Price Predictions for Airbnb Accomodations in Oslo, Norway

Marei Freitag, Joel Beck
Chair of Statistics, University of Göttingen

March 3, 2022

Abstract

The data provided by the *Inside Airbnb* project contains various information about Airbnb accommodations available in Oslo, Norway, which are used to predict the price of an offer per night. For this purpose, a Deep Learning approach is applied where the focus is on explainability and interpretability of the model. After preprocessing the data, a fully connected feed-forward neural network is implemented to model the desired output variable. In addition, several classical Machine Learning models are implemented to evaluate the performance of the Neural Network. This is followed by an analysis of the results with special attention to the outliers in the data and to which extent they influence the model's performance. In particular, a Variational Autoencoder is used in context of the outlier analysis, which creates two-dimensional latent space representations that allow to illustrate the difficulties that the outliers cause.

Keywords: Deep Learning, Regression, Interpretability, Variational Autoencoder

1 Introduction

The *Inside Airbnb*[1] project (Cox, 2022) was created to quantify the impact of short-term apartment rentals enabled by the Airbnb company. In what follows, we will use this published data to model the prices per night for an Airbnb apartment in Oslo, Norway, using a Deep Learning approach.

The publicly available dataset includes various information about the offered accommodations in Oslo. More precisely, it contains the details of the offer, such as the number of beds, what kind of bathroom is available or in which neighborhood the apartment is located. Furthermore, it provides information about the uploaded pictures and the assigned reviews for each accommodation. The goal of this work now is to establish a Deep Learning model to predict the price of an accommodation per night in this city. Thereby, especially the explainability and interpretability of the implemented model is in focus.

First, to get all the relevant information from the data, we performed feature engineering to mainly extract additional features from the given images of the accommodations and the ratings. From these and the already given data we selected the relevant predictors for our model using different feature selection algorithms. After preprocessing the data, we set up a fully connected feed-forward neural network to predict the prices of the accommodations per night. We also implemented several classical Machine Learning models to be able to evaluate the performance of our neural network.

Subsequently, we considered the results. To do so, we compared the performance of these different approaches and determined the impact of each predictor variable. In a further step, we focused on the sensitivity of our neural network to outliers in the data and how the model is able to deal with them. In particular, we used a Variational Autoencoder to create two-dimensional latent space representations that illustrate the difficulties caused by the outliers.

This work is structured accordingly. In the first chapter, we focus on the data as well as the associated preprocessing steps and provide an overview of the methods used. Then, in the next section, we take a detailed look on the predictive performance of the different models and analyze their interpretability. Finally, we will discuss the results and suggest possible extensions to this project in the conclusion.

2 Methods

2.1 Preprocessing

This section covers all necessary preprocessing steps that build the foundation for all further modeling tasks. Here, we mostly focus on feature *engineering* by extracting information from the image and text data contained in the Airbnb Dataset. However, feature *selection* and transformations of the response variable turned out to be essential components of the

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

preprocessing pipeline and are discussed in Appendix A.2 and Appendix A.3.

In addition, some basic data cleaning steps were necessary right from the start. These mostly consisted of converting data types, splitting text-based variables into more convenient numeric or boolean features and aggregating rare categories of categorical variables into one larger *Other* group to stabilize estimation. In order to use the predictor variables as model inputs later, all features of categorical nature were transformed to dummy variables whereas all numeric features were standardized.

Extending the original feature set with new informative covariates, which are created from (combinations of) existing variables, is a central aspect of every data analysis. Besides many numeric features, the Airbnb data contains *images* as well as *reviews* in raw text form. These data types in particular require some extra care which will be the focus of this section.

2.1.1 Image Processing and Modeling

Besides the metric and text-based features, the Airbnb data contains images of the listed apartments as well as of the corresponding hosts. This section describes how we used these images, with strong focus on the apartment rather than the host pictures, for the purpose of price prediction.

The main idea was to build a Convolutional Neural Network (CNN) that predicts the price solely based on the image content itself and add these predictions as well as the number of available images for each listing to the main feature set. Since there exist multiple images per apartment, the predictions were averaged within each group to obtain an output array of one row per listing that is compatible with the dimensions of the remaining features.

Before the images can be used as model input, they had to be *scraped* from each listing's webpage. Further, some basic transformations were necessary. Details about these image proprocessing steps can be found in Appendix A.1.

Image Modeling

For extracting semantic features out of the images we used the `ResNet18` architecture that was pretrained on the large `ImageNet`[2] database (Russakovsky et al., 2015) and can be accessed in Python via the `PyTorch`[3] library (Paszke et al., 2019). Ideally, due to learning from a large collection of labeled images in a supervised setting, this model is able to extract meaningful features from our own much smaller input data out of the box. As usual in *transfer learning*, the weights of the pretrained model are frozen and the output layer is replaced by a trainable custom layer specific to our needs.

One potential issue arises if the dataset used for pretraining differs significantly from the new custom data: If the pretrained network is very deep, the learned feature representations right before the final layer could be very specific to the output classes of `ImageNet` and might not generalize well to our apartment images.

[2] https://www.image-net.org/

[3] https://pytorch.org/

There are multiple options to handle this issue:

- Out of the vast collection of freely available pretrained models, choose one that is comparably shallow. With roughly 11 million parameters `ResNet18` is much more 'lightweight' than the more commonly used `ResNet50` or `ResNet101`.
- Do not freeze the weights of the pretrained model completely but rather fine tune them during training, i.e. modify *all* weights by backpropagating through the entire network. We did not investigate this option further due to its high computational cost.
- Cut the pretrained model at an earlier intermediate layer with the hope that, at this point, very generic and widely applicable feature representations are learned. In Theory, these features might generalize better to our custom data. In practice, however, this option did not improve our results.

It turned out that a *single* custom layer, mapping from 512 neurons directly to a single neuron which represents the scalar price prediction, was not expressive enough. The performance improved by instead appending a small Fully Connected Network at the end, containing three layers and `ReLU` activation functions.

To ensure that the chosen design is not majorly flawed, we constructed a separate much smaller CNN with only a handful of convolutional blocks as a benchmark model. Although the performance differences were not as large as desired, the pretrained `ResNet` indeed indicated more promising results.

Image Results
Using only the content of the available images, the pretrained `ResNet18` achieved a Mean Absolute Error (MAE) of 579 NOK (approx. 58 Euros) on the validation set. In comparison, the *Null Model* of always predicting the mean price achieved a MAE of 630 NOK without a log-transformation of the price and a MAE of 569 NOK with a log-transformation. Thus, the raw predictive power of the images alone was very small.

However, the *correlation* of the CNN predictions with the true price was 0.41. This indicates some limitations of the correlation as a useful metric on the one hand, but at least positive tendencies of the CNN predictions on the other hand. In fact, the network struggled the most with capturing the wide *range* of prices and almost always predicted values close to the center of the (log-) price distribution.

Although the general idea of categorizing images into price ranges based on image features sounds very appealing, taking a look at the actual input images reveals how challenging this task actually is. Figure 6 in the Appendix displays a random collection of input images with the corresponding true and predicted prices. Considering the difficulty of the task, it is highly doubtful that humans could provide much more accurate predictions.

