FACULTY OF ECONOMICS AND
BUSINESS ADMINISTRATION

InClass Kaggle Competition: Hotel Room Price prediction

Technical report

Course: Machine Learning

Teacher:	Prof. Dr. Dries Benoit
	Lukas De Kerpel (TA)

Students:	Team 10		
	Artur Tyvaert	01706871	artur.tyvaert@ugent.be
	Simon De Lange	01907811	simondla.delange@ugent.be
	Stijn Van Ruymbeke	01904796	stijn.vanruymbeke@ugent.be
	Viktor Vandenbulcke	01803499	viktor.vandenbulcke@ugent.be

Academic year: 2022–2023

GHENT
UNIVERSITY

Contents

1 Introduction

The low-margin profit structure of the hospitality industry requires hotel managers to maximize profits by setting prices at an optimal (dynamic) level. Therefore, the authors create a high-performing hotel room rate prediction tool to facilitate this difficult process for the managers.

In this paper, the authors elaborate on how they were able to construct high-performing hotel room rate prediction models, which models were tested, and which models are to be preferred.

During the entire process, the guidelines of the CRISP-DM framework (figure 1) were followed to optimize the quality of the work.

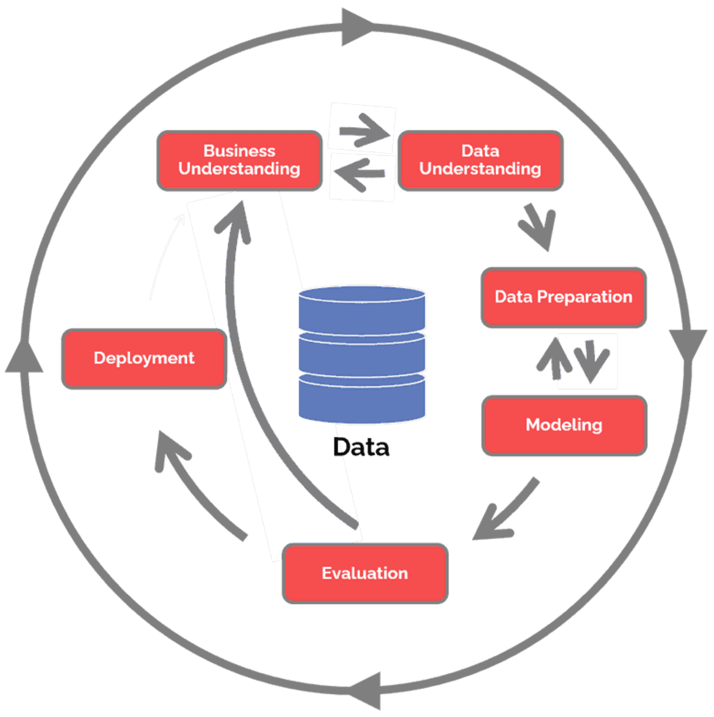

Figure 1: CRISP-DM framework

2 Data Preprocessing

The authors use a layered data approach. In the data exploration phase, the raw (Bronze) data is used. In the data cleaning step, the Bronze data is cleaned into the Silver data. Lastly, feature engineering is done on the Silver data to create the Gold data on which the models will be trained.

2.1 Data Exploration

Following the CRISP-DM framework, the first step in creating business value is to understand and gain insights in the raw data. To be able to get valuable insights into the variables, the authors created different visuals for each individual variable, depending on whether the variable was categorical or numerical. Moreover, they also looked at the different values of each variable and its univariate summary statistics. All these steps and comments per variable can be found in the R-script 'Data_exploration.R'.

2.2 Data Cleaning

Before cleaning the data, the data was split into a training and validation set to prevent data leakage (figure 2). A 80/20 split of the data was used.

