DET NATUR- OG BIOVIDENSKABELIGE FAKULTET
KØBENHAVNS UNIVERSITET

House Pricing
Using Machine Learning Methods

Thommas Sylvester Biener

Supervisor: Rolf Poulsen
May 31th 2022

Master Thesis in Mathematics-Economics
Department of Mathematical Sciences, University of Copenhagen

Abstract

The aim of this thesis is to apply extensive machine learning methods on house pricing. We utilize different regression models for approximating sale prices in the setting of Copenhagen from 2016 trough 2021. Initially, data is scrapped online, and intensively preprocessed using custom transformers. Next, exhaustive model work is performed with the following models: Linear-, Polynomial-, Random Forest-, Neural Network- and Ensemble Stacking regression. Each model contributes with better estimates for sale prices in the listed order. Inspired by the article *Han Lee & William C. Strange*, we make an empirical analysis of the role of the asking price in a Copenhagen market setup. This analysis shows that in a market with economic growth, the asking price direction search is reduced being in accordance with the findings in the article. Finally testing the Ensemble Stacking estimates as potential asking prices.

Contents

1 Introduction

Houses and apartments being sold in Copenhagen is a concept most people can grasp. Imagine we were about to give a valuation of some housing in Copenhagen. We would look at common attributes like location, square meters, number of rooms, number of toilets, when the house was built and many more. Some of these attributes we would be very concious about. Other public available housing attributes are maybe thought of more unconsciously or not even considered. Attributes like what the people in the proximity are voting, the criminal rate in the proximity and the sea level could be examples of such. We seek out to investigate if we using machine learning methods can explain the actual sale price with a combination of all such attributes.

To begin with, chapter 2 contributes to the theoretical framework for using machine learning and more explicitly using regression models. We explain the fundament of using machine learning in general and what it is. We explain some of the main challenges of machine learning. How we need to split the data such we do not infer bias. Then a precise approach method to make sure we can generalize estimation results. We then introduce the settings of the actual models. First, simple linear regression as an attempt at a benchmark model. Then explaining further, more complex, models which have their fortes. Polynomial regression, Random forest regression, Neural Network regression. Ending up explaining how to combine all mentioned models into a model which should in all cases never be worse than the best model. This model we call Ensemble Stacking regression. Further we introduced what regularization is. This section is to cement the foundation for a proper analysis of the housing market in Copenhagen.

In addition chapter 3 presents a complete analysis. Going from data gathering through intense model optimization such approximations of the optimal sale price, given each specific model, can be made. Expanding on the initial thought of the samples doing a quick look through it. To proceed to make more exhaustively preprocessing of the data for optimal model work. Splitting the data such we have a training set and a holdout validation (test) set. We look into feature engineering. We then start the process of finding the best transformations to the overall samples for our models. We do this by testing with a linear regression model. Making a pipeline that can take care of all transformations. Further we try to optimize each model introduced in chapter 2. Each we optimize by different approaches. We discuss the losses of each model and how we expect them to perform by comparing the benchmark linear regression model. We then take the holdout validation set into consideration and test performance of each model. Beyond we discuss performance enhancements such as including potential additional features.

Further chapter 4, following the article *What is the role of asking price?* by Lu Han and William C. Strange, provides a theoretical framework and an empirical analysis of asking prices in directing search. We then attempt to do similar empirical modeling in our market setting. Taking into consideration we potentially are working with very different markets. We explain the critical considerations of assumptions and follow up by making the multivariate linear regression relation between bidders and asking price as done in the article. Continuing we introduce our model estimates from section 3 as potential asking prices. A follow up discussion of this is then made.

All implementations can be seen in the Github repository found in Appendix A

2 Theoretical Framework

2.1 Machine Learning

The theory of this section is mainly is mainly based on [Géron, 2019] and [Yevgeny, 2019]. If other references are used it will be stated.

Machine Learning as a concept

What is Machine Learning? Experience with Machine Learning so far brings forth this 'home-made' definition

> *"Machine Learning is the job of making computers notice patterns which humans cannot through data." - Thommas Biener, 2022*

The definition from the author of the most cited book for this thesis is [Géron, 2019]

> *"Machine Learning is the science (and art) of programming computers so they can learn from data." - Aurélien Géron, 2019*

Machine Learning has been around for decades in somewhat broad terms, from the definition above. An early and hugely mainstream Machine Learning application, which changed the lives of millions of people for the better, is the so-called spam filter from the early 1990s. This spam filter is a program that tries to classify emails as spam or ham by analyzing flagged spam emails from users worldwide. For one to write a generic program that would tell spam from ham would require a long list of complex rules. And then, at the same time, regularly updating this list, since spam-mail-senders update frequently to avoid filters, would be a probably impossible and never-ending mission. However, by getting mail recipients to declare/red flag spam mails and then feeding Machine Learning algorithms this data (these spam emails), it can find patterns and "easily" classify new emails as spam. The more impressive part is that the algorithms can evolve with the ever-changing spam mails as long as the recipients keep red-flagging spam mails.

Machine Learning can be used in a variety of different ways. The example above is of a classification problem, but it can also be used to solve regression tasks which is precisely what we are going to do throughout this thesis.

Some of the most used Supervised Machine Learning algorithms are:

- Linear Regression
- Decision Trees and Random Forest
- Neural Network

While it is tough to determine which algorithms are used the most, these are definitely used a lot, and we will cover them in detail.

Lastly, considering that most Machine Learning applications are for predictions, it is worth pointing that the real goal of such applications is not to see clear patterns in the samples. It is to correctly apply patterns learned from old samples to make predictions about new samples.

2.1.1 Supervised Learning

Machine Learning can be somewhat depicted into two categories, either supervised or unsupervised. *"The most basic and widespread form of machine learning is supervised learning"* - [Yevgeny, 2019]. In a classical supervised learning setup, the learner, the practitioner of Machine Learning, is given samples. The derivation of a prediction rule for new samples is the primary goal.

The Supervised Learning set up

We wish to be very mathematically clear in how our models are set up. Thus we formalize and write the notations we use throughout, as in [Yevgeny, 2019].

- $\mathcal{X}$ - The sample space which will also be depicted as the feature space.
- $\mathcal{Y}$ - The label space.
- $X \in \mathcal{X}$ - The unlabelled sample.
- $(X, Y) \in (\mathcal{X} \times \mathcal{Y})$ - The Labelled sample.
- $S = (X_1, Y_1), \ldots, (X_n, Y_n)$ - A training set.
- $h : \mathcal{X} \to \mathcal{Y}$ - A hypothesis, prediction rule, a function which takes inputs from $\mathcal{X}$ to $\mathcal{Y}$.
- $\mathcal{H}$ - A hypothesis set.
- $\ell(Y', Y)$ - The loss function predicting Y'.
- $\hat{L}(h, S) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(X_i, Y_i)$ - The empirical loss of h on S.

Further more we assume the pairs of S, i.e. (X_i, Y_i), are samples i.i.d. according to an unknown but fixed distribution $p(X, Y)$. Hence we also have the expected loss of h where the expectation is taken with respect to $p(X, Y)$:

$$L(h) = \mathbb{E}[\ell(h(X), Y)]. \tag{2.1}$$

For regression models the most used loss functions are *square loss*:

$$\ell(Y', Y) = (Y' - Y)^2, \tag{2.2}$$

the *absolute loss*:

$$\ell(Y', Y) = |Y' - Y|, \tag{2.3}$$

and *relative loss*:

$$\ell(Y', Y) = \frac{|Y' - Y|}{Y}. \tag{2.4}$$

Moreover, these loss functions are what we use as measures for how well the hypothesis, prediction rule, h, generalizes to new samples. We have then established these loss functions, and with them comes the perk of being able to compare models.

2.1.2 Batch Learning

Batch learning is when Machine Learning applications are applied and a (or multiple) hypotheses h is found. This is then a final model with output and can be used on "identical"[1] samples (we will later make key assumptions about what "identical" samples are [2.1.4]). If the sample was to evolve and features were to change a huge deal, the model would see its hypothesis, h, which optimized the loss function on the original samples, might make dramatically worse predictions than an optimum h^* for these new samples with updated features. We need to understand this as *the algorithm takes no new information into account.*

Batch Learning is typically done offline and can be referred to as *Offline Learning.* If we wanted this Batch Learning application to learn about new instances, we would have to train the program all over to find this new optimum h^*. This application would then have to be trained on these new instances and all the old data. And then, the former model would be canceled, and the new model put to the task. When training sometimes lasts for long periods, the reintroduction of updated models is not always immediately possible. Training on a lot of data requires CPU, memory space, disk space, and more. Hence, it might be impossible to use Batch Learning on huge data. Furthermore, if we wish for a system that can learn autonomously on limited devices. Since carrying around large amounts of training data and using immense resources on training every specified time period is clearly not optimal.

These are some of the limitations of Batch Learning and why another technique as *Online Learning* was introduced, which learns incrementally [Géron, 2019].

We have mentioned the word model a few times and used it very loosely, and thus we try to clarify the actual interpretation and how it is used throughout.

2.1.3 Model-based Learning

As mentioned earlier, we wish to categorize our Machine Learning applications on how well they generalize, not just how good of a loss we get on the training. The system needs to make sound predictions on new samples. There are two main approaches to generalization, instance-based and model-based. We will focus on (you guessed it) model-based since it is what we use thoroughly throughout this thesis.

We wish for generalization on new data samples by building a model. Thus we make models with some predefined characteristics, which we hope the data samples will follow more or less. We try to find trends throughout the data. The model then needs specific parameters, and we choose these by the performance measure mentioned earlier, the loss functions.

One distinction we have to clarify is that a model can here be interpreted as a 'model' as in a linear regression model, Random forest model, or a Neural Network model. 'Model' can also be interpreted as a fully specified model architecture with input and output. Moreover, 'model' can also be interpreted as a final trained model with a hypothesis, h, which is a specified function of parameters. What we use has to be read in context.

Machine Learning can, in most settings, be summarized in the following four steps:

- Studying the data.
- Selection of a model.
- Optimization of a hypothesis, h, on a specified training set.

[1]Identical is an exaggeration hence the apostrophes

- Applying this found the best candidate of h^* on new data and hope for a great generalization.

Now that we have grasped some of the greatness and impeccable success that Machine Learning has had, we wish to set the record straight and mention some of the main challenges. It is not all promises of the earth.

2.1.4 Main Challenges in Machine Learning

The main goal of Machine Learning is to make predictions that generalize well by selecting an algorithm and training it on some data. This also indicates which two concepts that challenge any practitioner; Either the algorithm selection is problematic or the data is not optimally processed (or is just horrible.).

Some of the challenges with the data are:

- Insufficient Quantity of Training Data

Much data is often required for Machine Learning algorithms to function well. Even small and simple Machine Learning algorithms require (most of the time) thousands of samples to be effective. Furthermore, complex problems go up to enormous scales. Fortunately, it is sometimes possible to reuse parts of existing models.

- Nonrepresentative Training Data

If the training data is nonrepresentative of new samples which we wish to generalize our hypothesis, h, to a conclusion, will be very unlikely to be of any interest. Nonrepresentative Data could be outliers, i.e., crazy expensive house prices in a regression house pricing model. It is also vital that we do not have too much *sampling noise*[2], which can mostly be avoided when the amount of data is increased. On the other hand when we need to make sure we do not introduce *sampling bias* when generating a lot of data.

- Poor Quality Data

This is pretty self-explanatory, but nonetheless, we will make two points about the quality since what is easily an extremely time-consuming process is data cleansing. Firstly errors in the data have to be fixed since it will make the Machine Learning algorithms' job of finding underlying patterns very difficult. Such errors could be spotted as weird outliers. These can then be fixed manually or discarded. Secondly, missing values will most likely exist in non-cleaned data, which must be taken care of.

- Irrelevant Features

Again pretty self-explanatory, but it can be rather challenging to tell some useful features from other irrelevant ones. It is critically important to think thoroughly about the relevance of the features. This is called *Feature Engineering* and involves feature selection, as mentioned. It also involves feature extraction, the art of combining existing features such they become more relevant. Lastly, it can also involve the creation of new features by gathering new data.

Now we have listed some of the most common issues with samples, but we also wish to emphasize some of the challenges with the models:

- Overfitting the Training Data

[2]Chance based data.

Overfitting is a word for overgeneralizing. This can be a question of the Machine Learning algorithm is making "rules", a hypothesis h, by some specific instances, happening in the training data just by chance which does not generalize well (or maybe at all). More generally, overfitting happens when the ML algorithm is too complex relative to the amount and noisiness of the training data. We can do away by simplifying the model or cleaning the data more thoroughly, e.g., loss measures listed above. Constrain models to be simpler and reduce the risk of overfitting is called *regularization.*

- Underfitting the Training Data

Exactly as stated, underfitting is the opposite of overfitting and is when the model is too simple to learn the underlying pattern, and thus predictions will be inaccurate. Fixing this will be building a more complex model, doing more feature engineering, or reducing regularization on the model.

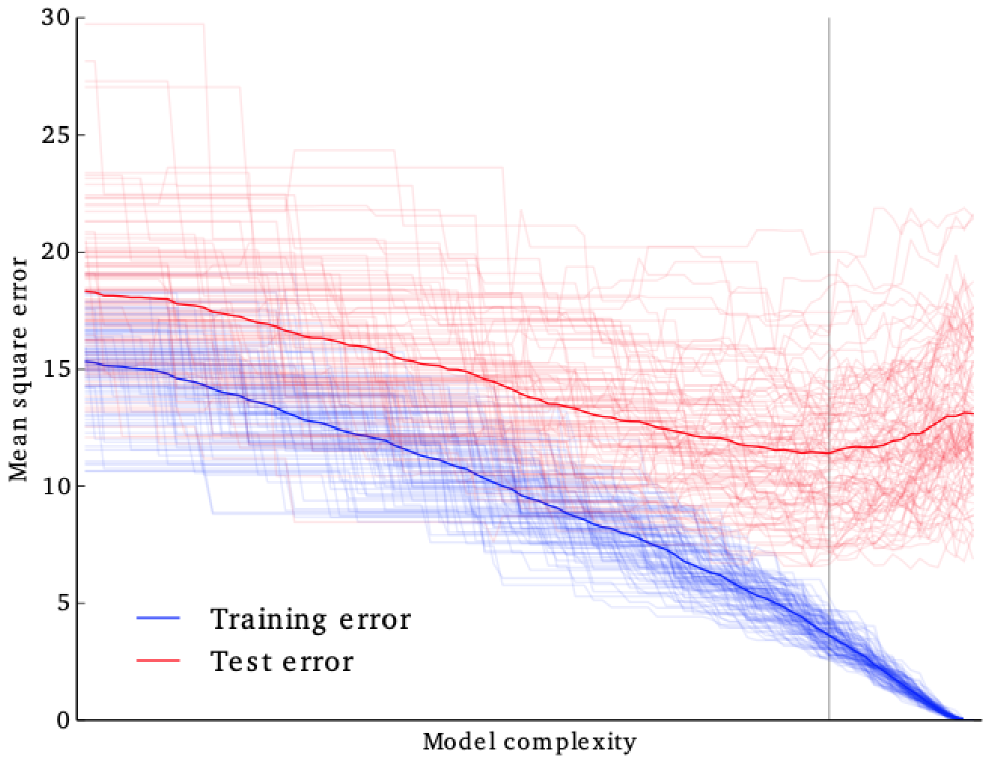


Figure 2.1: This graph is taken from [Louppe, 2014]. The graph is of multiple random forest models with unique hyper parameters.

The graph is done for a large set of different random forest models with unique hyperparameters deciding their performance. What is seen on the graph is a very general implication of machine learning methods. This graph shows how increasing model complexity will optimize training loss but make for worse generalization to validation sets or test sets. This is known as the bias-variance trade-off. We always wish to make reasonable generalization estimates. As seen by the vertical line in the graph.

- Hyperparameter Tuning

A hyperparameter is a parameter of a learning algorithm, e.i. Regularization can often be tweaked through the learning algorithm and is then a hyperparameter. Moreover, tuning hyperparameters are a huge part of making good generalizations. However, to tweak these hyperparameters, it is necessary to evaluate the different models with differing sets of hyperparameters and compare losses. We use new data to see if our hypothesis, h, generalizes well when evaluating a model. This is often just a part of the full data set, which can be split into training and test sets. Often, this is done in an 80:20 ratio. We can then test how well h is doing on the test data, which the model has never seen. Let us assume the model is doing less than wished. We will have to train a new model with new hyperparameters and re-evaluate it. This is not as simple as it sounds since now we have used our test set to make decisions about parameters, and thus we run the risk of not being able to generalize h very well if we evaluate on the same test set. We thus need to make sure we have some samples we check generalization with. To do this, we need to use a *holdout validation set* which is only ever used when we are satisfied with the model to test generalization. These are some of the challenges in tweaking hyperparameters; Handling data.

There are many things to consider when making a model we wish to generalize well. Sample-based and model-based challenges. We do some further analysis on the tuning aspect of machine learning.

Generalization when tuning

A very central topic in machine learning is the credibility of the methods. The classical process of finding optimum hypotheses, h^*, can be seen as a selection process. We can then describe the selection process as follows:

- First, we start with a hypothesis set $\mathcal{H}$, which is a set of plausible prediction rules.
- Second, we observe a sample S sampled i.i.d. according to a fixed and unknown distribution $p(X, Y)$.
- Third, we select a prediction rule $\hat{h}^*_S$ which is the best of seen hypotheses in $\mathcal{H}$ measured by the empirical performances $\hat{L}(h, S)$ of the hypotheses in $\mathcal{H}$.
- Last $\hat{L}(h, S)$ is then applied to predict labels for new samples X.

Remembering that $\hat{L}(\hat{h}^*, S)$ is observed and $L(\hat{h}^*_S)$ is unobserved. It is undesirable to have small $\hat{L}(\hat{h}^*, S)$ and large $L(\hat{h}^*_S)$ since based on $\hat{L}(\hat{h}^*, S)$ we believe $\hat{h}^*_S$ performs well though this is most likely not true.

We make two key assumptions about the samples:

1. *The samples in S are i.i.d.*
2. *The new samples (X, Y) come from the same distribution as the samples in S.*

These assumptions are generally used in practical supervised, batch learning model-based Machine Learning. An example could be a classification problem trained on American Hollywood movies applied to American TikTok videos. Both have some messages, are primarily in English, and have special effects, but the prediction accuracy can be tremendously volatile. Hence by the assumptions, we get:

$$\begin{aligned}\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\hat{L}(h,S)\right] =&\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\frac{1}{n}\sum_{i=1}^{n}\ell(h(X_i),Y_i)\right]\\ =&\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\ell(h(X_i),Y_i)\right]\\ =&\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{(X_i,Y_i)}\left[\ell(h(X_i),Y_i)\right]\\ =&\frac{1}{n}\sum_{i=1}^{n}L(h)\\ =&L(h).\end{aligned}\tag{2.5}$$

The above equations says for any fixed prediction rule that is independent of S, the empirical loss is an unbiased estimate of the true loss, $\mathbb{E}\left[\hat{L}(h,S)\right]=L(h)$. Intuitively the assumptions makes sure new samples (X,Y) are in no way different from the samples in S; it does not matter which are new samples and which are old samples. However when the selection of $\hat{h}^*_S$ based on S is made, this S matters a whole lot. Hence $\mathbb{E}\left[\hat{L}(\hat{h}^*_S,S)\right]\neq\mathbb{E}\left[L(\hat{h}^*_S)\right]$[3] since $\hat{h}^*_S$ is made using exactly the sample S. And from the perspective of the selection process the samples in S are not exchangeable with the new samples (X,Y). As seen below

$$\begin{aligned}\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\hat{L}(\hat{h}^*_S,S)\right] &=\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\frac{1}{n}\sum_{i=1}^{n}\ell(\hat{h}^*_S(X_i),Y_i)\right]\\ &=\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\ell(\hat{h}^*_S(X_i),Y_i)\right]\\ &\neq\mathbb{E}_{(X,Y)}\left[\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\ell(\hat{h}^*_S(X),Y)\right]\right]\\ &=\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[\mathbb{E}_{(X,Y)}\left[\ell(\hat{h}^*_S(X),Y)\right]\right]\\ &=\mathbb{E}_{(X_1,Y_1),\dots,(X_n,Y_n)}\left[L(\hat{h}^*_S)\right]\\ &.\end{aligned}\tag{2.6}$$

This is why it is so important not to investigate the whole sample at all times, hence why it is common practice to split data into training and test set. Suppose we find an optimal $\hat{h}^*_S$ one that minimizes the loss and then uses this hypothesis on the test set and then realizes it is not performing up to par. If any tweaking is done, whether it is on the model's hyperparameters or others, a bias is introduced. We have to set aside a test set that is not looked upon before all model work is finished since we wish to infer as little bias as possible.

There are multiple ways of optimizing the prediction rule and finding $\hat{h}^*_S$. One way is to part the total samples into three parts, a training set, a validation set, and the mentioned holdout validation set[4]. And then find the optimal loss on this validation set. And since the model is trained on the training set, there should be inferred much bias. We choose a cross validation method, which is not very different, to test our models throughout.

[3]$\hat{h}^*_S$ is a random variable depending on S and expectation taken with respect to this randomness
[4]Often just mentioned as the test set.