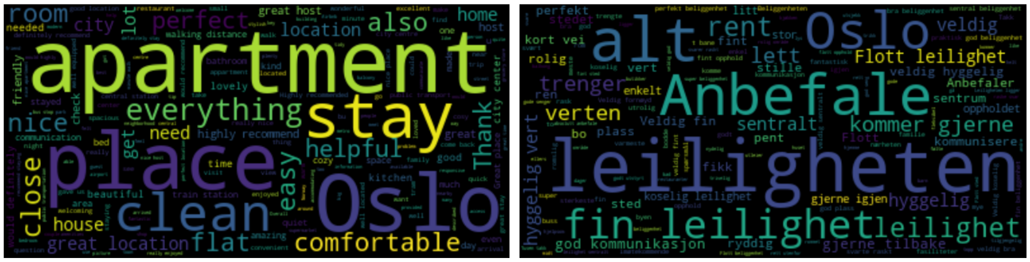

Figure 1: Wordclouds in English and Norwegian

2.1.2 Reviews

In order to extract the most important information from the reviews, a number of analysis steps were performed.

First, we used the `langdetect`[4] package to determine the language of each review. With this information, we tried to get some insight into the internationality of the guests. Since English and Norwegian are the most commonly used languages, we then created two so-called word clouds in these languages to visualize the most frequently used words in the reviews. To just have the important words in this representation, a list of given stop words are used to extract them. What is most striking in Figure 1 is that in both images predominantly positive words are printed. The largest and thus most frequently used words in English are *apartment*, *Oslo* and *place* and further *clean*, *comfortable*, *helpful* and *easy*. Similarly, in the Norwegian plot there are mainly positive expressions like *anbefale* (engl. *recommend*), *fin* (engl. *fine*) and *flott leilighet* (engl. *great apartment*).

The results of the language detection are stored per listing along with the number of reviews per listing, the median length of a review, the number of different languages as well as a list of languages in which the reviews for that apartment were written. The percentages of Norwegian and English reviews are also recorded for each accomodation.

In addition, to obtain a more informed analysis of the reviews, we also performed a detailed sentiment analysis for each review. Sentiment analysis is used to detect the underlying emotion of a text. Therefore, it classifies the text as either positive or negative.

To do this, we used the `transformers`[5] package, more specifically we used the `pipeline`[6] function with the `task` argument set to *sentiment-analysis*, which is used for classifying sequences according to positive or negative sentiments. The used model is DistilBERT (Sanh et al., 2020), a small, fast and light Transformer model, a distilled version of BERT (Devlin et al., 2019), which achieves significantly faster results. Our implementation is inspired by (Selvaraj, 2020).

Finally, we used the sentiment analysis to determine the ratio of negative reviews to the total number of reviews per listing, which is then stored for the subsequent feature selection.

[4] https://pypi.org/project/langdetect/
[5] https://huggingface.co/transformers/v3.0.2/model_doc/distilbert.html
[6] https://huggingface.co/transformers/v3.0.2/main_classes/pipelines.html

2.2 Models

This section describes the statistical models that we used to predict apartment prices based on their feature set. The goal is to construct different models of varying complexity and compare both their in-sample and their out-of-sample performance on the Airbnb data.

At this point in the data pipeline, all preprocessing steps are completed such that all covariates have the correct data type for the prediction models. Whereas the selected classical Machine Learning models described in Section 2.2.1 use input features as `numpy`[7] arrays, the Neural Network in Section 2.2.2 first converts them to *Tensors*, the canonical data type across all popular Deep Learning frameworks.

2.2.1 Classical Models

When attacking the prediction task directly with a Neural Network, there are two questions that we are not able to answer:

1. How do we know if a performance value is *good*?
2. Is fitting a Neural Network appropriate in the first place? Maybe we can get away with a much simpler, more interpretable and, thus, preferable model?

To address these questions, we selected four classical Machine Learning models of increasing complexity from the `scikit-learn`[8] library (Pedregosa et al., 2011) to serve as benchmark models for our custom Neural Net. These include:

- `Linear Regression`: Simple, well understood in terms of underlying theory and highly interpretable.
- `Ridge Regression`: Still very interpretable with a closed form analytical solution, adds one hyperparameter to the equation.
- `Random Forest`: Very flexible model with many hyperparameters determining e.g. the number of regression trees and the tree depth. Can be applied to many contexts and often works 'out of the box'.
- `Histogram-Based Gradient Boosting`: Modern and fast tree-based gradient boosting algorithm. Comes with a large number of tunable hyperparameters. Some of them are similar to the Random Forest parameters, others are more specific to the *Boosting* instead of the *Bagging* approach such as the learning rate. Similar to the very popular `XGBoost`[9] (Chen and Guestrin, 2016) and `LightGBM`[10] (Ke et al., 2017) implementations that regularly outperform Deep Neural Networks in Kaggle competitions on tabular data.

[7] https://numpy.org/

[8] https://scikit-learn.org/stable/

[9] https://xgboost.readthedocs.io/en/stable/

[10] https://lightgbm.readthedocs.io/en/latest/

All models were fitted with *Cross Validation*. Except for the Linear Regression this included (extensive) Hyperparameter Tuning. Since the parameter space for both of the tree-based models is very high-dimensional, we used a *Randomized Search* algorithm (Bergstra and Bengio, 2012) rather than the traditional *Grid Search* approach. This allows for greater variation along more 'important' dimensions while simultaneously keeping the computation time feasible.

2.2.2 Neural Network

In concurrence with the Seminar's topic, the focus of our project is centered around constructing a custom Neural Network that is trained in a supervised setting and learns to map feature combinations to price predictions for the Airbnb data set. The following section highlights the key components of the model's architecture. A brief overview of additional (hyper-)parameters, that were of minor importance, is given in Appendix A.4.1.

The network's design is restricted by two factors:

1. The number of input features in the *first* layer has to equal the number of variables in the feature set.
2. The number of output features in the *last* layer is fixed to 1 since the network directly predicts the price as a scalar quantity.

All intermediate layers and thus both, the *width* and the *depth* of the network, are free to choose. Inspired by empirical findings in the literature we tried several different configurations and finally settled with a fairly simple *block structure*.

Starting from roughly 60 input features, the network width is first blown up to 256 features before steadily decreasing it again to a single output neuron in the final layer. More precisely, the architecture consists of 6 **blocks** with 64, 128, 256, 128, 64 and 8 output features in addition to the linear input and output layers.

The structure of each so-called `Linear Block` is displayed in Figure 2. We now explain the reasoning behind including or excluding elements in this configuration.

Linear Layer
Each `Linear Block` is based on a four-element cycle that is repeated twice. Each cycle starts with a fully-connected layer containing the majority of trainable weights in the network. The first linear layer determines the output dimension of this block, in our case mostly doubling or halving the number of input features. Since both linear layers are immediately followed by a `Batchnorm` Layer, the additional *Bias* term is redundant and thus omitted.