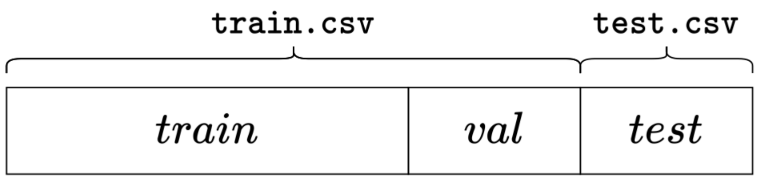

Figure 2: Data split

Next, the bronze data was cleaned. First, integer values were extracted from lead_time. Then, the percentage of missing values per column were detected and imputed. However, some variables also had Null or 'n/a' values that were not detected by colMeans and were handled manually. Missing values for some variables (e.g. nr_babies, nr_booking_changes) were considered to be zero as a 'n/a' value for babies indicates that there are no babies. Consequently, these values are not considered missing. Next, the detected missing values were imputed for each dataset with the mode (categorical) or the median (numerical) computed on the training set to avoid data leakage. As missing values could be informative, a flag function that indicates whether or not an observation had missing values for a certain variable was created.

Furthermore, the outliers in the training data were handled. First, the authors looked at each variable individually with the quantile function to detect valid and invalid outliers. Invalid outliers were handled as a missing value, while valid outliers were brought back to a |z-score| of 3. Handling outliers was important as normalization was performed hereafter.

Lastly, the dates in the different datasets were parsed and extracted into time variables such as year and month. Besides, the missing values of the variable last_status_date were also imputed. More details about this can be found in the R-script 'data_cleaning.R'.

2.3 Feature Engineering

Feature engineering is the process of designing and creating new features from raw data that can improve the performance of models.

First, the authors created some additional features that could improve the model performance. The authors created "room_type_conflict" which equals 1 if the assigned room type isn't equal to the reserved room type. The column "assigned_room_type" was deleted due to the high correlation with "reserved_room_type". More columns were deleted for this reason. Other features that were created by the authors:

- week_of_month: indicates in which part of the month the arrival date is located. The month was split in 4-5 parts, always splitting after the seventh day.
- time_between_arrival_checkout and time_between_arrival_cancel: difference in days between arrival and checkout (positive) or cancellation (negative).
- nr_weekdays and nr_weekenddays: decomposes "nr_nights" in weekdays and weekend days.

Then, dummy variables were created for all categorical variables using the dummy package for flexibility. As some categorical variables had high cardinality, this was limited to 10 for most of these variables, except for the variables listed in Table 1.

Variable name	Cardinality	Reasoning
month_arrival	12	Seasonality
booking_agent	8	7 booking agents and 'n/a' all have frequency > 1000
booking_company	2	TA expert recommendation
country	15	TA expert recommendation

Table 1: Cardinality exceptions

Next, two indicator variables were created. One for number of babies and one for number of children. For both these variables, more than 90% of the values were zero and there was only few variation in the remaining values. Therefore, it was decided that most information in these variables would be captured by indicator variables.

Further, the variables "lead_time", "days_in_waiting_list", "time_between_arrival_checkout" were log transformed to make the distribution less skewed and reduce the range of their values. Furthermore, all numerical features were normalized as they did not approximate a normal distribution. As a final step, the correlation between the different variables was checked. For variables with a correlation above 80%, the variables that contained the most information were retained.

An overview of deleted and created features can be found in Appendix C. More details on how the feature engineering was done in practice can be found in the R-script 'Feature_engineering'.

3 Modelling

3.1 Structure Of the Models

To be able to train and compare the performance of the models in an efficient way, each model was trained on the training set. Then, the model was evaluated by calculating the RMSE on the validation set. Based on these RMSEs, the best performing model(s) of each category were selected and then trained on all the data (train and validation set). Afterwards, these were used to make predictions on the test set. The computed validation set RMSE for the different models can be found in Appendix A.

3.2 Linear Models

Linear models assume a linear relation between the independent variables and dependent variable. These models are easy to understand and implement, but are often an oversimplification of reality. Therefore, they serve as a baseline for more advanced ones.

3.2.1 Linear Regression

A linear model was used to regress average_daily_rate on all other variables of the gold data.

3.2.2 Stepwise Selection

As there are a lot of predictors, best subset selection is unfeasible. Stepwise selection provides a feasible alternative. While training these models, the parameter nvmax was set equal to ncol(train_X) = 102, hence, including all the variables from the basetable.

First, forward stepwise selection was performed. This is a method for selecting the optimal subset of features for model building by adding one feature at a time, based on the significance of the improvement. The optimal number of features in this case was determined to be 95. Second, backward stepwise selection was performed. This method begins with a model that includes all features and removes features one at a time, starting with the least significant feature. Again, the optimal number of parameters was 95. Next, sequential replacement stepwise selection combines these two methods as it selects the optimal number features for building a model by adding and removing features. Here, the optimal number of parameters was 99.

