Utrecht University

Predicting traveler demand using explainable models and feature engineering

by

Colino Sprockel

Submitted to the Artificial Intelligence Graduate Program

in partial fulfillment of the requirements for the degree of

Master of Science

Graduate Program in Artificial Intelligence

Utrecht University

2023

Predicting traveler demand using explainable models and feature engineering

APPROVED BY:

dr. Heysem Kaya	....
(Thesis Supervisor)	
dr. Thijs van Ommen	....

DATE OF APPROVAL: DD.MM.YYYY

ACKNOWLEDGEMENTS

I want to thank Heysem Kaya, not only for the support and insight he provided me during writing this thesis. But also for his patience and generous investment of his time. I want to thank Tjebbe Hepkema for his critical but always helpful feedback, his support in preparing my presentations, and his general helpfulness if I was unsure about something either academically or personally. I want to thank Marie Koorneef for her feedback and for providing a more practical view, which was a nice contrast to the academically inclined Heysem and Tjebbe. Lastly, I want to thank the whole of Team Sigma, the team I was a part of during the writing of this thesis for their hospitality, encouragement, and friendship.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

n	Number of distinct data points in the dataset
p	Number of variables available for use in making predictions
$\mathcal{D}$	Dataset containing n distinct data points
$\mathbf{X}$	Matrix representing the independent variables
$\mathbf{x}_i$	Independent variables of a single datapoint
$\mathbf{y}$	Vector representing the dependent variables or targets
y_i	Target or dependent variable for the i-th datapoint
$\hat{y}_i$	Model's approximation or estimation of y_i
$\hat{\mathbf{y}}$	Model's approximation or estimation of $\mathbf{y}$

LIST OF ACRONYMS/ABBREVIATIONS

ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
AI	Artificial Intelligence
β-VAE	Beta Variational Autoencoder
CART	Classification And Regression Tree
CNN	Convolutional Neural Network
CoST	Contrastive Learning of Disentangled Seasonal-Trend Representations for Time Series Forecasting
DNN	Deep Neural Network
EBM	Explainable Boosting Machine
FS	Forward Selection
GBDT	Gradient Boosting Decision Tree
GLM	Generalized Linear Model
Grad-CAM	Gradient-weighted Class Activation Mapping
ID3	Iterative Dichotomiser 3
LGBM	Light Gradient-Boosting Machine
LIME	Local Interpretable Model-Agnostic Explanations
LRP	Layer-wise Relevance Propagation
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ReLU	Rectified Linear Unit
RFE	Recursive Feature Elimination
RNN	Recurrent Neural Network

SCS	System Causability Scale
SFS	Sequential Floating Search
SGD	Stochastic Gradient Descent
SHAP	SHapley Additive exPlanation
SUS	System Usability Scale
VAE	Variational Autoencoder
XAI	eXplainable Artificial Intelligence

1. INTRODUCTION

1.1. Problem Statement

The Nederlandse Spoorwegen (NS - Dutch Railway company) is a vital part of Dutch infrastructure that provides public transport through trains, transporting an average of 1 million travelers a day. The population density and small size of the country offer unique challenges to the planning of a quality timetable [1]. The NS aims to provide quality transportation and seating for every traveler. Unfortunately, this is not possible at all times and sometimes trains are overcrowded, leaving travelers unhappy. To help provide information to travelers, NS predicts short-term train crowdedness. Now, when travelers plan their journey in the NS app they are informed of the crowdedness of the route they intend to take. This is done by showing a drawing of either 1 person (low occupancy, there should be plenty of seats left), 2 people (medium occupancy, there should still be some seats left), or 3 people (very crowded, there are likely no seats left).

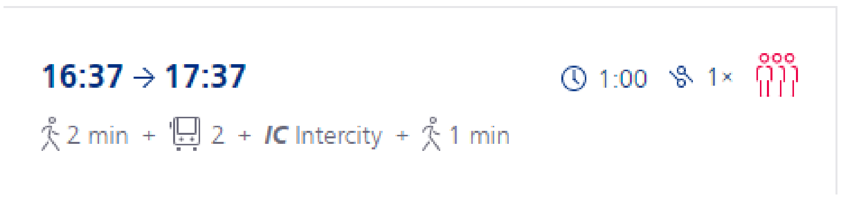


Figure 1.1: Example of a crowded journey, as indicated by the red pictogram of three people

This information allows travelers to adjust their expectations, leading to higher satisfaction. Alternatively, the traveler can use this information to pick a less crowded journey by either changing their departure time or taking a different route. An added benefit of the latter is that it improves traveler distribution by shifting travelers from crowded trains to less crowded trains. One of the challenges NS faces is coping with rush hour demands in the morning and evening. Since material and resource demands are determined mostly by peak travel demand, increasing transportation capacity during rush hour through the deployment of extra rolling stock and personnel is very costly. This shows the potential of methods that can motivate travelers to travel outside of rush hours as a cost-effective method to improve travelers' experiences.

Unfortunately, it is difficult to adjust the train schedule and the material used based on the short-term crowdedness predictions, since changing the deployment of rolling stock is a complex puzzle. Other, longer-term forecasts are used for that purpose. There are some exceptions, such as the route to the train station of the popular beach town Scheveningen, where the NS is able to use short-term crowdedness predictions to adjust rolling stock rapidly. On this route in particular, the NS struggled to correctly estimate transportation demand in advance, leading to incredibly overcrowded trains sometimes and mostly empty trains at other times. They have now implemented a dedicated model that incorporates weather forecasts to produce high-quality forecasts.

The short-term crowdedness prediction currently in use by the NS consists of two steps. The first one involves forecasting the number of passengers who will travel from a given origin station (O) to a given destination station (D) every half hour. To accomplish this, the NS utilizes a probabilistic time series forecasting DeepAR model [2] for busy origin-destination (OD) pairs and a Savitsky-Golay smoothing algorithm [3] using data from the previous week's passenger counts for OD pairs with lower volumes. The prediction takes place at night and predicts the current day, the next day, and the day after. The input features are only the number of passengers in the past, and a binary feature indicating the day is a national holiday. The second step is to predict which trains these passengers will choose. A RAPTOR algorithm [4] is used to determine the options going from O to D at every minute of the day. Then the passengers are distributed over the journey options according to a predetermined formula that takes into consideration factors such as the type of train, travel time, and the number of transfers.

Predicting train crowdedness remains a challenging problem and wrong predictions can heavily impact customer satisfaction. We aim to improve these predictions using explainable AI methods to understand causal relations between the inputs and outputs, accelerate feature engineering and diagnose problems in outlier forecasting errors.

1.2. Objectives

The main goal is to engineer a method that can achieve performance measures close to state-of-the-art models such as DeepAR at the OD-level, with the added benefit that the resulting model is interpretable, meaning that data scientists can understand the effect of individual features on the model's predictions.

The first approach is to engineer a set of intelligible features such that a glassbox model such as linear regression or explainable boosting machine [5] can achieve adequate performance whilst maintaining an acceptable level of interpretability by keeping the number of features limited and intelligible. Feature engineering will first be done by handcrafting features through expert knowledge, iteratively adding removing, and adjusting features based on the insights we receive when auditing glassbox models trained on these features. Later, we will create additional features using existing dimensionality reduction tools such as singular value decomposition and variational autoencoders.

Finally, we examine the use of black-box models such as Light Gradient-Boosting Machine (LGBM) paired with SHAP for explainability [6]. The inputs for these black-box models will be the features we designed through feature engineering and dimensionality reduction as mentioned above.

The main reason for requiring interpretability or explainability would be for auditing outlier predictions and iteratively improving model performance through feature engineering and/or model modifications. Finally, we want to analyze various eXplainable Artificial Intelligence (XAI) methods in the time series domain and report our experiences, the benefits, and the drawbacks of these methods.

2. Background on Methods and Literature

Achieving interpretability can be done through a variety of different paths: one can decide to restrict model selection to models that are interpretable by default, such as shallow decision trees, explainable boosting machines (EBM), and sparse multiple linear regression. Unfortunately, the same properties that make the models interpretable also limit their expressivity and some performance is sacrificed when the relationship between input variables and targets is more complicated. If this limitation is severe enough, one may opt to use black box models such as (deep) neural networks and LGBM, followed by a secondary method that is able to provide explanations about the model's decisions.

The methods that can provide explanations are divided into two groups, model-agnostic and model-specific techniques. Model agnostic approaches treat the model as a black box and provide explanations purely by observing and sometimes modifying the input and output of the model. Therefore, they can be applied to any machine learning model. Examples of model-agnostic techniques are partial dependence plots, SHAP and LIME [6–8]. Model-specific techniques do utilize internal knowledge of the model in providing explanations, these methods are therefore not applicable to all types of models but restricted to certain kinds. An example that is only applicable to convolutional neural networks is Grad-CAM, which highlights areas of importance in the classification of images [9].

In this thesis, n represents the number of distinct data points in the dataset, and p denotes the number of variables that are available for use in making predictions. Such that $\mathcal{D} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathbb{R}, 1 \leq i \leq n\}$.

The matrix $\mathbf{X}$ represents the independent variables, where each row of the matrix corresponds to a single datapoint, and each column corresponds to a particular variable. The targets, or dependent variables, are represented by a vector $\mathbf{y} \in \mathbb{R}^n$, where each element of the vector corresponds to the target for the corresponding row of the matrix $\mathbf{X}$. In other words, the ith element of $\mathbf{y}$ is the target for the ith datapoint: y_i, which is represented by the ith row of $\mathbf{X}$. When we speak of a single sample, we use the notation of $\mathbf{x}_i$ which represents the independent variables of that sample where $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \cdots, x_{i,p})$, and y_i to represent the target or ground truth belonging to that sample. The symbols $\hat{y}_i$ and $\hat{\mathbf{y}}$, are used to denote a model's approximation or estimation of y_i and $\mathbf{y}$, respectively.

In this thesis, we shall sometimes use a matrix $\mathbf{Y} \in \mathbb{R}^{D \times T}$ to denote a univariate time series, where each row corresponds to a single day and each column represents a specific time point within that day. Therefore, $\mathbf{Y}$ can be written as:

$$\mathbf{Y} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,T} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,T} \\ \vdots & \vdots & \ddots & \vdots \\ y_{D,1} & y_{D,2} & \cdots & y_{D,T} \end{bmatrix}. \tag{2.1}$$

The value $y_{d,t}$ refers to the target value at the t-th time point of day d. Here, D represents the total number of days and T denotes the total number of time points within a day.

This notation is used throughout the thesis to explain and analyze our methods, deviations are explicitly mentioned.

2.1. Explainable and Interpretable Machine Learning

It is commonly thought that there exists a tradeof between model interpretability and model performance, where one has to sacrifice one to gain more of the other. Recently, more researchers are starting to fight against that notion, such as Rudin *et al*, who state that *" it is important not to assume that one needs to make a sacrifice in accuracy in order to gain interpretability. In fact, interpretability often begets accuracy, and not the reverse. Interpretability versus accuracy is, in general, a false dichotomy in machine learning."* [10]

In fact, Rudin *et al* argue that it is possible to achieve both high interpretability and high performance in machine learning models. They argue that interpretability can often improve model performance, because it allows practitioners to better understand the model and how it is making decisions. This can lead to better data preprocessing and transformation, better model selection, hyperparameter tuning, and feature engineering, which can all contribute to improved model performance. Additionally, the authors argue that interpretability is important for the practical deployment of machine learning models. For example, in many real-world applications, the underlying distribution of the data changes over time, this so-called domain shift can have far-reaching consequences for the real-world performance of the model [10]. In the case of black-box models, problems arise because of the difficulty troubleshooting the model, which can cause long periods of an inaccurate model in use, or in some cases the necessity to take the model offline until troubleshooting is completed and a fix is found.

It is difficult to give a full definition of interpretability that covers all possible uses. Doshi-Velez and Kim define interpretability as *"the ability to explain or to present in understandable terms to a human"* [11]. In general, humans are only able to hold a limited amount of information in working memory [12], therefore explanations that are understandable need to be short and of limited complexity [13].

The level of interpretability of a system does not only depend on intrinsic factors, such as the number of variables involved and the complexity of the operations performed to produce an output. It also depends on the specific human involved and the circumstances in which an explanation is to be used: it is *context dependent.* If the human is a machine learning expert with domain knowledge, more complex explanations can be considered to be interpretable [14]. The circumstance in which the human is provided the information also matters: for example in the case of a time constraint, or cognitive overhead from having to pay attention to multiple tasks simultaneously [15]. In those cases, the explanation should be made simpler to achieve the desired level of interpretability.

Since what is considered understandable to a human depends on many factors intrinsic to the system, and also on the kind of human that uses these explanations, the definition above should not be seen as a yes or no question. Instead, it should be seen as a spectrum on which AI systems can be placed, ranging from wholly opaque neural networks with billions of parameters on one side, and, for example, simple tree or rule-based models on the other side [16].

To assess model interpretability, Doshi-Velez and Kim [11] distinguish between three levels of evaluation. The highest level is the *application-grounded* level where interpretablity is examined by its effects on actual outcomes. This way a system is directly tested on the task it was built for. An example is a homework-hint system that was evaluated on the post-test scores of its users [17]. Unfortunately, experiments at this level are often difficult, expensive and time-consuming to perform. An easier alternative is *human-grounded* evaluation, where humans perform simplified tasks to evaluate model interpretability. A common approach at this level is model simulation: where humans are presented with input and the model or an explanation and are tasked with simulating the model's prediction [18]. Common metrics are accuracy, response time, and subjective difficulty [13, 18]. The simplest level, *functionally-grounded*, requires no human experiments. Instead, a formal definition of interpretability is defined and used to measure explanation quality. These quantitative measures are usually objective and can be based on observations from application-grounded and human-grounded evaluations, such as the depth of a decision tree or the maximum number of leaf nodes. Thus we indirectly rely on research performed at the *application-grounded* and *human-grounded* levels to decide if we can consider a model interpretable.

In the case of black-box models coupled with an explainability method like SHAP or LIME, we are interested in the *fidelity* of the explanation, by which we mean how closely the explanation mimics the original model [19]. If the fidelity of an explanation is low, it means that the explanation is doing a poor job of explaining what is happening inside the model. One must be aware that, even with methods that perfectly explain why a model predicts a certain outcome there is still the possibility of a Rashomon effect: where separate models can rely on completely different covariate information and reach a similar prediction [10].

In our case, we aim to use explanations to audit and improve model performance, therefore explanations need only be understandable by domain experts and not necessarily by travelers. For the remainder of this thesis, we will treat a system that is capable of providing explanations of its predictions that are understandable to domain experts as explainable or interpretable systems.

2.2. Introduction to Time Series

A time series is a sequence of numerical values over time. Time series forecasting is the challenge of predicting future values in the series based on its past. Forecasting is generally categorized into one-step forecasting and multi-step forecasting. One-step forecasting means only predicting one future value in the series, this is a markedly simpler challenge than multi-step forecasting since the system can heavily rely on the most recent known values. Multi-step forecasters use either autoregression or multi-output forecasting.

Autoregressive forecasting is an approach where the underlying model is only capable of one-step ahead forecasting and the previous predictions are fed back into the model as new data points to predict further into the future. Given that the model relies on its own predictions to make further predictions there is a big risk of propagating errors of increasing size the further one tries to predict the time series [20]. Furthermore, there is the property that the underlying model treats the recursively generated inputs as true values since it was trained in one-step ahead forecasting on a training set with known values. The most well-known method is Autoregressive Moving Average (ARMA), where Autoregressive and Moving Average models are combined. Autoregressive Integrated Moving Average (ARIMA) is an expansion of this method, where differencing is applied to the time series to make it more stationary [21]. These and other related models have been the standard approach for some time.

An alternative is multi-output forecasting, where one model predicts multiple time points at once [20]. The advantage is that one is guaranteed that the model only uses historic information. The downside is that this generally leads to more complicated models, for example in the case of linear regression, this would require the use of complicated features to capture the (often nonlinear) relationships between historical data, the time of day, and the target variable.

A common method to mitigate this issue in multi-output forecasting is to train one model for each time point, meaning that each model is fitted to target variables of *one* time slot, which saves the model from modeling those challenging nonlinear relationships. If you want to predict one full day ahead using multi-output forecasting, you would need a number of models equal to the number of time points in one day. In the case of NS data, this would result in 48 models if we want to forecast one day into the future (1 model per half-hour time slot), if we want to predict a full two days into the future this number would jump to 96 models.

Recently, we are seeing a shift toward deep learning, and hybrid systems, which combine domain-specific models with deep learning components [20]. This shift towards deep learning has resulted in an increase in the opaqueness of the models, and an increase in the computational cost of training and running these models.

2.3. Intrinsically Interpretable Models

2.3.1. Decision Trees

Decision trees are a supervised learning method that can be applied to both regression and classification tasks. These models work by following a series of decision rules based on feature values, starting from the root node and proceeding down the tree until a leaf node is reached. In the case of a classification problem, the leaf node contains a class. In the case of a regression problem, the leaf node contains an estimated numerical value. The branching factor of a decision tree represents the number of branches at each node. The simplest case is a binary decision that branches into two paths at each node. Each branching point on a binary decision tree can be represented mathematically in the following form:

$$\text{if } x_j \leq s \text{ then } \hat{y} = c_1, \text{ else } \hat{y} = c_2, \tag{2.2}$$

where x_j is the variable of interest and s denotes the threshold value. c_1 and c_2 are the subsequent paths the algorithm takes depending on the decision rule. If c_i is a leaf node, it returns a value. If it is not a leaf node, it will forward to the next decision rule. The result of these decision rules is the partitioning of the input space into a series of regions, where each region corresponds to a particular predicted value.

Decision trees can be constructed though a variety of algorithms, such as ID3 and CART [22,23]. These algorithms typically select decision rules by maximizing some criterion such as entropy or the increase in the Gini index, which can be seen as measures of node purity.

Decision trees are popular because they are easy to understand and interpret. The tree structure of a decision tree allows for a clear and visual representation of the decisions and splits made by the model, which can be helpful for explaining the model's predictions to others. However, the interpretability and risk of overfitting can be impacted by several factors such as tree depth and the number of features considered by the model. As the depth of the tree increases, the model becomes more complex and harder to fully grasp on a global level. The interpretability of a decision tree can be improved by limiting the depth of the tree, limiting the number of features used, and carefully selecting the thresholds for making decisions at each node. It can also be helpful to use techniques such as pruning or feature selection to simplify the tree.

Unfortunately, classical decision trees such as the ones discussed above, are not very expressive and struggle to model complex relationships. Methods that expand the expressivity of the model, such as bagging and boosting, also make it less interpretable.

2.3.2. Multivariate Linear Regression

Multiple linear regression is a popular method, where the relationship between a dependent variable $\mathbf{y}$ and p independent variables or features in $\mathbf{X}$ is modeled linearly:

$$\hat{y}_i = f_\Theta(\mathbf{x}_i) = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} \cdots \theta_p x_{i,p}, \tag{2.3}$$

where θ_0 is the bias, and θ_j with $j \in \{1, 2, ..., p\}$ are the learned weights. Multiple linear regression has some mathematical properties of interest. The first assumption is linearity: it assumes the relationships between feature variables and the target variable is linear. This means that the change in the dependent variable $\hat{y}_i$ is proportional to the changes in the independent variables. For example, if the independent variable $x_{i,j}$ increases by one unit, the dependent variable is expected to increase by a constant amount θ_j. Note that linearity implies that the constant amount θ_j is independent of the starting value of the independent variable $x_{i,j}$. If the relationship between variables is nonlinear, the predictions made by the model may not be accurate. The second assumption is homoscedasticity which means that the model error should be relatively constant along the values of the dependent variables. Furthermore, we have the assumption of the absence of multicollinearity. Which requires that features are not strongly correlated. Having multicollinearity does not negatively impact model performance. However, it does heavily impact model interpretability as we have no way of knowing which of a couple of highly correlated variables actually *causes* the change in the target variable.

The bias θ_0 and the weights θ_j are found by minimizing the residual sum of squares (RSS)

$$\texttt{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = ||\mathbf{y} - \hat{\mathbf{y}}||_2^2, \tag{2.4}$$

where y_i is the ground truth of variable y in the ith sample, $\hat{y}_i$ is the predicted value for variable y in the ith sample, and $||\cdot||_2$ the Euclidean norm. This means the cost function can be written as

$$L(\Theta) = \sum_{i=1}^{n} \left(y_i - \left(\theta_0 + \sum_{j=1}^{p} x_{ij}\theta_j \right) \right)^2, \tag{2.5}$$

where, Θ represents the collection of all the weights, including the bias θ_0 and the weights for the features θ_1 through θ_p. This leads to the optimization of the model with:

$$f_\Theta = \arg\min_{\Theta} L(\Theta). \tag{2.6}$$

To prevent overfitting and to reduce the number of variables impacting predictions we can apply a regularization method. The two most popular ones are L_1 (LASSO) and L_2 (ridge) regularization. Tibshirani was the first to introduce L_1 regularization [24], which adds a penalty term that penalizes the absolute values θ_j by a certain factor λ:

$$f_\Theta = \arg\min_{\Theta} L(\Theta) + \lambda \sum_{j=0}^{p} |\theta_j|. \tag{2.7}$$

Hoerl and Kennar were the first to introduce L_2 regularization [25], which penalizes the model by the square of the model weights:

$$f_\Theta = \arg\min_{\Theta} L(\Theta) + \lambda \sum_{j=0}^{p} \theta_j^2. \tag{2.8}$$

When considering explainability, L_1 regularization is generally preferred since it pushes models to become sparse, with most weights set to zero. Whereas L_2 regularization tends to lead to models with lots of small weights since the squared penalty is tiny for small numbers.

Multiple linear regression is generally considered to be a glassbox model, as the impact of a variable x_j on the target variable y can be directly inferred from the weight θ_j: changing feature values x_j by one unit changes the prediction by θ_j when we keep all other values the same. One caveat is that the human mind has a working memory of limited capacity and when the number of features with nonzero weights θ_j is very large it may be difficult for us to make sense of the model. To improve interpretability one can opt to transform continuous variables into (multiple) binary features. Binary features are easy to interpret since we only have to look at the bias θ_0 and the coefficients θ_j of the features that have a value of 1 to have an understanding of the output.

In linear models, the feature importance is the absolute value of the t-statistic, which is the estimated value of its weight scaled with its standard error [26]:

$$t_{\theta_j} = \frac{\theta_j}{\mathrm{SE}(\theta_j)}, \tag{2.9}$$

where SE() is the standard error of estimation. This incorporates the fact that we consider a feature x_j for which we are more certain about its related weight θ_j, it is rated as more important. Note that multicollinearity increases the standard error of estimation of the affected weights θ_j and therefore leads to lower t-statistics. Since the weight θ_j is directly related to the scale of $\mathbf{x}_j$, it is important to perform scaling on all features to get relevant t-statistics, with z-normalization being the most popular method:

$$x_{i,j,\mathrm{norm}} = \frac{x_{i,j} - \mu_{\mathbf{x}_j}}{\sigma_{\mathbf{x}_j}}, \tag{2.10}$$

where $\mu_{\mathbf{x}_j}$ represents the mean of the feature $\mathbf{x}_j$ and $\sigma_{\mathbf{x}_j}$ represents the standard deviation of the feature x_j. This transformation results in a feature with a mean of 0 and a standard deviation of 1.

