Prague University of Economics and Business
Faculty of Informatics and Statistics

nám. W. Churchilla 4, 130 67 Praha 3 - Žižkov

Prague Airbnb price per night prediction

Alena Etrichová (etra00), Lukáš Volf (voll03), Viliam Virba (virv00)

4IT439 – Data-X – applied data analytics models in real world tasks
Summer semester 2024

Table of Contents

Introduction

The purpose of this assignment is to analyze Prague's Airbnb rent listings data and **predict the price per night as precisely as possible**. The dataset is taken from the Airbnb website and is free to use, licenced as *CC BY 4.0 DEED Attribution 4.0 International*.

During this assigment, we have first gathered the datasets, tried to understand the data and prepare it for usage in our analysis. In doing so, we described the data in a form of short description for each column, indication of missing values, type of values, unique values and outliers. We also show some descriptive statistics where applicable. In addition, we present a few data visualisations to make a better sense of the data. Lastly, we have performed data processing and cleaning operations (e.g. getting rid of columns unimportant for our task, column aggregation, dummy values conversion, null values records removal etc.)

In the next part of this task, we have split the dataset into 3 parts – training, validation and testing. Afterwards, we will train a couple of selected models (each specialising on a slightly different machine learning method) on our training data and validate the results for each model. Then, we have compared these results and selected the best model to solve our problem.

Finally, we are present the results of our analysis, together with a description and interpretation of our conclusion. This part will also contain additional visualisations for better explainability.

This assignment is part of academic course *4IT439 – Applied data analytics models in real word tasks* taught at *Faculty of Informatics and Statistics, Prague University of Economics and Business (FIS VŠE)*.

1. Data

To predict the price per night for Airbnb rent listings, we will use datasets provided from Inside Airbnb website1. Our goal is to analyze Prague listings, so we will only use data assigned to Prague, Czech Republic.

For the purpose of our analysis, we chose datasets listings.csv that provides summary information and metrics for various rent listings in Prague. Based on the description from the website, this dataset contains generally quarterly data over the period of last 12 months (InsideAirbnb, 2024).

1.1. Data Understanding and Preparation

In first place we looked at our dependent variable that we wanted to predict and that is price per night. Since price per night was formatted with dollar sign we needed to convert it into float datatype for further working with this variable. We looked at the summarizing statistics of price and found out that there is more than 7 percent of missing values. Since it is not so big proportion we decided to not fill this values with "artificial" calculated values but simply filter this rows from dataset so we got 8 708 rows left from 9 388 in first place.

Then we looked on potential independent variables specifically numeric ones.

[1] https://insideairbnb.com/get-the-data/

Again looking at missing values we filter out columns that has more than 20 percent of missing values because that was columns that were completely blank. After that we worked with uniqueness of values and remove scrape id that was almost absolutely no unique and also id that was perfectly unique since it was primary key. Next step was fill in missing values in columns that left and we did it by median of each of these numeric columns. For purposes of finding most correlated features we filter outliers by deleting rows where at least one outlier detected by quantiles was present and create correlation matrix. Then we looked at absolute values of columns and choose the ones that were higher than 0.1. By looking at correlation matrix and pairs of correlated columns only with that features we also solve multicollinearity and tested which columns were significant in linear regression. We find out that best were accommodates,review_scores_location and host_total_listings. Based on this findings we come back to dataset in state before filtering outliers by all numeric columns and we filter data frame only by these three column plus dependent variable price outliers and work with these 6 303 rows.

Then we needed to reformat non-numeric variables to numeric or at least to categorical and greate from them dummy variables. We also added once more to enrich dataset that was air distance from city centre. Edited were percentages of host response and acceptance rate then bathrooms and bedrooms count. We created dummies for room_type 4 categories and then columns
host_is_superhost_t,host_has_profile_pic_t,host_identity_verified_t,instant_bookable_t and bathroom_type_shared. With this features and also ones that were selected from original numeric variables we split the data and use them differently in regression, decision trees and neural net models.

