Notes to Preprocessing

1 Feature Engineering

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. Besided many numeric features the Airbnb data contains *images* as well as *reviews* in raw text form. These data types in particular require some preprocessing steps which will be the focus of this section.

In addition, some basic data cleaning steps were necessary. 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.

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 (while focusing on the apartment pictures) for the purpose of price prediction.

The main idea was to build a Convolutional Neural Network 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 afterwards within each group to obtain an output array of equal length to all remaining features.

1.1.1 Webscraping

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

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

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

1.1.2 Preprocessing

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 contrast to classification tasks however, where Data Augmentation is used to expand the training set and improve simultaneously generalization, this approach is not immediately transferable to a regression context since we have to guarantee that the label (i.e. the price) remains unchanged for each 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.

At a final step all images were normalized, separately for each color channel. Ic case of the pretrained model explained in the next section we used the same values for normilization that were used during training on the large `ImageNet` database. The mean values and standard deviations for each color channel are provided in the `PyTorch` documentation.

1.1.3 Modeling

As mentioned above, we used a pretrained Convolutional Neural Network for modeling. 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 pretraine 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 from the new custom data: If the pretrained network is very deep, the learned features before the final layer could be very specific to the Output Classes of `ImageNet` and not generalize well to our images.

There are multiple options to handle this scenario:

- Out of the vast collection of freely available pretrained models, choose one that is comparably shallow. We chose `ResNet18` with roughly 11 million parameters.
- 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 before the last layer with the hope that, at this point, very generic and widely applicable features of images are extracted. These features might in theory generalize better to our data. In practice, however, this option did not improve our results significantly.

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

To ensure that the chosen design is not majorly flawed, we constructed a separate much smaller Convolutional Neural Network 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.

1.1.4 Results

Using only the content of the available images, the pretrained `ResNet18` achieved a Mean Absolute Error of 579 NOK (approx. 58 Euros) on the Validation Set. In comparison, the *Null Model* of always predicting the mean price achieved an 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 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 2 displays a random collection of input images. Considering the difficulty of the task it is actually highly doubtful that humans could provide much more accurate predictions.

2 Reviews

In order to extract the most important information from the reviews, we have performed the following analyses.

First, we used the *langdetect* package to determine the language of each review. With this information, we tried to get some insights about the internationality of the guests. Since English and Norwegian are the most commonly used languages, we then created two so-called word clouds in this languages to visualize the most frequently used words in the reviews and to give us a short overview of the given ratings. 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 the plots (s. Fig. 1) is that in both predominantly positive words are printed. The largest printed and thus most frequently used words in English are *apartement*, *Oslo* and *place* and further *clean*, *comfortable*, *helpful* and *easy*. Also in the Norwegian plot there are mainly positive ex-

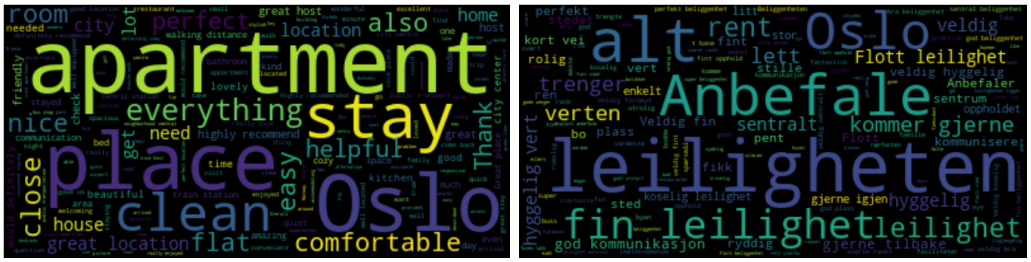

Figure 1: Wordclouds in English and Norwegian

pressions like *"anbefale"* (engl. recommend), *"fin"* (engl. fine) and *"flott leilighet"* (engl. great apartement).

The results of the language detection are stored per *listing_id* in a new data frame called *reviews_features*. This data frame contains 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 listing id. Finally, this data frame was added to the *listings* data frame on which feature selection is performed later. (s. cite sentiment)

In addition, to obtain a more informed analysis of the reviews, we also performed a detailed sentiment analysis for each review. Sentiment analysis are 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* package, more specifically we used the *pipeline* function with the *task* argument set to *sentiment-analysis*, which is used for classifying sequences according to positive or negative sentiments (s. documentation). The used model is DistlBERT, a small, fast and light Transformer model, a distilled version of BERT algorithm, which achieves significantly faster results. (s. documentation transformers). Finally, we used the sentiment analysis to determine the ratio of negative reviews to the total number of reviews per listings id, which is also added to the *reviews_feature* data frame.

3 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 such as multiple

columns containing some kind of `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 and the apartment price must have some connection 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 take care about 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 of the number of missing values reducing the size of the data set and its 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 exluding important predictors that shine in combination with other variables rather than on their own.

Hence, in a second step we fit an auxiliary Linear Regression Model including *all* of the available features (except for the trivial ones such as the `picture_url`) and analyzed the absolute magnitude of the coefficients to get the relevant coefficient here. Since all variables are standardized these magnitudes are within the same range and unit-independent and can thus be compared.

