	2017-01-01 06:55:10	2017-01-01 15:43:33	GRATIS
Telenor P1 P-hus	2017-01-01 07:07:35	2017-01-01 15:27:36	GRATIS
Telenor P1 P-hus	2017-01-01 07:34:21	2017-01-01 19:53:22	CONTRACT

Figure 7: *ParkingTransactions* example.

HourlyWeather was created from weather data obtained via the date search function on yr.no on 21 March 2019. For all days in the date span of *CafeteriaTransactions* a URL on the same format as [27] was accessed, measurements were extracted with regex searches and added to *HourlyWeather*. The measurements were made at Bygdøy observation station, which is about three kilometers from the Telenor cafeterias. The columns and some sample entries from *HourlyWeather* are shown in Figure 8.

datetime	temperature	precipitation	humidity
2016-10-01 00:00:00	10.4	0.0	60
2016-10-01 01:00:00	9.8	0.0	60
2016-10-01 02:00:00	8.8	0.0	63
2016-10-01 03:00:00	8.1	0.0	68
2016-10-01 04:00:00	6.0	0.0	80

Figure 8: *HourlyWeather* example.

3.1.2 Data Analysis and Feature Extraction

CafeteriaTransactions

CafeteriaTransactions entries span from 3 October 2016 to 26 February 2019, which is a span of 877 days. Figure 9 shows the amount of cafeteria transactions that fall on each day. There are no transactions recorded during weekends. There are 1036656 transactions recorded between 10:00 and 14:00, while there are only 1410 transactions recorded earlier than 10:00 and only 2555 transactions recorded later than 14:00. All unique values in the "point of sale" column in *CafeteriaTransactions* are:

```
"PC1045FBUUnion HouseSelv", "PC1033FBUeattheStreetSelvbet",
"Pc1032FBUeattheStreet", "PC1020FBUFresh4YouSelvbV",
```

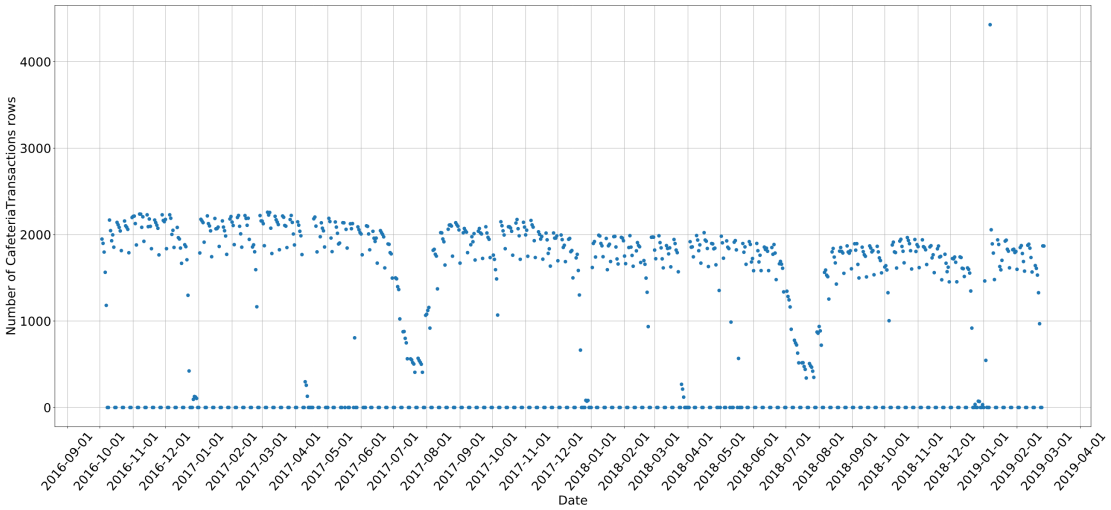


Figure 9: The number of *CafeteriaTransactions* rows that fall on each day.

```
"PC1023FBUFresh4YouSelvbet", "PC1044FBUSoup&Sandwich",
"PC1043FBUSoup&SandwichSelv", "PC1041FBUSoup&SandwichSelv",
"PC1031FBUeattheStreet", "PC1021FBUFresh4You",
"PC1042FBUSoup&SandwichSelv", "PC1034FBULemon Graden",
"PC1046FBUUnion House", "PC1024FBUCafeAroma",
"PC1013FBUOasenSelvHÃ¸yre", "PC1022FBUFresh4You", "PC1015HotSpot",
"PC1047FBUSoup&SandwichSelv", "PC1036Fbu eattheStreetSelvbet"
```

Only the three cafeterias Eat the Street, Soup & Sandwich and Fresh 4 U are relevant for the thesis. Therefore, the only relevant entries are entries with one of the following values in the "points of sale" column:

```
"PC1033FBUeattheStreetSelvbet", "Pc1032FBUeattheStreet",
"PC1031FBUeattheStreet", "PC1036Fbu eattheStreetSelvbet",
"PC1020FBUFresh4YouSelvbV", "PC1023FBUFresh4YouSelvbet",
"PC1021FBUFresh4You", "PC1022FBUFresh4You",
"PC1044FBUSoup&Sandwich", "PC1043FBUSoup&SandwichSelv",
"PC1041FBUSoup&SandwichSelv", "PC1042FBUSoup&SandwichSelv",
"PC1047FBUSoup&SandwichSelv"
```

All *CafeteriaTransactions* entries have the value "Lunch (fixed price /unit)" in the "menu choice" column. For this reason, the column was ignored.

There are many different values for "price", with some erroneous values, such as "3072195". Most values are around 40, with a significant minority of values being equal to 0, indicating that the employee pays for a subscription instead of paying per meal. The "price" column was also ignored.

ParkingTransactions

Each row in *ParkingTransactions* corresponds to one process of recording an entrance and an exit for the same person. The earliest "start datetime" value is 30 September 2016 and the latest "end datetime" value is 20 February 2019. Figure 10 shows the amount of "start datetime" values that fall on each day. The figure is similar to Figure 9, except for a period of outliers around February 2017. Let "delta" in the context of a *ParkingTransactions* row be the time difference between "start datetime" and "end datetime" and let delta be negative iff. "start datetime" is later than "end datetime". Figure 11 shows histograms for deltas in different ranges. Rounded to the nearest thousand, 832000 rows have a delta smaller than 16 hours, 3000 rows have a delta smaller than one minute, and 26000 rows have a delta greater than 16 hours. Two rows have a negative delta, with the smallest being -17 seconds. The largest delta is 184 hours, rounded to the nearest hour. The first and second histograms show that most deltas have values that should be expected if *ParkingTransactions* were recorded by a valid system: deltas tend to be smaller than 12 hours and most deltas are around 8 hours long, corresponding to an employee's entire workday. The third histogram shows that deltas greater than 16 hours are concentrated around multiples of 24 hours, with a large peak around 24 hours. The 24 hour peak is possibly due to employees' exits not being registered, which leads to the transaction not being finalized until the employee enters the next day, around 24 hours later. Later peaks around multiples of 24 might be due to similar situations, but with more days between the two entrances.

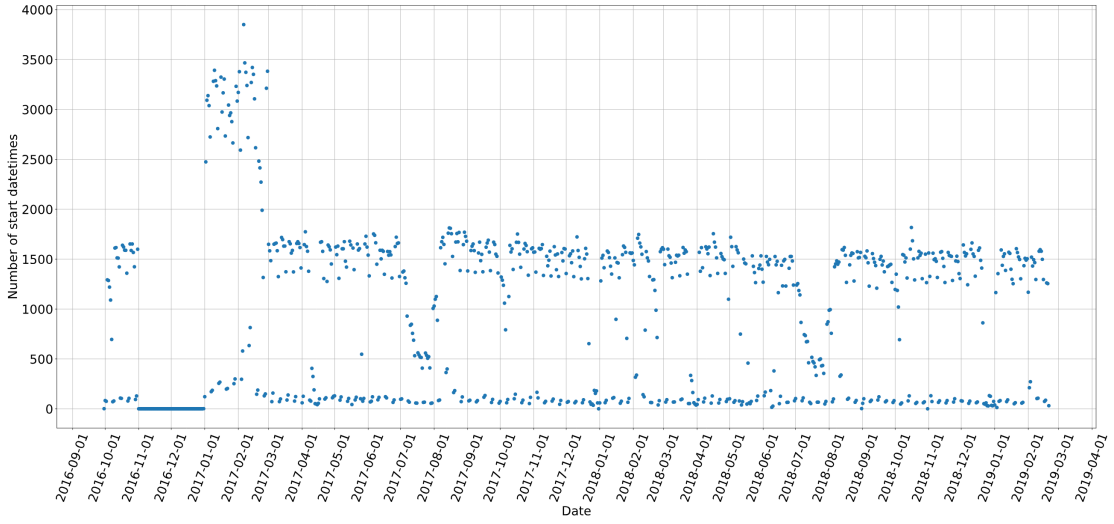


Figure 10: Number of "start datetime" values in *ParkingTransactions* that fall on each day.

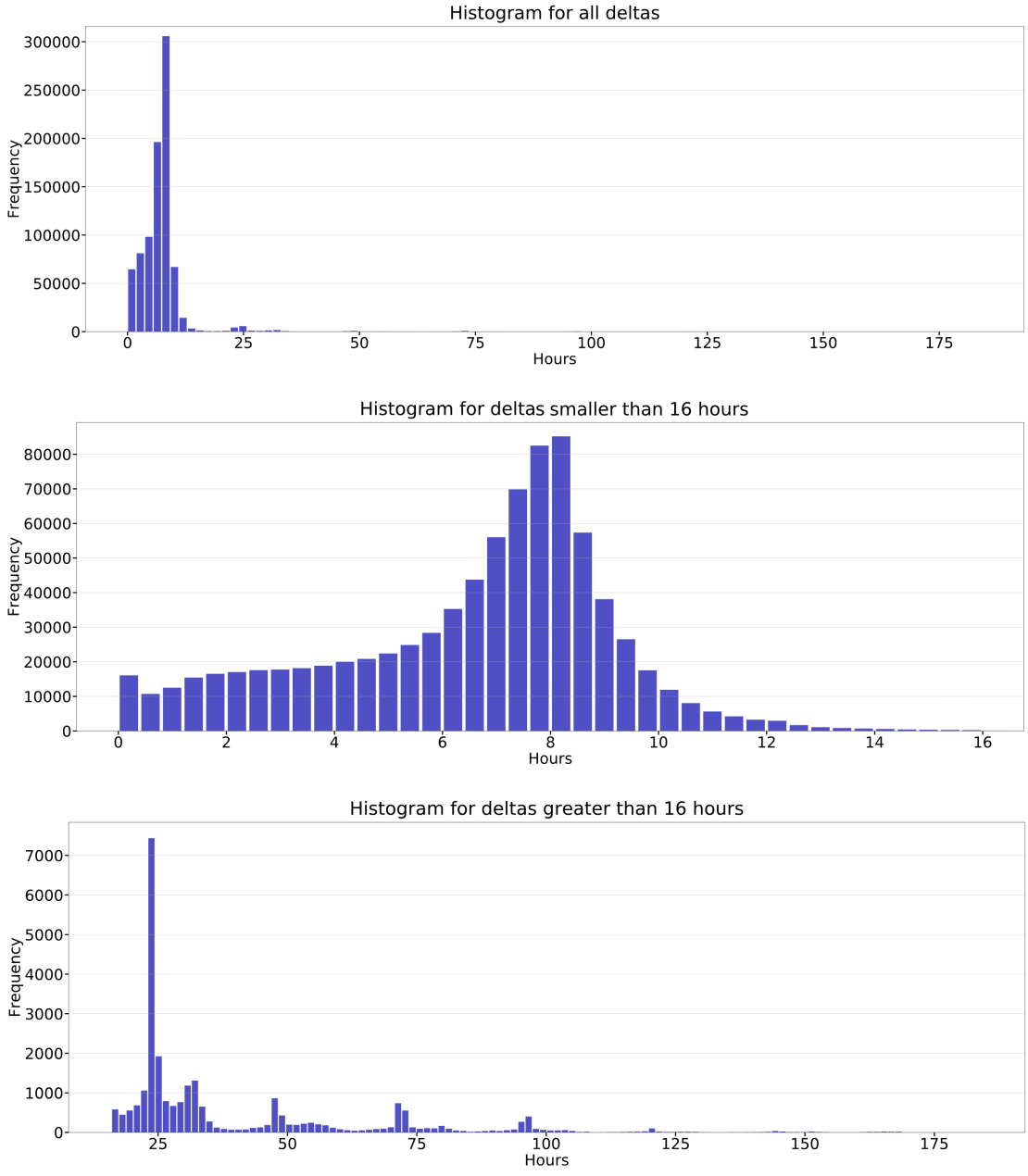


Figure 11: Histograms for $ParkingTransactions$ deltas.

HourlyWeather

HourlyWeather contains data for every hour spanning from 00:00 1 October 2016 to 23:00 28 February 2019. To demonstrate the correctness of the data, average values of temperature, precipitation and humidity measurements per day are shown in Figure 12. Temperature is in the form of degrees Celsius. Precipitation is given as millimeters. When precipitation is in the form of snow, the millimeters are measured after melting the snow. One millimeter precipitation is roughly one centimeter snow, depending on the type of snow. Humidity is the relative humidity given as a percentage.

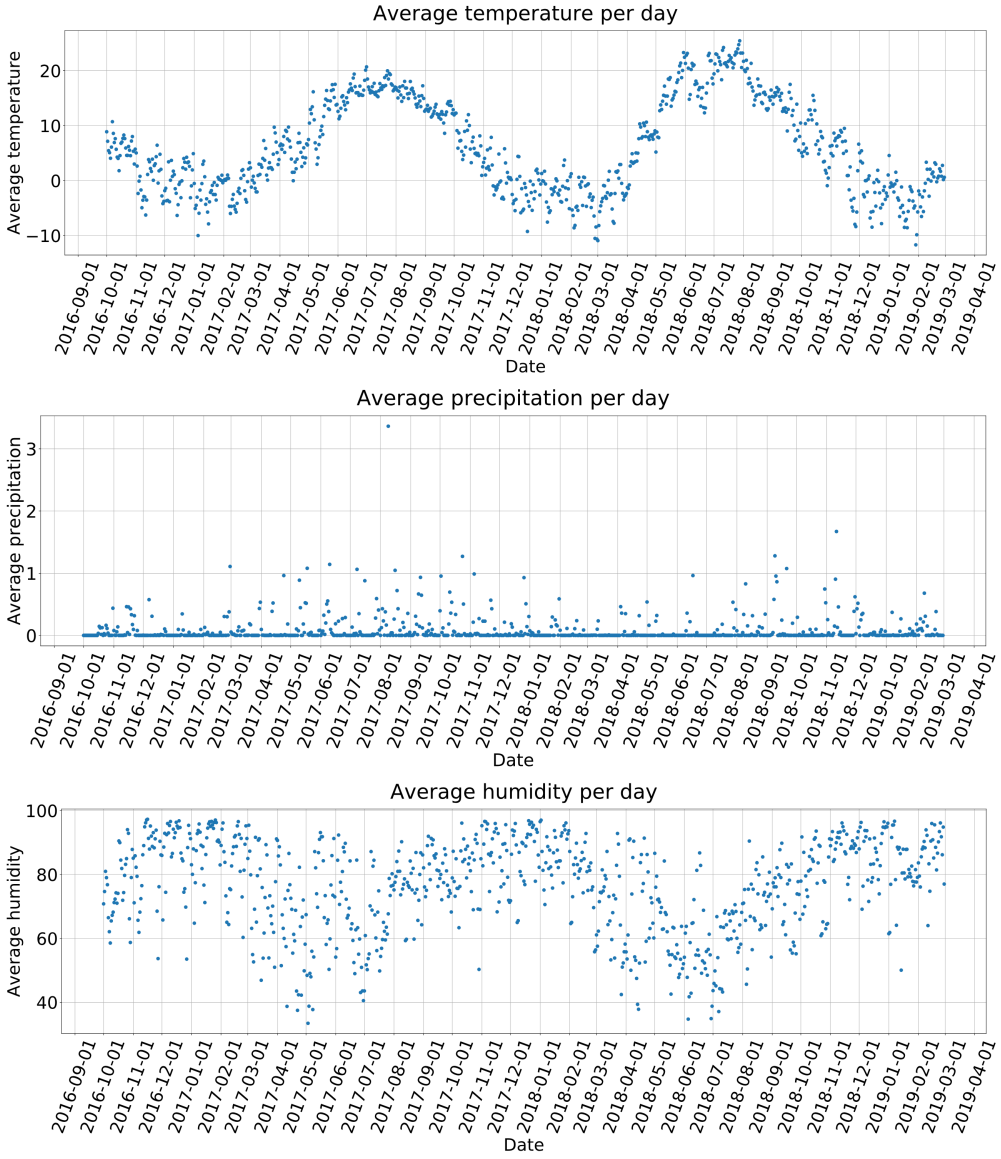


Figure 12: Average value for hourly weather measurements per day.