3.2.3 Shrinkage Methods

These methods constrain or shrink the coefficient estimates, which reduces their variance. To control the shrinking, a hyperparameter (λ) and a penalty term are added to the minimization problem. First, a ridge regression model was trained. The optimal λ was found by performing 10-fold cross validation and this optimal λ was used to train the ridge regression model. The same procedure was used to train a lasso regression model.

3.3 Non-Linear Models

Following models all relax the linearity assumption. This holds improvements over the preceding models in term of predictive capabilities, at the cost of explainability.

3.3.1 Non-Linear Transformations

In this section, non-linear transformations are performed on numerical variables.

3.3.1.1 Polynomial Regression

Polynomial regression is a regression analysis technique in which polynomials are used to predict the dependent variable. Models that contained all features from the basetable as independent variables were constructed and these were augmented with polynomial functions (up to the fourth degree) of the two numerical variables 'lead_time' and 'time_between_arrival_cancel'. Anova tables were used to identify the optimal model. After analyzing the anova table, it was found that the model with polynomial terms to the third degree performed best.

3.3.1.2 Splines

In essence, splines are piecewise polynomials with smoothness conditions. Data is split into sections, defined by the knots, and polynomials are fitted to each section. Because it's difficult to define these knots for this many variables, degrees of freedom were used as parameter. Degrees of freedom is a measure for flexibility of the spline. Models were created that contained all features as independent variables, augmented with splines of the numerical variables with degrees of freedom of 4, 5 and 6. To find the optimal model, these models were trained and compared in an anova table. The model containing splines with 5 degrees of freedom performed best.

3.3.1.3 Generalized Additive Models

Generalized additive models (GAMs) extend the standard linear model by allowing nonlinear functions for each variable, while still maintaining additivity. The authors constructed a GAM that contained all features as independent variables, augmented with a polynomial function (degree = 3, optimal from section 3.3.1.1) of one of the numerical variables mentioned before, in 3.3.1.1, and a spline (df = 5, optimal from section 3.3.1.2) of the other numerical variable. Another GAM was created where the polynomial function and spline were fitted on the other variable. The authors compared these models by creating an anova table and found that these two models have similar performance.

3.3.2 Tree-Based Models

Tree-based models utilize tree structures to make predictions based on feature values. These models build an ensemble of decision trees where the final prediction is made by aggregating the predictions of the individual trees. The authors looked at adaptive cross-validation from the caret package for tuning these models where it was computationally feasible. As it turned out, this was only feasible for XGBoost and boosting for a reasonable number of cross-validation folds and tested models, because of their efficiency. By using adaptive CV, settings that are clearly sub-optimal can be discarded rendering tuning more efficient. For other models, a grid search was used in combination with cross validation to find the optimal parameters for each model. The authors focused on the tuning of random forests and XGBoost models as these have proven high predictive power on these types of datasets.

3.3.2.1 Regression trees

A simple single decision tree is fitted with cross validation. As can be expected, a high validation RMSE is found.

3.3.2.2 Bagging

Bagging is an ensemble technique in which multiple trees are trained on different bootstrap samples while using all the variables when performing a split. Then, the resulting predictions are averaged to make the final prediction.

3.3.2.3 Random forests

Random forests are an extension of bagging where a split only considers a random sample (mtry) of the p predictors. Consequently, the trees are decorrelated.

As the authors think random forests can significantly improve predictions, different models were created. Firstly, a model was created with mtry = p/3 (=34) predictors (the default for regression) and ntree = 150. Secondly, a model was tuned over 5 values of mtry (28, 31, 34, 37, 40). However, this model was unable to outperform the standard random forest model on the validation set.

3.3.2.4 Boosting

Boosting is a machine learning ensemble technique in which multiple models are trained sequentially, with each model trying to correct the mistakes of the previous model, and their predictions are combined to make a final prediction.

3.3.2.5 XGBoost

XGBoost is a highly scalable implementation of gradient boosted decision trees, designed for speed and performance. Predictions on the validation set yielded the lowest RMSE of all tested models. Hyperparameter tunings can be found in Appendix D (table 7).