Cross-Validation

A standard technique for adjusting hyperparameters of prediction models are to *Cross-Validation*, CV. As the approach of [Igel, 2019], say we use k-fold cross-validation on our sample S. We in reality are partitioning S into k near equal size subsets, i.e. making $S_1, \dots, S_k$ where

$$\lfloor |S|/k \rfloor \leq |S_i| \leq \lceil |S|/k \rceil,$$

and we assign each data point of S randomly to the subsets. We furthermore define the union of all samples except those in the subset S_i:

$$S_{\setminus i} \equiv \bigcup_{j=1,\dots k \,\wedge\, j \neq i} S_j. \tag{2.7}$$

Then for each $i = 1, \dots, k$, an individual model with a hypothesis, h, is built, applying the algorithm to the training data S_i. We then evaluate the model using some loss function using the not included samples, S_i, as the test set. The average of the k folds losses are then taken, and this evaluation is called the *cross-validation performance*. It will be used as a predictor of the performance of the prediction rule/hypothesis h.

$$CV(h_{S_i}) = \frac{1}{k}\sum_{i=1}^{k} L(h_{S_{\setminus i}}) = \frac{1}{k}\sum_{i=1}^{k}\left(\frac{1}{|S_{\setminus i}|}\sum_{l=1}^{|S_{\setminus i}|} \ell\left(h(X_l \cap S_{\setminus i}), Y_l \cap S_{\setminus i}\right)\right) \tag{2.8}$$

The typical choices of k-folds is 5 or 10 [Hastie et al., 2008]. The case where the folds and the size of the sample align is known as *leave-one-out* cross-validation. Where computing is done using all data except one observation. This is often *not* favorable and does require a lot of computational time, but in certain instances, it can be of use.

We have introduced machine learning and, more specifically, what we will be using for categories within machine learning. Furthermore, we have gone through many challenges with just machine learning. Specifically, it is essential to understand the concepts. We are doing supervised batch model-based learning. Moreover, we finally arrive at the models we will use to make, hopefully, good prediction rules.

2.2 Model theory

The theory of this section is mainly is mainly based on [Yevgeny, 2019], [Géron, 2019], [Louppe, 2014], [Nielsen, 2019] and [Breiman, 1996]. If other references are used it will be stated.

We will theorize most of what we wish to use for analyzing the samples. Our goal is to get reasonable estimates of what housing in Copenhagen and Frederiksberg is being sold. Throughout the section, we will set up the theoretical need for using our chosen four[5] main models. First, The plainest model is a linear regression model. This particular model will also be used for some of the cleaning of the data. Second, we use an ensemble model consisting of a random forest model. Third, we will build a Neural Network. And in the end, we will again look at an ensemble method, this one called, Ensemble Stacking regression.

[5]Actually five models if one considers the polynomial model in which we utilize linear regression as a separate model.

Regression tasks are the special case of supervised learning with

$$\mathcal{X} = \mathbb{R}^d \quad \text{and} \quad \mathcal{Y} = \mathbb{R},$$

where d is the feature space. As a benchmark we use the linear regression model but we expect it to be a very complex task to find rules of predictions. Finding a relationship between each feature and the label.

2.2.1 Linear Regression

Supervised Batch Linear Regression based Learning or just a *Linear Regression model* [Yevgeny, 2019]. We will use this model as a sanity check and as a benchmark. As earlier mentioned we let $S = (x_1, y_1), \ldots, (x_n, y_n)$ be our sample. We wish to find a prediction rule of form $h(x) = \theta^T x$. Where θ is the prediction rule also representable as feature weights and $\theta \in \mathbb{R}^d$. Let $X \in \mathbb{R}^{n \times d}$ be the unlabelled sample set matrix and let $x_1^T, \ldots, x_n^T$ be the corresponding vectors of all the features for each sample:

$$X = \begin{pmatrix} - & x_1^T & - \\ & \vdots & \\ - & x_n^T & - \end{pmatrix} \tag{2.9}$$

And let

$$y = (y_1, \ldots, y_n)^T \tag{2.10}$$

be the vector of labels. The goal of Linear Regression is to find a narrow relation between the prediction rule and the labels. This relation will be measured by the empirical loss function (we here use square loss as an example):

$$\begin{aligned} \hat{L}(h(x), S) =& \frac{1}{n} \sum_{i=1}^{n} \ell\left(h(x_i), y_i\right) \\ =& \frac{1}{n} \sum_{i=1}^{n} \ell\left(\theta^T x_i, y_i\right) \\ =& \frac{1}{n} \sum_{i=1}^{n} \left(\theta^T x_i - y_i\right)^2 \\ =& \frac{1}{n} ||X\theta - y||^2. \end{aligned} \tag{2.11}$$

By tweaking θ we can narrow this relation and thus minimize the loss. And if new samples (x, y) come from the same distribution as the samples in S (and S is i.i.d.), the prediction rule will generalise well.

When the number of samples is larger than the number of features, i.e., $n > d$, there will be a significant possibility that the linear system $X\theta = y$ does not contain a solution. A solution is only happening if y by chance is in the linear span of the columns of X, which is highly unlikely. An approximation will do, and this is done by finding a projection of y on to the column space of X.

This can be algebraic. We are looking for the best solution $X\theta = y$. This is unlikely since y is outside of the span of X; hence best we can do is find $y_* = X\theta$. y_* is the projection of y onto the column space of X, i.e., we are looking for y_* such the remainder $y - y_*$ is perpendicular to the projection.

$$(y - y_*) \perp y_*, \qquad \langle (y - y_*), y_* \rangle = 0.$$

We know that y_* belongs to the image of X thus, the remainder must be in the nullspace of X^T, meaning:

$$\begin{aligned} X^T(y - y_*) =& X^T(y - X\theta) = 0 \\ X^T y =& X^T X\theta \\ \theta =& \left(X^T X\right)^{-1} X^T y \end{aligned} \tag{2.12}$$

In the last step, the independence of the columns of X was used. This means, at best we can solve the a linear system $y_* = X(X^T X)^{-1} X^T y$ and get the prediction rule of our linear system $\theta = (X^T X)^{-1} X^T y$

Expanding linear regression to high dimensions

Without too much effort, we expand the theory above to high dimensions in the sense of polynomials. Rather easily, we expand the complexity of the linear regression and expect this will be able to fit training data much better. We wish to fit our data $S = (x_1, y_1), \ldots, (x_n, y_n)$ with a polynomial of degree m. The model we wish a solution for is then $y = \theta_k x^m + \theta_{m-1} x^{m-1} + \cdots + \theta_1 x + \theta_0$. We then only have to map our features x_i into a feature vector:

$$x_i \rightarrow (x_i^m, x_i^{m-1}, \ldots, x_i, 1)$$

And then apply same above theory on the following high dimensional linear system:

$$\begin{pmatrix} x_1^m & x_1^{m-1} & \ldots & x_n & 1 \\ x_2^m & x_2^{m-1} & \ldots & x_n & 1 \\ & & \vdots & & \\ x_n^m & x_n^{m-1} & \ldots & x_n & 1 \end{pmatrix} \begin{pmatrix} \theta_m \\ \theta_m{-1} \\ \vdots \\ \theta_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{2.13}$$

To get the parameter vector $(\theta_m, \theta_{m-1}, \ldots, \theta_1, \theta_0)^T$

Polynomial models of higher degrees are not practical. Creating all cross interactions between each feature will make the feature space enormous. Thus, we will only see a polynomial of second degree. A way to get to a somewhat middle ground between a linear model and a polynomial model is to use regularization. Here Lasso regularization is functional. Least Absolute Shrinkage and Selection Operator Regression (Lasso), as mentioned in [Géron, 2019], adds a penalty term to the loss function. It uses the ℓ_1 norm of the parameter vector.

$$Lasso(\theta) = L(h(x), S) + \alpha \sum_{i=1}^{d} |\theta_i|. \tag{2.14}$$

A significant consequence of using Lasso regularization is eliminating features of most negligible significance, setting the associated parameter value to zero. Thus running a polynomial with lasso regularization where we suspect many cross features to be of very little importance will give us a chance of finding an optimum prediction rule, h^*. The α parameter tells the algorithm how harsh it should be at deleting features. Setting it to zero is just a regular regression; further, setting it to one will make a constant function.

2.2.2 Random Forest

In trying to get better estimates, we include a Random Forest regression. We wish to take advantage of how the algorithm makes discrete decisions on the features; this model performs well if the labels are clustered somehow.

Before explaining a random forest, we need to know what a decision tree is. We will try to explain the principles of a decision tree with a geometrical approach. We know a graph, G, consists of a finite and non-empty set, V, of vertices and a finite set, E, of edges from [Lützen, 2019]. We will start with four definitions as stated in [Louppe, 2014]:

Def. 1. A *Tree* is a graph $G = (V, E)$ in which any two vertices (or nodes) are connected by exactly one path.

Def. 2. A *Rooted tree* is a tree in which one of the nodes has been designated as the *Root*. Additionally, all edges are directed away from the root.

Def. 3. If an edge from t_1 to t_2 exists then node t_1 is said to be the *Parent* of node t_2 and t_2 is the *Child* of node t_1.

Def. 4. In a rooted tree, a node is said to be *internal* if it has one or more children and *Terminal* if it has no children. Terminal nodes are also known as *Leaves*.

A *Decision Tree* can then be defined as a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ represented by a rooted tree where any node, t, represents a subspace $X_t \subseteq X$ with the root node t_0 corresponding to X itself. Internal nodes, t, are labeled with a *split*, $s_t \in h_t^{split}$. Where h^{split} is a set of splitting rules that divides the space X into disjoint subspaces throughout the tree, and h_t^{split} is the splitting rules that divides the space X_t at t into corresponding children nodes. This means the children of node, t, is split into v sets:

$$X_t^{(1)}, X_t^{(2)}, \ldots, X_t^{(v)} \subset X,$$

with each new children node consisting of

$$c_j = X_t \cap \left(X \backslash \left(\bigcup_{i=1,\, i \neq j}^{v} X_t^{(i)} \right) \right) \tag{2.15}$$

Further, the terminal nodes are labeled with the best prediction of y, i.e., the prediction label value $\hat{y}$. The algorithm takes the mean of all sample labels where their corresponding samples have run the same way through the tree, which gives the hypothesis of the tree, $h^{tree}(x) = \hat{y}$[6]

We can calculate how well the expected global empirical loss performs by calculating the expected local empirical loss in each terminal node:

$$\begin{aligned}
\mathbb{E}_{X,Y}\left[L(h(x), S)\right] &= \mathbb{E}_{X,Y}\left[L(h^{tree}(X), Y)\right] \\
&= \sum_{t \in \mathcal{T}} P(X \in \mathcal{X}) \mathbb{E}_{X,Y|t}[L(h^{tree}(X), Y)] \\
&= \sum_{t \in \mathcal{T}} P(X \in \mathcal{X}_t) \mathbb{E}_{X,Y|t}[L(\hat{y}, Y)]
\end{aligned} \tag{2.16}$$

[6]This h^{tree} also contains hyperparameters which we will get into in a bit.

Here τ denotes the set of terminal nodes in which h^{tree} makes. P is the probability and the last expectation is the local empirical loss of the model at node t. Gives, if we wish to optimize the loss globally we can just do it leaf-wise. And by using squared loss as done previously we can find the optimal label value $\hat{y}^*$:

$$\begin{aligned} \hat{y_t}^* &= \underset{\hat{y}}{\operatorname{argmin}}\, \mathbb{E}_{X,Y|t}[(\hat{y} - Y)^2] \\ &= \mathbb{E}_{X,Y|t}[Y] \end{aligned} \tag{2.17}$$

This value will be approximated using estimates since for computing we would need to know the probability:

$$\begin{aligned} \hat{y_t} &= \underset{\hat{y} \in S}{\operatorname{argmin}} \frac{1}{N_t} \sum_{x,y \in S_t} (\hat{y} - y)^2 \\ &= \frac{1}{N_t} \sum_{x,y\ \in S_t} y. \end{aligned} \tag{2.18}$$

We have explained what a decision tree is and how to estimate the loss. But the way we have defined the tree at this given time will be a rooted tree with $|Y|$ edges to the leaves. Hence precise criteria will be made on the unlabelled sample, and it will make a leaf for every labeled sample. This means the training loss will be 0, but the generalization will probably be terrible.

We need to put some criteria on the tree to perform better. We could put a criterion of a certain number of children per internal node, but this will not keep the tree growing until every labeled sample is used. Therefore we need to put up some stopping criteria which will stop the tree from growing and will be a good prediction.

This is a case just as we discussed before with overfitting, the model becomes too complex and will catch noise in the samples, which will make it compute bad generalizations. Therefore the best tree is a tree that is neither too complex or too simple (too deep or too shallow). Thus some stopping criteria are inferred, such before h becomes to specific for the specific samples, leaf nodes are found. Obviously, the tree should not split if:

$$\exists\, y = y' \vee x = x' \ \forall\, (x, y), (x', y') \in S_t.^7 \tag{2.19}$$

Furthermore to prevent overfitting we complement the stopping criterion with heuristics halting splitting if S_t has become too small or if no sufficiently good splits can be found.

(a) Set t as a terminal node if it contains less than N_{min} samples.

(b) Set t as a terminal node if its depth $d_t \geq d_{max}$. Where d_{max} is a threshold maximum depth wanted.

(c) Set t as a terminal node if $p(t)\Delta i(s^*, t) < \beta$. That is the total decrease in impurity is less than some value, β.
Where $p(t) = \frac{N_t}{N}$ is the estimated probability, and

$$\Delta i(s^*, t) = i(t) - (p_1 i(t_1) + \cdots + p_M i(t_M)),$$

and the probabilities are given by $p_1 = \frac{N_{t_1}}{N_t}, p_M = \frac{N_{t_M}}{N_t}$, further is

$$i(t) = \frac{1}{N_t} \sum_{x,y \in S_t} (y - \hat{y}_t)^2$$

[7] This is actually two criteria $y = y'$ indicates samples are homogeneous and $x = x'$ indicates unlabelled samples are each locally constant.

(d) Set t as a terminal node if no split where children all can count at least N_{leaf} samples.

We still need to expand on what the optimal split, s^*, is. The usual splits are the splits defined by a single variable and resulting in non-empty subsets S_t:

$$h_t^{split} = \{s | s \in \bigcup_{j=1}^{p} h_t^{split}(X_j), S_{t_1} \neq \emptyset, S_{t_2} \neq \emptyset, \dots, S_{t_M} \neq \emptyset\}, \tag{2.20}$$

Then the best split for each variable is a question of finding:

$$s_j^* = \underset{\substack{s \in h_t^{split}(X_j) \\ S_{t_1},\dots,S_{t_M} \neq \emptyset}}{\arg\max} \; \Delta i(s,t), \tag{2.21}$$

and the best of the best splits is then given by

$$s^* = \underset{\substack{s_j^* \\ j=1,\dots,p}}{\arg\max} \Delta i(s_j^*, t). \tag{2.22}$$

With this an actual hypothesis, h^{tree}, which can generalize somewhat well can be made.

Going from a decision tree model to a random forest model is now no tricky part. We are making a forest containing T decision trees. We build each of these trees with bootstrapped samples from the original samples. The new samples have the same size. Make optimal splitting choices as shown above for each of the T trees. Then running samples through each of these trees gives a prediction:

$$h^{tree,(i)}(x) = \hat{y}^{(i)}$$

And then taking the mean of these across all trees gives the final ensemble random forest prediction:

$$h(x) = \frac{1}{T} \sum_{i=1}^{T} h^{tree,(i)}(x) = \frac{1}{T} \sum_{i=1}^{T} \hat{y}^{(i)} \tag{2.23}$$

One could call a random forest model or algorithm a meta learner. We are learning from multiple other models and making predictions from these. This averaging upon many trees should reduce overfitting a great deal.

2.2.3 Neural Network

For this section these two books are used [Dreyfus, 2005] and [Nielsen, 2019]

Neural Networks can have many free parameters (the weights and biases at each node) and handle them very well. The downfall of the polynomial regression is the forte of neural networks. We suspect very complex interactions between models; hence the neural network should be better at finding these cross interactions.

A *Neural Network* is, as with the other methods we have discussed, a way to infer a prediction rule or a hypothesis, h, to make predictions, and again it is a job of minimizing a loss function to update this, h, and find the optimal prediction rule that will generalize well.

The Neural Network consists of an input layer, potentially a number of hidden layers and an output layer. as visualized below.

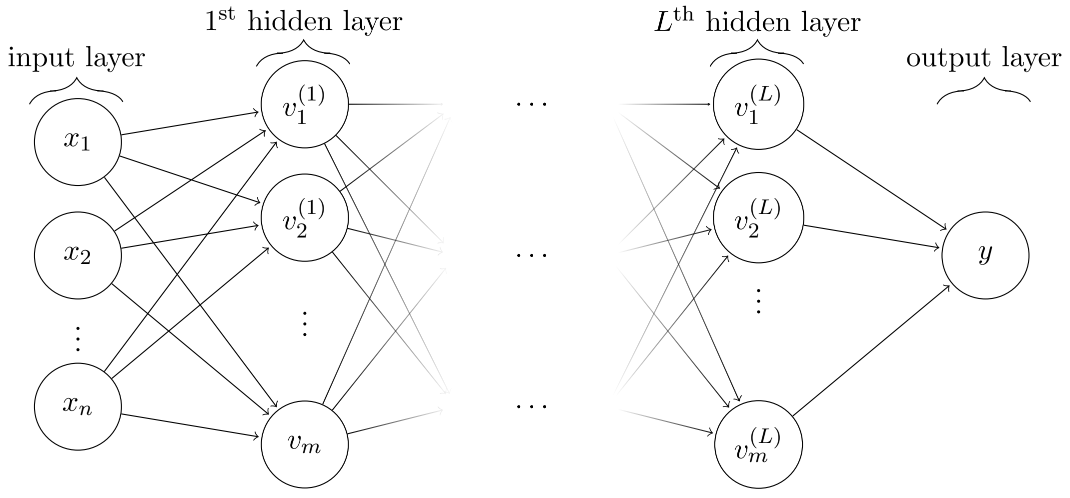


Figure 2.2: Network graph of a $(L+1)$-layer perceptron with n input units and one output unit. The l^{th} hidden layer contains $m^{(l)}$ hidden units.

In the input layer, denoted on the graph as $x_1, \dots, x_n$, we have each feature in our sample. Then potentially a number of hidden layers with a number of neurons, denoted as $v_1, v_2, \dots, v_m$, which taken the output from the previous layer outputs a nonlinear combination. Finally an output layer which takes input from previous layer and by some parametrizing function gives the final prediction.

For most neural networks each *neuron* takes the weighted sum of inputs with a bias term

$$w = \theta_0 + \sum_{i=1}^{n} \theta_i x_i = \theta_0 + \theta\mathbf{x}, \tag{2.24}$$

these weights and this bias are initially set and are each different from one another. Then on each of these weighted sums an activation function is used and thus the non-linearity. We consider a function defined in a neural network set up as the RELU activation function, which is merely given by

$$v(w) = \text{RELU}(w) = \max(0, w) = \max(0, \theta_0 + \theta\mathbf{x}). \tag{2.25}$$

The RELU function is linear for all positive values and zero for non-positive values. This is called a *feedforward* network.

This setup by itself will have a very tricky job of making good predictions when we initialize some weights randomly. We need to define the algorithm to approximate weights and bias terms such the predictions are good and, more importantly, generalize well. We infer a loss function, and we will optimize the model by gradient descent.

Gradient Descent

When thinking about gradient it is a matter of finding an optimum. By calculating the derivative we get the slope and hence know which directing to take to get closer to an optimum. Since we measure how good of a prediction we make by some loss it is this loss function in which we need to differentiate. In clarifying words by taking the derivative of the loss function in regards to the parameters, θ_j we calculate how much of a change in the loss a small change in the parameters will infer. The partial derivatives are given by

$$\frac{\partial}{\partial \theta_j} L(h(x), S) = \frac{\partial}{\partial \theta_j} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i) = \frac{\partial}{\partial \theta_j} \frac{1}{n} \sum_{i=1}^{n} \ell(\theta_i x_i + \theta_0, y_i). \tag{2.26}$$

Hence the gradient is then

$$\nabla_\theta L(h(x), S) = \begin{pmatrix} \frac{\partial}{\partial \theta_1} L(h(x), S) \\ \frac{\partial}{\partial \theta_2} L(h(x), S) \\ \vdots \\ \frac{\partial}{\partial \theta_n} L(h(x), S) \end{pmatrix}. \tag{2.27}$$

This gradient vector points uphill thus we subtract it from the weights to go the opposite direction. This corresponds to one step in the gradient descent algorithm:

$$v \to v' = \theta - \eta \nabla_\theta L(h(x), S), \tag{2.28}$$

Where η is the learning rate, a small positive parameter, corresponds to us choosing the size of the downhill. We will then use this update rule again to further minimize the loss function by the parameters, weights, and bias. We will do this repeatedly until a global minimum is found and an optimal prediction rule that makes the best possible predictions considering the hyperparameters.