Label Definition

In chapter 4, forecasts for the aggregate number of transactions over the three cafeterias Eat the Street, Fresh 4 U and Soup & Sandwich will be made, in addition to individual forecasts for each cafeteria.

The label for the aggregates forecast is defined as the number of transactions in $CafeteriaTransactions$ that were recorded between 10:00 and 14:00 in any of the three cafeterias Eat the Street, Fresh 4 U and Soup & Sandwich. This label will later be referred to as "aggregate label".

The label for the individual forecasts for a given cafeteria x is defined as the number of transactions in $CafeteriaTransactions$ that were recorded between 10:00 and 14:00 at cafeteria x. The label for cafeteria x will later be referred to as "cafeteria x label".

When the word "count" is used without elaboration, it will refer to a given value for the label.

Train-test Split

For aggregate and individual forecasts, the selected instances will be split in a 50-50 train-test split with the earliest 50% of instances being training data. Evaluation will be done in an expanding window process that goes chronologically through the testing set to simulate a real application of the forecasting method.

Only the training set for the aggregate forecast will be used for feature selection and model selection. The testing set will not be used in order to prevent overfitting to the instances models are evaluated on.

Selection of Days for the Aggregate Forecast

There are certain days that will be ignored when making forecasts because the label is irregular and the day's inclusion would introduce noise to the training and evaluation process, thus reducing models' accuracy for normal days and making the model scores less comparable.

$CafeteriaTransactions$ contains rows for several days that would be classified as "inneklemt dag" in Norwegian or "Brückentag" in German, with the latter literally translating to "bridge day". These days will therefore be referred to as bridge days from now on. Bridge days are single days in between two days off. Employees tend to take bridge days off to get a continuous period away from work. The exact definition of bridge day will be the following:

> *Bridge day definition*: Let "single-day holiday" be defined as any of the following days: Labor Day, Constitution Day, Ascension Day and Whit Monday. Let "day off" be defined as any day that is either a day during Easter holidays, a day during Christmas holidays, a single-day holiday or a weekend. A "bridge day" will be defined as a day preceded and followed by a day off.

There are several labels that deviate strongly from surrounding labels, such as labels for holidays and bridge days and a day in January 2019 with twice as many transactions as the surrounding days. The definition of outliers was based on the label's distance from the mean of the surrounding period in terms of interquartile range. The outlier definition itself was not affected by holidays or bridge days. The outlier detection function DetectOutliers is shown

in Algorithm 1. For the aggregate forecasts, the dates with outlier labels were defined as DETECTOUTLIERS(Aggregate, 1.5).

Algorithm 1 Function for detecting outlier labels

1: **function** DETECTOUTLIERS($cafeteria$, $threshold$)
2: $labels \leftarrow$ all labels for $cafeteria$, except bridge days, holidays and days where the label is 0
3: $dates \leftarrow$ all dates in the dataset, except holidays and days where the label is 0
4: Let $get\_outlier\_dates$ be the IQR-based function for detecting outliers which is defined in listing A.1
5: $outlier\_dates \leftarrow get\_outlier\_dates(labels, dates, threshold)$,
6: **return** $outlier\_dates$
7: **end function**

All labels for the aggregate forecasts are shown and classified in Figure 13. The points on the graph have the style of the highest category on the legend that the point belongs to.

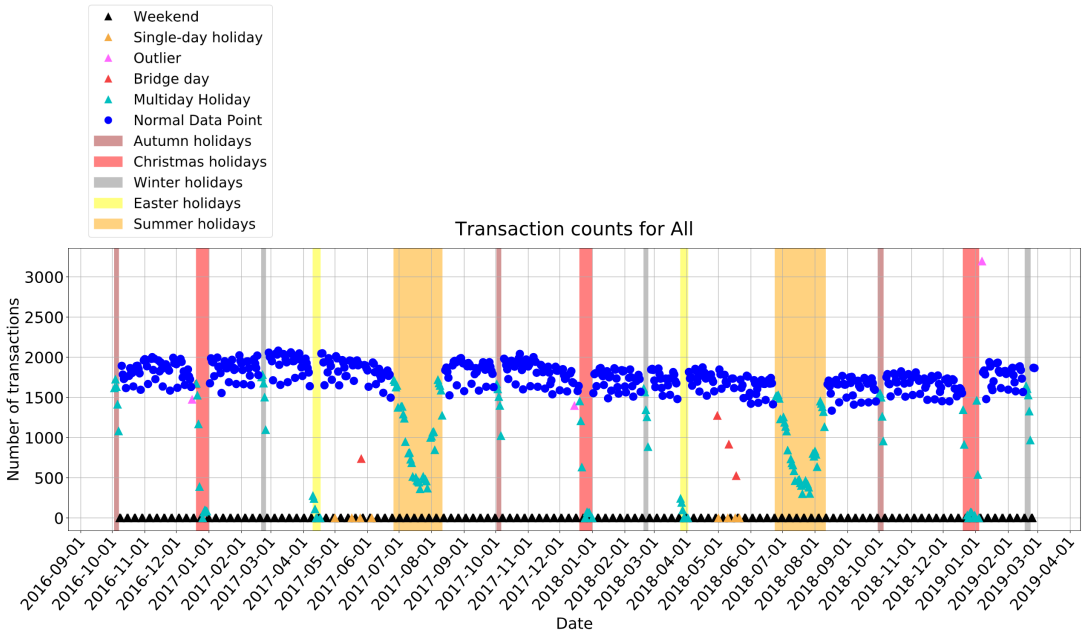


Figure 13: Labels for the aggregate forecasts.

The set $AggregateLabels$ of aggregate labels for which there will be made aggregate forecasts is the set of labels shown as normal data points in Figure 13.

The reason holidays were ignored was that the dataset did not contain a large enough span of time. Holidays cause yearly repeating patterns in the label value. All statistical methods implemented could have handled holidays to some degree by using values preceding the target by one year or by using the values immediately preceding the target. All machine learning methods implemented could have handled holidays to some degree by using holiday-related features. Despite this, holidays were removed because the dataset could not be split into a train-

test split of separate non-shuffled sets such that one part of the training set is used to learn yearly patterns, one part of the training set is used to evaluate performance over an entire year and the test set is used to evaluate performance over an entire year.

Bridge days were ignored for a similar reason to the previous paragraph: there would not be enough bridge days in each part of the train-test split. Machine learning methods could in theory have handled bridge days with bridge day features and the SARIMA model could have been expanded to a SARIMAX model taking bridge days into account by using exogenous variables.

The set of training labels for aggregate forecasts will be referred to as $AggregateLabelsTrain$ and the set of testing labels for aggregate forecasts will be referred to as $AggregateLabelsTest$. With a 50-50 train-test split of the selected instances, the dates for $AggregateLabelsTrain$ span from from 10 October 2016 to 4 December 2017 and the dates for $AggregateLabelsTest$ span from 5 December 2017 to 26 February 2019.

Selection of Instances for Individual Forecasts

Selection of instances for individual forecasts was done identically to selection of instances for the aggregate forecasts, with the exception of detecting outliers differently. All labels for each cafeteria are shown in Figure 14. There are days where one of the cafeterias is closed, but the aggregate forecast label for the day is not an outlier. These days are seen as exceptions that should be ignored in individual forecasts. Outliers days were defined as any day that is returned at least once when running DetectOutliers($cafeteria$, 1.7) for all $cafeteria$ in {Eat the Street, Fresh 4 U, Soup & Sandwich}. In other words, outlier labels for individual forecasts are defined as the result of DetectOutliersIndividual in Algorithm 2.

Algorithm 2 Function for detecting outlier labels for individual forecasts

```
1: function OutlierDetectionIndividual
2:     outlier_dates ← {}
3:     for each cafeteria ∈ {Eat the Street, Fresh 4 U, Soup & Sandwich} do
4:         outlier_dates ← outlier_dates ∪ DetectOutliers(cafeteria, 1.7)
5:     end for
6: end function
```

The set of dates for which there will be made individual forecasts is the set of dates where the label is shown as a normal data point in Figure 14. This set will be referred to as $IndividualDates$. The set of labels for these dates will be referred to as $IndividualLabels$.

With a 50-50 train-test split of the selected instances, the training set dates for individual forecasts span from 10 October 2016 to 4 December 2017 and the training set dates span from from 5 December 2017 to 26 February 2019. Let $IndividualLabelsTrain$ and $IndividualLabelsTest$ be the training and testing labels respectively.

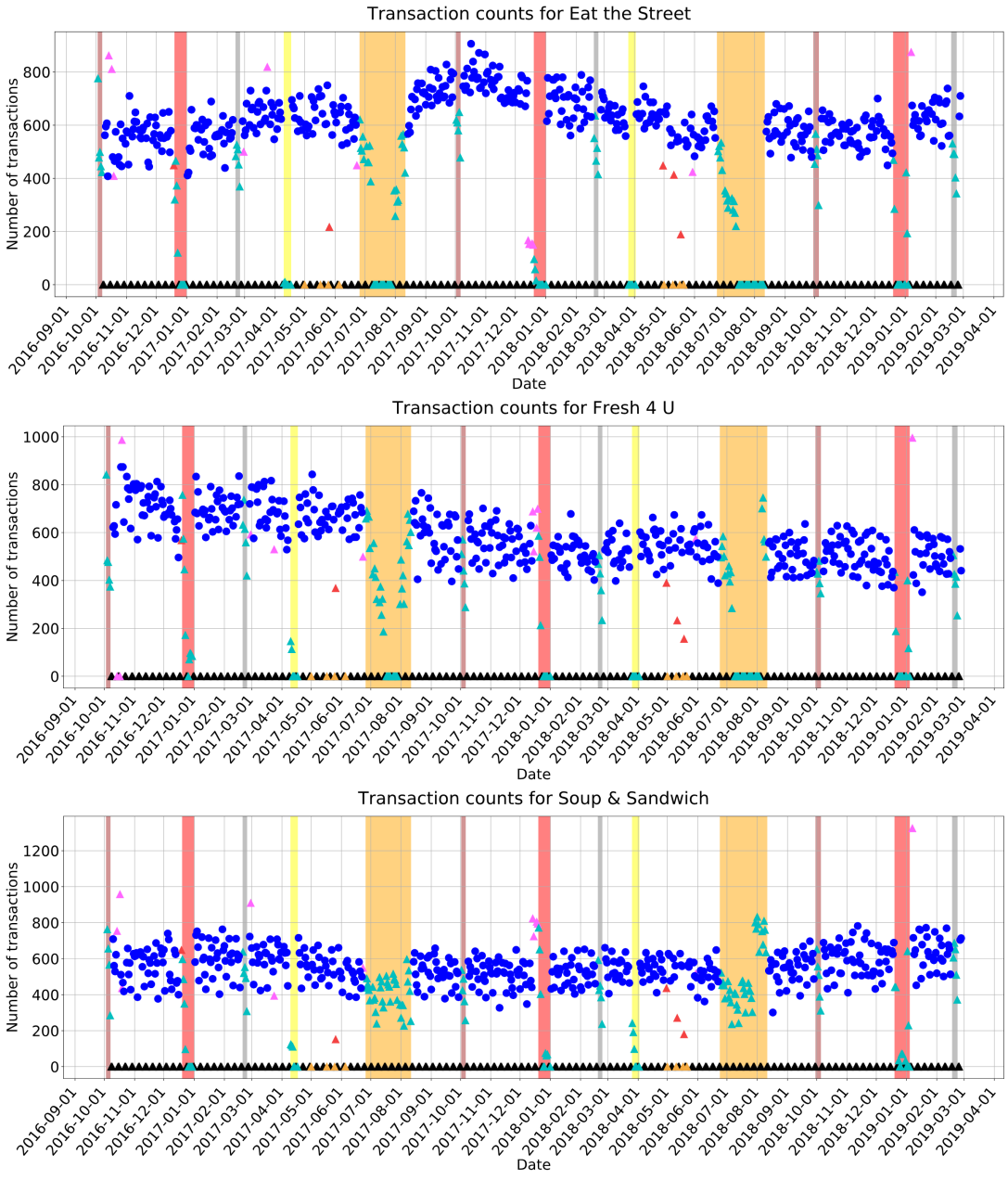


Figure 14: Labels for the individual forecasts.

Seasonality of the Labels

Figure 15 shows an autocorrelation plot (ACF) and a partial autocorrelation plot (PACF) for the aggregate label for the dates in $AggregateDatesTrain$. There is a clear 5-lag seasonality visible on the ACF. Each label has a strong positive correlation with labels $5n$ instances before, where n is a positive integer. The correlation is stronger for lower n. Since there are no weekends, a lag of five corresponds to one week. The PACF shows that most of the correlation can be explained by the correlation for lags of five because the partial autocorrelation is weak for higher lags. This shows that the data has weekly seasonality. The partial autocorrelations above the significance level around a lag of 115 could be due to biannual seasonality, but there are several partial autocorrelation values of similar magnitude for earlier lags, which are likely to be due to chance. Correct?

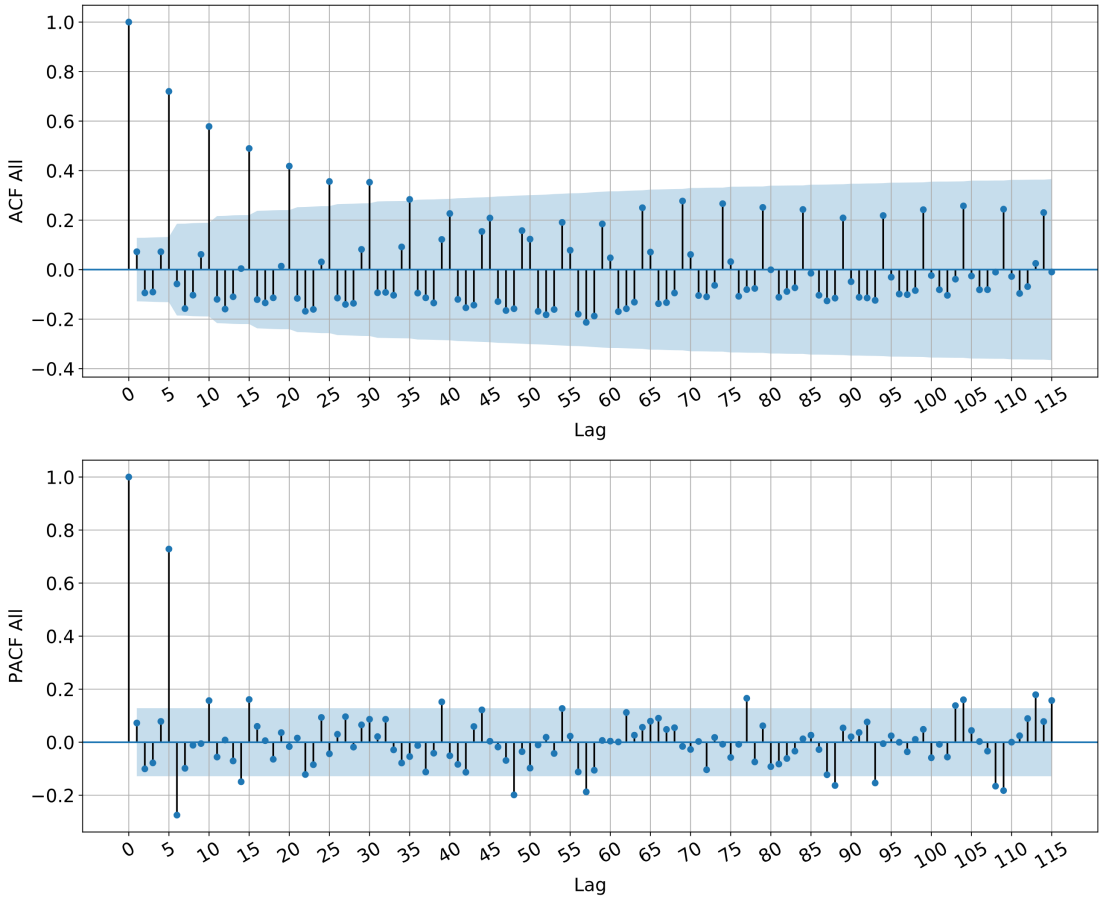


Figure 15: Autocorrelation and partial autocorrelation plots for the aggregate label for the dates in $AggregateDatesTrain$.

