INCLASS KAGGLE COMPETITION: HOTEL ROOM PRICE PREDICTION

Technical report

Robert Xu

Ruben Mertens

Hanne Van Impe

Sien Van Herreweghe

TEAM 12

MACHINE LEARNING 2022-2023

TABLE OF CONTENTS

Introduction

Fixing the right price in order to maximize your revenue was and will always be a very hard task. For this reason, we wanted to investigate this aspect regarding hotel prices and gain insights into the main factors driving these hotel room prices. We tried to come up with the optimal prediction model for the hotel room rate in order to support hotel managers in optimizing their pricing strategy.

After obtaining the data, we immediately got to work and analyzed the different factors that influence the average daily rate. In this report, we will explain to you how we handled the data and what prediction models we performed. Firstly, we explored the data, followed by the data preprocessing, where we cleaned the data precisely and performed feature engineering. Thereafter, we moved on to the prediction models. We will explain to you how we performed each model, why we chose this model, and which different steps we executed to obtain the optimal form of the regarding model. Lastly, we will terminate this report with a short comparison and conclusion.

Data exploration and cleaning

First of all, we explored the data to get some useful insights in order to do the data cleaning and feature engineering better and more efficiently. This entails plotting and calculating the mean and median for numerical variables and the mode for categorical variables on the training data. Besides that, we also gained insights by looking at the training data on the Kaggle website since we could easily see how many missing values each variable had. *Booking_company, nr_booking_changes* and *nr_babies* had a lot of missing values, therefore we decided to exclude these variables from the data. *Days_in_waiting_list* consisted for 90% of zero's and 7% of missing values, so we excluded this one as well. Moreover, *booking_agent* had a lot of different values, so we decided to leave this one out as we think this would not provide a lot of useful information but would only make the models more complex.

After the data exploration, we started the data splitting. This is a very important step as we must prevent data leakage as this would result in an overly optimistic model. The data was split into a training-, validation- and test set. We decided to split the training data into 70% training set and 30% validation set, which resulted in a training set of 58.124 observations, a validation set of 24.910 observations and a test set of 35.585 observations.

We started the data cleaning by replacing the missing values by a constant. We imputed the missing values for the categorical variables with the mode except for *last_status.* We used the mode from the training set for the validation set and test set, to avoid data leakage. The mode of *last_status* is

"check-out", but due to the link between the variables *canceled* and *last_status*, we decided to impute the missing values based on the value of *canceled*. Since this column had no missing values, the imputation would be more accurately for *last_status*. Thereafter, we dropped the variable *canceled* from the data as it provides more or less the same information. We chose *canceled* and not *last_status* because *last_status* has the values: check-out, canceled and no show, while *canceled* only has no cancellation or stay cancelled. Furthermore, we extracted the month and day of the week of the variable *arrival_date* as these variables are more useful than the date itself. We decided not to do this for *last_status_date* because this would not give us a lot of additional information compared to the increase of complexity. Thus, we excluded the variables *arrival_date* and *last_status_date* from the data. We also chose not to extract the year for *arrival_date*, as this would not give much insights for future years.

To continue, we imputed almost all numerical variables with the median except for *lead_time*, for which we chose the mean. Again, the mean and median of the training set were used for the validation- and test set. We noticed that for the variables: *nr_previous_bookings, previous_cancellations* and *previous_bookings_not_canceled*, the missing values could be calculated accurately using the other 2 variables, if these 2 variables didn't have missing values for that observation, otherwise, we used the median. After these calculations, *previous_bookings_not_canceled* was dropped, as the other 2 provided enough information and we wouldn't want to make the models too complex.

Furthermore, we handled the outliers by imposing an upper limit on the numerical variables. First, we did this by using the Z-scores but these values did not make much sense, therefore we decided to determine the upper limit ourselves. Imposing a lower limit was not necessary. We did see that *average_daily_rate* has around 900 observations of the training data which are equal to 0 which is odd. We noticed that for around 1/3 of these values the *market_segment* was "Complementary". As the definition of complementary is that you have been granted to stay in a hotel without having to pay, we could conclude that an *average_daily_rate* equal to zero should not be treated as an outlier. However, there are still 2/3 of the observations which could not be explained by this reason. After taking a closer look, we found some other possible reasons. The most common ones were: missing values for *market_segment* in the raw data which could mean that this *market_segment* was also complementary, *nr_nights* equal to 0, *nr_adults* equal to 0… Therefore, we again concluded to not treat it as an outlier and leave it as it is.