3.3.3 Support Vector Machines

Even though Support Vector Machines are mainly used for classification purposes, it also works for regression tasks. Therefore, the authors decided to create a very basic SVM model with a linear kernel. The goal of this model is mainly to benchmark it against the linear and tree-based models.

3.3.4 Neural Networks

Several neural networks were trained on the training data. To find the best model, the authors experimented with different architectures. Within the architecture of a neural network, there are two important variables that need to be tuned. First, the number of hidden layers. Second, the number of hidden units. To find the optimal number of hidden layers, models with one up until four hidden layers were built. Next, different parameters were tuned for these models with a grid search, including the number of hidden units per layer. For all hidden layers a 'ReLu' activation function was used. Then, the output layer used a linear function as this is a regression problem. The optimal parameters for each layer can be found in Appendix E.

Within the code, the guide in figure 3 was followed to train the different neural networks.

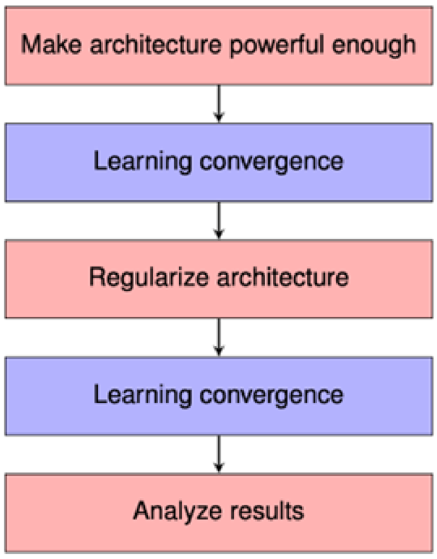

Figure 3: Training guidelines

3.3.4.1 Architecture

First, a powerful architecture that could possibly overfit the data was made for each of the four models. For each layer, the number of neurons that were used and the results of the model will be discussed below. The approach starts with building a wide model with few layers, evolving to less wide but deeper models that have more model complexity.

One layer model As there are a lot of variables, the authors started by building a wide model with a lot of neurons in the first layer. A grid search was performed for values from 128 to 640 neurons (with steps of 128). The number of neurons is higher than the number of input variables to overfit the data. This is not considered a problem as multiple regularization techniques (described below) are used. This one layer model gives us the best results of all neural networks with a RMSE of 22.26.

Multiple layer model For the models with multiple layers, the same approach as in the first layer is used. First, the data was overfitted by using a lot of neurons in the first layer. However, as the number of layers increased, the the number of neurons in each layer were decreased to reduce model complexity. For the two and three layer model, we go as wide as 384 neurons in the first layer. For the four layer model we reduce the number of neurons in the first layer to 128.

3.3.4.2 Architecture Regularization

Second, to reduce overfitting of the model, three regularization methods were used. First, a dropout layer was added after each regular layer in order to reduce overfitting. An optimal dropout rate was found by adding two parameter values to the grid search, namely a rate of 0.3 and 0.4. The authors also experimented with other values in the beginning, but these were never chosen for the optimal model. This technique was used in combination with a maxnorm constraint. This regularization technique is used to prevent overfitting in neural networks by limiting the maximum Euclidean norm of the weights of the network. The evaluated values were 0.5, 1, 2 and 3.

As a final step, an early stopping criterion was introduced to reduce overfitting. During training, the mean squared error on the validation set was periodically evaluated. If the metric had not improved for 20 iterations, the training process was interrupted and the current model was defined as the final model. This reduced overfitting and decreased the computation time of the model.

3.3.4.3 Learning convergence

Next, the learning rate was tuned. This is a hyperparameter that determines the step size at which the optimizer makes updates to the model parameters during training. It determines how quickly or slowly the model learns. Two different values for the learning rate were used in the grid search, namely 0.01 and 0.001.

Further, a batch size of 128 was used. As there are 66 425 training observations, this results in approximate 519 minibatch gradient updates per epoch. The batch size is a hyperparameter that determines the number of observations used for each gradient step. It is an important factor in the training process because it can affect the speed and convergence of the model.