Figure 16 shows an ACF and a PACF for the Eat the Street label for the dates in $IndividualDatesTrain$. For Eat the Street, the label's correlation is strongest with the immediately preceding values and

the correlation sinks until below significance at lag 5.

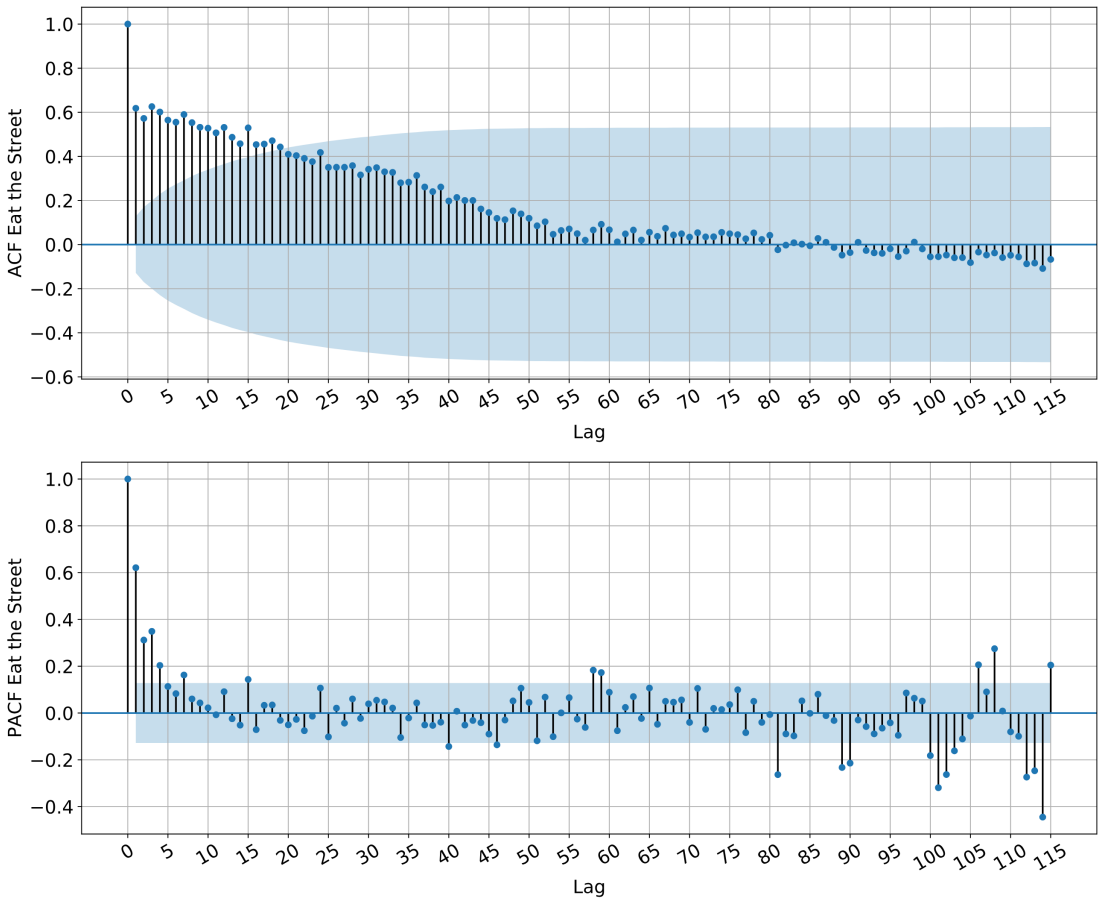


Figure 16: Autocorrelation and partial autocorrelation plots for the Eat the Street label for the dates in $IndividualDatesTrain$.

Figure 17 shows an ACF and a PACF for the Fresh 4 U label for the dates in $IndividualDatesTrain$. Fresh 4 U has a similar ACF and PACF to the aggregate forecast, but with a PACF value at lag 1 that is almost as large as lag 5. This indicates that the label one weekday before is a better predictor than the label five weekdays before for the label value on a given day for this cafeteria.

Figure 18 shows an ACF and a PACF for the Soup & Sandwich label for the dates in $IndividualDatesTrain$. Soup & Sandwich also has a similar ACF and PACF to the aggregate forecast.

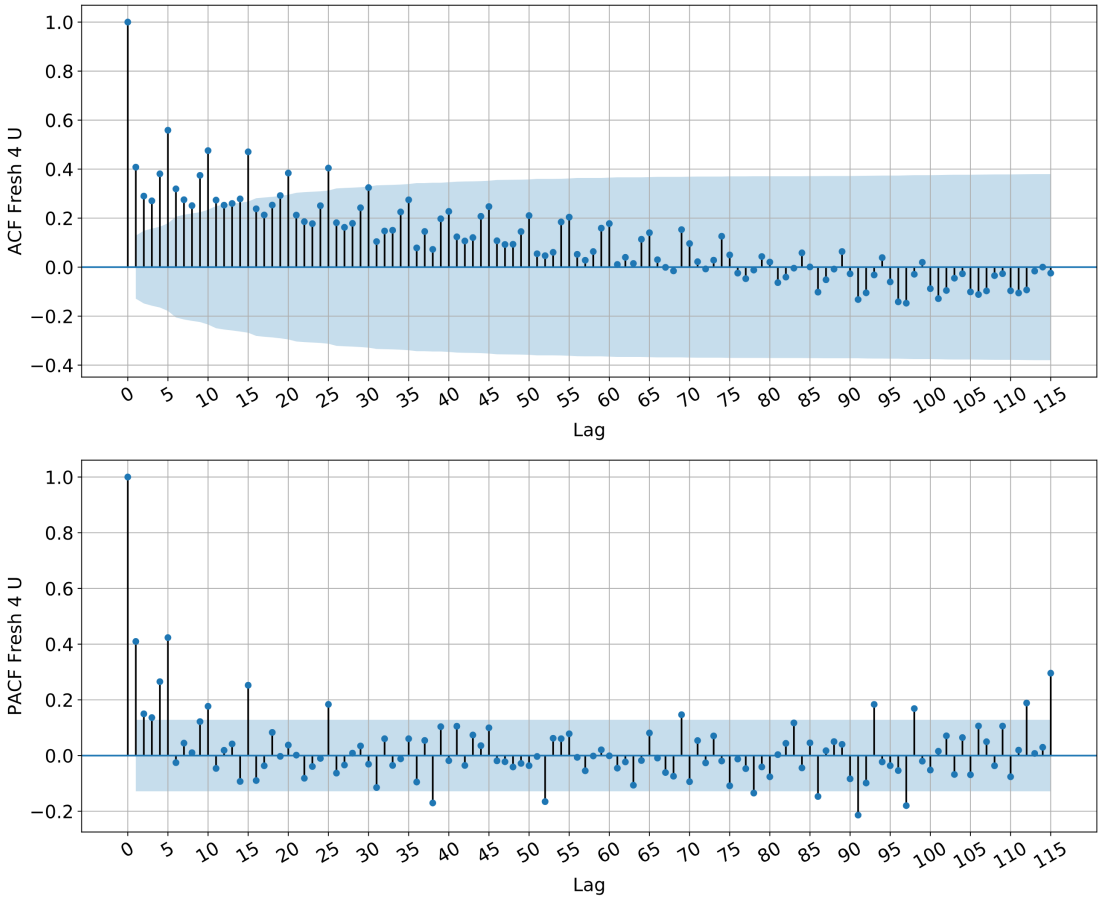


Figure 17: Autocorrelation and partial autocorrelation plots for the Fresh 4 U label for the dates in $IndividualDatesTrain$.

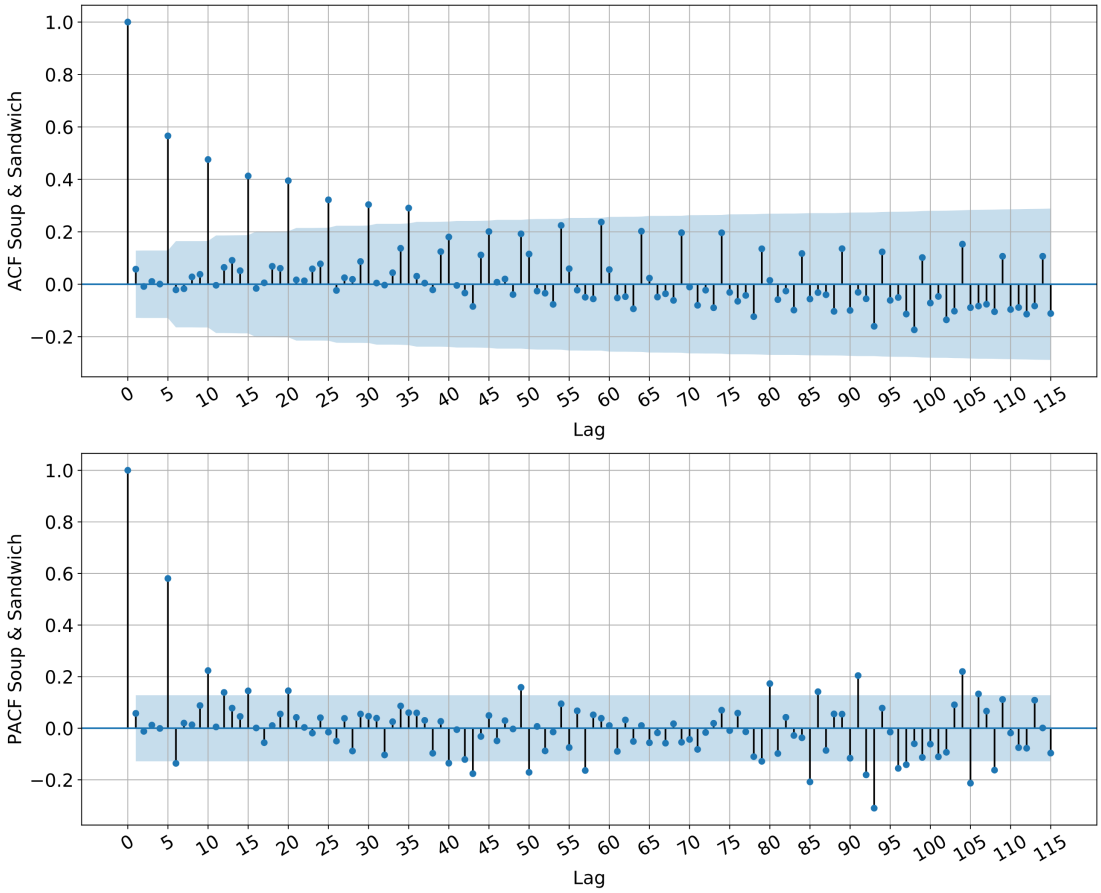


Figure 18: Autocorrelation and partial autocorrelation plots for the Soup & Sandwich label for the dates in $IndividualDatesTrain$.

Historical Features

Two types of historical features were extracted from the aggregate label values of previous instances: "count x weekdays before" and "average x weeks before".

"count x weekdays before" means the count that was recorded x weekdays before. For example, if the count is 1700 on a Friday, then "count 1 weekday before" is 1700 for the following Monday. During feature extraction, when the count x weekdays before was missing, the count $x+5n$ weekdays before was used as the feature value instead, where n is the lowest integer for which the count $x+5n$ weekdays before is not missing. This type of imputation was chosen because of the weekly seasonality of counts.

"average x weeks before" is the average count over all weekdays in the week x weeks before the instance. For example, for a given Tuesday, the "average 1 week before" value is the average count on Monday through Friday in the preceding week. During feature extraction, if the week x weeks before the instance was missing a count for a weekday, then the average $x+n$ weeks before was used as the feature value instead, where n is the lowest integer such that there are counts for all weekdays in the week $x+n$ weeks before.

Parking Features

The final parking features were of the form "parked cars total 'hour of the day'", with "total" meaning that the number is including parked cars in all three parking garages. These features were created in a process explained in detail in Algorithm 3. Figure 19 shows the result of plotting the label against the parking features using the first half of the dataset. The figure reveals a cluster of outlier values for the parking features, correponding to the outliers around February 2017 in Figure 9. Outliers were detected by selecting all values more than 1.5 times the interquartile range away from the mean value of the feature in the training data, which is the the same outlier definition as shown in detail in Listing A.1. During feature preprocessing, missing or outlier parking features were imputed by using the mean training data value for the feature. For example, a missing or outlier value in "parked cars 06:00" would be replaced by the mean of all "parked cars 06:00" values in the training data. The scatter plots of parking features against the label in Figure 21 are equivalent to the scatter plots in Figure 19 after removing the outliers and imputing values.

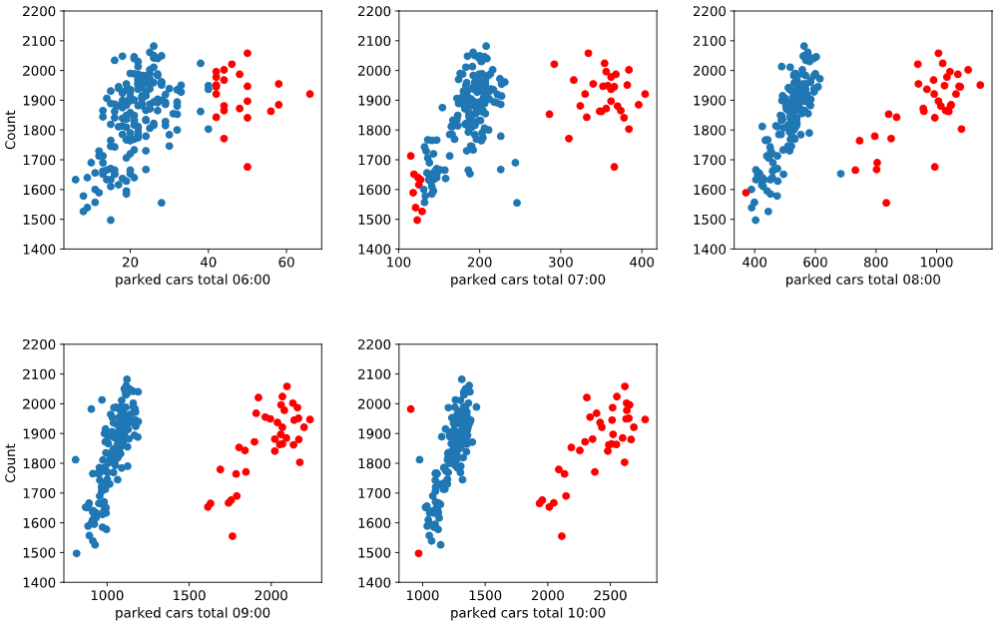


Figure 19: Plots of the label against parking features in the first 50% of the dataset. Points where the parking value is classified as an outlier are red, while the rest are blue.

Algorithm 3 Process of creating parking features by iterating chronologically through events

```
1:  procedure ParkingFeatureCreation
2:      Remove values in ParkingTransactions where end time is earlier than start time
3:      Let Q be an empty list
4:      for each row in ParkingTransactions do
5:          Based on the values of row, add a "entry at time X" event to Q and a "exit at time
            X" event to Q. If the exit time is later than midnight of the entry day, set the exit
            time to midnight
6:      end for
7:      For every day in a span containing the dates in CafeteriaTransactions, add a "Write
        at time X" event for times 06:00, 07:00, ..., 12:00 to Q
8:      parkedCars ← 0
9:      Let CSV be a CSV with columns "date", "parked cars total 06:00", "parked cars total
        07:00", ..., "parked cars total 12:00"
10:     currentLine ← an empty string
11:     Sort Q such that events with earlier times are first
12:     for each event in Q do
13:         if event is an entry event then
14:             parkedCars ← parkedCars + 1
15:         end if
16:         if event is an exit event then
17:             parkedCars ← parkedCars − 1
18:         end if
19:         if event is a write event then
20:             Append parkedCars to currentLine
21:             if event's time of day is not 12:00 then
22:                 Append "," to currentLine
23:             else
24:                 currentDate ← the date of event
25:                 Prepend currentDate followed by "," to currentLine
26:                 Append "\n" to currentLine
27:                 Write currentLine to CSV
28:             end if
29:         end if
30:     end for
31: end procedure
```

Weather Features

The weather features extracted from $HourlyWeather$ are "average lunchtime temperature", "maximum lunchtime precipitation" and "average lunchtime humidity". Creation of weather features was a straightforward process of computing averages or maximum values of hourly measurements at 10:00, 11:00, 12:00, 13:00 and 14:00 for each day. The maximum value was chosen for precipitation under the assumption that large amounts of precipitation in bursts have a larger impact on how weather is perceived than small continuous amounts.