Batchnorm
`Batchnorm` Layers (Ioffe and Szegedy, 2015) often lead to a more well-behaved training process. By means of first standardizing the activations within each minibatch and rescaling them afterwards, the network *learns* a convenient magnitude of the activations throughout training. This tends to be particularly effective for *saturating* activation functions such as

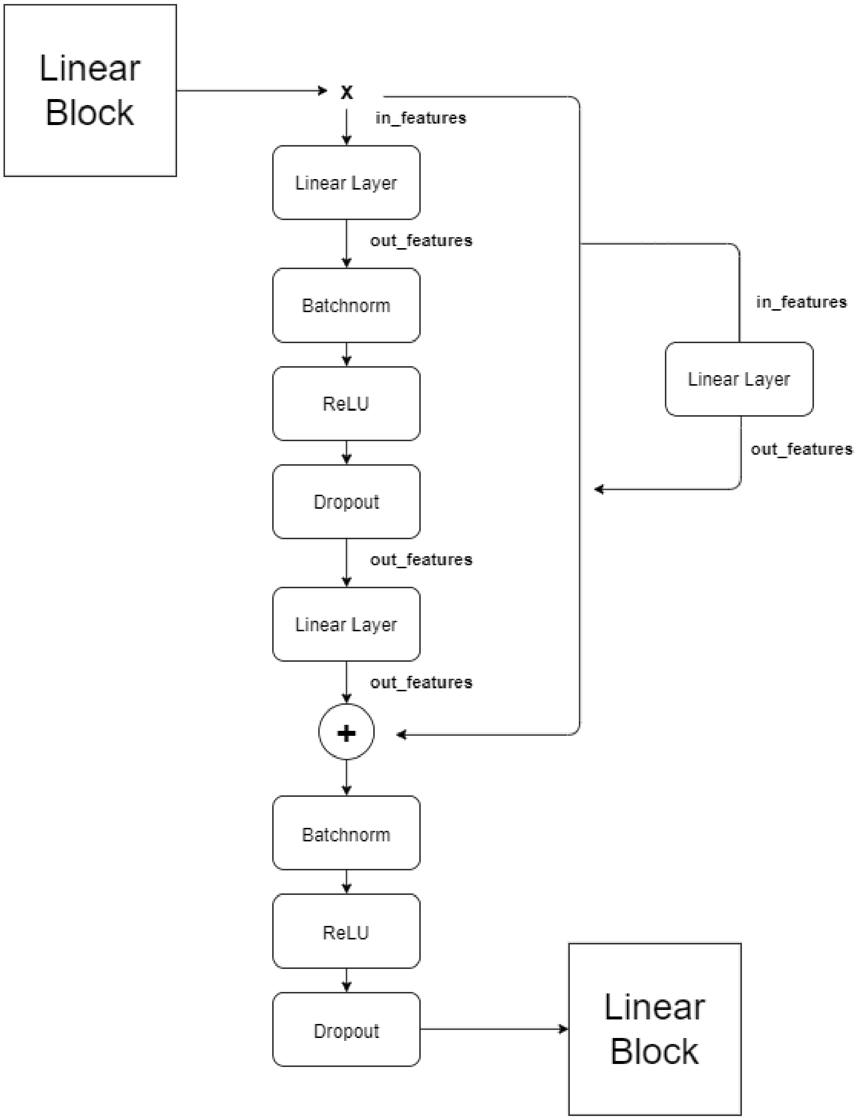

Figure 2: Architecture of each Block in the Fully Connected Neural Network

`sigmoid` or `tanh`. However, positive effects for training have also been observed for the `ReLU` activation function, which is used in our design. Of all individual components of the `Linear Block` structure, batch normalization was the least important and provided only marginal benefits.

ReLU
Out of the numerous options for including nonlinearities into the network and thus expanding the function space that the network is able to learn, we settled for the default choice of the *Rectified Linear Unit* activation. While briefly experimenting with the related `Leaky ReLU` and `Exponential ReLU`, the differences were insignificant and could be easily attributed to other factors such as random weight initialization.

Dropout

The `Dropout` Layer (Srivastava et al., 2014) at the end of each cycle turned out to be the most impactful factor for the network's *generalization* ability. In fact, due to the different behaviour of dropout during training and inference, we could directly control the relative performance on training and validation set by varying the dropout probability p.

More specifically, the network overfitted drastically by setting p to zero. This actually provided the valuable insight that the current function class is flexible enough to model the task at hand properly. Thus, there was no need to artificially expand the network size beyond the current quite compact architecture with roughly $280,000$ trainable parameters, saving computation time as a nice side-effect.

Figure 8 in the Appendix displays the shift in performance metrics on training and validation set when steadily increasing the dropout probability p. While not beneficial for overall performance, dropout delivers a leverage to make the model perform much better on out-of-sample data compared to the dataset that it was actually trained on.

This behaviour can be explained by the *ensemble* interpretation of dropout (Goodfellow et al., 2016): In each epoch, a different subset of neurons is excluded to contribute to training and thus a distinct subset model, nested in the full model, is responsible for the prediction output. If the dropout probability is extremely high, these subset models are simply too different from each other and do *not agree* on a common range of sensible prediction values.

In contrast, during inference, any randomness should be avoided and the network uses all of its neurons. Although there is no influence on the model *weights* in this stage, the model pools all subset models together and *averages out* the diverging effects of each component, ultimately resulting in better performance metrics.

Residual Connections

One special feature of the `Linear Block` architecture in Figure 2 is the *Residual Connection* which adds the block input to the output of the second linear layer within each block. This component was originally implemented with a fairly deep network in mind since *Residual Blocks* introduced by the famous `ResNet` architecture (He et al., 2015) are a very popular tool for stable gradient flow by preventing the issue of vanishing gradients.

As mentioned above, the actual depth of the final model turned out to be limited. Therefore we decided to enable inclusion and exclusion of skip-connections in the network by a boolean parameter which allows for easy comparisons between the model designs. Although the residual component did not have a major impact on performance, it was slightly beneficial in most settings such that there was no reason to remove it from the architecture.

There are two aspects worth noting about the residual connection in Figure 2:

1. The addition of the input with the output of the second linear layer is performed *before* the final `Batchnorm` and `ReLU` layers of this block. This guarantees a nonnegative input in the expected range for the next `Linear Block` since the raw outputs of a fully connected layer are (in theory) unbounded.

2. One constraint of the residual architecture is that all components that contribute to the addition step must share the same dimensions. In case of a fully connected layer dealing with two-dimensional tensors, this amounts to an equal number of features in the second dimension. As illustrated by the annotations in Figure 2, this restriction requires an extra step if the number of input features is different from the number of output features in this particular block. In our case, the dimensions adjustment is performed by an additional linear layer exclusively for the block input x.

3 Results

3.1 Evaluation Metrics

Whereas the Mean Squared Error Loss was used for *training* the Neural Network due to its convenient differentiability with regards to backpropagation, we preferred the Mean Absolute Error and the R^2 value for *evaluating* the different models. The MAE measures absolute deviations between true and predicted price and is therefore interpretable on the same scale as the original data. While the R^2 certainly has some inherent flaws, it provides a scale-independent and highly interpretable measure of overall model fit.

When using the log-price for model fitting, all error metrics were computed on the *original* price scale to obtain an interpretable MAE value. More specifically, all computations were performed *after* backtransforming the fitted values for the log-price to the raw price scale. Since the results for the MAE and R^2 are highly dependent on the exact transformation procedure, they are in particular *not comparable* with approaches that use a different evaluation strategy.

To illustrate this difference in more detail, denote the error metrics as functions of the true and predicted values:

$$\begin{aligned} MAE(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \\ R^2(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \end{aligned}$$

With this notation, we computed $MAE\left(\mathbf{y}, \exp\left(\widehat{\log(\mathbf{y})}\right)\right)$ which is of course not identical to $\exp\left(MAE\left(\log(\mathbf{y}), \widehat{\log(\mathbf{y})}\right)\right)$, i.e. backtransforming the error on the log-scale. To stay consistent, we similarly computed $R^2\left(\mathbf{y}, \exp\left(\widehat{\log(\mathbf{y})}\right)\right)$ on the original scale with the backtransformed fitted values instead of $R^2\left(\log(\mathbf{y}), \widehat{\log(\mathbf{y})}\right)$, where the latter usually leads to a better score.