2.3.3. Generalized Linear Models

Linear regression assumes a linear relationship between the independent variables in $\mathbf{X}$ and the target variable $\mathbf{y}$. Unfortunately, this assumption excludes many cases, such as the Bernoulli distribution, where the target can only assume two values: 0 or 1. Nelder and Wedderburn were the first to propose the Generalized Linear Model (GLM) as a solution [27]. To handle nonlinear relations between features and target variables the linear regression model is expanded with a link function g, which performs an operation on the result of the linear regression:

$$g(f_\Theta(x)), \tag{2.11}$$

where $f_\Theta(x)$ is defined in equation 2.3.

A common example of a GLM is logistic regression, where the link function $g()$ is the logistic function, which squeezes the outcome between 0 and 1:

$$g(f_\Theta(\mathbf{x}_i)) = \frac{1}{1 + \exp(-f_\Theta(\mathbf{x}_i))}. \tag{2.12}$$

Adding a link function to a linear model makes interpretation more difficult. Changes of one unit to a variable x_j still change $f_\Theta(\mathbf{x}_i)$ by θ_j. However, when g is nonlinear in $f_\Theta(\mathbf{x}_i)$, changes to $f_\Theta(\mathbf{x}_i)$ have different effects on the outcome depending starting value of $f_\Theta(\mathbf{x}_i)$ [26]. The optimal parameters Θ of a GLM are found in a manner similar to linear regression:

$$f_\Theta = \arg\min_\Theta \sum_{i=1}^{n} \left(y_i - g \left(\theta_0 + \sum_{j=1}^{p} x_{ij}\theta_j \right) \right)^2 . \tag{2.13}$$

Regularization can be applied in the same manner as to regular linear regression, by adding an L_1 or L_2 penalty term to the cost function as shown in eq 2.7 and eq 2.8.

2.3.4. Generalized Additive Models

Generalized Additive Models (GAM) are the superclass to which linear regression, logistic regression, and many others belong. Any model of the following form is considered a GAM:

$$\hat{y} = \theta_0 + \sum_{j=1}^{p} f_j(x_j) \tag{2.14}$$

This is similar to the linear regression model, with the modification that each $\theta_j x_j$ is replaced by a more flexible shape function $f_j(x_j)$, which can be a nonlinear function. Compared to GLMs, instead of one general nonlinear link function that maps the feature values $\mathbf{x}$ to the target y, we have p different nonlinear functions f_j that map the feature values $x_j \in \mathbf{x}$ to the target y. This gives us more flexibility in modeling the relationship between the features in $\mathbf{X}$ and our target variable $\mathbf{y}$ compared to GLMs.

The component functions $f_j(x_j)$ are commonly learned via fitting splines as shown by Friedman to model the relationship between x_j and the target [7]. A spline function s_k is a prototype function such as $s_k(x) = x$, $s_k(x) = x^2$, or

$s_k(x) = \cos(x)$. The component functions are represented by additively combining the product of spline functions with the weights β_k: $f_j(x_j) = \beta_0 + \sum_{k=1}^{u} \beta_k s_k(x)$, where u indicates the number of splines.

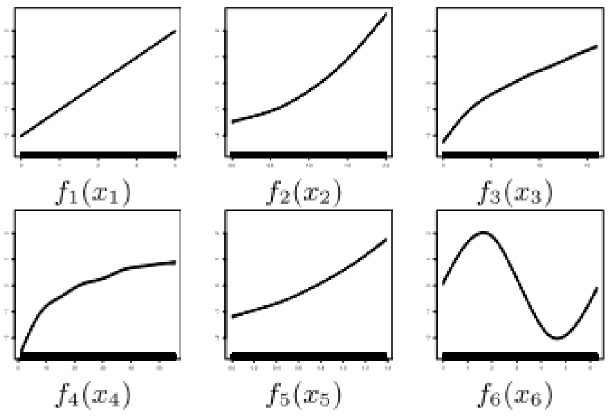


Figure 2.1: Illustration of example GAM component functions learned via spline fitting. Output value of $f_j(x_j)$ is shown on the y-axis, increasing values of x_j are shown on the x-axis. Adapted from Lou *et al* [28].

In GAMs, the overall learning takes place in a greedy manner by iteratively fitting a new component function $f_j(x_j)$ to the residual as shown by Friedman [7].

Although the shape functions $f_j(x_j)$ are allowed to be nonlinear, the resulting model is still an additive function, since the result is the sum of θ_0 and $f_j(x_j)$ it is considered a part of the family of linear models. Linear regression is also part of the GAM family since you can define $f_j(x_j) = \theta_j x_j$.

Interpreting GAMs is a bit more challenging than interpreting multiple linear regression, the relationships between features in $\mathbf{x_i}$ and $\hat{y}_i$ can be nonlinear and the slope (derivative) of the function can be different for different values of x_j, see Figure 2.1 for some examples. Plotting shape functions can help understand how the model operates and a form of global feature importance can be calculated using the outcomes of $f_j(x_j)$ weighed by the probability density function of x_j. In this case, Z-score normalization of all features is necessary to get fair comparisons between calculated feature importances.

Although more expressive than GLMs and multiple linear regression, GAMs are still incapable of modeling complex interactions between features.

2.3.5. Explainable Boosting Machine

Explainable Boosting Machine (EBM) is a tree-based GAM variant developed by Lou *et al* [5] that incorporates gradient boosting and pairwise feature interactions found through the GA^2M algorithm [29]. An EBM model takes the form of:

$$\hat{y} = \theta_0 + \sum_{j=1}^{p} f_j(x_j) + \sum f_{jk}(x_j, x_k). \tag{2.15}$$

Even without modeling feature interactions, the combination of gradient boosting and complex shape functions allows it to perform close to black-box models like Random Forest [28].

Gradient boosting is a method where weak (simple) prediction models (sometimes called learners) are built sequentially and each model is trained by minimizing the residual left by the ensemble of the previous weak models [7]. A weak learner is a simple model that performs better than chance but still relatively poorly. In the case of EBMs, weak learners only have access to one or two features to predict the outcome.

These weak learners are greedily fitted to the residual. With each iteration, p new models are added that are each fitted on a single feature. The learning rate is set very low so it takes thousands of iterations until the algorithm converges, the advantage is that the order in which the models are fitted does not matter anymore. After training, there are thousands of tree-based models s_k for each feature that can be combined additively:

$$f_j(x_j) = \sum_{k=1}^{u} \gamma_k s_k(x_j), \tag{2.16}$$

where γ_k is the weight given to that particular weak learner, and u indicates the total number of weak learners for feature j.

As an illustration, we show a classifier model trained on the US adult income data set [30], which predicts whether adults have an income above or below 50K [5].

To increase computational efficiency when making predictions, the component functions $f_j(x_j)$ are aggregated following equation 2.16 and memorized, after which the thousands of weak learners s_k can be forgotten. After completing the base model of the form $\hat{y} = \theta_0 + \sum_{j=1}^{p} f_j(x_j)$, EBM finds pairwise feature interaction through GA^2M. Where the model is used as a base and feature interactions that most improve on the residual of the resulting model are greedily added iteratively. The number of feature interactions modeled is generally kept low (e.g., 5-10 feature pairs).

Unlike other gradient boosting methods such as XGBoost, LGBM and Random Forest, EBM models are interpretable glassbox models: the contribution of a feature x_j to the final prediction can be understood by plotting f_j, and f_{jk} in

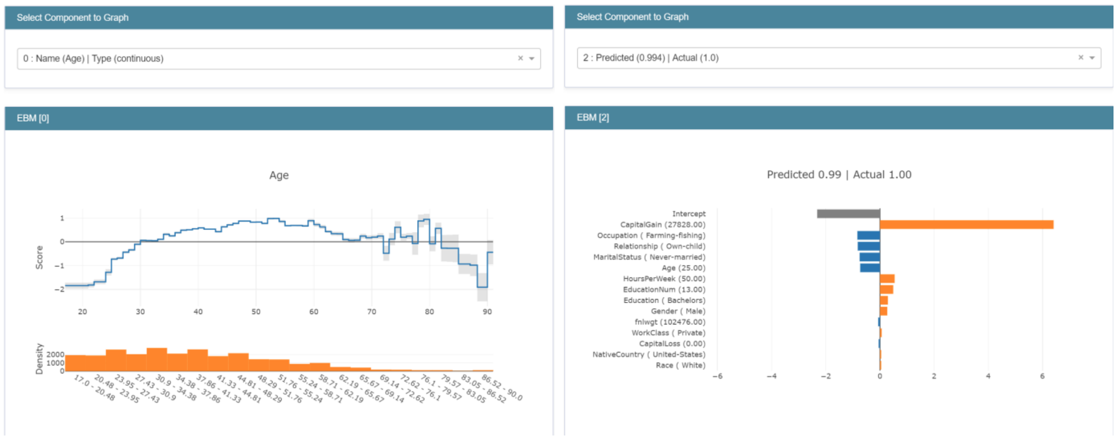

Select Component to Graph
0 : Name (Age) | Type (continuous)
EBM [0]
Age
Score
Density
Select Component to Graph
2 : Predicted (0.994) | Actual (1.0)
EBM [2]
Predicted 0.99 | Actual 1.00
Intercept
CapitalGain (27828.00)
Occupation (Farming-fishing)
Relationship (Own-child)
MaritalStatus (Never-married)
Age (25.00)
HoursPerWeek (50.00)
EducationNum (13.00)
Education (Bachelors)
Gender (Male)
fnlwgt (102476.00)
WorkClass (Private)
CapitalLoss (0.00)
NativeCountry (United-States)
Race (White)

Figure 2.2: Examples of insights generated through an EBM predicting if adults have an income above or below 50K. Left: top panel: vertical axis shows the impact on the classification score. The horizontal axis shows value of variable *age*. The blue line shows the function f_{age}, where the grey shade indicates uncertainty in the prediction.The bottom panel shows the distribution of *age* values in the training set. Right: A single prediction instance, the panel shows the most important features, ranked by absolute impact on the prediction value, we see that the feature ”CapitalGain” had a large positive impact. Adapted from Nori *et al* [5].

the case of feature interactions. The effect e_{ij} of a feature value x_j on a single prediction $\hat{y}_i$, as shown in the right panel of Figure 2.2, is calculated by through

$$e_{i,j} = f_j(x_{i,j}) - \frac{\sum_{a=1}^{n} f_j(x_{a,j})}{n}. \tag{2.17}$$

Because of its additive nature, each feature independently contributes to the prediction in a modular manner, making it simple to understand the impact of each feature on the prediction.

2.4. Feature Selection Methods

For the benefit of interpretability and explanations, we are interested in having a feature set with low multicollinearity and redundancy. As shown earlier, features can also be selected through linear regression with regularization such as LASSO, this is called a penalty approach to feature selection [31]. In general, there are two other main categories of feature selection methods, which are called filter methods and wrapper methods.

Filter methods are generally ways of ranking variables based on their relevancy and redundancy, and are often the first step of feature selection [32]. Examples of filter methods are Chi-Square test, ANOVA, and mutual information [33]. Filter methods are attractive since they do not require a specific machine learning algorithm and are therefore computationally cheaper. One downside of the mentioned filter methods is that they only consider univariate relationships between each feature and the target variable. More advanced filter methods exist, but they are beyond the scope of this thesis.

Wrapper methods use a learner to examine the performance of models trained on different feature subsets to optimize the feature set. There are several wrapper algorithms that train models on different subsets of features and keep the subsets that perform the best on the validation set. One of the most basic methods is called forward selection (FS). It begins with an empty set and adds attributes one by one in a greedy manner. At each step, FS adds the feature that most increases model performance on the validation set. Backward stepwise selection (BS) begins with the set containing all features and greedily removes one feature at a time. Both algorithms cannot go back on their decision, once a feature is added or removed the change is permanent [32]. This drawback can be overcome by (sequential) Floating Search (SFS), which allows the removal (in case of FS) of the weakest features in the formerly selected set to avoid traps in local minima [34]. in the case of standard FS or BS, one has to train $\frac{p(p+1)}{2}$, where p is the total number of features. This cost is even further increased in the case of SFS. Instead of selecting features by their direct improvement in model performance, other criteria can also be used to select variables, such as the Akaike Information Criterion [35].

The choice of the learning algorithm is important since the computational cost of wrapper methods is high. Models are trained on a large variety of subsets of features to select a feature at each step. Since the computational cost increases exponentially with the number of features considered, it is generally advised to first use filter and penalty-based methods to restrict the feature set before using wrapper methods [31]. Another downside is that not all subsets are considered since a greedy approach is used, unfortunately testing all possible subsets soon becomes intractable, when the number of features p grows too large. The main benefit of wrapper methods is that it evaluates the actual behavior of a model trained on a certain subset, instead of just examining statistical properties of features as is the case with filter methods [31].

Various strategies can be employed to speed up the wrapper methods, mainly by increasing the training speed of the model through hyperparameters. In some cases, one can train models on a subset of the data if learning curve analysis shows negligible performance increase with larger data sets.

In the case of models that require hyperparameter tuning, such as LGBM, implementing wrapper methods presents an additional challenge. The optimal approach would require hyperparameter tuning at each subset of features considered by the wrapper algorithm. However, this is often impractical due to computational limitations and time restraints. To circumvent this difficulty, one could approximate good hyperparameters by performing hyperparameter tuning just once on a representative feature set, found through a faster method such as a wrapper method utilizing a faster model such as linear regression.

Another potential method of speeding up wrapper methods at the cost of some exploration is to constrict the search space by using explainability methods to rank features by their prediction effects. Starting with the complete feature set, we can iteratively remove the lowest-ranking features and measure performance on the validation set as with the previously mentioned wrapper method, and rank the features again. Ranking of features can be performed using t-statistics in the case of linear Regression, the intrinsic global feature importance statistics in the case of EBM, and global SHAP feature effects in the case of black box models. Recall that in the case of FS, one has to train $\frac{p(p+1)}{2}$ models, where p is the total number of features. When restricting the search space through calculating feature importance we are left to train at most p models

One needs to be aware of the downside of making adjustments to speed up the computation time, either by reducing the number of models one needs to train or increasing the training speed. Any adjustments we make to speed up computation time create a different environment from the one in which our found feature set will operate. It is essentially a trade-off between computational efficiency and the quality of the feature selection.

2.4.1. Mutual Information

Mutual Information (MI) is a measure of the amount of information that knowledge about one variable gives about another variable:

$$\mathrm{MI}(\mathbf{x}_j;\mathbf{y}) = H(\mathbf{x}_j) - H(\mathbf{X}_j|\mathbf{y}) = H(\mathbf{y}) - H(\mathbf{y}|\mathbf{x}_j). \tag{2.18}$$

Where:

- $H(\mathbf{x}_j)$ and $H(\mathbf{y})$ are the entropies of X and Y, where a higher value indicates higher entropy (and less knowledge)
- $H(\mathbf{x}_j|\mathbf{y})$ and $H(\mathbf{y}|\mathbf{x}_j)$ are the conditional entropies of $\mathbf{x}_j$ given $\mathbf{y}$ and $\mathbf{y}$ given $\mathbf{x}_j$.

When two variables $\mathbf{x}$ and $\mathbf{y}$ are independent, knowing information about one doesn't give you information about the other. In this case, the MI would be 0, since $H(\mathbf{x}_j) = H(\mathbf{X}_j|\mathbf{y})$. If the two variables carry information about each other $H(\mathbf{x}_j) < H(\mathbf{X}_j|\mathbf{y})$, which leads to a positive value for $\mathrm{MI}(\mathbf{x}_j;\mathbf{y})$. MI is discussed in more detail in [36].

2.5. Black Box Models

2.5.1. LGBM

Light Gradient-Boosting Machine (LGBM) is a Gradient Boosting Decision Tree (GBDT) package developed by Microsoft, with many options to increase training speed over rivals such as XGBoost [37], [38]. In contrast to EBM, which is also a GBDT, LGBM allows for more flexibility in the weak learners, as the weak learners in EBM generally only have access to a single, or at most two features. In the case of LGBM, more elaborate trees that can model higher order feature interactions are allowed. In contrast to other GBDT algorithms such as XGBoost, LGBM grows its trees through leaf-wise growth, instead of depth-wise growth. This enables the construction of trees that more effectively minimize the loss function. Similar to XGBoost, in addition to the loss function's gradient (first-order) derivative, LGBM also uses the Hessian (second-order) derivative.

The main speed advantage of the default LGBM algorithm is achieved by splitting continuous variables into discrete bins, a technique called the Histogram-based Gradient Boosting method [38]. Though not enabled by default, LGBM offers two further improvements to increase training speed. The first is Gradient-based One-Side Sampling. Gradient-based One-Side Sampling keeps all samples with a large gradient and randomly samples from the remaining instances, this focuses the training processes on instances that are hard to predict, over instances that the model already predicts correctly. By leaving out many of the samples with small gradients, significant speed-up is achieved. The second is Exclusive Feature Bundling. Exclusive Feature Bundling finds exclusive features (meaning that they do not take non-zero values simultaneously) and bundles them into one feature. This leads to a reduction of features, which significantly speeds up the training process. See the original LGBM paper for a more in-depth discussion of these methods [38].

Many methods exist to perform hyperparameter tuning, an effective and model-agnostic method is the Tree-Structured Parzen estimator (TPE). TPE uses Bayesian optimization to estimate the relation between model performance and hyperparameters, leading to a more efficient method of searching through the hyperparameter space when compared to grid search and random search [39].

As the resulting model is an ensemble of many trees, LGBM is considered a black-box model. Explaining LGBM outputs can be done using TreeSHAP [40], which is a fast, model-specific implementation of SHAP that is only applicable to trees.

2.5.2. DeepAR

DeepAR is an autoregressive and probabilistic forecasting method that uses a Recurrent Neural Network with long short-term memory (LSTM) [2]. For an introduction to Neural Networks (NN), we refer to the book *Pattern Recognition and Machine Learning* [41]. The underlying model of DeepAR is trained to provide forecasts in the form of a probability distribution. When a Gaussian distribution is chosen, the model outputs a mean $\mu \in \mathbb{R}$ and a standard deviation $\sigma \in \mathbb{R}^+$ at each timestep. In the case of positive count data, a better choice of distribution is the negative binomial distribution, which is characterized by its mean $\mu \in \mathbb{R}^+$ and shape $\alpha \in \mathbb{R}^+$ [42]. DeepAR produces forecasts by taking samples from the probability distribution provided by the underlying model. These sample values are used autoregressively by the model to provide probabilistic outputs at further time points up to the end of the chosen forecasting horizon. The resulting n forecasted time series will be aggregated to provide a final prediction by taking the mean or median values at each time point. Although DeepAR is considered a black box model, some valuable insight into the uncertainty of the forecasted values can be gained thanks to its probabilistic approach. As seen in Figure 2.3, we can show confidence bounds indicating what percentage of the trajectories

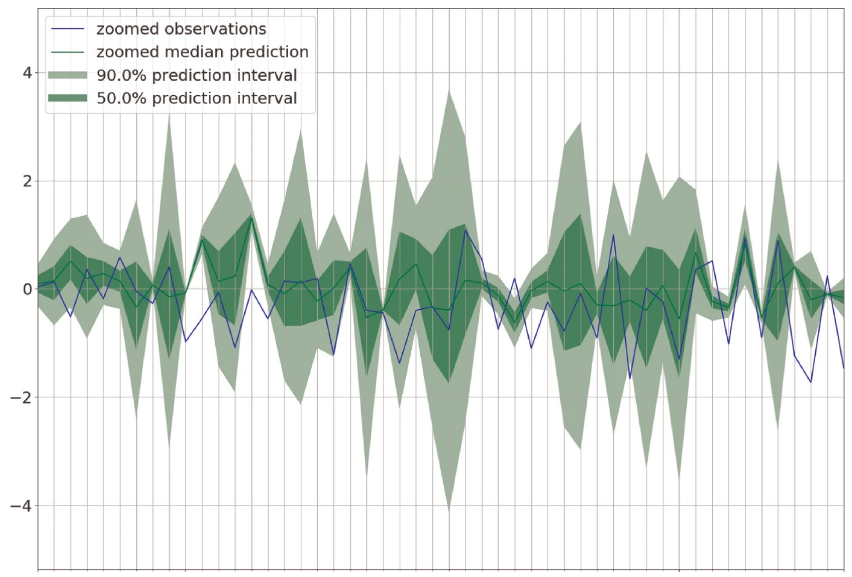


Figure 2.3: Example of a 50 day DeepAR forecast of IBEX-35, a Spanish stock market index using covariates indicating emotional sentiment in news articles. The dark green area indicates that at least 50% of the simulated trajectories pass through that area. Adapted from Consoli *et al* [43].

passed through a certain area. A tight spread indicates DeepAR is quite sure of its predictions whereas a wide spread indicates great uncertainty. DeepAR also has the ability to handle multiple time series as input, which is useful for scenarios where the time series are related and the forecasting of one can provide valuable information for the forecasting of the others. For a more in-depth discussion of the DeepAR methods, we refer the reader to the paper by Salinas *et al* [2].

2.6. Post-Hoc Explainability Methods

Glassbox models are intelligible by default and often do not need to rely on post hoc methods for interpretability. Post-hoc explainabilty methods are especially helpful when dealing with opaque models such as multi-layer NN or model ensembles.

2.6.1. Shapley Additive Explanations

SHapley Additive exPlanations (SHAP) is a model-agnostic, post-hoc method to provide explanations for individual predictions developed by Lundberg [6]. SHAP is based on the coalitional game theory of Shapley values and has the desirable properties of local accuracy, missingness, and consistency [6]. To generate an explanation for a prediction based on the feature vector $\mathbf{x}_i$, SHAP creates coalitions of features, in a coalition, some feature values are *known*, and their true value is used, features not part of the coalition are considered missing. There are multiple approaches to assigning values to the missing features:

- Taking the average value of the feature in the data set.
- Randomly sampling from the feature's values in the data set.
- Sampling from the marginal distribution.

By default, the python SHAP package samples from the marginal distribution in the data set [6].

The feature effect $e_{i,j}$ of a feature j at instance i, is calculated by examining the difference in prediction between coalitions with and without the feature j. Coalitions with very few members get high weights since we can clearly see the impact of the few features involved. On the other hand, coalitions that contain close to all features also get a high weight, since if a feature can still impact the prediction when so many other features are known, it must be an important one. The weight of a coalition is given by:

$$\frac{|S|!(p-|S|-1)!}{p!}, \tag{2.19}$$

where $|S|$ is the number of features in the current coalition, and p is the total number of features. The general formula to calculate the feature effect $e_{i,j}$ of feature j of instance i using SHAP is:

$$\phi_j(x_i) = \frac{1}{p} \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(p - |S| - 1)!}{p!} [f_{S \cup \{j\}}(x_{S \cup \{j\}}) - f_S(x_{i,S})], \tag{2.20}$$

where F is the complete feature set, and $f_S(x_{i,S})$ is the models prediction when it has access to all features, and $f_{S \cup \{j\}}(x_{S \cup \{j\}})$ the model's prediction when it only has access to coalition of features $S \cup \{j\}$.