A way of calculating these gradients are called back propagation. It is a question of computing the loss term δ^l for each layer, as mentioned in [Nielsen, 2019].

$$\delta^l = \frac{\partial L(h(x), S)}{\partial \theta^l}, \qquad \text{l=2,...,L+1} \tag{2.29}$$

Where l is the layers except the input layer. We know the loss of the output layer.

$$\delta^L = \Sigma(\hat{y}) \nabla_v L(h(x), S) \tag{2.30}$$

Where Σ is a square matrix whose diagonal entries are the values of $v(w)$ and the all other entries are zero. We can write the loss for each layer by the loss of the subsequent layer.

$$\delta^l = \Sigma(v^l)(\theta^{l+1})^T \nabla_v L(h(x), S) \delta^{l+1} \tag{2.31}$$

Thus by calculate backwards and because we know the output layer loss we can get a loss for each layer:

$$\delta^l = \Sigma(v^l)(\theta^{l+1})^T \dots \Sigma(v^{L-1})(\theta^L)^T \Sigma(\hat{y}) \nabla_v L(h(x), S) \tag{2.32}$$

This is what the gradient descent algorithm uses.

Choosing the learning rate, η, is essential to ensure the algorithm works appropriately. We need to choose it small enough, so the algorithm does not miss the optimum and diverges and large enough for the algorithm not to be too exhaustive.

A problem with regular gradient descent is how exhaustive the process of computing the derivative for the whole sample space is at every iteration. Another extreme is *Stochastic Gradient Descent*. At every iteration of the gradient descent algorithm, the gradient is computed on a single random instance, making this method undeniably faster. Instead of slowly decreasing to an optimum, this algorithm will decrease the loss only on average. When close enough to the global optimum, it will continue to bounce around the optimum. This algorithm can find excellent, fast approximations of the optimum. If the loss is not convex, there may be local optimums in which the regular gradient descent cannot escape; this stochastic approach has a higher chance of escaping locals and finding the global.

Finally, the last algorithm is called *Mini Batch Stochastic Gradient Descent*. It is a mix between the regular and the stochastic gradient descent. At each step of the algorithm, instead of computing gradients for the whole samples or sole instances, we select random batches, which are a selected part of the sample, to compute gradients. By this procedure, the exhaustiveness of gradient descent is minimized to a degree, and it will get closer to the actual optimum than SGD.

2.2.4 Ensemble Stacking Model

Each of the previously mentioned models can, to some degree, generalize. Different models have different forte and we wish to utilize these differences to ensemble a superior model.

Forming a linear combination of different predictors to improve prediction accuracy is what we call *Ensemble Stacking Regression*. Suppose we have K predictors $v_1, \ldots, v_K$ all trying to predict labels $\hat{y}$ by the same training set $S = (X_1, Y_1), \ldots, (Y_n, X_n)$. Using these sub-models in a meta-model, we can predict even better. An example could be to use predictor, v_k, a linear regression, v_{k+1} a random forest regression, and v_{k+2} a neural network regression, ensemble these into this meta stacked regression to predict better.

A standard approach is to get a collection of predictors and then find the single best among these. Our take on this is to use the collection of the models to make a single powerful predictor.

Ensemble stacking works by learning the relationship between the predictors' result of each of the ensembled models on out-of-sample predictions and the actual value. The model is trying to find the relationship that is most stable and in turn has less variance. Using the, k, predictors[8] as base model predictors training and predict out of sample data. And then a chosen meta-model fits the predictions from the base models and then learns to combine the predictions optimal.

By utilizing cross-validation we get these out-of-sample predictions. The is as follows:

1. Using K-fold cross-validation, as explained earlier to separate the sample into K-folds.
2. Hold out one of the folds and train all of the base models to the other folds.
3. Predict the held-out fold using the base models.
4. Get out-of-sample predictions for all K folds by iterating all previous steps the K times.
5. Use all of these out-of-sample predictions, i.e., meta-features, as input to the meta-model.

[8] At least to base model predictors.

6. Let the meta-model find an optimum relation between the meta-features.

This ensemble stacking model is then a question of choosing the best meta-model. Choosing too complex a meta-model will lead to overfitting the base model predictions. On the other hand, too shallow of a meta-model may cause the final prediction to be highly sensitive to changes in the data. This is due to the strong correlation in the base model predictions. Hence bad generalization.

Noting the section about linear regression. Throughout the ensemble stacking model work, we will use Ridge Linear regression as the meta-model. Ridge regression or Ridge regularization, as [Géron, 2019] states, is using the same optimization approach as the linear regression and Lasso regularization but with another loss function. For Ridge half the ℓ_2 norm is added to the loss:

$$Ridge(\theta) = L(h(x), S) + \alpha\frac{1}{2}\sum_{i=1}^{d}\theta_i^2 \tag{2.33}$$

This α is how hard we wish to regularize the model. $\alpha = 0$ is linear regression and α large forces all parameter values close to zero resulting in a flat line as the regression prediction. We use a regularized model as the meta-model to try and control the very correlated inputs.

With all mentioned models, linear, polynomial, random forest, neural network, and ensemble stacking, we explained how they have their own individual way of predicting. By looking at them each as their own, we get a lot of information about the samples and labels. Furthermore, machine learning a good approach is always to run multiple models to see if one can get the best of all worlds.

3 House Pricing Analysis

3.1 Data Gathering

To be able to use any of the proposed models and hence compute any estimates for housing, we need data. We wish to estimate on only samples we can scrape online and is available for everyone. We scrape from a few different web pages. Arguably the most important is the actual sale price of the housing. If this is not included we have no use for the sample. Multiple web pages are used to get features in all kinds of shapes and sizes hopefully bringing some information the models can use.

We scrape from the start of 2016 till the end of 2021. Meaning the data is quite up to date. We do this analysis on data in Copenhagen; hence we only scrape in Copenhagen. We scrape from four different web pages.

- Boligsiden.dk

Boligsiden is the first web page to be scraped from. We get the desired addresses for all sales within our time period. Afterward, each page for every address is scraped where a sales price is denoted. This web page contains mostly sales information.

- Dingeo.dk

We wish to take a lot of different aspects into consideration and this is where dingeo.dk has its forte. For each address we scrape again. From this web page we scrape how far above water the house is, a noise of the proximity parameter, flooding risk, election area and what politically party the majority are voting for, just to give a few examples of how broad the scraping is.

- Hvorlangterder.dk

The next web page scraped from is hvorlangterder.dk and this page gives us a meassure of distance to a lot of (maybe) useful features. The distance from the housing to the closest hospital, metro, beach. To give some examples.

- Statbank.dk

The last web page we scrape and actual use is statbank.dk. Here we get market and economic features. Monthly based price indexes for housing, employment rate for Copenhagen and all of Denmark etc.

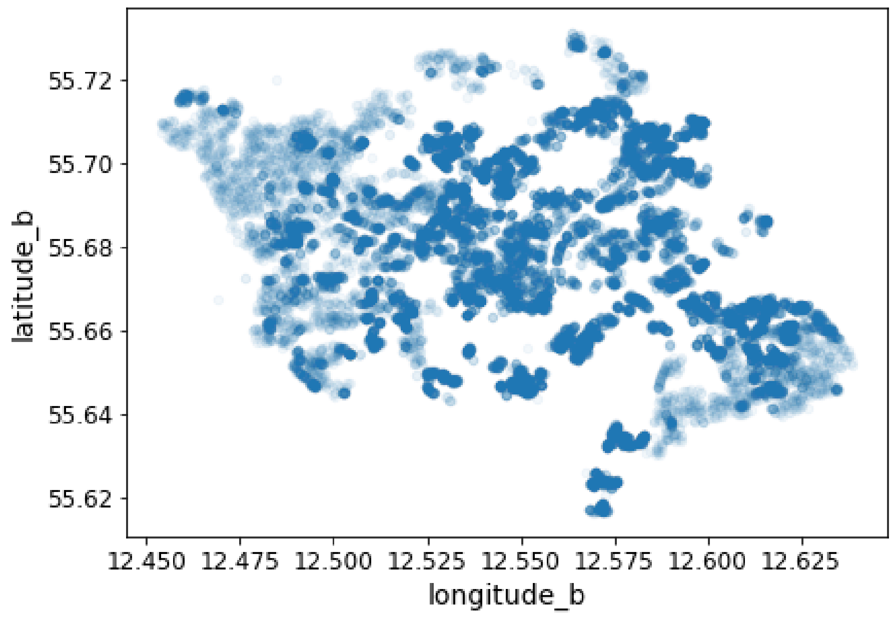


Figure 3.1: Every scraped housing by their latitude and longitude.

3.2 Data Handling

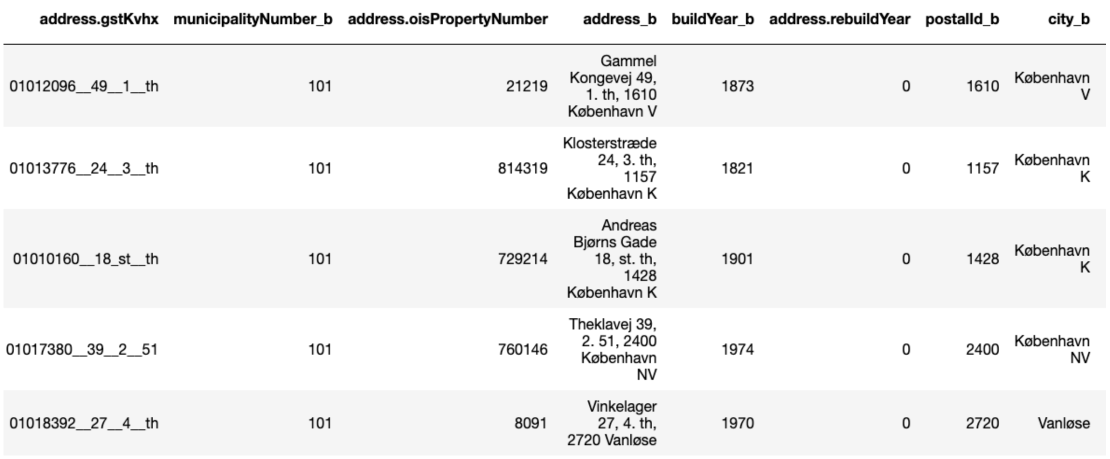
3.2.1 First Glance

We now have our data set. Believing scraping from different web pages will align data perfectly and having data for each housing will be very naive. Obviously, our gathered dataset is not ready for model work yet. We will first make the data workable, including deleting a lot of addresses from the data set, e.g., because of duplicates or simply missing data.

We will do the minimum preparation of the data. As our first task we load our file, **raw-Data_final.csv**, and take a good look at the data.

Shape of data	
Sample Space:	57011
Feature Space:	219

The size of the data seems rather reasonable. Though when inspecting other machine learning projects the sample size seem to be on the lower end of the spectrum. We go into the data further quickly notice some issues.



	address.gstKvhx	municipalityNumber_b	address.oisPropertyNumber	address_b	buildYear_b	address.rebuildYear	postalId_b	city_b
	01012096__49__1__th	101	21219	Gammel Kongevej 49, 1. th, 1610 København V	1873	0	1610	København V
	01013776__24__3__th	101	814319	Klosterstræde 24, 3. th, 1157 København K	1821	0	1157	København K
	01010160__18_st__th	101	729214	Andreas Bjørns Gade 18, st. th, 1428 København K	1901	0	1428	København K
	01017380__39__2__51	101	760146	Theklavej 39, 2. 51, 2400 København NV	1974	0	2400	København NV
	01018392__27__4__th	101	8091	Vinkelager 27, 4. th, 2720 Vanløse	1970	0	2720	Vanløse

Figure 3.2: Example of our data showing first five rows and first eight columns

In our superficial investigation we found a lot of duplicates which are all of the houses being resold within the six year period we have scraped over. Keeping only the latest sold houses of the duplicates and removing empty features puts us at:

Shape of data	
Sample Space:	44805
Duplicates:	12206
Feature Space:	195

This is a huge loss but required nonetheless. We also notice when looking at the feature `’address.latestSale.saleType’` there are different kinds of sales methods `’Fri handel’, ’Familiehandel’, ’I øvrigt’, ’Auktion’, nan`. We only wish to utilize `’Fri handel’` known as free trade since the other parameters will skew our modeling in a direction not wishful[1]. This brings our sample size down to:

Shape of data	
Sample Space:	39964
Non-free trade:	4841
Feature Space:	195

We get Python to give us some info about the data and describe it: First we use `.info()` on our data, and get non-null count and the `dtype` for each feature:

[1]E.g., family trade and Auction deals are usually much cheaper.

Column #	Feature	Non-Null Count	Dtype
0	`address.gstKvhx`	40285 non-null	object
1	`municipalityNumber_b`	40285 non-null	int64
⋮			
34	`propertyValuation_b`	39356 non-null	object
35	`saleDate_b`	40285 non-null	object
36	`salePrice_b`	40285 non-null	object
⋮			
81	`numberOfFloors_b`	36838 non-null	float64
82	`floor_b`	36838 non-null	float64
83	`address.latestForSale.floorName`	31953 non-null	object
⋮			
140	`AVM_pris_d`	38113 non-null	object
141	`Kælder`	3323 non-null	object
142	`Location`	40285 non-null	object
143	`daycare_h`	40285 non-null	float64
⋮			

Table 3.1: An information table showing features, non-Null Count and data types respectively.

What we find especially important is the non-null count. Before we can model at all we need to do something with the nulls. How we handle these empty features for certain samples is different depending on which feature we are looking at. Looking at `Kælder`, i.e. Basement, we realise that only the 3323 samples actually have basements thus keeping zeroes make a great deal of sense.

There are a lot of useless features which we have scraped. All these features will be removed with transformers one can find in the Github repository. We make a pipeline that utilizes all of our transformers so we can quickly look at the data from different angles. A lot of features are removed by doing this, which is not necessarily a bad thing. Considering our sample size it is preferable for us to get rid of useless features. Referencing back to what we wrote about poor quality data.

A few notes to remember and which we will follow are that the more features we keep the more of the sample size will potentially be removed due to features having sample points without data. Fewer houses have data points for every feature. Thus just selecting every possible feature might introduce a bias towards a specific subset of the sample where all data for each feature is available. This depends on whether the missing data points are a problem with scraping or data collectors for the webpages; One could imagine that in certain areas it is easier/better/more needed to look into these features and in other areas the collectors do not bother for some reason. Furthermore, we wish for the features to have minimal correlation since they probably give the same information if they are highly correlated. We also need for our models to work properly. The sample size needs to be significantly larger than the number of features; this is to avoid the curse of dimensionality. Looking further into the data we let Python describe the data giving us the quantiles, min- and max-value, mean, standard deviation and the count.:

	salePrice_b	numberOfFloors_b	alfs_areaBasement	alfs_numberOfRooms	buildYear_b
count	40282.00	36835.00	36835.00	36835.00	40282.00
mean	4077053.28	1.15	6.41	3.14	1919.41
std	2561768.39	1.19	23.06	1.33	216.45
min	50000.00	0.00	0.00	0.00	0.00
25%	2388500.00	1.00	0.00	2.00	1907.00
50%	3500000.00	1.00	0.00	3.00	1936.00
75%	5000000.00	1.00	0.00	4.00	1975.00
max	61150000.00	32.00	893.00	16.00	2024.00

Table 3.2: Table of a quick description of more or less random features to give an idea of the data; `salePrice_b`, `numberOfFloors_b`, `alfs_areaBasement`, `alfs_numberOfRooms`, `buildYear_b`

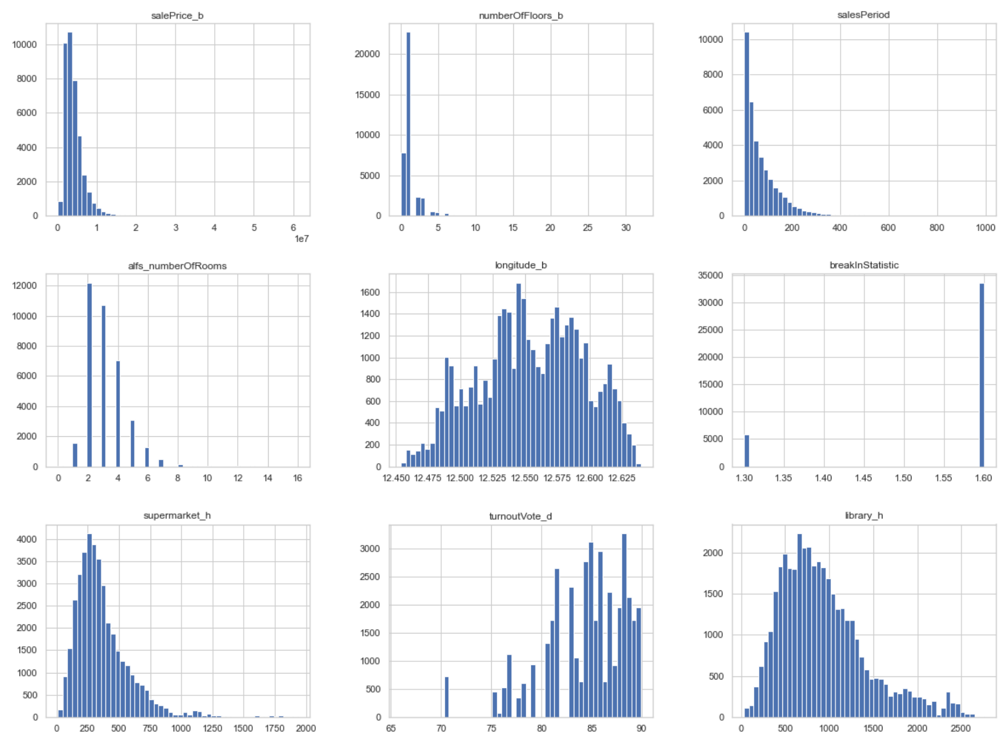


Figure 3.3: Histograms of some more features to look further into them; `salePrice_b`, `numberOfFloors_b`, `salesPeriod`, `alfs_numberOfRooms`, `longitude_b`, `breakInStatistic`, `supermarket_h`, `turnoutVote_d`, `library_h`

This table shows for five features. We know for `salePrice_b` we have no missing values and no zero-values. At first glance we see, some missing values for the other features. Second glance we notice that the mean and the 50% quantile are not too far apart, of course the difference is due to the very large max-value. Hence not a lot of values are this extraordinarily large else the mean would be smaller. A minimum sale of 50.000 ddk also seems rather small in Copenhagen. A very large basement is also noticed and need further investigation to tell if it is an outlier. Maybe the most suspicious of this table is that some houses have been build in 2024. This is obviously a mistake and this certainly needs taken care of. Again referencing what we wrote about poor quality of data. We see throughout the data some very remarkable outliers which we will deal with later on.

We make histograms to try and visualize the data more seen in figure 3.3.

As suspected we have some very heavy tails but in a lot of cases it make sense i.e. it make sense for `supermarket_h` and `library_h` to be very centered around small values. We notice break in statistics only have two values throughout, and will be a removal candidate. Multiple values have issues and will be fixed. But before we look much further into it we need to split the data and have a holdout validation set as explained in the earlier section "Generalization when tuning".

3.2.2 Second Glance

Before we take a deeper look at the data and do feature engineering it is vital to split the data and get a holdout-validation set such that we do not introduce unnecessary bias and end up overfitting.

We set aside 20 percent of the data for validation when we have done all model work. This corresponds to 7993 samples. The data is split by stratifying with regards to each city part of Copenhagen. We do this since we expect that the city part is rather important in predicting sale price. Visualized below

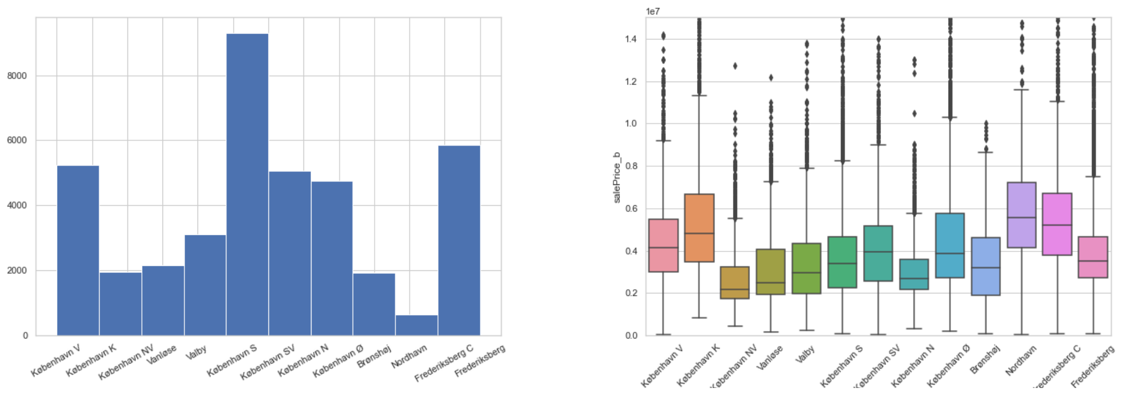


Figure 3.4: The left figure shows the number of sales in each city part as a histogram. The right figure shows a boxplot of the sales prices for each city part.

As expected we see some indication of where one live is a tell to what the sale price is. Thus we stratify such the same amount of each city part in both our training and the holdout validation set is the same percent wise as the initial data.

	Stratified Split		
City part	Initial data	Train data	Holdout-Val
København S	0.23	0.23	0.23
København Ø	0.12	0.12	0.12
Frederiksberg	0.10	0.10	0.10
Valby	0.08	0.08	0.08
København K	0.07	0.07	0.07
København N	0.07	0.07	0.07
København SV	0.06	0.06	0.06
København V	0.06	0.06	0.06
Vanløse	0.05	0.05	0.05
København NV	0.05	0.05	0.05
Brønshøj	0.05	0.05	0.05
Frederiksberg C	0.05	0.05	0.05
Nordhavn	0.02	0.02	0.02

Table 3.3: Stratified split where every number is percentage of the respective data.