Date Features

Date features are features extracted directly from the date of the instance.

"day of week" was chosen because of the strong weekly seasonality that was shown.

"day of month" was used because some of the works in Section 2.1 used it as a feature. The day of the month could have an effect due to anything that repeats monthly such as monthly events or payout of salaries.

"week of the month" could be more useful for machine learning methods than "day of the month" by grouping instances together to capture trends for larger time spans. "week of the month" is the calendar month rather than "how many seven day periods into the month". A few instances were in the sixth week of the month, and these instances' "week of the month" was set to five because most months contains five calendar weeks.

A feature called "semester factor 1" was created to capture a pattern repeating every semester. Figure 13 shows that labels tend to be higher closer to the middle of the semester. semester factor 1 is a number between 0 and 100 saying how close to the end of the semester the instance is. The formula for semester factor 1 is $\frac{delta\_first\_to\_instance}{delta\_first\_to\_last} \times 100$, where $delta\_first\_to\_instance$ is the number of days between the first instance of the semester and the instance for which semester factor 1 is being calculated for and $delta\_first\_to\_last$ is the number of days between the first instance of the semester and the last instance of the semester.

A feature called "semester factor 2" was created to capture a pattern repeating every semester, but in the opposite direction every time, and to support the weather features. "semester factor 2" is a number between 0 and 100 saying how close to the summer the instance is. "semester factor 2" is defined as equal to "semester factor 1" if the instance is in a spring semester, otherwise $100-$ "semester factor 1".

Feature Extraction Result

Table 6 shows the final set of features and their categories.

Category	Feature name
Historical features	count 1 weekday before count 2 weekdays before count 3 weekdays before count 4 weekdays before count 5 weekdays before count 15 weekdays before average 1 week before average 2 weeks before average 3 weeks before average 4 weeks before
Parking features	parked cars total 06:00 parked cars total 07:00 parked cars total 08:00 parked cars total 09:00 parked cars total 10:00
Weather features	average lunchtime temperature maximum lunchtime precipitation average lunchtime humidity
Date features	day of week day of month week of month semester factor 1 semester factor 2

Table 6: All features categorized by their source.

Classification into Categorical and Numerical

Another way to classify the features is into categorical and numerical. This will be used when one-hot encoding them in machine learning experiments. Table 7 shows each feature categorized as either categorical or numerical.

Category	Feature name
Categorical	day of week day of month week of month
Numerical	count 1 weekday before count 2 weekdays before count 3 weekdays before count 4 weekdays before count 5 weekdays before count 15 weekdays before average 1 week before average 2 weeks before average 3 weeks before average 4 weeks before parked cars total 06:00 parked cars total 07:00 parked cars total 08:00 parked cars total 09:00 parked cars total 10:00 average lunchtime temperature maximum lunchtime precipitation average lunchtime humidity semester factor 1 semester factor 2

Table 7: All features classified as either categorical or numerical.

Forecast Horizons 0WDF, 2WDF and 15WDF

Forecasting methods will be divided into three categories based on their horizons. The categories will be called "zero weekday forecast" (0WDF), "two weekday forecast" (2WDF) and "15 weekday forecast" (15WDF). 0WDF, 2WDF and 15WDF methods make forecasts for the same day, two weekdays ahead and 15 weekdays ahead respectively. The forecast has to be ready before 10:00. How much earlier than 10:00 the forecast is ready depends on the method. The forecast horizon determines which features can be used. Table 8 shows which features are available for 0WDF, 2WDF and 15WDF.

Feature name	Available for 0WDF	Available for 2WDF	Available for 15WDF
count 1 weekday before	X		
count 2 weekdays before	X	X	
count 3 weekdays before	X	X	
count 4 weekdays before	X	X	
count 5 weekdays before	X	X	
count 15 weekdays before	X	X	X
average 1 week before	X		
average 2 weeks before	X	X	
average 3 weeks before	X	X	
average 4 weeks before	X	X	X
parked cars total 06:00	X		
parked cars total 07:00	X		
parked cars total 08:00	X		
parked cars total 09:00	X		
parked cars total 10:00	X		
average lunchtime temperature	X		
maximum lunchtime precipitation	X		
average lunchtime humidity	X		
day of week	X	X	X
day of month	X	X	X
week of month	X	X	X
semester factor 1	X	X	X
semester factor 2	X	X	X

Table 8: Available features for same day, two weekday and 15 weekday forecasts.

3.1.3 Feature Selection

To make sure that the machine learning model is not being confused by unnecessary noise, features should be analyzed and the useful ones should be carefully selected.

For each forecast horizon, feature selection was performed in two steps:

1. Features were initially narrowed down for each forecast horizon by examining scatter plots and correlation matrices.
2. Features were further narrowed down by removing sets of suspected noisy features one by one and calculating an average loss for each machine learning model with the new set of features. If the average loss decreased, the set of features remained removed for the rest of the

It is likely that different feature sets are better for different models because for instance, some models are more sensitive to noise, some models handle more complex relationships between features better, some models are more sensitive to highly correlated features and some models are better at learning feature interactions. Features were still chosen universally for all machine learning models because it was assumed that the average change in loss when removing a feature set from all models was a better indication of the feature set's usefulness than an evaluation of the feature individual to each model. This assumption was made for two reasons:

1. If a different feature set were to be created for every model, it would involve an evaluation of how different feature sets for each model individually. This evaluation is subject to a large amount of noise.
2. If a feature set's removal increases average loss over all models, it is an indication that the feature set carries useful information about the label.

3.2 Forecasting Methods

3.2.1 Time Series

Three statistical methods for time series forecasting will be implemented: seasonal naive, ETS and SARIMA. These methods were chosen because they have given good results in several related works, as shown in Section 2.1. Seasonal naive is a simple and quick method that performed well when [11] were forecasting the number of dinners in a university dining facility. ETS was not selected as a best method by any related work, but it often a powerful method in other time series applications.

3.2.2 Machine Learning

Six machine learning methods will be implemented: LightGBM, decision tree, random forest, SVR, XGBoost and ANN. LightGBM is the only method that has not been used in any of the similar forecasting papers referenced in Section 2.1, but it is a similar method to XGBoost. Among methods that have been used in a referenced forecasting paper, decision tree is the only one that has not been used in a best method. No method stated that pure XGBoost was among the best methods.

Chapter 4

Experiment and Implementation

In this chapter, several questions will be answered through experiments. A feature selection experiment will be performed to find the best machine learning features for each forecast horizon. An experiment to find the best statistical forecasting method for the aggregate forecast and an experiment to find the best machine learning method for the aggregate forecast will be performed. After this experiment, all methods will be implemented for individual cafeteria forecasts to see how well the best methods perform and to see if the same methods perform best across cafeterias. An ensemble method returning the average forecast over all forecasting methods except the two worst-performing ones will be tested to see if it can perform even better than all of its constituent models. A weather feature experiment will finally be performed to confirm that removal of weather features in the feature selection experiment led to better 0WDF aggregate machine learning forecasts.

4.1 Feature Selection

4.1.1 Question

Which features should be used for each forecast horizon?

4.1.2 Experiment setup

For each forecast horizon, the goal of feature selection was to start with a set $Features$ containing all available features for the horizon and remove features until $Features$ only contains the final selection of features for the horizon. Only the training data was used during feature selection in order to prevent overfitting to the testing data. Feature selection consisted of two steps.

The first step of feature selection was to judge the available features by the scatter plot of the aggregate label against the feature and by the Pearson correlation matrix. Figures 20 and 21 show scatter plots of the aggregate label against each feature for the training data. Figure 22 shows the Pearson correlation matrix. The first step of feature selection yielded a set *Remove*

of features to remove from *Features* immediately and a set *Suspects* of features to consider for removal in the next step.

The second and final step of feature selection was a type of backward elimination and will be referred to as EWBE which stands for "expanding window backward elimination". EWBE is shown in Algorithm 4. EWBE removes features in *Suspects* from *Features* one by one and examines the change in average MSE in an expanding window process over all machine learning models. If the suspect's removal increased MSE, it was put back in *Features*. EWBE went through suspects in the order they are written in *Suspects*.

Algorithm 4 Expanding window backward elimination

INPUT: A set *Features* of features a set *Suspects* of features to try and remove
OUTPUT: A set of selected features

```
1:  function EWBE(Features, Suspects)
2:      lowest_mse ← EvaluateFeatureSet(Features)    // Algorithm 5
3:      for each suspect ∈ Suspects do
4:          Features ← Features − {suspect}
5:          current_mse ← EvaluateFeatureSet(Features)
6:          if current_mse < lowest_mse then
7:              lowest_mse ← current_mse
8:          else
9:              Features ← Features ∪ {suspect}
10:         end if
11:     end for
12:     return Features
13: end function
```

Algorithm 5 Function that evaluates each model in an expanding window process using the given feature set and returns the average MSE

INPUT: A set $Features$ of features to use
OUTPUT: Average MSE over all models using the feature set

1: **function** EVALUATEFEATURESET($Features$)
2: $models \leftarrow$ {LGBMRegressor, DecisionTreeRegressor, RandomForestRegressor, SVR, XGBRegressor, ANN}
3: $Q1 \leftarrow$ the first half of $AggregateLabelsTrain$
4: $Q2 \leftarrow$ the second half of $AggregateLabelsTest$
5: $mses \leftarrow$ {}
6: **for each** $model \in models$ **do**
7: $Q2\_pred \leftarrow$ the prediction for $Q2$ yielded by an expanding window process where $Q1$ is training labels, $Q2$ is test labels, the forecasting model is $model$, features used are $Features$ and the number of train-test splits is 10
8: $mse \leftarrow$ the MSE between $Q2$ and $Q2\_pred$
9: $mses \leftarrow mses \cup \{mse\}$
10: **end for**
11: **return** average value in $mses$
12: **end function**

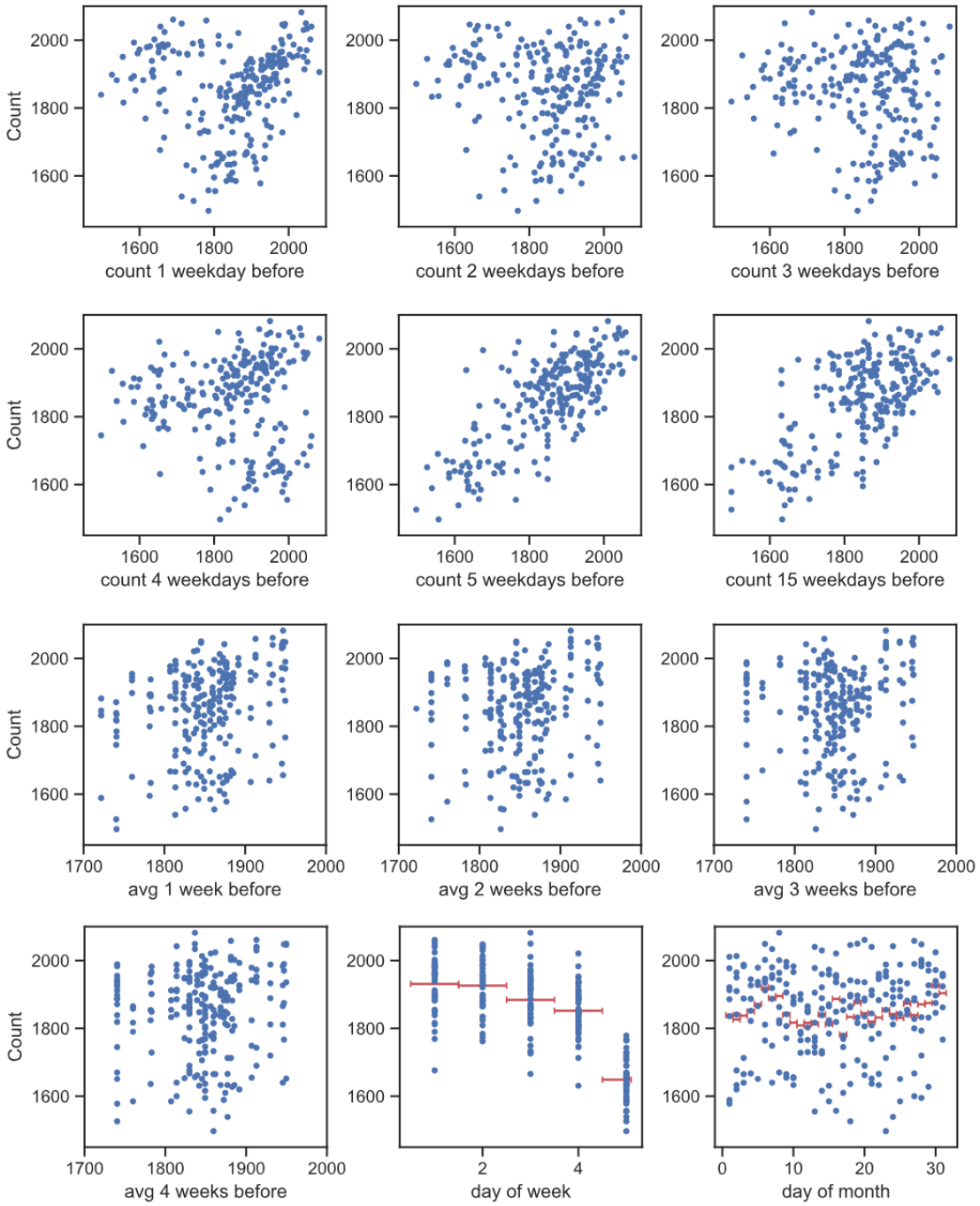


Figure 20: Scatter plots of label against features (1/2). The y-position of red lines is the average value of points within the x-span of the line.

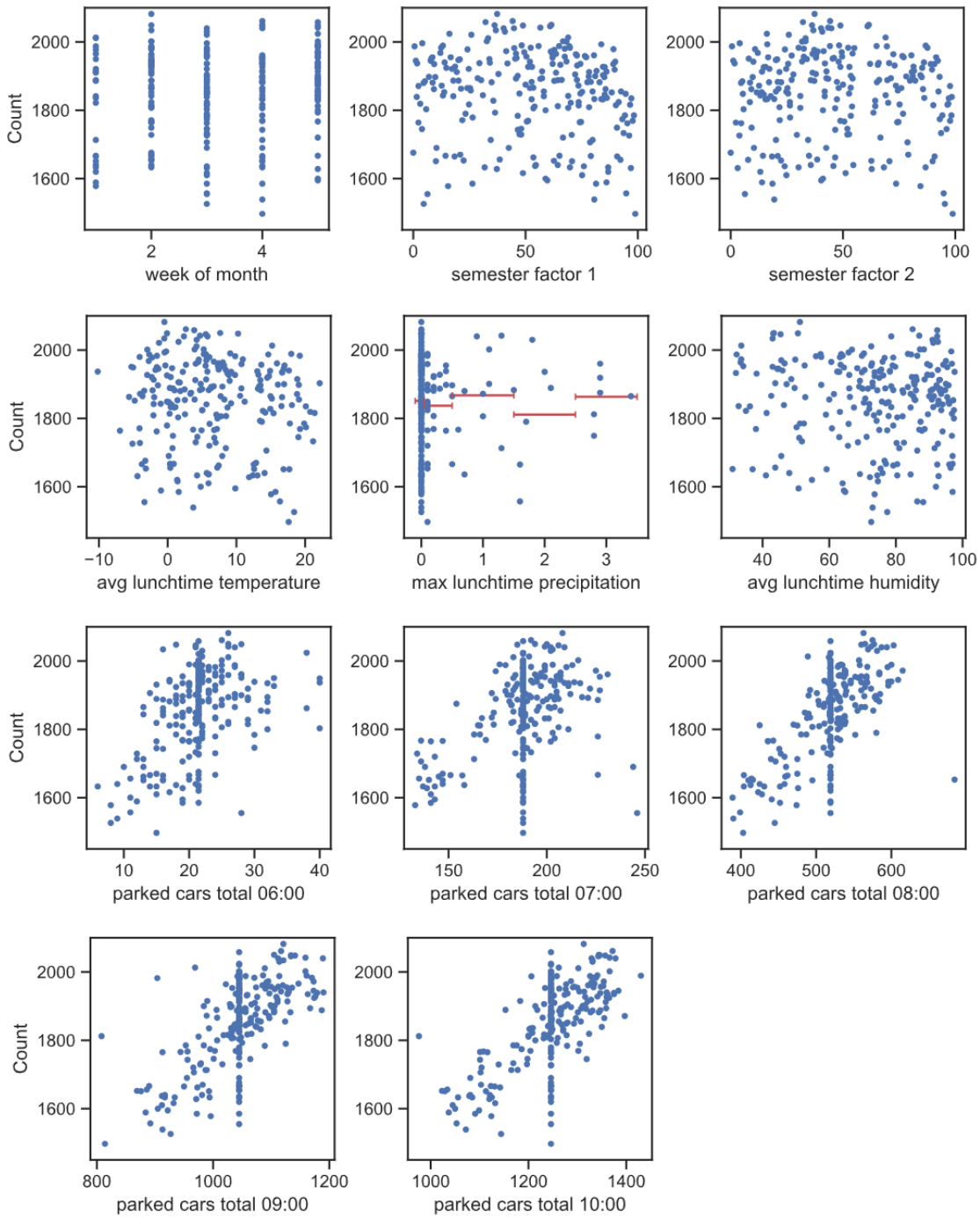


Figure 21: Scatter plots of label against features (2/2). The y-position of red lines is the average value of points within the x-span of the line.