By creating SHAP explanations for all instances in the test set, we can calculate a measure of *global* feature importance. For each feature j we sum the *absolute* value of the feature effect calculated at each instance:

$$\texttt{global\_importance}(j) = \sum_{i=1}^{n} |e_{i,j}|. \tag{2.21}$$

Dividing by the number of instances gives us the average feature importance of the feature j. In the case of unbalanced data sets it could be wise to add a further normalization step, by applying max-abs normalization on the feature effects of each instance i:

$$\texttt{max\_abs}(\mathbf{e}_i) = \frac{|\mathbf{e}_i|}{\max_i(|\mathbf{e}_i|)}. \tag{2.22}$$

This ensures that feature effects of predictions of high traffic OD-pairs don't have an outsized impact.

To see how well SHAP explanations represent the decision-making of the model, we run fidelity plots, where for each instance i in the test set, we take the true values of the top k SHAP features and set all other features to the training set average value. The (in)fidelity of this model $f_k()$ is the squared error $[f_k(\mathbf{x}_i) - f(\mathbf{x}_i)]^2$ between itself and the base model that has access to all true values. We calculate this per instance and then average it on the validation or test set to get an overall fidelity per number of top k features [44]. At $k = 0$, the error is greatest, and at $k = p$ it is zero, it is generally plotted by converting the errors to a ratio between 0 and 1.

TreeSHAP is a model-specific variant that was also proposed by Lundberg [40], that can be used on tree-based models such as simple decision trees, but also on more complex gradient boosted trees such as LGBM. TreeSHAP does not directly permute the feature values to see their impact on the prediction. Instead, it traverses the trees and follows both sides of the split when a feature is not in the coalition S. The main advantage of TreeSHAP is its speed: the computational complexity of regular SHAP is $\mathcal{O}(tl2^p)$, which scales exponentially with the number of features. Whereas TreeSHAP has a complexity of $\mathcal{O}(tld^2)$, where, d is the maximum depth of the trees, l is the maximum number of leaves, and t is the number of trees [26].

2.7. Hybrid approaches and representation learning

Representation learning is about the discovery of useful representations of data, where a data representation is a method of transforming the raw samples into some new, more manageable space [45]. An example application would be the representation of one day of an OD time series as a 6-dimensional vector, down from the raw time series which is a 48-dimensional vector. Reduced representations

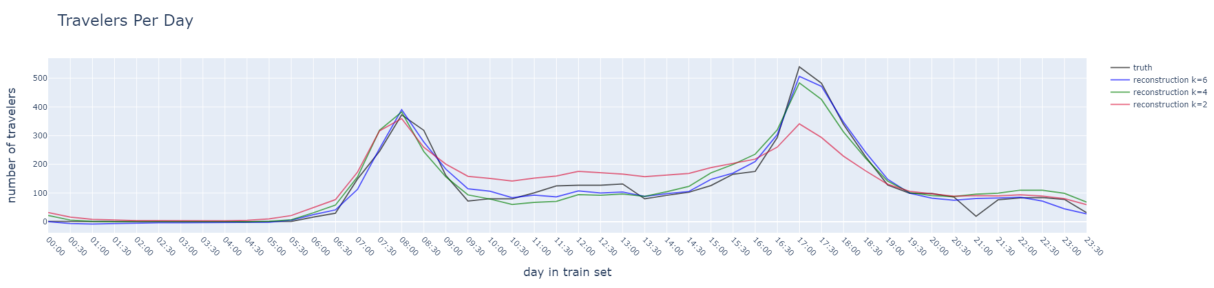


Figure 2.4: Quality of reconstruction when representing one day of one OD-pair using the top 2, 4, and 6 components.

of this kind could enable our models to consider more of a time series history, without massively expanding the number of features. The reduced number of features necessary to represent the history of a time series results in a smaller risk of overfitting [46], and possibly a more understandable model for humans.

However, some information is generally lost during such a dimensionality reduction. The severity of this information loss is closely related to the number of latent variables we choose to use to represent the raw data, where the use of fewer variables yields a simpler model, at the cost of greater information loss. The task of deciding the optimal number of latent variables is contentious since it is essentially a trade-off between the level of dimensionality reduction and the information loss in the reduced representation.

2.7.1. Singular Value Decomposition

A common approach to representation learning is through dimensionality reduction techniques such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). SVD is a matrix factorization technique that decomposes a matrix into the product of three matrices.

SVD is a linear technique, meaning that it only operates on linear relationships between variables. It has the property that it can factorize any matrix, regardless of its size or shape. When performing dimensionality reduction through SVD, the singular values of the matrix are ranked, with the larger singular values explaining more variance in the data. The first k components, where k is the number of dimensions we want to reduce the data to, are then selected. This guarantees that the most variance in the data is represented using only the first k components, which are linear combinations of the original variables. By selecting only the most important components, it is possible to reduce the dimensionality of the data while still retaining as much information as possible. An example of how accurately SVD can represent on day of a time series using only 2, 4, or 6 values can be seen in Figure 2.4.

Given SVD's linear nature, nonlinear relations present in time series may be impossible to properly encode. Multi-layer neural networks are widely known for their ability to learn nonlinear features. If nonlinear traits are present in our data, these might hamstring the ability of SVD to learn highly compressed representations. Using the nonlinear nature of multi-layer neural networks we might be able to achieve a more accurate representation at the same level of dimensionality reduction. Furthermore, SVD treats all variables equally and does not take temporal relationships into account. To leverage the temporal relations present in time series, a neural network architecture with convolutions or RNN may be used [47]. For a more in-depth discussion of SVD we refer you to the book *Data-Driven Science and Engineering* [46].

2.7.2. β-VAE

Hybrid approaches generally consist of a black box step in which a (deep) NN learns an effective feature representation at reduced dimensionality, which is taken as input by a simple statistical model to make the prediction. A method developed by Higgins *et al*, called β Variational Autoencoder (β-VAE), can perform the dimensionality reduction step and produce disentangled features, meaning that the features produced by the model make sense to humans [48].

A disentangled representation is one where each of the units in the latent representation is sensitive to changes in only one data generating factor, while being relatively insensitive to changes in other factors [48]. The benefits of such a representation are twofold: higher robustness to noise, and higher interpretability [49].

Using hybrid approaches to generate highly expressive disentangled features through deep learning encoder-decoder structures looks promising. CoST, a novel framework for learning disentangled representations of time series followed by simple regressor models outperforms pure black box models based on LSTM RNNs and others [50].

β-VAE is able to create *interpretable factorized latent representations* on image data [48]. It might be possible to adapt the method to be applicable to time series. The paper does note that high disentanglement leads to worse reconstructive quality. This is a trade-of between reconstruction quality and disentanglement.

Total Correlation Variational Autoencoder (Beta-TCVAE) is an improved version of β-VAE developed by Chen *et al* [51]. β-TCVAE works by adding an additional term to the loss function, which penalizes the model if the latent variables are correlated with each other, encouraging the latent variables to be statistically independent.

The difference between methods like SVD and β-VAE is threefold: first, the independent latent variables in SVD are not necessarily disentangled, second, β-VAE can learn nonlinear relationships, and third, unlike SVD, β-VAE does not require the latent variables to be completely orthogonal.

The Variational Autoencoder (VAE) [52] is a generative model that learns to reconstruct high-dimensional data through a multi-layer NN. A low-dimensional bottleneck layer separates the model into two separate parts: an encoder and a decoder. The encoder is the part of the model that condenses the input data $\mathbf{x}$ into a lower dimensional 'latent' representation $\mathbf{z} = \textit{encoder}(\mathbf{x})$, and the decoder network reconstructs (or generates) the input data from this latent representation $\hat{\mathbf{x}} = \textit{decoder}(\mathbf{z})$. To enable learning the model through stochastic gradient descent, Kingma and Welling proposed the reparameterization trick, which entails the modeling of the latent dimension $\mathbf{z}$ as a collection of Gaussian distribution based on $\mathbf{x}$:

$$q_\phi(\mathbf{z}|\mathbf{x}), \tag{2.23}$$

where ϕ are the parameters of the NN, and $q_\phi(\mathbf{z}|\mathbf{x})$ represents a Gaussian distribution with its mean and standard deviation generated by the encoder. The decoder part of the model, parameterized by θ, then generates an estimate $\hat{\mathbf{x}}$ of the original sample from this latent representation $\mathbf{z}$. Technically, the decoder generates a Gaussian distribution $p_\theta(\mathbf{x}|\mathbf{z})$ for each individual variable in $\mathbf{x}$ from which we sample to obtain the estimated data point $\hat{\mathbf{x}}$. In practice, we can often use the means of the distributions to reconstruct the data.

The parameters are learned through the following loss function:

$$\mathcal{L}(\phi, \theta; \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})), \tag{2.24}$$

where the first term represents the reconstruction loss, which is the expected log-likelihood of the original data given the latent representation, and the second term is the Kullback-Leibler (KL) divergence, which enforces the distributions $q_\phi(\mathbf{z}|\mathbf{x})$ to be close to the standard normal distribution $\mathcal{N}(0, 1)$.

The β-VAE [48] is an expansion of the loss function with the hyperparameter β, which controls the trade-of between the KL-divergence and the reconstruction loss:

$$\mathcal{L}(\phi, \theta; \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})). \tag{2.25}$$

As shown above, the β value plays a controlling role in managing the trade-off between reconstruction loss (reconstruction accuracy) and KL divergence (disentanglement). Increasing the β will lead to an emphasis on the KL divergence term, thus prioritizing disentanglement over reconstruction. Finding the optimal β value is challenging given the subjective nature of the intelligibility of the latent space, which is our main reason for pursuing disentanglement.

One approach to finding a suitable β value would be to first calculate the reconstruction error at $\beta = 0$, and increase β until the reconstruction error surpasses a defined threshold:

$$\texttt{reconstruction\_error\_at}\beta = 0 \times \texttt{threshold} \leq \texttt{reconstruction\_error\_at\_}\beta \geq 0. \tag{2.26}$$

This approach seeks to maximize disentanglement while keeping the drop in reconstruction accuracy within acceptable limits.

Recall that the final goal of the β-VAE model is to produce intelligible features that summarize the history of a time series for improving forecasting. This means that the level of disentanglement, and the reconstruction quality are at best proxies to our goal. To examine the intelligibility of the latent dimension we will need to perform manual inspection. The most common method of interpreting latent dimension is through a process called single latent traversals [53]: we first generate base values for the latent vector, this can be done either by encoding a representative instance or by setting all latent values to 0, something that is rendered practical through the KL divergence loss, which encourages a latent variable distribution that is close to $\mathcal{N}(0, 1)$. By permuting one latent variable at a time through a range of $[-3, 3]$, and keeping all other latent variables at a value of 0 (or the latent values generated through a seed time series), we can inspect the effects of each latent variable through plotting the time series that get produced by inserting permutations of latent space $\mathbf{z}$ into the decoder.

2.8. Related Work in Explainable Time Series Forecasting

There is a large body of work concerning explainable time series classification, unfortunately, most of these methods are unsuitable for application in forecasting. For example methods that use counterfactuals as explanations [54], or use representative subsequences called shapelets that maximally represent a given class [55, 56]. The literature concerning explainable time series forecasting is considerably less expansive.

The temporal fusion transformer (TFT) architecture proposed by Lim *et al* in 2021 [57] has been rapidly gaining popularity since its publication. TFT not only outperforms DeepAR on benchmarks, but it can also provide attention-based explanations about its predictions. Many applications using this architecture are popping up: Wu *et al* used historical tourism volume and big data from search engines and travel forums to predict future tourism demand using an optimized TFT architecture [58]. Gunnarson used TFTs to predict supermarket food demand in an effort to reduce food waste [59]. Santos *et al* used the TFT architecture for day-ahead solar power production where it outperformed XGBoost and a baseline LSTM NN model [60]. Of special interest is the paper of Zhang *et al* who forecast metro traveler flow using TFT [61], which is very similar to our OD forecasting.

Various methods exist that can provide accurate time series forecasting with built-in interpretability, such as N-BEATS [62]. The authors of N-BEATS provide some methods that can provide explanations such as on the seasonality trends observed by the model. Although this architecture achieves high scores on forecasting benchmarks, there is no research on the usefulness of the explanations it provides. Alternatively, we have the temporal fusion architecture proposed by Lim *et al* [57]. Although Lim *et al* show some examples of how the explanations the system provides can be useful, they have not tested if the explanations are actually intelligible to humans in the real world.

There are hybrid approaches like CoST that learn disentangled representations of time series followed by simple linear models, where the disentangled nature makes the representation potentially understandable [49,50]. Though interpretable in theory, the intelligibility of for example CoST has never been examined in practice.

Todo *et al* use a convolutional variational autoencoder (CNN VAE) to perform dimensionality reduction on high-dimensional multivariate ECG data [47]. It outperforms alternate approaches such as wavelet decomposition. In contrast to the OD time series, the time series used in this publication are highly irregular, with seasonalities that have irregular periodicity. Finally, we have Jabeur *et al* who show how a combination of an XGBoost forecasting model with SHAP explanations results in interpretable high-accuracy forecasting of gold prices [63].

Although the methods mentioned above all achieve impressive state-of-the-art performance, how interpretable these explanations are to humans remains debatable.

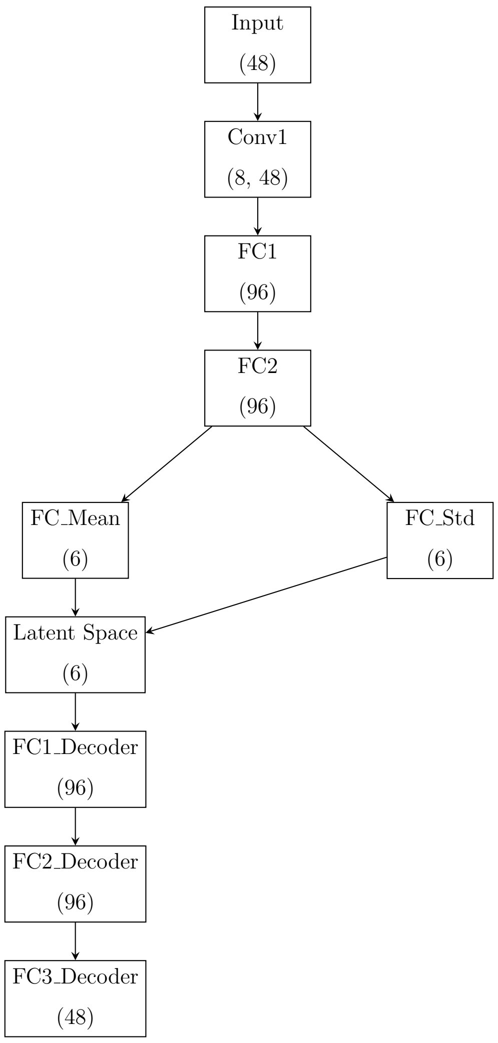


Figure 2.5: Architecture of the β-VAE model used in the experimental phase. The convolutional layer has a kernel of size 3, padding of 1, and stride of 1.

3. Research Approach

3.1. Proposed method

Recall that the goal of this thesis is to answer the following main research question:

MRQ *Will an explainable or interpretable machine learning model with engineered features outperform the baseline DeepAR model, in terms of performance measures such as MAE, in predicting train traveler demand, where traveler demand is modeled as a collection of time series?*

To answer this question we divide it into four research questions (RQ):

RQ 1 *Are the interpretable models with handcrafted features significantly better compared to the baseline model in terms of performance measures such as MAE?*

RQ 2 *Are hybrid approaches of representation learning followed by an interpretable model significantly better compared to the baseline model in terms of performance measures such as MAE?*

RQ 3 *Are black-box models, such as LGBM, significantly better compared to the baseline model in terms of performance measures such as MAE, when using techniques like SHAP for interpretability, and using the input features from RQ 1 and RQ 2?*

RQ4 *How faithful are the SHAP explanations generated for the LGBM models, in terms of performance measures such as MAE, when we are only allowed to* *consider the 7 most important features of each prediction?*

3.1.1. Research Question 1

We used multiple linear regression, and EBM as glassbox models to forecast traveler demand. The input features are historical time series data and exogenous data such as the presence of holidays. We iteratively engineer features, train models and evaluate feature importance using explainability methods followed by pruning unimportant or unintelligible features, see Figure 3.1. Features were created by consulting with domain experts, and exploratory experiments. Feature pruning was performed based on feature importance measures generated using the filter and wrapper methods described in section 2.4.

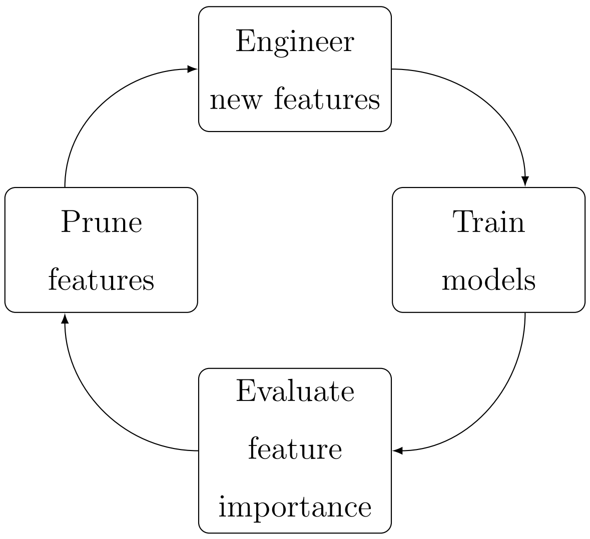


Figure 3.1: Strategy to iteratively engineer an optimal set of features

Forecasting performance was compared with the baseline DeepAR model on a one day forecasting horizon, meaning the models predict one day in the future. Multi-output models are used that make forecasts of 24H into the future at midnight. The primary metric used to evaluate performance is MAE.

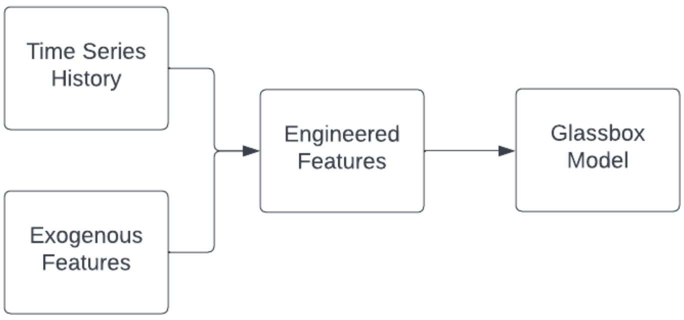


Figure 3.2: Model pipeline used for answering RQ 1

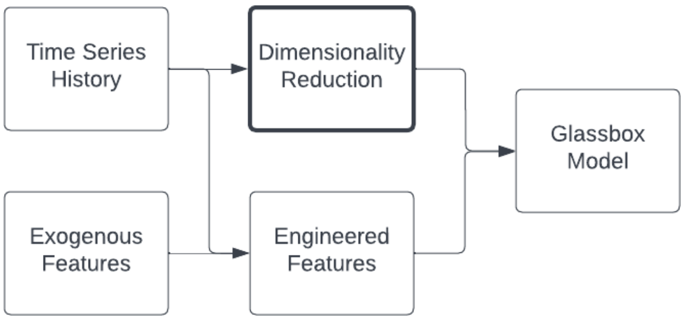


Figure 3.3: Model pipeline for answering RQ 2

3.1.2. Research Question 2

We used various dimensionality reduction and representation learning methods such as β-VAE, and SVD. The learned representations are added as additional features to the glassbox models from RQ1. Forecasting performance was once again be measured on a one-day horizon against DeepAR. Additionally, the models are compared against models that receive the additional features without dimensionality reduction applied, to see if dimensionality reduction is beneficial over adding a large number of features.

3.1.3. Research Question 3

LGBM models were trained using the inputs designed when answering RQ 1 and RQ 2. Parameter tuning was performed using a multivariate Tree-structured Parzen Estimator optimization algorithm [64] on the validation set. TreeSHAP was used to gain insight into the model [6, 40].

3.1.4. Research Question 4

We used TreeSHAP for the LGBM models to examine relative feature importances at global and local levels. We produced fidelity plots for the SHAP explanations to examine if explanations with only the top k features closely resemble the forecasts of the complete models.

3.2. Performance Measures

To measure the predictive performance of all models, we will primarily use Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y_i}|. \tag{3.1}$$

MAE was chosen over Root Mean Squared Error (RMSE) because we believe that bigger errors are not exponentially more impactful than smaller ones.

This is because one of the major use cases of OD-level forecasting is in predicting train-level crowdedness. To calculate train level crowdedness, OD-level forecasts of many OD-pairs are used, minimizing the impact an outlier prediction in a single OD-pair has.

Another common error measure used in time series forecasting is Mean Absolute Percentage Error (MAPE), which calculates the mean absolute error as a percentage of the true value. MAPE is unsuitable in our case, since we have time series with values that are zero, or close to zero. Dividing by these values to get percentage errors leads to very large, but essentially meaningless numbers.

3.3. Data for Experimental Validation

3.3.1. NS Data

The NS currently provides transportation to approximately 250 stations in a fully reachable network. This means there are around $250^2 = 62500$ Origin Destination (OD) pairs that a traveler can pick as a start-end combination and each OD pair has its own historical time series since the beginning of 2018. Most of these OD time series are incredibly sparse and therefore have little impact on actual train crowdedness. Sorted by number of travelers, there are about 7 500 relevant OD pairs that can give a relative complete picture of train occupancy. These 7 500 most active ODs represent around 95% of daily travelers. We can represent around 99% of the travelers with 18050 OD pairs, leaving the remaining 45 706 OD pairs representing just 1% of travelers.

We used data from only a subset of OD-pairs to answer the research questions for practical reasons. Expanding our research to encompass all 67000 OD-pairs would lead to excessively long and expensive compute times. However, a substantial number of OD-pairs need to be used to receive reliable results. To balance

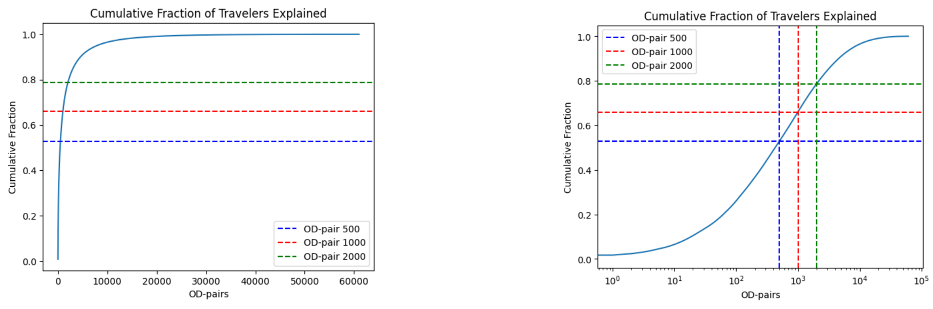


Figure 3.4: Fraction of travelers contained in the biggest n OD pairs. The horizontal axis shows the number of OD-pairs included, sorted from largest to smallest, normal scale on the left, and log scale on the right plot.

computational costs and practical relevance, we used the 2000 OD-pairs with the highest volume. This subset of OD-pairs contains just under 80 % of all trips between (2022-04-25) and (2023-04-25). Unfortunately, this means we are still missing 20 % of all trips.

The sparse nature of most OD-pairs makes forecasting extra challenging. We have captured some of that sparsity in the OD-pairs between 1000 and 2000, but the extreme sparsity of the remaining 65000 OD-pairs remains an interesting challenge: aggregated, these time series with low volume still contribute a number of passengers that can be the difference between a busy and quiet train.