Feature engineering

Feature engineering is an important step in the machine learning process, as it can significantly improve the performance of machine learning models. We used the following steps:

- Scaling

We scaled the numeric features in our dataset and got the mean μ and standard deviation σ for both columns in the training set. Thereafter, we standardized the training set and used μ and σ to standardize the validation and test set.

- Encoding categorical features

We performed dummy encoding for categorical features such as *assigned_room_type, customer_type, last_status, market_segment, meal_booked, month_arrival_date*, and *day_arrival_date.* We chose dummy encoding for *meal_booked*, instead of integer encoding because some models do not perform well using integer encoding such as linear regression.

- Extracting features

Furthermore, we noticed that *reserved_room_type* and *assigned_room_type* often have the same value for an observation. Thus, we decided to delete *reserved_room_type* and make a new variable instead, namely *desired_room*, which equals to 1 when the reserved room is the same as the assigned room and 0 if otherwise. This way, we could reduce the number of columns since *reserved_room_type* would have needed much more columns when it's dummy encoded. In addition, we also deleted the variables *country* and *booking_distribution_channel* for the same reason.

The models

In this part of the report, we will elaborate on the models we performed and what results we came to after finetuning each model until the optimal form. Firstly, we will explain which parameters were tuned and how they were tuned in order to increase performance and prevent overfitting and where this tuning was performed on the training set and tested on the validation set. Only the better models were retrained on training + validation set with the obtained optimal parameters and finally tested on the test set. Thereby, every model's RMSE and MAE will be shown and every model's link with the Occam's razor will be explained.

Finally, after all the different models are enlightened, an overall overview of all the errors and a comparison between the different models will be provided in order to have a clear look of the performance of each model.

Linear model

The first model we executed to fit our data was the linear regression model. Since linear regression is prone to overfitting, we opted for 2 shrinkage methods, namely the Ridge and the Lasso regression. For both methods, we performed hyperparameter tuning for lambda using 5-fold cross-validation. The optimal value for lambda resulted in the following RMSE's and MAE's, where the RMSE and MAE using Ridge and Lasso do not differ a lot from each other.

Table 1: Errors of the linear model

	RMSE	MAE
Linear model (Ridge)	46.9450739620167	24.0875589083806
Linear model (Lasso)	46.9452004237641	24.0878229405589

Polynomial model and GAM

After finishing the linear model, we performed a polynomial regression. Besides the normal polynomial model, we also created a regularized polynomial model where we included both the Lasso and the Ridge penalty once.

In order to find out which degree of the numerical variables would be the best option, we performed for each of these variables an ANOVA test. Thereafter, we created an optimal polynomial model, containing the categorical variables in a linear way and the best polynomial degree of the numeric variables. We used the poly() function here since these terms are uncorrelated, which is not the case when using the I() function.

Thereafter, we wanted to regularize the polynomial model by adding a penalty term in an attempt to minimize the errors. We made again use of the ANOVA tests and created the polynomial model, using the optimal degree of the numeric variables. Only now, we used the I() function in stead of poly() since we could not fit both the lasso nor ridge regression model when using the poly() function. We fitted both the Ridge regression model and the Lasso regression model, both using cross validation in order to find the optimal lambda for the model, and computed the RMSE and MAE for each model.

As you can see in the table below, neither the polynomial nor the regularized polynomial model results in a lower error or a better model. We did not expect for these kind of changes to have a

major positive impact on the model, whereby we immediately performed 2 generalized additive models, namely smoothing splines and local regression.