We then continue our data analysis with the remaining 80% of the samples corresponding to 31971 samples. Having the number of features of our caliber makes the data analysis rather cumbersome. We thus manually check each feature and make a rather harsh exclusion of features we deem useless. We continue with the following features:

```
'salePrice_b', 'propertyValuation_b', 'paymentCash_b', 'AVM_pris_d', 'Antal Etager',
'BuildingUnion_area', 'OMXC20_s', 'Opførselsesår' , 'city_b', 'outerwall_d',
'WaterHardness', 'aboveSea_d', 'airport_h', 'alfs_area', 'alfs_areaWeighted',
'alfs_buildYear_d', 'alfs_postal', 'areaBasement', 'areaResidential_b',
'breakInStatistic', 'busstop_h', 'coast_h', 'daycare_h', 'doctor_h', 'floor_b',
'forest_h', 'hospital_h', 'junction_h', 'lake_h', 'latitude_b', 'library_h',
'longitude_b', 'metro_h', 'mortgageRate_s', 'numberOfBaths_bd', 'numberOfFloors_b',
'numberOfRooms', 'numberOfToilets_bd', 'pharmacy_h', 'priceChangeMPriorIndex_s',
'priceChangeYPriorIndex_s', 'priceIndex_s', 'propertyCharges', 'publicbath_h',
'radonRiskCategory_d', 'rebuildYear_b', 'roadtrain_h', 'salesPeriod', 'salesYear_b',
'school_h', 'soccerfield_h', 'sportshall_h', 'strain_h', 'supermarket_h', 'train_h',
'turnoutVote_d', 'unemploymentRateCPH_s', 'unemploymentRateDK_s', 'usage_d',
'heating_d', 'itemtypeName', 'biggestParty_d', 'electionArea_d', 'radonRisk_d',
'quarter_b', 'roof_d', 'energyMark_b', 'floodingRisk_d', 'quarter0_b',
'kitchen.content_d', 'noise_d'
```

At first we calculate the correlation for each feature up against the sales price, our label feature. We instead calculate the correlation against the sales price for each numeric feature.

We see a lot of features are highly correlated with the sales price. We will take a closer look at specific features which are highly correlated; `areaResidential_b`, `alfs_area`, `alfs_numberOfRooms`. We note that features for our model are not just selected from the most correlated and down. Some of these features have the same information thus it does not make much sense to include them all. We will use these features with mostly identical information later to make new features.

Pearson correlation by salePrice	
`salePrice`	1.00
`areaResidential_b`	0.83
`alfs_area`	0.81
`alfs_numberOfRooms`	0.70
`Antal værelser`	0.66
`numberOfToilets_bd`	0.52
`numberOfBaths_bd`	0.42
`alfs_areaBasement`	0.30

Table 3.4: The eight highest calculated correlations against the the label, salePrice, excluding the payment features.

3.2.3 Feature Engineering

We have now looked at the features in some different settings and we wish to feature engineer and combine features to make more suitable features. The samples, since scraped from multiple different web pages, contains a few features which are actually almost the identical.

These features as seen to the right are pretty straight forward to manufacture since they are primarily the same. We take the best of the features i.e. the features with the least missing samples and with the most realistic data points, and fill in empty slots with the other similar features. These features were all highly correlated and getting rid of features containing the same information will make the job of the regression models easier. Next up we look at the correlation of the new features and see if we have improvement.

We will see the effects of applying discussed transformations to the samples below. From now on our selection and manufacturing will be based upon model results from a baseline line regression model. Checking if we can improve losses by manufacturing or to see if some manipulation actually worsen the model.

Feature engineering of similar features
`alfs_periodTotal`
`salesPeriod`
`dateRemoved - dateAdded`
`dateRemoved - dateAnnounced`
`Ombygningsår`
`alfs_rebuildYear`
`rebuildYear_b`
`buildYear_b`
`alfs_area`
`Boligstørrelse`
`alfs_areaBasement`
`Kælder`
`alfs_areaWeighted`
`vægtet areal`

Table 3.5: Feature engineering for sales period, rebuild year, housing area, basement area and weighted area. The features have priority from the top and then the features below fill in empty spaces receptively.

3.3 Selection of Data Transformations by Linear Regression Losses

We have made a linear regression model, and the first step we do is to standardized all the non-labels, we do this by scaling the data by:

$$x^{(i,j)}_{scaled} = \frac{x^{(i,j)} - \mu_{x^{(j)}}}{\sigma_{x^{(j)}}} \tag{3.1}$$

where i=1,...,n different samples, j=1,...,d different features, $mu_{x^{(j)}}$ and $sigma_{x^{(j)}}$ is the mean and standard deviation for each the specific feature. Our selected loss measure is the squared loss. We chose to take the square root of the loss when representing it. We validate our data by cross-validation exactly as mentioned earlier. The idea is to get the two losses as small as possible and as close to each other as possible. This will indicate that we can fit the data well and generalization will not be too big of an issue. Immediately our linear regression is terrible and we will do some quick fixing.

	Initial Lin Reg	Log Labels Lin Reg	Log Labels no Outlier
Training rmse:	98e+4	0.2284	0.1535
Mean CV rmse:	33e+8	2811	4008
Min-Max CV rmse:	(92e+4, 33e+9,)	(0.208, 2803)	(0.149, 38170)
Standard deviation:	10e+9	8406	11401
Sample size:	27964	27964	25605

Table 3.6: Linear regression with modified labels and samples: The initial model with no modifications done to the samples, the log labels model is as it states the labels have been log transformed and the last column we have removed outliers.

In the initial linear regression model we notice how we are overfitting since the cross-validation loss is much higher than the training loss. The model is really just unable to explain the data at all. When the labels get log-transformed, we notice a significant improvement for both losses. Still, indication of overfitting, meaning this model would not generalize well. Next, outliers are removed as noted earlier. Further we choose to remove houses that are too expensive or too cheap considering the area. Hence we calculate the square meter price and remove the 0.05 and 0.95 quantiles. As the QQ-plots below show, the samples plausibly follow a normal distribution without these outliers.

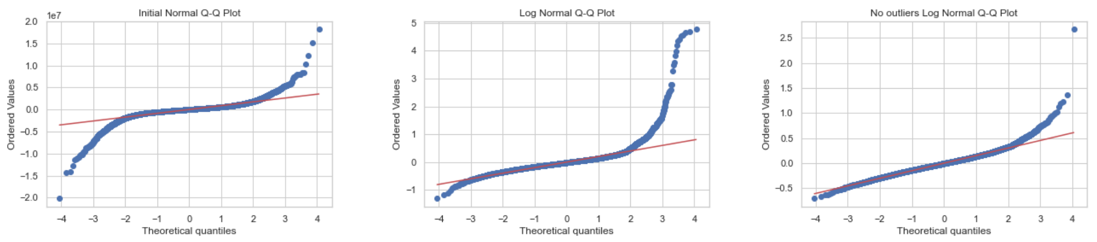


Figure 3.5: Three QQ-plots for each model as in the table above.

We choose to combine sales price and area of the housing to find square meter price and remove outliers and not just take the highest and lowest valued housing to make sure we do not introduce clear lines for maximum prices or minimum prices. We will compare a scatter plot below:

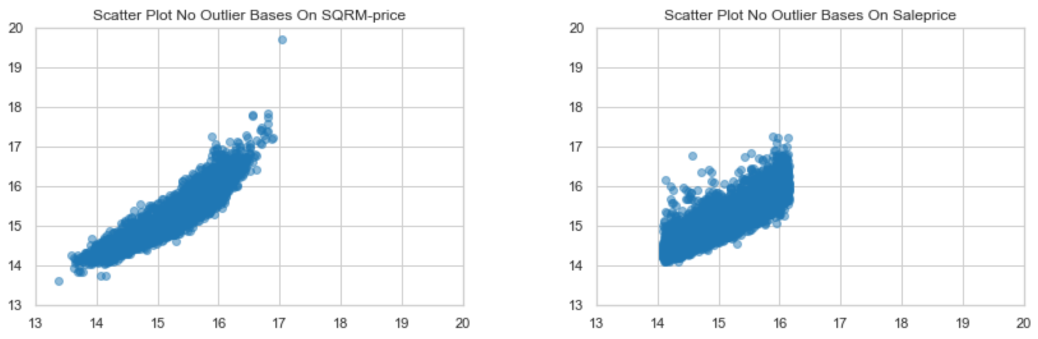


Figure 3.6: Scatter plots. The left with outliers removed by square meter price. The right the outliers are removed by sales price.

The biggest problem with just removing by price alone is that these houses might be naturally high because they are as tremendous as their price. We do not wish for these naturally expensive houses to be excluded. Further, a more complex prediction model might set limits on the houses; hence if we try to predict larger or smaller houses, the model will automatically just set prices, disregarding the actual inputs to a degree. One could combine even more features for a more complex removal of outliers but we are satisfied.

To further improve out model we introduce the feature engineering as explained in previous section.

	Feature Engineered	Imputation introduced	None Info Removed
Training rmse:	0.1539	0.1539	0.1549
Mean CV rmse:	2628	2628	0.1561
Min-Max CV rmse:	(0.1491, 26281)	(0.1491, 26281)	(0.1501, 0.1634)
Standard deviation:	7884	7884	0.0043
Sample size:	25605	25605	25615,

Table 3.7: Linear regression for three further modified models. First column with feature engineering applied. Second column with imputation. Third column with removal of features with no information.

In this table we get some great improvement on the loss. The feature engineering which is done is explained prior. We notice a drop in our cross-validation estimates and in the standard deviation, clearly a good sign. Imputation is done by replacing all samples for specific features with the most common value in that specific feature. This is done to `kitchen.content_d, outerwall_d', roof_d'`. We see no change, which indicates all the missing samples for these features have been removed by other modifications on the data. The reason we do not impute on other features is due to how we wish to do our analysis. One can impute missing values or exclude them. We want to make a close to complete case analysis, hence we do not wish for a lot of imputing. These three features we choose to impute on is because of how almost all houses are of the same category within them.

We choose to remove features with none or close to no information information which is not given by other kept features. The following features are removed: `longitude_b, latitude_b, city_b, Antal Etager, WaterHardness, quarter0_b, Opførselsesår, itemTypeName`, `BuildingUnion`, `areaResidential_b`. The first three are removed because of how we wish to have the postalID pinpoint where the house located. We realized `Antal Etager` is flawed hence removed and `WaterHardness` have too little information. `quarter0_b, Opførselsesår, itemTypeName`, `areaResidential_b` all have superior features which hold most of the information these gives. `BuildingUnion` is misrepresenting apartment unions compared to single houses.

We see an extraordinarily improvement in prediction we see if we can do any better by removing some of the highly correlated variables as visualised below by a pearson plot.

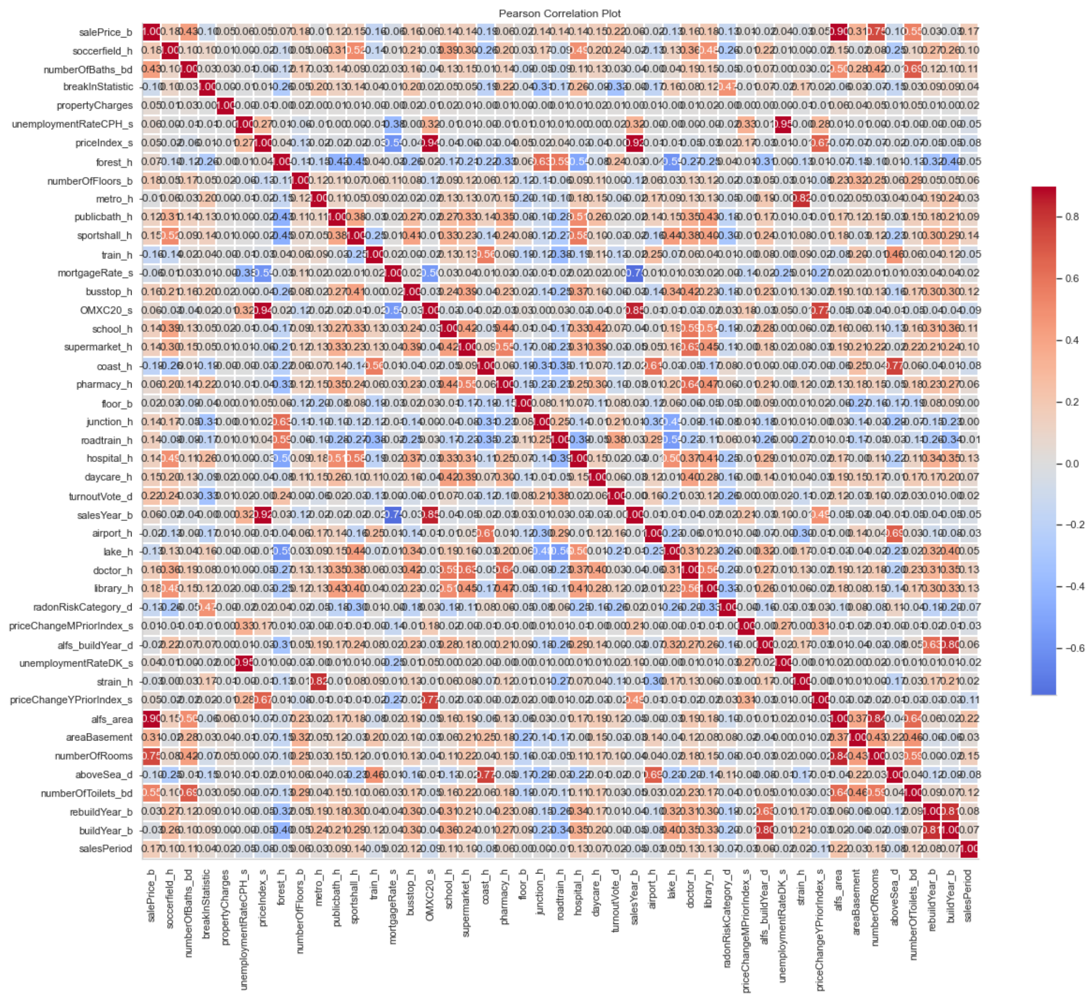


Figure 3.7: Pearson correlation plot of all numerical features.

	High Corr Removed	Best Feature Selection	Rare Categorical
Linear rmse:	0.1555	0.1595	0.1597
Mean rmse:	0.1567	0.1603	0.1604
Min-Max CV rmse:	(0.1505, 0.1643)	(0.1547, 0.1687)	(0.1550, 0.1687)
Standard deviation:	0.0043	0.0044	0.0044
Sample size:	25615	25615	25615

Table 3.8: Linear regression for further modified samples. First by removing features with high correlation. Second by removing worst mutual information score features. Last to assemble rare occurrences into rare categories for their respective feature.

This table shows three further modifications. When we look at the correlation plot above we notice few variables are highly correlated. But nothing too problematic. We removed `unemploymentRateDK_s` sine it is very correlated with `unemploymentRateCPH_s`. Further we remove `priceChangeYPriorIndex_s` and `alfs_buildYear_d`. And see only a very slight difference in the root mean square error.

We then use mutual information regression on all the features to see if any are neglectable. It is a way of calculating the mutual dependence between the feature and the label. I quantifies the amount of information obtained about one variable by observing the other. We refer to this for a more detailed explanation [Mutual Information]. This approach is used because we wish to get rid of features with no predictive power. We note that all categorical features are also involved here, this means we are calculating the mutual dependence for all one-hot encoded features i.e. `electionArea_d_6. Vest`. This calculation has randomness upon it; hence we get slightly different results every time hence we do the calculations multiple times and see if we can find some connections between the zero values.

Features	Mutual Information Score
heating_d_Ovn til fast og flydende brændsel	.00023
electionArea_d_6. Nord	.00017
electionArea_d_5. Syd	.00011
roof_d_Stråtag	.00002
heating_d_Centralvarme med to fyringsenheder	.00002
kitchen.content_d_Ingen fast kogeinstallation	.00000
heating_d_Centralvarme med én fyringsenhed	.00000
electionArea_d_6. Øst	.00000
electionArea_d_6. Vest	.00000
electionArea_d_5. Vest	.00000

Table 3.9: Mutual information scores for 10 features with the smallest value.

Doing the computations over and over we notice these features are all very close to zero every time thus we remove them. We remove all heating, kitchen content and election area, one hot features. The first two have little to no information. The election area actually have some information but we deem it to be in the postal code feature. Furthermore when one hot encoding election area we get an extra 61 variables which we very much wish to get rid of if no harm is done to the prediction. We notice a very slight difference in the root mean square loss but we deem it to be a fair trade off.

In the last column, we have not chosen to remove the roof feature because we can feature engineer the rare occurrences in each feature into a rare category. We do this for *outerwall_d* and `roof_d` with a tolerance of 0.015 e.g., all categories within these features which are present less than 1.5% are all categorised into a rare category done for the the features respectively. By doing this we get rid of the last zero information category feature and we reduce the number of features to 82. A small note we make is, our approach is and was not to find most useful predictors but to get rid of useless ones.

This concludes our data processing steps and we see how this linear model is somewhat effective at predicting the sale prices of housing. Data preprocessing is an endless art, and more could have been done. It is really a question of satisfaction.

3.4 Linear Regression

We wish to optimize our linear regression model and we do this by applying some regularization if we can get a better performance. First lets check in more detail just how well the regular Linear regression works on our final data.

Loss Measures	Scores
Training rmse:	0.1597
Mean CV rmse:	0.1604
Mean Relative error	0.1232
Mean Absolute error	526608.8510

Table 3.10: Linear regression. The training root mean square, mean cross validation rmse, mean relative error and mean absolute error.

In the table we see the scores of a standard linear model running on our samples, remembering the pipeline the samples have gone through. We need to keep an eye out for the Linear rmse against the mean cross-validation rmse since this is our sign of how well the model will generalize. Without much difficulty, the model seems to generalize somewhat well. We also notice that the linear model predictions only deviate about 12% percent from the actual prices. We make a scatter plot

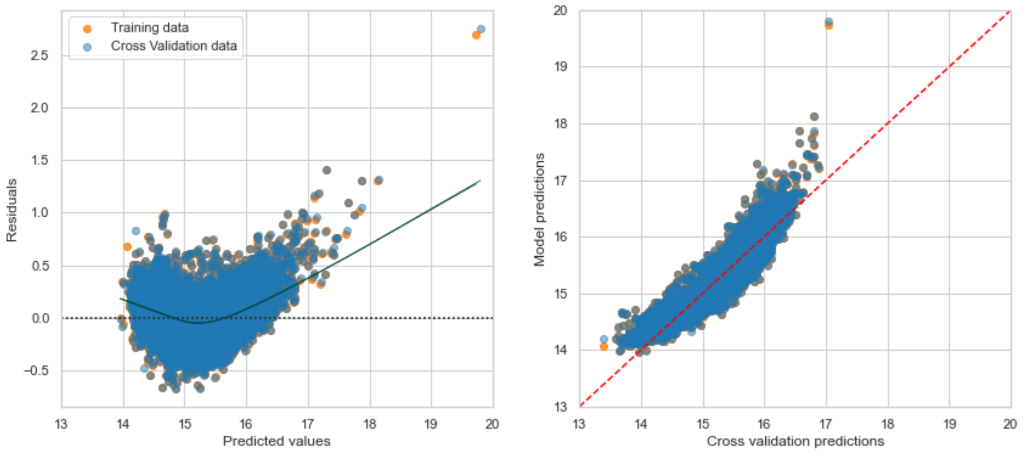


Figure 3.8: The linear regression model. Left we see a residual plot comparing Training data predictions and Validation sets predictions. Right we see a scatter plot of training predictions and cross-validation predictions

This is very formidable for just a linear model. Even though these losses are so close to each other we wish to experiment with some regularization. We try lasso regularization:

This is a job of setting a penalizing term as the ℓ_1 norm with a multiplicative constant, α_lasso. We show different values of alpha with the Lasso linear regression:

Loss Measures	Scores					
α_{lasso}	0.0000	0.0001	0.0010	0.0100	0.1000	1.0000
Training rmse:	0.1597	0.1600	0.1622	0.1748	0.2524	0.4704
Mean CV rmse:	0.1604	0.1606	0.1627	0.1750	0.2525	0.4704
Mean Relative error	0.1232	0.1235	0.1259	0.1388	0.2137	0.4094

Table 3.11: Showing different losses for Lasso linear regression with α_{lasso} values of 0, 0.0001, 0.001, 0.01, 0.1 and 1 respectively.

We see how scores actually get progressively worse as α_{lasso} rises, indicating that we do not wish to penalize on the prediction parameters. Lasso linear regression lets the prediction parameter values be zero. It makes feature selection, and this means the number of features included in the model decreases. When $\alpha_{lasso} = 1$, we have a straight line meaning all parameter values are zero.

Amount of $\theta_i = 0$ for different α_{lasso}						
α_{lasso}	0	0.0001	0.001	0.01	0.1	1
# $\theta_i = 0$	0	6	30	57	77	80(all)

Table 3.12: How many parameter values are set as zero depending on the α_{lasso} value.

How quickly a lot of these parameters gets set to zero and how little the loss actually change is a huge indication of how we could have been much harsher in our preprocessing state since it seem like a lot of these features are not actually very big predictors of house prices.