3.4. Feature Engineering

Through exploratory data analysis, expert knowledge, and experimentation, a collection of potential features have been designed. These features can be roughly categorized into two categories: exogenous and endogenous features. Exogenous features are those that are constructed using information not available in the time series data itself, such as national holidays, and school vacations. On the other hand, endogenous variables are those that can be derived from the time series itself. In order to maintain internal consistency, the information derived from the time series should not include any data beyond the prediction point.

In the context of school vacations, we must be wary that different regions in the Netherlands can have different school vacation dates, meaning that it could be a school vacation in the region of the destination station or vice versa. To account for this we resort to information from the Dutch government, which keeps track of these dates [65], and link it to our station location data. This yields two distinct school vacation features, one for the destination and one for the origin station. National holidays are identical nationwide and are therefore easy to implement.

Turning our attention to the endogenous variables, these features are the backbone of our models' predictive ability, features have been designed that capture certain historic properties such as trends, outliers, and general patterns in each of our 2000 time series. Exploratory analysis through lagged autocorrelation, which can be seen in Figure 3.5 shows high correlations between travel behavior of past days, especially with the previous day, and 7, 14, 21, and 28 days in the past, which is the same day of the week, one to 4 weeks ago.

The full list of features can be examined in Appendix A.

3.4.1. Handcrafted Features

In this section, we will use the notation of a univariate time series denoted as $\mathbf{Y} \in \mathbb{R}^{D \times T}$ where D represents the total number of days and T represents the number of time points within each day. The variable $y_{d,t}$ represents the target

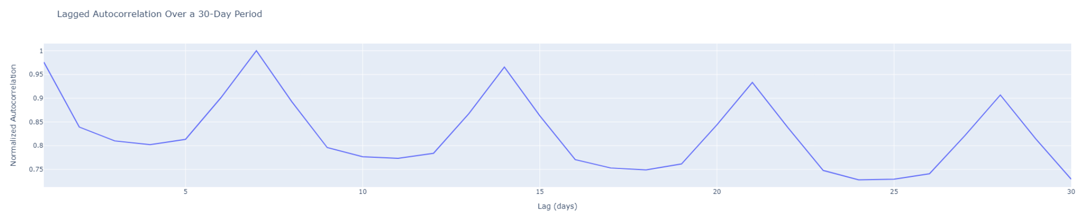


Figure 3.5: Visualization of the accumulated impact of max-abs normalized lagged autocorrelations for 48 half-hour intervals in a day over 2,000 distinct time series, as discussed in Appendix C.1. The data showcases the correlation extent of traveler behavior with its various lagged durations up to 30 days. The normalization ensures each time series contributes equally to the visualization.

value for day d at time point t.

The strong repeating patterns in our time-series data indicate that the type of features we need to engineer may differ from those used in other feature engineering projects on time series data. In contrast to domains like EEG, ECG, and stock forecasting, we do not require complex features that capture shifting seasonalities and frequencies. Instead, we focus on daily and weekly patterns, with attention to outliers and slow-moving upward or downward trends.

We developed a base set of features that give a coarse-grained representation of historic data points of an individual time series. The features focus mostly on aggregate data from 7 days ago, to capture travel behavior on the same day of the week, one week ago. And data from 1 day ago, to notice very recent trends in travel behavior. Also included are the days of the week one-hot-encoded. Training preliminary models with this base feature set uncovered some challenges in producing accurate forecasts.

- **Sensitivity to noise:** The model was too sensitive to variations in the history of 7 days ago.
- **Slow adjustment to sudden drops:** If there was an unexpected drop in travel, such as during a maintenance period, the model was slow to adjust and overestimated travel.
- **Slow recovery after sudden drops:** Similarly, after a sudden drop in travel, the model was slow to recover and continued to underestimate for a longer period.
- **Chronic underestimation of some extremely busy Origin-Destination (OD) pairs:** Certain OD pairs that experienced high-volume traffic were consistently underestimated by the model.
- **Increased errors during morning and evening rush hour:** The model struggled to accurately predict the surge in travel during peak hours.

To reduce the sensitivity to random changes 7 days, ago we introduced moving average versions of the features that represent general traits of the historical travel behavior of 1 and 7 days ago. For example, we created moving average features that track the average number of travelers on a specific day of the week, over the last 4 and 6 weeks.

Two forms of predictable processes that can produce outliers have been identified and included as features: national Holidays and school vacation periods. Some processes are harder to model in advance due to insufficient data quality, such as the occurrence of large social events such as festivals and football matches. Furthermore, we have (planned) maintenance and unplanned outages that not only cause massive drops in the number of travelers on these routes. There is also an impact on other routes that suddenly have to service an increased number of travelers through reroutes. The two main flows of historical data the prediction model uses are that of the traffic of 7 days ago, and that of yesterday. We devised two log-ratio features to provide additional context to the data of these two days, the benefit being that the additional features enable the model to learn complex interaction between type of outlier and traveler data. These features compare the moving average number of travelers on a given day of the week, compared to the last time that day of the week occurred.

$$\text{Outlier Feature 7 days ago} = \log_2\left(\frac{\text{6 Week Moving Average of Travelers} + 1}{\text{Number of Travelers 7 days ago} + 1}\right) \tag{3.2}$$

$$\text{Outlier Feature 1 day ago} = \log_2\left(\frac{\text{6 week Moving Average of Travelers} + 1}{\text{Number of Travelers 1 day ago} + 1}\right) \tag{3.3}$$

Using this knowledge the model should be more knowledgeable about the state of its recent historical data. It can use the outlier information from 1 day ago to rapidly adjust to major changes in travel behavior, and use the outlier information from 7 days ago to understand how reliable recent historical data is. Since large reductions in the number of travelers over an OD-pair can occur during maintenance or outages, it is possible to get very large negative values for the outlier features. We decided to limit values below -4 to -4 to prevent those occurrences to have an outsized effect on these feature values.

To mitigate the underestimations a *days since start* feature has been created, since part of the underestimation may be caused by lower travel numbers in the early dates in the train set due to slow post-covid ramp-up.

To reduce errors during morning and evening rush hour two approaches were devised, both related to providing more detailed historical information about the morning and evening rush hour time points. The first approach is to include the peak rush hour number of travelers, for morning and evening rush hour. We find the half hour where the number of travelers are highest and encode this number of travelers as the features moring_peak_travelers and evening_peak_travelers. Using the peak traveler information we created two more features, that describe how 'pointy' the rush hour is, by taking the logarithm of the ratio between the peak travel numbers and the total number of travelers in the morning and evening respectively (features 23 and 24 in Appendix A).

In the feature representation part, we will use the dimensionality reduction methods from section 2.7 to efficiently represent data to provide more info on the overall shape of the rush hour business, specific to each OD pair and day of the week.

3.4.2. Features for Dimensionality Reduction

The number of values in the time series' past that are potentially relevant is large. Completely representing just 1 full day at the most detailed granularity of 30 min time slots gives us 48 additional features. Our aim is to keep the number of relevant features for our model relatively low, such that a human can better grasp what happens when the model makes a prediction.

Through autocorrelation analysis, we found that the time series history of 7 days ago is the day in the past that is most highly correlated with the current day. Furthermore, we saw that models tend to prefer features that incorporate some kind of moving average, to increase the robustness of the feature. Combining these two properties gives us 48 moving average features, one per time slot during the day:

$$\text{moving average}(d, t) = \sum_{w=1}^{4} \frac{y_{d-7w,t}}{4}, \tag{3.4}$$

where $t \in \{1, 2, \ldots, 48\}$ denotes the time slot during the day, and d denotes the day in the time series that we wish to predict. To keep the number of features relatively low we will attempt to represent these 48 moving average features using dimensionality reduction techniques SVD, and β-VAE. This allows us to roughly represent the 48 features as a smaller number of features.

4. Experimental Results

4.1. Experimental Setup

Covid has substantially impacted travel behavior in 2020, 2021, and parts of 2022. Therefore, we have decided to use only data from 2022-02-05 and onward. The data set has been partitioned into a training, validation, and test set using time-based splitting to ensure that there is no contamination or information leakage between sets:

- Training set starts from 2022-02-05 and ends on 2023-01-04 (334 days).
- Validation set starts from 2023-01-05 and ends on 2023-02-05 (31 days).
- Test set starts from 2023-02-06 and ends on 2023-04-30 (84 days).

The training and validation sets are used to develop optimal models, which include hyperparameter tuning and feature selection. The performance of the best models will be assessed on the test set to answer our research questions.

The t-statistics and p-values are calculated using paired t-tests comparing the MAE value of the respective model against the MAE value of the DeepAR model for each of the 2000 OD-pairs. To be more precise, per model, we compile an MAE vector of 2000 elements, each representing the MAE at a different OD pair. Then, this MAE vector is compared with the MAE vector of 2000 elements from the DeepAR model.

4.1.1. Base Model Design

We use a multi-output model as our base model. This model makes predictions for exactly one day ahead, with a separate model used to predict each half-hour timeslot. We made this choice because of the large impact that the time of day has on forecasting. If we used a single model, the model would need to take feature interactions between time of day and the other features into account quite extensively. Alternatively, it would require extensive feature engineering to design complex features that take these relations into account. This would result in a single, but vastly more complex model, with features that would presumably be less intelligible.

Throughout the process, we utilized three versions based on this design. The first is using linear regression models from the sklearn package [66], as models in the multi-output design. The second is with using LGBM models implemented from the LGBM package [37], and the third is with Explainable Boosting Machines implemented through [5].

The linear regression models are completely standard, without a regularization penalty, and the EBM models used default parameters. All feature engineering, selection, and hyperparameter tuning was performed using just the training and validation sets. All results of LGBM models on the test set include hyperparameter tuning using multivariate TPE, with parameters (n_trials:50, n_startup_trials: 20).

4.1.2. DeepAR

We used the DeepAR implementation from the gluonts probabilistic forecasting package [67]. The DeepAR comparison model was trained on the same data set. The optimal hyperparameters were:

- epochs: 50
- num-layers: 4
- num-cells: 40
- rest: default settings in gluonts 0.1.0.2.

Training the model on the 2000 OD-pairs took approximately one hour and forecasting on the test set took approximately 6 hours to complete.

4.2. Evaluation of Models using Exclusively Handcrafted Features

44 features have been handcrafted, based on the ideas discussed in Section 3.4. The full list of handcrafted features with detailed descriptions can be found in Appendix A. A correlation heatmap can also be seen in Figure B.3. Using this feature set we used the FSF wrapper method, which compensates for the greedy nature of FS by allowing the removal of selected features. We will refer to the feature set that resulted from the wrapper method with LR as lr_sfs_features. Furthermore, we used an LR model with a LASSO regularization of $\alpha = 0.1$ and considered all features that had an absolute sum of coefficients below 2 across all 48 models to be removed. We will refer to this feature set as strict_lasso_features. MI was also performed as an alternative method to perform feature selection, the results of which were not further utilized, but they can be seen in Figure B.1 in the

Appendix. The feature set, lr_sfs_features, proved superior to strict_lasso_features, achieving lower RMSE (6.40 vs 6.46) on the validation set[1] . The LR wrapper method was completed first, which led to the identification of the optimal LR feature set. To find the optimal feature set for the LR model we used a random sample of 5 % of the data since learning curve analysis showed no clear impact on model performance, as can be seen in Figure C.2. The results of these feature selection methods can be seen in Figure 4.1. Agreement of selected features between LASSO, and SFS approaches using LR can be seen in table 4.1. The agreement between the lgbm_sfs_features and lr_sfs_features feature sets can be seen in table 4.2.

Table 4.1: Agreement between strict_lasso_features and lr_sfs_features

	lr_sfs_features	Not lr_sfs_features
strict_lasso_features	21	7
Not strict_lasso_features	10	6

Table 4.2: Agreement between lgbm_sfs_features and lr_sfs_features

	lr_sfs_features	Not lr_sfs_features
lgbm_sfs_features	20	8
Not lgbm_sfs_features	7	9

Subsequently, the lr_sfs_features feature set was used to perform hyperparameter tuning for LGBM using TPE. Using those hyperparameters, FSF was performed to find the optimal set of handcrafted LGBM features. We will refer to this feature set using lgbm_sfs_features. Since EBM was computationally intensive to train, we had to restrict the total number of models to train. We decided on 4

[1]We later switched to using MAE as the primary error measure, and the strict_lasso_features feature set actually achieves a lower MAE than lr_sfs_features on the validation set (2.77 vs 2.84). For consistency, a better choice would have been to use the strict_lasso_features feature set instead, however, we expect only small performance differences between the two.

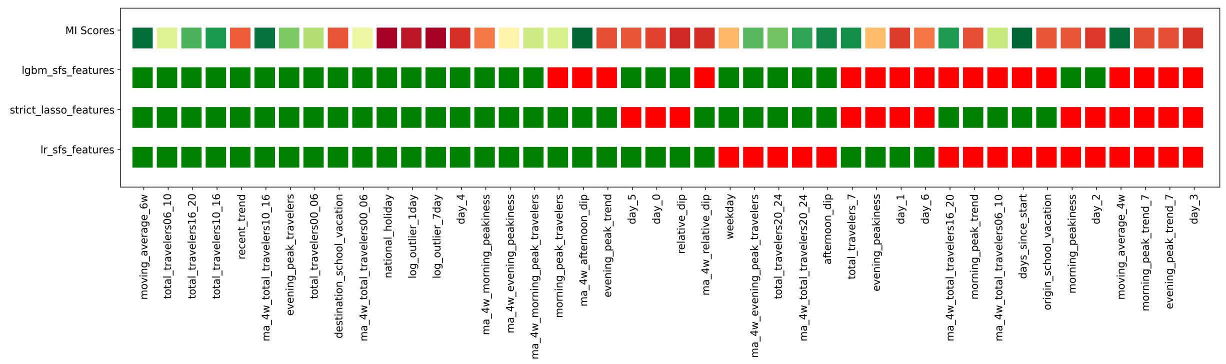


Figure 4.1: Handcrafted feature sets found through methods described above. Here, green signifies that the feature is included in the set, while red indicates exclusion. lgbm_sfs_features is the feature set found through FS with SFS and LGBM as model. lr_sfs_features is the feature set found through FS with SFS and LR as model. strict_lasso_features is the feature set found through eliminating features with very small or zero weighted corresponding coefficients. Note the color gradients for the Mutual Information feature selection method, where a greener color means the feature carried more information about the target variable and a red color indicates very low information about the target variable. The value of the feature that carried the most mutual information, *days since start*, was reduced to improve the visualization. Since all features get a non-zero importance value, inclusion/exclusion of the features depends on the cutoff point. See Figure B.1 for more insights into the MI feature importance

potential feature sets: the 2 optimal feature sets found through SFS, the intersection, and the union of those feature sets. The optimal feature set for EBM, which led to the highest forecasting accuracy on the validation set, was determined to be lgbm_sfs_features. A visualization of global feature importance of this EBM model can be seen in Figure B.2 in the Appendix.

Model	Feature Set	MAE	p-value	t-statistic
lr_base	lr_sffs_features	3.1910	$< 10^{-6}$	-12.275
lgbm_base	lgbm_sfs_features	**2.7659**	0.0761	1.774
ebm_base	lgbm_sfs_features	3.1324	$< 10^{-6}$	-16.769
DeepAR	TS history + national_holidays	2.8145	-	-

Table 4.3: MAE of base models with handcrafted feature sets on the test set, and paired t-test results from comparison with the baseline DeepAR model.

Comparison of these models, called lr_base, lgbm_base, and ebm_base with DeepAR can be seen in table 4.3. t-statistics and p-values were calculated using paired t-tests as discussed in section 4.1. Although the lgbm_base model has the best MAE, it is not significantly superior to DeepAR (p: 0.0761).

4.3. Evaluation of Models using a Combination of Handcrafted Features and Dimensionality Reduction Features

To see if we can improve model performance using additional features in a compressed form, we performed dimensionality reduction on the moving average of the last weeks same-day data, as discussed in Section 3.4.2. To find the optimal number of singular values, we iterated from 1 to 20 singular values and tested model performance on the validation set,of which the results can be found in Figures C.4, C.5. We decided to truncate at 6 components since the following components were hard to interpret by humans. What follows is that each day of a time series is represented by a unique set of six values, which, when multiplied by their corresponding components and aggregated, closely approximate the original time series. Components are ranked by importance in terms of the proportion of variance they explain, the top component has the highest importance. The shape of the top 12 components can be seen in Figure C.6, which shows increasingly erratic patterns in components with smaller singular values. The shape of the top 6 components of the SVD learned on the training data can be seen in more detail in Figure 4.2. For the β-VAE reduction method, we constructed a β-VAE model as specified in Figure 2.5. All time series data was transformed to a [0,1] scale, which enabled the use of a sigmoid function in the final layer of the decoder [2] . After an optimization process, which is discussed in Appendix C.5. As can be seen in Figures C.9, C.10, predictive performance of both LR and LGBM models was hardly impacted by the hyperparameters of the β-VAE model. Therefore, the selection of the main β-VAE used from this point onward was made purely from the point of understandability. Latent traversal plots were generated to evaluate the intelligibility of the latent variables inside the models. The most understandable model was the result of hyperparameters with a latent dimension of 6 and β = 3e-5, which can be seen in Figure 4.3. For example, we can see that latent variable #1 encodes the balance between morning and evening rush hours in numbers of travelers. Latent variable #2 encodes the overall scale of the time series the model represents.

[2]Unfortunately, other activation functions in the final layer led to the consistent occurrence of exploding gradients in the training process.

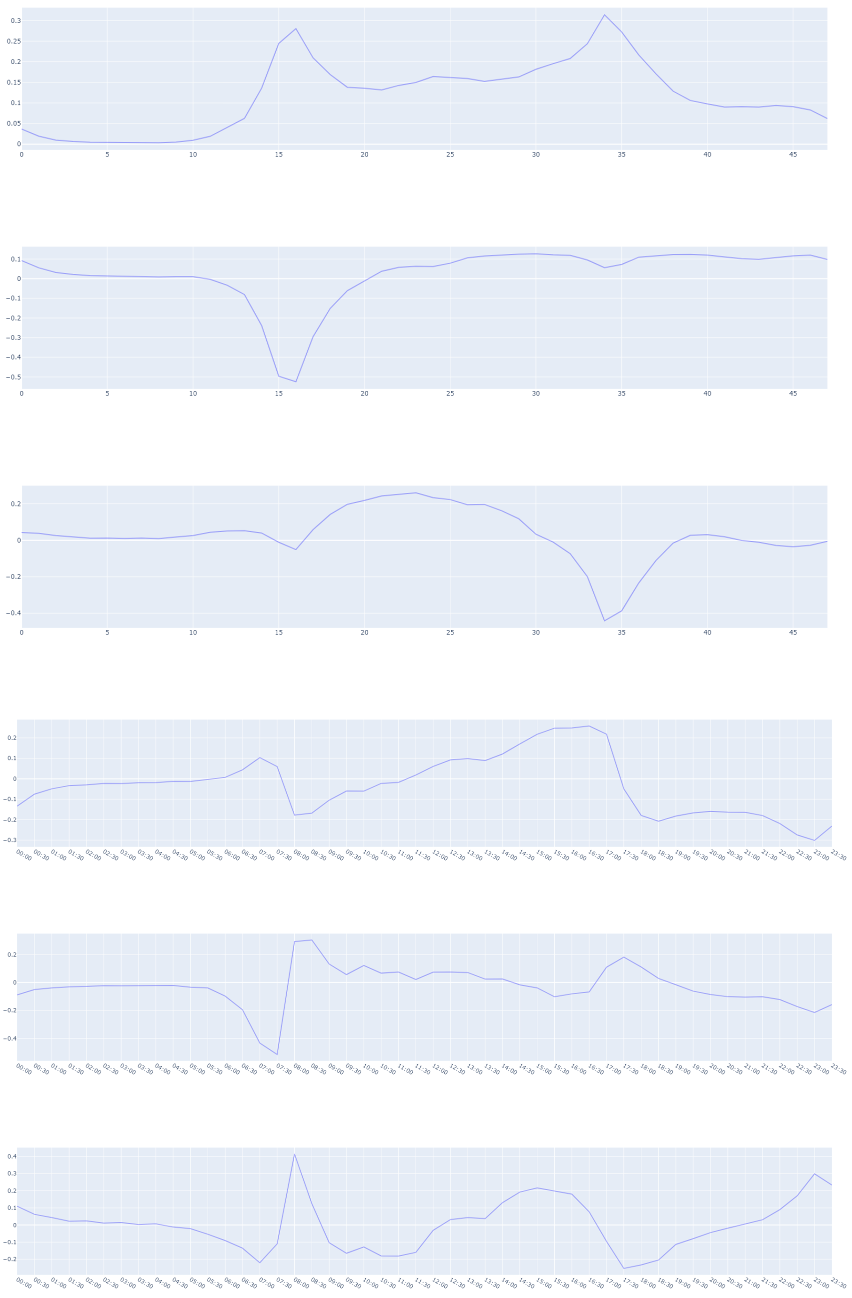


Figure 4.2: The shape of the six most important components learned when training an SVD on time series with a length of 1 day. the horizontal axis shows the time slots and the vertical axis shows how a change to the latent variable of that component changes the resulting time series.

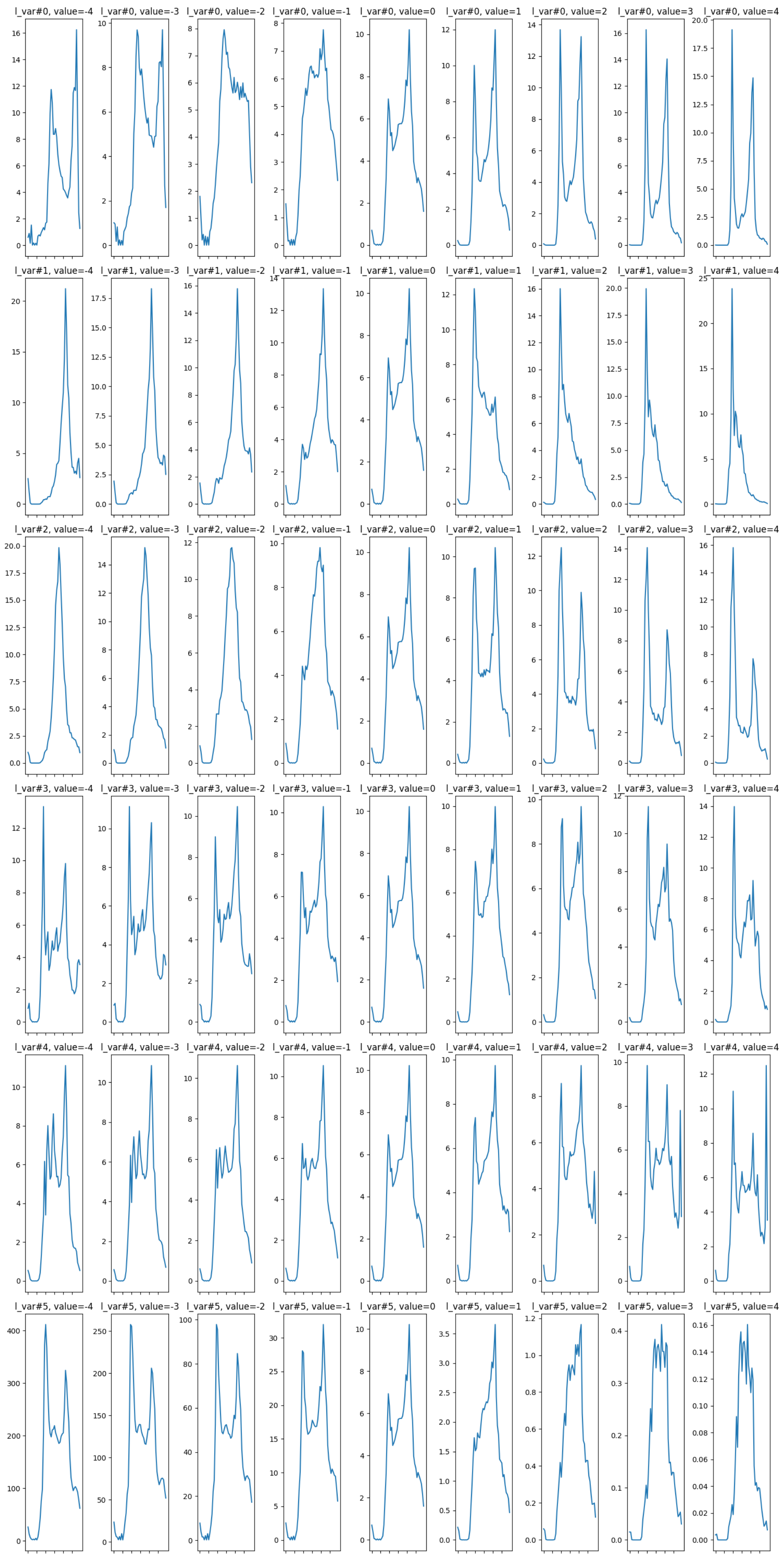


Figure 4.3: Latent traversal of the β-VAE model we used to generate additional features at a reduced dimensionality. Each subplot shows the generated time series if all latent variables are kept at 0, except for the latent variable we are examining. The horizontal axis shows the time, which is 24 hours, and the vertical axis shows the number of travelers at each time slot. The overall plot can be used to see what each variable encodes.