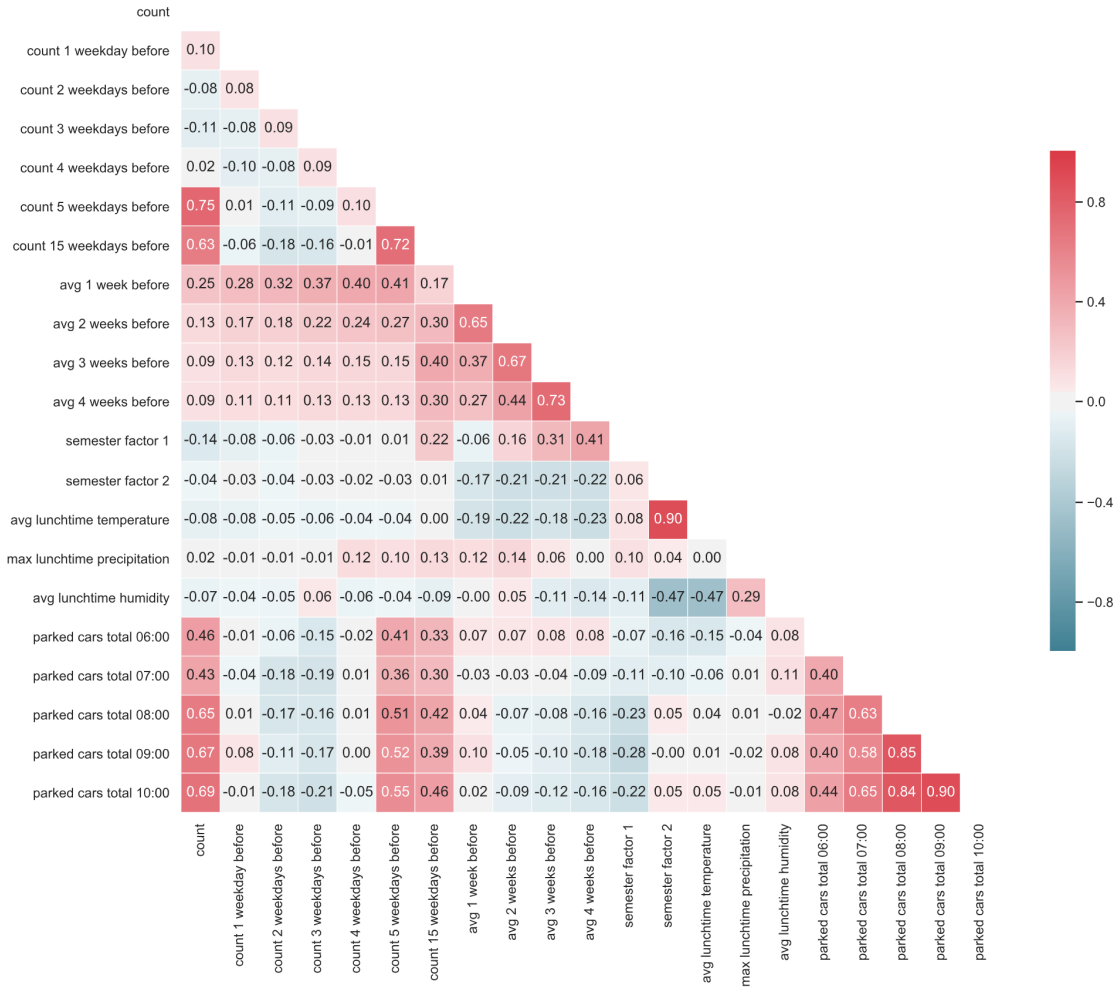


Figure 22: Pearson correlation matrix with label and features.

0WDF

Let *Features* be all available features for the 0WDF horizon as shown in Table 8.

The scatter plot for "count 1 weekday before" shows a clear non-random pattern. The scatter plots for "2 weekdays before", "3 weekdays before" and "4 weekdays before" are similar to "count 1 weekday before", but more spread. These four features have low correlations with the label. "count 5 weekdays before" and "count 15 weekdays before" are highly correlated, which corresponds well with the strong relationship shown in the scatter plot with the label and "day of week". "count 15 weekdays before" is assumed to not carry much useful information not contained in "count 5 weekdays before" because it is a similar value, only with a longer lag. "avg 1 week before", "avg 2 weeks before", "avg 3 weeks before" and "avg 4 weeks before" are highly correlation with each other. It is not likely that the ones further back in time carry useful information in addition to the ones further ahead in time. Therefore, only "avg 1 week" before will be kept.

The scatter plots for "day of month", "avg lunchtime temperature", "max lunchtime precipitation" and "avg lunchtime humidity" do not show clear patterns. It is likely that "avg lunchtime temperature", "max lunchtime precipitation", "avg lunchtime humidity" and "semester factor 2" can be viewed together to predict whether the weather is perceived as good or bad by employees, but it is not certain that any of the machine learning models can make significant use of that relationship. For that reason, the features' collective inclusion will be tested in the removal test.

Parked car features are highly correlated with each other. The ones later in the day are more correlated with the label. Since the correlation between "parked cars total 08:00" and the aggregate label is almost as high as the correlation between "parked cars total 10:00" and the aggregate label, only "parked cars total 08:00" will be kept such that the simulated forecast would be available earlier in the day. "parked cars total 06:00" and "parked cars total 07:00" are assumed to not carry much useful information not contained in "parked cars total 08:00".

"semester factor 1" has a stronger relationship with the aggregate label than "semester factor 2" according to the scatter plots and the Pearson correlation. "semester factor 2" contains information about proximity to the summer, which is the opposite of proximity to semester end for autumn semesters. For this reason, "semester factor 2" might be useful in addition to "semester factor 1", so removal of each of them will be tested in a removal test.

The final result of feature selection step 1 is:

- $Remove \leftarrow$ {"2 weekdays before", "3 weekdays before", "4 weekdays before", "count 15 weekdays before", "avg 2 weeks before", "avg 3 weeks before" and "avg 4 weeks before", "day of month", "parked cars total 06:00", "parked cars total 07:00", "parked cars total 09:00", "parked cars total 10:00"}
- $Suspects \leftarrow$ {"1 weekday before", {"avg lunchtime temperature", "max lunchtime precipitation" and "avg lunchtime humidity"}, "semester factor 2", "semester factor 1"}

Running EWBE gives:

1. EvaluateFeatureSet(*Features*) returned an MSE of 3460.
2. "count 1 day before" was removed from *Features*

3. EvaluateFeatureSet(*Features*) returned an MSE of 3647
4. "count 1 day before" was put back in *Features*
5. {"avg lunchtime temperature", "max lunchtime precipitation" and "avg lunchtime humidity"} were removed from *Features*
6. EvaluateFeatureSet(*Features*) returned an MSE of 3420
7. "semester factor 1" was removed from *Features*
8. EvaluateFeatureSet(*Features*) returned an MSE of 3519
9. "semester factor 1" was put back in *Features*
10. "semester factor 2" was removed from *Features*
11. EvaluateFeatureSet(*Features*) returned an MSE of 3520
12. "semester factor 2" was put back in *Features*

According to EWBE, only the weather features are irrelevant among the suspects. Inclusion of both "semester factor 1" and "semester factor 2" could indicate that they contain useful independent information, with the former capturing trends based on how far into the semester the date is and the latter capturing trends based on how close to the summer the date is. Inclusion of "semester factor 1" and "semester factor 2" could also have happened by chance or because the ANN is using the factors to overfit to the validation data.

2WDF

Let *Features* be all available features for the 2WDF horizon as shown in Table 8.

Since "count 1 weekday before" is not available, "count 2 weekdays before" will be a suspect for the removal test, while "count 3 weekdays before" and "count 4 weekdays" before will be removed.

"count 15 weekdays before" will be removed due to similar reasoning as in Section 4.1.2.

Since "avg 1 week before" is not available, "avg 2 weeks before" will be used and "avg 3 weeks before" and "avg 4 weeks before" will be removed.

"semester factor 1" and "semester factor 2" will be suspects for removal. One hypothesis is that their inclusion in Section 0WDF was by chance. Another hypothesis is that "semester factor 2" is connected to the weather features that are not available for this forecast horizon.

"day of month" will be removed because it does not show a clear pattern.

- *Remove* ← {"3 weekdays before", "4 weekdays before", "count 15 weekdays before", "avg 3 weeks before", "avg 4 weeks before", "day of month"}
- *Suspects* ← {"2 weekdays before", "semester factor 1", "semester factor 2"}

Running EWBE gives:

1. EvaluateFeatureSet(*Features*) returned an MSE of 4365
2. "count 2 weekdays before" was removed from *Features*
3. EvaluateFeatureSet(*Features*) returned an MSE of 4361
4. "semester factor 1" was removed from *Features*
5. EvaluateFeatureSet(*Features*) returned an MSE of 5038
6. "semester factor 1" was put back in *Features*

7. "semester factor 2" was removed from *Features*
8. EVALUATEFEATURESET(*Features*) returned an MSE of 4445
9. "semester factor 2" was put back in *Features*

15WDF

Let *Features* be all available features for the 15WDF horizon as shown in Table 8.

"semester factor 1" and "semester factor 2" will be suspects for removal due to the same reasoning as in Section sec:feature-selection-2d-question.

- $Remove \leftarrow \{\}$
- $Suspects \leftarrow$ {"semester factor 1", "semester factor 2"}

Running EWBE gives:

1. EVALUATEFEATURESET(*Features*) returned an MSE of 5310
2. "semester factor 1" was removed from *Features*
3. EVALUATEFEATURESET(*Features*) returned an MSE of 5320
4. "semester factor 2" was removed from *Features*
5. EVALUATEFEATURESET(*Features*) returned an MSE of 5274

"semester factor 1" was kept, like for 2WDF, giving more support to the hypothesis that "semester factor 2" is more useful together with weather information, but the selection of semester factors to include could be due to inherent randomness or other mechanism than the one hypothesized.

4.1.3 Result

The selected features for each forecast horizon are shown in Table 9.

Feature name	0WDF available	0WDF selected	2WDF available	2WDF selected	15WDF available	15WDF selected
count 1 weekday before	X	X				
count 2 weekdays before	X		X			
count 3 weekdays before	X		X			
count 4 weekdays before	X		X			
count 5 weekdays before	X	X	X	X		
count 15 weekdays before	X		X		X	X
average 1 week before	X	X				
average 2 weeks before	X		X	X		
average 3 weeks before	X		X			
average 4 weeks before	X		X		X	X
parked cars total 06:00	X					
parked cars total 07:00	X					
parked cars total 08:00	X	X				
parked cars total 09:00	X					
parked cars total 10:00	X					
average lunchtime temperature	X					
maximum lunchtime precipitation	X					
average lunchtime humidity	X					
day of week	X	X	X	X	X	X
day of month	X		X		X	
week of month	X	X	X	X	X	X
semester factor 1	X	X	X	X	X	
semester factor 2	X	X	X	X	X	

Table 9: Available and selected features for same day, two weekday and 15 weekday forecasts.

4.2 Aggregate Forecast with Statistical methods

4.2.1 Question

Which statistical method gives the forecast for the aggregate label with the lowest MSE for each specific forecast horizon?

4.2.2 Experiment setup

Naive

The aggregate label has strong 5-weekday seasonality as shown in the PACF plots in Section 3.1.2 and the scatter plot of the aggregate label against the "count 5 weekdays before" feature in Section 4.1. For this reason, seasonal naive with season length 5 (SN5) will be used for the 0WDF aggregate naive forecast and for the 2WDF aggregate naive forecast, and seasonal naive with season length 15 (SN15) will be used for the 15WDF aggregate naive forecast. Table 10 shows which type of naive forecast was chosen for each horizon for the aggregate forecast, along with naive forecast types for later experiments.

Context	Naive forecast type
0WDF, Aggregate	SN5
0WDF, Eat the Street	SN1
0WDF, Fresh 4 U	SN5
0WDF, Soup & Sandwich	SN5
2WDF, Aggregate	SN5
2WDF, Eat the Street	SN3
2WDF, Fresh 4 U	SN5
2WDF, Soup & Sandwich	SN5
15WDF, Aggregate	SN15
15WDF, Eat the Street	SN15
15WDF, Fresh 4 U	SN15
15WDF, Soup & Sandwich	SN15

Table 10: Naive forecast types for each cafeteria and horizon.

In order to ensure that missing values do not cause the naive forecasts to go out of phase, the naive forecast will be made for a time series where values have been imputed such that there is a value for the entire work week iff. there is at least one label for the work week in $AggregateLabels$. The relevant forecasts will be extracted from the forecast for the imputed time series and used as the final forecast for $AggregateLabelsTest$. For the first work week where there is at least one label in $AggregateLabels$, imputation will be done by inserting the value n weeks later, where n is the lowest possible integer. For the rest of the work weeks where there is at least one label in $AggregateLabels$, imputation will be done by inserting the value n weeks earlier, where n is the lowest possible integer. The function GETLABELSIMPUTED for cre-

ating an extended time series is shown in Algorithm 6. GETLABELSIMPUTED(Aggregate) returns the imputed time series described in this paragraph, in addition to a train-test split of that time series to correspond with $AggregateLabelsTrain$ and $AggregateLabelsTest$.

Algorithm 6 Impute values for the given list of labels assuming weekly seasonality

INPUT: A value $cafeteria$ that determines cafeteria to create an imputed time series for
OUTPUT: An imputed list of labels and a train-test split of the imputed list of labels that corresponds to the train-test split for $cafeteria$

function GETLABELSIMPUTED($cafeteria$)
  $labels \leftarrow$ the labels for $cafeteria$
  $relevant\_dates \leftarrow \{\}$
  for each $date$ correponding to an $labels$ element **do**
    Add all weekdays in the same week as $date$ to $relevant\_dates$
  end for
  $labels\_imputed \leftarrow$ A collection indexed by date, with the same length as $relevant\_dates$, where every element is -1
  for each $date$ correponding to an $labels$ element **do**
    Set the element at index $date$ in $labels\_imputed$ to the the value $labels$ contains for $date$
  end for
  $first\_five\_weekdays \leftarrow$ the first five indices in $labels\_imputed$
  for each $date \in first\_five\_weekdays$ **do**
    if $labels\_imputed[date]$ = -1 **then**
      $labels\_imputed[date] \leftarrow$ the label in $labels$ n weeks later with n as small as possible
    end if
  end for
  for each $date \in relevant\_dates$ **do**
    if $labels\_imputed[date]$ = -1 **then**
      $labels\_imputed[date] \leftarrow$ the label in $labels$ n weeks earlier with n as small as possible
    end if
  end for
  $first\_test\_date \leftarrow$ the earliest date in $labels$
  $labels\_imputed\_train \leftarrow$ the part of $labels\_imputed$ where the index is earlier than $first\_test\_date$
  $labels\_imputed\_test \leftarrow$ the part of $labels\_imputed$ where the index is later than or equal to $first\_test\_date$
  return $labels\_imputed$, $labels\_imputed\_train$, $labels\_imputed\_test$
end function

ETS

Let $AggregateLabelsImputed$, $AggregateLabelsImputedTrain$ and $AggregateLabelsImputedTest$ be the respective results of GETLABELSIMPUTED(Aggregate).