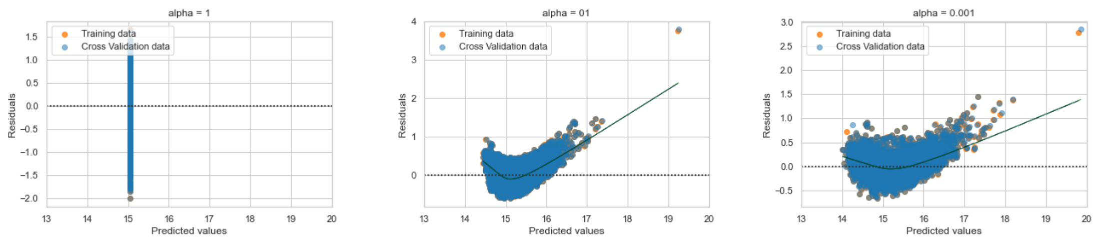


Figure 3.9: Linear regression model. Residual plots comparing training prediction and cross validation prediction for different values of α

Above we see three residual plots with different values of α_{lasso} in the Lasso linear regression and we conclude that we are going forward with the linear model without any penalization.

3.4.1 Polynomial Expansion

Now we wish to expand this linear model to a non-linear regression model i.e. we wish to expand the model to a polynomial model of second degree. As to why we will not check for a higher degree than second will become clear very soon.

Transforming the features into a polynomial of second degree means we go from 80 features to 3320. This is because we cross products of all features. Having information about each interaction with each other features. This a lot of features and just below a magnitude of ten away from our sample space, this can be seen as of the losses.

Loss Measures	Scores
Training rmse:	0.0924
Mean CV rmse:	167797.7831
Mean Relative error:	0.0696
Mean Absolute error:	272342.5116

Table 3.13: Polynomial linear regression. The training root mean square, mean cross-validation rmse, mean relative error and mean absolute error.

As expected the polynomial model can fit the training set much better though we do see a great difference between the training rmse and the cross validation rmse which is greatly indicating we are overfitting. This makes much sense since we went from 80 features to 3320 features. We illustrate with a residual plot below:

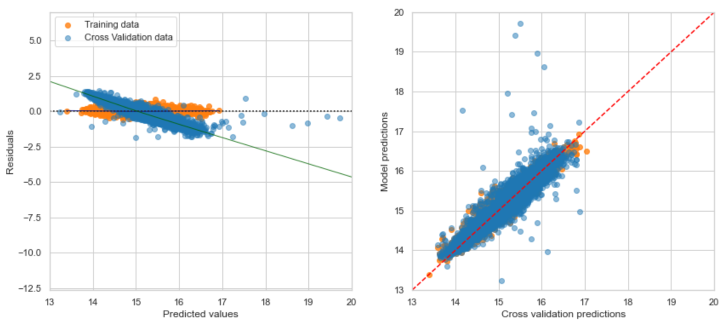


Figure 3.10: Polynomial regression without any regularization. Left we see a residual plot comparing Training data predictions and Validation sets predictions. Right we see a scatter plot of training predictions and cross validation predictions

Looking at the figure we notice how the polynomial model is able to compensate for the non-linearity in samples. We now introduce regularization such we can attempt to get better expected generalization for this model. We will try to penalize with Lasso regression forcing a lot of the parameters to be zero thus making the model less complex. We make similar regression table as above with lasso regularization :

Amount of $\theta_i = 0$ for different α_{lasso} and scores respectively						
α_{lasso}	0	0.0001	0.001	0.01	0.1	1
# $\theta_i = 0$	89	2173	2953	3229	3292	3320(all)
Training rmse:	0.0924	0.1031	0.1156	0.1383	0.2168	0.3042
Mean CV rmse:	167797.7831	0.1230	0.1201	0.1397	0.2172	0.3044
Relative loss:	0.0696	0.0777	0.08815	0.1074	0.1705	0.2520

Table 3.14: How many parameter values are set as zero depending on the α_{lasso} value and their rsme and relative loss respectively.

The Lasso algorithm introduces a convergence error when computing with small values of α_{lasso}, which we will take somewhat lightly upon and rather look at the difference in the training versus CV validation rmse. Looking at the table when $\alpha_{lasso} = 0.001$ we start overfitting more and more. Thus for us to try and keep a good generalization we will continue with the $alpha_{lasso} = 0.01$ model. We did try with a value between 0.001 and 0.01 but had problems with convergence and again for the sake of generalization we set our final polynomial model of second degree as the lasso regression with $\alpha_{lasso} = 0.01$. And we expect it to make about two percent point better generalizations than the linear model when using the test (holdout validation) set.

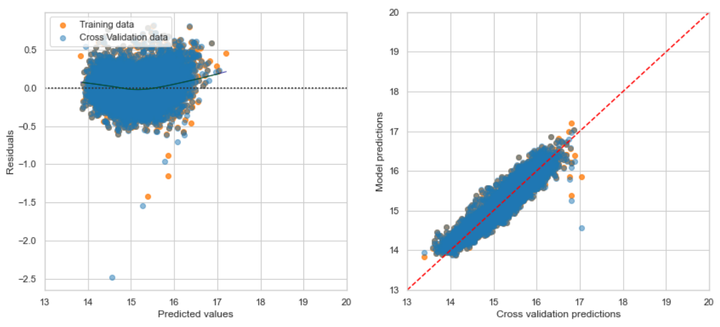


Figure 3.11: Polynomial regression with lasso regularization, $\alpha_{lasso} = 0.01$. Left we see a residual plot comparing Training data predictions and Validation sets predictions. Right we see scatter plot of training predictions and cross validation predictions

We notice how the cross the residual plot is now rather nice. It catches almost all of the non-linearity.

3.5 Random Forest

It actually seems like we can get somewhat close at estimating the sales prices and we will check whether we can come even closer with a random forest regression model. The first go with default hyper parameters we get

Loss Measures	Scores
Training rmse:	0.0418
Mean CV rmse:	0.1124
Mean Relative error:	0.0308
Mean Absolute error:	125362.0276

Table 3.15: The training rmse, mean cross validation rmse, mean relative error and mean absolute error for the random forest regression. Hyper parameters are: n_estimators=100, min_samples_split=2, min_samples_leaf=1, max_features=1.0.

When looking at the mean absolute and mean relative loss we notice how they are way better than what we have seen before. We mustn't get distracted and remember to compare the training set and the validation sets' rmse. Here we notice a clear indication that this model is overfitting the training set compared to the validation sets. We will illustrate the model below.

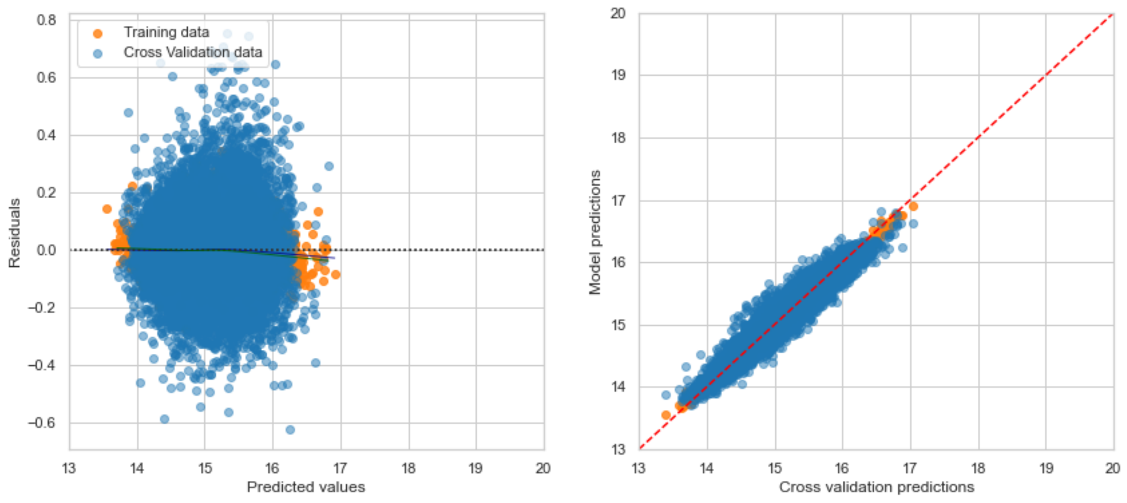


Figure 3.12: Random forest regression with hyperparameters: n_estimators=100, min_samples_split=2, min_samples_leaf=1, max_features=1.0. Left we see a residual plot comparing Training data predictions and Validation sets predictions. Right we see scatter plot of training predictions and cross validation predictions

Here we notice the validation estimates are clearly more prominent in the extremes. We need to hyper-parameter-tune this model as the non-corresponding rsme's, thus overfitting, is an indication of bad generalization.

There are multiple ways to go about it. This we know we are overfitting meaning the model is too complex for the samples. Thus, changing the hyperparameters and making the random forest algorithm less complex will be a start. We will do a grid search and let this grid search find the best model by comparing the loss on the cross-validations. To be more clear and because we do not precisely know where to search, we will not search throughout all these parameters listed below because it is too exhausting. Instead we do a random grid search where a random combination of the hyperparameters are selected and what we choose is how many different combinations we run through.

Hyper Parameter	Search Space
n_estimators:	{50, 100, 150, 200, 250, 300}
max depths:	{4, 8, 12, 16, 20}
min_samples_split:	{20, 60, 100}
min_samples_leaf:	{5, 10, 15}
max_features:	{2, 3, 4, 5, 6, 7, 8}

Table 3.16: The span of the random grid search

We let the grid search run for 200 iterations which is arguable a small run considering we have $6 \times 5 \times 3 \times 3 \times 7 = 1890$ combinations. None the less we find a estimate of the best possible prediction rule, h, with the hyper parameters: max_depth=16, max_features=8, min_samples_leaf=5, min_samples_split=20, n_estimators=200. Giving the following losses:

Loss Measures	Scores
Forest rmse:	0.1022
Mean CV rmse:	0.1263
Mean Relative error:	0.0762
Mean Absolute error:	310135.9108

Table 3.17: The rmse, mean cross validation rmse, mean relative error and mean absolute error for the random forest regression. Hyper parameters are: max_depth=16, max_features=8, min_samples_leaf=5, min_samples_split=20, n_estimators=200.

We actually found a solution where we overfit less which is great, we do yet another grid search where we focus around this solution. And to minimize the different between the training rmse and CV validation rmse. We get the following parameters: max_depth=14, max_features=60, min_samples_leaf=30, min_samples_split=150, n_estimators=100. We get the following losses:

Loss Measures	Scores
Forest rmse:	0.1248
Mean CV rmse:	0.1337
Mean Relative error:	0.0968
Mean Absolute error:	389895.2025

Table 3.18: The rmse, mean cross validation rmse, mean relative error and mean absolute error for the random forest regression. Hyper parameters are: max_depth=14, max_features=16, min_samples_leaf=30, min_samples_split=110, n_estimators=100

With a corresponding residual plot:

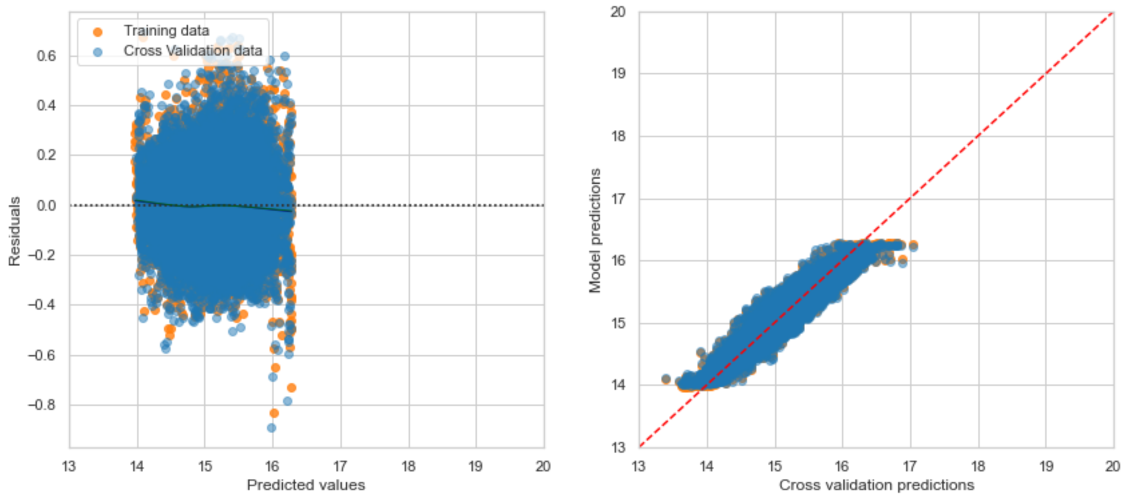


Figure 3.13: Random forest regression with hyperparameters: max_depth=14, max_features=16, min_samples_leaf=30, min_samples_split=110, n_estimators=100. Left we see a residual plot comparing Training data predictions and Validation sets predictions. Right we see scatter plot of training predictions and cross validation predictions

Looking at the plot above we see, this random forest model with these specific hyperparameters neglects the more extreme samples, which we take into consideration when looking at the generalization. We further note that the difference between the training loss and the mean CV loss is not as small as previously seen. Across all different hyperparameters trying to find a minimum mean CV loss that is very close to the training loss this is what we estimate as the optimal. We noticed most of the computations with different hyperparameters gave different rsme values but surprisingly almost the same difference in the rsme values.

3.6 Neural Network

We wish to see if we can get a Neural Network model to perform better on the data than we have seen previously. We have some concerns with the sample size since the neural network's forte is really to utilize a huge amount of samples. In our first attempt at making a prediction rule we use the hyperparameters: 1 hidden layer, 40 neurons, learning rate equal to 0.003 and a batch size of 64.

Loss Measures	Scores
Training rmse:	0.4053
CV mean rmse:	0.1888
Mean Relative error:	0.2895
Mean Absolute error:	1163400.8135

Table 3.19: The rmse, validation rmse, mean relative error and mean absolute loss for the neural network regression. Hyperparameters are: 1 hidden layer, 40 neurons, learning rate equal to 0.003 and a batch size of 64.

We see how we are underfitting and thus the model is not complex enough. We need to note that when fitting this model, we included a callback function that stops the algorithm if the validation loss has not improved within the last ten epochs. This is why we are underfitting so much. We need to either get rid of this callback function or change sensitivity for stopping i.e., make sure it has to run 20 or 30 times without improvement before stopping. We choose to set this sensitivity to 25 and notice how the models are now most of the time running through the total amount of epoch which is 500. We wish to hyperparameter tune this model as well, we do this by grid searching as of the random forest though this time we do not random grid search since the parameter space is somewhat smaller.

Hyper Parameter	Search Space
n_hidden:	{1, 2, 3, 4, 5, 6}
n_neurons:	{20, 30, 40, 50, 60}
learning rate:	{0.0003, 0.0008, 0.003}

Table 3.20: The span of the grid search for neural network hyper parameters.

By running through these $6 \times 5 \times 3 = 90$ models callback sensitivity be 25 and max epochs be 500 the estimated optimal model i.e. the model with the lowest root mean squared loss is with the parameters: 4 hidden layers, 30 neurons in each layer and a leaning rate of 0.003. We get the following scores:

Loss Measures	Scores
Training rmse:	0.1199
CV mean rmse:	0.1242
Mean Relative error:	0.0914
Mean Absolute error:	372901.1244

Table 3.21: The rmse, validation rmse, mean relative error and mean absolute loss for the neural network regression. Hyper parameters are: 4 hidden layer, 30 neurons, learning rate equal to 0.003, batch size of 64 and running for 500 epochs.

And the corresponding residual plot.

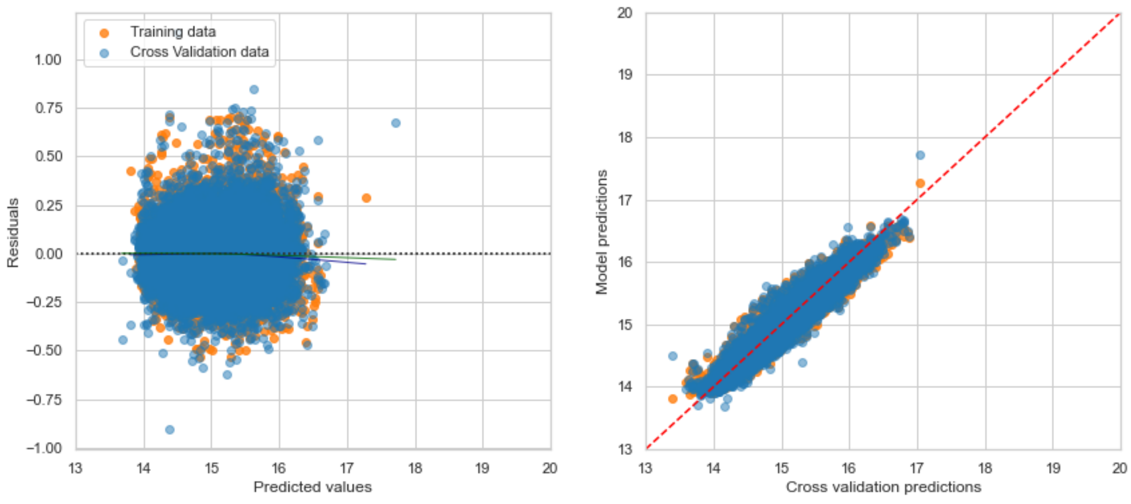


Figure 3.14: Neural Network regression with hyperparameters: 4 hidden layer, 30 neurons, learning rate equal to 0.003, batch size of 64, and running for 500 epochs. Left we see a residual plot comparing Training data predictions and Validation sets predictions. Right we see scatter plot of training predictions and cross validation predictions

We see how we found a model which we expect to generalize well. It outperforms all seen models so far and thus we are satisfied with this model. If we do analysis of most useful features it is possible for us to try manually increase the weights on said features. This could potentially increase prediction results.

3.7 Ensemble Method

We have used four different models and this was all for a reason, we wish to use the output from all the other models and feed it into this meta-model, the ensemble stacking model. We attempt to get an even better performance than seen by the Neural network.

Loss Measures	Scores
Training rmse:	0.1090
CV mean rmse:	0.1117
Mean Relative error:	0.0825
Mean Absolute error:	341657.1699

Table 3.22: Ensemble Stacking Regression using ridge as meta predictor, combining linear, lasso polynomial linear, random forest and a neural network.The training rmse, cross validation rmse, mean relative error and mean absolute loss for the neural network regression.

Furthermore, the corresponding residual plot is very pleasing.

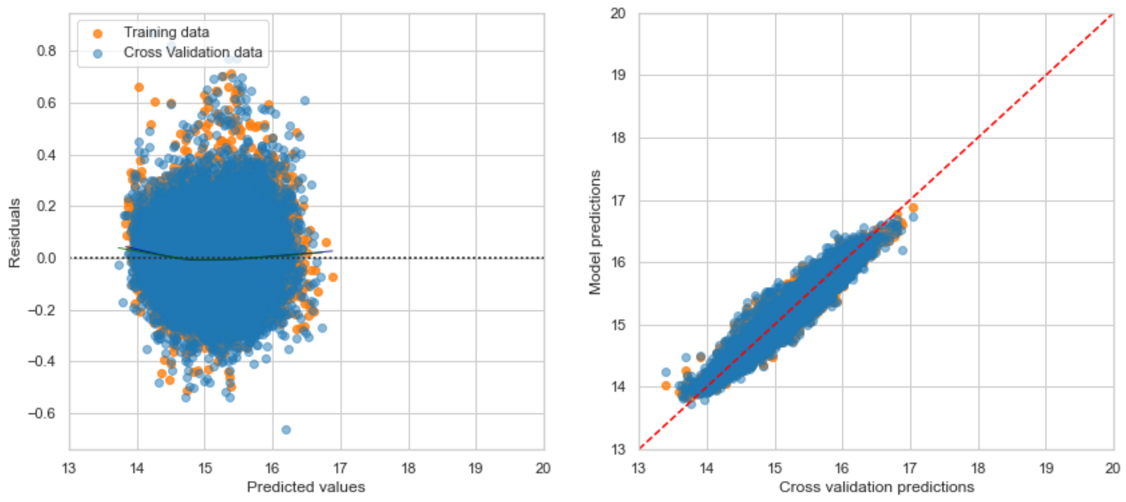


Figure 3.15: Ensemble Stacking Regression using ridge as meta predictor, combining linear, lasso polynomial linear, random forest and a neural network. Left we see a residual plot comparing Training data predictions and Validation sets predictions. Right we see a scatter plot of training predictions and cross-validation predictions

Running this Ensemble stacking model we see how we expect this model to be able to use all the other models and outperform each of them. In the algorithm ridge regularization is used. We internally cross-validate on multiple values of α_{ridge}. Moreover, the model is automatically using the best hyperparameter value. We expect this model to generalize well considering the training and mean validation sets mean loss. We plot the first 100 predictions of each model below.