Model	Feature Set	MAE	p-value	t-statistic
lr_vae	lr_sffs_features + β-VAE	3.111	$< 10^{-6}$	-9.67
lr_svd	lr_sffs_feature + SVD	2.974	$< 10^{-6}$	-4.93
ebm_vae	lgbm_sfs_features + β-VAE	3.055	$< 10^{-6}$	-12.14
ebm_svd	lgbm_sfs_features + SVD	2.952	$< 10^{-6}$	-6.42
lgbm_vae	lgbm_sfs_features + β-VAE	**2.628**	$< 10^{-6}$	6.11
lgbm_svd	lgbm_sfs_features + SVD	**2.640**	$< 10^{-6}$	5.87
DeepAR	TS history + national_holidays	2.8145	-	-

Table 4.4: Comparison of models with access to handcrafted features, and additional historical data in a reduced form, compressed by β-VAE or SVD.

All three model types (LR, LGBM, EBM) show improvement over their versions that only have access to handcrafted features. Dimensionality reduction through SVD seems slightly superior to β-VAE for LR and EBM models. For the LGBM models, β-VAE was slightly better.

4.4. Evaluation of Models using a Combination of Handcrafted Features and the Additional Features in Uncompressed Form

Furthermore, models were trained where we do not compress the additional features from section 3.4.2. All three model types perform better with uncompressed data than with compressed data. However, the number of features that are considered by these models has roughly doubled, making the models potentially less intelligible.

As an additional model, we propose a LASSO model with $\alpha = 0.33$, which yields a high-accuracy, sparse model, that is interpretable by humans. As input features, it received the lr_sfs_features, and the uncompressed features from section 3.4.2. The α value was decided using Figure B.4, which plots the MAE on the validation set for different values of α, and also plots the average number of nonzero weights for those α values. This allowed for a balanced decision which took both models sparsity and accuracy into account. It achieves an MAE of 2.7967, which is not significantly different from DeepAR (p: 0.5502), whilst having an average of around 12 nonzero coefficients, including its bias.

Model	Feature Set	MAE	p-value	t-statistic
lr_no_comp	lr_sfs_features + uncompressed	**2.8021**	0.7182	0.3610
lasso_no_comp	lr_sfs_features + uncompressed	**2.7956**	0.5502	0.5975
lgbm_no_comp	lgbm_sfs_features + uncompressed	**2.5487**	$< 10^{-6}$	8.5193
ebm_no_comp	lgbm_sfs_features + uncompressed	2.8639	0.0197	-2.3334
DeepAR	TS history + national_holidays	2.8145	-	-

Table 4.5: Comparison of models with access to handcrafted features, and additional historical data in an uncompressed form. Note the addition of the lasso_model,

4.5. Fidelity of SHAP explanations

SHAP yields explanations of the LGBM models. However, only explanations that consider all features in the feature set are a 100% match with the actual prediction. Given the limited working memory capacity of humans, it is important to visualize how faithful the explanations are compared to the actual model, when we only consider the top k features. To assess the quality of truncated SHAP explanations, we run fidelity plots as discussed in section 2.6.1. The fidelity of the explanations was examined using the first 7 days of the test set: (2023-02-06 to 2023-02-12). We see that the lgbm_base model that only uses the lgbm_sfs_features feature set achieves better explanation fidelity compared to the other models.

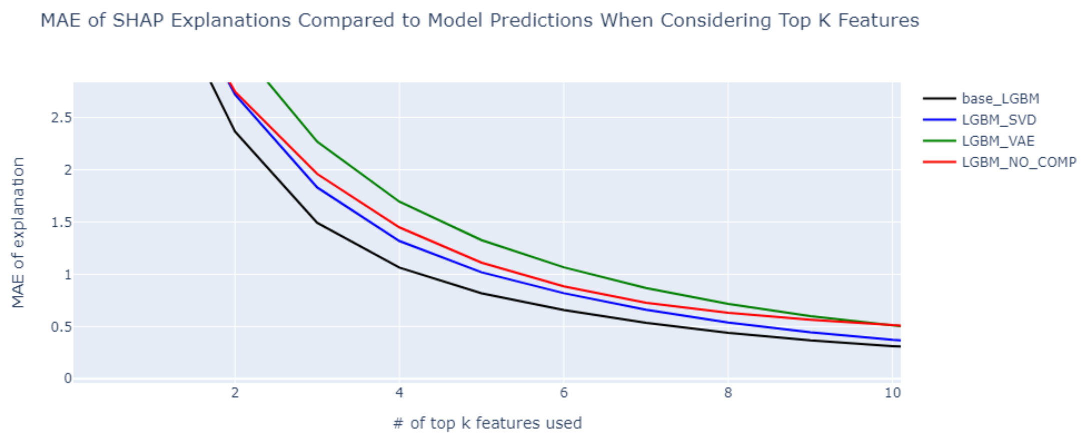


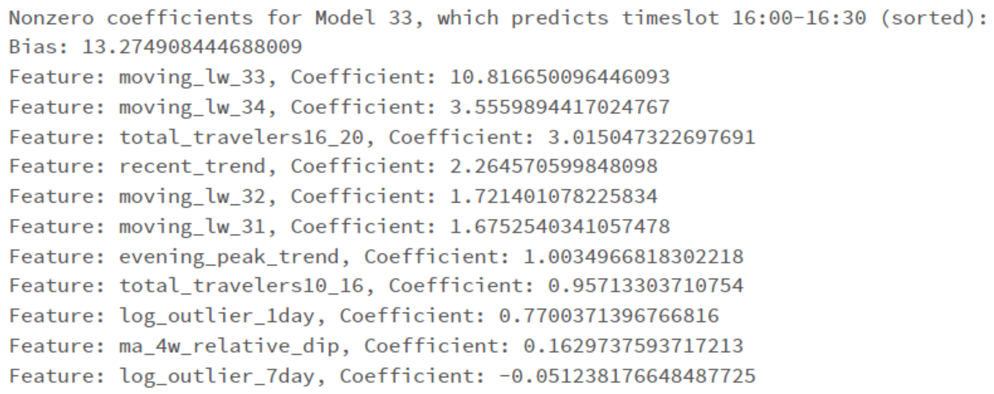
Figure 4.4: Infidelity of SHAP explanations when limiting the explanation to the top #k components, measured as the MAE between the predictions of the SHAP explanations versus the predictions of the actual model. The horizontal axis shows the number of top feature effects were taken into account to approximate the original prediction. The vertical axis shows the MAE between the truncated SHAP approximation and the original prediction.

LGBM with β-VAE features shows the worst explanation fidelity at $k \leq 9$. At $k = 10$ SHAP explanations for all four models achieve an `MAE` < 0.55

4.6. Intelligibility of selected models

Our results above show two models of interest: the lasso_no_comp, and the lgbm_no_comp model. The LASSO model if of interest, because of its glassbox nature and relative sparsity of its coefficients, while at the same time performing equal to DeepAR. The lgbm_no_comp is of interest since it achieves the lowest MAE of all models on the test set.

To showcase the interpretability of the lasso_no_comp, we show the full internals of the model that forecasts the timeslot (16:00-16:30) in Figure 4.5. How often the features are selected by the 48 LASSO models that make up the lasso_no_comp model can be seen in Figure B.5 in the Appendix.



```
Nonzero coefficients for Model 33, which predicts timeslot 16:00-16:30 (sorted):
Bias: 13.274908444688009
Feature: moving_lw_33, Coefficient: 10.816650096446093
Feature: moving_lw_34, Coefficient: 3.5559894417024767
Feature: total_travelers16_20, Coefficient: 3.015047322697691
Feature: recent_trend, Coefficient: 2.264570599848098
Feature: moving_lw_32, Coefficient: 1.721401078225834
Feature: moving_lw_31, Coefficient: 1.6752540341057478
Feature: evening_peak_trend, Coefficient: 1.0034966818302218
Feature: total_travelers10_16, Coefficient: 0.95713303710754
Feature: log_outlier_1day, Coefficient: 0.7700371396766816
Feature: ma_4w_relative_dip, Coefficient: 0.1629737593717213
Feature: log_outlier_7day, Coefficient: -0.051238176648487725
```

Figure 4.5: All nonzero coefficients of model 33, the LASSO model that predicts the timeslot of 16:00-16:30. All nonbinary features are z-normalized, allowing for a straightforward calculation of feature effects. moving_lw_# are the names of the uncompressed features discussed in 3.4.2, the number indicates the time slot, with indexing starting at 1, identical to the model indexing.

The lgbm_no_comp model is our top performer in terms of accuracy. Using SHAP, we can get an understanding of individual predictions by modeling the relationship between the features and the prediction. in Figure 4.7, a randomly selected forecast is shown with its SHAP explanation. To inspect which features are globally important to the model, we calculated global feature importance as discussed in section 2.6.1. The figure containing the SHAP global feature importances can be viewed in Figure B.6 the Appendix.

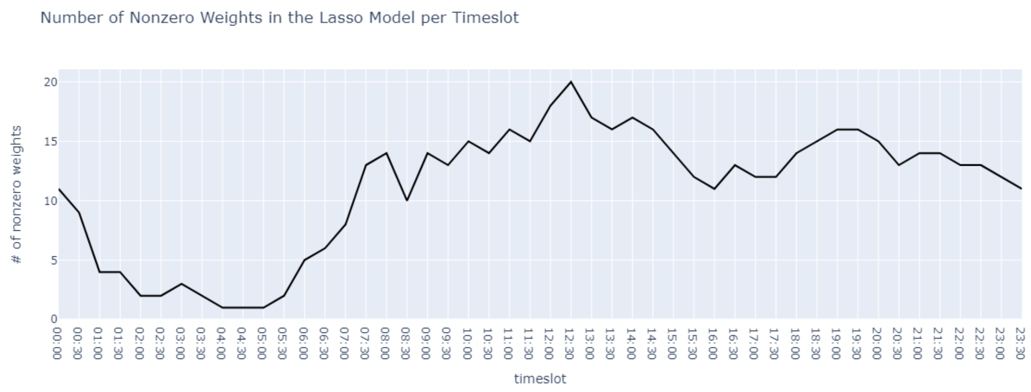


Figure 4.6: Number of nonzero weights per model in the LASSO multi-output model discussed in section 4.6. The horizontal axis shows the timeslot to which the specific model belongs, and the vertical axis shows the number of nonzero coefficients. Models that predict early morning timeslots have a low number of relevant features

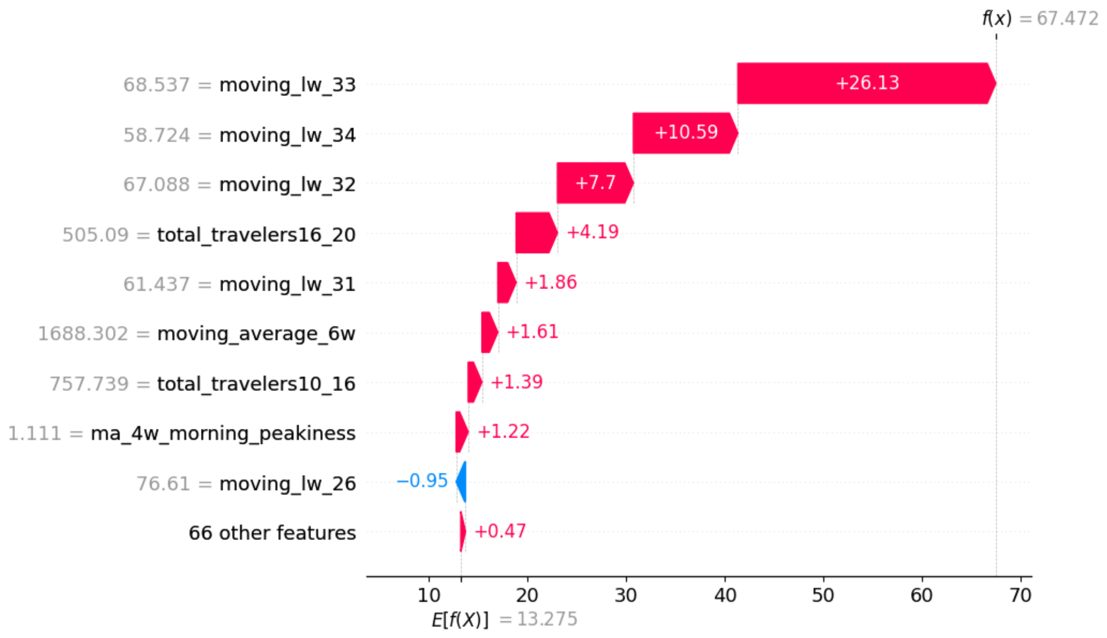


Figure 4.7: SHAP explanation of a single prediction by the lgbm_no_comp model that predicts the timeslot of 16:00-16:30. The prediction, if all features were unknown can be seen near the bottom: $E[f(X)] = 13.275$. Feature values are shown next to the feature names on the left side. The SHAP weighted feature effect on the prediction value can be seen in or next to the colored bars. Where a red colored bar and a plus sign indicate the feature value increased the prediction value. The prediction $\hat{y}$ of the model can be seen in the top right corner as $f(X) = 67.472$, which is close to the true value $y = 68.817$

5. Discussion and Future Work

While this thesis presents models that are significantly more transparent than the DeepAR baseline model, it did not answer some questions that may be interesting for future work to adress. One key aspect is the evaluation of interpretability. We have successfully developed a Lasso model with an average of 12 nonzero weights that performs on par with DeepAR, and the lgbm_no_comp model that surpasses DeepAR and can be combined with SHAP to gain insight. We did not evaluate the perceived interpretability of these models by data scientists and other professionals who would potentially use these models.

Using the terms coined by Doshi-Velez and Kim, we have not evaluated the intelligibility of our models on the *application-grounded*, and *human-grounded* levels. Therefore, a direction of future work is to include a user study that examines the usefulness and intelligibility of the models in practice. For an investigation on the *application-grounded* level, this study should involve data scientists and other professionals, replicating as close to real-life usage scenarios as possible to test their performance at the *application-grounded* level. Alternatively, a more feasible study would be on the *human-grounded* level, where humans perform simplified tasks to evaluate interpretability.

We have opted to not convert some continuous features into categorical features that might improve interpretability at the potential loss of some accuracy. Future research could explore which features lend themselves well to such conversion. Furthermore, future work could try to quantify how much more intelligible categorical features are, by performing experiments on the *human-grounded* level.

This would enable a more evidence-based approach in deciding which kind of features to engineer.

In the case of the Lasso model, some further performance gains may be relatively easy to achieve by changing the design of some features: features that incorporate logarithmic transformations and ratios are hard to use by LR models, because of their linear nature and inability to model feature interactions. If an outlier is shown on a logarithmic scale, a LR model has no idea how much it should change the prediction, since it can't access other features to get an idea of the scale of the time series. For example, a reduction of 50% of travelers leads to a log_outlier value of -1, but that -1 has a different meaning if there are 500 travelers of 50 travelers on that time series. In the first case, the feature value should create a -250 effect on the predicted number of travelers, and in the second scenarie a -25 effect. A LR model is incapable of modeling that in its current design. For an optimal Lasso model, features should be engineered in a way that they provide useful information without requiring the model to understand the scale of the time series. This can be achieved by avoiding logarithmic transformations and focusing on absolute differences, among other strategies. Especially in the case of the log_outlier_1day feature, further improvements may be possible. Its number 1 rank in the global feature importance of the lgbm_no_comp model (Figure B.6) indicates that it contains a lot of potentially useful information that the Lasso model fails to exploit currently. It may be of interest to design alternative outlier features to see if they improve the performance of LR models.

Preliminary experiments were run to investigate the effect of the weather on travel behavior. No clear link could be found between temperature or precipitation and travel behavior, but the exploration had a limited scope. Only weather data from one central location was used (De Bilt), as a consequence, for stations far away from De Bilt, the weather data was less relevant. Furthermore, for both temperature and precipitation features, we did not experiment with delta features (features that indicate positive or negative change compared to a number of days earlier). Within this limited scope, the addition of continuous or categorical weather features seriously decreased model performance on the validation set. As is shown by the Scheveningen model mentioned in the introduction, weather data can be highly beneficial in some specific cases. It might be possible to identify a subset of OD-pairs, like the ones to Scheveningen station that do show a strong relationship between weather and travel behavior. Subsequently, a specific model could be trained on this subset of OD-pairs that does incorporate weather data.

Preliminary experiments were run to investigate if the clustering of OD-pairs could improve predictive performance. Experiments of clustering OD-pairs using K-means clustering with a Manhattan distance measure showed insignificant effects when cluster membership was added as a collection of binary features. Since the time series show strong weekly seasonality, we chose to represent a week as a single instance, meaning each OD-pair only had 52 instances for an entire year of data. This limited the quality of the clustering that we could perform since we needed to maintain a relatively low dimensionality compared to the number of instances per OD-pair. Future work could include alternative clustering methods and distance measures. Lower dimensional representations of each week through methods such as SVD and β-VAE seem promising avenues to deal with the curse of dimensionality. Training separate models per cluster is also a possibility, however, this would further complicate the prediction pipeline for presumably modest gains.

The test set was relatively long (84 days) and the DeepAR model in production is retrained every week. No retraining occurred during our experimental phase, it would be interesting to see how much weekly retraining or online learning impacts model performance, and if there are different trends depending on the model type. Additionally, we have not yet tried using a weighted training set, in which recent data is assigned higher significance. This method could potentially improve predictive performance by modeling relationships that change over time more accurately, especially when combined with retraining the model every week.

We only assessed one-day ahead forecasting, but the team at NS currently provides high-quality forecasting up to three days ahead. Future work could expand the models capabilities to forecast multiple days ahead, either by adding a new multi-output model for each further day (with a slightly modified feature set) or through autoregressive forecasting.

Surprisingly, the EBM models were worse than the LR models, even though they are renowned for their ability to combine interpretability with performance close to complex black box models. A lack of hyperparameter tuning due to computational constraints may be the issue, and future work could examine the effect of hyperparameter tuning on EBM performance.

Though the dimensionality reduction methods of SVD and β-VAE are relatively effective, they fail to provide the intelligible models we wanted: the latent variables are hard to interpret, and not compressing this data leads to better models in terms of predictive performance and intelligibility.

In the case of β-VAE, the LGBM model relies on many different latent variables for it to be able to use the information stored inside, as is shown by its poor performance on the SHAP infidelity plot. This is contrasted with the lgbm_no_comp model, which mostly focuses on just three variables: the moving average values of its own timeslot, and the one before and after, which shows that a larger feature set can actually lead to a more succinct explanation.

The β-VAE could benefit from alternative scaling approaches and activation functions in the final layer. The final layer sigmoid activation function is suboptimal since most values it processes are in the range of 0.01 to 0.001. Because of [0,1] scaling when there are outlier days like kings day on busy OD-pairs really suppresses all other values, which doesn't play into the strength of sigmoid activation. Maybe a more smoothed sigmoid function would have been better.

In the experimental phase, we mostly used LR without regularization as one of the base models, and only switched to a LR model with LASSO regularization to propose a final model wich uses the lr_sfs_featureset and the uncompressed additional features. Future work could include LR models with LASSO regularization at all experimental parts, and also measure how the sparsity of the resulting models changes based on the feature set.

The SHAP explanations create the illusion of perfect fidelity, but the explanations are merely a linear approximation of the model's behavior on a local scale. This illusion is even further strengthened by the SHAP fidelity plot, which seems to suggest perfect explanations if all features are included.

Most coefficients in the lasso_no_comp model are relatively small, future work could investigate the model's performance when it is truncated at the 7 largest coefficients, which is the average working memory capacity of humans. Additionally, plots similar to the SHAP (in)fidelity plot can be generated to assess the quality of the model when one can only examine the top k coefficients.

The popularity of the moving_average_6w feature (the only feature that has a moving average that is not 4 weeks) indicates that experimenting with moving averages of varying lengths is an interesting avenue for future work to explore. Feature selection will play an important role, since there will be many potential features if one considers moving averages of different lengths, with very high collinearity between them.

The use of the multi-output architecture where each individual model only predicts one timeslot must be noted as an important reason as to why the predictions are easy to understand. We assume that the results of this thesis are of interest to other time series domains that also contain time series with high regularity and periodicity, where a multi-output model can be anchored in a similar manner. Examples of such domains include time series for electricity demand, as well as those related to various types of human traffic and crowdedness.

5.1. On Computational Demands and Environmental Effects

From both a business and ecological perspective, the computational costs of training, running, and maintaining ML models are highly relevant. For all tests, azure databricks compute clusters of type Standard_F64s_v2 were used [68]. Performance may vary between clusters of the same type, the results concerning computational runtime below should therefore be interpreted cautiously.

As mentioned earlier, DeepAR took 1 hour to train and took 6 hours to predict the full test set. This translates to roughly 4 minutes of inference time per day to predict. In comparison the LR, and LGBM models took < 5 minutes to train, and prediction of the full test set took < 10 seconds. EBM models took between 4-6 hours to train, and could also predict the full test set within 10 seconds. However, our models require an engineered feature set to function. Computing all features for the training, validation and test sets took around 1 hour and 20 minutes. This translates to roughly 10 seconds of compute to generate the features for 1 day of forecasting. Furthermore, runtime analysis showed that 51% of that time was spent creating 4 features (morning_peak_trend, evening_peak_trend, morning_peak_trend_7, evening_peak_trend_7) that are hardly used at all. The only feature that is used is morning_peak_trend, which is a part of the lr_sfs_features set. The final Lasso model never has a nonzero coefficient for any of these 4 features. Therefore, we believe a further reduction by half in computing the features can be achieved by removing these 4 features.