The ETS implementation was an instance of the statsmodels.tsa.holtwinters.ExponentialSmoothing class [28]. Upon initialization, the class received the arguments $trend$, $damped$, $seasonal$, and

seasonal_periods and when running ExponentialSmoothing.fit, the arguments *optimized*, *use_boxcox* and *remove_bias* were given. Trend was set to None. *seasonal_periods* was set to 5. Optimized was set to True. The remaining parameters were chosen by performing a grid search and choosing the parameter combination that got the lowest MSE in an expanding window process where the first half of *AggregateLabelsImputedTrain* was training data, the second half of *AggregateLabelsImputedTrain*. The combinations were taken from the following parameter grid:

- *trend*: {None}
- *damped*: {True, False}
- *seasonal*: {"add", "mul"}
- *seasonal_periods*: {5}
- *use_boxcox*: {True, False}
- *remove_bias*: {True, False}
- *optimized*: {True}

After performing the grid search, the chosen parameters were:

- *trend*: None
- *damped*: False
- *seasonal*: "mul"
- *seasonal_periods*: 5
- *use_boxcox*: False
- *remove_bias*: False
- *optimized*: True

Evaluation was done in an expanding window loop. For the 0WDF forecast, *AggregateLabelsImputedTrain* was training data and *AggregateLabelsImputedTest* was test data. For the 2WDF forecast was similar, but with the two latest training labels unavailable when making the forecast. For the 15WDF forecast, the last 15 training labels were unavailable for making the forecast. The expanding window process is described in Algorithm 8. The forecasts for aggregate 0WDF ETS, aggregate 2WDF ETS and aggregate 15WDF ETS are StatisticalForecast(Aggregate, 0WDF, ETS), StatisticalForecast(Aggregate, 2WDF, ETS) and StatisticalForecast(Aggregate, 15WDF, ETS) respectively.

Algorithm 7 Make a forecast for the specified cafeteria and horizon using the specified statistical method

INPUT: A value $cafeteria$ that determines which cafeteria to make forecasts for, a value $horizon$ that determines the forecast horizon and a value $model$ that determines which model to use
OUTPUT: A forecast for the specified cafeteria with the specified horizon

1: **function** StatisticalForecast($cafeteria$, $horizon$, $model$)
2: $labels\_imputed$, $labels\_imputed\_train$, $labels\_imputed\_test \leftarrow$ GetLabelsImputed($cafeteria$)
3: $labels\_imputed\_test\_pred \leftarrow$ ExpandingWindowStatistical($labels\_imputed\_train$, $labels\_imputed\_test$, $horizon$, $model$)
4: $labels\_test\_pred \leftarrow$ a list containing the elements in $labels\_imputed\_test\_pred$ that correspond with the test labels for $cafeteria$
5: **return** $labels\_test\_pred$
6: **end function**

Algorithm 8 Generate an expanding window forecast for a given time series using the method and horizon specified

```
function ExpandingWindowStatistical(labels_imputed_train, labels_imputed_test, horizon, model)
    labels_imputed_test_pred ← an empty list
    if horizon is 0WDF then
        history ← labels_imputed_train
    else if horizon is 2WDF then
        removed_values ← a list containing the last 2 elements of labels_imputed_train
        history ← labels_imputed_train except the last 2 elements
    else:
        removed_values ← a list containing the last 15 elements of labels_imputed_train
        history ← labels_imputed_train except the last 15 elements
    end if
    for each i = 0, 1, 2, ..., length(labels_imputed_test) - 1 do
        fit model to history
        if horizon is 0WDF then
            prediction ← a 1 step forecast with model
        else if horizon is 2WDF then
            prediction ← a 3 step forecast with model
        else:
            prediction ← a 16 step forecast with model
        end if
        Append prediction to labels_imputed_test_pred
        if horizon is 0WDF then
            labels_imputed_test[i] to history
        else if horizon is 2WDF then
            if i < 2 then
                Append removed_values[i] to history
            else
                Append labels_imputed_test[i − 2] to history
            end if
        else:
            if i < 15 then
                Append removed_values[i] to history
            else
                Append labels_imputed_test[i − 15] to history
            end if
        end if
    end for
    return labels_imputed_test_pred
end function
```

SARIMA

Let $AggregateLabelsImputed$, $AggregateLabelsImputedTrain$ and $AggregateLabelsImputedTest$ be the respective results of GETLABELSIMPUTED(Aggregate).

The SARIMA implementation was an instance of the statsmodels.tsa.statespace.sarimax.SARIMAX class [28]. Upon initialization, the class received the arguments $order$, $seasonal\_order$ and $trend$. $order$ and $seasonal\_order$ are tuples containing the arguments (p, d, q) and (P, D, Q, m) respectively. The parameters were chosen by performing a random search of 90 parameter combinations and choosing the parameter combination that got the lowest MSE in an expanding window process where the first half of $AggregateLabelsImputedTrain$ was training data, the second half of $AggregateLabelsImputedTrain$. The 90 combinations were taken from the following parameter grid:

- p: {0, 1, 2}
- d: {0, 1}
- q: {0, 1, 2}
- P: {0, 1, 2}
- D: {0, 1}
- Q: {0, 1, 2}
- m: {5}
- t: {"n","c","t","ct"}

After performing the grid search, the chosen parameters were:

- p: 2
- d: 0
- q: 2
- P: 1
- D: 0
- Q: 1
- m: 5
- t: "n"

Evaluation followed the same process as ETS evaluation using Algorithm 8. The forecasts for aggregate 0WDF SARIMA, aggregate 2WDF SARIMA and aggregate 15WDF SARIMA are STATISTICALFORECAST(Aggregate, 0WDF, SARIMA), STATISTICALFORECAST(Aggregate, 2WDF, SARIMA) and STATISTICALFORECAST(Aggregate, 15WDF, SARIMA) respectively.

4.2.3 Result

ETS scored best for all horizons. The biggest difference between naive forecasting and ETS is for the 0WDF forecasts where naive MSE is 8705 and ETS MSE is 4657. For this forecast, the ETS MAPE is 25% smaller than the naive MAPE. MSE and MAPE is shown in Tables 13, 14 and 15.

One weakness all forecasting methods display is that there are periods where they frequently forecast too low and periods where they frequently forecast too high. These periods are shared by all methods, which means that there is less potential to increase accuracy by aggregating

the forecasts.

4.3 Aggregate Forecast with Machine Learning

4.3.1 Question

Which machine learning method gives the forecast for the aggregate label with the lowest MSE for each specific forecast horizon?

4.3.2 Experiment setup

This section will first explain the general evaluation process by which all machine learning methods were evaluated. It will then explain specific implementation details for each method, such as parameter grids, Python libraries and ANN design.

General Evaluation Algorithm

The expanding window process that was followed when generating forecasts with each machine learning model is shown in Algorithm 9. For a horizon $horizon$, any machine learning model $model$'s aggregate forecast is the result returned by MACHINELEARNINGFORECAST(Aggregate, $horizon$, $model$). Forecasts were made one step ahead each time. The forecast horizon was simulated by using the features chosen for the horizon in Section 4.1.

Algorithm 9 Make a forecast for the specified cafeteria and horizon using the specified machine learning method

INPUT: A value $cafeteria$ that determines which cafeteria to make forecasts for, a value $horizon$ that determines the forecast horizon and a value $model$ that determines which model to use
OUTPUT: A forecast for the specified cafeteria with the specified horizon

function MACHINELEARNINGFORECAST($cafeteria$, $horizon$, $model$)
    $labels\_test\_pred \leftarrow$ an empty list
    if $cafeteria$ is Aggregate **then**
        $labels\_train \leftarrow AggregateLabelsTrain$
        $labels\_test \leftarrow AggregateLabelsTest$
    else:
        $labels\_train \leftarrow IndividualLabelsTrain$
        $labels\_test \leftarrow IndividualLabelsTest$
    end if
    $selected\_feature$ the set of features selected for the current $horizon$ in Table 9
    $train \leftarrow$ an empty two-dimensional array with rows and columns such that train[i] is the i'th row
    $test \leftarrow$ an empty array on the same form as $train$
    for each $label \in labels\_train$ **do**
        Append a row to $train$ that contains $label$ and the corresponding features for $label$
    end for
    for each $label \in labels\_test$ **do**
        Append a row to $test$ that contains $label$ and the corresponding features for $label$
    end for
    $current\_train \leftarrow$ a copy of $train$
    for each i = 0, 1, 2, ..., length($test$) - 1 **do**
        $current\_test \leftarrow$ a copy of $test[i]$
        PREPROCESSHISTORICALFEATURES($current\_train$, $current\_test$) 10
        PREPROCESSPARKINGFEATURES($current\_train$, $current\_test$) 11
        One-hot encode categorical features in $current\_train$
        One-hot encode categorical features in $current\_test$
        Scale $current\_train$ features by removing the feature mean and scaling to unit variance
        Perform the same transformation as the previous step on the features in $current\_test$
        FITMACHINELEARNINGMODEL($model$, $current\_train$) 12
        $prediction \leftarrow model$'s prediction when using the feature values in $current\_test$
        Append $prediction$ to $labels\_test\_pred$
        Append a copy of $test[i]$ to $current\_train$
    end for
    return $labels\_test\_pred$
end function

Algorithm 10 Impute missing values for historical features with the mean label in the training data

```
1: procedure PreprocessHistoricalFeatures(train (test has the same features))
2:     mean_label ← the mean value of the label in train
3:     for each historical feature feature in train and test) do
4:         Impute missing values for feature by inserting mean_label
5:     end for
6: end procedure
```

Algorithm 11 Detect and remove parking outliers, then impute missing values for parking features with the feature's training data mean

```
1: procedure PreprocessParkingFeatures(train, test)
2:     for each parking feature feature in train (test has the same features) do
3:         q1 ← the 25th percentile of values for feature in train
4:         q2 ← the 75th percentile of values for feature in train
5:         iqr ← q3 − q1
6:         lower_bound ← q1 − 1.5 × iqr
7:         upper_bound ← q3 + 1.5iqr
8:         for each value value for feature in train do
9:             if value < lower_bound or value > upper_bound then
10:                Remove value from train
11:            end if
12:        end for
13:        for each value value for feature in test do
14:            if value < lower_bound or value > upper_bound then
15:                Remove value from test
16:            end if
17:        end for
18:        feature_mean ← the mean value for feature in train
19:        for each value value for feature in train do
20:            if value is empty then
21:                Insert feature_mean in value's place
22:            end if
23:        end for
24:        for each value value for feature in test do
25:            if value is empty then
26:                Insert feature_mean in value's place
27:            end if
28:        end for
29:    end for
30: end procedure
```

Algorithm 12 Fit the machine learning model to the given 2D array of labels and features

procedure FITMACHINELEARNINGMODEL(*model*, *current_train*)
  if *model* is ANN **then**
    current_train_train ← the first 80% of *current_train*
    current_train_test ← the last 20% of *current_train*
    Fit *model* to *current_train_train* with *current_train_test* as validation data, as explained in detail in the ANN Section
    model ← the model from the epoch that scored best on *current_train_test*
  else
    param_grid ← the parameter grid that was specified for *model* in Listing 4.1
    For each parameter combination *params* from *param_grid*, calculate MSE in a 10-fold cross-validation process where *current_train* is the dataset, and *model* tuned with *params* and otherwise default values is the forecasting model
    Tune model with the parameter combination that gave the lowest MSE in the previous step and fit model to *current_train*
  end if
end procedure

All Models Except ANN

Table 11 shows which class each model was an implementation of. Parameter grids for machine learning models except ANN are shown in Listing 4.1. Default parameters were used except the parameters in the corresponding parameter grid for the model.

Model	Python class
LGBMRegressor	LGBMRegressor lightgbm.LGBMRegressor [29]
DecisionTreeRegressor	DecisionTreeRegressor sklearn.tree.DecisionTreeRegressor [30]
RandomForestRegressor	RandomForestRegressor sklearn.ensembleRandomForestRegressor [30]
SVR	SVR sklearn.svm.SVR [30]
XGBRegressor	XGBRegressor xgboost.XGBRegressor [31]

Table 11: Python classes for machine learning models.

```
# MODEL -> PARAMETER GRID
model_param_grid = dict()

model_param_grid["LGBMRegressor"] = {
        "max_depth": [5,10],
        "min_data_in_leaf": range(5, 30, 10),
    }
model_param_grid["DecisionTreeRegressor"] = {
        'max_depth': range(1, 20, 2),
        "min_samples_split": [x/100 for x in range(1, 40, 10)]
    }
model_param_grid["RandomForestRegressor"] = {
        "max_depth": [6,12],
        "n_estimators": [100]
    }
model_param_grid["SVR"] = {
        "C": [x*10000 for x in range(50, 90, 6)],
        "gamma": [x/10000000 for x in range(50, 90, 6)]
    }
model_param_grid["XGBRegressor"] = {
        'n_estimators': [100],
        'max_depth':[1,2,3,7]
    }
```

Listing 4.1: The Python dictionary containing parameter grids that were used for each model.

ANN

The ANN was implemented with Keras [32]. The ANN design used the keras classes keras.models.Sequential, keras.layers.Dense and keras.layers.Dropout and is shown in Listing 4.2. The ANN is specifically an MLP. MSE was used as the loss function. The Adam optimizer was used for training.

Training was done in 500 epochs. Early stopping was done with the tensorflow.keras.callbacks.EarlyStopping class with patience 100 and a minimum validation loss delta of 0.001. Data was shuffled when fitting.

```
model = Sequential()
model.add(Dense(80, input_dim=[...]))
model.add(Dropout(0.3))
model.add(Dense(80))
model.add(Dense(1))
```

Listing 4.2: ANN design.

4.3.3 Result

For 0WDF, ETS gets relatively good results, but is slightly outperformed by machine learning methods SVR and ANN. This is probably because good features such as parking features are available at that horizon. For all later horizons, ETS outperforms machine learning models.

4.4 Individual Cafeteria Forecasts

4.4.1 Question

Will the methods that get the lowest MSE for the individual cafeteria forecasts be different from the methods that get the lowest MSE for the aggregate forecasts?

4.4.2 Experiment setup

Naive

Table 10 shows which types of naive forecasts were made for each cafeteria and horizon. As was indicated by the PACF plots in Section 3.1.2, the labels for Eat the Street, Fresh 4 U and Soup & Sandwich show weekly seasonality like the aggregate labels, so the naive methods for these cafeterias are SN5 when the horizon allows it, otherwise SN15. Eat the Street had the shortest possible season lengths at 0WDF and 2WDF, which are seasonal naive with season length 1 (SN1) and seasonal naive with season length 3 (SN3) respectively. SN1 is the same as the non-seasonal naive forecast. For 15WDF it seems like strongest possible predictor for the Eat the Street label is the value 15 steps back, so SN15 was used at this horizon.

Forecasts were made for an imputed time series and the relevant forecast values were extracted, similarly to the Naive section for aggregate forecasts, but with the relevant cafeteria $cafeteria$ used to get the time series with GETLABELSIMPUTED($cafeteria$).

ETS

For each cafeteria $cafeteria$ in {Eat the Street, Fresh 4 U, Soup & Sandwich} and each horizon $horizon$, a forecast was made by running STATISTICALFORECAST($cafeteria$, $horizon$, ETS).

SARIMA

For each cafeteria $cafeteria$ in {Eat the Street, Fresh 4 U, Soup & Sandwich} and each horizon $horizon$, a forecast was made by running StatisticalForecast($cafeteria$, $horizon$, SARIMA).

Machine Learning

For each cafeteria $cafeteria$ in {Eat the Street, Fresh 4 U, Soup & Sandwich}, each horizon $horizon$ and each machine learning model $model$, a forecast was made by running MachineLearningForecast($cafeteria$, $horizon$, $model$).

4.4.3 Result

MSE and MAPE is for all individual forecasts is shown in Tables 13, 14 and 15.

For aggregate forecasts, SVR, ANN and ETS are best at 0WDF, ETS and LGBMRegressor at 2WDF and ETS at 15WDF.

For Eat the Street, SVR XGBRegressor and RandomForestRegressor tend to be the best of the machine learning methods, with ANN doing relatively well at 15WDF, but they are outperformed by ETS, especially at long horizons.

For Fresh 4 U, ETS, RandomForestRegressor and XGBRegressor (and SARIMA) perform best at all horizons, but ETS outperforms the machine learning methods.