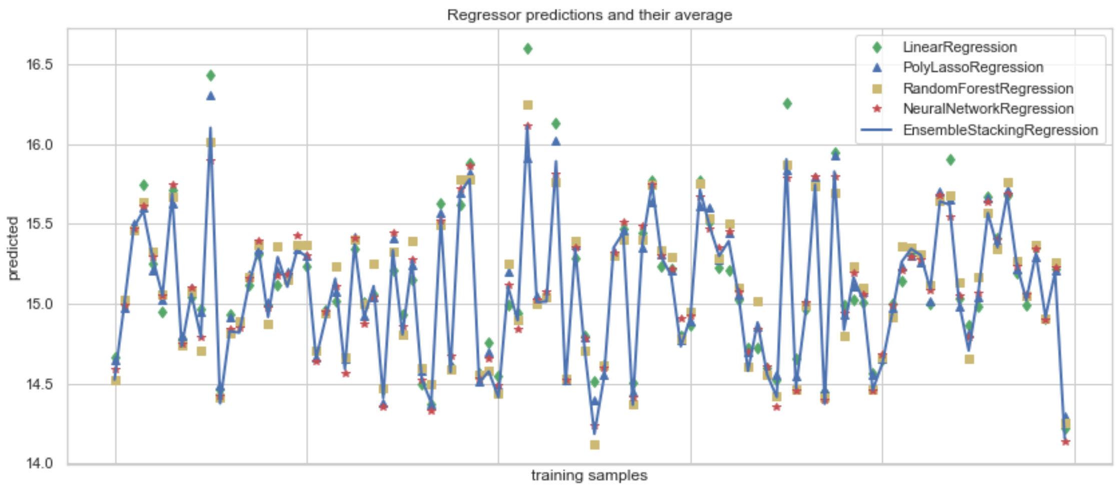


Figure 3.16: How we are preprocessing our data

We see they are all making very similar predictions, though the linear regression is most definitely outputting predictions deviating the most. As expected from the results of the training loss and the residual plot. We see how the Ensemble Stacking model follows the neural network the closest but actually uses the other models to make predictions that are slightly different. Surprisingly we see for some estimates the Ensemble Method is not actually between the highest model prediction and lowest model prediction. Meaning it is able to utilize the differences in the input models to make better predictions very well.

3.8 Results

We have tuned and selected a qualified guess on the best parameters for each model and it is time to see if they are generalizing well. We expect them to do so. Especially due to how well we match the training loss with the cross validation mean loss. For the sake of visualization we make all the residual plots for the generalization to the test set.

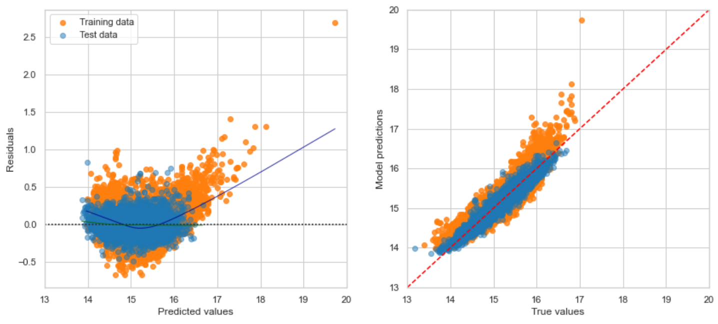


Figure 3.17: The linear regression model. Left we see a residual plot comparing Training data predictions and holdout validation set predictions. Right we see scatter plot of training predictions and True values

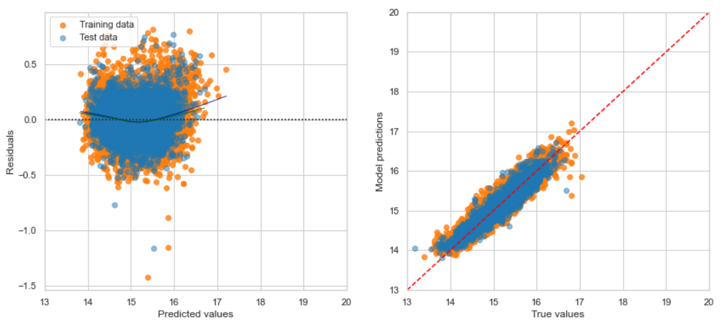


Figure 3.18: Polynomial regression with lasso regularization, $\alpha_{lasso} = 0.01$. Left we see a residual plot comparing Training data predictions and holdout validation set predictions. Right we see scatter plot of training predictions and True values

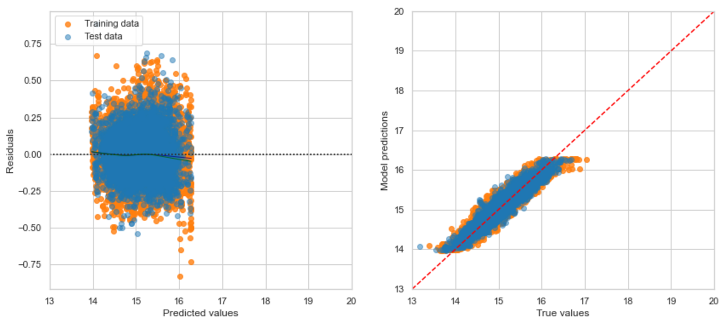


Figure 3.19: Random forest regression with hyperparameters: max_depth=14, max_features=16, min_samples_leaf=30, min_samples_split=110, n_estimators=100. Left we see a residual plot comparing Training data predictions and holdout validation set predictions. Right we see scatter plot of training predictions and True values

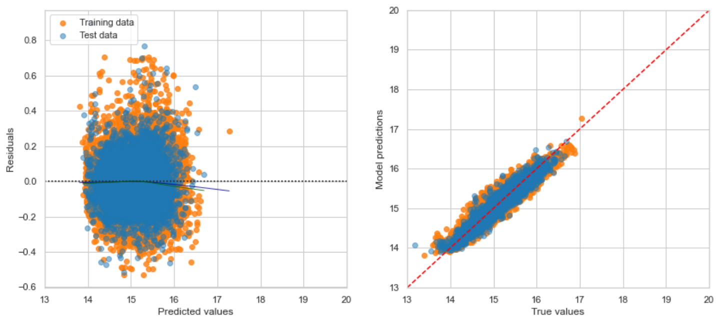


Figure 3.20: Neural Network regression with hyperparameters: 4 hidden layer, 30 neurons, learning rate equal to 0.003, batch size of 64, and running for 500 epochs. Left we see a residual plot comparing Training data predictions and holdout validation set predictions. Right we see scatter plot of training predictions and True values

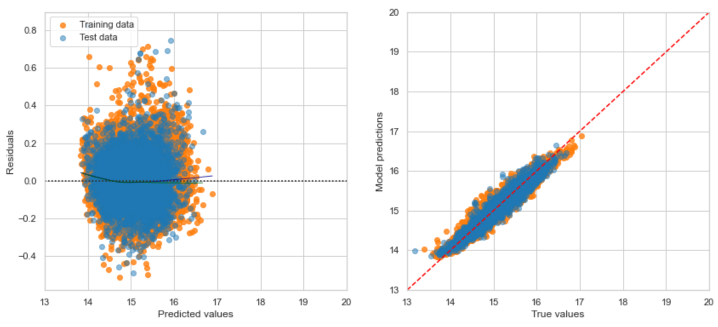


Figure 3.21: Ensemble Stacking Regression using ridge as meta predictor, combining linear, lasso polynomial linear, random forest and a neural network. Left we see a residual plot comparing Training data predictions and holdout validation set predictions. Right we see scatter plot of training predictions and True values

First of is the Linear regression model:

	Training			Holdout Validation		
	RMSE	MRE	MAE	RMSE	MRE	MAE
Linear	0.1597	0.1232	526608.8510	0.1592	0.1227	509559.0709
Lasso Polynomial	0.1383	0.1074	427708.4603	0.1397	0.1081	424951.4983
Random Forest	0.1248	0.0968	389895.2025	0.1319	0.1032	406411.2936
Neural Network	0.1199	0.0914	372901.1244	0.1221	0.0938	372436.5360
Ensemble Stacking	0.1090	0.0825	341657.1677	0.1172	0.0889	362492.4838

Table 3.23: The root mean square loss, mean relative error and mean absolute error for the training and holdout validation set respectively. Best performance models are seen.

	Within 20 %	Within 15 %	Within 10 %
Linear	83.4013	70.5486	52.9310
Lasso Polynomial	86.1755	75.0000	56.6458
Random Forest	87.1630	76.8495	59.5611
Neural Network	91.3950	81.3166	62.5078
Esemble Stacking	93.1505	83.8245	64.1693

Table 3.24: Generalization estimates within 20%, 15% and 10% respectively for each model. All numbers are percentages.

We conclude our model work by summarizing. We started with some rough samples which needed a great deal of handling. Entering the universe of preprocessing which potentially is a neverending process, we found ourselves satisfied. By going through the pipeline seen from table 3.6 trough table 3.8 making decisions based on the squared loss, or to be more specific the root mean square loss. We then started the proper modeling with our benchmark model being the linear regression model. We saw how it was not terrible at predicting the house pricing in Copenhagen with a mean relative error of 0.1227 and being able to contain 83.3% guesses within a margin of 20%. We saw it could not quite catch the nonlinearity in the data. Thus we used a polynomial regression of second degree which, by applying lasso regression, was able to catch the nonlinearity in the data and ended up with an even better MRE of 0.1081. We then moved on to the random forest model which we used for two reasons. First because of how we expect its discrete decision-making on the features to catch the data well. In addition, the last model is an ensemble of all the models and we wish to utilize different models within this meta-model. The random forest model exceeded the polynomial model with an MRE of 0.1032, and as expected had the most considerable difference between the loss looking at training to holdout. On top of this, predictions with a neural network were made. Moreover, as expected it yet again exceeded with a greater performance of losses including the MRE of 0.0938. Last we ensemble all the models and used a ridge regression as a meta learner taking the predictions of all the other models as input and rightfully guessed it outperformed all the other models with an MRE of 0.0889.

Are we actually satisfied with the models? Can we improve the models even further? A short answer to the first question is yes due to how we arrive at predictions results exceeding similar seen attempts at predicting house pricing. As to why, we should not be satisfied, could be answered by the second question. The process of finding optimal hypothesis, h^*, which is to find the best possible generalization rule given the limitations of the specific model is a never-ending process in practice. Besides a model is only as good as the data it receives. In other words and a typical machine learning saying: garbage in garbage out. Hence feature engineering more exhaustively could potentially optimize the predictions. Especially for the random forest model and the neural network we grid searched to find best practice model hyperparameters, this can be done more exhaustively for better performance. As discussed in previous chapter we can also potentially optimize the prediction rule for each model by trying to find new features which predictive power is an improvement for the sale price in Copenhagen from 2016 to 2021. Below are two scenarios of trying to improve the predictive power of the samples.

A little discussion of asking price and AVM-price as predictors

As mentioned we did not wish to use these features as predictors. We wished for the characteristics of the house to speak for themselves. Getting the best possible predictions without using the asking price and AVM-price[2]. These two features are both made by professionals. Real estate agents set the asking price and the AVM-price is a model prediction made by professional data scientists. We calculate the losses between the asking price and sale price and between AVM-price and sale price:

	Asking Price			AVM-Price		
Year	RMSE	MRE	MAE	RMSE	MRE	MAE
2016-2021	0.1156	0.0483	185873.9628	0.2203	0.2065	749357.8581
2017-2021	0.1162	0.0477	186517.3791	0.1911	0.1761	651273.6114
2018-2021	0.1141	0.0470	182571.3643	0.1680	0.1505	557790.0100
2019-2021	0.1164	0.0467	181857.9761	0.1465	0.1265	471663.5916
2020-2021	0.1188	0.0453	181080.3564	0.1015	0.0830	318725.1477
2021-2021	0.1333	0.0477	192341.8060	0.0510	0.0343	139111.3314

[2]For more information of the AVM-price see [Geomatic, 2021]

In the table above we see how the Asking price, or the listing price in other words, is very close to the actual sale price. It is a very close approximation of the actual price for all the different periods. The story is a bit different for the AVM-price. This estimate is quite bad, containing years prior to 2020 and our estimates are performing overall a lot better than this model. However, when just looking at the latter years the AVM-prices outperform our model significantly. We can see how models including these features are performing in [Madsen, 2021]

Predicting with Corona

A potential predictor in the model is Corona. The last two years, 2020 and 2021, were mainly about Corona. Nevertheless, when we wish to include it as a predictor we need to specify exactly how. The obvious approach is to include the infection rate. This is what we see being done in [Madsen, 2021] with less than satisfactory results. We have further concerns with this number as the predictor since the infection rate estimate is highly dependent on the number of tests being executed. Even more critical of such an estimate is the strategy of the government. This changed a great deal; hence the number of executed tests each day was of magnitudes different looking at different periods throughout. Other predictors like when press conferences were held, introduced or declined restrictions could potentially contain important information. We choose to avoid using Corona altogether mainly because of the small analysis in [Madsen, 2021] not invoking hope.

Scraping maps pictures

We try to expand our feature space including maps images. Moreover, do a small analysis of doing so. The idea behind taking maps-images from above each housing into consideration is, to begin with, expanding the data for the models. In addition, we imagine it to be a very objectively 'picture' of what is around the housing. By looking at the colors in each image we can get the percentage of water, large roads, small streets, and more in the proximity of the housing.

We use *Wego.here.com* for scraping each image since this map is made with colors that contrast well and this map uses different colors for roads and green areas of different sizes. Hence it is possible to get a rather realistic look at what exactly is within near proximities of the housing.

The more well-known Google maps could have been used though a few challenges with this map. Google's security makes sure an automated address search is much too tedious for starters. This can be fixed with some luxurious extra features buying the google API associated. Another point is maps have a well-functioning recommender system. This lets one know what is around of restaurants and sightseeings which is just noise for now but could potentially be valuable and again this could most likely be fixed easily with the API. To conclude and not the least it cost money.

While scraping these images from *Wego.here.com* it is vital that too much memory is not being used. Thus every time an image is scraped it is convenient to crop it directly. The less pixels the less memory is used. In the following we see an example of a scraped image where we crop everything unwanted away. In the last figure we have an average of 300 meters around the address.

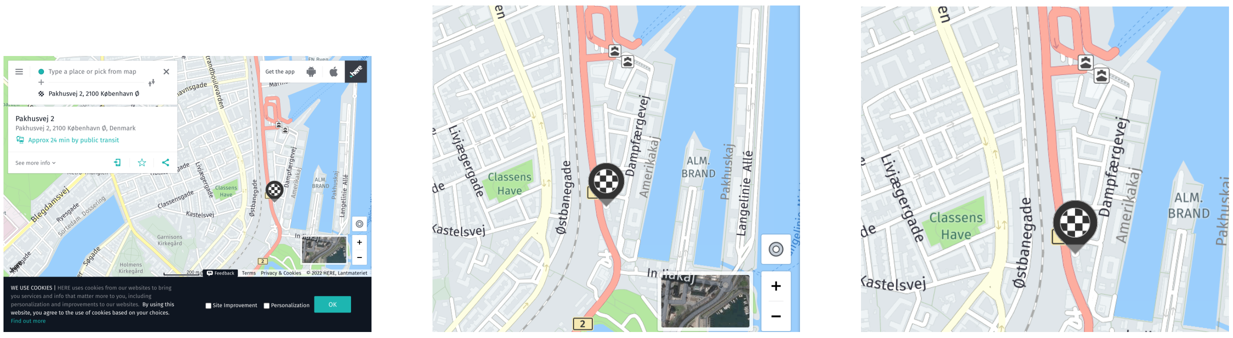
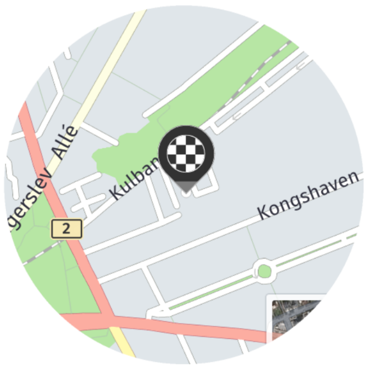


Figure 3.22: How we want to crop every picture.

We crop one more time and and make the image circular around the actual address, just to see another address in action we use a different:



We then get the predictors we wish for by collecting the red, green, and blue color pixels and using the percentage of each color to the whole image as the values of each attribute, respectively.

Further modeling of this relatively quick analysis was found dissatisfactory thus we did not bring it forth. Future work could be to optimize this scraping, maybe find a better maps page without a huge spot where each address is located and optimize the cropping. Potentially expanding to satellite images could be a fascinating analysis.

4 The importance of an asking price

4.1 Model Setup

Post our analysis we wish to bring forward an idea of what these house price estimates we compute can actually be used for in practice. [Han, Strange, 2016] proposes a setup that tries to explain the role of the asking price. This is interesting from our perspective since we can attempt to see whether using our price estimates gives a noticeable difference. The article explains the model set up and, how the asking price can direct search theoretically, the sellers choice of the asking price theoretically and does an empirically showing the theory in practice.

4.1.1 Partial Equilibrium

The article explains the game as a one-period model where we first think of it as the seller initiates and sets an asking price. Then buyers subsequently make choices of whether or not to visit the house, this is the search. The house is ultimately sold as a process of negotiation.

The setup is a partial equilibrium model:
Seller has the reservation price of x_L. Buyers draw their reservation price $x \in \{x_L, x_H\}$ where the probability of $x = x_H$ is δ and hence the probability of $x = x_L$ is $1 - \delta$

$$P(x = x_H) = \delta, \qquad P(x = x_L) = 1 - \delta.$$

After visitation whether x is x_L of x_H is revealed to the seller and the buyer. It costs, c, to visit the buyers. And a cost s for the sellers when a visit is being held. All participants are assumed risk neutral. After the search the sale price is determined.

If no asking price is assumed. Seller and $n \in \{1, \ldots, N\}$ buyers will negotiate the price. $n = 1$ is trivial. For $n \geq 2$ and all $x_n = x_H$, the price will give a surplus to the seller and the sale price will become x_H.
Suppose $n \geq 2$ and all $x_n = x_L$ then no surplus to split and the sale price will become x_L. If $n \geq 2$ and $x_n = x_H$ and $x_{-n} = x_L$[1] then the seller negotiates with the n buyers and the price depends on the relative bargaining power of the participants.

Let $\theta \in [0, 1]$ be the sellers bargaining power then the price after negotiation will be

$$p = \theta x_H + (1 - \theta) x_L \tag{4.1}$$

Thus the asking price is not considered. For the asking price to be of importance [Han, Strange, 2016] claims the asking price needs to be low enough to initiate any search. Defining a_θ to be the critical value of the asking price and sets it as the bargaining price:

$$a_\theta = \theta x_H + (1 - \theta) x_L. \tag{4.2}$$

[1]Here the $-n$ notation is to express the rest, all but buyer n.

Furthermore the asking price must entail some sort of commitment which will be explained later. A commitment that holds unless multiple buyers are willing to pay prices exceeding the asking price.

This is the overall game [Han, Strange, 2016] proposes and the timing of decisions and events is as follows:

1. The seller sets the asking price considering the visitation cost, s.
2. Buyers choose to visit knowing the asking price, a, their visitation cost, c, and knowing the distribution of x.
3. $x \in \{x_L, x_H\}$ values are revealed to all participants, buyers and sellers.
4. Price is determined by bargaining or acceptance of the asking price.

4.1.2 Asking price can direct search

When a price ceiling binds asking price can direct search despite being neither a posted, floor or ceiling price. For $n = 1$ only one visitor and $a \in \{x_L, x_H\}$ if $x = x_L$ i.e. a low type buyer the sale price of the house will also be x_L. If $x = x_H$ indicating a high type buyer and remembering we only have one visitor which would accept the asking price if and only if it is below the critically level:

$$a \leq a_\theta = \theta x_H + (1 - \theta) x_L. \tag{4.3}$$

The asking price is valuable to buyers as a commitment only if their bargaining power does not already enable them to command a lower price.

For $n \geq 2$ we need to look at three cases:

The Traditional case: All $x \leq a$ making all buyers of low type since $a \in (x_L, x_H)$ for all buyers. Giving the sale price of $p = x_L$ which is the valuation of the buyer. Hence seller's profit and the utility of the buyer are zero. This corresponds to a high asking price which will in turn get negotiated down. Defining τ to be the probability of this traditional case with the outcome of the sale price being strictly below asking price, where

$$\tau = (1 - \delta)^n \tag{4.4}$$

Second we have an *Acceptance* case. Let $n \geq 2$ and only one buyer draws a match value above the asking price. Commitment binds when the asking price is not too high as (4.3). If this is upheld the utility of the high type buyer is $x_H - a$ and the seller's profit is $a - x_L$. The probability of the acceptance case is the probability of exactly one buyer being willing to pay more than the asking price while all other visitors draw a low type value given as

$$\alpha = n\delta(1 - \delta)^{n-1}. \tag{4.5}$$

Third is a *Bidding War* case. Again we have $n \geq 2$ who draws high or low match values but this requires at least two buyers are drawing of high value. In this case as in the traditional case the buyers get no surplus i.e., buyer utility of 0, but the seller gets the profit of $x_H - x_L$. Furthermore, the probability can be expressed as one minus the other two probabilities:

$$\beta = 1 - \tau - \alpha = 1 - (1 - \delta)^n - n\delta(1 - \delta)^{n-1}. \tag{4.6}$$

As long as there are some visitors and the asking price is between x_L and x_H, this model allow for all three possibilities.