6. Conclusion

We have shown that the black box DeepAR model can be matched or even surpassed by models that are far less opaque. Of special interest are two models: the lasso_no_comp, and the lgbm_no_comp model. The Lasso model matches DeepAR in terms of predictive performance, whilst at the same time maintaining total transparency and limited complexity. The LGBM model surpasses the DeepAR model in terms of predictive performance, whilst being explainable through SHAP.

The proposed methods are not only improvements in terms of accuracy and/or transparency. They are computationally less demanding as well, which could lead to savings of energy and greenhouse gasses and enable more flexible deployment.

We have also experimented with the dimensionality reduction methods SVD and β-VAE in an attempt to create hybrid models with superior predictive performance whilst at the same time remaining understandable by humans. This approach yielded less impressive results, the meaning of the features generated by SVD and β-VAE are challenging to interpret. Furthermore, using the additional time series history without compressing those variables led to superior predictive performance and intelligible features, where a sparse set of intelligible features had the most predictive impact at each prediction.

REFERENCES

1. Goverde, R. and I. Hansen, “TNV-Prepare: Analysis of Dutch railway operations based on train detection data”, *Computers in Railways*, Vol. 7, pp. 779–788, 2000.

2. Salinas, D., V. Flunkert, J. Gasthaus and T. Januschowski, “DeepAR: Probabilistic forecasting with autoregressive recurrent networks”, *International Journal of Forecasting*, Vol. 36, No. 3, pp. 1181–1191, 2020.

3. Savitzky, A. and M. J. Golay, “Smoothing and differentiation of data by simplified least squares procedures.”, *Analytical chemistry*, Vol. 36, No. 8, pp. 1627–1639, 1964.

4. Delling, D., T. Pajor and R. F. Werneck, “Round-based public transit routing”, *Transportation Science*, Vol. 49, No. 3, pp. 591–604, 2015.

5. Nori, H., S. Jenkins, P. Koch and R. Caruana, “Interpretml: A unified framework for machine learning interpretability”, *arXiv preprint arXiv:1909.09223*, 2019.

6. Lundberg, S. M. and S.-I. Lee, “A unified approach to interpreting model predictions”, *Advances in neural information processing systems*, Vol. 30, 2017.

7. Friedman, J. H., “Greedy function approximation: a gradient boosting machine”, *Annals of statistics*, pp. 1189–1232, 2001.

8. Ribeiro, M. T., S. Singh and C. Guestrin, “” Why should i trust you?” Explain-

ing the predictions of any classifier", *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

9. Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization", *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

10. Rudin, C., C. Chen, Z. Chen, H. Huang, L. Semenova and C. Zhong, "Interpretable machine learning: Fundamental principles and 10 grand challenges", *Statistics Surveys*, Vol. 16, pp. 1–85, 2022.

11. Doshi-Velez, F. and B. Kim, "Towards a rigorous science of interpretable machine learning", *arXiv preprint arXiv:1702.08608*, 2017.

12. Miller, G. A., "The magical number seven, plus or minus two: Some limits on our capacity for processing information.", *Psychological review*, Vol. 63, No. 2, p. 81, 1956.

13. Lage, I., E. Chen, J. He, M. Narayanan, B. Kim, S. J. Gershman and F. Doshi-Velez, "Human evaluation of models built for interpretability", *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, Vol. 7, pp. 59–67, 2019.

14. Hong, S. R., J. Hullman and E. Bertini, "Human factors in model interpretability: Industry practices, challenges, and needs", *Proceedings of the ACM on Human-Computer Interaction*, Vol. 4, No. CSCW1, pp. 1–26, 2020.

15. Plass, J. L., R. Moreno and R. Brünken, “Cognitive load theory”, , 2010.

16. Došilović, F. K., M. Brčić and N. Hlupić, “Explainable artificial intelligence: A survey”, *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 0210–0215, IEEE, 2018.

17. Williams, J. J., J. Kim, A. Rafferty, S. Maldonado, K. Z. Gajos, W. S. Lasecki and N. Heffernan, “Axis: Generating explanations at scale with learnersourcing and machine learning”, *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, pp. 379–388, 2016.

18. Poursabzi-Sangdeh, F., D. G. Goldstein, J. M. Hofman, J. W. Wortman Vaughan and H. Wallach, “Manipulating and measuring model interpretability”, *Proceedings of the 2021 CHI conference on human factors in computing systems*, pp. 1–52, 2021.

19. Guidotti, R., A. Monreale, S. Ruggieri, F. Turini, F. Giannotti and D. Pedreschi, “A survey of methods for explaining black box models”, *ACM computing surveys (CSUR)*, Vol. 51, No. 5, pp. 1–42, 2018.

20. Lim, B. and S. Zohren, “Time-series forecasting with deep learning: a survey”, *Philosophical Transactions of the Royal Society A*, Vol. 379, No. 2194, p. 20200209, 2021.

21. Siami-Namini, S., N. Tavakoli and A. S. Namin, “A comparison of ARIMA and LSTM in forecasting time series”, *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pp. 1394–1401, IEEE, 2018.

22. Quinlan, J. R., “Induction of decision trees”, *Machine learning*, Vol. 1, No. 1, pp. 81–106, 1986.

23. Steinberg, D., “CART: classification and regression trees”, *The top ten algorithms in data mining*, pp. 193–216, Chapman and Hall/CRC, 2009.

24. Tibshirani, R., “Regression shrinkage and selection via the lasso”, *Journal of the Royal Statistical Society: Series B (Methodological)*, Vol. 58, No. 1, pp. 267–288, 1996.

25. Hoerl, A. E. and R. W. Kennard, “Ridge regression: Biased estimation for nonorthogonal problems”, *Technometrics*, Vol. 12, No. 1, pp. 55–67, 1970.

26. Molnar, C., *Interpretable Machine Learning*, 2 edn., 2022, `https://christophm.github.io/interpretable-ml-book`.

27. Nelder, J. A. and R. W. Wedderburn, “Generalized linear models”, *Journal of the Royal Statistical Society: Series A (General)*, Vol. 135, No. 3, pp. 370–384, 1972.

28. Lou, Y., R. Caruana and J. Gehrke, “Intelligible models for classification and regression”, *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 150–158, 2012.

29. Lou, Y., R. Caruana, J. Gehrke and G. Hooker, “Accurate intelligible models with pairwise interactions”, *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 623–631, 2013.

30. Becker, B. and R. Kohavi, “Adult”, UCI Machine Learning Repository, 1996,

DOI: https://doi.org/10.24432/C5XW20.

31. Desboulets, L. D. D., “A review on variable selection in regression analysis”, *Econometrics*, Vol. 6, No. 4, p. 45, 2018.

32. Karagiannopoulos, M., D. Anyfantis, S. Kotsiantis and P. Pintelas, “Feature selection for regression problems”, *Educational Software Development Laboratory, Department of Mathematics, University of Patras, Greece*, 2004.

33. Duch, W., “Filter methods”, *Feature extraction: foundations and applications*, pp. 89–117, Springer, 2006.

34. Pudil, P., J. Novovičová and J. Kittler, “Floating search methods in feature selection”, *Pattern recognition letters*, Vol. 15, No. 11, pp. 1119–1125, 1994.

35. Akaike, H., “Maximum likelihood identification of Gaussian autoregressive moving average models”, *Biometrika*, Vol. 60, No. 2, pp. 255–265, 1973.

36. Kraskov, A., H. Stögbauer and P. Grassberger, “Estimating mutual information”, *Physical review E*, Vol. 69, No. 6, p. 066138, 2004.

37. LightGBM Development Team, “LightGBM: A highly efficient gradient boosting decision tree”, [Computer software]. Available from https://lightgbm.readthedocs.io, 2023.

38. Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree”, *Advances in neural information processing systems*, Vol. 30, 2017.

39. Bergstra, J., R. Bardenet, Y. Bengio and B. Kégl, "Algorithms for hyper-parameter optimization", *Advances in neural information processing systems*, Vol. 24, 2011.

40. Lundberg, S. M., G. G. Erion and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles", *arXiv preprint arXiv:1802.03888*, 2018.

41. Bishop, C. M. and N. M. Nasrabadi, *Pattern recognition and machine learning*, Vol. 4, Springer, 2006.

42. Chapados, N., "Effective Bayesian modeling of groups of related count time series", *International conference on machine learning*, pp. 1395–1403, PMLR, 2014.

43. Consoli, S., M. Negri, A. Tebbifakhr, E. Tosetti and M. Turchi, "Forecasting the IBEX-35 stock index using deep learning and news emotions", *International Conference on Machine Learning, Optimization, and Data Science*, pp. 308–323, Springer, 2021.

44. ter Burg, K. and H. Kaya, "Comparing Approaches for Explaining DNN-Based Facial Expression Classifications", *Algorithms*, Vol. 15, No. 10, p. 367, 2022.

45. Ridgeway, K., "A survey of inductive biases for factorial representation-learning", *arXiv preprint arXiv:1612.05299*, 2016.

46. Brunton, S. L. and J. N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*, Cambridge University Press, 2022.

47. Todo, W., B. Laurent, J.-M. Loubes and M. Selmani, "Dimension Re-

duction for time series with Variational AutoEncoders", *arXiv preprint arXiv:2204.11060*, 2022.

48. Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework", , 2016.

49. Bengio, Y., A. Courville and P. Vincent, "Representation learning: A review and new perspectives", *IEEE transactions on pattern analysis and machine intelligence*, Vol. 35, No. 8, pp. 1798–1828, 2013.

50. Woo, G., C. Liu, D. Sahoo, A. Kumar and S. Hoi, "CoST: Contrastive Learning of Disentangled Seasonal-Trend Representations for Time Series Forecasting", *arXiv preprint arXiv:2202.01575*, 2022.

51. Chen, R. T., X. Li, R. B. Grosse and D. K. Duvenaud, "Isolating sources of disentanglement in variational autoencoders", *Advances in neural information processing systems*, Vol. 31, 2018.

52. Kingma, D. P. and M. Welling, "Auto-encoding variational bayes", *arXiv preprint arXiv:1312.6114*, 2013.

53. Burgess, C. P., I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins and A. Lerchner, "Understanding disentangling in β-VAE", *arXiv preprint arXiv:1804.03599*, 2018.

54. Ates, E., B. Aksar, V. J. Leung and A. K. Coskun, "Counterfactual Explanations for Multivariate Time Series", *2021 International Conference on Applied Artificial Intelligence (ICAPAI)*, pp. 1–8, IEEE, 2021.

55. Wang, Y., R. Emonet, E. Fromont, S. Malinowski, E. Menager, L. Mosser and R. Tavenard, “Learning interpretable shapelets for time series classification through adversarial regularization”, *arXiv preprint arXiv:1906.00917*, 2019.

56. Li, G., B. Choi, J. Xu, S. S. Bhowmick, K.-P. Chun and G. L.-H. Wong, “Efficient shapelet discovery for time series classification”, *IEEE transactions on knowledge and data engineering*, Vol. 34, No. 3, pp. 1149–1163, 2020.

57. Lim, B., S. Ö. Arık, N. Loeff and T. Pfister, “Temporal fusion transformers for interpretable multi-horizon time series forecasting”, *International Journal of Forecasting*, Vol. 37, No. 4, pp. 1748–1764, 2021.

58. Wu, B., L. Wang and Y.-R. Zeng, “Interpretable tourism demand forecasting with temporal fusion transformers amid COVID-19”, *Applied Intelligence*, pp. 1–22, 2022.

59. Gunnarsson, A. and C. Franc, “Food waste reduction through sales forecasting using temporal fusion transformers”, , 2021.

60. Santos, M. L., X. García-Santiago, F. E. Camarero, G. B. Gil, P. C. Ortega *et al.*, “Application of Temporal Fusion Transformer for Day-Ahead PV Power Forecasting”, *Energies*, Vol. 15, No. 14, pp. 1–22, 2022.

61. Zhang, W., C. Zhang and F. Tsung, “Transformer Based Spatial-Temporal Fusion Network for Metro Passenger Flow Forecasting”, *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 1515–1520, IEEE, 2021.

62. Oreshkin, B. N., D. Carpov, N. Chapados and Y. Bengio, “N-BEATS: Neu-

ral basis expansion analysis for interpretable time series forecasting", *arXiv preprint arXiv:1905.10437*, 2019.

63. Jabeur, S. B., S. Mefteh-Wali and J.-L. Viviani, "Forecasting gold price with the XGBoost algorithm and SHAP interaction values", *Annals of Operations Research*, pp. 1–21, 2021.

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

65. Rijksvoorlichtingsdienst, "Schoolvakanties - Rijksoverheid.nl", , 2021, `https://data.overheid.nl/dataset/172c5d6a-6129-4e3c-9691-ab0a5ebefad6`, dataset bestaande uit data met betrekking tot schoolvakanties in Nederland. Beheerd door Rijksoverheid.nl, met gegevens van de aanbieder als de Rijksvoorlichtingsdienst. Bron catalogus: https://data.overheid.nl. Toegang: Publiek. Status van de dataset: Beschikbaar. Licentie: CC-0 (1.0). Koninkrijksdeel Nederland.

66. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, 2011, version 1.2.2.

67. Alexandrov, A., K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen and Y. Wang, "GluonTS: Probabilistic and Neu-

ral Time Series Modeling in Python", *Journal of Machine Learning Research*, Vol. 21, No. 116, pp. 1–6, 2020, http://jmlr.org/papers/v21/19-820.html.

68. Microsoft, "Fsv2-series", https://learn.microsoft.com/en-us/azure/virtual-machines/fsv2-series, n.d., accessed: [12-08-2023].

APPENDIX A: Appendix 1: Full List of Considered Features

Feature 1: day_0: Binary feature that is true on Monday, false otherwise.

Feature 2: day_1: Binary feature that is true on Tuesday, false otherwise.

Feature 3: day_2: Binary feature that is true on Wednesday, false otherwise.

Feature 4: day_3: Binary feature that is true on Thursday, false otherwise.

Feature 5: day_4: Binary feature that is true on Friday, false otherwise.

Feature 6: day_5: Binary feature that is true on Saturday, false otherwise.

Feature 7: day_6: Binary feature that is true on Sunday, false otherwise.

Feature 8: Weekday: Binary feature that is true during weekdays, false otherwise.

Feature 9: national_holiday: Binary feature indicating whether it is a national holiday or not.

Feature 10: origin_school_vacation: Binary feature that is true is there if a Dutch school vacation at the origin station.

Feature 11: destination_school_vacation: Binary feature that is true is there if a Dutch school vacation at the destination station.

Feature 12: total_travelers_7: total number of travelers at the same day of the week, 7 days ago.

$$\sum_{t=1}^{T} y_{d-7,t} \tag{A.1}$$

Feature 13: recent_trend: Change in total number of travelers between the last similar day, and 7 days earlier. Where the last similar day d^* is the closest day that has the same value for the weekday feature.

$$\sum_{t=1}^{T} y_{d^*,t} - \sum_{t=1}^{T} y_{d^*-7,t} \tag{A.2}$$

Feature 14: moving_average_4w: Moving average of the total number of travelers at that day of the week, calculated over the past 4 weeks.

$$\sum_{w=1}^{4} \sum_{t=1}^{T} \frac{y_{d-7w,t}}{4} \tag{A.3}$$

Feature 15: moving_average_6w: Moving average of the total number of travelers at that day of the week, calculated over the past 6 weeks.

$$\sum_{w=1}^{6} \sum_{t=1}^{T} \frac{y_{d-7w,t}}{6} \tag{A.4}$$

Feature 16: total_travelers00_06): Total number of travelers between midnight and 6 AM, 7 days ago.

$$\sum_{t=1}^{12} y_{d-7,t} \tag{A.5}$$

Feature 17: total_travelers_06_10: Total number of travelers between 6 AM and 10 AM, 7 days ago.

$$\sum_{t=13}^{20} y_{d-7,t} \tag{A.6}$$

Feature 18: total_travelers_10_16: Total number of travelers between 10 AM and

4 PM, 7 days ago.

$$\sum_{t=21}^{32} y_{d-7,t} \tag{A.7}$$

Feature 19: total_travelers_16_20: Total number of travelers between 4 PM and 8 PM, 7 days ago.

$$\sum_{t=33}^{40} y_{d-7,t} \tag{A.8}$$

Feature 20: total_travelers_20_24: Total number of travelers between 8 PM and midnight, 7 days ago.

$$\sum_{t=41}^{48} y_{d-7,t} \tag{A.9}$$

Feature 21: morning_peak_travelers: Maximum number of travelers at a single timepoint during morning rush hours.

$$\max_{t\in[13,20]} y_{d-7,t} \tag{A.10}$$

Feature 22: evening_peak_travelers: Maximum number of travelers at a single timepoint during evening rush hours.

$$\max_{t\in[33,40]} y_{d-7,t} \tag{A.11}$$

Feature 23: morning_peakiness: Logarithm base 2 of the ratio of (total travelers between 6 AM and 10 AM, 7 days ago + 1) to (morning peak travelers + 1).

$$\log_2\left(\frac{(\text{feature } 17) + 1}{(\text{Feature } 21) + 1}\right) \tag{A.12}$$

Feature 24: evening_peakiness: Logarithm base 2 of the ratio of (total travelers between 4 PM and 8 PM, 7 days ago + 1) to (evening peak travelers + 1).

$$\log_2\left(\frac{(\text{feature } 19) + 1}{(\text{Feature } 22) + 1}\right) \tag{A.13}$$

Feature 25: afternoon_dip: Minimum number of travelers between 7:30 and and 16:30, 7 days ago.

$$\min_{t\in[16,34]} y_{d-7,t} \tag{A.14}$$

Feature 26: relative_dip: Ratio of the afternoon dip (Afternoon Dip) to the sum of morning peak travelers and evening peak travelers, plus 1.

$$\frac{(\text{Feature } 25) + 1}{(\text{Feature } 21) + (\text{Feature } 22) + 1} \tag{A.15}$$

Feature 27: log_outlier_1day: Logarithm base 2 of the actual number of travelers yesterday divided by the 6 week moving average of yesterday's total travelers.

$$\log_2\left(\frac{\sum_{t=1}^{T} y_{d-1,t} + 1}{(\sum_{w=0}^{5}\sum_{t=1}^{T}\frac{y_{d-1-7w,t}}{6}) + 1}\right) \tag{A.16}$$

Feature 28: log_outlier_7day: Logarithm base 2 of the actual number of travelers on the same day of the week last week divided by the moving average of total travelers on the same day of the week in the previous 6 weeks.

$$\log_2\left(\frac{(\text{Feature 12})+1}{(\text{Feature 15})+1}\right) \tag{A.17}$$

Feature 29: morning_peak_trend: slope of regression line fitted through last three peak morning rush hours (7, 14, and 21 days ago)

Feature 30: evening_peak_trend: slope of regression line fitted through last three peak evening rush hours (7, 14, and 21 days ago)

Feature 31: morning_peak_trend_7: the difference between peak morning rush hour last week, and the week before.

$$\text{Feature 21} - \max_{t\in[13,20]} y_{d-14,t} \tag{A.18}$$

Feature 32: evening_peak_trend_7: the difference between peak evening rush hour last week, and the week before.

$$\text{Feature 22} - \max_{t\in[33,40]} y_{d-14,t} \tag{A.19}$$

Feature 33: days_since_start: Number of days since the start of the time series.

$$d \tag{A.20}$$

Feature 34-44: 4 Week Moving Average versions of features 16-26.

Additional Features: moving_lw_1-48: The moving average features that we compressed by the dimensionality reduction methods, and used without compression by the no_comp models.

$$\text{moving\_lw\_t} = \sum_{w=1}^{4} \frac{y_{d-7w,t}}{4} \tag{A.21}$$

APPENDIX B: Appendix 2: Additional Figures

In this appendix we to provide some further illustrations and insights.

Figure B.1 shows the feature importance as calculated using Mutual Information, Since MI calculates univariate relationships between features and target variables, we find some very different results when compared to other methods.

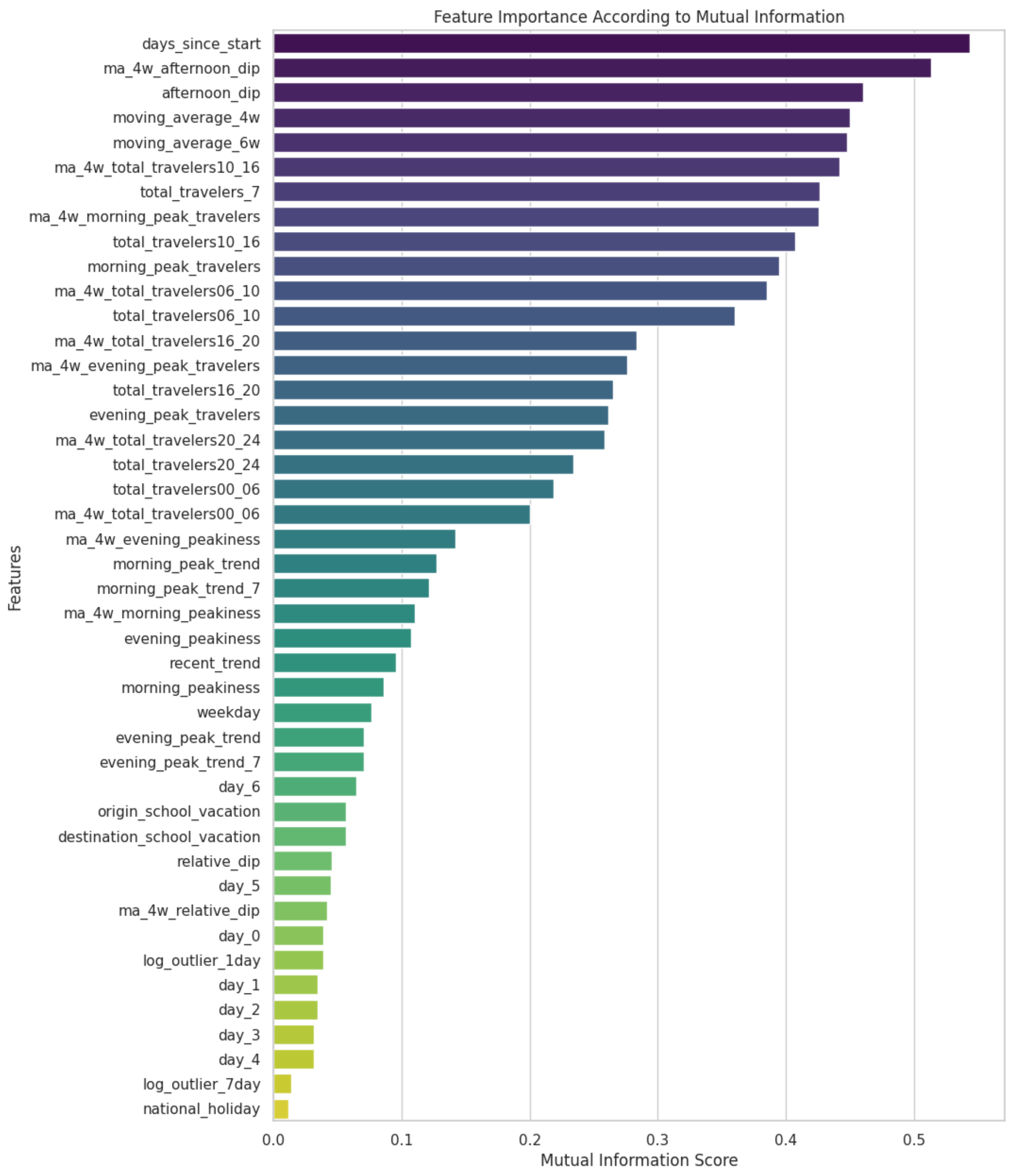


Figure B.1: Feature Importance through Mutual Information: Analyzing the shared information between the target and selected features, based on the average of all data points in the training set.