Furthermore, the number of epochs to train each model needed to be determined. This was done by visually checking the convergence rate of the models while it was running. The one- and two-layer models were tuned with 200 epochs. This choice was a trade-off between the computation time and the incremental improvements in performance on some tuning models. Besides, a stopping criterion for models that stopped improving was defined. For the three and four layered models, 150 epochs were used. However, the authors noticed that these models were sometimes too complex for the data. This is illustrated in the validation curve of one of the tuning runs for a four layer model (see Appendix F). In these cases, the models were stopped by the stopping criterion very soon. The optimal tune for each model, was retrained using 300 epochs. This gave a small increase in performance for the one and two layered model as these models still had very incremental improvements after 200 epochs.

3.3.4.4 R-code explanation

In order to train the different neural networks, the following R-code was written. First, the main file 'Neural_networks' defines the grid search parameters for each model. Furthermore, there are different R-scripts in which we define each model: layer1_model, layer2_model… When performing a tuning run for a model with the function 'tuning_run', the main file uses these R-script to train a particular model with the corresponding grid search values. While tuning the models, the results are stored in the tuning directory of each layer: _tuning, _tuning2, _tuning3… So, it is also possible that previous grid searches are stored in this directory. As the model complexity increases, more and more combinations are possible in the grid search. Therefore, the parameter 'sample' was used of the function that reduces the number of possible combinations that are tested. For the one-layer model, sample was set to 0.4, for the two layer model to 0.3 and the others to 0.2 due to the increase in number of combinations. In order to fit the data, the validation set approach was used that is available in the 'fit' function. A percentage of 20% was specified that will be used as validation data to find the optimal hyperparameters based on the validation performance.

3.3.4.5 Results

Of all the NN-models trained, the one-layer model gave the best performance with a RMSE of 22.26348. This model had 512 neurons in the first layer, a dropout value of 0.4, a maxnorm constraint of 1 and a learning rate of 0.001. For the other models, the performance decreases substantially (see Appendix A). However, while training the different models, the authors noticed in the validation curve that there is a wide gap between the training and validation loss of the models. This could indicate overfitting of the data, but this is not likely due to the regularization techniques. A more likely explanation is that the models were too complex. Therefore, it is logical that the one-layer model yields the best performance. Besides, more data would also help address this issue. This trend was also noticed while training our best model (see Appendix F). Besides, there it is also clear that the models performance only improves incrementally after epoch

50. Therefore, it is recommended to use less epochs if a decrease in computation time is desired.

4 Evaluation

The results of the previous section are summarised in Appendix A (Table 4). Compared to the linear benchmarks, more complex models (XGBoost, RF, Bagging, ANN) perform better on the validation set. Of course, all coins have two sides, which is something that will be discussed in section 5 Reflection when the topic of Occam's razor is discussed. Based on the the validation RMSEs, the authors decided to retrain a Sequential Replacement stepwise regression model, an XGBoost model, a Random Forest model, and an Artificial Neural Network for the Kaggle competition. These models were trained on the validation and train set, using hyperparameters obtained in section 3 Modelling. The Public leaderboard test set RMSE results can be found in Appendix B (Table 5).

At the end, the authors included the variable day_of_month_arrival and this proved to provide a better prediction using XGBoost (Table 2). As this feature was only included at the end of the assignment, all information in this report is based on the first version of gold data. Therefore, the authors decided to make 'Feature_engineering2' which is the same as the main FE file, except for the part that adds the new variable. The hyperparameters tuned on this basetable are the same from the previous basetable.

Model	Validation set RMSE	Kaggle test set RMSE
XGBoost after FE2	17.2251	16.22623

Table 2: XGBoost After FE2

The authors recommend using tree based ensembles based on their predictive performance. Especially XGBoost has performed well, partly because it is more efficient than random forests and the other tree ensembles discussed. This means more XGBoost configurations can be tested within a reasonable time and the chance of finding a model with good predictive power increases.

5 Reflection

The authors note that the more advanced and complex the model, the better the performance. This confronts machine learning engineers and hotel managers with a trade-off: predictive performance can be significantly improved at a cost of interpretability and often computation time.

However, at some point someone has got to draw a line. The authors spent significantly more time on creating an artificial neural network than on a random forest or a simple

Lasso regression. The question can be asked whether this was worth it. The authors believe that to make the ANN worth the while, more training data and computational resources are needed.