Now suppose $a \geq x_H$. While $a_\theta < x_H$ any asking price at or above x_H will give the same output as an asking price equal to a_θ. Hence an asking price this high is of no gain. Now suppose $a \leq x_L$ if two or more visitors are of high type, $x = x_H$ the sale price will still be $p = x_H$. If all draws low type then the sale price will be $p = x_L$. The difference is if exactly one visitor draws a high type. With $a \leq x_L$ there are now both low and high type buyers who at least will be weakly willing to accept the asking price. This makes it likely that the house will go to the high bidder with a sale price of $p = x_L$.

The expected sale price of the house is given by

$$\mathbb{E}[p] = \alpha a + \beta x_H + \tau x_L,$$

where we see how the asking price is directly influencing the expected sale price by the first term in the equation. We will see how the asking price can also indirectly impact the expected sale price by directing buyer search.

Buyer visitation happens when the utility of a visit/search is above the cost, taking as given the search choices of the other buyers. The equilibrium number of visits must give visitor n expected utility greater than or equal to the search cost. And for visitor $n + 1$ utility is less than search cost. We assume buyers are aware of other buyer's interests.

As noted earlier buyer bargaining power allows them to retain some of the value created by search. The greater the buyers' bargaining power the greater the incentive for a search and the asking price must be additional lower for additional search incentives.

Suppose that asking price is irrelevant, with $a \geq a_\theta$. The first buyer who searches a given house has the positive ex post utility only when they are of a high type. This happens with probability δ. This indicates that expected utility of a visit v is

$$v = \delta[x_H - \theta x_H - (1 - \theta)x_L] = \delta(1 - \theta)[x_H - x_L]. \tag{4.7}$$

This is exactly the probability that a buyer is high type times the negotiated share of the surplus. If the expected utility when only one buyer visits is less than search cost, c, then there will be no visits and the house will not be sold. Setting the expected utility to be at minimum the search cost, c, makes sure there will be visitors:

$$v = \delta(1 - \theta)[x_H - x_L] \geq c. \tag{4.8}$$

We can then compute the minimum level of bargaining power such at least one buyer will visit:

$$\delta(1 - \theta)[x_H - x_L] \geq c \tag{4.9}$$

$$\Leftrightarrow \delta(1 - \theta) \geq \frac{c}{[x_H - x_L]} \tag{4.10}$$

$$\Leftrightarrow -\theta \geq \frac{c}{\delta[x_H - x_L]} \tag{4.11}$$

$$\Leftrightarrow -\frac{c}{\delta[x_H - x_L]} \leq \theta \tag{4.12}$$

This tells us that seller is forced to put out an asking price to encourage search since otherwise no buyers are willing given their weak bargaining power.

To really understand the role of the asking price in this set up for a start we consider the first buyer's search. The expected utility of the first buyer who visits is

$$\upsilon_1 = \delta[x_H - a] + (1-\delta)\cdot 0 \tag{4.13}$$

We can then find the maximum level of asking price that encourage buyers to visit by equating above (4.13) with the search cost

$$\delta[x_H - a_1] = c \Leftrightarrow a_1 = x_H - c/\delta. \tag{4.14}$$

For $n \geq 2$ visitors there is no expected utility in both the traditional and the bidding war case. Indicating that the probability of getting a positive expected utility is the probability of the acceptance case, which gives an expected utility with n visitors of

$$\upsilon_n = (1-\delta)^{n-1}\delta[x_H - a] \tag{4.15}$$

Hence for getting n visitors the asking price must be at maximum:

$$(1-\delta)^{n-1}\delta[x_H - a] = c \Leftrightarrow a_n = x_H - c/[\delta(1-\delta)^{n-1}]. \tag{4.16}$$

Which we call the inverse demand schedule that sellers face as the sequence of asking prices $\mathbf{A} = \{a_n | n = 1, 2, ..., N\}$. The key aspect of the demand schedule is that it captures how an asking price, that is not a posted, ceiling or floor price, can direct a search process. The first and most important implication is that a lower price is required to encourage more visits. Isolating in (4.16) for n gives:

$$\begin{aligned}
c &= (x_H - a_n)(\delta(1-\delta)^{n-1}) \\
\Leftrightarrow (1-\delta)^{n-1} &= \frac{c}{\delta[x_H - a_n]} \\
\Leftrightarrow (n-1)\ln(1-\delta) &= \ln\left(\frac{c}{\delta[x_H - a_n]}\right) \\
\Leftrightarrow n\ln(1-\delta) &= \ln\left(\frac{c}{\delta[x_H - a_n]}\right) + \ln(1-\delta) \\
\Leftrightarrow n &= \frac{\ln\left(\frac{c}{\delta[x_H - a_n]}\right)}{\ln(1-\delta)} + \ln(1-\delta).
\end{aligned} \tag{4.17}$$

And by the first order condition with respect to a:

$$\frac{\partial n}{\partial a} = \frac{1}{\ln(1-\delta)}\frac{1}{x_H - a} < 0 \tag{4.18}$$

A second noteworthy property of the demand schedule is that the amount of search that can be provoked by reducing the asking price is bounded. Formally the maximum number of visits that can be encouraged, N, is defined by:

$$[x_H - x_L] = \frac{c}{\delta(1-\delta)^{N-1}} \tag{4.19}$$

We also note. First the elements of the demand schedule a_n decrease in c. This is immediate from differentiating (4.16). Higher search cost requires even lower asking prices to encourage a given number of visits. Second the elements a_n increase in x_H. When a good match is worth more, then an associated greater search for any level of asking price is established. Since both of these variables are associated with greater demand for any given house they will be useful below in considering how asking price operates in boom and bust markets. Third the maximum possible number of visits, N, is decreasing in buyer search cost, c, and increasing in good match quality x_H. The effect of δ is ambiguous.

Further investigation of (4.16) reveal the following:

$$\frac{\partial^2 n}{\partial a \partial x_H} = -\frac{1}{\ln(1-\delta)} \frac{1}{(x_H - a)^2} > 0 \tag{4.20}$$

and

$$\frac{\partial^2 n}{\partial a \partial \delta} = -\frac{1}{1-\delta} \frac{1}{\ln(1-\delta)^2} \frac{1}{x_H - a} > 0 \tag{4.21}$$

Equation (4.20) is indicating the greater the value of a good match the weaker is the negative effect of the asking price on the number of visitors. And equation (4.21) indicates the larger fraction of high type visitors the weaker the negative effect on the number of bidders. This is effective in an analysis of asking price on search in booms and busts.

In a boom asking price has a smaller role in directing search. The intuition is that in a boom the surplus that buyer anticipates from visiting is large and this reduces the necessity of using the asking price to attract visitors. Thus in busts the search direction is stronger.

These results as stated by [Han, Strange, 2016] clarifies the importance of the asking price and how it varies in the micro-founded model. And the results importance in the marketing aspect of selling houses, i.e. Home sellers are often advised to set "competitative" asking prices.

4.1.3 The house seller's choice of asking price.

In the article [Han, Strange, 2016] states the seller sets the asking price to maximize expected surplus taking the relationship between n and a as given. Choosing an asking price that leads to $n = 1$ the expected profit is

$$\pi_1 = \delta[a - x_L] - s \tag{4.22}$$

When only one visitor is expected the asking price will be set to a_1 and we can substitute from equation (4.14)

$$\begin{aligned} \pi_1 =& \delta[a_1 - x_L] - s \\ =& \delta[(x_H - c/\delta) - x_L] - s \end{aligned} \tag{4.23}$$

Setting the profit to be at least zero yields this constraint:

$$x_H - x_L \geq \frac{c+s}{\delta} \tag{4.24}$$

Hence the seller need at least one visitor to be profitable. For an asking price that leads to $n \geq 2$ the expected profit is:

$$\begin{aligned}
\pi_n &= n(1-\delta)^{n-1}\delta[a_n - x_L] + (1-(1-\delta)^n - n(1-\delta)^{n-1}\delta)[x_H - x_L] - ns \\
&= n(1-\delta)^{n-1}\delta\left[\left(x_H - \frac{c}{\delta(1-\delta)^{n-1}}\right) - x_L\right] + (1-(1-\delta)^n - n(1-\delta)^{n-1}\delta)[x_H - x_L] - ns \\
&= n(1-\delta)^{n-1}\delta[x_H - x_L] - nc + [x_H - x_L] - (1-\delta)^n[x_H - x_L] - n(1-\delta)^{n-1}\delta[x_H - x_L] - ns \\
&= (1-(1-\delta)^n)[x_H - x_L] - n(c+s) \qquad (4.25)
\end{aligned}$$

The seller will choose from the demand schedule $\mathbf{A}$. Choosing any other asking price gives the same number of visits as an element of A but has lower expected profits.

Let the sequence of expected profits be given by

$$\Pi = \{\mathbb{E}\pi_n | a_n \in \mathbf{A}\}. \qquad (4.26)$$

The seller decides to pick an element of the asking price sequence $\mathbf{A}$ to obtain the maximum of Π. It is known Π has a maximum given it a finite set. In characterizing this optimal asking price the difference between profit at n and profit at $n-1$ is a key factor:

$$\begin{aligned}
\Delta\pi_n &= \pi_n - \pi_{n-1} \\
&= (1-(1-\delta)^n)[x_H - x_L] - n(c+s) - \big((1-(1-\delta)^{n-1})[x_H - x_L] - (n-1)(c+s)\big) \\
&= (1-\delta)^n[x_H - x_L] + (1-\delta)^{n-1}[x_H - x_L] - (c+s) \\
&= (1-(1-\delta))\,(1-\delta)^{n-1}[x_H - x_L] - (c+s) \\
&= \delta(1-\delta)^{n-1}[x_H - x_L] - (c+s) \qquad (4.27)
\end{aligned}$$

It is known $\pi_1 > 0$ and calculations from above implies $\Delta\pi_n > 0$. We note $\partial\Delta\pi_n/\partial n < 0$. And at the maximum amount of visitors, N, we have no extra profit.

This means for $s > 0$ the maximizing asking price is associate with a number of visits $n \leq N$. The maximizing asking price a_n^* will satisfy $\Delta\pi_n > 0$ and $\Delta\pi_{n+1} > 0$.

Previously it was asserted a seller is not willing to set an extreme asking price at either x_L or x_H. Setting an asking price at x_H leads to zero visitors. And is thus dominated by any asking price giving a positive number of visits. Second, an asking price equal to x_L is dominated unless $x_L = a_N$. Which is a knife-edge case where $a = x_L$ fails to be dominated. We thus suppose that $a_N > x_L$.

Since the seller's optimal asking price is on the interior of (x_L, x_H) the probabilities of the traditional, bidding war and acceptance cases are all positive. Concluding that the asking price can have a commitment role, encouraging search even when the asking price is neither a posted, a floor or a strictly ceiling price.

4.2 The approach by textitHan and Strange.

In the article [Han, Strange, 2016] *Han* and *Strange* work with data primarily based on a survey of recent buyers in a large USA metropolitan area. They sent out a survey to most people they could find information on. From online sales webpages they got the addresses of sold houses in the proximity. From this they could buy access to the owners of these addresses. With a response rate of 19.2% they got their data points. *Han* and *Strange* claims this is the first dataset that provides information on home search, bargaining and bidding behavior. At that time.

As stated by *Han* and *Strange*, as in the market of Denmark, sellers initiate the search process listing the property. And the home is often sold at a later date than the date of listing. Sometimes 'bully' offers are made earlier. There may be made multiple rounds of offers. Bidders do not meet with each other and they also do not meet with the seller directly[2]. Potential buyers will consult with their agents prior to bidding, discussing among other things the competition that they are likely to face from other bidders. In their environment bidders usually register their intent to bid prior to the quasi-auction. Hence bidders will know the number of competing competitors. In some cases only one bidder comes forth and the sales price is negotiated. In others multiple bidders. In both cases asking price directs search activity. *Han* and *Strange* states that the key assumptions of this sort of institutional practice are common across markets. A keynote they are making is that though the institutions are similar across markets, different markets have experienced booms and busts very differently. This is very important to note for our further analysis of the subject.

In their survey two main questions were stated "Were there other people actively bidding on the home when you submitted your first offer?" and "If yes, about how many bidders were there?". And these are the important question in this analysis since *Han* and *Strange* wish to do an empirical investigation of the role of asking price directing search.

Han and *Strange* notes three findings from the survey. First, a notable fraction of the below-, at-, and above-list sales. Second, their data shows explicit evidence for the presence of multiple bidders who are interested in the same houses. This justifies the bidding war possibility that was modeled. Third the patterns of bidding behavior are related to the relationship between sales and asking price.

4.2.1 Han and Strange Empirical results

Han and *Strange* presents a log-linear specification that regresses the number of bidders on the asking price. The folling equation is estimated:

$$\ln(N_{ijt}) = \alpha + \beta \ln(P_{ijt}) + \eta_t + \tau_j + \varepsilon_{ijt}, \tag{4.28}$$

where N_{ijt} is the number of bidders for house i in district j at period t, P_{ijt} is the corresponding asking price, X_{ijt} is the corresponding house attributes, η_t is the month and year fixed effect, τ_j is the district fixed effects, and lastly the ε_{ijt} is the noise term.

They state the model parameters in table 4 in appendix B (copy from the article). Moreover, with the introduction of housing district dummies only the coefficient on the asking price parameter is negative. $\beta_{\log(askingprice)} = -0.14$. Adding transaction period dummies changes the coefficient to $\beta_{\log(askingprice)} = -0.12$. Further adding housing attributes makes the coefficient further negative, $\beta_{\log(askingprice)} = -0.22$. This, as *Han* and *Strange* explain, indicates when lowering the asking price by 10% the number of bidders increases by 2.2%. These results are

[2]This a bit of an exaggeration since some people might sell their own houses and thus do meet the bidders.

consistent with the model's key predictions about the role of the asking price directing search. This estimate should be treated as a lower bound for the true directing effect of the asking price. This is, firstly, because of how the actual bidders are used as a proxy for the number of visitors. In the stylized model *Han* and *Strange* use all visitors bid, but in reality it is believed the large majority of visitors do not actually bid. Since the number of bidders provided a lower bound for the number of visitors a downward bias in the estimated responsiveness of the buyers to the asking price is introduced and indicated the actual directing role of the asking price is even stronger. Secondly, the number of bidders may be misreported by the homeowners hence the coefficient on the number of bidders will also be biased downward in magnitude. Third the data probably does not register for all the housing characteristics observed by homebuyers again introducing a downward bias.

Another key prediction of their model is how the strength of the directing role of asking price varies depending on the state of the housing market and the degree of atypically houses[3]. In busts when fewer high-type buyers and when the existing high-type buyers value houses less the directing effect of the asking price on buyers is stronger. On the contrary, when in a boom economy, many high type buyers valuate higher, giving a further decrease in asking price is need to direct more search.

4.3 What we see in our Market

The key implications of the theory we wish to focus upon is how the asking price can direct search depending on the market economy. Thus we wish to establish how the market economy has been in the six year period we have scraped from.

We make three graphs which are supposed to clarify the conjunctions of the market in Copenhagen in the time period.

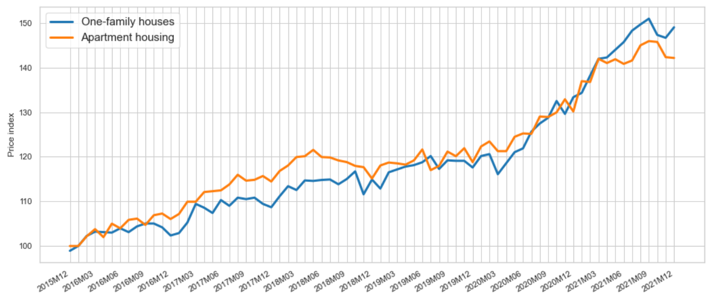


Figure 4.1: Price index of one-family houses and apartment housing from 2016 through 2021 in Copenhagen.

[3]We will not look at atypically houses as a problem in our set up.

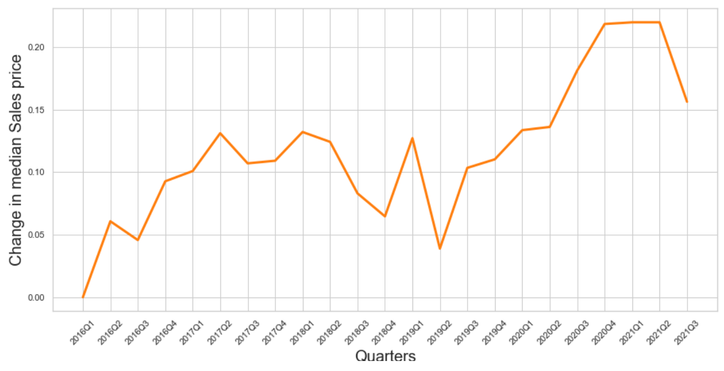


Figure 4.2: Changes in median sales prices from 2016 through 2021 in Copenhagen.

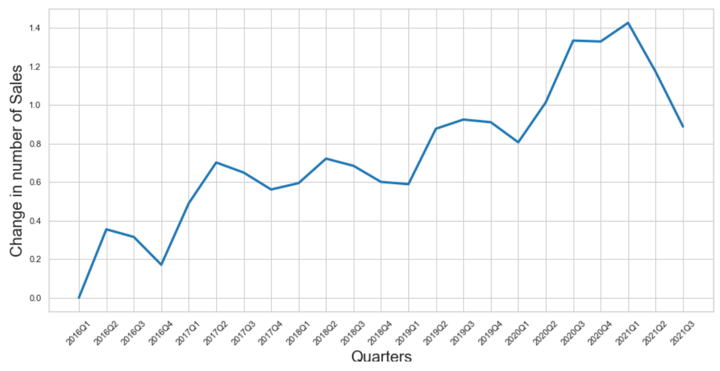


Figure 4.3: Change in number of housing sales from 2016 through 2021 in Copenhagen.

The graphs all show a clear tendency of the market economy being in a boom. Housing is getting more expensive and more sales are being issued.

4.4 Some empirical attempts

We wish to utilize the theory and [Han, Strange, 2016]'s empirical analysis in application to the Copenhagen market in our time period. For any crossover application we will have to make some fundamental assumptions.

To begin with, an essential part of [Han, Strange, 2016] findings are due to the survey they made to get information of actual bidders. I.e., an estimate of the number of bidders of each sale as a lower bound for search. This survey is not easily replicated and what we instead do is we assume their market and our marked are comparable. A discussion of the assumptions is found later. If the markets are similar, we can use the results they have. We make a similar table as 4.1.

Year	Sale/List ratio	Below Asking	At Asking	Above Asking	Mean Asking Price	# Sales
2016	0.9823	0.6016	0.2520	0.1466	3570332	635
2017	0.9892	0.5690	0.2623	0.1688	3870031	877
2018	0.9856	0.6674	0.2014	0.1312	3799407	884
2019	0.9792	0.7515	0.1458	0.1027	3883477	1022
2020	0.9968	0.6360	0.2055	0.1585	3908379	1445
2021	1.0260	0.4924	0.2140	0.2936	3908774	1182

Table 4.1: Sales prices compared to the asking price from 2016 through 2021. And number of sales for each year.

In the article the mean and standard deviation of the number of bidders for the sale price below, at and above the asking price. With this we can estimate the number of bidders we would have had by simulation. [Han, Strange, 2016] states the number of bidders is overdispersed relative to the Poisson distribution. Looking at 1 in appendix B. We notice it follows somewhat a negative gamma distribution. And thus we can use the means and standard deviations they have used.

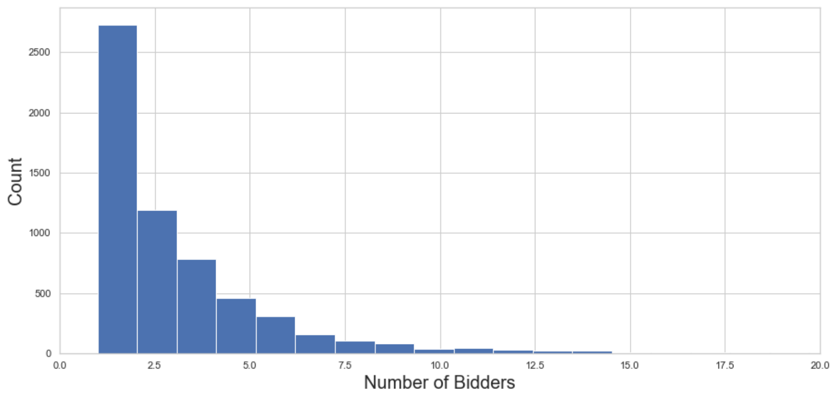


Figure 4.4: Simulated number of bidders using the negative gamma distribution.

Looking at the graph we notice a somewhat high overdispersion still but we are satisfied and will continue with these simulations. We wish to estimate the multivariate linear regression as in (4.28), for convenience we write the equation again:

$$\ln(N_{ijt}) = \alpha + \beta \ln(P_{ijt}) + \eta_t + \tau_j + \varepsilon_{ijt}. \tag{4.29}$$

We have estimates for the label values, and number of bidders. We have features indicating the district or in our setting postal code, and a feature for the period on the market. In the model [Han, Strange, 2016] includes a specific set of house attributes. We have most of what they use and a lot more. We also have a feature for seasonal effects. We use the following features for our regression model:

`paymentCash_b, numberOfBaths_bd, floor_b, alfs_area, areaBasement, numberOfRooms, numberOfToilets_bd, rebuildYear_b, salesPeriod, outerwall_d, quarter_b, usage_d, roof_d, postalId_b`

With these features, the label feature and the number of bidders. We get the parameter estimates as in the article. We get the parameter value

$$\beta = -0.0034368, \tag{4.30}$$

which in the context corresponds to lowering the asking price by 10% indicates an increase in the number of bidders by 0.0034%. This is very consistent with the theory. We know we are in a boom economy hence the role of the asking price is rather small and nonetheless lowering the asking price should increase search. We do the same regression with our predicted estimates and the AVM price.