2. Modelling

To predict the prices as accurately as possible, we have tried different models and their hyper-parameters. The models we tried are:

- Linear Regression
    - Standard Linear Regression
    - Ridge Regression
    - LASSO Regression
    - ElasticNet
- Decision Trees
    - Single Tree
    - Random Forest
    - CV Forest
- Neural network

Hyper-parameter optimization was done by GridSearchCV during training and validation of the models (except for Neutral network). Neutral networks were instead validated with a hold out on the test data. The cross validation (CV) was 5-fold. To find best hyper-parameters script with different hyper-parameter settings was run multiple times. The finest detail of hyper-parameter grid was then kept in the script. Best hyper-parameter settings were assessed by negative mean squared error (NMSE), which is just a negative MSE. The reason is that sklearn cross validation assesses higher return values as better than lower return values (Skicit-learn, 2024), therefore the negative of MSE is calculated instead of the regular positive.

1.1. Linear Regression

Linear Regression is probably the pickiest in terms of data preparation. It cannot digest categorial data, is susceptible to outliers or works better with normalized / standardized metric variables. We dealt with categorial data by One Hot and Dummy Encoding, removed outliers, but did not normalize the metric variables.

Possible improvements can be made by normalizing the data (mostly the variable price). Moreover, we tried only linear regression not polynomial so the model could be improved by adding polynoms as well.

Compared to Neutral Nets or Trees, Linear regression has almost no hyper-parametres to optimize. We optimized only the level of regularization of different models.

1.1.1. Standard Linear Regression

For standard linear regression we have not used any hyper-parameters, therefore we did not use GridSearchCV but cross_validate by sklearn.

```
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)

lin_reg_cv = cross_validate(model, x_train, y_train, cv=5,
                            scoring='neg_mean_squared_error')
```

1.1.2. Ridge Regression

```
hyper_grid_ridge = {'alpha': [3,5,10,20],
                    'fit_intercept': [True],
                    'random_state': [seed],}
ridge_model_cv = Ridge()
grid_search_ridge = GridSearchCV(estimator = ridge_model_cv, param_grid =
                                 hyper_grid_ridge, cv = 5, n_jobs = -1,
                                 verbose = 2,scoring='neg_mean_squared_error')

grid_search_ridge.fit(x_train, y_train.values.ravel())

print(grid_search_ridge.best_params_)
```

Optimal level of regularization (alpha) was minimal (=1), but we kept 3.

1.1.3. LASSO Regression

```
hyper_grid_lasso = {'alpha': [3,5,10,20],
                    'fit_intercept': [True],
                    'random_state': [seed]}
lasso_model_cv = Lasso()
grid_search_lasso = GridSearchCV(estimator = lasso_model_cv, param_grid =
                                 hyper_grid_lasso, cv = 5, n_jobs = -1,
                                 verbose = 2,
                                 scoring='neg_mean_squared_error')

grid_search_lasso.fit(x_train, y_train.values.ravel())

print(grid_search_lasso.best_params_)
```

Again, the optimal level of regularization (alpha) was minimal (=1), but we kept 3.

1.1.4. ElasticNet

```
hyper_grid_elnet = {'l1_ratio': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                    'alpha': [3,4,6,8,10,20,50,100],
                    'fit_intercept': [True],
                    'random_state': [seed]}
elnet_model_cv = ElasticNet()
grid_search_elnet = GridSearchCV(estimator = elnet_model_cv, param_grid =
                                 hyper_grid_elnet, cv = 5, n_jobs = -1,
                                 verbose = 2,
                                 scoring='neg_mean_squared_error')

grid_search_elnet.fit(x_train, y_train.values.ravel())

print(grid_search_elnet.best_params_)
```

Even for Elastic Net regularization the optimal alpha was minimal, and we kept 3. The optimal l1_ration was 0,9 meaning that lasso regularization was more important.

1.2. Decision Trees

Decision trees are transparent and usually easier to interpret. However, the larger the tree, the harder to interpret. Moreover, some models do not create only one tree but many, e.g. Random Forest.

Even though one of their benefits should be robustness against outliers, it does not mean outliers are not a problem. The reason is that mean squared error (MSE), which is usually used for validation or scoring of the model, is calculated as an average and is therefore susceptible to outliers.

Another disadvantage is overfitting, to which they are quite susceptible to. Hyper-parameters or other methods are needed to prevent overfitting.