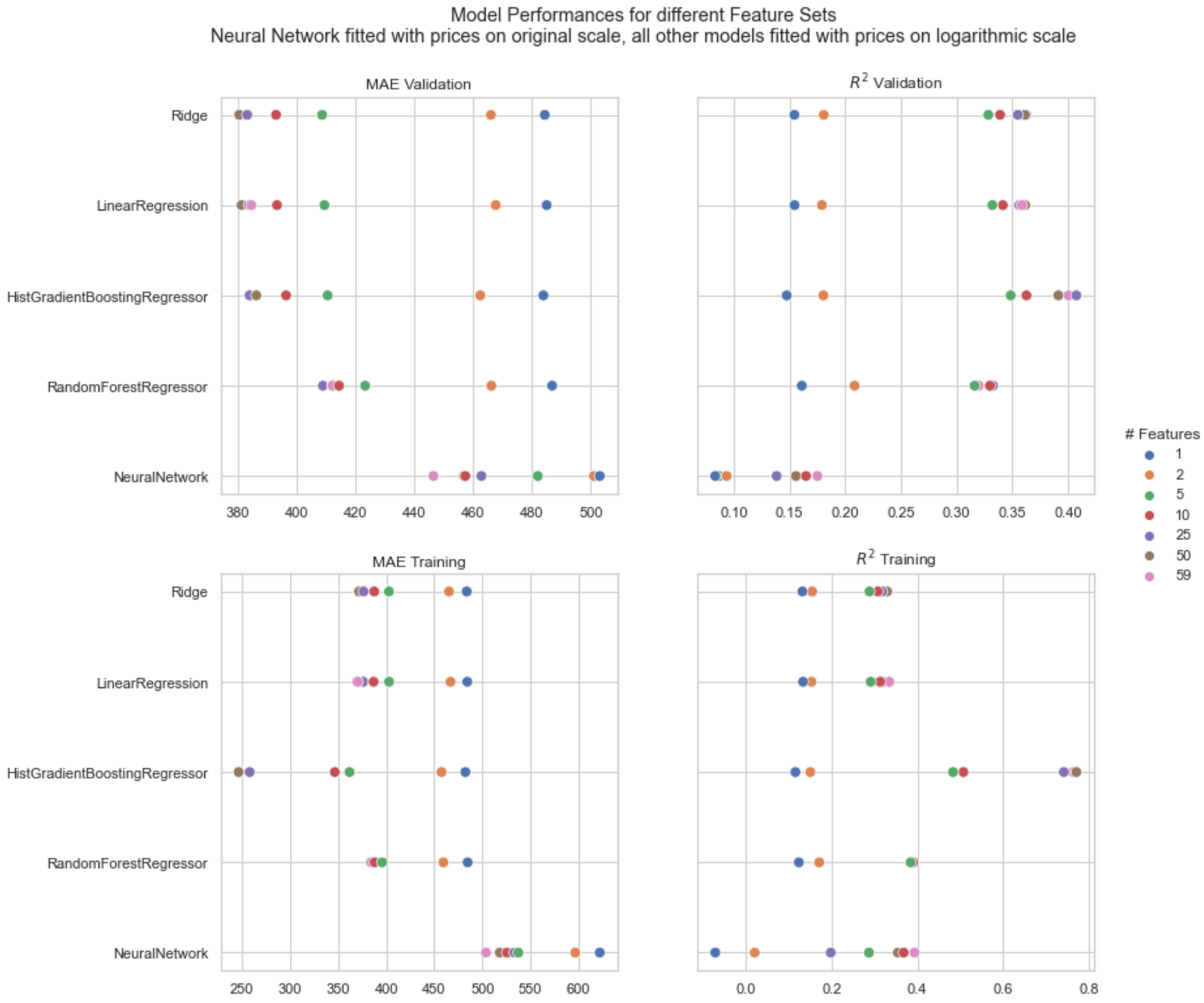

Figure 3: Performance Comparison of Classical Models with Neural Network

3.2 Predictive Performance

Figure 3 shows performance metrics for all fitted models on both, the training set and the validation set. The different colors indicate how many features were selected by the RFE algorithm for this particular fit out of 59 total predictor variables. Unsurprisingly, one or two features (in case of the RFE the number of bedrooms and the (number of) accomodates) provide too little information to model the task appropriately.

However, including merely 5 features (in this case adding the 30 day availability, the indicator variable for the *Frogner* neighbourhood and, notably, the price predictions from the Convolutional Net based on the image data) results in a very competitive performance for most models on the validation set.

The two subplots on the left display the MAE and tell a very similar story to the subplots on the right that show the R^2: Generally, more features lead to better performance on training and validation set. In case of the flexible Gradient Boosting algorithm and, to some minor extent, for the Random Forest, a high number of input features lead to *overfitting* such that the performance on the training set is far superior to the values on out-of-sample data. In contrast, models with few parameters such as Linear Regression or Ridge Regression generalize very well to the validation set with virtually no performance drop.

One exception to this rule is the highly complex Neural Network which appears to perform *better* on out-of-sample data. This phenomenon can be explained by the different behaviour of Dropout and Batchnorm layers in training and inference mode on the one hand and the presence of price outliers, which is discussed in Section 3.3.2, on the other hand.

When comparing the models *within* each subplot, all classical Machine Learning models perform similarly on the validation data. This finding emphasizes two aspects:

1. The prediction task does **not** require overly complex models and linear models perform just fine.

2. We can expect predictions within a distance of roughly 400 NOK (40 Euros) on average to the true price. Further, a R^2 value of around 0.4 is the best we can hope for. These statements hold for fitting on the *entire* training set, Section 3.3.2 reveals how the performance radically improves when excluding some observations.

In comparison to all `scikit-learn` models, our custom Neural Net appears to underperform. We have to keep in mind though, that evaluation on the validation set is overly optimistic for those models whose hyperparameters were tuned on this data during cross validation. The best simulation of performance on truly unseen data is thus provided by the *test* set that remained untouched up to this point.

Model	MAE	R^2
Linear Regression	404.709	0.298
Ridge	405.932	0.294
Random Forest	444.166	0.268
HistGradientBoosting	412.243	0.387
Neural Network	402.24	0.333
Top2 Average	404.848	0.296
Top3 Average	399.315	0.343
Top4 Average	404.206	0.332
Top5 Average	408.116	0.27

Table 1: Test Set Performance of Classical Machine Learning Models, our custom Neural Network and Ensemble Predictions

Table 1 compares test set metrics for the best performing models on the validation set within each function class, i.e. the Neural Net version with all 59 features and the Random Forest model with only 25 features were selected. In addition, we added some simple ensemble models that predict the average estimate from the top 2, 3, 4 or all five models, respectively.

The Random Forest model indeed seems to have overfit on the validation set due to hyperparameter tuning, such that it now performs worst. Quite surpringly, the Neural Net shows a significant performance boost on the test data and is now very competitive with all other models. One plausible reason for this phenomenon is again related to outliers and explained in Section 3.3.2.

Averaging predictions from multiple independent algorithms indicates promising results: The Top 3 ensemble achieves the lowest out-of-sample MAE of slightly below 400 NOK. As indicated by Figure 3 this particular ensemble averages the price predictions of the following three model configurations:

1. Linear Regression with 50 selected features.
2. Ridge Regression with 50 selected features and a penalty weight of $\alpha = 14$.
3. Histogram Gradient Boosting Model with all 59 features, a *learning rate* of 0.09, a *maximum tree depth* of 18, a *maximum number of leaf nodes* within each tree of 30 and a *minimum number of samples* per leaf node of 4.