Table 2: Errors of the moving beyond linearity models

	RMSE	MAE
Polynomial model	47.8232444073489	24.2590743626267
Polynomial model (Ridge)	47.8232444073485	24.2590743626255
Polynomial model (Lasso)	47.8232444073485	24.2590743626255
Smoothing spline model (Lasso)	46.9450187663735	24.0883525771904
Local regression model (Lasso)	46.9450187663735	24.0883525771905

For the spline model, we made use of smooth splines for all numerical variables and performed a Lasso regression on the obtained model, using cross validation. Thereafter, we did the same for the local regression and calculated for both models the RMSE and MAE. As you can see, neither of these models resulted in an improvement compared to the linear model.

As the complexity increased due to higher degrees, smoothing splines or the local regression, but the performance did not, we prefer the linear model according to Occam's razor.

Random forest

As a proceeding model, we chose random forest as it is a type of an ensemble machine learning algorithm, which is often used for regression tasks.

The hyperparameters in random forest include *ntree*, the number of decision trees in the forest and *mtry*, the number of features considered by each tree when splitting a node.

We used hyperparameter tuning on *mtry* with three different values, namely p, p/2, and √p, which resulted in the following RMSE's and MAE's on the training set:

Table 3: Errors of Random Forest (hyper parameter tuning)

Mtry	RMSE	MAE
4.582576	24.77480	17.05255
10.500000	22.11402	14.04748
21.000000	21.67507	13.06907

Looking at this table, we can conclude *mtry* = 21 is the optimal value as it gives the lowest RMSE. However, we had overfitting. One way to apply regularization is to use the mtry hyperparameter, which controls the number of features that are considered at each split in the decision tree. A

smaller value of *mtry* will result in a simpler and less complex decision tree, which can help to reduce overfitting. Another way to apply regularization is the use of the *min.node.size* hyperparameter, which controls the minimum number of samples required to create a new split in the decision tree. A larger value of *min.node.size* will result in a simpler and less complex decision tree, which can also reduce overfitting.

Thus, to prevent overfitting, we simply pick the lowest value for *mtry*, the squared root of number of predictors, instead of the optimal value we get from hyperparameter tuning. Therefore, we chose *min.node.size* = 500 and an *ntree* = 100. Although we tried an increase of ntrees up to 500, it did not improve. This led to a RMSE on the training set of 24.69395.

Table 4: Errors of Random Forest

RMSE	MAE
41.7082616838757	16.9499372629923

Afterwards, we made predictions on the validation set, whereof the results are shown in table 4. We still have overfitting at the end, whereby a possible conclusion could be that the random forest model is not a suitable model for our data, given the presence of overfitting.

Support vector regression

The next model in our process was support vector regression. The parameters we tuned using 5-fold cross-validation, were *sigma* and *cost* but due to the high computing power needed, we could not dive really deep into this. For the training of the model, we only tried the method svmRadial. Since the hyperparameter tuning took a very long time, we did not look further into other methods such as svmLinear. The parameter's range and optimal value after tuning are shown in the table below.

Table 5: Range & optimal values of the parameters (SVR)

	Sigma	*Cost*
Range	0.01 - 0.1 - 1	1 - 10 - 100
Optimal value	0.01	10

Table 6: Errors of SVR

RMSE	MAE
41.3513970085533	15.5208363929206

Both errors have decreased which means the SVR model is a further improvement. As already explained, we were not able to tune the parameters further so we can conclude the SVR model performs quite well but could possibly perform even better.

GBM (Gradient Boosting Machine)

To continue, we fitted a GBM model to the data, wherefore we again tuned several parameters, namely *n.trees, shrinkage, n.minobsinnode, and interaction.depth.* For this hyperparameter tuning, the following ranges were used and it resulted in the optimal values below, which were used in the final form of the model:

Table 7: Range & optimal values of the parameters (GBM)

	Shrinkage	*N.minobsinnode*	*Interaction.depth*	*N.trees*
Range	0.001 - 0.01 - 0.1 - 1	5 - 10 - 15 - 20	1:13	500 - 1000 - 2000 - 2250 - 2500 - 2750 - 3000- 4000
Optimal value	0.1	5	10	2250

Table 8: Errors of GBM

RMSE	**MAE**
40.0568441506894	14.7029830342072

As you can see, the errors for this model decreased a bit more compared to SVR. We can surely say this model is already quite a good fit for our data. Despite the model's good performance, the running was very time consuming. In order to resolve this time problem, we tried to fit the next model, namely light GBM.