1.2.1. Single Tree

For Single Tree modelling, hyper-parameters max_depth and min_sample_leaf were optimized.

```
hyper_grid_tree = {'max_depth': [10,12,13,15,17],
                   'min_samples_leaf': [0.008,0.01,0.02],
                   'random_state': [seed]}
tree_model_cv = DecisionTreeRegressor()

grid_search_tree = GridSearchCV(estimator = tree_model_cv, param_grid =
                                hyper_grid_tree, cv = 5, n_jobs = -1,
                                verbose = 2, scoring='neg_mean_squared_error')

grid_search_tree.fit(x_train, y_train.values.ravel())

print(grid_search_tree.best_params_)
```

Optimal hyper-parameters found by the CV were:

max_depth = 12 and min_samples_leaf = 0.01.

1.2.2. Random Forest

For Random Forest model, hyper-parameters: n_estimators, max_features, min_sample_split and max_deph were given to the GridSearchCV. Min_sample_split is set a bit higher than minimum 2 to help prevent overfitting.

```
hyper_grid_forest = {'n_estimators': [400,800],
              'max_features': [0.1, 0.2, 0.3, 0.4],
              'min_samples_split': [4],
              'max_depth': [15,20,25]}
forest_model_cv = RandomForestRegressor()

grid_search_forest = GridSearchCV(estimator = forest_model_cv, param_grid =
                                  hyper_grid_forest, cv = 5, n_jobs = -1,
                                  verbose = 2,
                                  scoring='neg_mean_squared_error')

grid_search_forest.fit(x_train, y_train.values.ravel())

print(grid_search_forest.best_params_)
```

Optimal hyper-parameters found by the CV were:

max_depth = 15, max_features = 0.1, n_estimators = 400.

1.2.4. Gradient Boosting

For Gradinet Boosting model, hyper-parameters: learning_rate, subsample, max_depth, colsample_bytree, n_ estimators were given to the GridSearchCV.

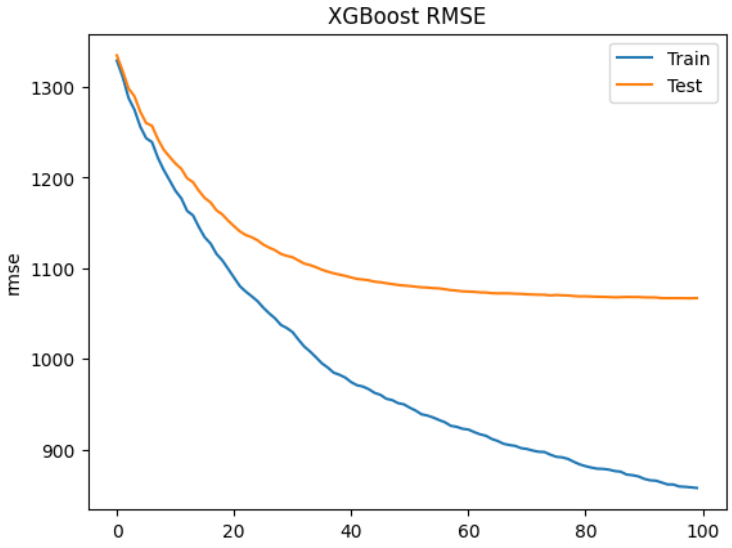

Figure 1 XBGoost RMSE

Number of estimators was set to 60 after running the model for the first time and vizualizing the resiadual mean squared error (RMSE) evolution during the training. To visualize RMSE for out-sample data, data for testing was used in the eval_set and n_estimators was set to 150. As can be seen from the graph, RMSE for testing data is decreasing fast roughly untill it reaches 60 estimators, then the decrease of RMSE is less significant. Due to early_stopping_rounds, the model stopped earlier at n_estimators = 100.

```
hyper_grid_gb = {'learning_rate': [0.03,0.05,0.08],
    'subsample': [0.2,0.6,0.8,1],
    'max_depth': [5,8,10],
    'colsample_bytree': [0.5],
    'n_estimators':[60]}
gb_model = xgb.XGBRegressor(seed=seed, objective='reg:squarederror')
grid_search_gb = GridSearchCV(estimator=gb_model,
                        param_grid=hyper_grid_gb,
                        scoring='neg_mean_squared_error',
                        cv=5,
                        verbose=1,
                        n_jobs=-1)

grid_search_gb.fit(x_train, y_train, eval_set=[(x_train, y_train),
verbose=True, early_stopping_rounds=10)

print(grid_search_gb.best_params_)
```