3.3 Understanding and Interpretation

Interpreting the results of high-dimensional and complex statistical models is notoriously difficult. This section approaches the challenge from two opposite angles.

First, we reduce the *complexity* by fitting a simple and well-understood Linear Regression model with the same features that were selected for the best-performing Neural Network and analyze the coefficients.

Second, we reduce the *dimensionality* by leveraging a different Neural Network and visualize the results in two-dimensional space.

3.3.1 Feature Importance

One idea to draw conclusions about which features were most important for the Neural Network is to start with a random noise input and leverage Gradient information to optimize for new artificial inputs that maximally or minimally activate the output neuron, i.e. leading to very low or very high price predictions. We chose a conceptually simpler approach and analyzed the feature importance of an auxiliary Linear Regression model instead based on the idea that important features for a simpler Linear Regression model might also be important for a more complex Neural Network.

Therefore we preselected 25 of the network's input features with the `RFE` algorithm, which is introduced in Appendix A.2, and compared the coefficient *magnitudes*. As noted before, all predictors were standardized at the beginning of the data pipeline such that this comparison is meaningful. The results are shown in Figure 4.

It is worth noting that the two most important features based on various different feature selectors within the `scikit-learn` library were always the number of *bedrooms* and the

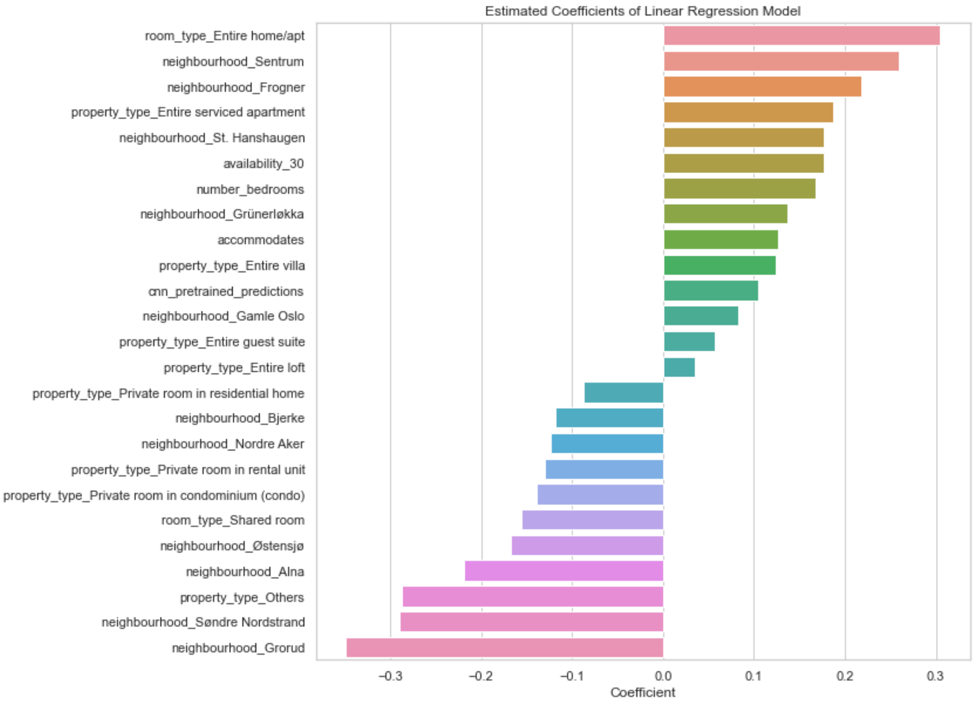

Figure 4: Estimated coefficients of a Linear Regression model for 25 preselected features

(number of) *accomodates* in this order, both measuring the apartment's *capacity*.

The coefficient plot, however, is dominated by categorical features: In agreement with human intuition, the *room* type, the *property* type and the *neighbourhood* strongly influence the predicted price. Unsurprisingly, the property types *entire home* and *entire villa* are connected with high prices, whereas the room type *shared room* correlates with cheaper apartments. The price predictions from the CNN fitted on the image data indicates a significant positive effect, conditioned on all other selected features.

When analyzing the *marginal* effect of neighbourhood on price by e.g. ordering the neighbourhoods according to their median price, this order is nearly identical to the ranking in Figure 4 with *Frogner* at the top and *Grorud* at the bottom. Interestingly, apartments in the *Sentrum* (central area) have a large positive coefficient in the regression, yet the second to lowest median price. This finding indicates the presence of *confounders*: The city center might plausibly be connected to fewer rooms and smaller apartment sizes overall pulling the median price down. These correlations are accounted for in the regression model but not in the naive bivariate analysis.

3.3.2 Sensitivity to Outliers

Table 2 shows performance metrics for the Neural Network fitted on a subset of the data in the validation set. As indicated by the first column, the dataset was reduced by consecutively cutting off observations with the highest prices. More precisely, the first row refers to the entire data whereas the last row excludes the top 10% most expensive apartments.

Quantile Threshold	MAE	R^2
0.0	443.35	0.16
1.0	337.59	0.51
2.5	282.17	0.53
5.0	240.57	0.54
10.0	214.76	0.49

Table 2: Mean Absolute Error and R^2 value of the Neural Network on the validation set after removing the highest quantiles of the price distribution from the data set

By omitting just the top 1%, the MAE reduces by over 100 NOK (about 10 Euros) and the R^2 rapidly jumps to over 0.5. Importantly, this high influence of outliers is **not** diminished by a transformation of the response distribution like a log-transformation.

The values in Table 2 also suggest an explanation why the Neural Network performance increases from the training to the validation set and then again from the validation to the test set: When partitioning the entire data into training, validation and test set, most of the extreme outliers were assigned to the training set since it is the largest of the three. The validation and test set are similar in size, so it just happened by chance that the test set contains fewer price outliers which drive the error metrics up compared to the validation set, even though the model is exactly the same.

In contrast, the classical Machine Learning models were fitted with the more robust cross-validation approach. Hence, extreme outliers were contained in the validation fold only once out of multiple evaluations. By ultimately averaging the error metrics across all folds their impact is diminished and we cannot detect a strong performance boost between training and validation set.

Based on Table 2 it is clear that the Neural Network lacks the ability to capture the entire price range accurately. In fact, the predicted prices are much more concentrated around the median price than their observed counterparts. Table 2 Yet, in theory, the Neural Network should be flexible enough to approximate **any** function reasonably well. This raises the question why the Network is not able to learn during training to map the observations with the highest prices to corresponding high price predictions?

Recall that the Network only sees the feature *inputs* during training. Thus, in order to discriminate between price outliers and non-outliers, their feature combinations must be *separable* in the full high-dimensional feature space. One idea to analyze locations in the input space is to compute Nearest Neighbours for highly-priced listings e.g. via simple Euclidean distances or Cosine similarities. This approach, however, is difficult to generalize beyond pairwise comparisons.