This approach, of course, is not perfect as well since multiple highly correlated features having a strong *combined* impact on price could end up with

rather small individual estimated coefficients due to *sharing* their predictive / explanatory power. To circumvent this potential issue, we relied on *algorithmic* feature selectors provided by the `scikit-learn` library that are (ideally) capable of separating the effects of strongly correlated features and select only a small subset of them.

We decided to apply the following algorithms: *RFE* (Recursive Feature Selection), *SelectFromModel*, *SelectKBest*, *SequentialFeatureSelector* and *VarianceThreshold*. The detailed results are shown in the appendix. Because the *RFE* showed the best performance and the selected subset of original features can be immediately interpreted as potentially most important features, so we performed an even more detailed analysis with this feature selector afterwards.

The overall results of these algorithms lead to additionally adding the variable `property_type`. But thereby some of the observed connections had to be taken with care. For example, by far the largest coefficient was attributed to the category *Houseboat* of the `property_type` variable. However, this observation turned out to be the only Houseboat contained in the dataset. To avoid strong influences of these outliers, we combined all rare categories together to a larger `Other` category and used the mean price of all observations contained in this category.

The last step consists of data preprocessing. Therefore we implemented a function column_transformer, which included the One-Hot Encoding of the categorical variables as well as the standardization of the numeric variables.

4 Price Distribution

One key aspect of exploratory data analysis is investigating the *distribution* of the outcome 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 3 illustrates 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 by comparing true and predicted price on the logarithmic scale was drastically reduced, leading to smaller gradients and thus smaller weight updates. This issue could be mitigated to some extent with a larger learning rate. However, in contrast to the untransformed version, the network still suffered from *vanishing gradients* and *loss plateaus*.

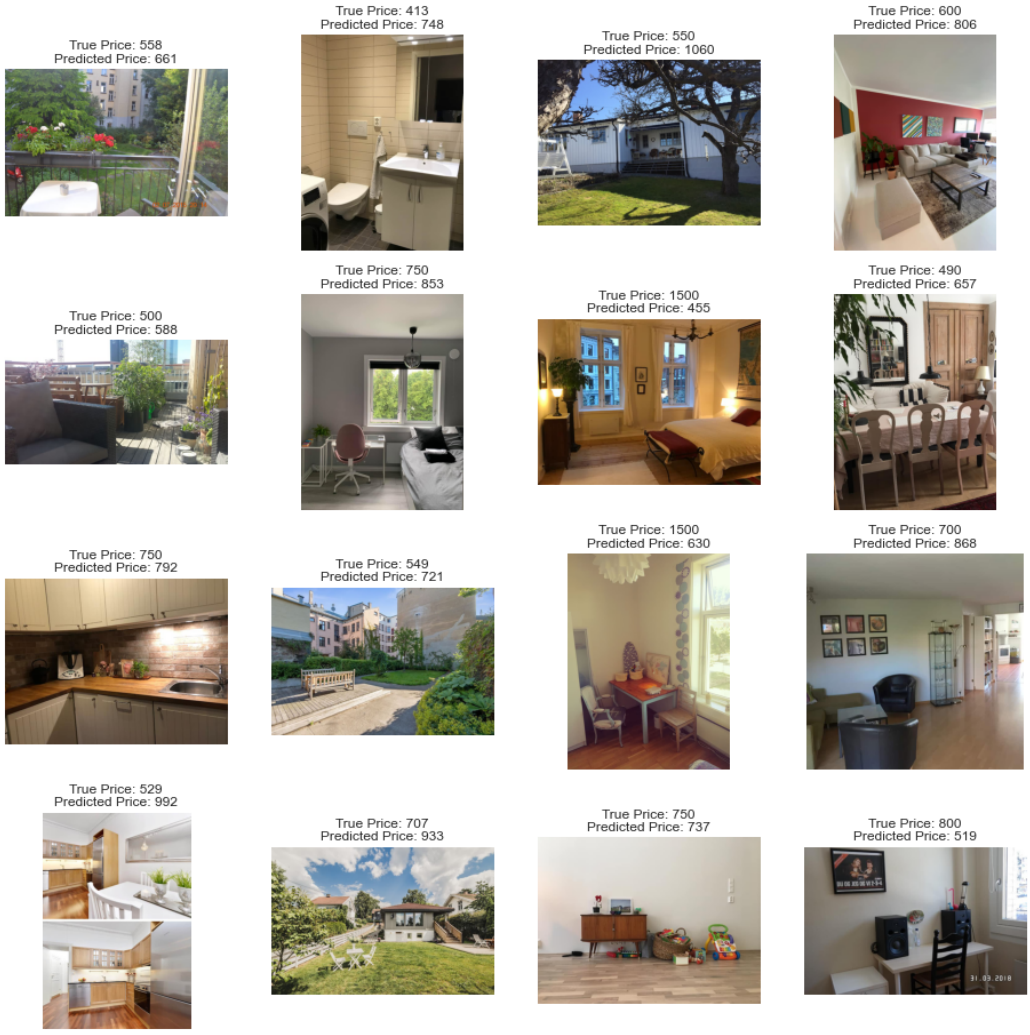

Figure 2: Images from Airbnb Apartments with true and predicted Prices

Figure 3: Distribution of Apartment Prices on original and logarithmic Scale