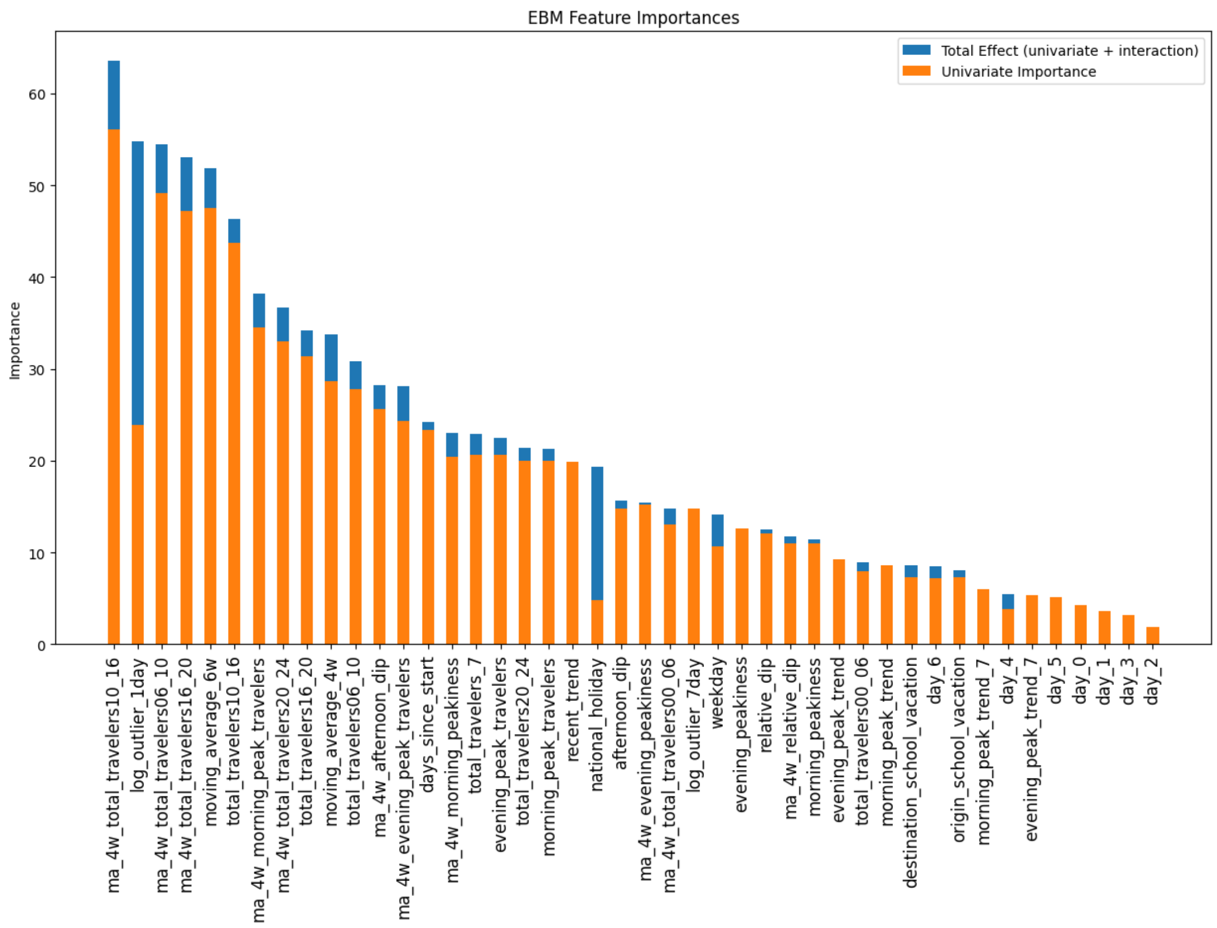


Figure B.2: EBM Global Feature Importance. We see the feature importance of all features in the EBM feature set, ranked from most important on the left to least important on the right. The orange bars indicate univariate feature importance, the effect of the feature through its univariate additive function. The global feature effects of the pairwise functions, where the values of two features are used,are shown in blue. Note the large pairwise feature effects that are attributed to the outlier_1_day and national_holiday features. These features are most relevant in interaction with other features.

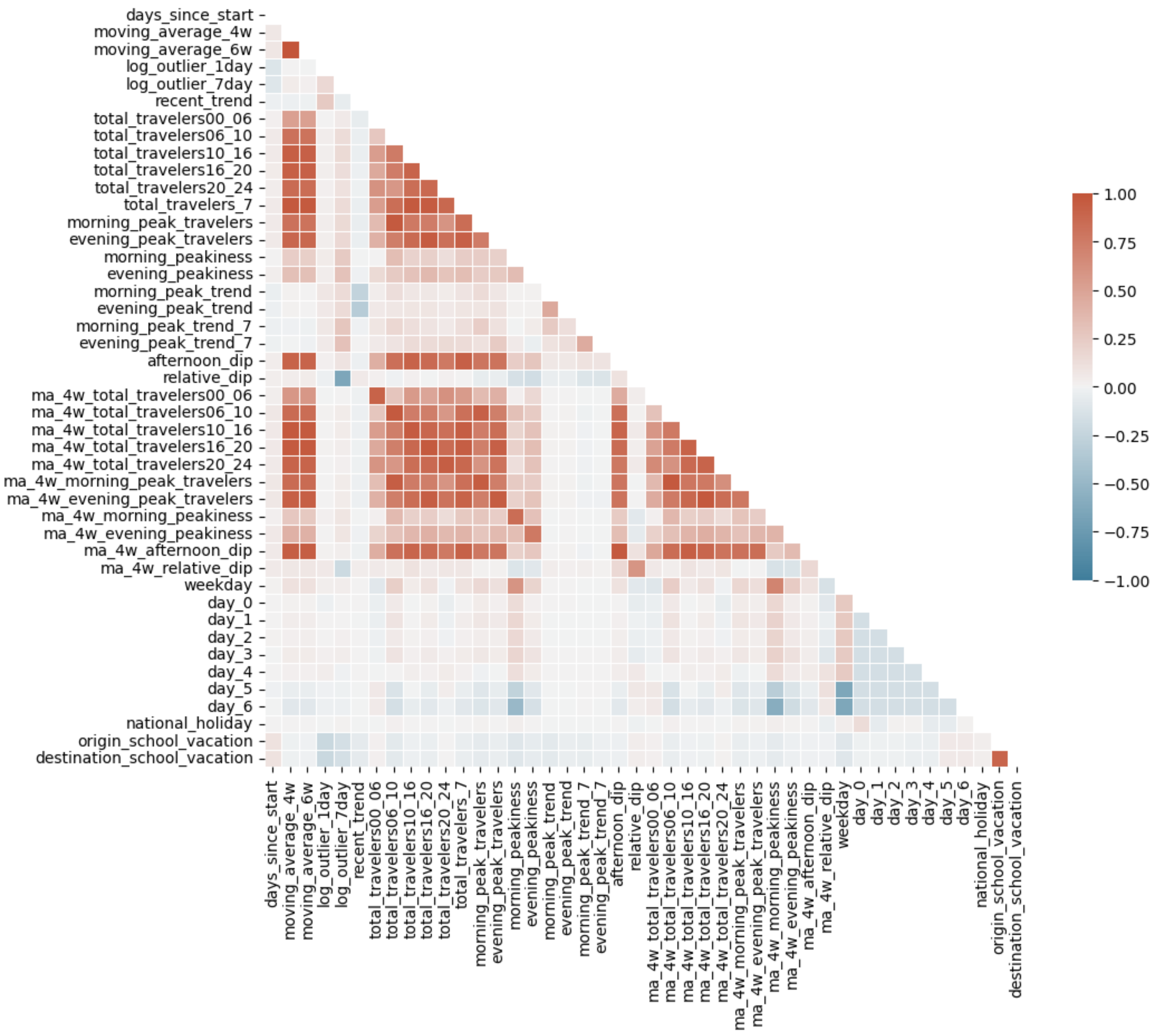


Figure B.3: Pearson correlation between features. Where a red color indicates a positive correlation, and blue a negative correlation. Brighter colors indicate a stronger correlation. Note the high correlation between features and their moving average versions.

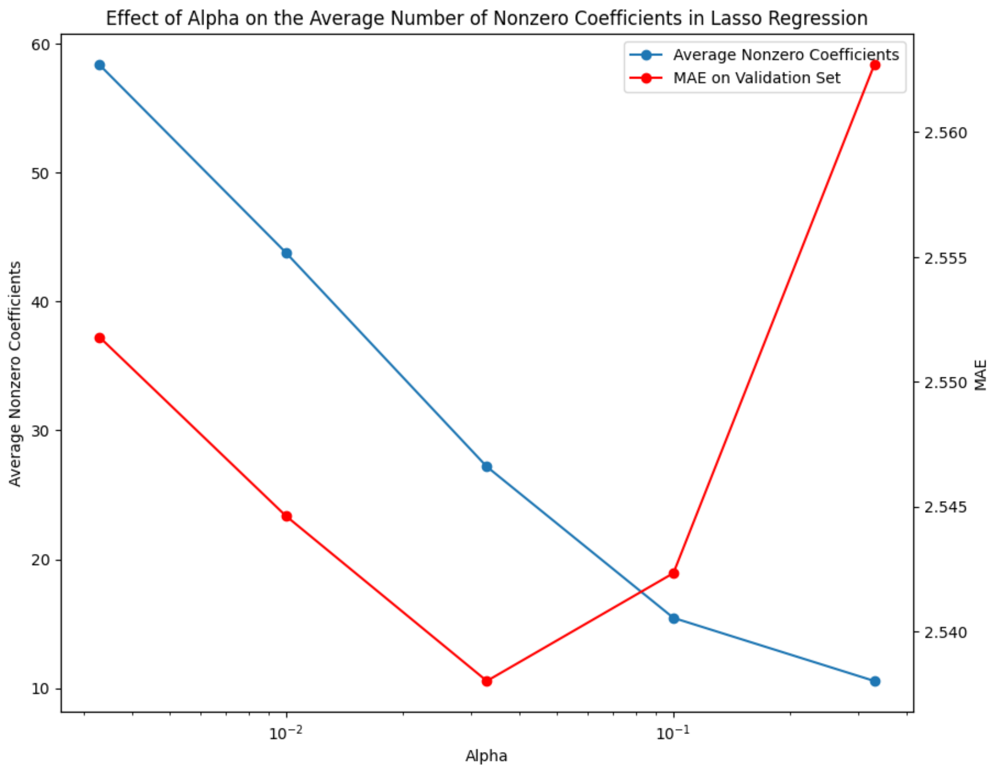


Figure B.4: Plot used to decide on an α value for the interpretable LASSO model discussed in 4.6. The horizontal axis shows the α values on a log scale. The red line shows the MAE of the LASSO model at the different α values, the MAE can be seen on the right hand side. The blue line shows the average number of nonzero coefficients at various α values

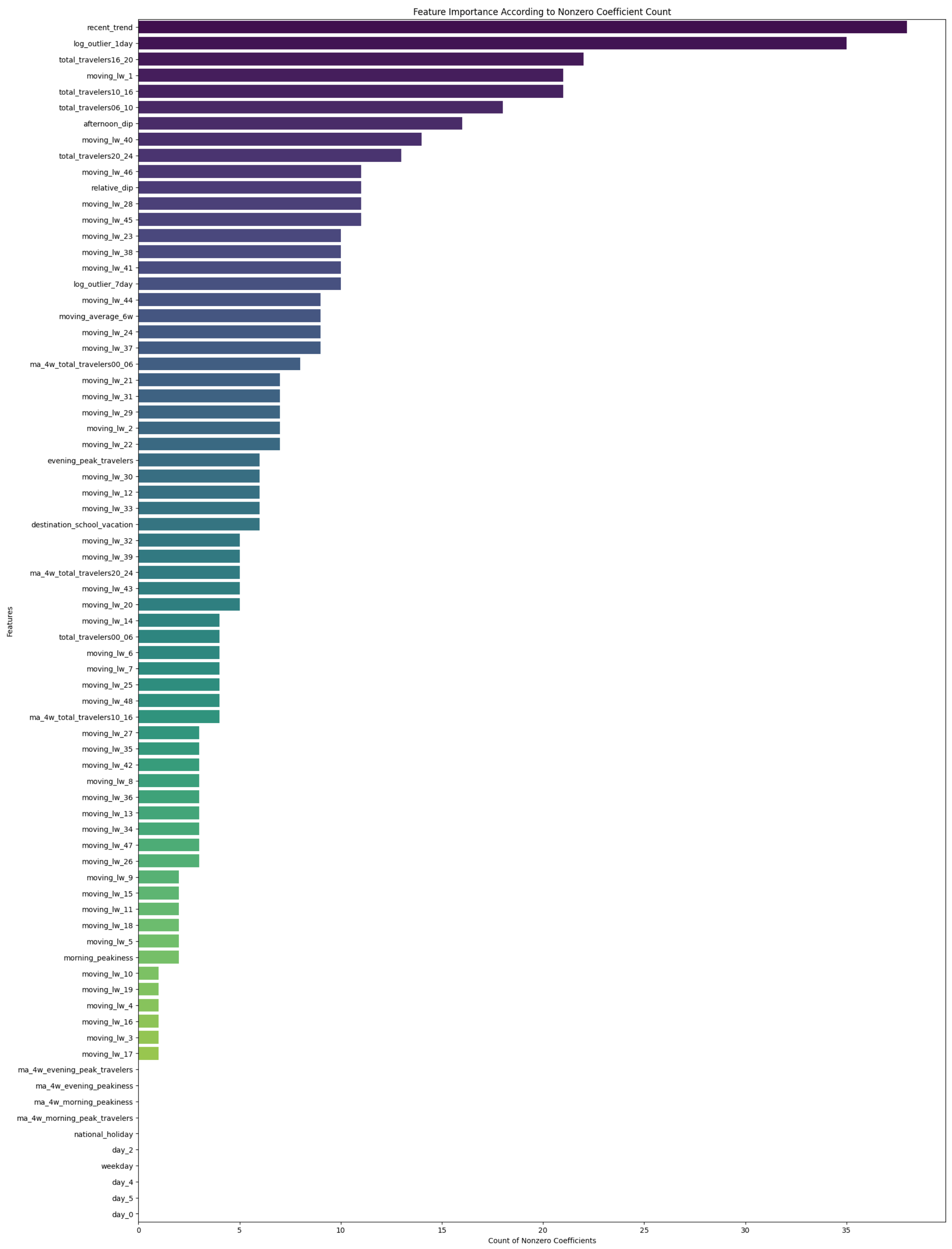


Figure B.5: Count of the number of nonzero coefficients corresponding to the features, across the 48 LASSO models in the LASSO multi-output model. Some features such as day_4 and national_holiday are selected by none of the models, the LASSO cost outweighing the potential accuracy increase. The moving averages of some time slots, such as moving_lw_1, and moving_lw_40 impact the predictions of many models, whereas the moving averages of other time slots have a less widespread influence.

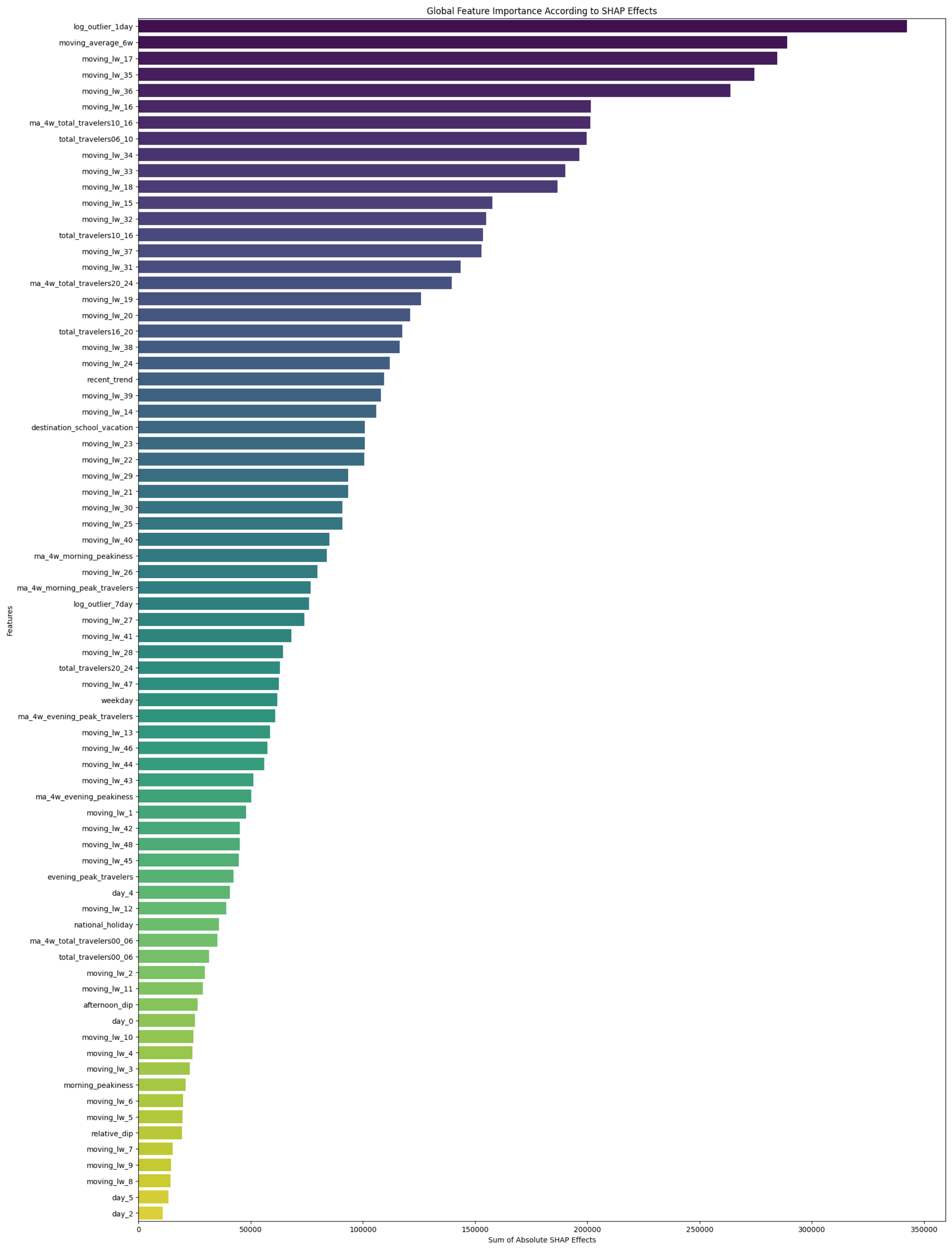


Figure B.6: Global feature importance as the sum of absolute SHAP feature effects, calculated as discussed in 2.6.1 over the first 7 days in the test set. It is important to note that no instance-wise normalization was applied, meaning that larger effects, such as those in instances on busy OD-pairs and busy time slots are represented in proportion to their influence in the final calculated feature significances. The length of the bar indicates the global feature importance according to TreeSHAP. The feature with the biggest effect was the log_outlier_1day feature, followed by the moving_average_6w feature.

APPENDIX C: Additional Background

C.1. Lagged Autocorrelation

We have a univariate time series $\mathbf{Y} \in \mathbb{R}^{D \times T}$ where D represents the total number of days and T represents the number of time points within each day. The value $y_{d,t}$ represents the target value for day d at time point t.

The aim is to investigate the temporal dependencies in the data by constructing a measure of autocorrelation between the value at a given time point and the value at the same time point from previous days. This is done for each OD-pair od, time point t, and day lag L from 1 to 30 days. This measure of autocorrelation, $\rho_{t,L}^{(od)}$, is given by:

$$\rho_{t,L} = \text{Corr}\left(y_{.,t}, y_{.-L,t}\right), \tag{C.1}$$

using Pearson correlation. Here, $y_{.,t}$ is the vector of values at time t across all days $d \in D$ and $d > L$. Conversely, $y_{.-L,t}$ is the same but shifted by L days.

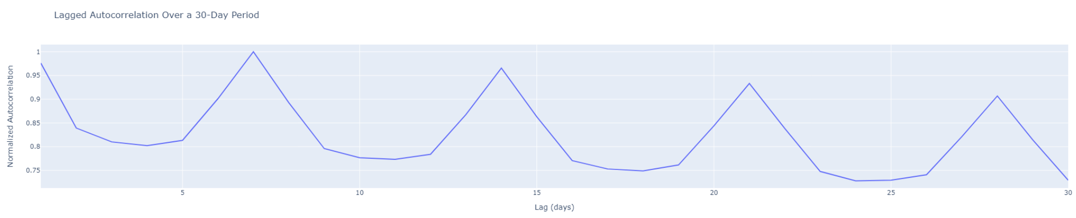


Figure C.1: Visualization of the normalized aggregated lagged autocorrelations over 2,000 distinct time series, following Equation C.3. The y-axis value of 1 indicates the day with the highest lagged autocorrelation, after normalization.

Subsequently, the normalized autocorrelation $\widetilde{\rho}_{t,L}$ is calculated for each time point t:

$$\widetilde{\rho}_{t,L} = \frac{\rho_{t,L}}{\max\ _{L\in[1,30]}\left(|\rho_{t,L}|\right)}. \tag{C.2}$$

This ensures that the correlations $\widetilde{\rho}_{t,L}$ are within a [-1,1] range for each time point t, such that all time points are equally considered. Lastly, the normalized autocorrelation values $\widetilde{\rho}_{t,L}$ for all time points t are summed together:

$$\rho_L = \sum_{t=1}^{T} \widetilde{\rho}_{t,L} \tag{C.3}$$

and then normalized to a range of [-1,1], to get the final autocorrelation per day lag, $\hat{\widetilde{\rho}}_L$:

$$\widetilde{\rho}_L = \frac{\rho_L}{\max\left(|\rho_L|\right)}. \tag{C.4}$$

Such that for each OD, we get the autocorrelation per day L: $\widetilde{\rho}_L$, where the day with the highest autocorrelation has a value of 1, or -1. Finally, we aggregate these autocorrelations for all OD pairs and normalize once again to get values in the range of $[-1, 1]$.

C.2. Feature Selection

To select a comprehensive set of features that are intelligible and lead to good model performance we take a multifaceted approach: We use wrapper methods as discussed in 2.4. A full list of features that were considered for feature selection can be viewed in Appendix A.