To get a more global view of the feature space instead, we want to visualize it. Since the full high-dimensional space cannot be visualized immediately, we approximate it with a two-dimensional *embedding* or *latent space.* If the price outliers are clearly separated from the remaining observations in this embedded space, the network is faced with a feasible

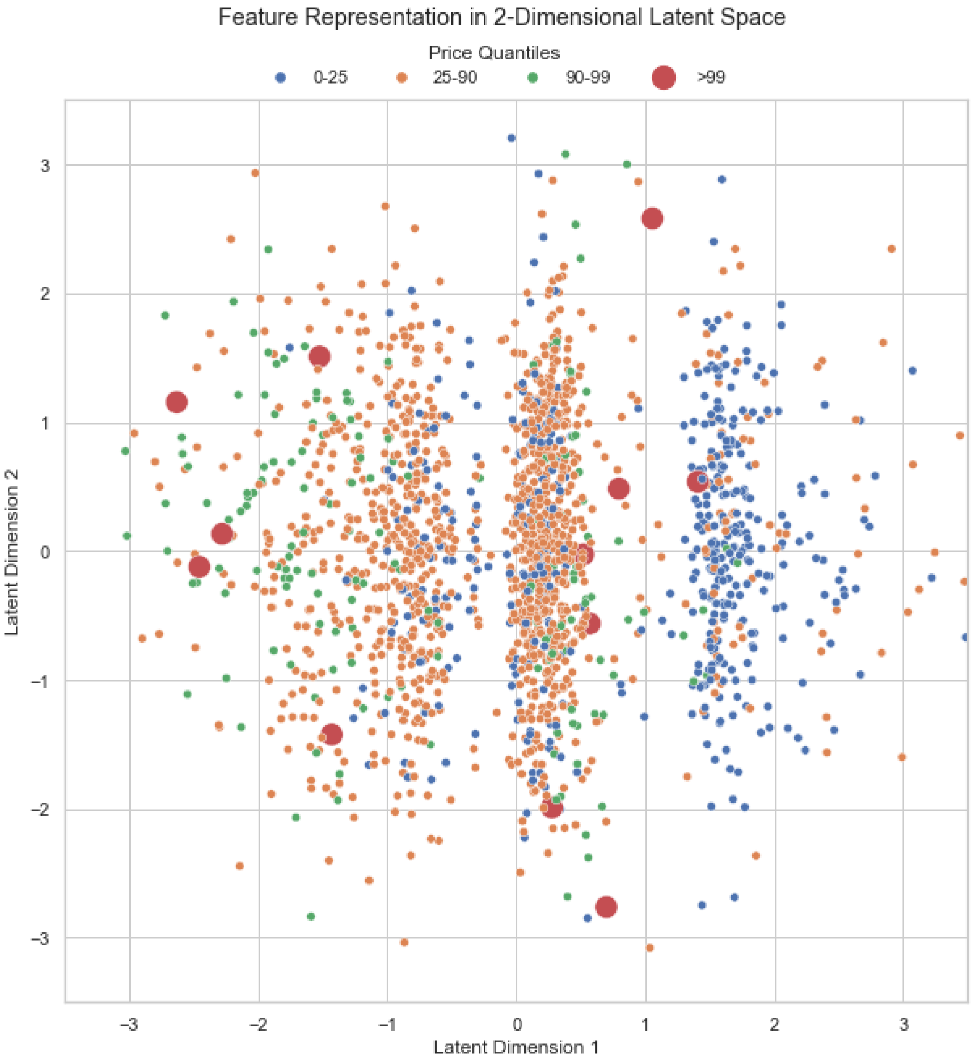

Figure 5: Feature Representation in a two-dimensional latent space embedding

task. If, however, the outliers are located in the *center* of the latent distribution, there is little hope to discriminate the most expensive apartments from any of their potentially much lower priced embedded neighbours. Keep in mind that all conclusions of the following analysis are based on the assumption that the two-dimensional embedding approximates the full feature space sufficiently well!

Modern Machine Learning methods provide a large toolbox for low-dimensional embeddings. Since the project is mainly focused on Deep Learning, we decided to use a *Variatonal Autoencoder* (Kingma and Welling, 2014). In contrast to deterministic Autoencoders, the VAE contains an additional loss term apart from the usual reconstruction loss that pulls the encoded latent space distribution towards an isotropic multivariate Gaussian distribution. For this reason, latent space visualizations of the VAE appear to be more spread out and output classes, such as price segments in our case, tend to be easier to identify.

Figure 5 visualizes the two-dimensional feature embedding. Already a *single* dimension, here the horizontal axis, seems so be sufficient to categorize the observations into price segments. Cheaper apartments in blue are clustered on the right, medium prices in orange in the middle and higher prices in green are located on the left.

However, the Top 1% of highest prices, the large red circles, do not seem to build their own cluster, they are spread out across the entire feature space. When the Network has to predict prices for some of the outliers in the *center* of the latent space, they appear indistinguishable to their medium or even low-priced Nearest-Neighbours based on their feature set. Hence, the network likely maps these outliers to comparably low price predictions resulting in large residuals with a high influence on the MAE and the R^2 value.

Assuming that the two-dimensional picture appropriately reflects the situation in the full feature space, there are two possible explanations for this lack of separability:

1. The collected data is simply not rich or expressive enough to capture all factors that contribute to very high prices.
2. Some apartments are listed at a price that does not represent their true value.

In the latter case, it can be argued that the entire task of predicting Airbnb prices is questionable in the first place: In order to draw connections from features to outputs, the process implicitly assumes observations whose listed price can be justified and explained by characteristics of the joint feature distribution.

Moreover, discriminating between these two options is nontrivial. One idea is to use features like *availability* or the *number of reviews* to approximate demand and classify price outliers with low demand as overpriced. However, it is entirely possible that an extremely expensive observation with very low demand is listed according to its true value, yet nobody is willing to pay such a high amount for an accomodation, regardless of its quality. Since such accommodations might very well be contained in a new data during deployment, the model should have learned to handle them appropriately during training. As a consequence, a blind removal of such outliers is difficult to justify and narrows down the set of feasible target distributions.

3.4 Price Predictions on Munich Dataset

To test the generalization ability of our implemented models, we applied them to the Airbnb data for *Munich*, which is also published by the *Inside Airbnb* project. This data set includes roughly twice as many accommodations as the previous data set for Oslo.

As expected, the flexible models such as Gradient Boosting and the Neural Network benefit most from the larger number of observations. The overall performance, however, is *not* significantly better compared to the original Oslo data: The Neural Net improved marginally from a MAE of 44.3 Euros for Oslo to 42.5 Euros for Munich on the validation set. One exception was the Gradient Boosting Model that now shows by far the best test set performance with a MAE of 32.8 Euros and a R^2 of 0.453. To further improve the Neural

Quantile Threshold	MAE	R2
0.0	42.49	0.31
1.0	35.32	0.44
2.5	30.26	0.42
5.0	25.2	0.45
10.0	22.94	0.4

Table 3: Mean Absolute Error and R^2 value of the Neural Network on the validation set after removing the highest quantiles of the price distribution from the Munich data set

Network performance, a wider or deeper model architecture could be considered that is specifically designed for the larger data set.

Similar to Oslo, the most important predictors of the Munich data set were *Property Type*, *Neighborhood*, and *Accommodates*. Furthermore, the new data set also suffers from outliers whose influence is quantified in Table 3. Excluding consecutive percentiles of the price distribution prior to fitting the model again leads to a significant improvements in MAE and R^2. In contrast to Oslo, however, we can not detect a huge jump from using the entire data to cutting off the top 1% of highest prices.

4 Conclusion

In order to predict Airbnb prices in Oslo, we constructed our own Deep Neural Network and compared its performance to a collection of well-established Machine Learning algorithms. Since we extracted numeric features out of the unstructured image and text data as part of the preprocessing pipeline, the input data fed into the models was of tabular nature.