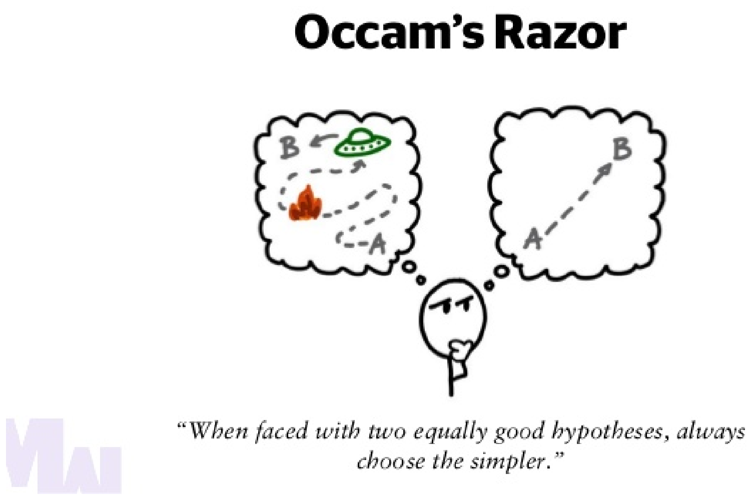
Occam's Razor
B
A
B
A
"When faced with two equally good hypotheses, always choose the simpler."

Figure 4: Occam's razor

With this dataset, the authors do not think that building an ANN is necessary, especially when considering Occam's razor (figure 4). If simpler, less computationally expensive models can lead to similar or even better performance, they are to be preferred.

Further, a feature selection method could be used to reduce the dimension of the problem, to reduce the risk of overfitting and to increase hyperparameter tuning speed. Principal Component Analysis was tried to reduce the number of numerical features, but this only reduced the number of features by 2 and didn't increase predictive performance on the validation set. Additionally, the authors also attempted to perform a FAMD (Factor Analysis of Mixed Data) analysis, but were unable to successfully implement it using the R programming language.

Another technique that could be used to increase predictive power is stacking: an ensemble technique where a higher-level model is trained on predictions of base models (figure 5). This could improve the prediction, especially when there are a high number of well-performing base models.

Lastly, the authors considered adding a feature "occupancy_rate" as they believe it would also enable a better price prediction. The business understanding led to the believe that when demand rises, so does the price. However, with the provided data, this proved to be a cumbersome task.

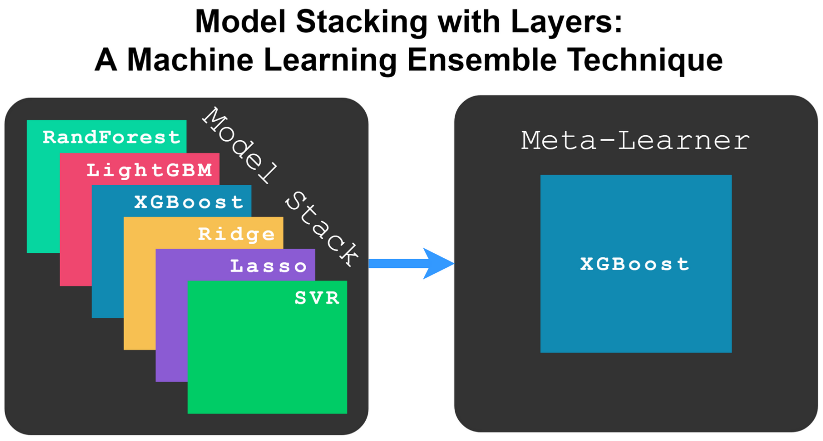

Figure 5: Model stacking

6 References

James, G., Witten, D., Hastie, T., Tibshirani, R. (2021). An Introduction to Statistical Learning with application in R (second edition).

Hotz, B. N. (2022, December 5). What is CRISP DM? Data Science Process Alliance. https://www.datascience-pm.com/crisp-dm-2/

N. Antonio, A. de Almeida, L. Nunes, Hotel booking demand datasets, Data Brief 22 (2019) 41–49, http://dx.doi.org/10.1016/j.dib.2018.11.126.