Optimal hyper-parameters found by the CV were:

colsample_bytree = 0.5, learning_rate = 0.08, max_depth = 8, subsample =1

1.2.5. Neural Net

Lastly, we tried to create a model of neural network for price prediction. First concept of neural networks has been introduced during the 1940s. Their development has come a long way since then, with major parts of research done during the periods of 1960s, 1980s and in the early 2000s. The idea is inspired by the human brain, as it uses interconnected nodes (also called neurons) in a layered structure. Every neural network consists of input layer, output layer and additional hidden layers. Each layer is then made of one or more nodes. Nodes between the layers are interconnected. Input layer processes the data, analyzes it and passes it to next layer. Hidden layers then take this input, process it based on the assigned weights and pass it further.

Processed data then reaches the output layer that provides the results (MIT News, 2017; Amazon, 2024).

In our case, we used Sequential model to define the network architecture. Before we could train the model, we had to reshape the data.

```
# Reshape input data for the model

x_train_reshaped = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],
1)).astype('float32')

x_test_reshaped = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],
1)).astype('float32')

# define the model

model = Sequential()

model.add(Dense(32, input_shape=(x_train_reshaped.shape[1],
x_train_reshaped.shape[2])))

model.add(Dense(32))

model.add(Dense(1))
```

After thorough testing, we decided to use three layer architecture – input, one hidden layer and output layer. Input and hidden layers have 32 nodes, output layer only one.

When it comes to additional parameters, we used Adam (Adaptive Moment Estimation) optimizer with learning rate of 0.01. We also included early stopping set up to 10 epochs and learning rate reducer. During testing, we found that the model performs best with batch size of 512 and number of epochs set to 100.

```
optimizer = optimizers.Adam(learning_rate=0.01)

rmse = metrics.RootMeanSquaredError()

model.compile(optimizer=optimizer, loss='mean_squared_error',
metrics=['mae', rmse])


early_stopping = callbacks.EarlyStopping(patience=10)

learning_rate_reducer = callbacks.ReduceLROnPlateau(monitor='val_loss',
patience=10, factor=0.2)


hist = model.fit(x_train_reshaped, y_train,

                 callbacks=[learning_rate_reducer, early_stopping],

                 batch_size=512,

                 epochs=100,

                 validation_data=(x_test_reshaped, y_test))
```

2.Validation results

Validation results were taken from the GridSearchCV with grid_search.best_score, which returns the cross validation results of NMSE of the model with best hyper-parameters. For the optimal hyper-parameter setting models coefficient of determination was printed as well.

Validation results of the absolute value of NMSE were stored in results dataframe. Only the Neural Network model was validated with hold out data.

```
gb_model_opt = grid_search.best_estimator_

print('MSE: ' + str(abs(grid_search.best_score_)))
print('R: ' + str(gb_model_opt.score(x_train, y_train)))
results = results._append({'model':'name',
                           'MSE': abs(grid_search.best_score_),
                           'R^2': model_opt.score(x_train, y_train)},
                                  ignore_index = True)
```

To the results dataframe column with RMSE was then added as a square root of MSE. Below we can see the results of validation:

model	MSE	R^2	RMSE
boosting	1125335	0,5934	1061
randoom forest	1146541	0,6705	1071
ridge	1285977	0,3006	1134
linear reg	1286045	0,3007	1134
lasso	1287764	0,2996	1135
single tree	1309147	0,3523	1144
elastic net	1407666	0,2305	1186
neural network	1831091	0,0023	1353

From the results of validation, Gradient Boosting model performer the best in terms of MSE or RMSE and was therefore chosen as the optimal model for testing and interpretation. Coefficient of determination of the models is not very high. The reason could be low correction of independent variables with the dependent variable.

When outliers were included the tree models performed very well in terms of coefficient of determination (around 0,8). The MSE results, however, were affected by the outliers.