For Soup & Sandwich, ETS is best at 0WDF and 2WDF, but XGBRegressor and DecisionTreeRegressor are slightly better at 15WDF. The DecisionTreeRegressor results are highly variable and tend to have high MSE otherwise, so DecisionTreeRegressor is still not one of the better methods.

The results are similar across cafeterias, but some methods tend to do better at different forecast horizons for one cafeteria, but not the others. ETS is also the best performing method for individual forecasts.

4.5 Ensemble Forecasts

4.5.1 Question

Will the MSE get lower if the average forecast for all models except naive and DecisionTreeRegressor are used?

4.5.2 Experiment setup

Forecasts that had already been generated for each horizon and cafeteria by ETS, SARIMA, LGBMRegressor, RandomForestRegressor, SVR and XGBRegressor were averaged and MSE and MAPE between test labels and the average were calculated.

4.5.3 Result

MSE and MAPE is for all cafeterias and horizons are shown Tables 13, 14 and 15. The only contexts where other models beat ensemble are 0WDF, 2WDF and 15WDF Soup & Sandwich and

2WDF Eat the Street. Among these contexts, ETS was best three times and XGBRegressor was best once. Ensemble was only slightly outperformed. Figure 23 shows the ensemble's forecast and percent errors for the aggregate 0WDF forecast. The ensemble still suffers from periods of forecasting too low and periods of forecasting too high. There graphs show an early outlier that none of the forecasting methods could handle. All forecasting methods had large errors in the period after test instance 200.

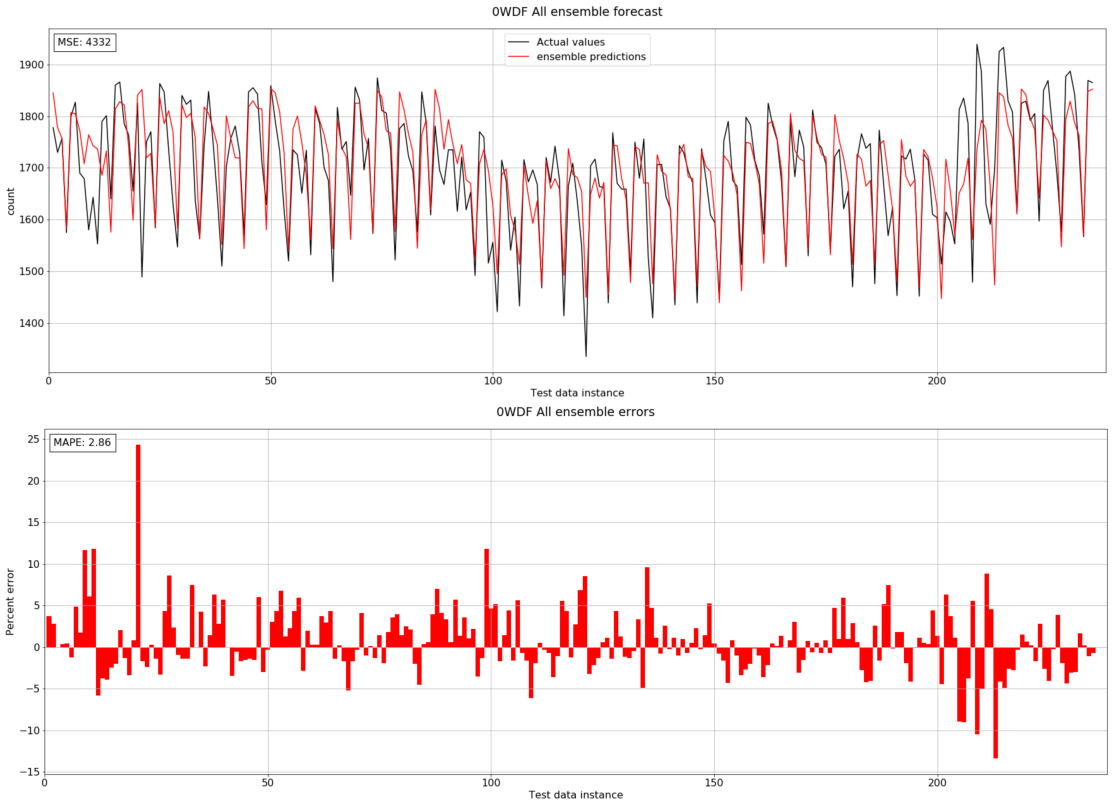


Figure 23: Ensemble forecast and errors for Aggregate 0WDF.

4.6 Weather Feature Test

4.6.1 Question

Will 0WDF machine learning methods perform better when using "average lunchtime temperature", "maximum lunchtime precipitation" and "average lunchtime humidity" in addition to the 0WDF features selected in Section 4.1?

4.6.2 Experiment setup

For each machine learning model *model*, a forecast was made by running MACHINELEARNINGFORECAST(Aggregate, 0WDF, *model*), with the only difference being that the features "av-

erage lunchtime temperature", "maximum lunchtime precipitation" and "average lunchtime humidity" were used in addition to the selected 0WDF features from Table 9.

4.6.3 Result

The MSE and MAPE for forecasts with and without weather features are shown for all machine learning methods in Table 12. Inclusion of weather features either gave the same result or slightly worse results indicating that weather features in their current form are not valuable features for the forecasting domain.

Model	Without weather features	With weather features
LGBMRegressor	5226, 3.17%	5294, 3.17%
DecisionTreeRegressor	7049, 3.84%	6999, 3.85%
RandomForestRegressor	5049, 3.15%	5137, 3.20%
SVR	4372, 2.76%	4346, 2.78%
XGBRegressor	5163, 3.18%	5179, 3.17%
ANN	4368, 2.83%	4728, 3.10%

Table 12: MSE, MAPE for 0WDF Aggregate machine learning methods, with and without weather features.

4.7 Discussion and Comparison

4.7.1 Relevant Features

The day of the week, the label one week before and the number of parked cars at 08:00 (ignoring 09:00 and 10:00 because of later availability) are the most relevant features. Feature selection per cafeteria was not performed. For individual cafeterias, the menu is likely to be the most important feature. Among the features that were available, there is still a possibility that the set of most relevant features is somewhat different between cafeterias. For instance, it seems like "count 1 day before" is more important for Eat the Street.

Weather was eliminated during feature selection and the choice was confirmed in Section 4.6. Weather in its current form is likely to add more noise and dimensionality to the machine learning methods than useful information.

Increasing the number of splits for the expanding window process in EvaluateFeatureSet could have led to more accurate results by increasing the average amount of training data available for each prediction. Splits were kept at 10 to reduce running time.

It is possible that different features are more useful for different machine learning methods, but feature sets were only selected for each forecast horizon in order to simplify the problem and under the assumption that a feature's inclusion being judged by several models' score was better than judging it for each model separately because it had less random variance than an individual score.

4.7.2 Best Method

MSE and MAPE for all methods is shown in Tables 13, 14 and 15. The ensemble is best for every cafeteria and horizon except Soup & Sandwich. This result is similar to Weiner et al. where a late fusion of the top five systems by taking their average beat all individual systems, but in Weiner et al. the ensemble beat individual systems with a large margin. When ignoring the ensemble, ETS, RandomForestRegressor, SVR and XGBRegressor perform well in many cafeteria, horizon combinations, with ETS being best in the most combinations.

Model	All	Eat the Street	Fresh 4 U	Soup & Sandwich
Naive	8705, 4.03%	4692, 9.11%	6167, 12.55%	4133, 9.34%
ETS	4657, 3.02%	3105, 7.56%	3616, 9.78%	**2762**, 7.73%
SARIMA	5448, 3.26%	3106, 7.54%	3757, 9.64%	3121, 8.07%
LGBMRegressor	5226, 3.17%	3706, 8.19%	3990, 10.58%	3453, 8.53%
DecisionTreeRegressor	7049, 3.84%	4060, 8.48%	4735, 11.11%	3767, 8.70%
RandomForestRegressor	5049, 3.15%	3377, 7.79%	3644, 9.92%	3268, 8.33%
SVR	4372, 2.76%	3135, 7.59%	4107, 10.36%	3031, 7.85%
XGBRegressor	5163, 3.18%	3127, 7.51%	3781, 10.26%	3069, 8.00%
ANN	4368, 2.83%	3713, 8.09%	4333, 10.65%	3179, 7.99%
Ensemble	**4332**, 2.86%	**3058**, 7.51%	**3405**, 9.62%	2849, 7.67%

Table 13: MSE, MAPE for 0WDF methods.

Machine learning methods except the ANN used 10-fold cross validation in the parameter grid search. 10-fold cross validation does not account for the ordering of data like expanding window cross validation does, so it could lead to inaccurate and optimistic results. It is also possible that 10-fold cross validation gives better results in the parameter search by making better use of the training data because a prediction is made once for each instance, as opposed to only the test instances in expanding window cross validation.

Hyperparameter grids not capturing any good parameter combinations could have made machine learning method performance worse. For instance, there were expanding window iterations where the chosen max_depth for LGBMRegressor was too large and yielded the warning "[LightGBM] [Warning] Accuracy may be bad since you didn't set num_leaves and 2^max_depth > num_leaves". Hyperparameter grids were selected by examining heatmaps of grid search scores when predicting $AggregateLabelsTrain$ with 0WDF features. It is not certain that these parameter grids will capture good parameter combinations for different horizons and cafeterias. For example, heatmaps for a wide range of SVR's C and gamma parameters for different horizon and cafeteria combinations are shown in Figure 24. The blue rectangle shows approximately where the chosen parameter grid for SVR is located. Figure 25 shows the heatmaps for

Model	All	Eat the Street	Fresh 4 U	Soup & Sandwich
Naive	8705, 4.03%	5885, 10.24%	6167, 12.55%	4133, 9.34%
ETS	5823, 3.50%	**3275**, 7.78%	3686, 9.88%	**2773**, 7.77%
SARIMA	6661, 3.69%	3308, 7.80%	3876, 9.73%	3041, 8.09%
LGBMRegressor	5978, 3.48%	4154, 8.71%	4299, 10.83%	3625, 8.83%
DecisionTreeRegressor	10530, 4.87%	4441, 9.07%	4588, 11.21%	3567, 8.50%
RandomForestRegressor	6490, 3.60%	3806, 8.21%	3948, 10.50%	3544, 8.51%
SVR	6455, 3.64%	3753, 8.43%	4171, 10.55%	3010, 7.87%
XGBRegressor	6617, 3.71%	3679, 8.04%	3729, 10.14%	3104, 7.96%
ANN	6773, 3.78%	4068, 8.48%	4134, 10.41%	3296, 8.28%
Ensemble	**5453**, 3.37%	3336, 7.88%	**3609**, 9.91%	2926, 7.80%

Table 14: MSE, MAPE for 2WDF methods.

the chosen parameter grid for SVR. From this example, it is clear that the SVR parameter grid is not well-adjusted for Fresh 4 U 2WDF in the training data. This is reflected in the testing data performance for SVR in the Fresh 4 U 2WDF context shown in Table 14.

Model	All	Eat the Street	Fresh 4 U	Soup & Sandwich
Naive	10482, 4.58%	5758, 10.20%	6830, 12.80%	5114, 10.18%
ETS	7942, 4.10%	4049, 8.56%	3991, 10.30%	3395, 8.26%
SARIMA	10270, 4.49%	4363, 8.88%	4445, 10.46%	3824, 8.66%
LGBMRegressor	8760, 4.33%	4476, 9.20%	4126, 10.55%	3559, 8.50%
DecisionTreeRegressor	11635, 5.10%	4562, 9.17%	4372, 10.72%	3127, 8.17%
RandomForestRegressor	8754, 4.26%	4074, 8.74%	4069, 10.53%	3198, 8.34%
SVR	9883, 4.55%	4259, 9.02%	4608, 11.00%	3238, 8.24%
XGBRegressor	8190, 4.05%	3962, 8.71%	4026, 10.20%	**2941**, 7.87%
ANN	10462, 4.86%	4133, 8.75%	4374, 10.67%	3285, 8.41%
Ensemble	**7508**, 3.91%	**3823**, 8.50%	**3797**, 9.97%	3088, 7.96%

Table 15: MSE, MAPE for 15WDF methods.

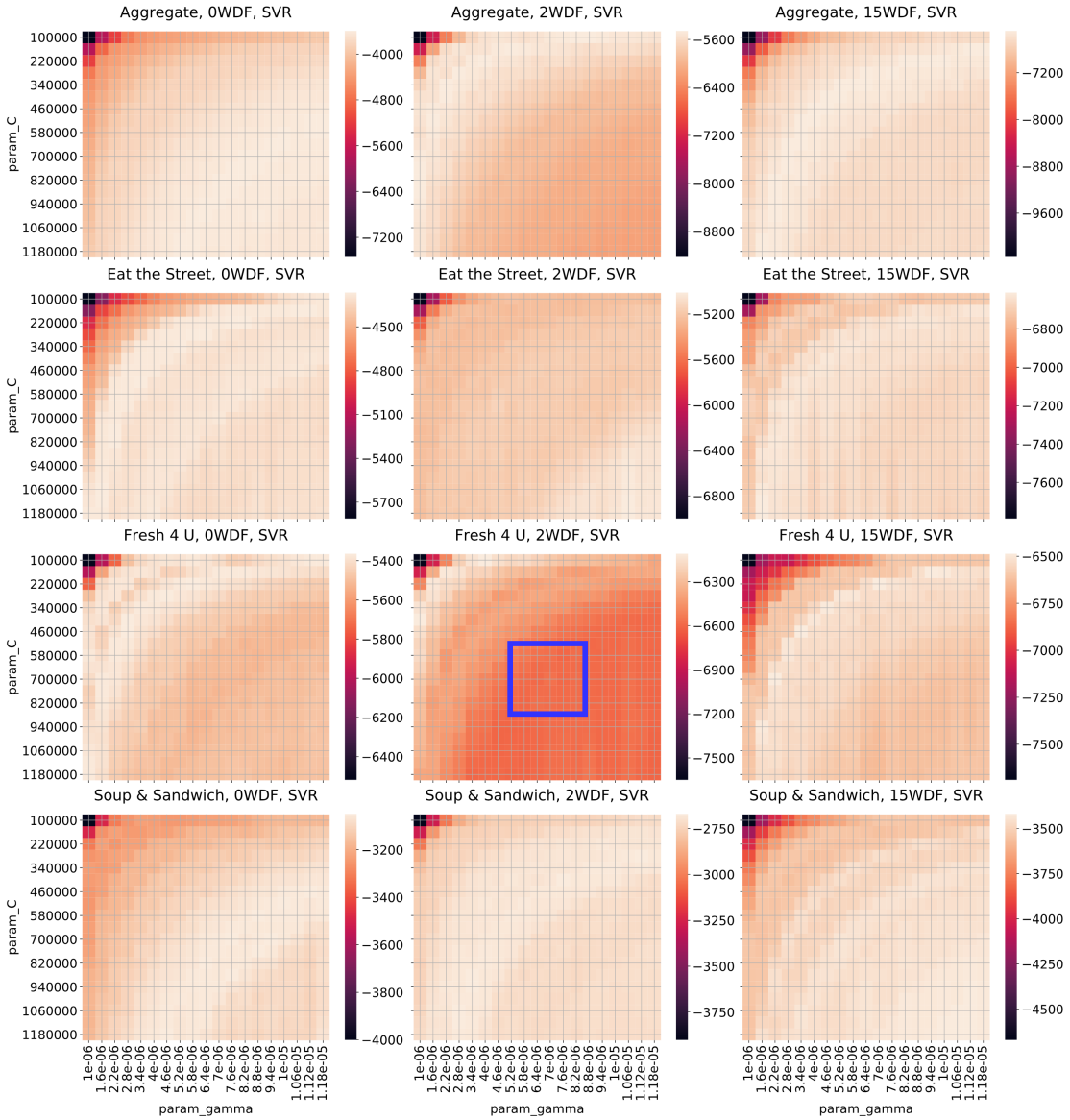


Figure 24: SVR parameter grid heatmaps for the training data of different horizon-cafeteria combinations, when using a large parameter grid.