In summary we can conclude that price prediction with a small tabular data set does not require overly complex models. Therefore, if we first look at the results of the classical Machine Learning approaches, an equally convincing performance of the linear and non-linear models can be observed. However, in our analysis the linear models seem to generalize even better to the validation set. Thus, classical Machine Learning models and in particular highly interpretable linear models indicate a competitive predictive performance on out-of-sample data.

The Neural Network, in contrast, seems to provide the worst results at first glance. Although further analysis revealed that the Deep Learning approach shows the best generalization ability overall when taking outlier effects into account, the lowest test set error is achieved by a straightforward ensemble of multiple individual predictive models.

Further, we emphasized the impact of using different feature set sizes both on the training and validation set by using several feature selection algorithms. We then focused on the influence of the existing outliers in the data set. Here it was very clear to see how drastically the results of the Neural Network improved when the top 1% of prices were excluded prior to model fitting. To get a better understanding of this behavior we implemented a

Variational Autoencoder that encodes the input feature space into a two-dimensional latent space. These embedded feature representations are then used to visualize and explain the sensitivity of the network's predictions with respect to outliers in the price distribution.

In addition, there are various options to extend our work. One particularly appealing idea for understanding the Neural Net's behaviour is the construction of *adversarial examples*. In the regression context one could try to leverage steep areas in the high-dimensional loss surface such that small perturbations of each input feature result in exploding price predictions.

Bounding the magnitude of each perturbation by a constant might not be directly transferable to regression tasks in presence of categorical features. As an alternative to the traditional approach borrowed from image classification, which is often based on the *Fast Gradient Sign Method* (Goodfellow et al., 2015), a well-behaved Linear Regression model could again be used as a benchmark model. Then, any popular gradient *ascent* algorithm can be used to find feature combinations that maximize the distance between the Linear Regression prediction and the Neural Net prediction. Since this approach does not naturally bound the input changes, the resulting loss function has to be heavily regularized to prevent pathological solutions.

A Appendix

The source code that produced all results, figures and tables can be accessed through our GitLab Repository[11]. Please follow the instructions in the `README` file in order to use our code for your own purposes.

A.1 Image Processing and Modeling

A.1.1 Webscraping

In the first step, the raw image links provided by the data set had to be converted to a convenient image format that the neural network is able to work with.

Therefore, we first used the `requests`[12] library to get the HTML Source Code of each listing's website. Next, the `beautifulsoup`[13] library served as HTML parser to find and extract all embedded weblinks that lead to images located on the front page of the listing's website. With this strategy we could extract the first 5 images for each apartment (if 5 or more were available) that were accessible through the front page source code.

Finally, we used the `requests` module again in combination with the `pillow`[14] package to decode the source content of all image adresses into two dimensional images.

A.1.2 Image Transformations

Before feeding these two dimensional images into the model, we performed some further preprocessing steps.

One very common technique when dealing with images is *Data Augmentation*. In a classification task, data augmentation is used to expand the training set and simultaneously improve generalization. However, this approach is not immediately transferable to a regression context since we have to guarantee that the label (i.e. the price) remains unchanged after the image transformation. Thus, we decided against standard transformations such as rotating the images or manipulating the color composition.

We did use image *cropping*, however, which, in our opinion, is one of the few applicable augmentations in regression contexts. After resizing all images to 256×256 pixels we randomly cropped a square area of 224×224 out of each image in the training set and cropped an equally sized area out of the image center in the validation set to avoid any randomness during inference.

As a final step, all images were normalized for each color channel separately. In case of the pretrained `ResNet`, we used the same values for normalization that were used during

[11] https://gitlab.gwdg.de/joel.beck/airbnb-oslo

[12] https://docs.python-requests.org/en/latest/

[13] https://beautiful-soup-4.readthedocs.io/en/latest/

[14] https://pillow.readthedocs.io/en/stable/

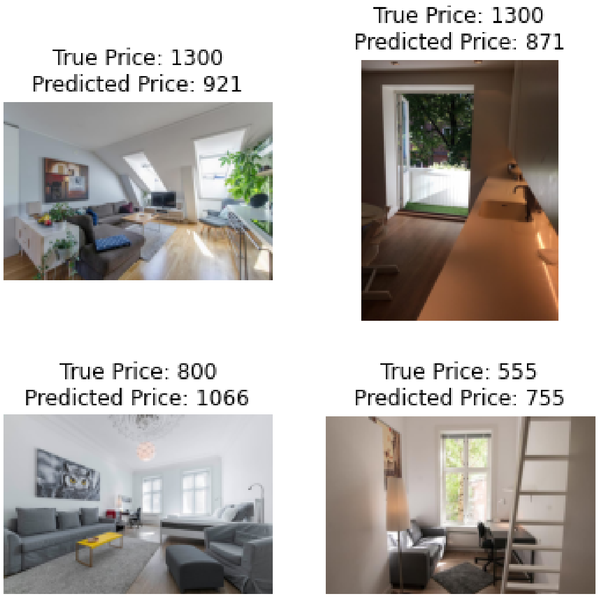

Figure 6: Images from Airbnb Apartments with true and predicted Prices

training on `ImageNet`. The mean values and standard deviations for each color channel are provided in the `PyTorch` documentation[15].

A.2 Feature Selection

Not all features out of the new combined and extended feature set are equally valuable to the model in terms of predictive power. In fact, some of the features could not even be transformed to meaningful predictors due to containing e.g. *id* information. Others were completely redundant in presence of a combination of original and/or manually constructed features and were thus dropped from the feature set.

Before starting the modeling, we intended to reduce the feature set even further to avoid strong correlations among the input predictors and possibly improve generalization performance to out-of-sample data. Thus, we agreed on a two-step strategy.

First, we manually selected features based on three criteria:

1. The variable in consideration must have some connection to the apartment price based on human intuition and background knowledge. For instance, we included variables containing information about the apartment's *size* such as the number of *accomodates*, the number of *bathrooms* and the number of *bedrooms*.

2. There has to exist some correlation with the price in a bivariate visualization, e.g. a *barplot* in case of categorical predictors or a *scatterplot* in case of numeric features.

3. Since our dataset was comparably small with roughly 3000 observations, we had to

[15] https://pytorch.org/vision/stable/models.html

pay special attention to missing values. Therefore, each variable whose missing values could not be imputed in a meaningful and uncontroversial manner was either selected or dropped based on the trade-off between potential data reduction and additional predictive value.

Up to this point, the feature selection process was solely based on bivariate relationships and self-chosen (arguably arbitrary) selection criteria. There was a high chance of excluding important predictors that primarily shine in combination with other variables rather than on their own.

Therefore, in a second step, we fitted an auxiliary Linear Regression model with *all* available features (except variables like the *picture url* that were not convertible to any useful format) and analyzed the absolute magnitude of the coefficients. Since all variables are standardized, these magnitudes are within the same range as well as unit-independent and can thus be compared.

Of course, even this approach is imperfect, as several highly correlated characteristics that have a strong influence on price could lead to rather small estimated individual coefficients due to their joint predictive/explanatory power. To circumvent this potential issue, we additionally used the *algorithmic* feature selectors provided by the `scikit-learn` library, which are (ideally) able to separate the effects of highly correlated features and, consequently, select only a small subset of them.

Thereby we focused on two algorithms:

- *Principal Component Analysis*: Reduces dimensionality with the additional benefit of creating *uncorrelated* linear combinations of existing features.
- *Recursive Feature Elimination*: Selects a subset of original features by leveraging an external estimator (in our case a *Support Vector Regressor*). We chose the `RFE` algorithm for the main analysis, since the selected features can be immediately interpreted and the performance on the selected feature set was slightly better compared to `PCA`.