Asking price predictor	β-value	T-statistics
Asking Price:	-0.0034	(1.8109)
Ensemble Stacking predictions:	-0.0048	(1.6945)
AVM Price:	-0.0044	(0.9679)

Table 4.2: Regresison model using asking price, ensemble stacking predictions and AVM prices as the role of asking price in the denoted regression model. With corresponding T-statistics.

We see no significant difference between the three values. This makes a great deal of sense considering the theory and how close these estimates are to each other. We conclude by mentioning the T-statistics, which by these low values is an indication of low reliability of the predictive power of this parameter. These values can be explained by the theory of a boom economy making the role of the asking price near neglectable.

The market [Han, Strange, 2016] are looking at is more tumultuous than ours. Looking at figure 5 in Appendix B, we see the impact of the financial crisis in 2008. Also noticing in the early years the economy is somewhat steady. Considering their research is done in USA where we believe the market is different than ours. Though we do not expect the house selling institution to be much different than of Copenhagen or Denmark there might have come some revolutionary laws post the financial crisis which make the settings very different. When looking at table 4.1 we get confirmation that the market of [Han, Strange, 2016] and our market of Copenhagen are different. The below list, at list and above list ratios in Table 2 from appendix B (their article) are different, potentially explained by the above. The numbers of table 4.1 do make somewhat sense when taking the theory of the boom economy, making asking price less important, into consideration. Setting an asking price differently will not attract neglectable more search. Therefore the asking price might as well be the valuation of the housing. In addition in the article it seems to be only houses and not apartments being taken into consideration. This is important to note especially when we consider the attributes of housing. In the article, they use a very specific set of housing attributes we do not know if they use those because of interior analysis and those attributes being key markers. They mention the regression is biased downwards due to bidders/visitors being able to see a greater amount of characteristics of the housing than used in the model. Thus we assume they are not considering more attributes due to convenience. And a question arises whether we should have used all attributes we have including exterior attributes. We attempted to follow the approach of the article. Thus we do not include our complete list of features/attributes.

Taking these thoughts into consideration, it is rather clear that simulations of the number of bidders across for sale prices below, at and above asking prices are a bit of a stretch. Further analysis of key market factors and a similar survey for bidders is needed to clarify this.

A side note we can include is how the market in 2022 seems to be about to shift into a none-boom market due to the interest rate increasing making it more expensive to take out loans. If the market actually decreases its boom tendencies it would be of greater interest to apply this theory. We would suspect the asking price actually being able to increase search.

To sum up, the results of using [Han, Strange, 2016]'s approach on the Copenhagen market need to be taken relatively lightly. We have to be very wary of the assumptions made. And with a further analysis of key factors in the Danish market needs to be made. Looking at this theory might become of much interest in the near future.

5 Conclusion

Throughout this thesis we had two quests. The first one being to estimate the best possible prediction rule $\hat{h}^*$. And the second being to answer the question of what would happen if we changed the asking price to actual estimates of sale price?.

Trying to get the best prediction rule for estimating house prices in Copenhagen from 2016 through 2021. In doing so we scraped excessively from mainly four web pages. Getting addresses, label values and sale information of the sales from Boligsiden.dk , then continued to expand the feature space with Dingeo.dk, Hvorlangterder.dk and Statbank.dk contributing with different attributes of the housing, distance to important places and market features, respectively. After the scraping was done we gathered all data into one file, *raw-Data_final.csv*, in which we started the cleaning approach. We looked more specifically into the samples and found among regular data points missing, impossible and weird data. Starting with a total of 57011 sales and 219 features we quickly got rid of the first 17047 samples just by deleting duplicates and only taking free trade sales into consideration. Before looking too much into the data we took 20% of the data out as a holdout validation set. The remaining 80% was used as training a training set. We did a stratified split by each city part of Copenhagen since we saw how sale prices were different in each part. A further look into the samples were made to get a more versatile perspective.

Then we made custom transformers that could handle the data and output a sample set that was modeling ready. Most importantly, we removed all samples with missing values and scaled the data. We combined all custom transformers into four pipelines. The first one handling the numeric features, the second to handling categorical features, third to combine these two, and the fourth to output the wished samples, this one dropping missing values making one-hot-features, ordinal-features and collecting rare categories within these features. Taking advantage of these pipelines we could run a lot of different linear regression models by quickly changing the data. Finding and estimate of the best possible sample set to estimate with was done by comparing root mean square loss for training and cross-validation sets. The final pipeline included removal of outliers by square meter price, feature engineering, removal of neglectable features, removal of high correlation features, removal of features with close to zero mutual information scores, collecting rare categories into a "rare-category" and then fixing the data by scaling, removing none complete samples, making one-hot and ordinal encoding. Optimizing a sample set is a never-ending process and we stopped here because of satisfying results. Due to how excessively we tested the samples we had a good idea of how well the first model, our benchmark model, performed.

Looking at the loss measures for our standard linear regression model we found it made surprisingly well predictions. It was able to make an estimation that deviates on average 12% from the actual sale price on both the training and holdout set. We saw how it was not able to catch the non-linearity in the sample. Seeing no signs of over-/under-fitting we still tried to Lasso-regularize it and found it having no effect of relevance. We tested whether transforming the feature space to a polynomial of second degree could describe the samples better and found us in a position where we overfitted the training set. The Lasso-regularization technique came in clutch and got this Lasso-polynomial model to predict and generalize about 1.5% point better than the benchmark model. We then tried random forest regression due to the idea of the features' discrete decision-making, making it perform well on the data. After correcting the hyperparameters in the Random Forest model such it did not greatly overfit we were able to get a prediction and generalization a bit better than the polynomial model, doing close to 2% point better than the benchmark model. Moving on to the Neural Network model by finding an estimate of the optimal hyperparameters got our estimation below the golden mark a mean relative deviation less than 10% on average from the sale price. This improved the prediction and generalization by about 3% point from the benchmark model. Other than testing the samples on different models we also wished to use them in a meta model applying ridge regression using each model estimates as input. This Ensemble Stacking model was able to outperform all tested models being able to make generalizations which on average have relative deviation about 3.5% point better than the benchmark model.

We then made some comments on the Asking price and the AVM-price. The Asking price being very close to the actual sale price and the AVM-price being a very good estimate of the sale price for the later years and bad estimates for earlier years. Including asking price would definitely increase model performance across the board but because we wished to make a model of only housing characteristics and attributes we did not include it. AVM-price is ambiguous and from literature results we did not include it. We made the same conclusion of corona as a predictor in the model. We then explained a possible procedure to get more features by scraping maps images and using color percentages as attributes. These features were found rather useless and were not included either.

Attempting to find the best possible estimates we wished to utilize them. Thus getting a theoretical background for the role of asking price, from [Han, Strange, 2016], invoked an interest in replacing actual asking prices with our estimates. In the article a relatively comprehensive survey was made to get the number of bidders for each housing sale. By using means and standard deviations included in the article we simulated the number of bidders corresponding to our sales. The two key implication of the article was the search for housing increased with the decrease in asking price and the role of asking price was very dependent on the market economy.

We found no real difference in using either our estimates or AVM-price instead of the asking price as the predictor for the number of bidders. There could be many reasons for this. But trying to explain it from the theoretical perspective it makes sense. We explained how the sales we are working with in the period 2016 through 2021 are in a boom economy, lots of sales and housing becoming more and more expensive. Hence, as mentioned, the role of the asking price is very small, by the theory.

To do this analysis we made many assumptions about somewhat identical markets in the article in their time period and in Copenhagen 2016 through 2021. And we explained how a further analysis is needed for the parameters to have real value. We also made assumptions on which attributes was important in the house and what not. We then claimed that further analysis of the role of the asking price will potentially be more attractive in the near future. The interest rate is increasing and loans becoming more expensive could decrease the boom effect in the housing market of Copenhagen.

Bibliography

[Awad, Khanna] Mariette Awad, Rahul Khanna *Efficient Learning Machines Theories, Concepts, and Applications for Engineers and System Designers*, Apress Open.

[Breiman, 1996] Leo Breiman (1996), *Stacked Regressions*, Kluwer Academic Publishers, Boston.

[Brownlee, 2020] Jason Brownlee (2020), *How to Perform Feature Selection for Regression Data*, , Data Preparation.

[Celi et al., 2019] Celi et al. (2019) *Leveraging Data Science for Global Health*, Springer.

[Chew, 2022] Beng Chew (2022), *How To Write Clean And Scalable Code With Custom Transformers & Sklearn Pipelines*, , Medium.

[Das, Cakmak, 2018] Sibanjan Das, Umit Mert Cakmak (2018), *Hands-On Automated Machine Learning, A beginner's guide to building automated machine learning systems using AutoML and Python*, Packt, Birmingham - Mumbai.

[Dreyfus, 2005] Gérard Dreyfus (2005) *Neural Networks Methodology and Applications*, Springer, Second Edition.

[Driscoll, 2018] Michael Driscoll (2018), *Python 101*, Leanpub.

[Geomatic, 2021] Geomatic (2021). Avm. https://geomatic.dk/dk/services/data-lake/ejendomsvaerdimodel-avm/.

[Géron, 2019] Aurélien Géron (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, Second Edition.

[Han, Strange, 2016] Lu Han, William C. Strange, (2016), *What is the role of the asking price for a house?*, Elsevier.

[Hastie et al., 2008] Haste et al. (2008), *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, Springer, Second Edition.

[Igel, 2019] Christian Igel (2019), *A Short Course in Data Mining Lecture Notes*, Department of Mathematical Science, University of Copenhagen.

[Kleppen, 2019] Eric Kleppen (2019), *Using FunctionTransformer and Pipeline in SkLearn to Predict Chardonnay Ratings*, , Towards Data Science.

[Lim, 2022] Yenwee Lim (2022), *Stacked Ensembles — Improving Model Performance on a Higher Level*, , Towards Data Science.

[Louppe, 2014] Gilles Louppe (2014) *UNDERSTANDING RANDOM FORESTS from theory to practice*, Department of Electrical Engineering & Computer Science, University of Liège.

[Lützen, 2019] Jesper Lützen (2019) *Diskrete Matematiske Metoder*, Second Edition, Department of Mathematical Science, University of Copenhagen.

[Madsen, 2021] Frederik Madsen (2021) *House pricing with machine-learning modelling the Copenhagen housing market 2016 - 2020.*

[Malato, 2020] Gianluca Malato (2020), *Feature selection via grid search in supervised models*, , Medium.

[Muralidhar, 2021] KSV Muralidhar (2021), *Creating Custom Transformers with Scikit-Learn*, , Towards Data Science.

[Mutual Information] *Mutual information*, https://en.wikipedia.org/wiki/Mutual_information, Wikipedia.

[Nielsen, 2019] Michael Nielsen (2019) *Neural Networks and Deep Learning*, http://neuralnetworksanddeeplearning.com/index.html.

[Pedregosa et at., 2011] Pedregosa et al. (2011), *Scikit-learn: Machine Learning in Python*, , JMLR.

[Yevgeny, 2019] Yevgeny Seldin (2019), *Machine Learning Lecture Notes*, Department of Mathematical Science, University of Copenhagen.

Appendix A

The Github repository with all the code:
https://github.com/ThommasSBiener/MasterThesis.

Appendix B

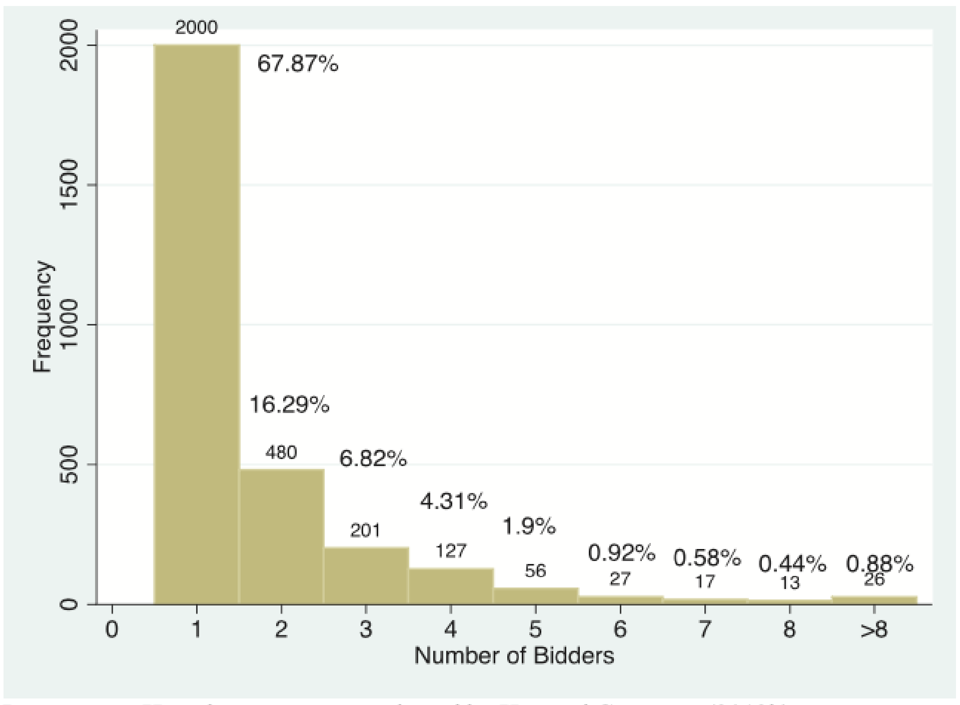
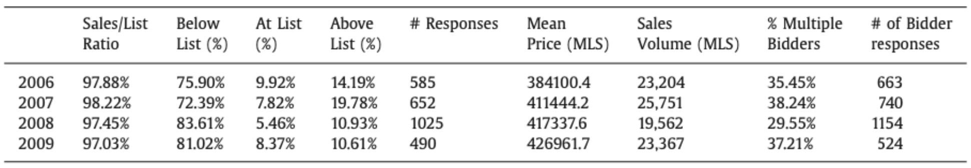
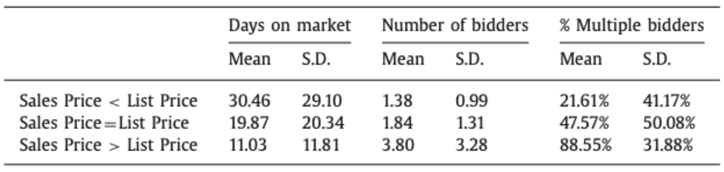


Figure 1: Used from [Han, Strange, 2016]



	Sales/List Ratio	Below List (%)	At List (%)	Above List (%)	# Responses	Mean Price (MLS)	Sales Volume (MLS)	% Multiple Bidders	# of Bidder responses
2006	97.88%	75.90%	9.92%	14.19%	585	384100.4	23,204	35.45%	663
2007	98.22%	72.39%	7.82%	19.78%	652	411444.2	25,751	38.24%	740
2008	97.45%	83.61%	5.46%	10.93%	1025	417337.6	19,562	29.55%	1154
2009	97.03%	81.02%	8.37%	10.61%	490	426961.7	23,367	37.21%	524

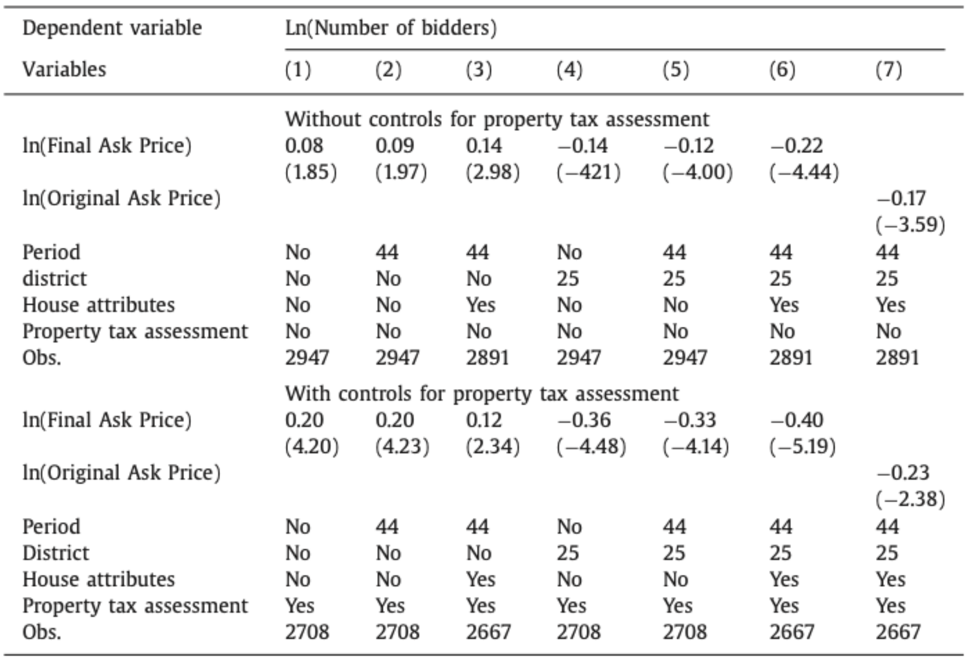
Figure 2: Used from [Han, Strange, 2016]



	Days on market		Number of bidders		% Multiple bidders	
	Mean	S.D.	Mean	S.D.	Mean	S.D.
Sales Price < List Price	30.46	29.10	1.38	0.99	21.61%	41.17%
Sales Price = List Price	19.87	20.34	1.84	1.31	47.57%	50.08%
Sales Price > List Price	11.03	11.81	3.80	3.28	88.55%	31.88%

Figure 3: Used from [Han, Strange, 2016]

Appendix B



Dependent variable	Ln(Number of bidders)						
Variables	(1)	(2)	(3)	(4)	(5)	(6)	(7)
	Without controls for property tax assessment						
ln(Final Ask Price)	0.08	0.09	0.14	−0.14	−0.12	−0.22	
	(1.85)	(1.97)	(2.98)	(−421)	(−4.00)	(−4.44)	
ln(Original Ask Price)							−0.17
							(−3.59)
Period	No	44	44	No	44	44	44
district	No	No	No	25	25	25	25
House attributes	No	No	Yes	No	No	Yes	Yes
Property tax assessment	No	No	No	No	No	No	No
Obs.	2947	2947	2891	2947	2947	2891	2891
	With controls for property tax assessment						
ln(Final Ask Price)	0.20	0.20	0.12	−0.36	−0.33	−0.40	
	(4.20)	(4.23)	(2.34)	(−4.48)	(−4.14)	(−5.19)	
ln(Original Ask Price)							−0.23
							(−2.38)
Period	No	44	44	No	44	44	44
District	No	No	No	25	25	25	25
House attributes	No	No	Yes	No	No	Yes	Yes
Property tax assessment	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Obs.	2708	2708	2667	2708	2708	2667	2667

Figure 4: Used from [Han, Strange, 2016]

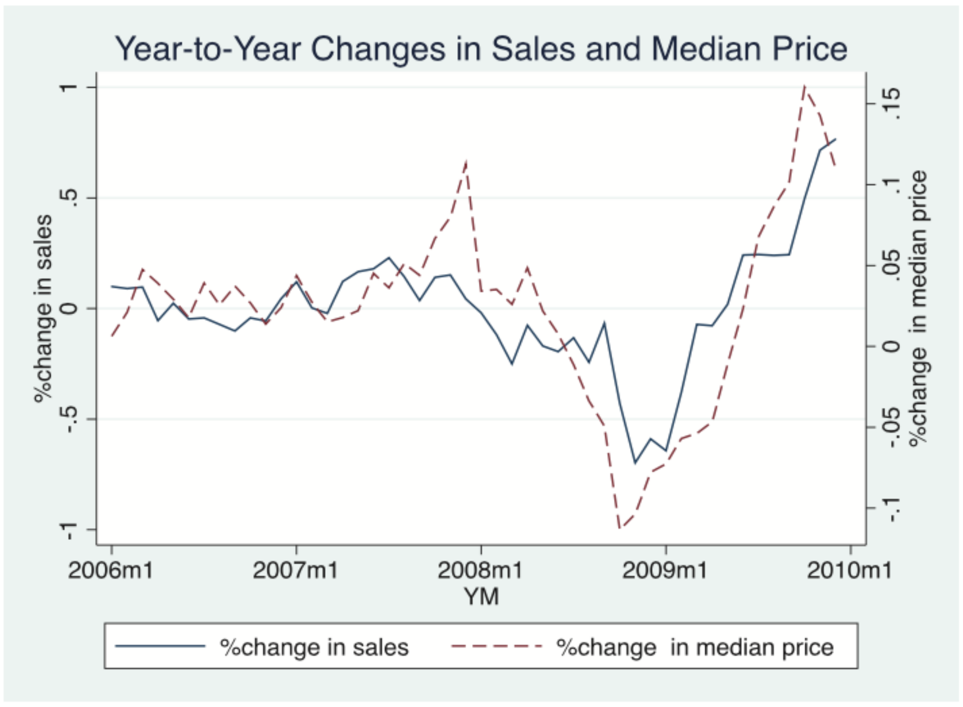


Figure 5: Used from [Han, Strange, 2016]