Jain, A. (2022, November 30). XGBOOST parameters: XGBoost parameter tuning. Analytics Vidhya. Retrieved December 17, 2022, from https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Mavuduru, A. (2022, August 9). What Occam's razor means in machine learning. Medium. Retrieved December 17, 2022, from https://towardsdatascience.com/what-occams-razor-means-in-machine-learning-53f07effc97c

https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10

automaticaddison, A. (2019, June 27). Occam's Razor and machine learning. Automatic Addison. Retrieved December 18, 2022, from https://automaticaddison.com/occams-razor-and-machine-learning/

Chen, T., & Guestrin, C. (2016). XGBoost. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. https://doi.org/10.1145/2939672.2939785

Kuhn, M. (2014b). Futility Analysis in the Cross-Validation of Machine Learning Models. Cornell University - ArXiv. https://doi.org/10.48550/arxiv.1405.6974

Hansen, C. (2020, January 16). Stack machine learning models: Get better results. IBM Developer. Retrieved December 19, 2022, from https://developer.ibm.com/articles/stack-machine-learning-models-get-better-results/

7 Appendix

Appendix A Internal Evaluation of Models

The created models were all trained on a training set. Afterwards a RMSE metric was calculated on a validation set. This provided the authors with a first sign of the performance of the model. This table contains all used model and their internal validation set performance.

Model	Validation set RMSE
Linear regression	30.71916
Forward stepwise regression	30.71786
Backward stepwise regression	30.71776
Sequential Replacement stepwise regression	30.70939
Ridge regression	32.33805
Lasso regression	30.71731
Polynomial regression	30.72
Splines	30.637
GAM	30.61176
Regression tree	36.44005
Bagging	21.8329
Random Forests (no CV)	18.5983
Random Forests (with CV)	18.61649
Boosting	21.05618
Boosting (with CV)	19.80656
Boosting (with adaptive_CV)	20.30953
XGBoost	17.63677
SVM	36.42701
Artificial Neural Network (1 layer)	22.26348
Artificial Neural Network (2 layer)	51.49626
Artificial Neural Network (3 layer)	35.8843
Artificial Neural Network (4 layer)	32.19931

Table 4: Internal evaluation (validation set)

Appendix B External Evaluation of Models

Based on the internal evaluation scores, certain models were selected to train on the train and validation set, and then used to make predictions on the test set. This table contains all of these models and their external performance. However, it should be noted that the best performing model is not in this table, but can be found in section 4 Evaluation.

Model	Public leaderboard test set RMSE
Sequential Replacement stepwise regression	30.38075
XGBoost	16.61752
Random Forests	18.58828
Artificial Neural Network (1 layer)	22.98558

Table 5: External evaluation (Kaggle public leaderboard)

Appendix C Features

Original feature	**Newly created or removed features**
id	*removed for train and val*
arrival_date	*arrival_date_weekday + year_arrival + week_of_month + month_arrival*
assigned_room_type	*removed: see reserved_room_type*
booking_agent	booking_agent
booking_company	booking_company
booking_distribution_channel	booking_distribution_channel
canceled	canceled
car_parking_spaces	car_parking_spaces
country	country
customer_type	customer_type
days_in_waiting_list	days_in_waiting_list
deposit	deposit
hotel_type	hotel_type
is_repeated_guest	is_repeated_guest
last_status	last_status
last_status_date	*time_between_arrival_checkout* in combination with last_status
lead_time	lead_time
market_segment	market_segment
meal_booked	meal_booked
nr_nights	*nr_weekdays + nr_weekenddays*
nr_previous_bookings	*removed because sum of previous_bookings_not_canceled and previous_bookings_canceled*
nr_adults	nr_adults
nr_babies	nr_babies
nr_booking_changes	nr_booking_changes
nr_children	nr_children
previous_bookings_not_canceled	previous_bookings_not_canceled
previous_cancellations	previous_cancellations
reserved_room_type	reserved_room_type + *room_type_conflict*
special_requests	special_requests
average_daily_rate	average_daily_rate

Table 6: Feature Engineering

Appendix D Tree Based Algorithms Parameters

Parameter	Value
nrounds	775
max_depth	10
eta	0.1095579
gamma	8.648076
colsample_bytree	0.4743906
min_child_weight	16
subsample	0.7799961

Table 7: XGBoost parameters

Appendix E Artificial Neural Network Parameters

Following tables provide the optimal parameters resulting from the tuning of the neural networks.

Parameter	Value
dropout	0.4
neurons 1	512
maxnorm	1
learning rate	0.001

Table 8: ANN layer 1 model parameters

Parameter	Value
dropout	0.3
neurons 1	384
neurons 2	128
maxnorm	1
learning rate	0.001

Table 9: ANN layer 2 model parameters

Parameter	Value
dropout	0.3
neurons 1	384
neurons 2	64
neurons 3	64
maxnorm	0.5
learning rate	0.001

Table 10: ANN layer 3 model parameters

Parameter	Value
dropout	0.3
neurons 1	128
neurons 2	64
neurons 3	32
neurons 4	32
maxnorm	1
learning rate	0.001

Table 11: ANN layer 4 model parameters

Appendix F Validation Curves

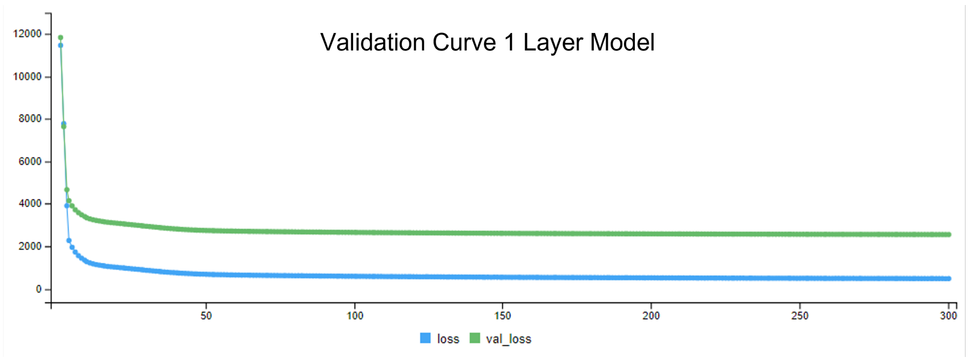

Figure 6: Validation Curve 1 Layer Model

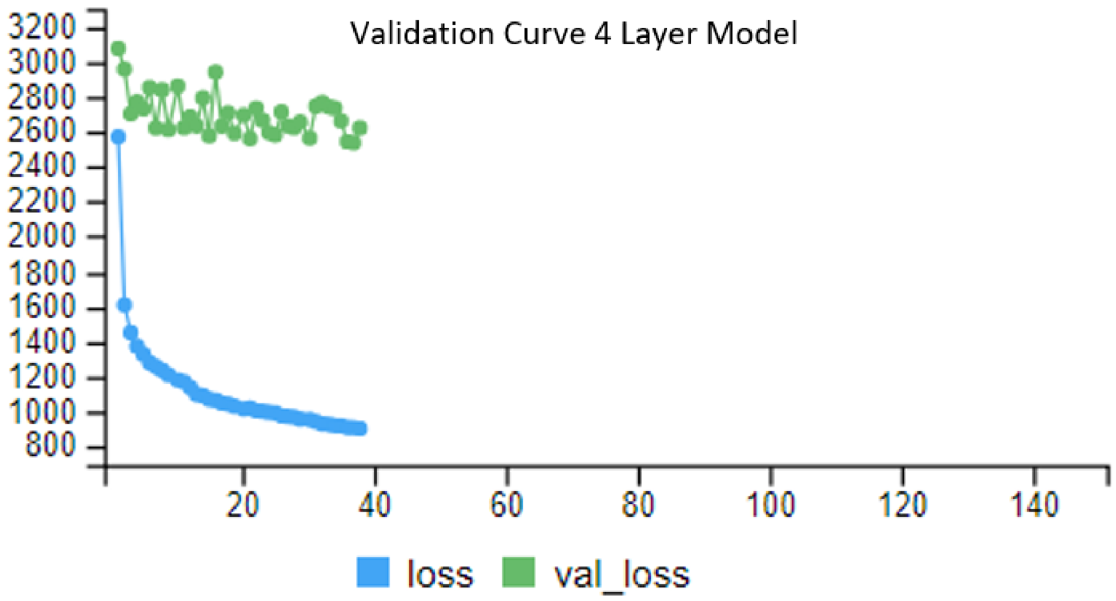

Figure 7: Validation Curve 4 Layer Model