The influence of the feature selector on the predictive and particularly the generalization performance is discussed in Section 3.

A.3 Price Distribution

One key aspect of exploratory data analysis is investigating the distribution of the response variable. In our case, the price distribution is highly right-skewed with a few very expensive listings pulling the mean and median of the price distribution further away from each other.

Some statistical models, such as *Linear Regression*, tend to perform better when the outcome distribution is symmetric and approximately normal, whereas some very flexible algorithms like *Neural Networks* do not make any distributional assumptions and are capable of modeling any kind of distribution accurately. Figure 7 illustrates that an approximate normal distribution can be achieved with a simple logarithmic distribution.

Whereas *all* of the classical models benefitted from the log-transformation resulting in lower error metrics, this was not the case for the Neural Network we used. In fact, training turned out to be more challenging, since the *magnitude* of the losses, that were computed by comparing true and predicted price on the logarithmic scale, was drastically reduced. As a consequence the Network's gradients and the weight updates were smaller within each minibatch. This issue could be mitigated to some extent with a larger learning rate. Yet, in contrast to the untransformed version, the model still suffered from *vanishing gradients* and *loss plateaus*.

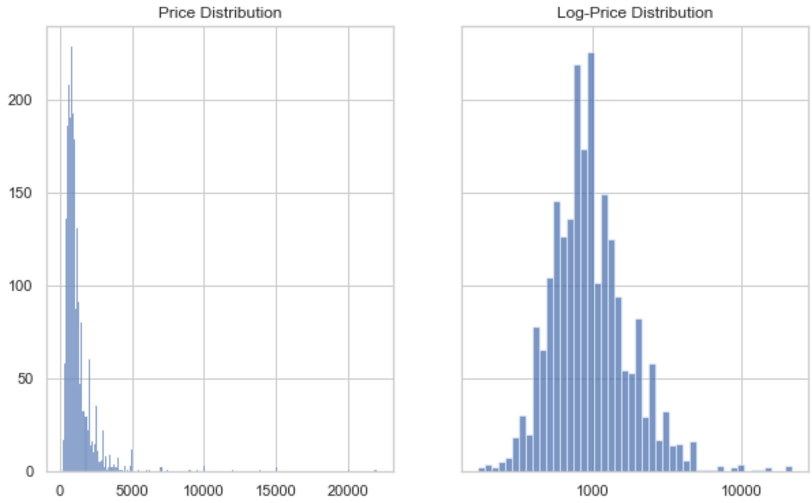

Figure 7: Distribution of Apartment Prices on original and logarithmic Scale

A.4 Neural Network

A.4.1 Further (Hyper)-Parameters

Optimizer and Weight Decay

We used the popular `Adam` algorithm (Kingma and Ba, 2017) for training the Neural Network. Compared to vanilla gradient descent (with optional momentum), the loss curves during the training process looked much smoother. Moreover, while pure gradient descent slightly benefitted from an additional penalty, neither the classical L_2 weight decay nor any other weight regularizer indicated improvements for training with `Adam`.

Loss Function

For the regression task of predicting the continuous price variable, we chose the standard continuously differentiable *Mean Squared Error* loss function. This comes with the additional nice property that, under the assumption of a conditional normal distribution of the price with constant variance, i.e. $y_i \mid \mathbf{x}_i \sim \mathcal{N}(\mu_i, \sigma^2)$, the model which *minimizes* the MSE-Loss simultaneously *maximizes* the Gaussian Likelihood and, thus, provides a *probabilistic* interpretation.

Learning Rate
Throughout training, we kept the learning rate fixed at 0.01. Using a larger learning rate at early stages of training showed promising effects such that we experimented with different learning rate *schedulers*, either decreasing the learning rate in fixed intervals linearly or exponentially or decreasing the learning rate whenever the validation loss seemed to stagnate for some given window.

However, the benefits of faster loss decay at the beginning of training were outweighed by too small learning rates at later stages in the training loop. As a consequence, all schedulers resulted in stagnating loss much faster than training with a constant learning rate from start to end.

Moreover, since computation time was not expensive due to the compact network size, we compensated the slower initial progress (compared to training with a learning rate scheduler) by simply training the model for more epochs, which ultimately lead to a lower training and validation loss. Thereby, we stored the model weights at the point of best performance on the validation set.

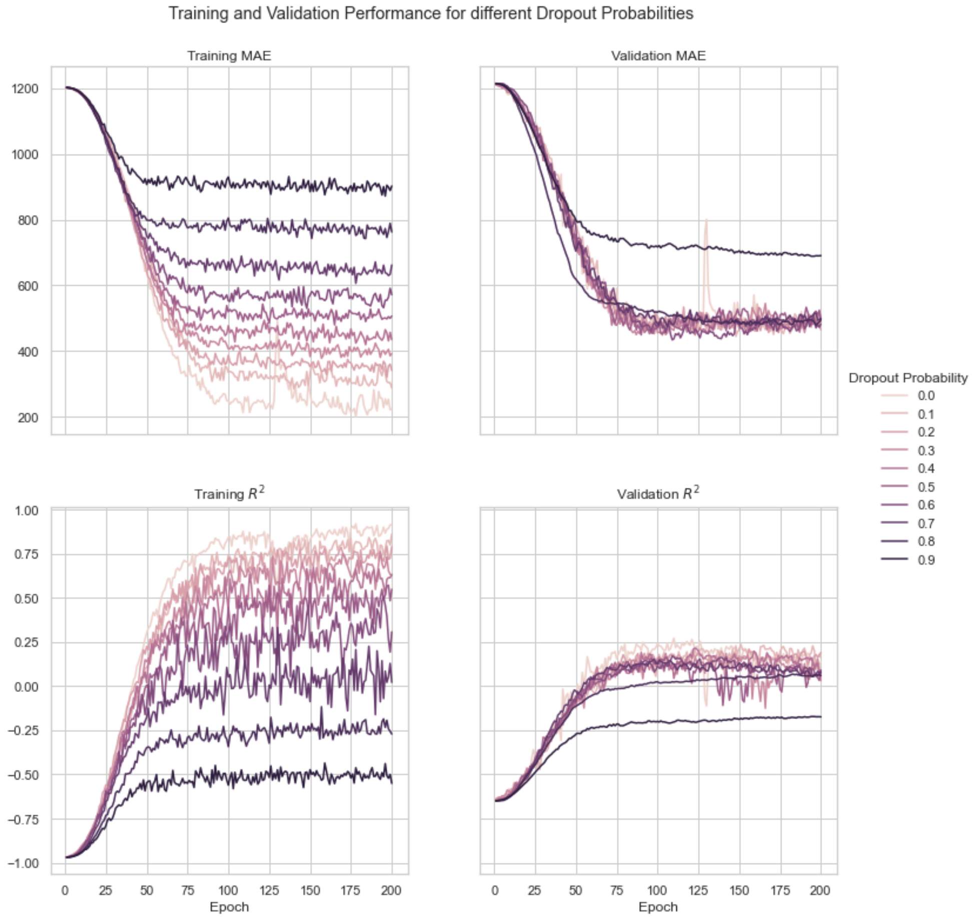

Figure 8: Impact of Dropout Probability on Training and Validation Performance

References

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.*, 13:281–305.

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

Cox, M. (2022). Inside Airbnb.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Kingma, D. P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

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

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575 [cs]*.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs]*.

Selvaraj, N. (2020). A Beginner's Guide to Sentiment Analysis with Python.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.