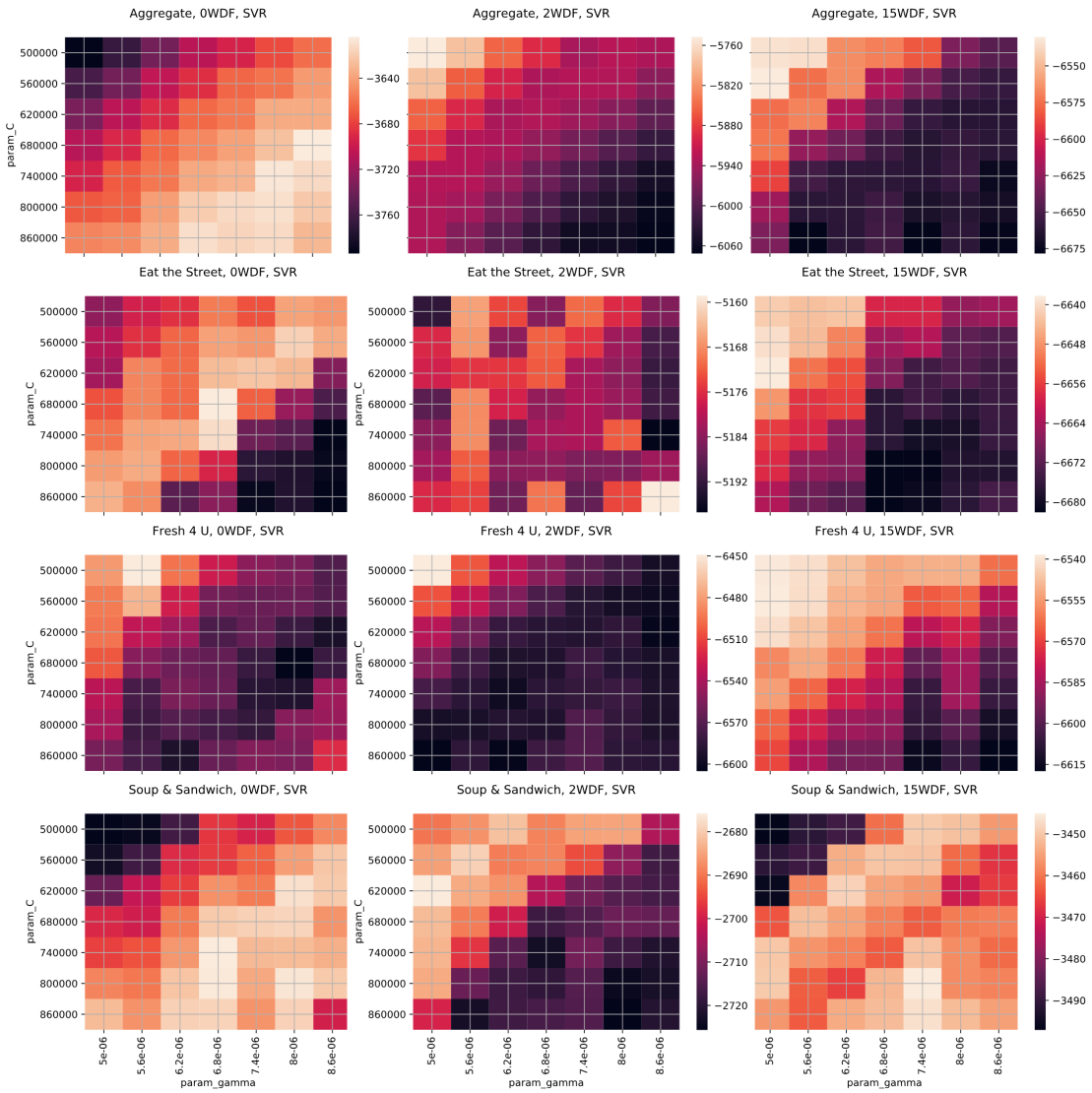


Figure 25: SVR parameter grid heatmaps for the training data of different horizon-cafeteria combinations, when using the selected parameter grid.

4.7.3 Confounding Variables

Extra training instances in machine learning 2WDF and 15WDF Models

To simulate forecasting ahead, machine learning models only use features available for their forecast horizon category, but 2WDF and 15WDF models use the same training sets as 0WDF models. In reality, the last two training instances would not be available for 2WDF models and the last 15 training instances would not be available for 15WDF models. It is unlikely that including these instances has a large enough effect to invalidate the comparison between machine learning models and statistical models in the 2WDF and 15WDF categories.

Training data contamination

Using testing data for feature selection is a form of training data contamination. All graphs in Chapter 3.1.1 in addition to Figure 13 include last half of the dataset which was used as testing data. These graphs were used to show completeness and correctness of the dataset and had only a small impact on feature selection.

Different definitions of outliers for parking features were tried for all instances in $SelectedInstances$. It was observed that the definition of outliers based on interquartile range worked well. A different conclusion could have been made if $SelectedInstancesTrain$ were used.

Chapter 5

Conclusion and Future Work

5.1 Research Questions/Conclusion

Research question 1: What features are useful when forecasting transactions in workplace cafeterias?

Ignoring features that were not available for this thesis, when forecasting transactions, it is most useful to use features capturing the weekly seasonality like "day of the week" and "count 5 weekdays before", semester factors that capture the seasonality repeating every semester, "week of the month" and parking features.

Research question 2: Which machine learning methods are best at forecasting the number of transactions in cafeterias?

For different cafeterias and horizons, different methods perform best with the selected features and parameter grids in this thesis. SVR, RandomForestRegressor, XGBRegressor perform best most of the time. The only large difference between implemented methods is that DecisionTreeRegressor and ANN have varying results that are often poor.

Research question 3: Do machine learning methods forecast cafeteria transactions better than statistical models?

Based on experiments with the selected features and parameter grids, ETS seems to outperform all implemented statistical methods when using features available for this thesis.

5.2 Future Work

5.2.1 Features

There was a forecasting approach mentioned in [13] that categorized menus into groups based on textual similarity and used the menu group as a feature together with time and weather features as input for a gradient boosting model. This approach only got the fifth best score. The office cafeterias might be different from the university cafeteria in the way that almost all

employees will go to at least one of the cafeterias and the choice of which cafeteria is most affected by menu and proximity. According to the accounts of several employees, the menu is the most important factor in deciding which cafeteria to choose. For this reason, the individual machine learning forecasts could probably achieve higher accuracy with menu features.

Since only the first 50% of the data was used for feature selection, features with yearly cycles were ignored, such as the historical feature "previous year, same day of the week, closest day of the year" and the date features "day of the year", "ISO week number" and "month" and date-features that say whether or not it is a holiday or how long into a certain holiday it is. Yearly date features were used by [15] in predicting restaurant visitors and [13] in predicting university cafeteria transactions. Yearly historical features were used by [16] in predicting daily sales of perishable items in a store. The first paper used yearly features despite only having data spanning over a little more than one year. The second paper had data for eight full years and the last had four and a half years of data. With more data, yearly features could probably increase machine learning accuracy.

5.2.2 Methods

In this thesis, parameter grids were selected for each machine learning model by only looking at 0WDF aggregate, but as was shown in 4.7.2, different horizons and different cafeterias require different parameters. In a future system, it would be best to select parameter grids for each combination of cafeteria and horizon.

The parameters for ETS and SARIMA were also chosen for all applications of the given model by only looking at 0WDF aggregate performance. A separate parameter search for each cafeteria is likely to give better results because different cafeterias' labels show different time series properties, such as the lack of 5-day seasonality in Eat the Street.

It would be interesting to try more MLP designs for this forecasting problem. ANNs were the best-performing methods in several similar forecasting papers and an ANN design similar to one of the submissions in Weiner et al. [13] was not tried in this thesis.

5.2.3 Different Labels

When ordering food that will be consumed over several days, it might not be as important to know how many people will come per day, but rather the average over several days.

When preparing food, it might be useful to know the distribution of the transactions throughout the day such that the enough is prepared without being unused for long periods of time which would that it could become dry or cold or spoil faster. Therefore a forecast for shorter periods of time such as the 5-minute periods in [13] could be a useful addition.

Bibliography

[1] Prechelt, L. 2000. Early stopping - but when?

[2] Silvennoinen, K., Heikkilä, L., Katajajuuri, J. M., & Reinikainen, A. 2015. Food waste volume and origin: Case studies in the finnish food service. *Waste Management, Vol 46, Pages 140-145*.

[3] Elander, M. 2016. Matavfall i sverige.

[4] Stensgård, A. E., Prestrud, K., Hanssen, O. J., & Callewaert, P. 2018. Food waste in norway.

[5] Gustavsson, J., Cederberg, C., Sonesson, U., Van Otterdijk, R., & Meybeck, A. 2011. Global food losses and food waste.

[6] Stenmarck, et al. 2016. Estimates of european food waste levels.

[7] Ansel, D. & Dyer, C. 1999. A framework for restaurant information technology. *Cornell Hotel and Restaurant Administration Quarterly, Vol 40, no. 3*, page 79.

[8] Scandic kuttet matsvinn med over 85 tonn i 2017. http://www.mynewsdesk.com/no/scandic-hotels-norge/pressreleases/scandic-kuttet-matsvinn-med-over-85-tonn-i-2017-2395216. Accessed: 21 October 2018.

[9] Östergren, K. et al. 2014. Fusions definitional framework for food waste.

[10] Maynard, N. 2019. Ai in retail: Segment analysis, vendor positioning market forecasts 2019-2023.

[11] Ryu, K. & Sanchez, A. 2003. The evaluation of forecasting methods at an institutional foodservice dining facility.

[12] Liu, X. & Sun, D. 2016. A study on the impact of online takeout based on the baidu index on the management of university cafeteria.

[13] Weiner, J. e. a. 2017. Bremen big data challenge 2017: Predicting university cafeteria load.

[14] Xinliang, L. & Dandan, S. 2017. University restaurant sales forecast based on bp neural network – in shanghai jiao tong university case.

[15] Ma, X., Tian, Y., Luo, C., & Zhang, Y. 2018. Predicting future visitors of restaurants using big data.

[16] Arunraj, N. S., Ahrens, D., Fernandes, M., & Müller, M. 2014. Time series sales forecasting to reduce food waste in retail industry.

[17] Liu, Y., Wei, W., Wang, K., Liao, Z., & Gao, J.-j. 2011. Balanced-sampling-based heterogeneous svr ensemble for business demand forecasting.

[18] Aburto, L. & Weber, R. 2007. A sequential hybrid forecasting system for demand prediction.

[19] Gurnani, M. e. a. 2017. Forecasting of sales by using fusion of machine learning techniques.

[20] Loureiro, A., Miguéis, V., & da Silva, L. F. 2018. Exploring the use of deep neural networks for sales forecasting in fashion retail.

[21] Abrishami, S., Kumar, P., & Nienaber, W. 2017. Smart stores: A scalable foot traffic collection and prediction system.

[22] Yang, Y., Pan, B., & Song, H. 2014. Predicting hotel demand using destination marketing organization's web traffic data.

[23] Cho, V. 2002. A comparison of three different approaches to tourist arrival forecasting.

[24] Efron, B. 1979. Bootstrap methods: another look at the jackknife.

[25] Hyndman, R. J. & Athanasopoulos, G. 2014. Forecasting: principles and practice.

[26] Domingos, P. 2012. A few useful things to know about machine learning.

[27] Yr, delivered by the norwegian meteorological institute and nrk. https://www.yr.no/place/Norway%2FOslo%2FOslo%2FBygd%C3%B8y/almanakk.html?dato=2016-11-30. Accessed: 29 May 2019.

[28] Seabold, S. & Perktold, J. 2010. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.

[29] Ke, G. et al. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 3149–3157.

[30] Pedregosa, F. et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[31] Chen, T. & Guestrin, C. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 785–794, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2939672.2939785, doi:10.1145/2939672.2939785.

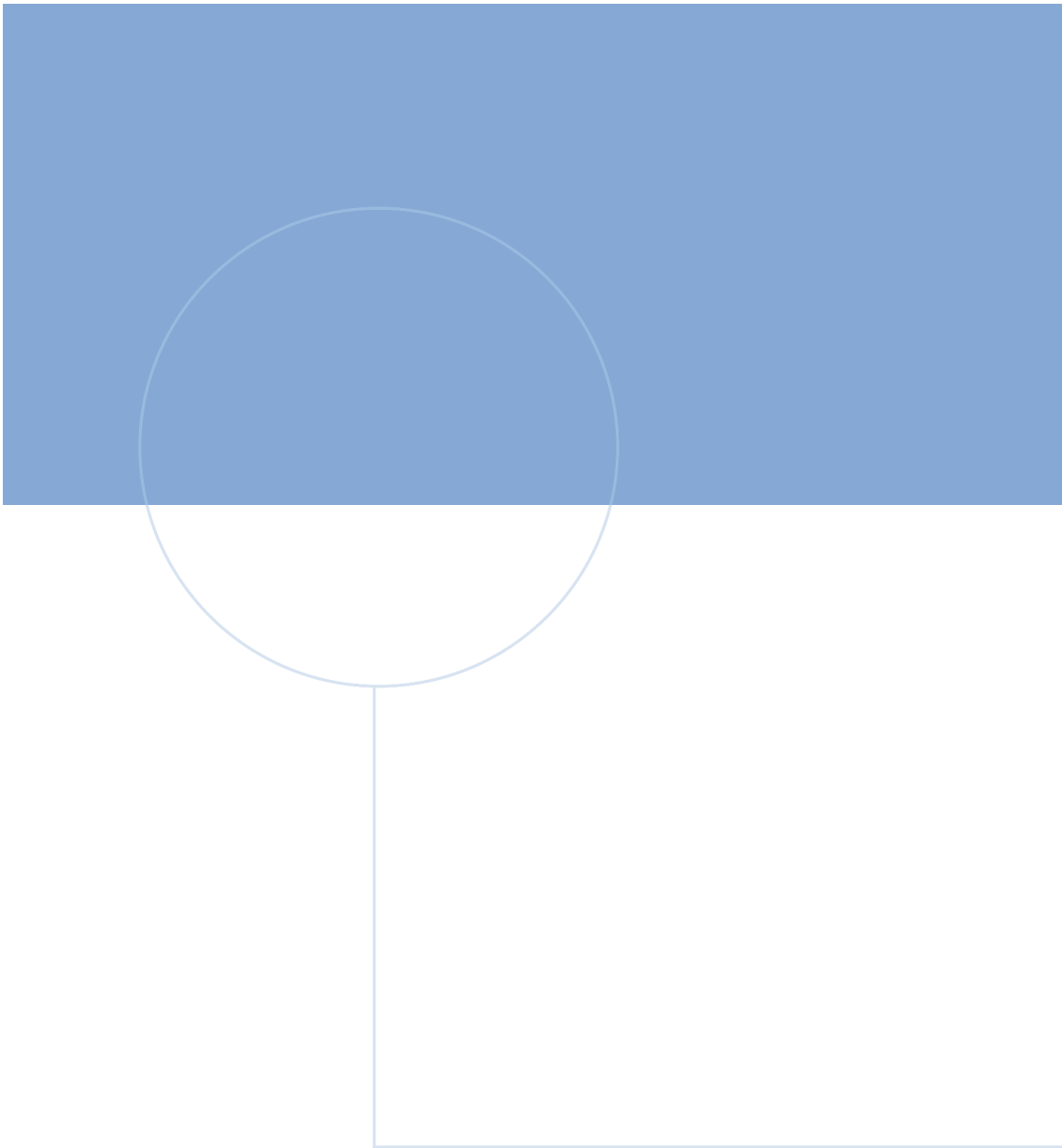
[32] Chollet, F. et al. 2015. Keras. https://keras.io.

Appendix A

Code Listings

```
def get_outlier_dates(dates, counts, iqr_threshold):
    """
    Takes arrays of dates and counts in corresponding order and
    returns a set of dates on which the count is an outlier.
    The count is an outlier if it is more than 1.5 the IQR away
    from the mean of the detrended array of counts
    """

    counts_detrended = np.full((len(counts),), np.nan)
    for i in range(len(counts)):
        window_min_i = max(i - 30, 0)
        window_max_i = min(i + 30, len(counts) - 1)
        window = counts[window_min_i : window_max_i]
        counts_detrended[i] = counts[i] - window.mean()

    q1, q3 = np.percentile(counts_detrended, [25, 75])
    iqr = q3 - q1
    outlier_lower_bound = q1 - (iqr_threshold*iqr)
    outlier_upper_bound = q3 + (iqr_threshold*iqr)

    outlier_dates = set()
    for i in range(len(counts_detrended)):
        if counts_detrended[i] < outlier_lower_bound or \
                counts_detrended[i] > outlier_upper_bound:
            outlier_dates.add(dates[i])

    return outlier_dates
```

Listing A.1: Outlier function.



NTNU
Norwegian University of
Science and Technology