3.Testing result and interpretation

We tested the best model with best hyper-parameters on hold out data and calculated the RMSE, which is 1068,97 czech crowns meaning that every prediction can vary by 1068,97 czech crowns.

```
mean_squared_error(y_test, gb_model_opt.predict(x_test), squared=False)
```

To interpret the model, we used SHAP values to determine the impact of each variable on the output. From the figure below, we can see that overall accommodates variable contributed to the output the most. Quite significant were also review_scores_location and distance_from_city_center_km.

```
explainer = shap.TreeExplainer(gb_model_opt)
shap_values = explainer(x_train)
shap.summary_plot(shap_values, plot_type='bar')
```

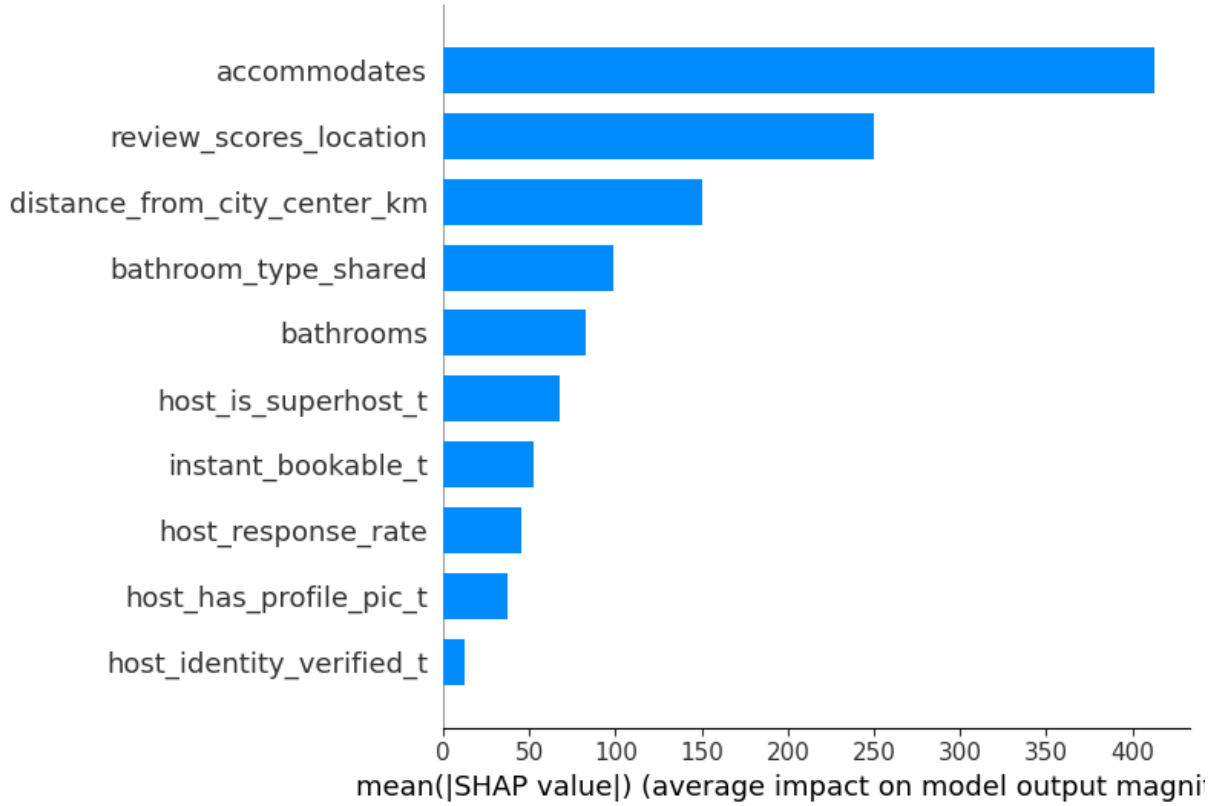

Figure 2 SHAP values bar plot

To see how each variable contributed we can use plot_type = 'dot' which will plot a beeswarm plot.

```
explainer = shap.TreeExplainer(gb_model_opt)
shap_values = explainer(x_train)
shap.summary_plot(shap_values, plot_type = 'dot')
```