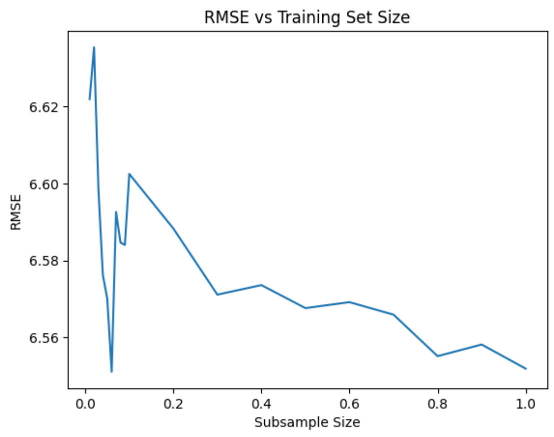


Figure C.2: Shows the relationship between the subsample of the training set used to train the linear regression model training and RMSE on the validation set. Differences in RMSE are very minor, the larger shifts in performance at low subsample values may be due to random effects.

For penalty based feature selection we used LASSO with a stronger penalty term alpha = 0.1

Since wrapper methods are computationally expensive, we are restricted to using linear regression and LGBM. To further reduce the computational demands, we explored the relation between model performance on the validation set and the size of the train set by randomly sampling a fraction of the train set to fit LR. All features were used in this stage. As shown in fig C.2, using more data only led to a very minor improvement in the case of linear regression. Therefore we decided to use a fraction of 0.05 of the full training set to perform feature selection using Forward Selection with Sequential Floating Search.

For Forward Selection (FS) with Sequential Floating Search (FSF) with LGBM, we tuned the hyperparameters on the feature set that resulted from the FS with SFS on the LR model.

C.3. LGBM hyperparameters

Hyperparameter tuning was performed for each LGBM model that received a unique feature set:

F. Set	**M. Depth**	**N. Leaves**	**L. Rate**	**N. Est.**	**M.C. Samples**	**M.C. Weight**	**F. Fraction**
Handcrafted	16	268	0.02159	411	41	6.68e-06	0.5
+ β-VAE 6	10	399	0.04380	438	108	1.56e-04	0.4
+ β-VAE 8	11	281	0.03366	448	122	7.84e-05	0.5
+ SVD 6	17	374	0.02218	342	106	4.90e-08	0.5
+ SVD 8	17	367	0.02267	488	95	1.26e-07	0.4
+ SVD 10	17	320	0.02299	470	145	5.69e-07	0.5
+ No compression	15	424	0.02107	462	175	3.18e-05	0.5

Table C.1: Hyperparameters of LGBM Models for Different Feature Sets. F. Set denotes the feature set used for the model, M. Depth is the maximum depth of the trees, N. Leaves is the number of leaves in the trees, L. Rate is the learning rate, N. Est. is the number of estimators, M.C. Samples is the minimum number of child samples, M.C. Weight is the minimum child weight, and F. Fraction is the fraction of features used. *Handcrafted* is the original optimal feature set found through FS with SFS on the complete set of handcrafted features. The subsequent feature sets contain the addition of the representations generated by the dimensionality reduction methods.

C.4. Selection of Latent Variables in Representation Learning

To select the optimal number of components for the SVD dimensionality reduction features, we iterated from 1 to 20 top-k components and trained LR and LGBM models on the train set. So these models receive 1-20 additional features besides their optimal classical feature set. Performance was examined on the validation set:

Visualization of the Vh matrix truncated at 12 top components shows that

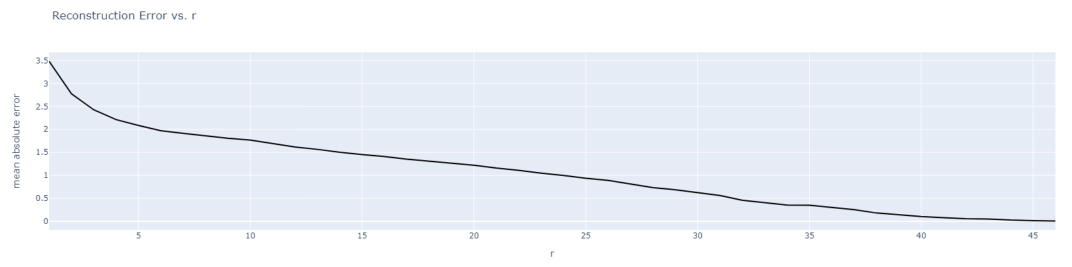


Figure C.3: Mean absolute error of time series reconstruction using SVD's top k components (the r is a typo). The horizontal axis shows the number of components at which the SVD representation was truncated

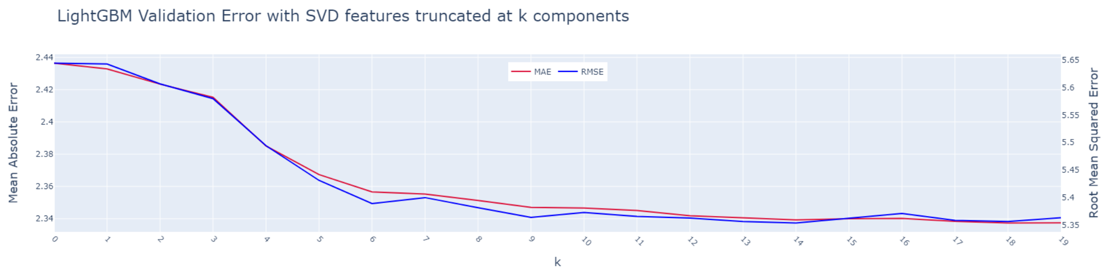


Figure C.4: Accuracy measurements on validation of LGBM with lgbm_sfs_features feature set, enhanced with top k components from SVD dimensionality reduction. For our results section, we used SVD representation truncated at the top 6 components since subsequent components were increasingly harder to interpret.

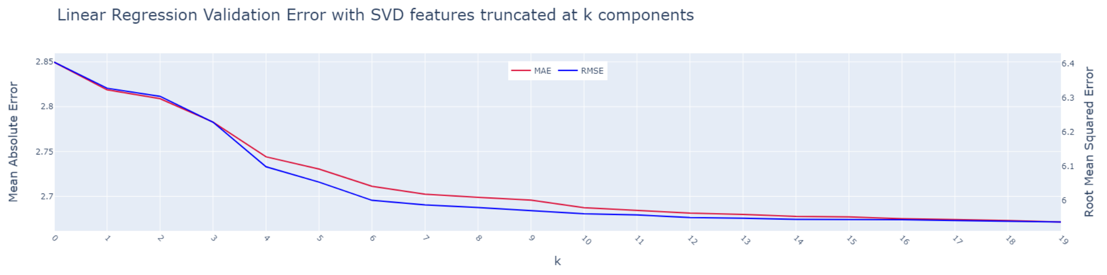


Figure C.5: Accuracy measurements on validation of Linear Regression with lr_sfs_features feature set, enhanced with top k components from SVD dimensionality reduction. For our results section, we used SVD representation truncated at the top 6 components since subsequent components were increasingly harder to interpret.

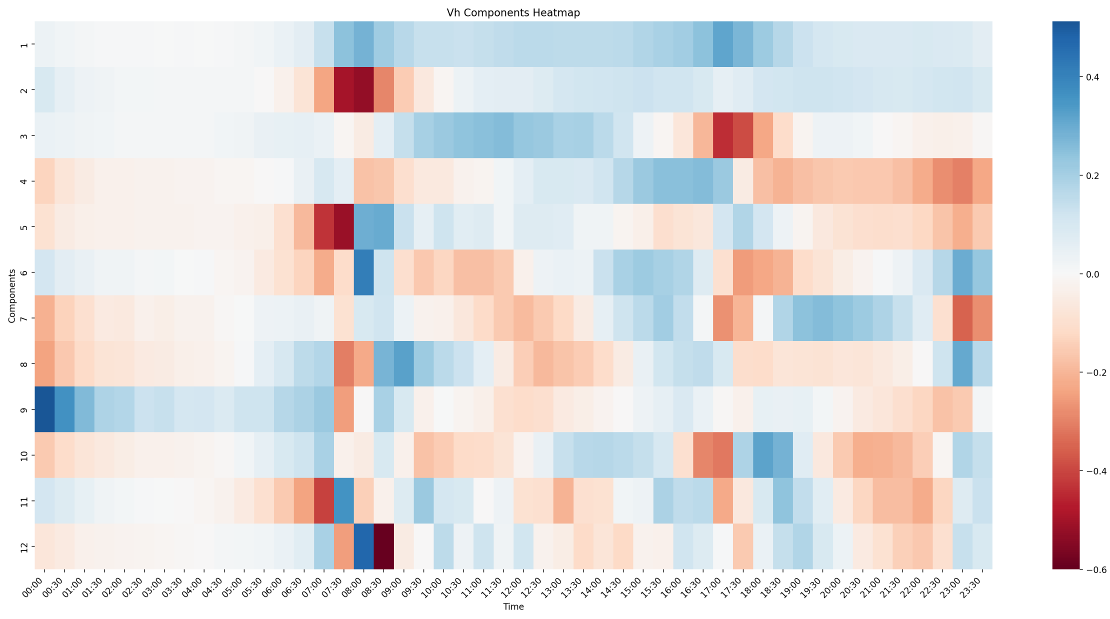


Figure C.6: Alternative visualization of the V matrix truncated at 12 top components. The start of the time slot is shown on the x-axis, the y-axis indicates the ranking of the components' importance, with 1 being the highest order component. Colors variations represent the magnitude of the respective element in the Vh matrix. Note that the polarity of these values flips for the entire row if the corresponding u value has a negative sign.

the lower order (higher index) components are more erratic and capture smaller variations in the data

C.5. β-VAE model selection

To inspect the relationship between β and reconstruction quality, four base β-VAE models with 6 latent variables were trained, with β values $(0, 1\text{e-}5, 1\text{e-}4, 1\text{e-}3)$. The reconstruction error on the validation set can be seen in Figure C.7. Manual inspection of the latent variables through single latent traversals showed that only the model with $\beta = 1e - 4$ created intelligible latent variables.

Six new β-VAE models were trained on the train set, with latent dimensions $|\mathbf{z}| \in (6, 8, 10)$ and β values 3e-5 and 1e-4. Reconstruction quality was assessed on the validation set, as shown in Figure C.8. These 6 models were used to represent the moving average of the same-day last 4 weeks, identical to the SVD method.

To assess which of these models generate the most informative features, we added the latent representations to the optimal handcrafted feature sets for LR and LGBM.

These representations were used as additional features, LGBM and LR models were trained on the test set and forecasting accuracy was assessed on the validation set. All six representations resulted in comparable performance increases, as can be seen in Figures C.9, C.10

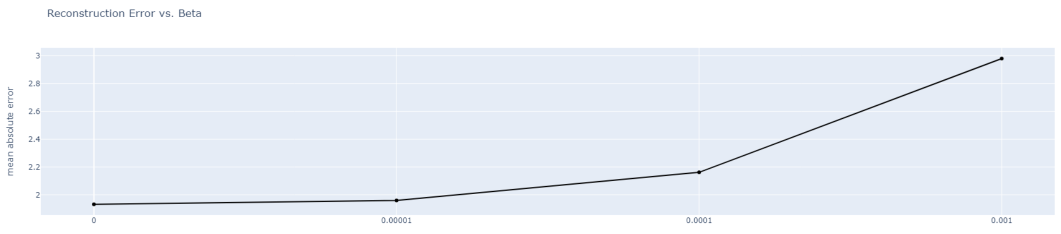


Figure C.7: Reconstruction accuracy of $\beta - VAE$ models with 6 latent units, at different values of β. Values on the x-axis indicate the β value that was used. Larger β values result in a worsening of the reconstruction quality.

C.5.1. Results For All Models

The results of all final models on the test set can be seen in Figure C.11, which shows that compression of historical data through SVD is superior to compression through β-VAE, but both methods are inferior to using no compression at all. In the case of using LGBM, there is less of a distinction between SVD and β-

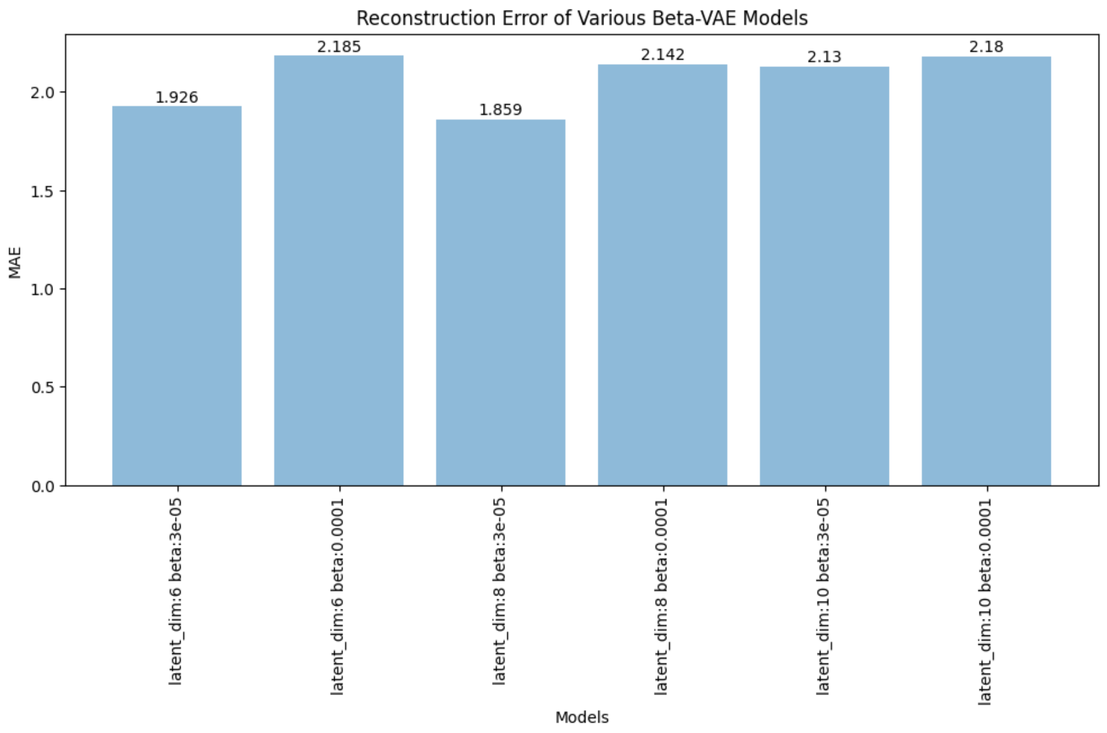


Figure C.8: Reconstruction accuracy of $\beta - VAE$ models. The name of the bar indicates the hyperparameters used to develop the β-VAE model. The two rightmost models, with a latent dimension of 10, fail to continue the expected trend of a reducing reconstruction error when increasing the information bottleneck

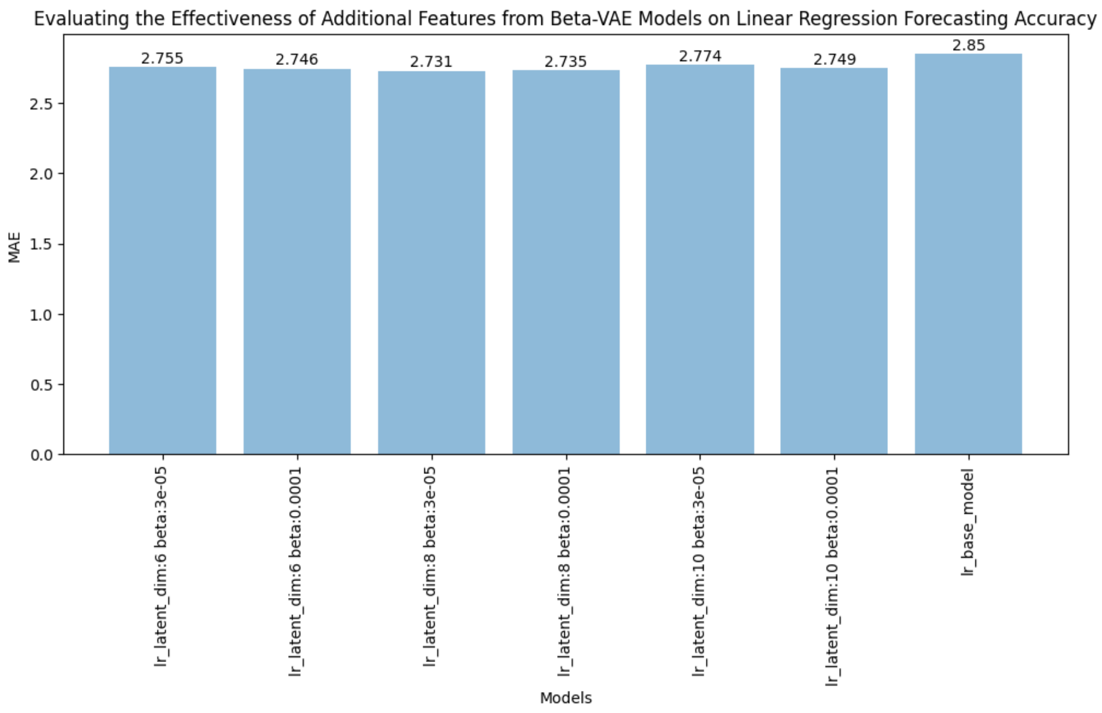


Figure C.9: Forecasting accuracy on the validation set of LR models that received latent representations generated through β-VAE models. The name of the bar indicates the hyperparameters used to develop the β-VAE model that generated the additional features.

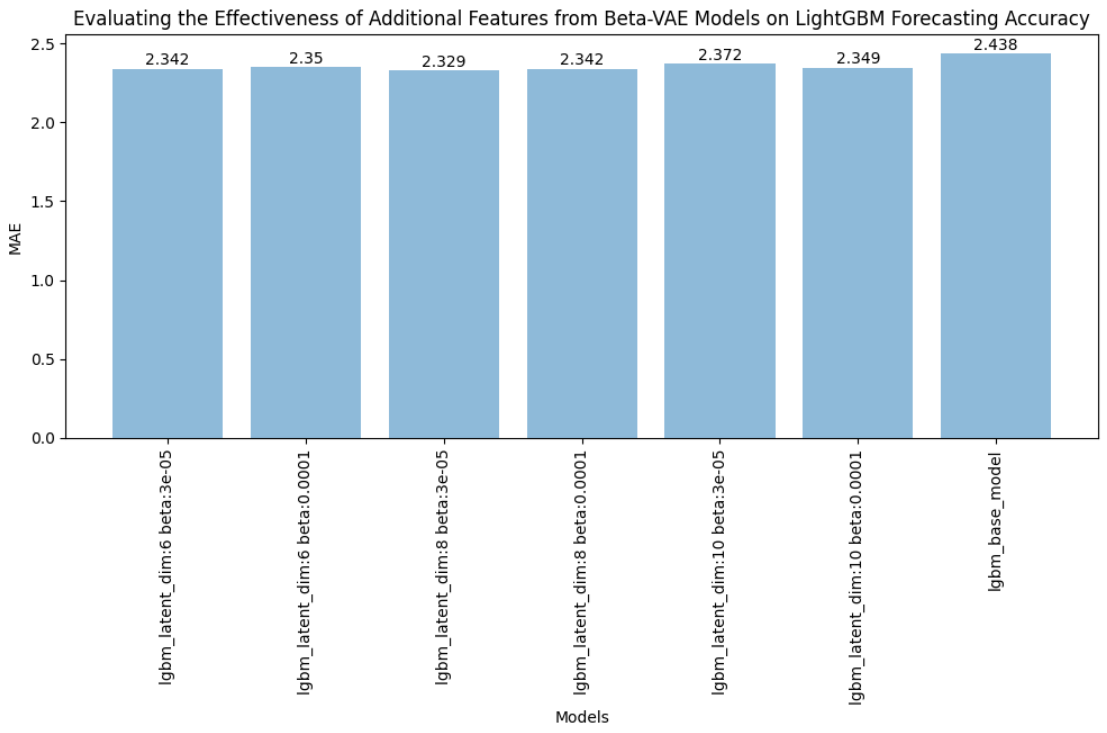


Figure C.10: Forecasting accuracy on the validation set of LGBM models that received latent representations generated through β-VAE models. The name of the bar indicates the hyperparameters used to develop the β-VAE model that generated the additional features.

VAE compression, but again, no compression is superior. We find that all LGBM models, except the base LGBM model, are better than DeepAR, The LR and Lasso models without dimensionality reduction are not significantly different from the DeepAR model.

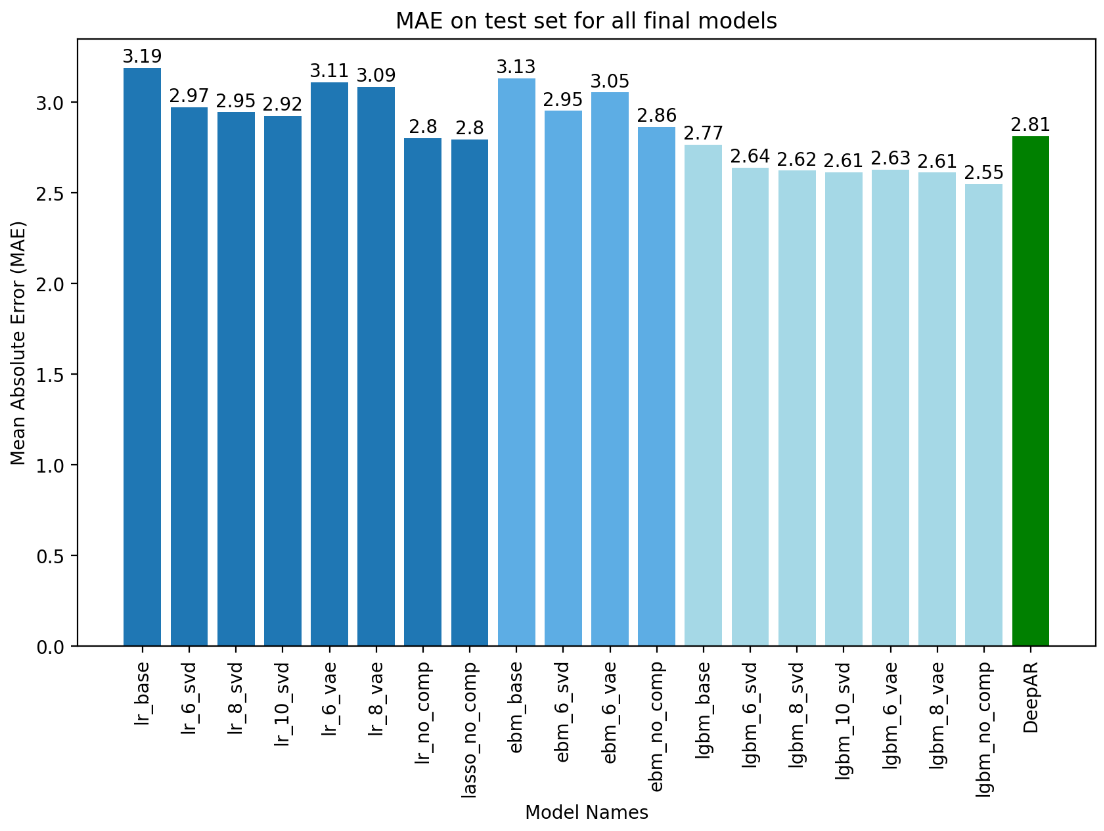


Figure C.11: Mean Absolute Error of all models on the test set. LR models are shown in dark blue, EBM models in regular blue, and LGBM models in light blue. The numbers indicate the size of the latent dimension used. For example, lr_6_svd is the LR model that has the base LR features plus the 6 most important SVD U values. When the model name ends with no_comp, that means the historical data was included in uncompressed form.