Light GBM

With the light GBM model, we expected the model's runtime to decrease so that we could dive deeper into the hyperparameter tuning in order to choose the optimal value more correctly. For this model, the package lightGBM was used and the parameters we tuned are: *learning_rate, num_leaves, max_depth* and *lambda_l1.* Again, an overview of the ranges and optimal values for the parameters is shown below.

Table 9: Range & optimal values of the parameters (Light GBM)

	Learning_rate	*Num_leaves*	*Max_depth*	*Lambda_l1*
Range	0.01 - 0.1 - 0.2 - 0.3 - 0.5 - 1	20 - 40 - 60 - 80 - 100 - 120 - 140 - 160	4:20	0.01 - 0.1 - 1
Optimal value	0.15	80	10	0.01

After training the model with the optimal values, the lightGBM model was found to have the lowest RMSE and MAE so far. In order to decrease the errors even more, we tried several things in the data preprocessing. Eventually, it turned out that integer encoding *meal_booked* instead of dummy encoding caused a slight decrease of the RMSE and MAE. Furthermore, we tried to apply a transformation on *average_daily_rate* by taking the square root because we noticed that the variable was right skewed. However, this did not result in an improvement nor a deterioration. The changes resulted in the following RMSE and MAE:

Table 10: Errors of Light GBM

RMSE	MAE
39.7322809704038	13.4160055318152

XGBoost

To continue, we targeted the XGBoost model as we also expected this model to run fast and perform well. The model was also trained with a set of hyperparameters, including the learning rate *eta* *maximum_depth*, *nrounds*, and *alpha* which represents L1 regularization. The RMSE is slightly higher compared to LightGBM but the MAE is smaller. Although we know that the RMSE and MAE differ in the way they penalize large errors, it is still hard to know which metric of the 2 is the best for our model. Therefore, we cannot conclude which model is better.

Table 11: Range & optimal values of the parameters (XGBoost)

	Eta	***Maximum_depth***	***Nrounds***	***Alpha***
Range	0.01 - 0.1 - 1	6:20	40 - 80 - 100 - 120 - 160	0.01 - 0.1 - 0.5 - 1
Optimal value	0.15	10	120	0.01

Table 12: Errors of XGBoost

RMSE	MAE
40.3254309054427	12.978664536275

Deep learning

Deep learning networks learn by discovering intricate structures in the data they experience. In order to fit a neural network, there really is not a standard procedure, it is more like a process of trial and error. Therefore, we tried 3 different approaches to fit the data: a wide model, a deep model and a wide and deep model.

We came to the conclusion that a combination of both a wide and a deep model had the lowest RMSE for our validation set. The structure of our neural network consisted of one input layer, 8 hidden layers (1000, 800, 600, 400, 200, 100, 80 and 40 units respectively) and one output layer. We ran 150 epochs with a batch size of 32 to fit the training data. Thereafter, we regularized our model to prevent overfitting. We increased the batch size to 64. On the first 5 hidden layers, we used a dropout rate of 30% and on the last 3 layers a rate of 20%. Thereby, we set a maximum value of 5 for the weights of our units for each hidden layer.

Table 13: Errors of Deep Learning

RMSE	MAE
40.3360880601974	14.3416000000000

Comparison of the models

After a long process of tuning, testing, predicting, etc…., we took a look at our final results and remarked 6 models that performed quite and almost equally well: Random forest, SVR, GBM, light GBM, XGBoosting and Deep learning. Since Random forest has both the highest RMSE and MAE of these 6 models, we would not opt for this model as our optimal one. The SVR, GBM and deep learning model took up a great amount of time when we ran it, which made them more complex. Thereby, the errors of these models were higher than the errors of light GBM and XGBoost. So although the performances of the models previously mentioned are quite similar, we would still opt for light GBM or XGBoost since these models resulted in the lowest errors and are the least complex models. So looking at Occam's razor, we can select GBM and XGBoost as our optimal models with no doubt.

Table 14: Errors of all the models

	RMSE	MAE
Linear model (Ridge)	46.9450739620167	24.0875589083806
Linear model (Lasso)	46.9452004237641	24.0878229405589
Polynomial model	47.8232444073489	24.2590743626267
Polynomial model (Ridge)	47.8232444073485	24.2590743626255
Polynomial model (Lasso)	47.8232444073485	24.2590743626255
Smoothing Spline model (Lasso)	46.9450187663735	24.0883525771904
Local Regression model (Lasso)	46.9450187663735	24.0883525771905
Random Forest	41.7082616838757	16.9499372629923
Support Vector Regression	41.3513970085533	15.5208363929206
Gradient Boosting Machine	40.0568441506894	14.7029830342072

Light GBM	39.7322809704038	13.4160055318152
XGBoosting	40.3254309054427	12.978664536275
Deep Learning	40.3360880601974	14.341600000000

Reflection

Although we are quite happy about our final findings, there are still some aspects that could have gone better or did not have the results we expected them to have.

Firstly, we had great expectations for our last model, namely the deep learning model since we spend a lot of time on processing this model. We do think that globally, we performed the right steps since we were able to close the gap of the RMSE, in contrast to the gap of the loss curve (cfr. Appendix). Hereby, we're convinced there are several things that we could have improved.

Thereby, we doubted the way we handled the replacement of the missing values at the beginning of our process. Hereby, we made use of the means and medians of the training set after the split with the validation set, so not the initial training set. But as we use the full training set to then predict with the test set, we had our doubts whether we should have used the means and medians of the full training set to impute the missing values on the test set. We tried this for the LGBM model to see whether it would make a big difference, but this was not the case.

Thus we can conclude that, despite we still could have improved several aspects in our project, we ended up being quite satisfied about our final results and most importantly, we learned a lot during the process.

Sources

- https://stackoverflow.com/
- https://www.rdocumentation.org/
- https://www.w3schools.com/
- https://www.geeksforgeeks.org/
- https://www.statology.org/

Appendix

Figure 1: Histogram of the "Average daily rate"

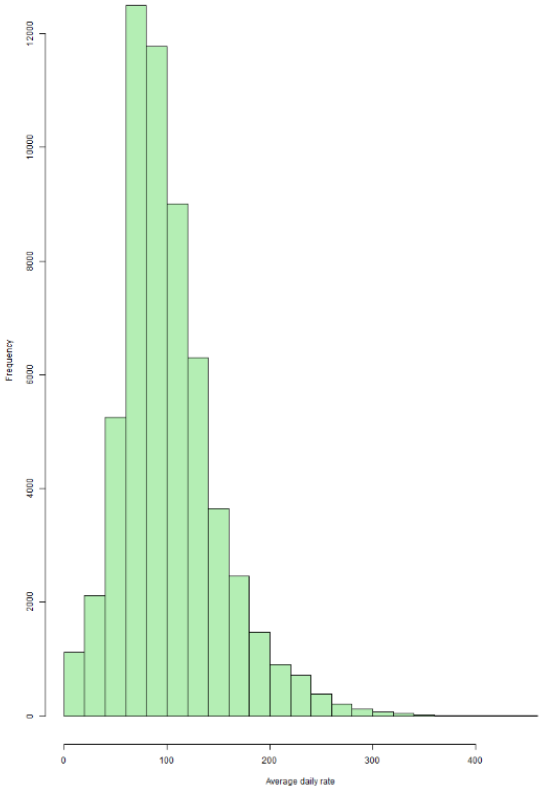

Figure 2: Histogram of the square root of "Average daily rate"

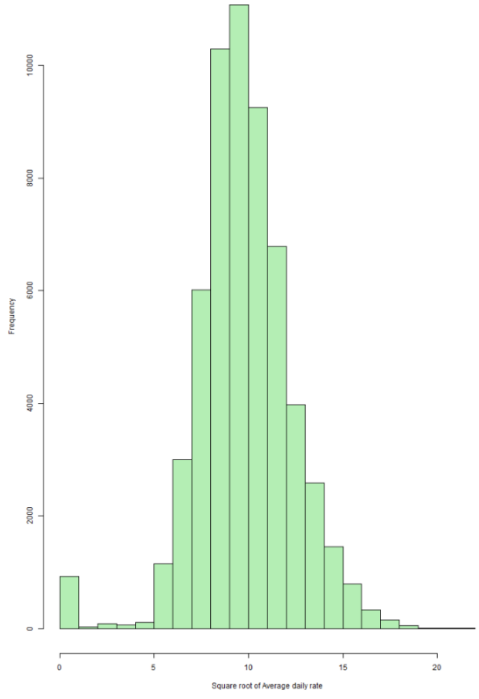

Figure 3: MSE and MAE of training and validation data from our neural network

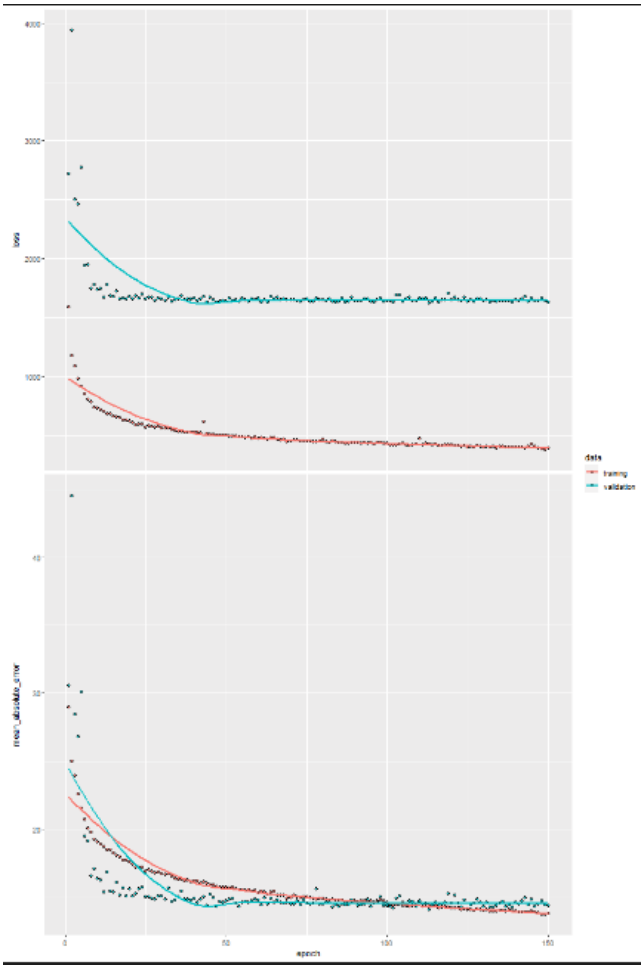

Table 15: Means of the numeric variables

Numeric variable	Mean
Car_parking_spaces	0.062582
Nr_adults	1.857714
Nr_children	0.103822
Nr_previous_bookings	0.218973
Previous_cancellations	0.088081
Previous_bookings_not_canceled	0.127316
Nr_nights	3.429912
Nr_booking_changes	1.458086
Days_in_waiting_list	2.306868
Special_requests	0.571835

Table 16: Medians of the numeric variables

Numeric variable	Median
Car_parking_spaces	0
Nr_adults	2
Nr_children	0
Nr_previous_bookings	1
Previous_cancellations	0
Previous_bookings_not_canceled	0
Nr_nights	3
Nr_booking_changes	1
Days_in_waiting_list	0
Special_requests	0
Nr_babies	NA
Lead_time	254

Table 17: Modus of the categorical variables

Categorical variable	Modus
Assigned_room_type	A
Reserved_room_type	A
Arrival_date	December 5 2015
Booking_agent	9
Booking_company	NULL
Booking-distribution_channel	TA/TO
Canceled	No cancellation
Country	Portugal
Customer_type	Transient
Deposit	No deposit
Hotel_type	City hotel
Is_repeated_guest	No
Last_status	Check-out
Market_segment	Online travel agent
Meal_booked	Bed & breakfast (BB)
Average_daily_rate	62\200

Table 18: Outliers

Name of the variable	Upper limit
Car_parking_spaces	3
Lead_time	365
Nr_adults	10
Nr_nights	21
Nr_nights	5
Nr_previous_bookings	15
Previous_cancellations	10