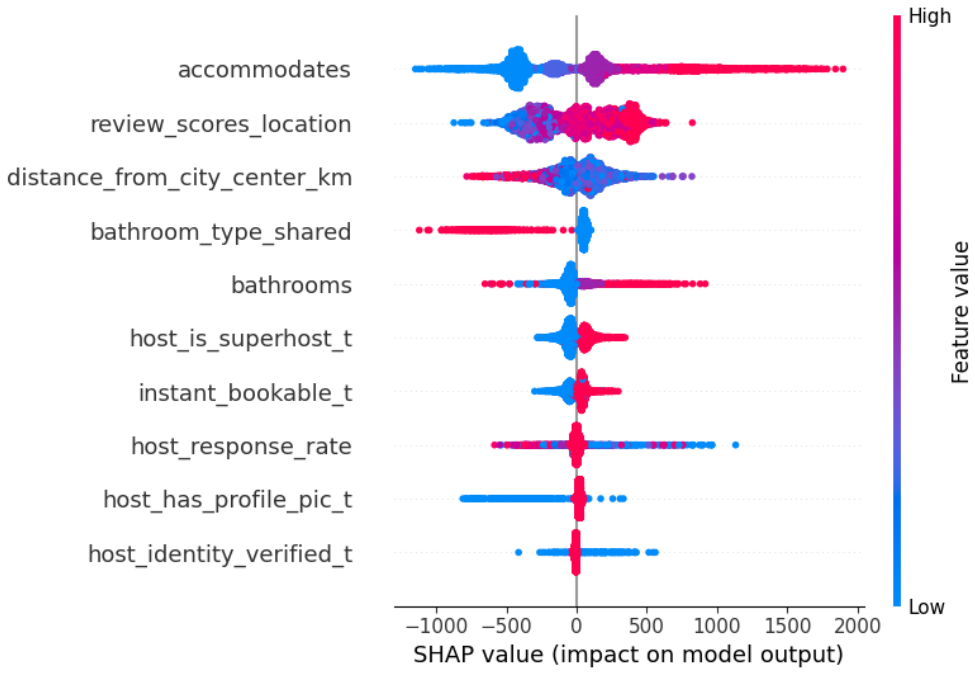

Figure 3 SHAP values beeswarm

From the figure 3 above, we can clearly see the impact of accomodates. Higher values increase the price, while lower values decrease it. A clear relationship can also be seen in bathroom_type_shared, where either shared or private bathrooms clearly decrease or increase the price. But there are also variables such as host_has_profile_pic or host_identity_verified_t does not clearly state the polarity of the impact.

Conclusion

In this assignment we have tried to predict the price per night of Prague's Airbnb rent listings as precisely as possible. We have worked with the dataset *listings.csv.gz*. We started by visualizing and the preparing the data for modelling.

The architecture of the modelling is as follows. 80 % of the dataset is used for training and validation. Validation is done by 5-fold cross validation for most of the models except for neural networks which used hold out test data for validation. During cross validation, different hyper-parameter settings were tried with GridSearchCV and the cross-validation results were returned for the best hyper-parameter setting of a model. The rest 20% was then used for testing the chosen best model with best hyper-parameter settings.

Finally, the best model after the validation was gradient boosting assessed by the lowest MSE was interpreted and test results were calculated. RMSE of the model is 1068,97 meaning that every prediction can vary by 1068,97 czech crowns. The most important variables were accommodates, review_scores_location and distance_from_city_center_km.

References

INSIDE AIRBNB, 2024. *Inside Airbnb: Get the Data | Airbnb*. *insideairbnb.com*. [online]. [cit. 25. 4. 2024]. Available at: https://insideairbnb.com/get-the-data/

MIT NEWS, 2017. *Explained: Neural networks | MIT News | Massachusetts Institute of Technology*. *news.mit.edu*. [online]. [cit. 29. 4. 2024]. Available at: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

AMAZON, 2024. *What is a Neural Network? - Artificial Neural Network Explained - AWS | Amazon*. *aws.amazon.com*. [online]. [cit. 29. 4. 2024]. Available at: https://aws.amazon.com/what-is/neural-network/

Scikit-learn, 2024. 3.3. *Metrics and scoring: quantifying the quality of predictions*. *scikit-learn.org*. Available at: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter