

CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

1. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

2. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

3. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

3.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

3.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (3.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (3.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

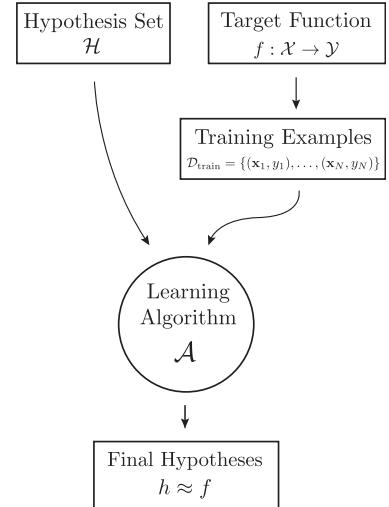


Figure 3.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵Also called expected error or the out-of-sample error.

⁶Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (3.3)$$

3.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (3.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(\mathbf{x}, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 1 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (3.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 2 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (3.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (3.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 3 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (3.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (3.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (3.10)$$

Theorem 1 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (3.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (3.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 2 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (3.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (3.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

3.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypotheses h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: [?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 3 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (3.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (3.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

3.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

3.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (3.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

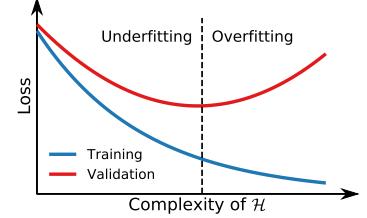


Figure 3.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (3.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned} \quad (3.19)$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (3.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (3.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \end{aligned} \quad (3.22)$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

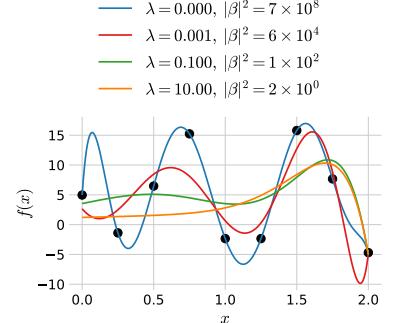


Figure 3.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (3.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by ?] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (3.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

3.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

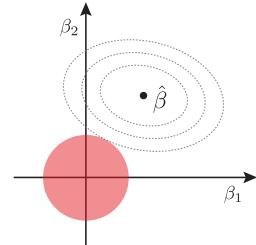


Figure 3.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constraint region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

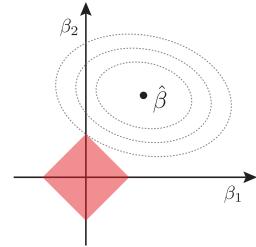


Figure 3.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constraint region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

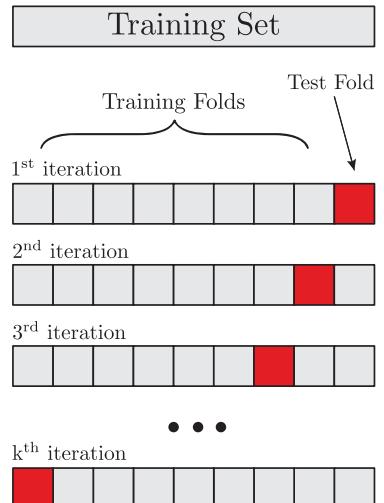


Figure 3.6: k -fold cross validation.

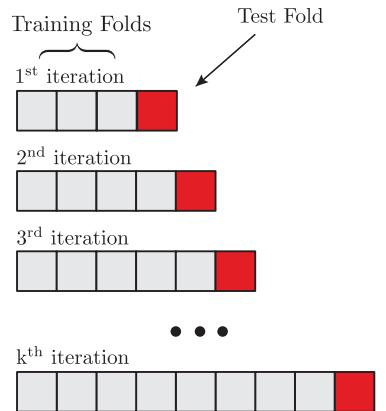


Figure 3.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

3.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

3.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (3.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (3.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (3.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

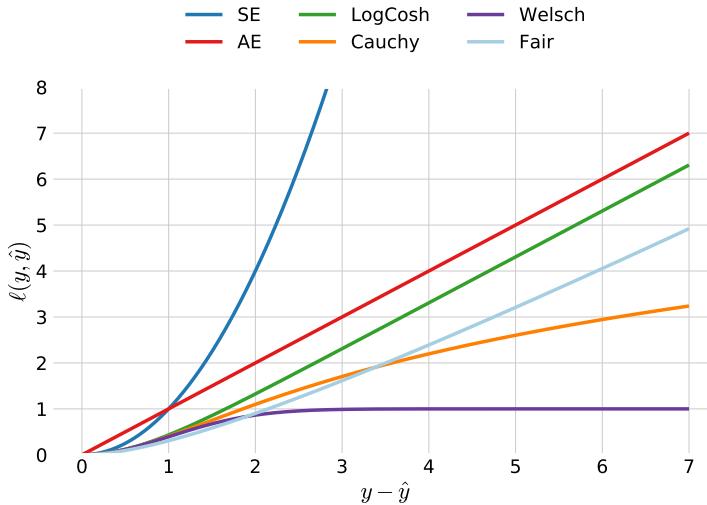


Figure 3.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

3.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (3.28)$$

3.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

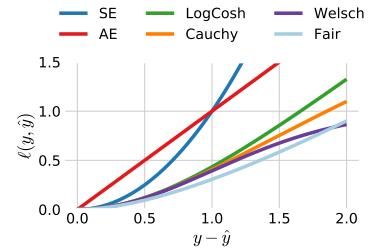


Figure 3.9: Zoom in of Figure ??.

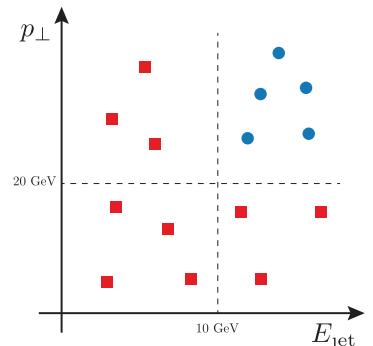
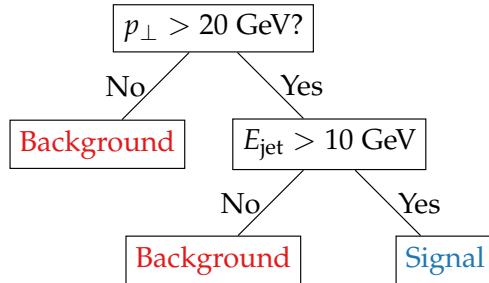


Figure 3.10: Illustration of the cuts a decision tree model make for signal in blue circles and background in red squares. This is a visualization in the feature space of the decision tree seen in Figure ??.



any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

3.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (3.29)$$

Figure 3.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 4 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (3.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (3.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (3.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (3.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (3.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (3.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

3.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

3.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (3.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

3.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (3.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

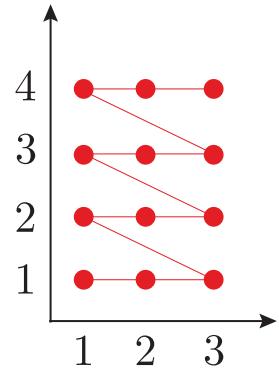


Figure 3.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (3.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

3.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

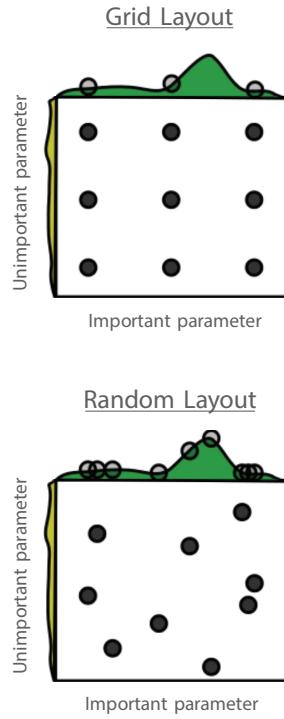


Figure 3.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the datadependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (3.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

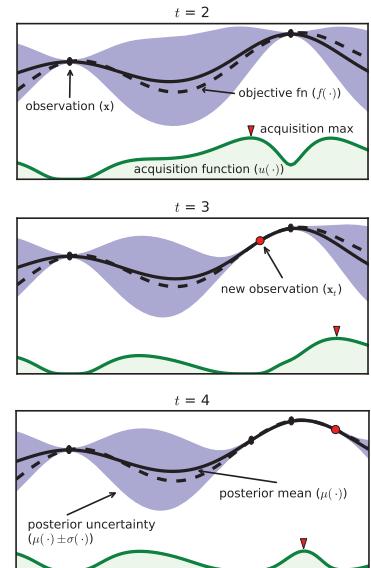


Figure 3.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

3.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (3.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 1 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (3.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 2 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (3.42)$$

Axiom 3 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (3.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (3.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure ???. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{\text{jet}}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{\text{jet}}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{\text{jet}}, p_\perp\}) - f_x(\{E_{\text{jet}}\})]. \end{aligned} \quad (3.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (3.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

⁴³ Not necessarily linearly correlated.

Axiom 4 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (3.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

4. Danish Housing Prices

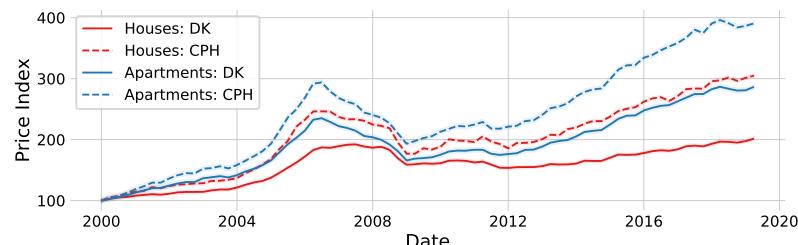
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 4.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

4.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

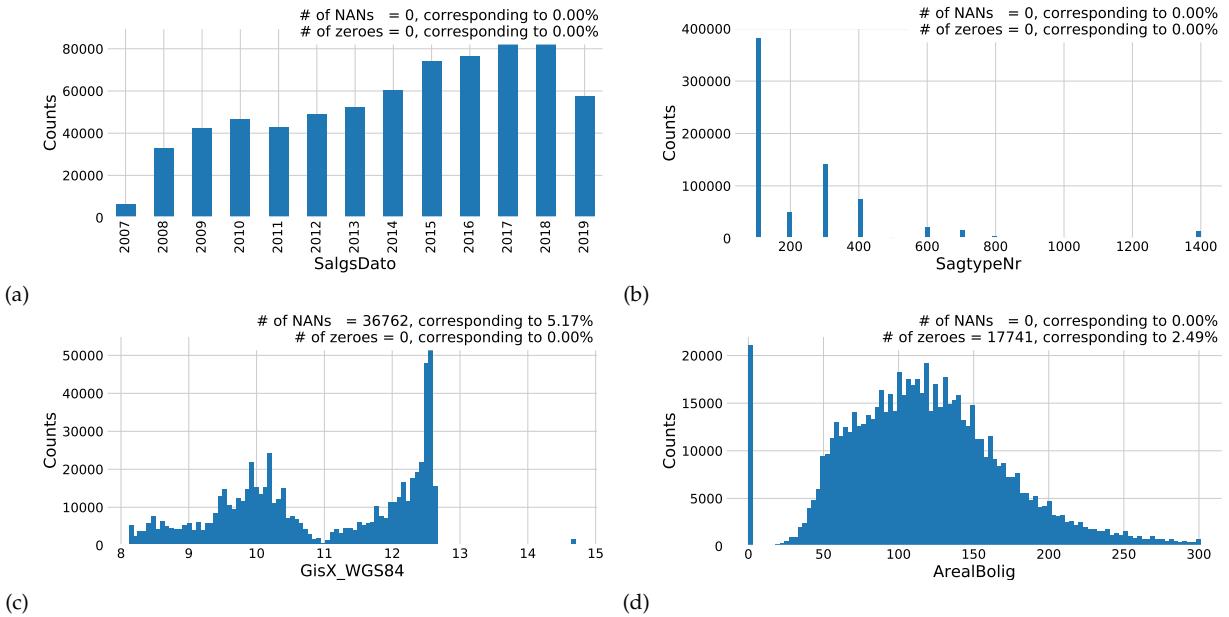
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 4.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 4.1: XXX TODO!.

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

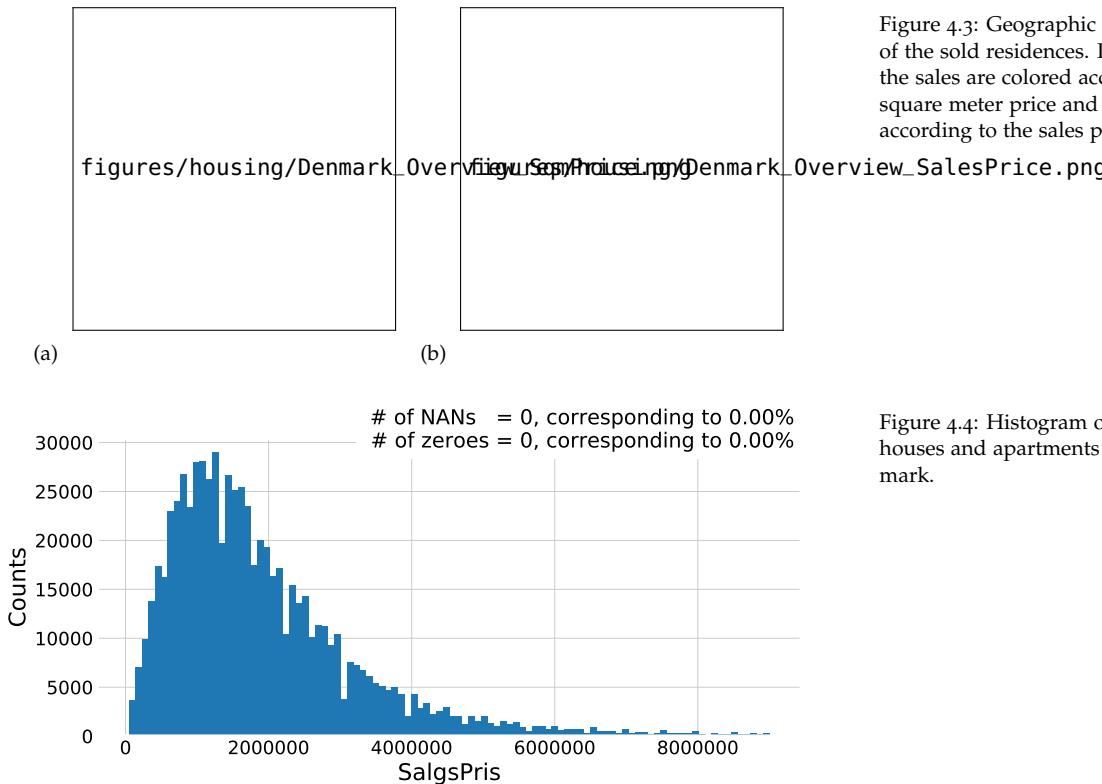


Figure 4.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

Figure 4.4: Histogram of prices of houses and apartments sold in Denmark.

4.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

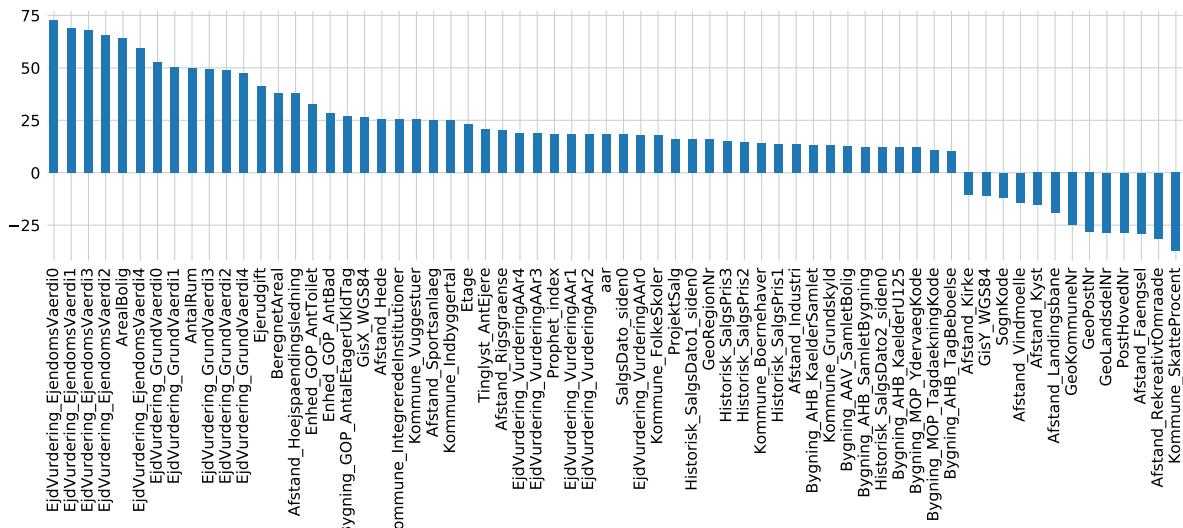


Figure 4.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. ?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

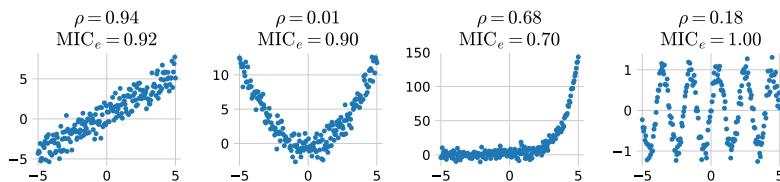


Figure 4.6: MIC non-linear correlation.
XXX TODO!

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SognKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

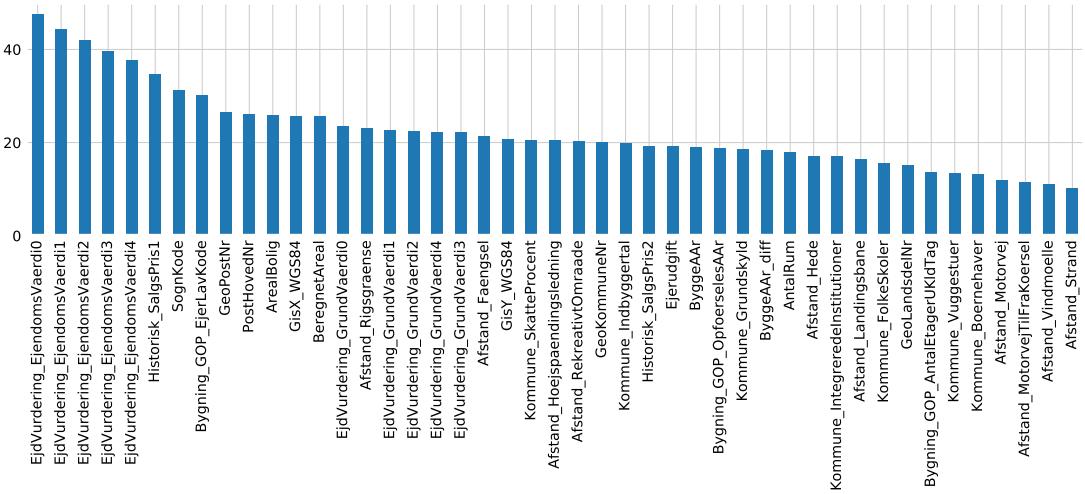
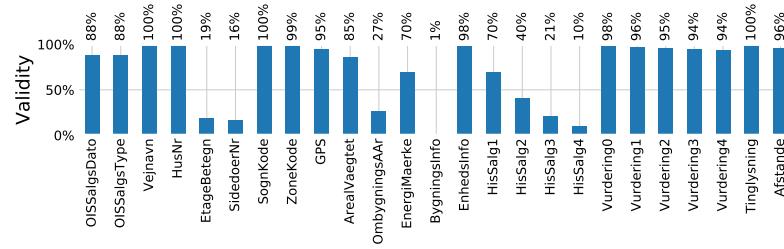


Figure 4.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where MIC > 10%.

4.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaedningsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 4.8: Percentage of valid counts for each variable grouped together in categories.

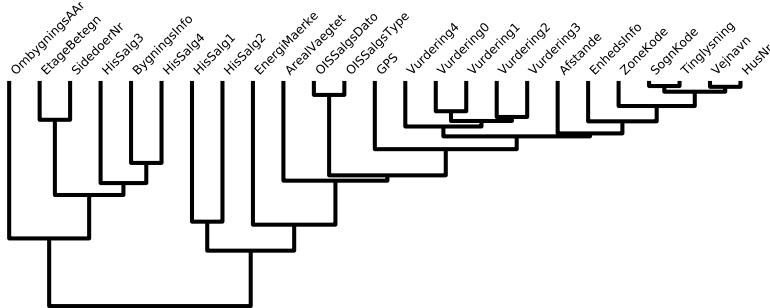


Figure 4.9: Validity Dendrogram
TODO!.

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

4.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 4.2: XXX TODO!.

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 4.3: XXX TODO!.

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 4.4: XXX TODO!.

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ???. The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, EnergiMaerke, is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

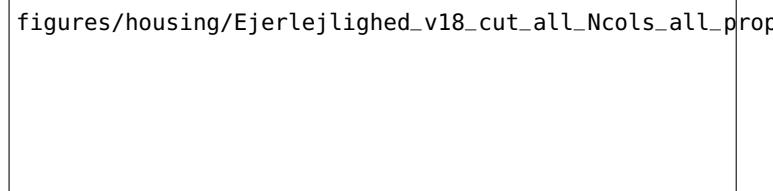
4.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by [] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (4.1)$$

where ϵ_t is a normally distributed error term. [] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ???. In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png

¹¹ In addition to only having the year the house was built

¹² In their paper, [] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 4.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

```
figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb
```

Figure 4.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (4.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

4.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (4.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (4.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (4.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

4.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 4.6: train test split XXX **TODO!**.

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 4.7: train test split tight XXX **TODO!**.

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (4.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (4.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??–?? in the appendix along with all of results for the houses in Table ??–??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

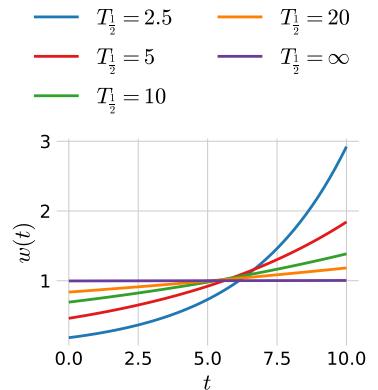


Figure 4.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

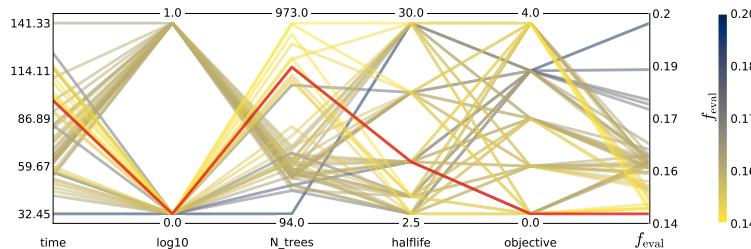
Table 4.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 4.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

4.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` variable controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 4.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The uncertainties here are the standard deviations (one standard deviation) and the 5-folds in the cross validation. The loss functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses.

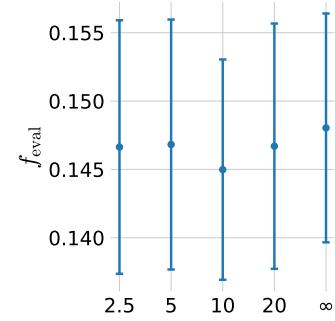


Figure 4.14: XXX Halflife $T_{\frac{1}{2}}$.

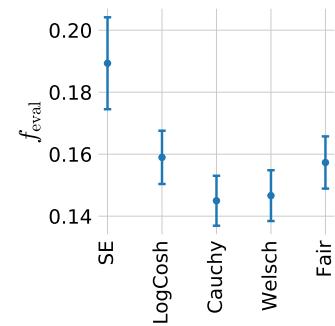


Figure 4.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 4.10: XXX

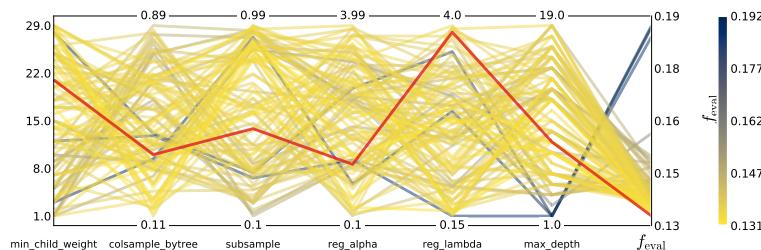


Figure 4.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

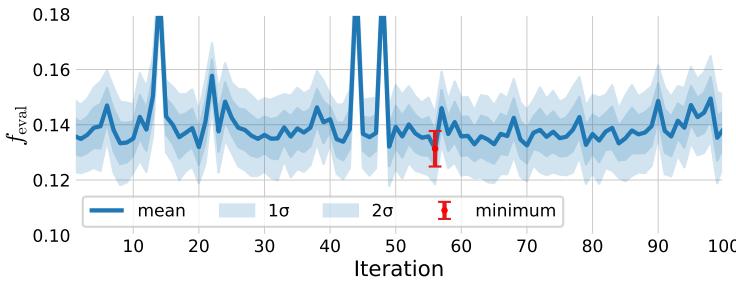


Figure 4.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

4.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 4.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

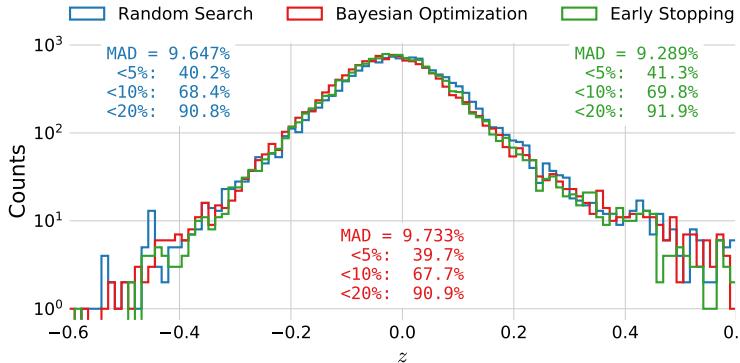


Figure 4.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

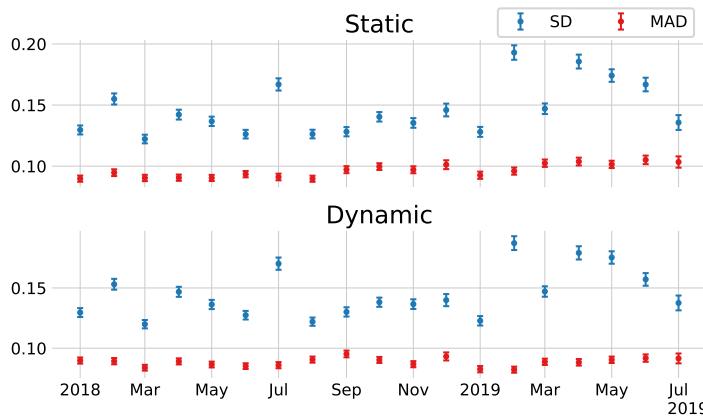


Figure 4.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model’s prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month’s sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (4.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (4.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predicitons are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

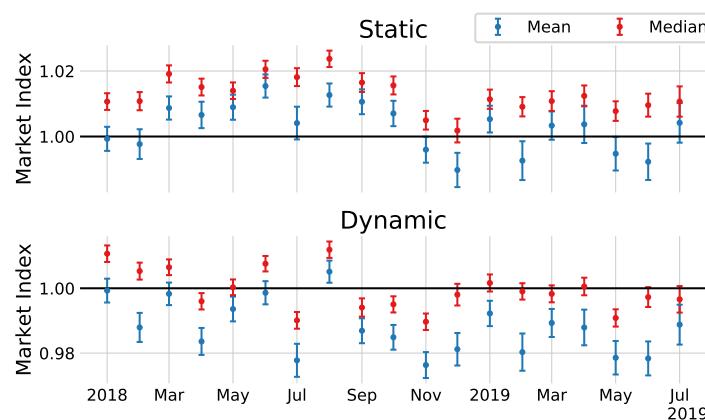


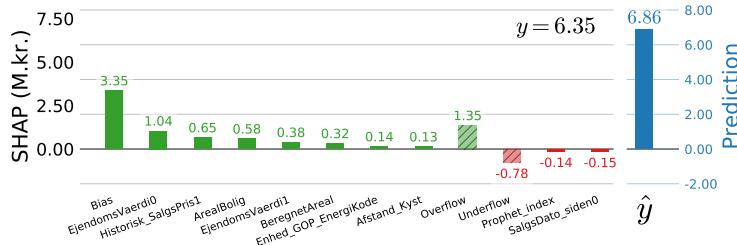
Figure 4.21: XXX TODO!.

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z	Table 4.12: XXX ejer
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$	
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012	
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002	

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z	Table 4.13: XXX villa
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007	
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016	
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002	

4.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 4.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~Of the total number of the entire dataset~~ model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

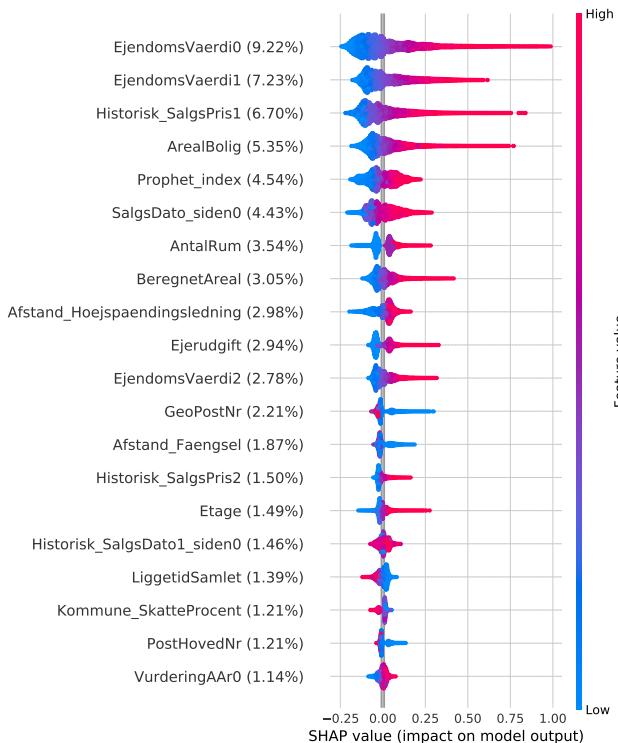
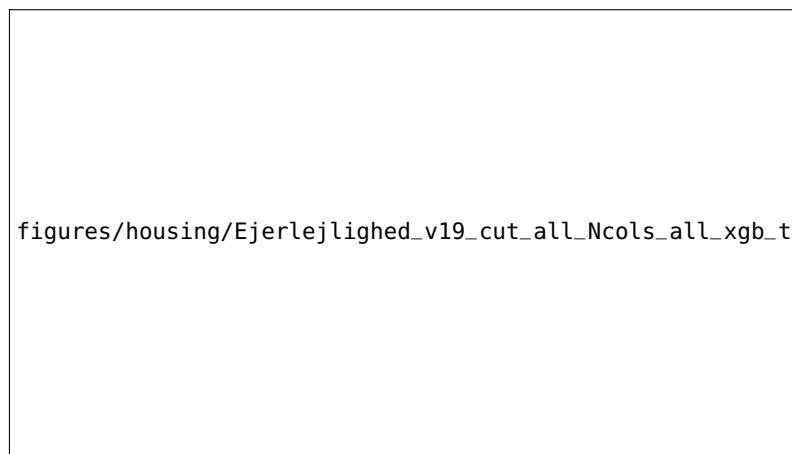


Figure 4.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.



`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 4.24: Feature importance of apartment prices using the XGB-model. XXX

4.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

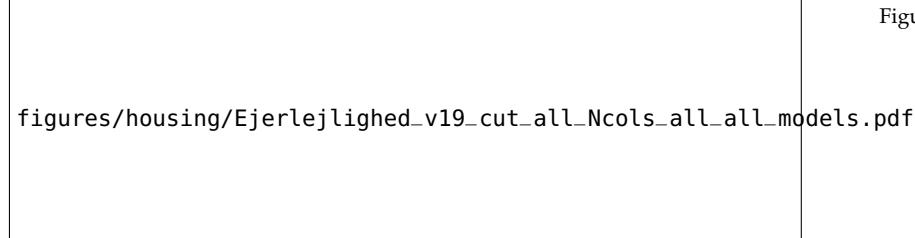


Figure 4.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (4.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (4.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

4.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

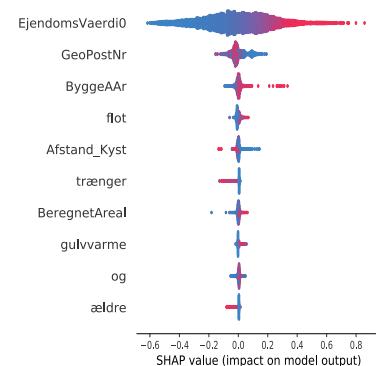


Figure 4.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
 $\arg\min$ 
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

5. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

6. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

7. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

7.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

7.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (7.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (7.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

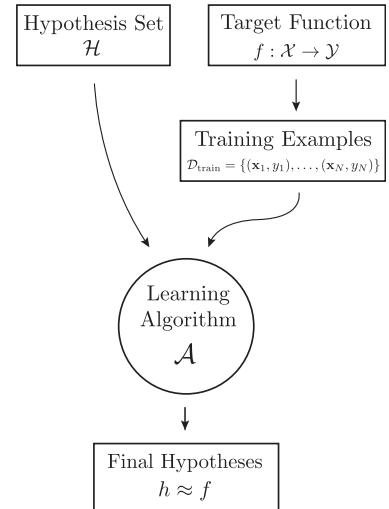


Figure 7.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵Also called expected error or the out-of-sample error.

⁶Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (7.3)$$

7.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypothesis \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (7.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(\mathbf{x}, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 4 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (7.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 5 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (7.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (7.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 6 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (7.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (7.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (7.10)$$

Theorem 5 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (7.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (7.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 6 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (7.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (7.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

7.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypotheses h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: [?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 7 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (7.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (7.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

7.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

7.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (7.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

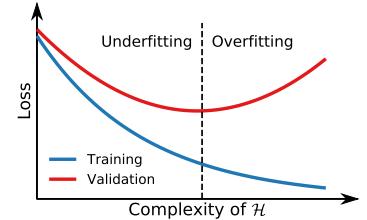


Figure 7.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (7.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned} \quad (7.19)$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (7.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (7.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \end{aligned} \quad (7.22)$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

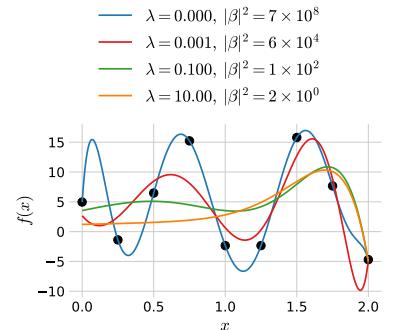


Figure 7.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (7.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by ?] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (7.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

7.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

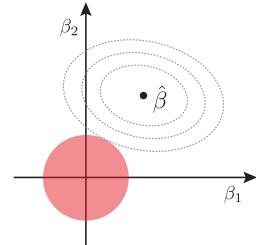


Figure 7.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constraint region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

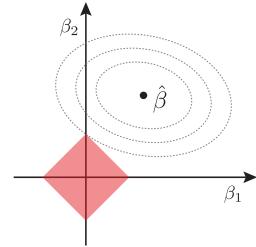


Figure 7.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constraint region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

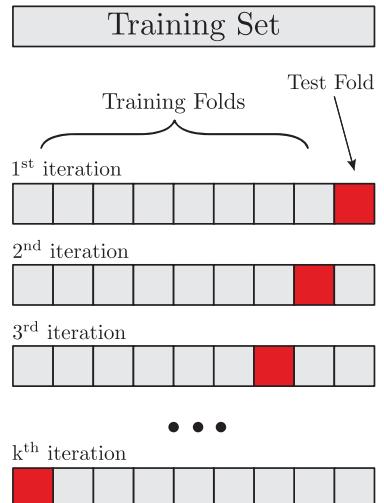


Figure 7.6: k -fold cross validation.

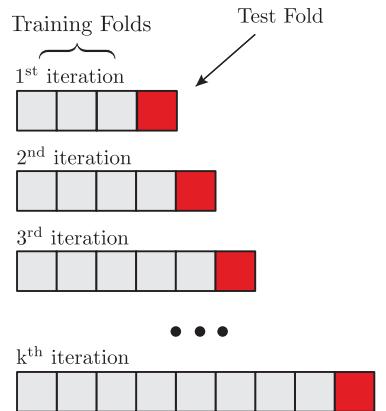


Figure 7.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

7.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

7.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (7.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (7.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (7.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

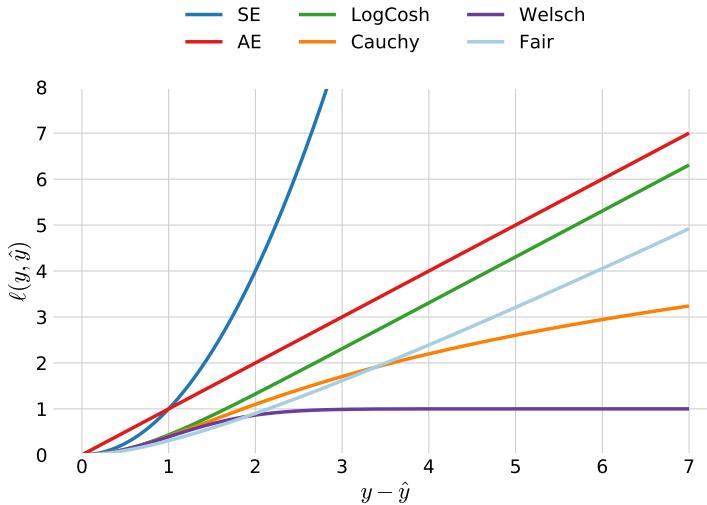


Figure 7.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

7.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (7.28)$$

7.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

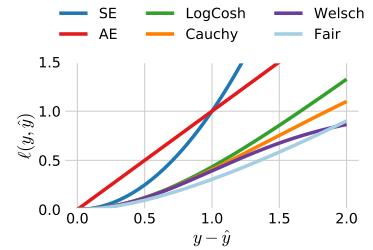


Figure 7.9: Zoom in of Figure ???.

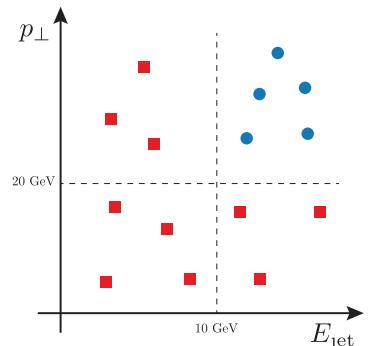
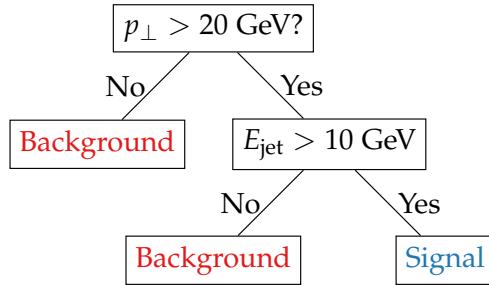


Figure 7.10: Illustration of the cuts a decision tree model make for signal in blue circles and background in red squares. This is a visualization in the feature space of the decision tree seen in Figure ???.



any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

7.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (7.29)$$

Figure 7.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 8 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (7.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (7.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (7.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (7.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (7.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (7.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

7.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

7.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (7.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

7.7.2 Random Search

To circumvent the problems of grid search, [?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (7.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

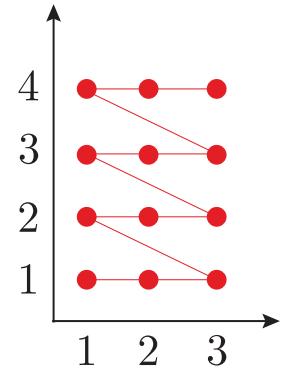


Figure 7.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (7.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

7.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

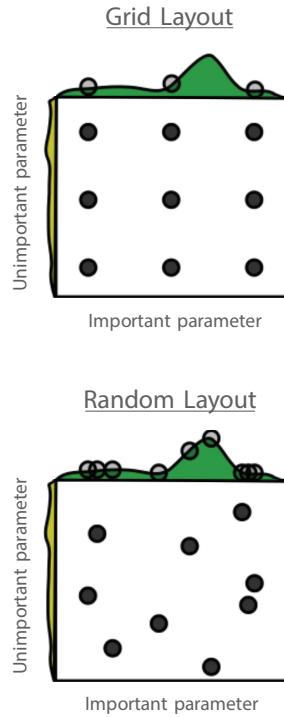


Figure 7.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the datadependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (7.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

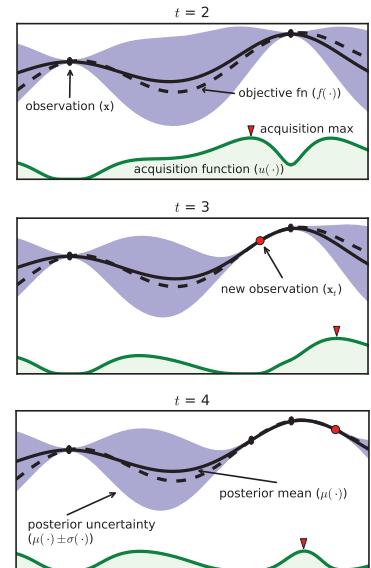


Figure 7.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

7.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (7.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 5 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (7.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 6 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (7.42)$$

Axiom 7 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (7.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (7.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure ???. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{\text{jet}}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{\text{jet}}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{\text{jet}}, p_\perp\}) - f_x(\{E_{\text{jet}}\})]. \end{aligned} \quad (7.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (7.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

⁴³ Not necessarily linearly correlated.

Axiom 8 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (7.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

8. Danish Housing Prices

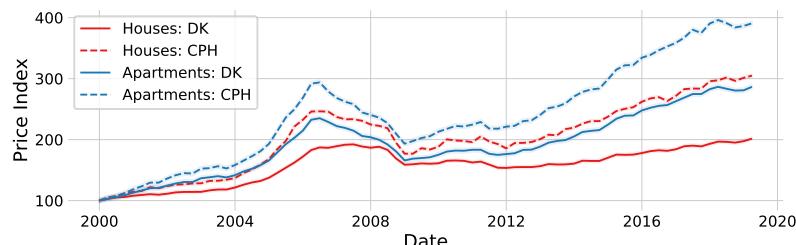
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 8.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

8.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

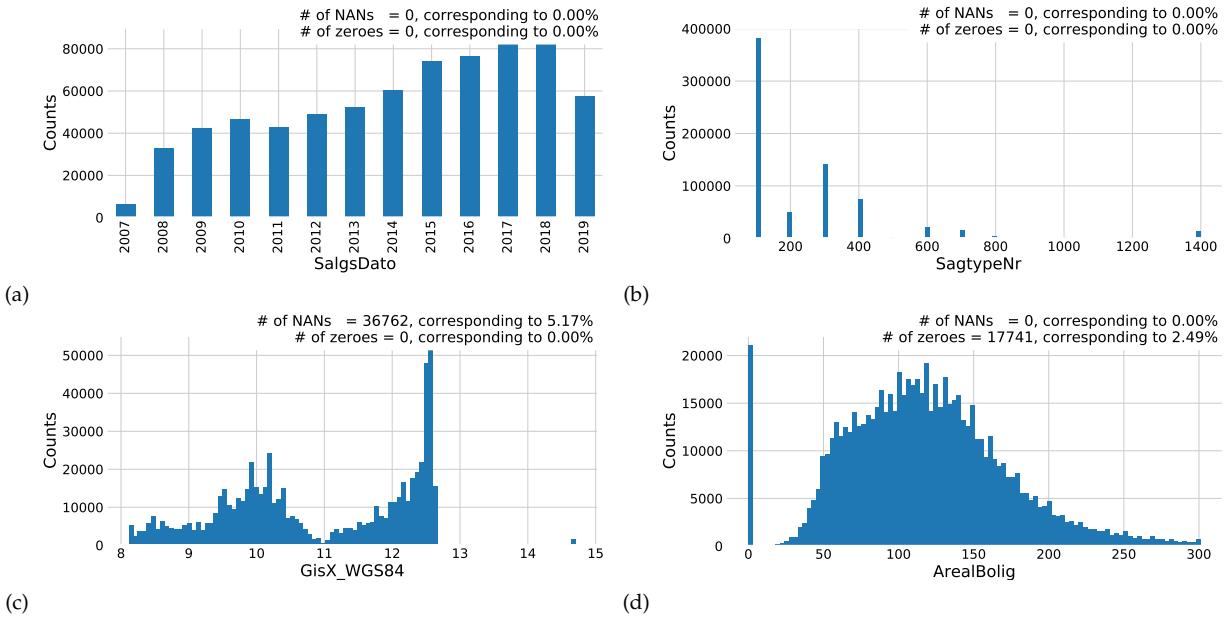
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 8.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Kækkehús
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 8.1: XXX TODO!.

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

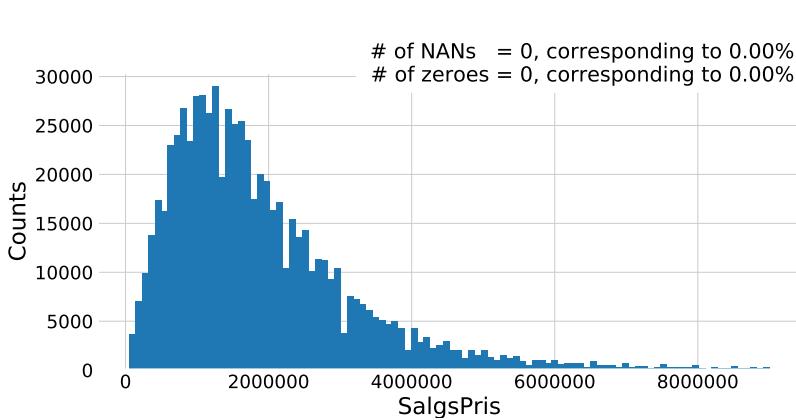
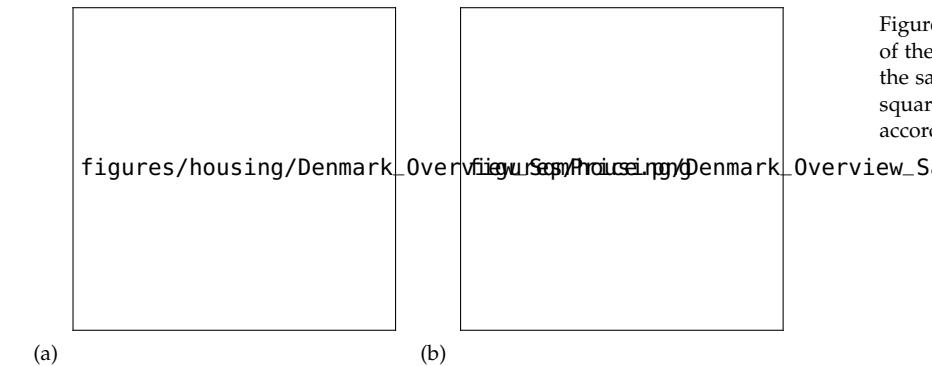


Figure 8.4: Histogram of prices of houses and apartments sold in Denmark.

8.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

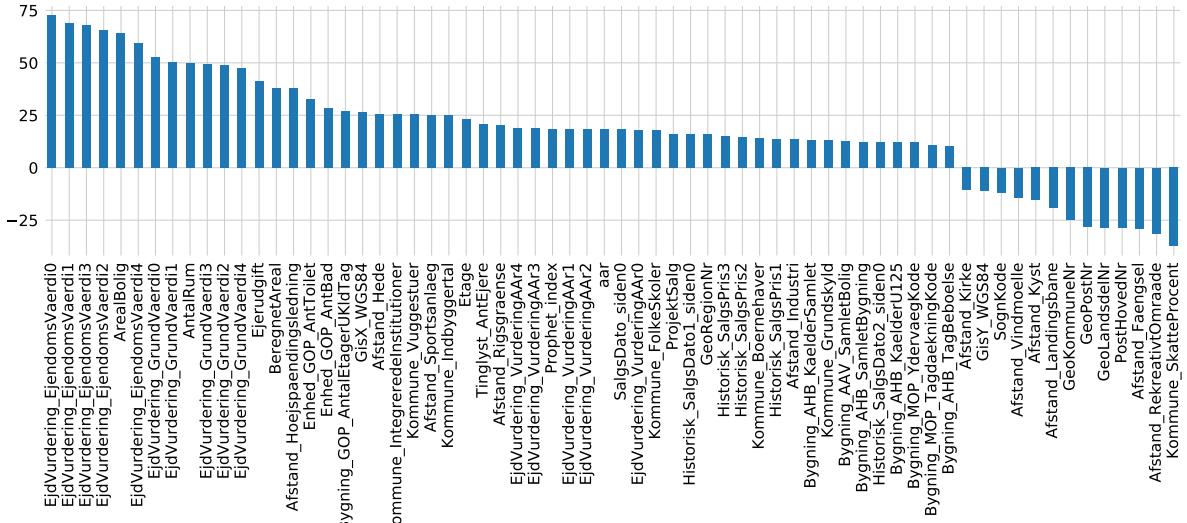


Figure 8.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. [?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

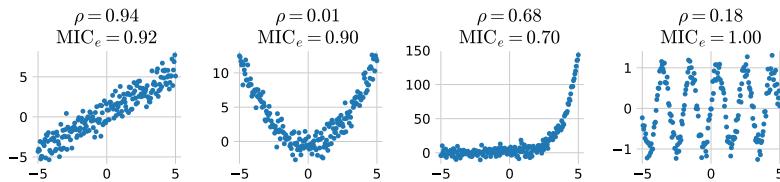
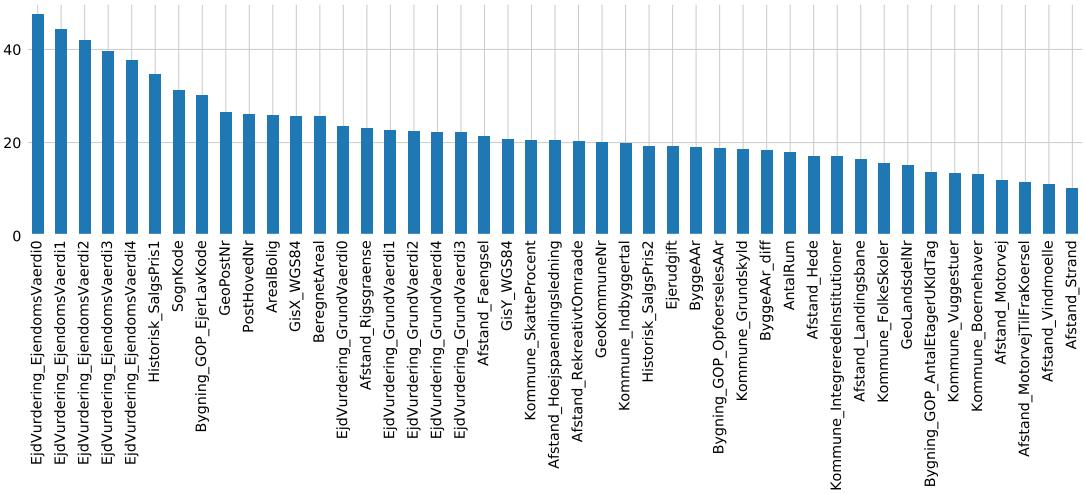


Figure 8.6: MIC non-linear correlation.
XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.



⁹ Distance to: Fængsel, Hede, Højspaedningsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 8.8: Percentage of valid counts for each variable grouped together in categories.

To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be seen that the name of the street, Vejnavn, and the number of the residence, HusNr, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, OmbygningsAAr, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

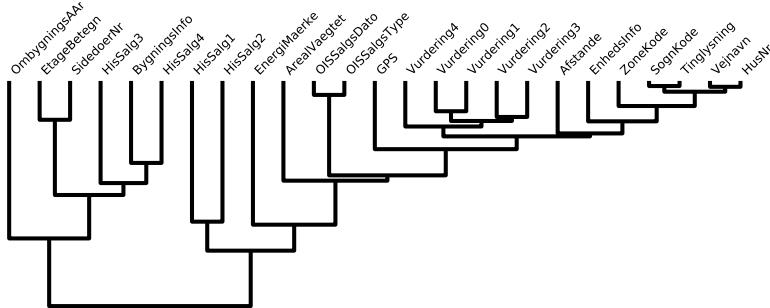


Figure 8.9: Validity Dendrogram
TODO!.

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

8.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 8.2: XXX TODO!.

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 8.3: XXX TODO!.

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 8.4: XXX TODO!.

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ???. The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, `EnergiMaerke`, is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

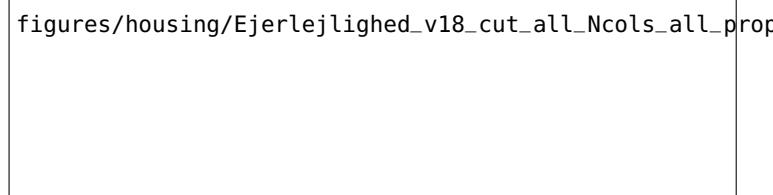
8.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by [] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (8.1)$$

where ϵ_t is a normally distributed error term. [] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ???. In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png`

¹¹ In addition to only having the year the house was built

¹² In their paper, [] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 8.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

```
figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb
```

Figure 8.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (8.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

8.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (8.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (8.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (8.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

8.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 8.6: train test split XXX **TODO!**.

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 8.7: train test split tight XXX **TODO!**.

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (8.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (8.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

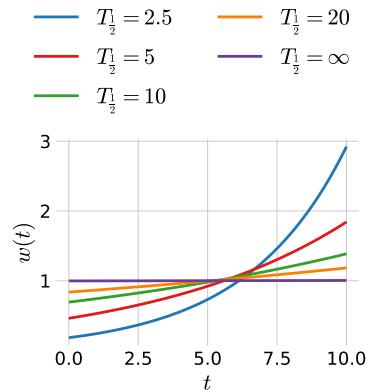


Figure 8.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

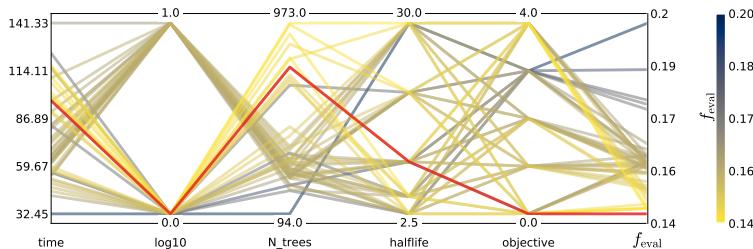
Table 8.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 8.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

8.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 8.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD_0 in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The uncertainties here are the standard deviations (one standard deviation) and the 5-folds in the cross validation. The loss functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses.

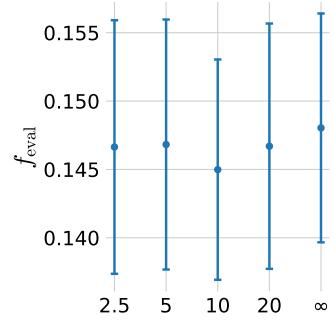


Figure 8.14: XXX Halflife $T_{\frac{1}{2}}$.

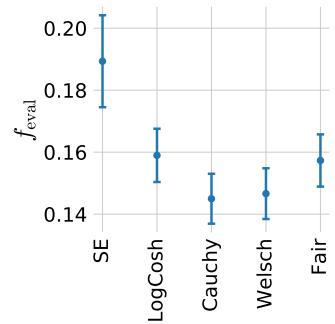


Figure 8.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 8.10: XXX

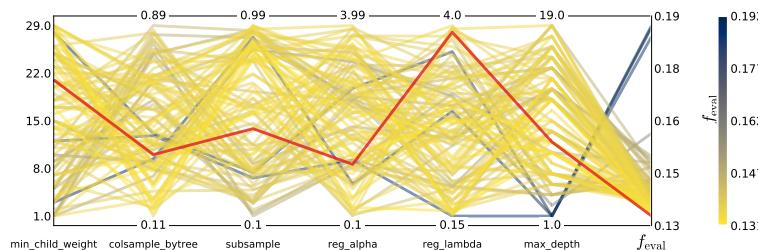


Figure 8.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

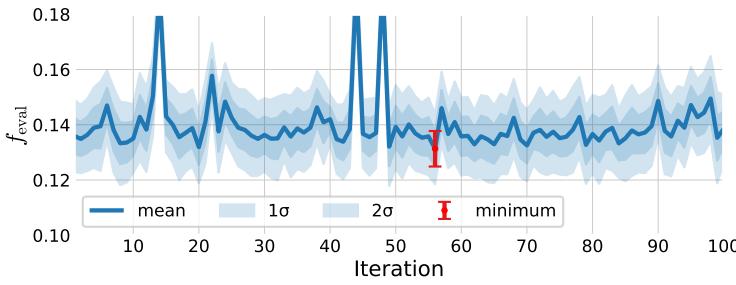


Figure 8.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

8.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 8.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

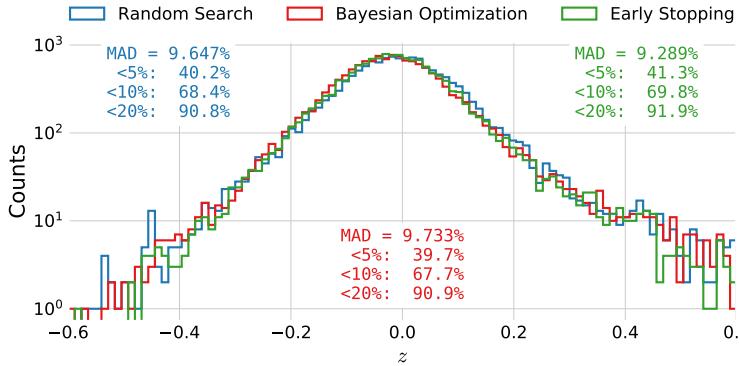


Figure 8.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

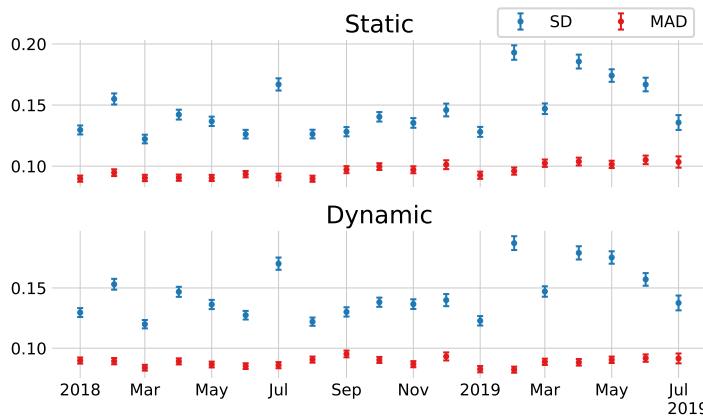


Figure 8.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model’s prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month’s sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (8.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (8.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predictions are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

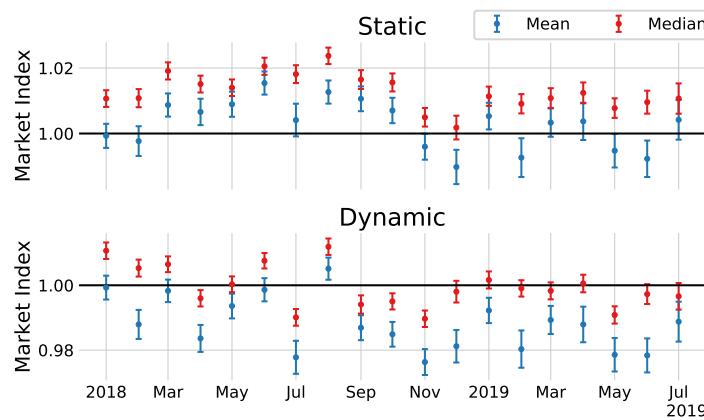


Figure 8.21: XXX TODO!.

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002

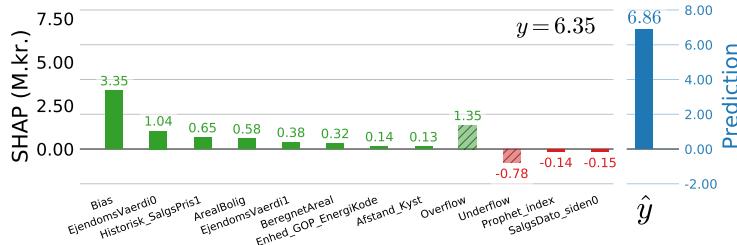
Table 8.12: XXX ejer

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002

Table 8.13: XXX villa

8.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 8.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~Of the total number of the entire dataset~~ model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

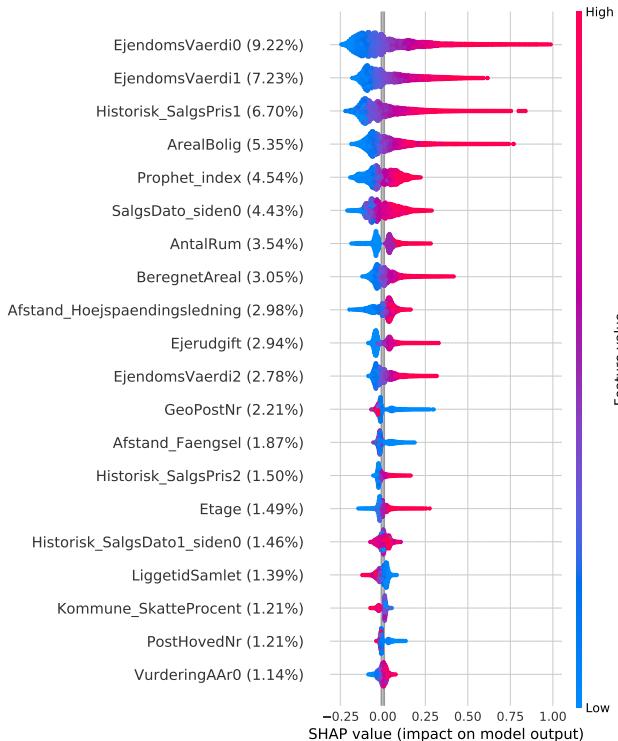
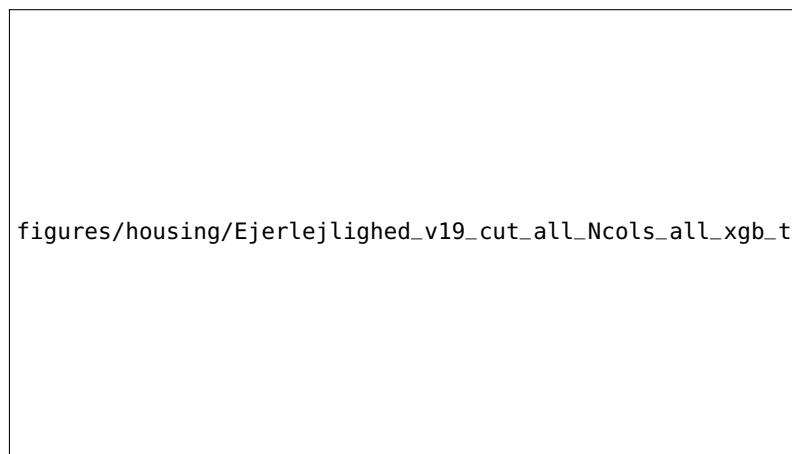


Figure 8.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.



`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 8.24: Feature importance of apartment prices using the XGB-model. XXX

8.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

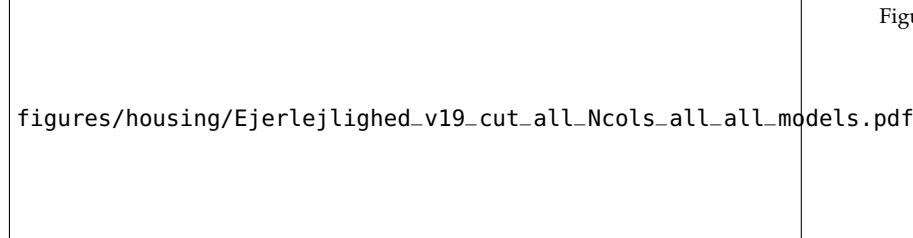


Figure 8.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (8.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (8.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

8.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

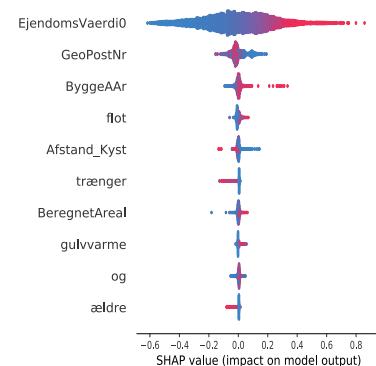


Figure 8.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
*arg min arg min
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

9. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

10. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

11. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

11.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

11.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (11.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (11.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

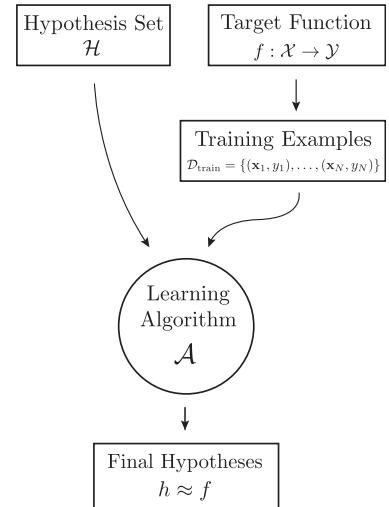


Figure 11.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (11.3)$$

11.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (11.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(\mathbf{x}, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 7 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (11.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 8 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (11.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (11.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 9 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (11.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (11.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (11.10)$$

Theorem 9 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (11.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (11.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 10 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (11.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (11.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

11.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: [?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 11 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (11.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (11.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

11.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

11.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (11.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

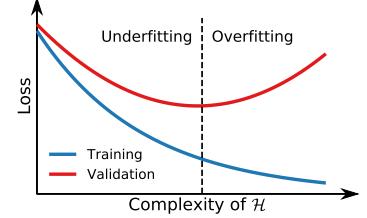


Figure 11.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (11.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (11.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (11.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (11.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \quad (11.22) \end{aligned}$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

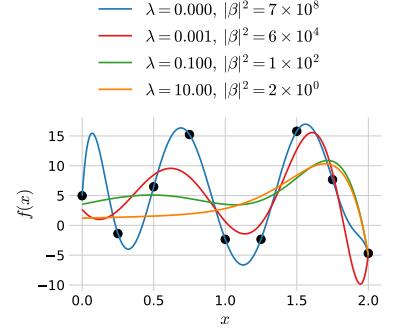


Figure 11.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (11.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by ?] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (11.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

11.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

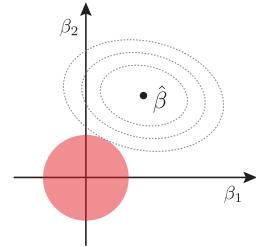


Figure 11.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

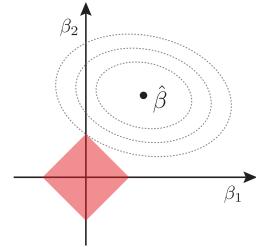


Figure 11.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

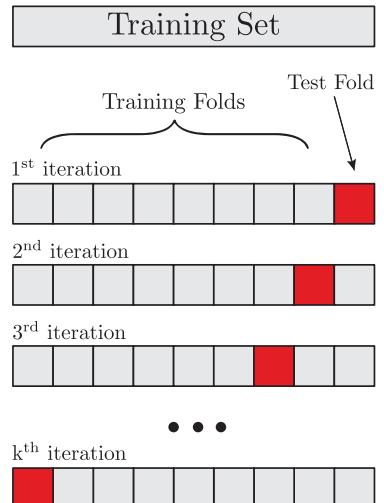


Figure 11.6: k -fold cross validation.

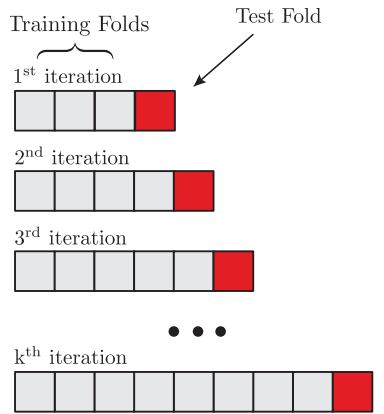


Figure 11.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

11.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

11.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (11.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (11.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (11.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

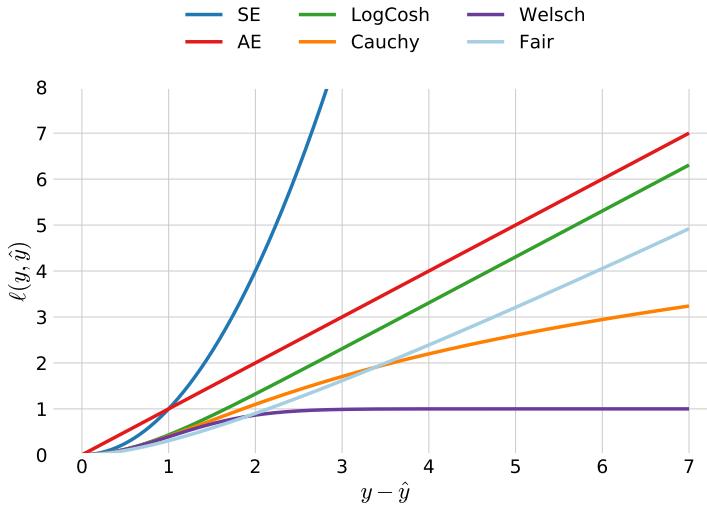


Figure 11.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

11.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (11.28)$$

11.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

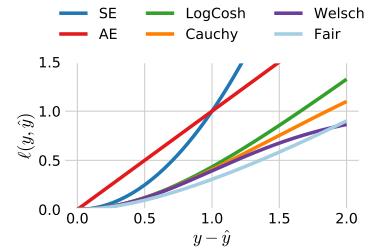


Figure 11.9: Zoom in of Figure ??.

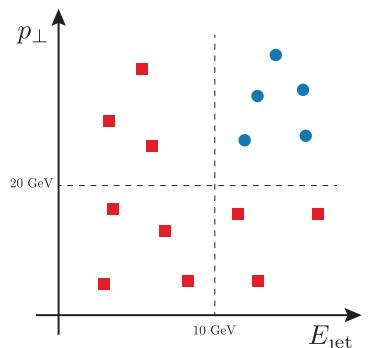


Figure 11.10: Illustration of the cuts a decision tree model make for *signal* in blue circles and *background* in red squares. This is an visualization in the feature space of the decision tree seen in Figure ??.

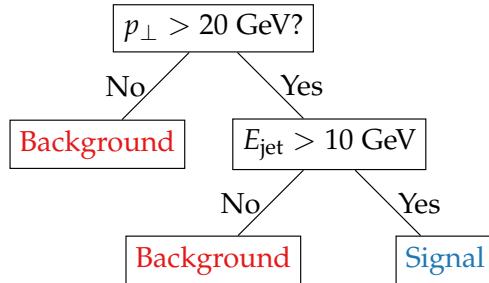


Figure 11.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

11.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (11.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 12 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (11.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (11.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (11.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (11.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (11.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (11.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

11.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

11.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (11.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

11.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (11.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

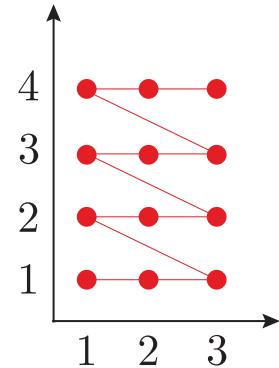


Figure 11.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (11.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

11.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

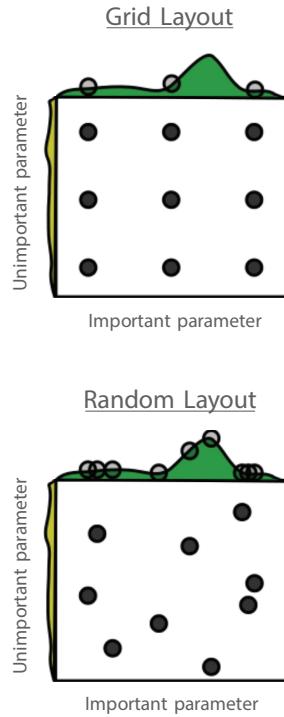


Figure 11.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the datadependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (11.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

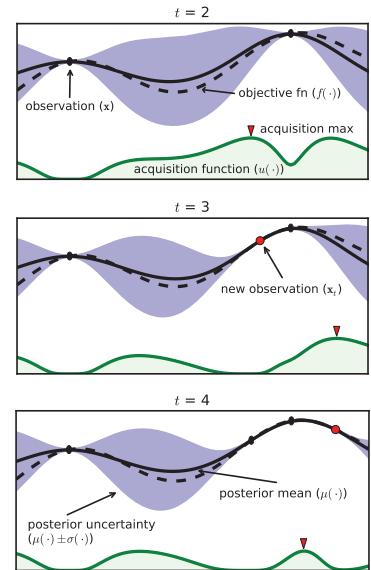


Figure 11.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

11.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (11.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 9 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (11.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 10 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (11.42)$$

Axiom 11 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (11.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (11.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (11.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (11.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

Axiom 12 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (11.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

⁴³ Not necessarily linearly correlated.

12. Danish Housing Prices

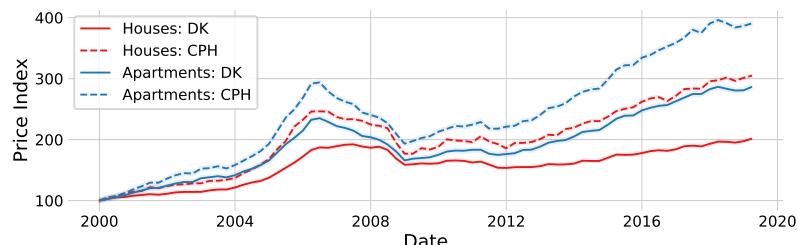
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 12.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

12.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

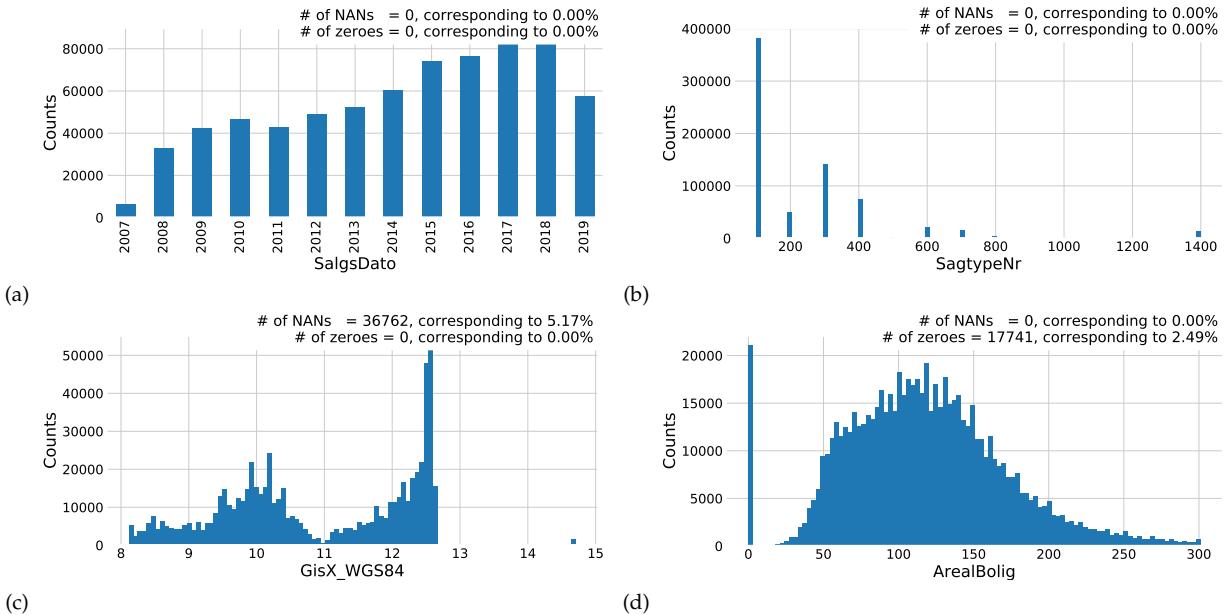
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 12.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 12.1: XXX TODO!

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

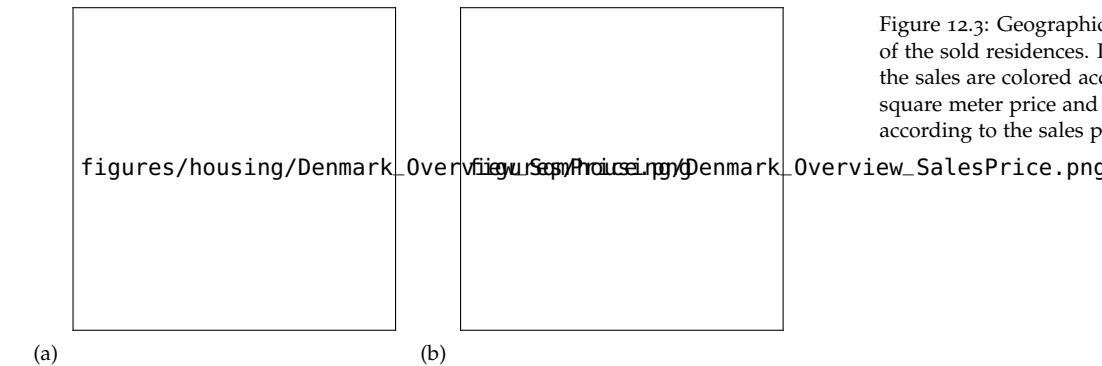


Figure 12.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

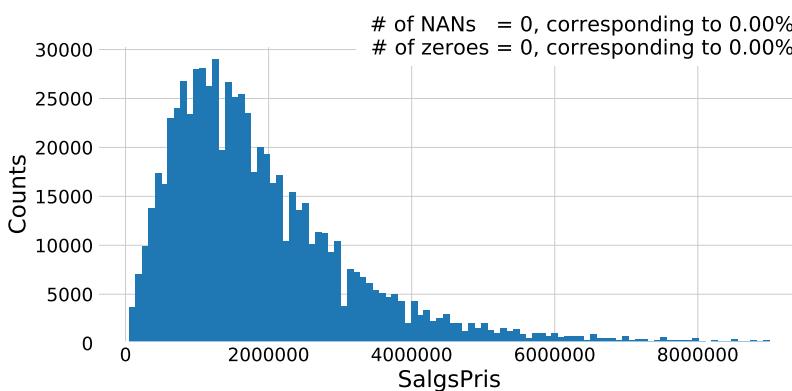


Figure 12.4: Histogram of prices of houses and apartments sold in Denmark.

12.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

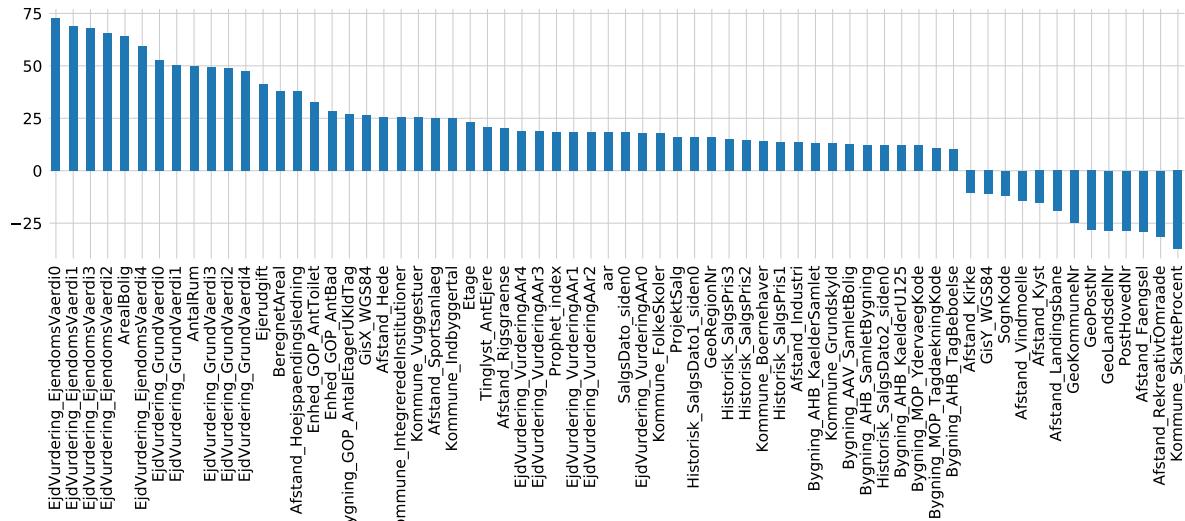


Figure 12.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. ?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

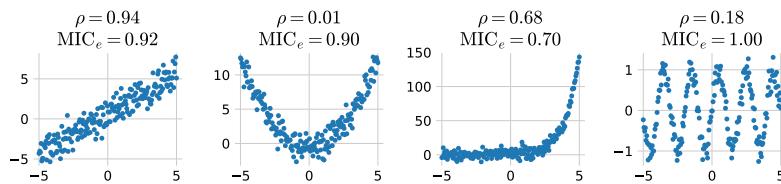


Figure 12.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SognKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

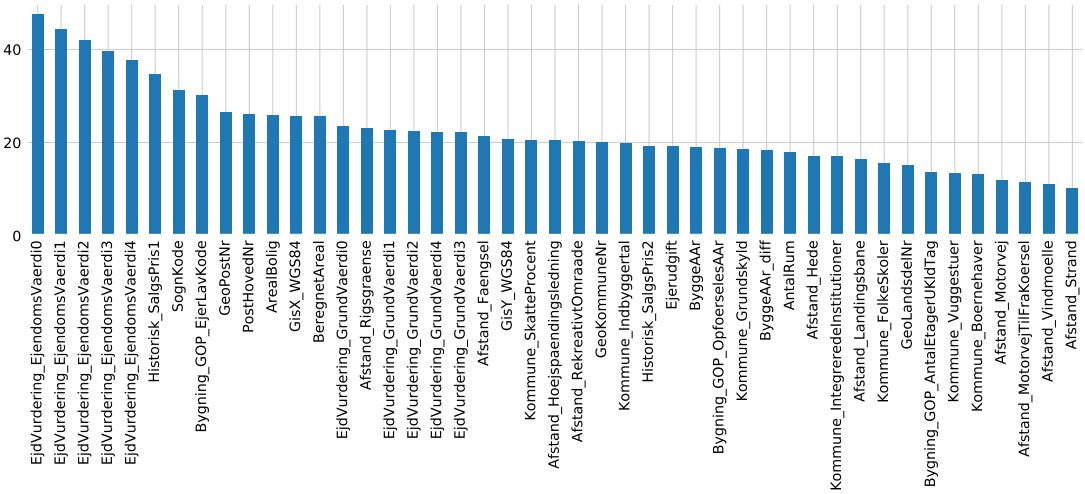
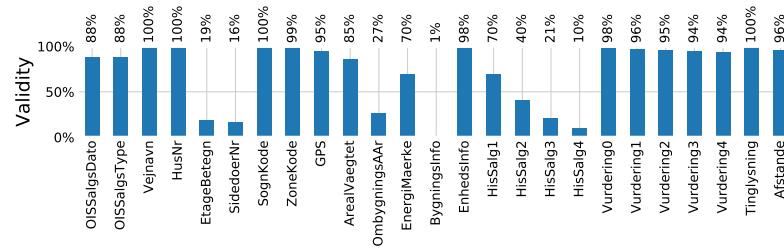


Figure 12.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where $\text{MIC} > 10\%$.

12.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningsAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaedningsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 12.8: Percentage of valid counts for each variable grouped together in categories.

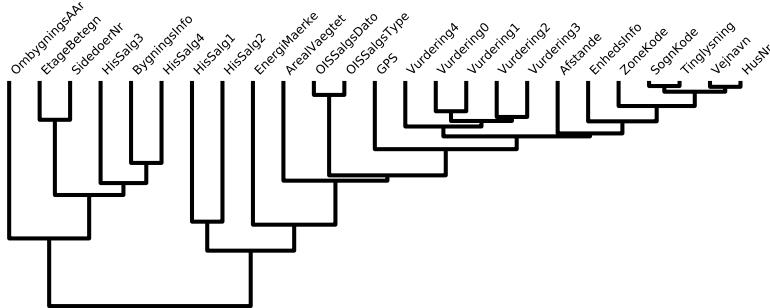


Figure 12.9: Validity Dendrogram
TODO!.

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

12.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 12.2: XXX TODO!

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 12.3: XXX TODO!

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 12.4: XXX TODO!

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ???. The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, `EnergiMaerke`, is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

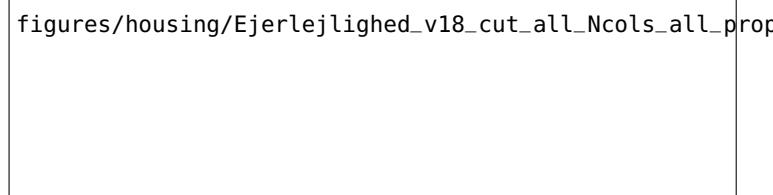
12.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by [] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (12.1)$$

where ϵ_t is a normally distributed error term. [] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ???. In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png`

¹¹ In addition to only having the year the house was built

¹² In their paper, [] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 12.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb`

Figure 12.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (12.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

12.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (12.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (12.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (12.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

12.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 12.6: train test split XXX **TODO!**

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 12.7: train test split tight XXX **TODO!**

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (12.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ??.

A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (12.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

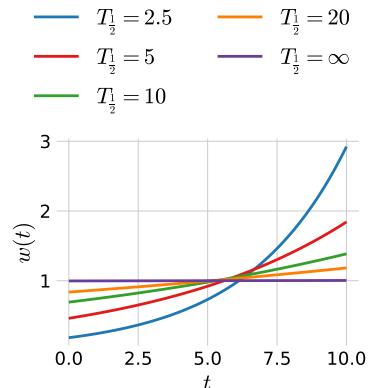


Figure 12.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

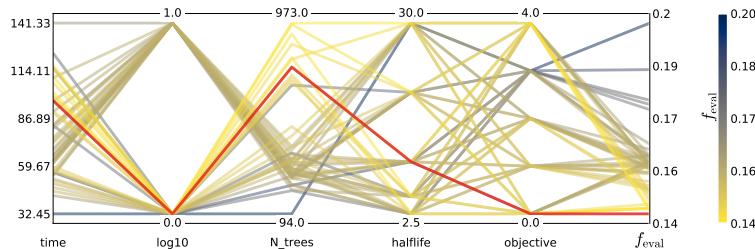
Table 12.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 12.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

12.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 12.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD_0 in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The standard deviations (and MAD_0) and the 5-folds in the cross validation's `Cauchy` (0), `Fair` (1), `LogCosh` (2) `SquaredError` (3), and `Welsch` (4) are mapped to the integers in the parentheses.

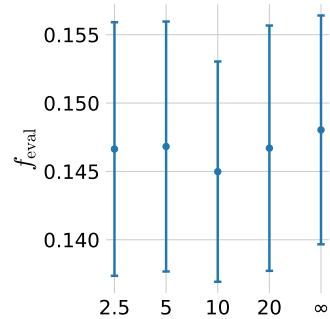


Figure 12.14: XXX Halflife $T_{\frac{1}{2}}$.

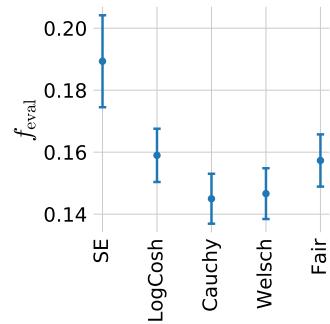


Figure 12.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 12.10: XXX

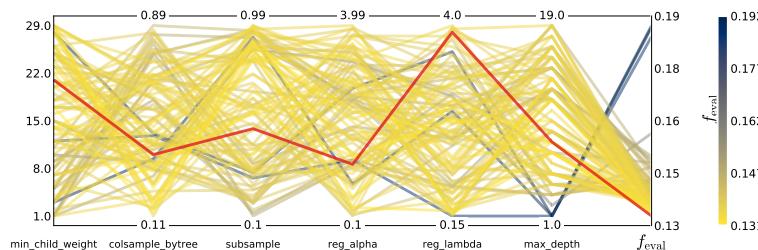


Figure 12.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

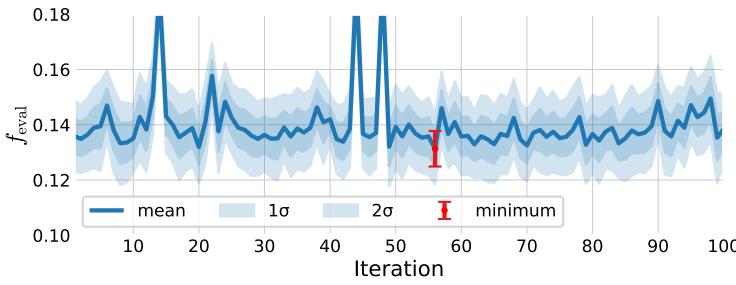


Figure 12.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

12.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 12.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

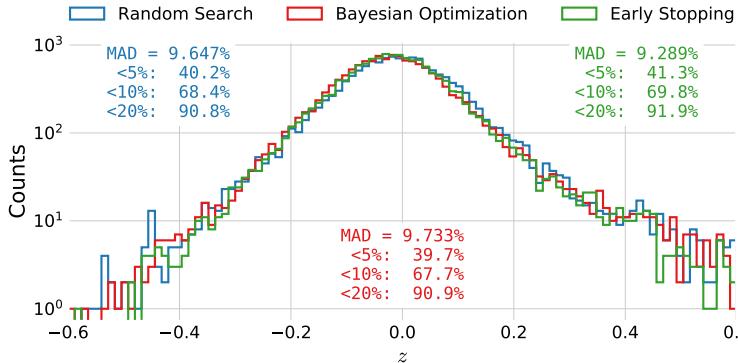


Figure 12.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

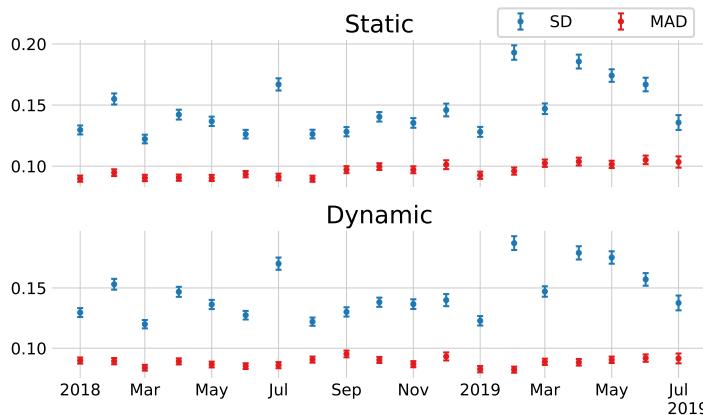


Figure 12.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model’s prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month’s sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (12.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (12.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predicitons are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

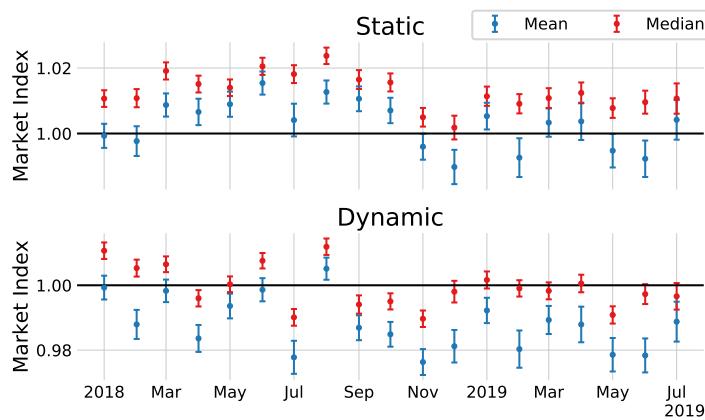


Figure 12.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002

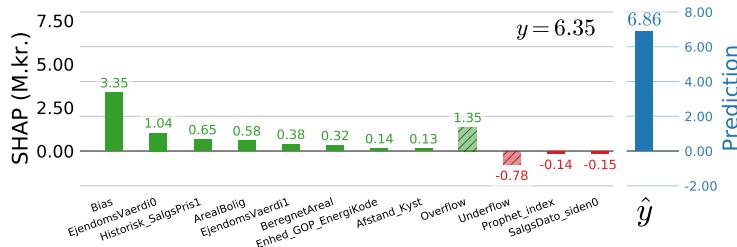
Table 12.12: XXX ejer

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002

Table 12.13: XXX villa

12.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 12.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~Of the total number of the entire dataset~~ model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

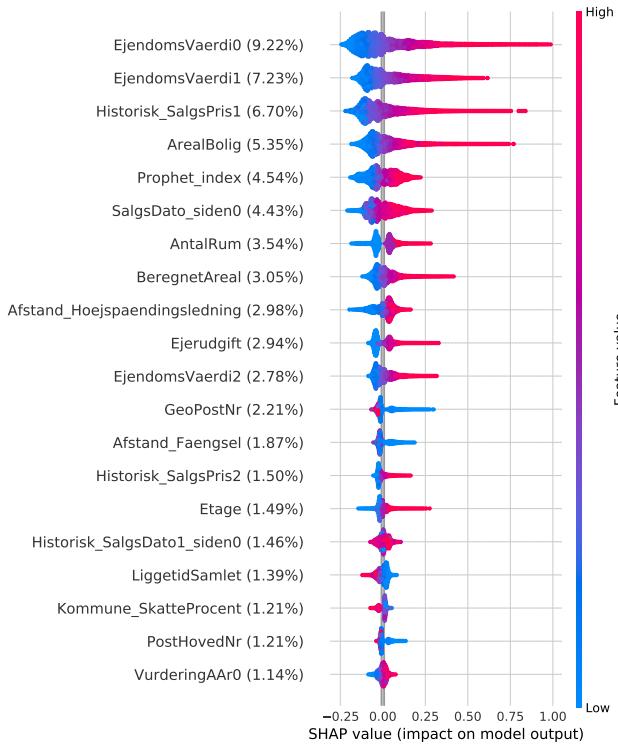


Figure 12.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.

`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 12.24: Feature importance of apartment prices using the XGB-model. XXX

12.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

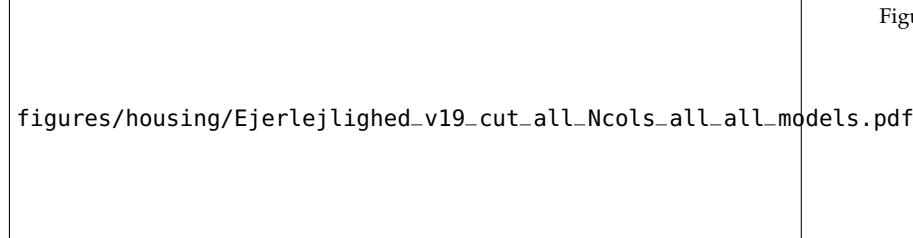


Figure 12.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (12.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (12.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

12.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

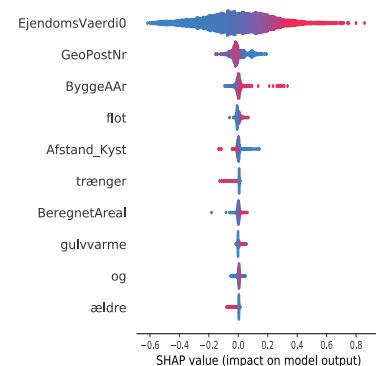


Figure 12.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
 $\arg\min$ 
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

13. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

14. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

15. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

15.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

15.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (15.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (15.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

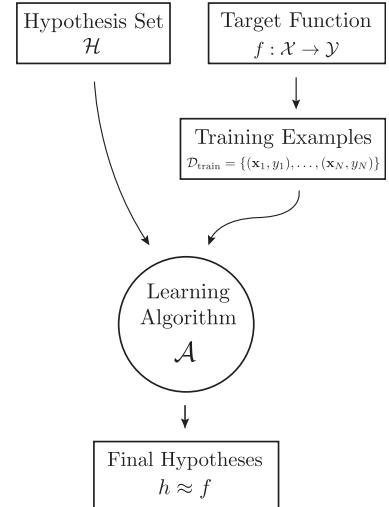


Figure 15.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (15.3)$$

15.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (15.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(x, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 10 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (15.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 11 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (15.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (15.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 12 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (15.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (15.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (15.10)$$

Theorem 13 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (15.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (15.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 14 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (15.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (15.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

15.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: [?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 15 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (15.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (15.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

15.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

15.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (15.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

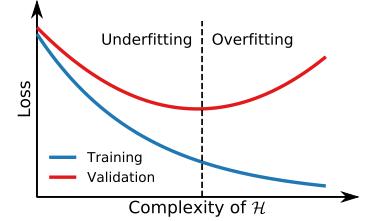


Figure 15.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (15.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (15.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (15.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (15.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \quad (15.22) \end{aligned}$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

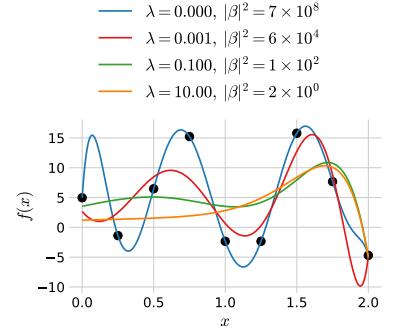


Figure 15.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (15.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by [1] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (15.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

15.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

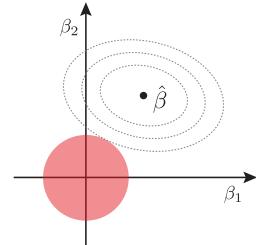


Figure 15.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

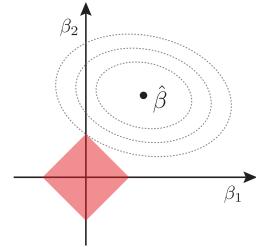


Figure 15.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

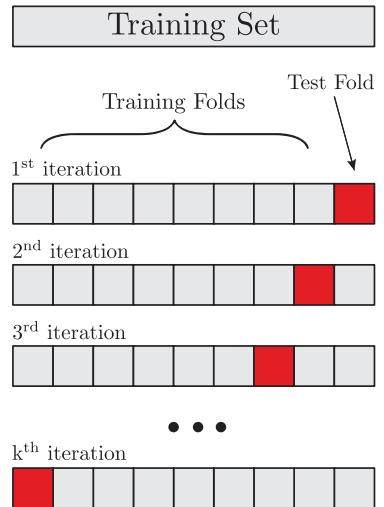


Figure 15.6: k -fold cross validation.

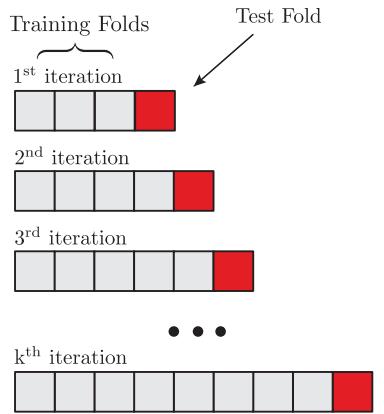


Figure 15.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

15.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

15.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (15.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (15.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (15.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

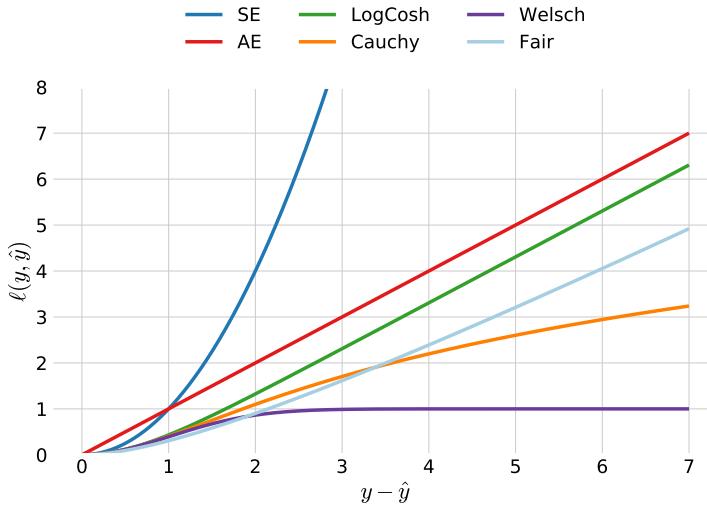


Figure 15.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

15.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (15.28)$$

15.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

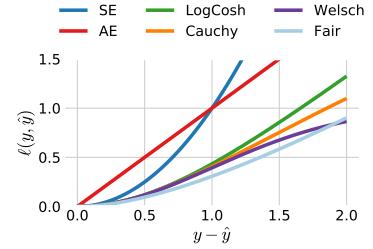


Figure 15.9: Zoom in of Figure ??.

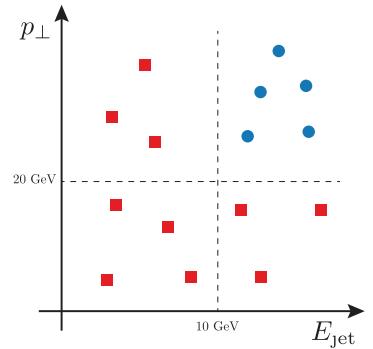


Figure 15.10: Illustration of the cuts a decision tree model make for signal in blue circles and background in red squares. This is a visualization in the feature space of the decision tree seen in Figure ??.

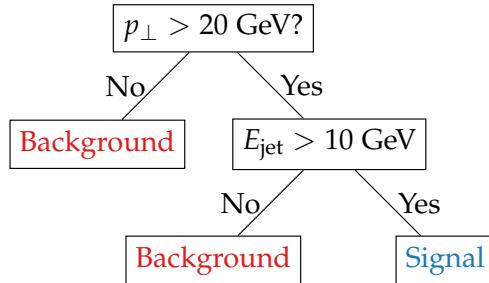


Figure 15.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

15.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (15.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 16 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (15.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (15.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (15.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (15.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (15.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (15.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

15.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

15.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (15.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

15.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (15.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

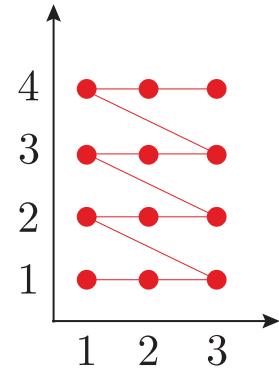


Figure 15.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (15.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

15.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

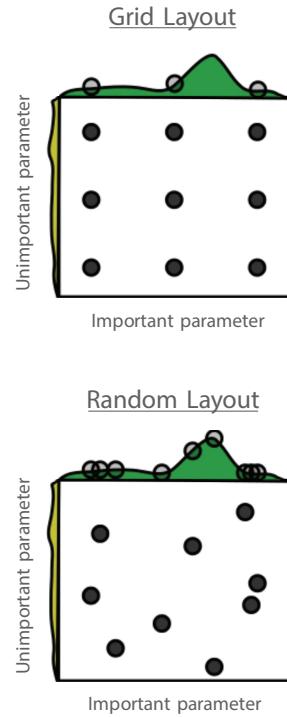


Figure 15.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the data-dependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (15.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

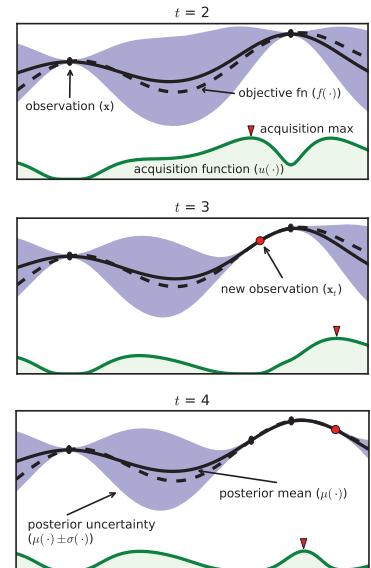


Figure 15.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

15.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (15.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 13 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (15.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 14 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (15.42)$$

Axiom 15 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (15.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (15.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (15.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (15.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

Axiom 16 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (15.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

⁴³ Not necessarily linearly correlated.

16. Danish Housing Prices

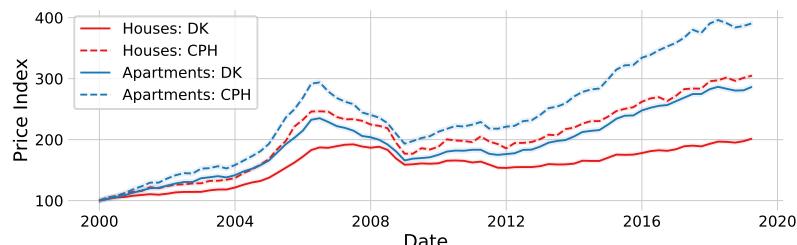
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 16.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

16.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

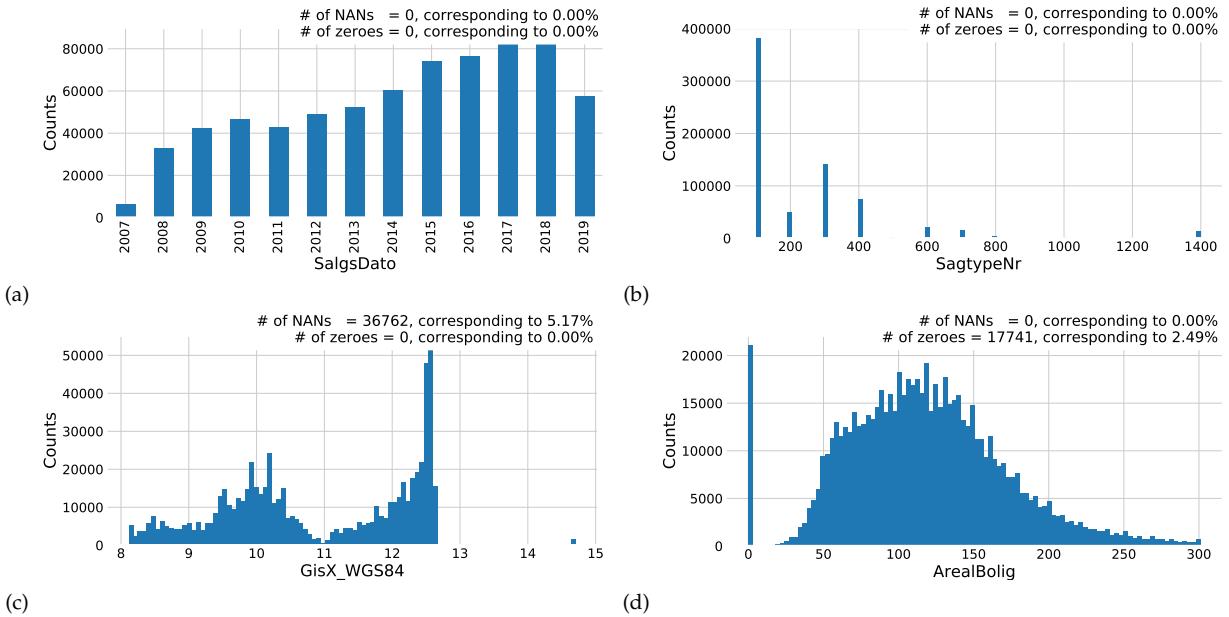
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 16.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 16.1: XXX TODO!

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

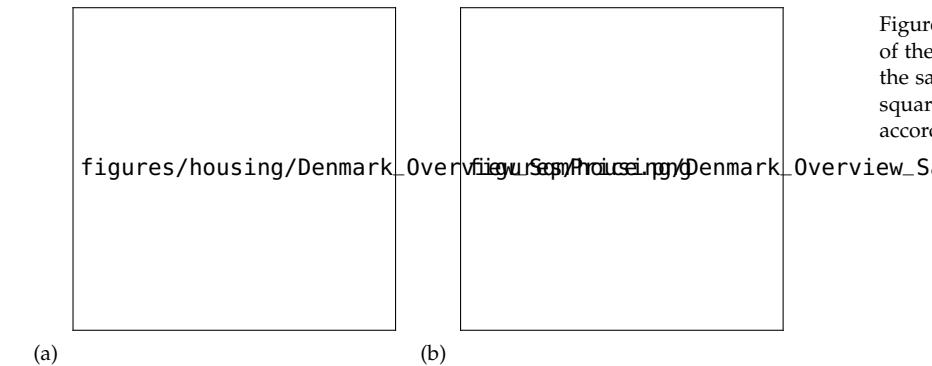


Figure 16.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

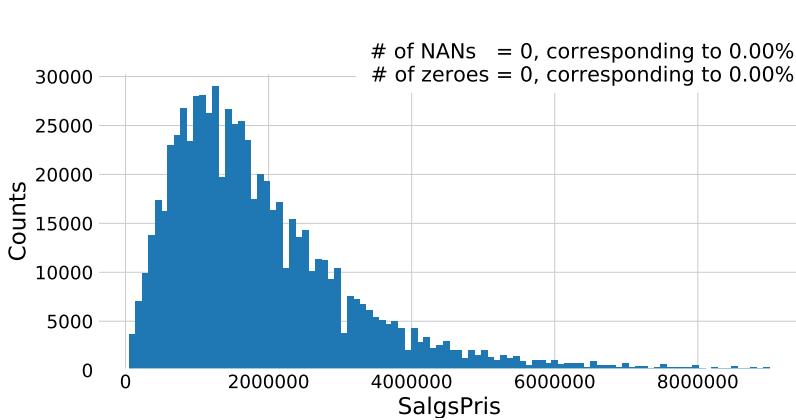


Figure 16.4: Histogram of prices of houses and apartments sold in Denmark.

16.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

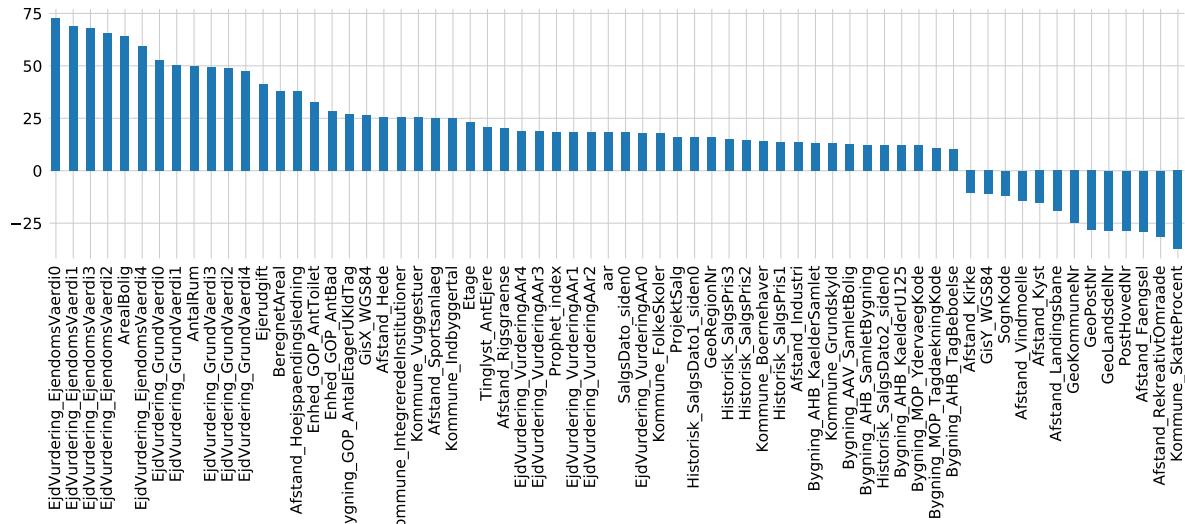


Figure 16.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. [?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

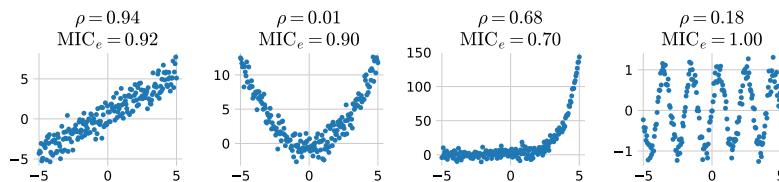


Figure 16.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

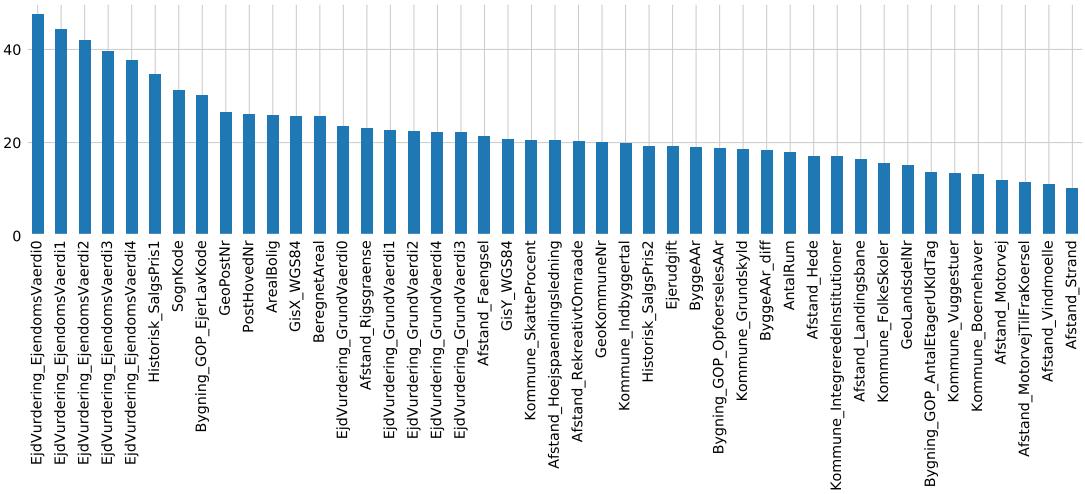
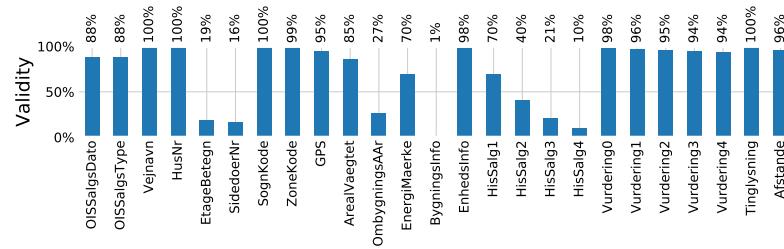


Figure 16.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where MIC > 10%.

16.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningsAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaerdingsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 16.8: Percentage of valid counts for each variable grouped together in categories.

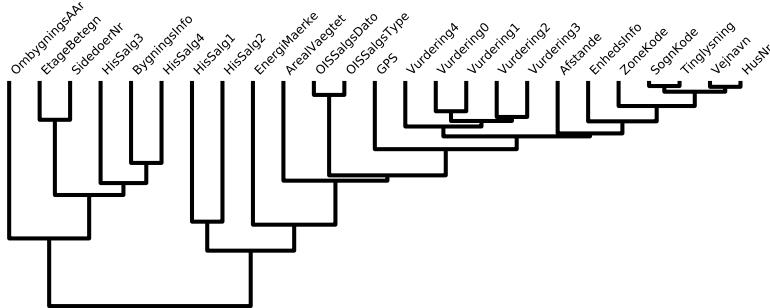


Figure 16.9: Validity Dendrogram
TODO!.

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

16.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 16.2: XXX TODO!.

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 16.3: XXX TODO!.

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 16.4: XXX TODO!.

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ?? . The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, EnergiMaerke , is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

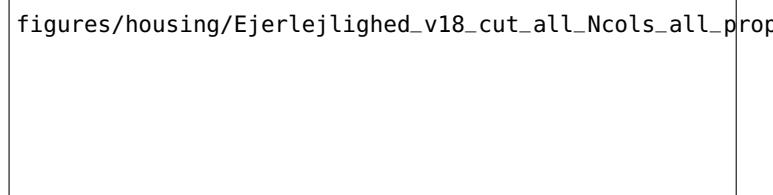
16.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by ?] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (16.1)$$

where ϵ_t is a normally distributed error term. ?] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ?? . In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png

¹¹ In addition to only having the year the house was built

¹² In their paper, ?] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 16.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

```
figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb
```

Figure 16.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (16.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

16.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (16.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (16.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (16.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

16.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 16.6: train test split XXX **TODO!**.

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 16.7: train test split tight XXX **TODO!**.

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (16.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (16.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

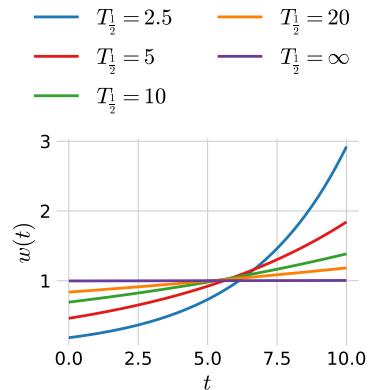


Figure 16.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

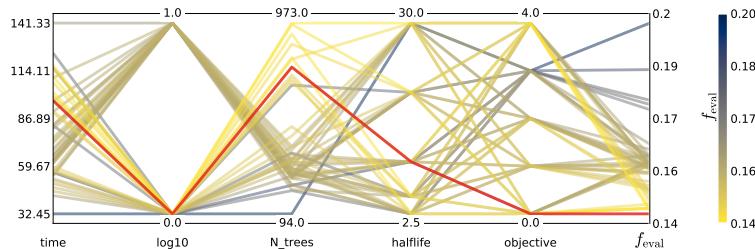
Table 16.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 16.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

16.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 16.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The standard deviations (and propagation) and the 5 folds in the cross validation's `Cauchy` (0), `Fair` (1), `LogCosh` (2) `SquaredError` (3), and `Welsch` (4) are mapped to the integers in the parentheses.

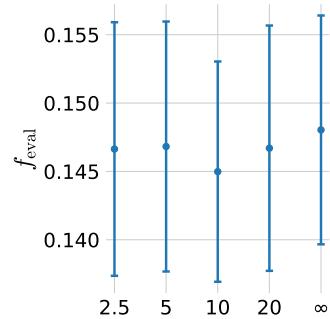


Figure 16.14: XXX Halflife $T_{\frac{1}{2}}$.

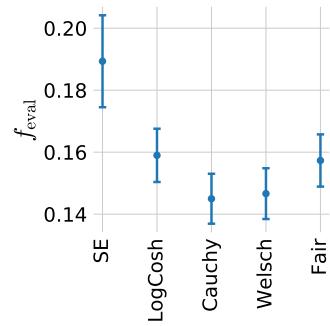


Figure 16.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 16.10: XXX

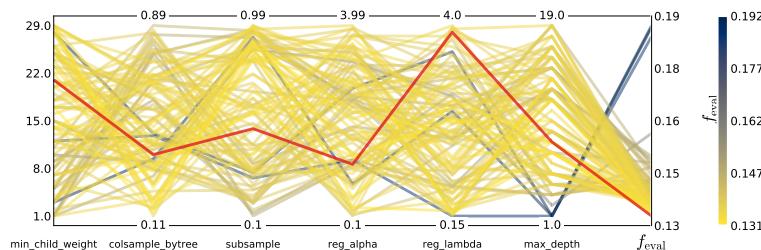


Figure 16.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

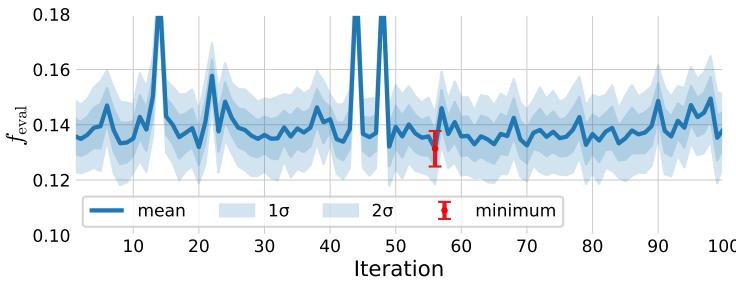


Figure 16.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

16.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 16.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as `MAD`

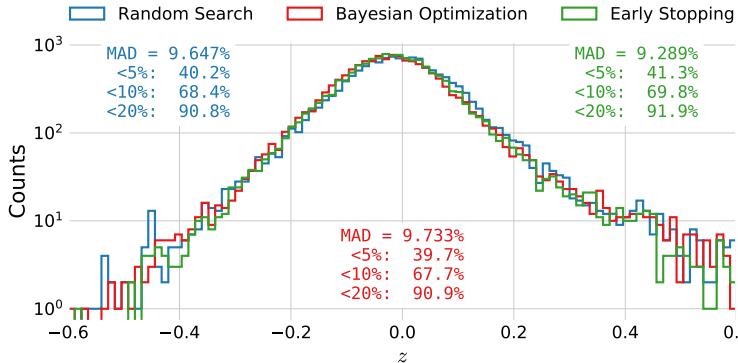


Figure 16.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

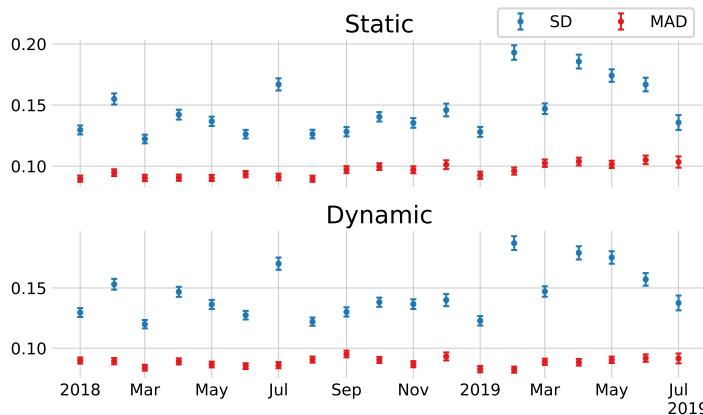


Figure 16.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month's sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

	Train	Test	2019
Normal	5.80 %	4.97 %	6.19 %
Tight	5.69 %	4.94 %	6.19 %

Table 16.11: XXX

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (16.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (16.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predictions are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

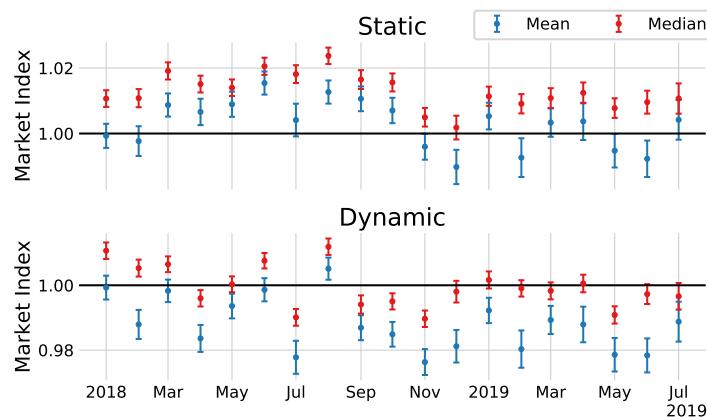


Figure 16.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002

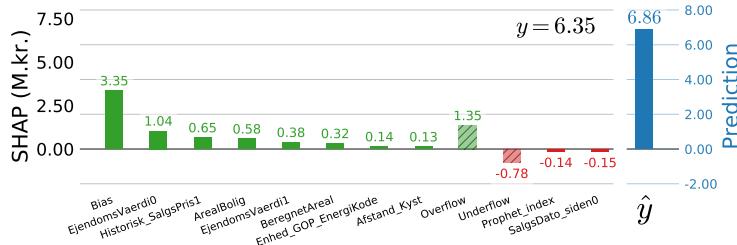
Table 16.12: XXX ejer

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002

Table 16.13: XXX villa

16.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 16.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~On the right hand side of the plot the entire dataset model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.~~

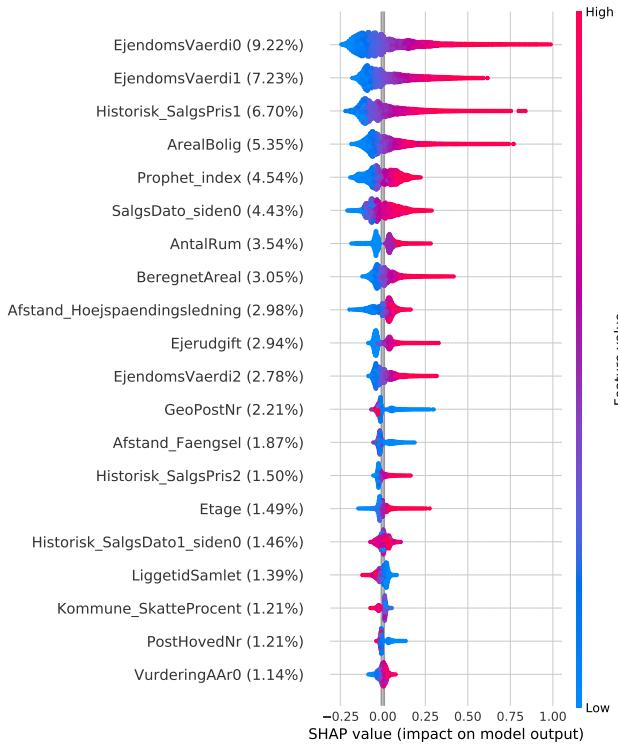
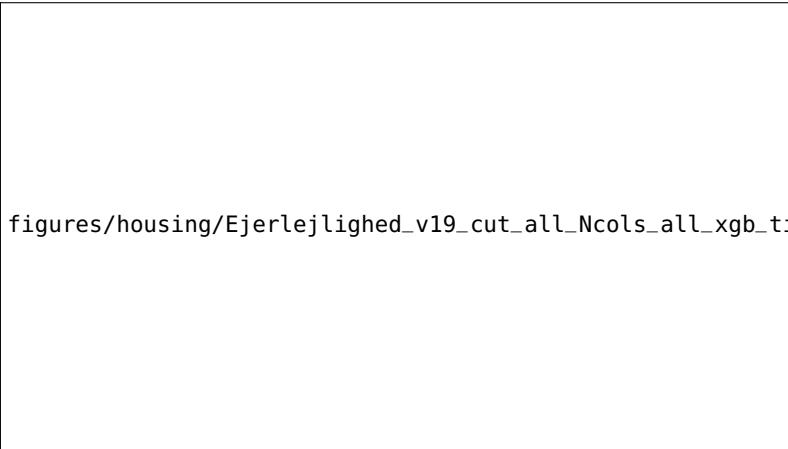


Figure 16.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.



figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf

Figure 16.24: Feature importance of apartment prices using the XGB-model. XXX

16.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same way as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optimized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

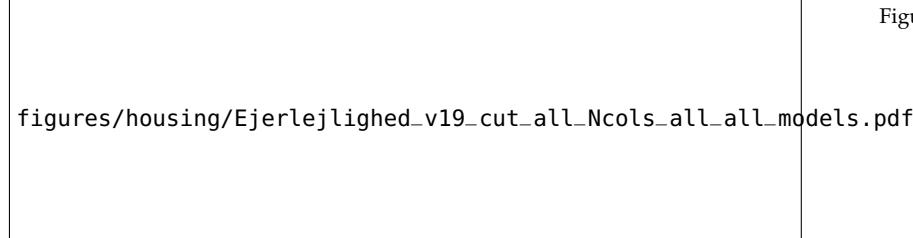


Figure 16.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (16.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (16.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

16.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

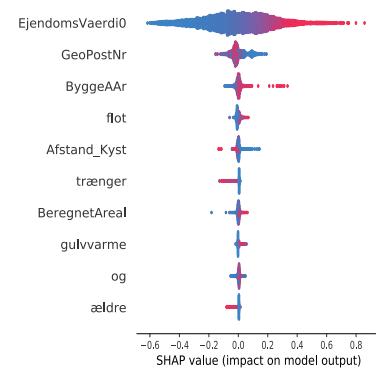


Figure 16.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
 $\arg\min$ 
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

17. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

18. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

19. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

19.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

19.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (19.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (19.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

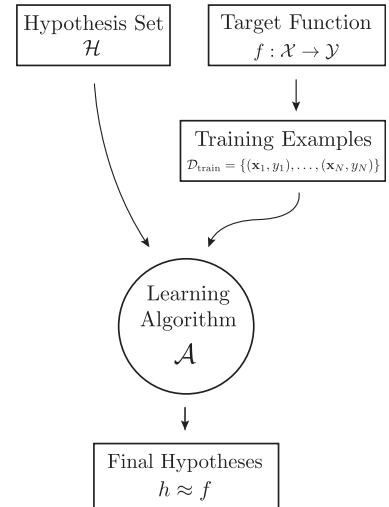


Figure 19.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (19.3)$$

19.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (19.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(\mathbf{x}, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 13 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (19.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 14 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (19.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (19.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 15 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (19.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (19.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (19.10)$$

Theorem 17 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (19.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (19.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 18 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (19.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (19.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

19.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: [?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 19 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (19.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (19.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

19.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

19.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (19.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

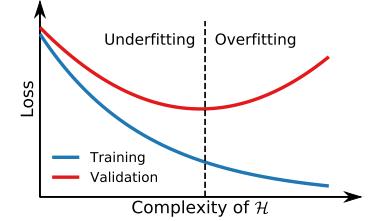


Figure 19.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (19.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (19.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (19.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (19.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \quad (19.22) \end{aligned}$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

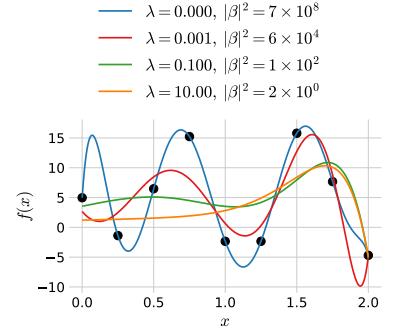


Figure 19.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (19.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by ?] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (19.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

19.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

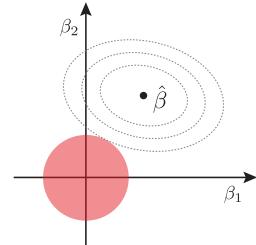


Figure 19.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

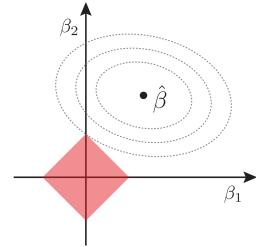


Figure 19.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

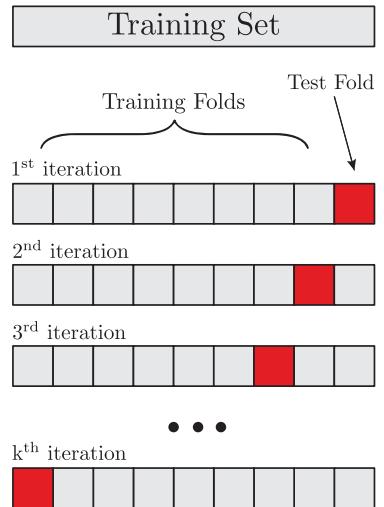


Figure 19.6: k -fold cross validation.

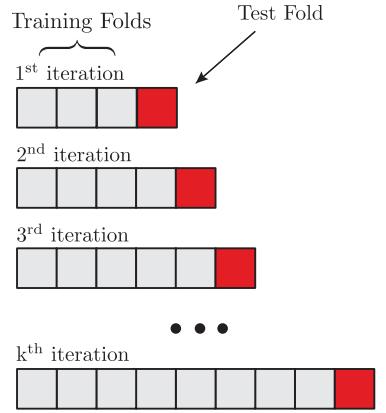


Figure 19.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

19.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

19.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (19.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (19.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (19.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

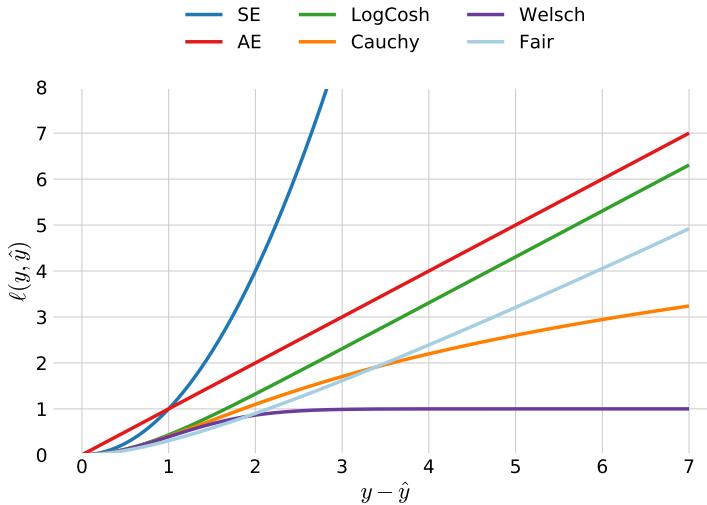


Figure 19.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

19.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (19.28)$$

19.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

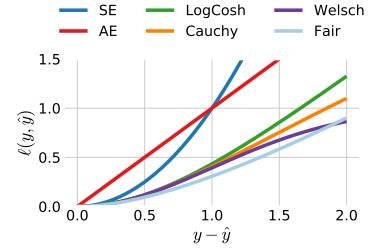


Figure 19.9: Zoom in of Figure ??.

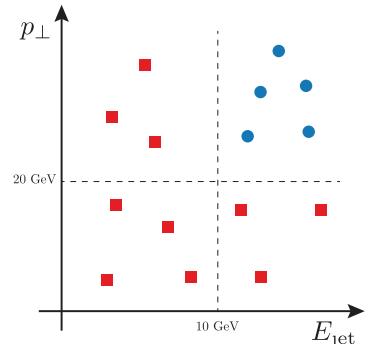


Figure 19.10: Illustration of the cuts a decision tree model make for signal in blue circles and background in red squares. This is an visualization in the feature space of the decision tree seen in Figure ??.

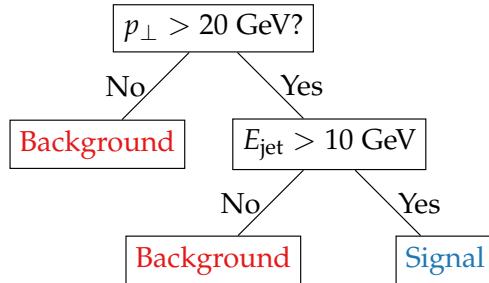


Figure 19.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

19.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (19.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 20 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (19.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (19.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (19.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (19.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (19.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (19.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

19.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

19.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (19.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

19.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (19.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

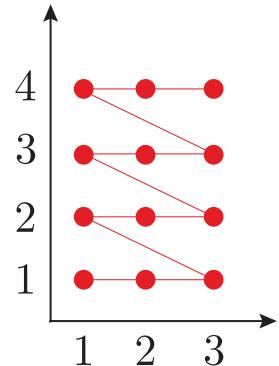


Figure 19.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (19.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

19.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

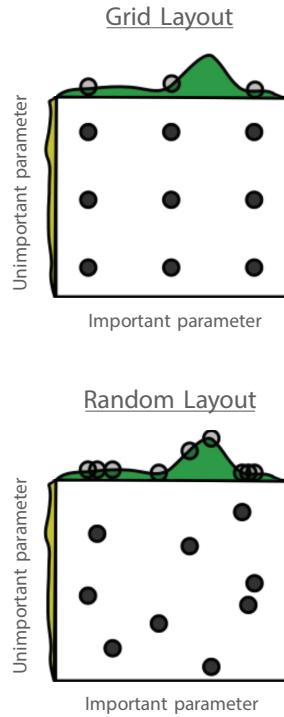


Figure 19.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the datadependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (19.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

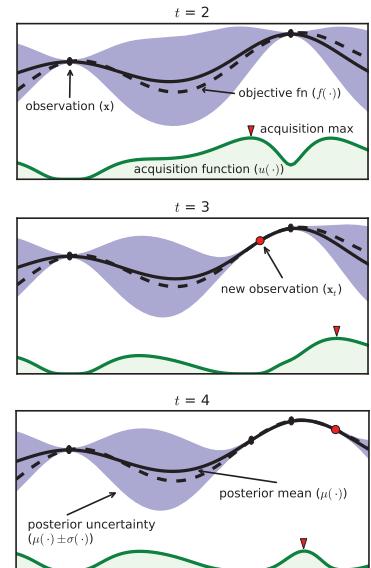


Figure 19.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

19.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (19.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 17 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (19.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 18 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (19.42)$$

Axiom 19 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (19.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (19.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (19.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (19.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

Axiom 20 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (19.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

⁴³ Not necessarily linearly correlated.

20. Danish Housing Prices

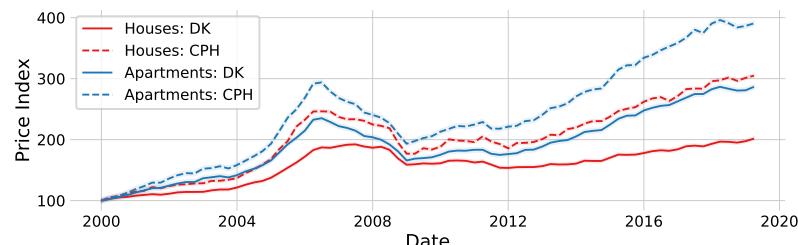
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 20.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation subproject.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

20.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

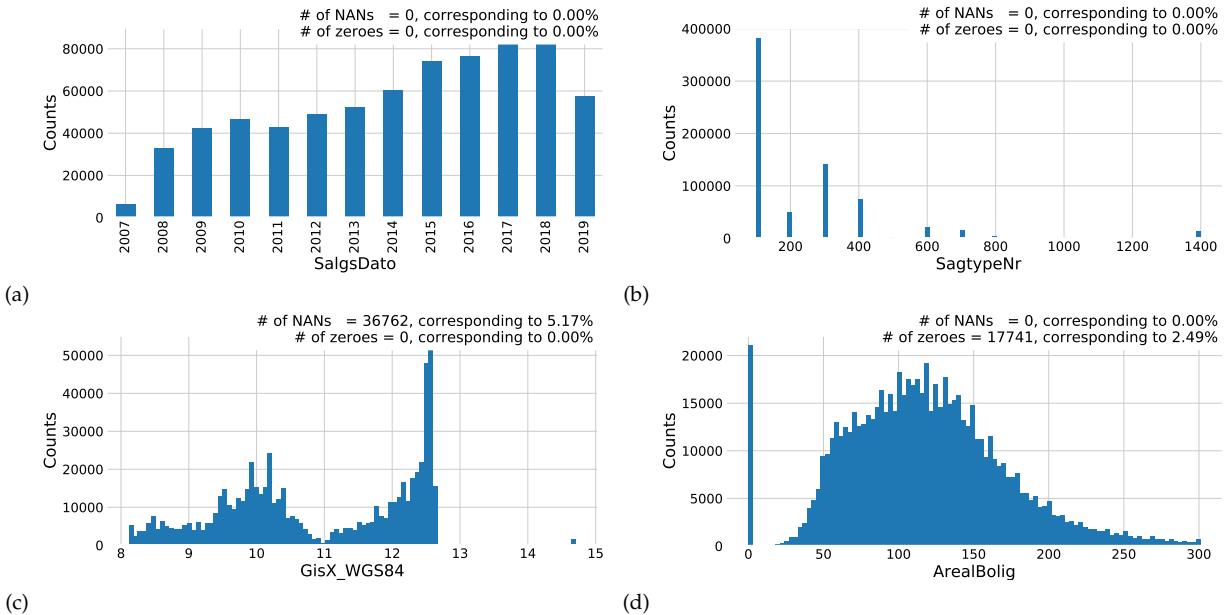
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

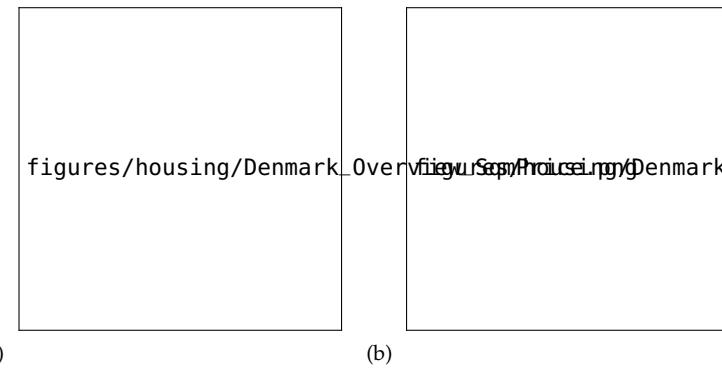
Figure 20.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehøus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 20.1: XXX TODO!

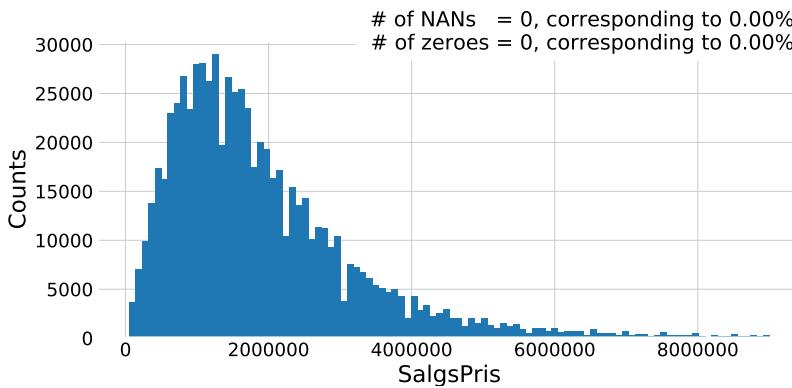
⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.



(a)

(b)



20.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

Figure 20.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

Figure 20.4: Histogram of prices of houses and apartments sold in Denmark.

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

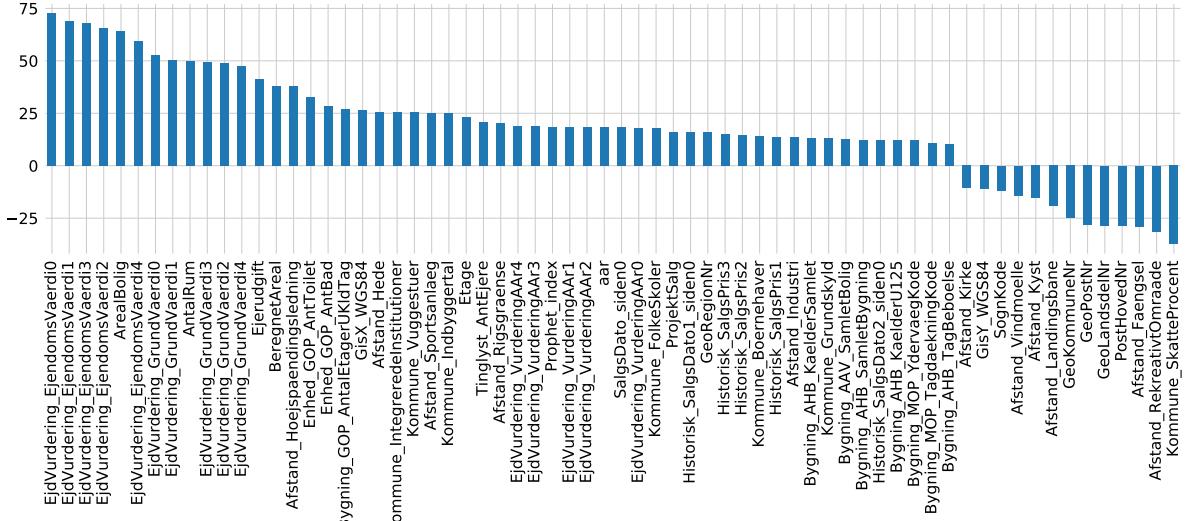


Figure 20.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. [?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

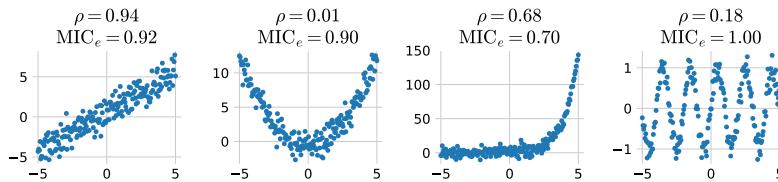


Figure 20.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

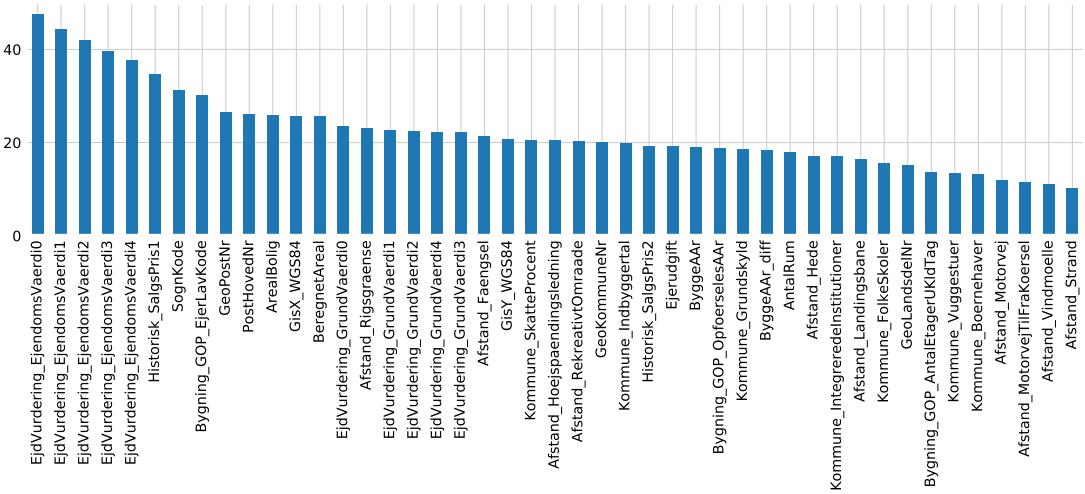
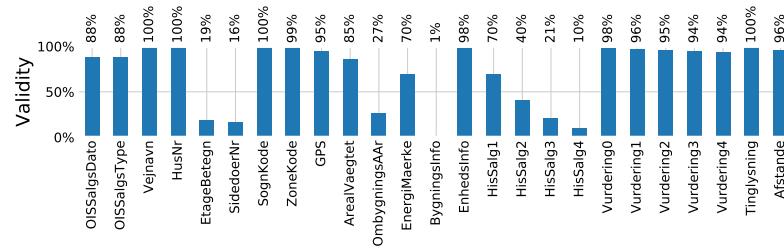


Figure 20.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where MIC > 10%.

20.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningsAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaerdingsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 20.8: Percentage of valid counts for each variable grouped together in categories.



Figure 20.9: Validity Dendrogram
TODO!

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

20.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 20.2: XXX TODO!

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 20.3: XXX TODO!

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 20.4: XXX TODO!

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ???. The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, `EnergiMaerke`, is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

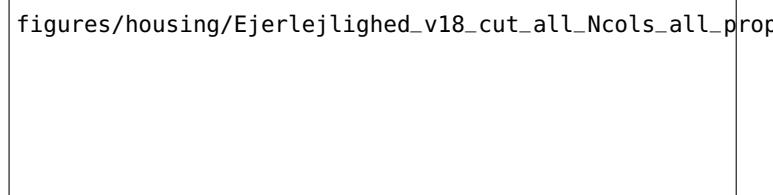
20.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by [] at Facebook. It is based on a decomposable time series model [] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (20.1)$$

where ϵ_t is a normally distributed error term. [] fit this equation with a generalized additive model (GAM) [] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ???. In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png`

¹¹ In addition to only having the year the house was built

¹² In their paper, [] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 20.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb`

Figure 20.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (20.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

20.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (20.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (20.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (20.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

20.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 20.6: train test split XXX **TODO!**

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 20.7: train test split tight XXX **TODO!**

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (20.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (20.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

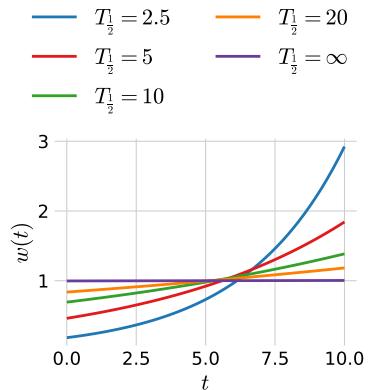


Figure 20.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

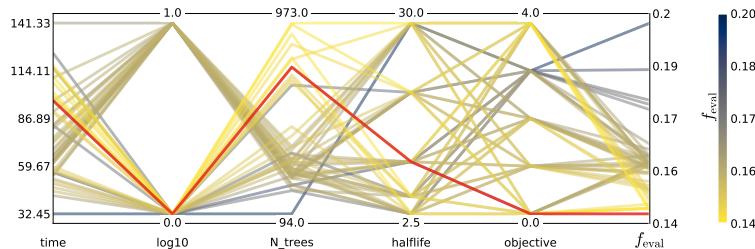
Table 20.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 20.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10 \text{ yr}$ is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

20.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 20.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD_0 in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The uncertainties here are the standard deviations (one standard deviation) and the 5-folds in the cross validation. The loss functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses.

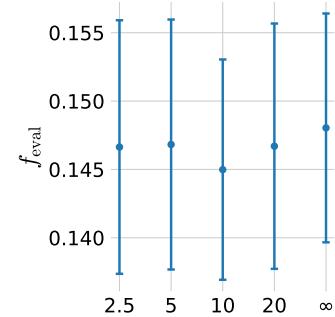


Figure 20.14: XXX Halflife $T_{\frac{1}{2}}$.

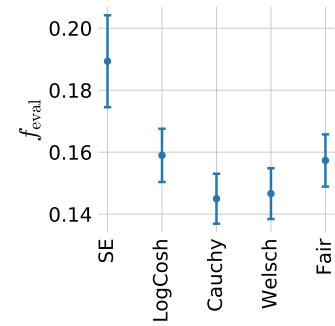


Figure 20.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 20.10: XXX

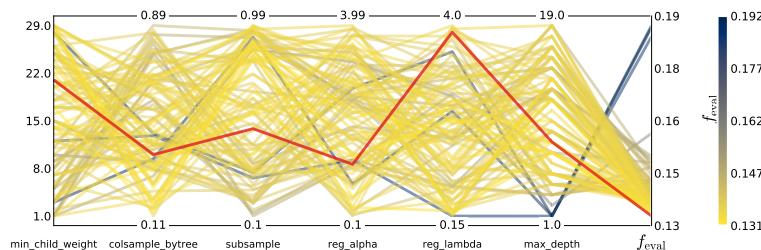


Figure 20.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

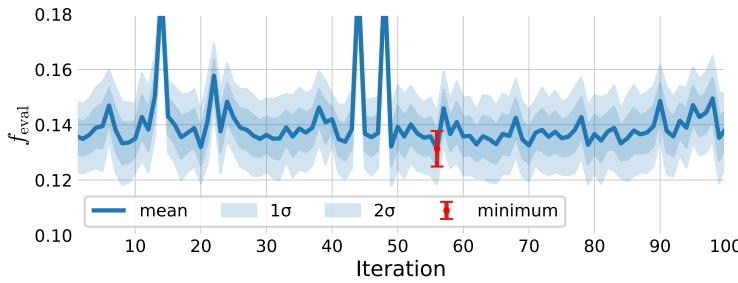


Figure 20.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

20.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 20.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

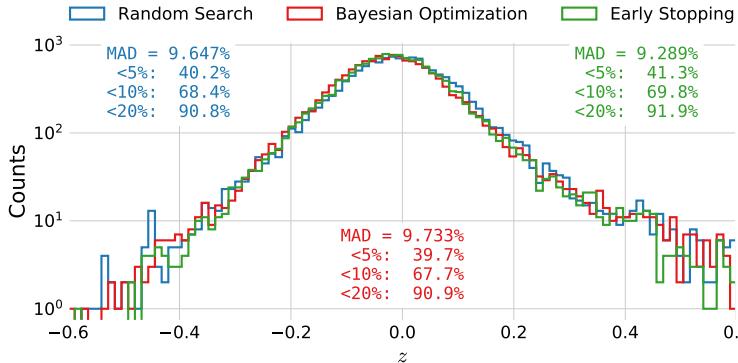


Figure 20.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

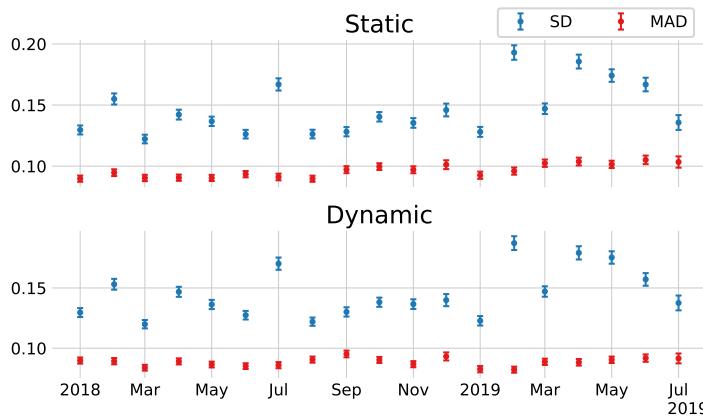


Figure 20.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month's sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (20.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (20.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predicitons are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

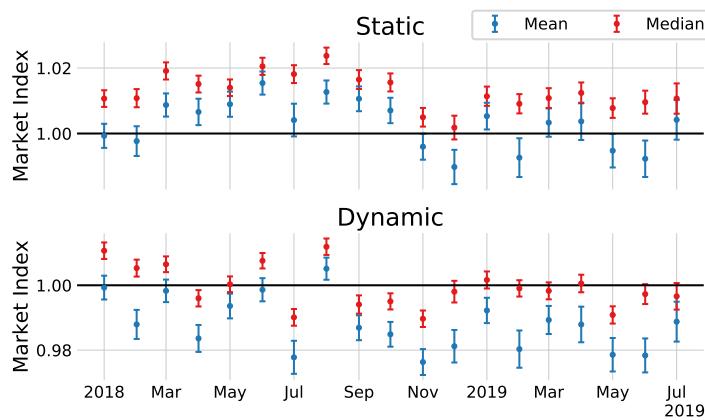


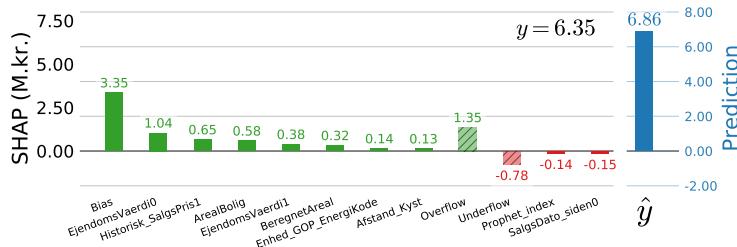
Figure 20.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z	Table 20.12: XXX ejer
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$	
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012	
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002	

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z	Table 20.13: XXX villa
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007	
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016	
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002	

20.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 20.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~On the right hand side of the plot~~ ~~the entire dataset~~ model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

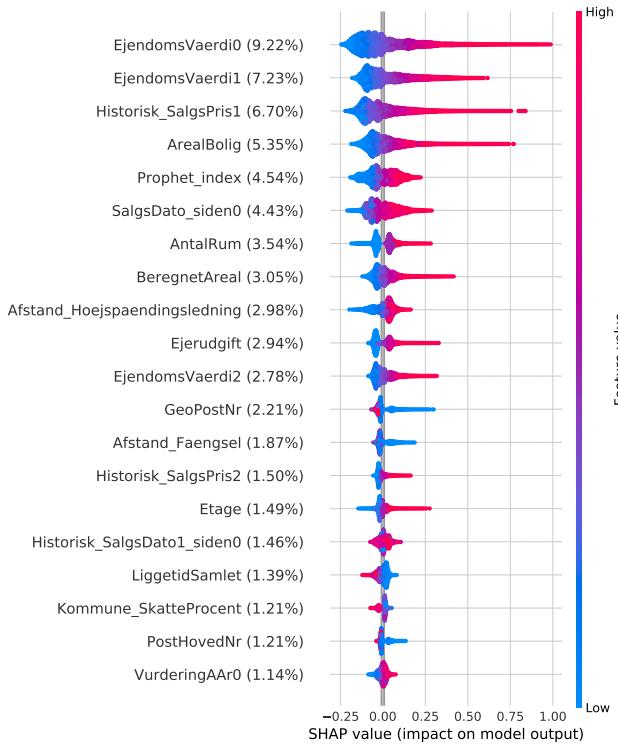


Figure 20.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.

`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 20.24: Feature importance of apartment prices using the XGB-model. XXX

20.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

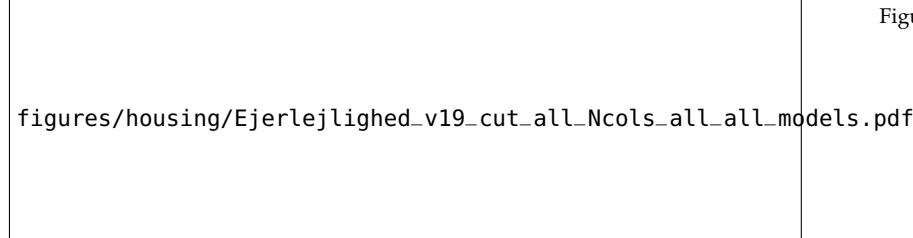


Figure 20.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (20.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (20.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

20.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

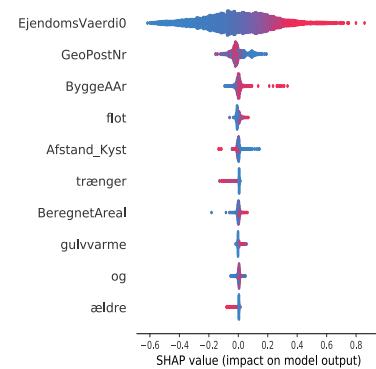


Figure 20.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
 $\arg\min$ 
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

21. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

22. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

23. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

23.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

23.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (23.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (23.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

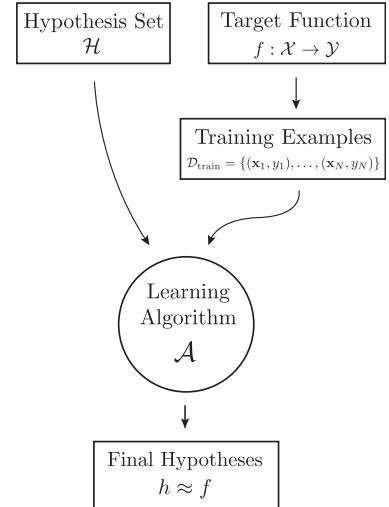


Figure 23.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (23.3)$$

23.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (23.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(x, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 16 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (23.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 17 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (23.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (23.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 18 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (23.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (23.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (23.10)$$

Theorem 21 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (23.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (23.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 22 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (23.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (23.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

23.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: [?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 23 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (23.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (23.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

23.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

23.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (23.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

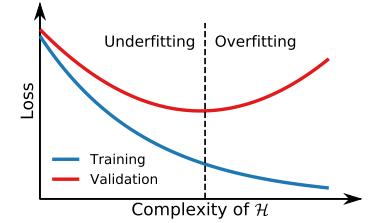


Figure 23.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (23.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (23.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (23.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (23.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \quad (23.22) \end{aligned}$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

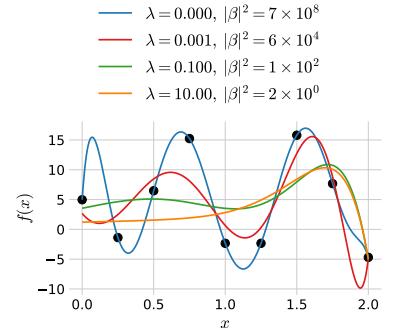


Figure 23.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (23.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by [1] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (23.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

23.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

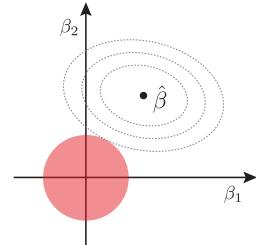


Figure 23.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

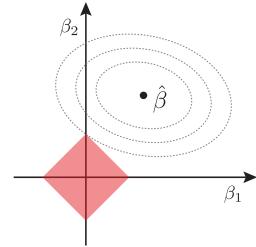


Figure 23.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

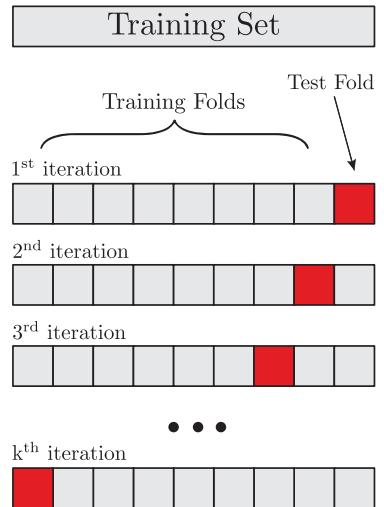


Figure 23.6: k -fold cross validation.

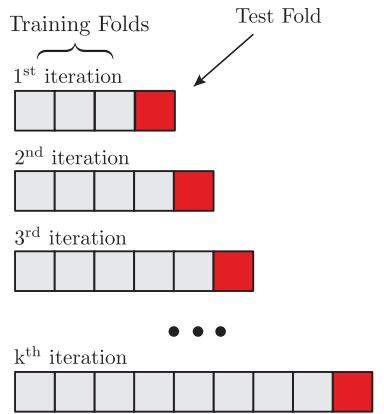


Figure 23.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

23.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

23.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (23.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (23.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (23.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

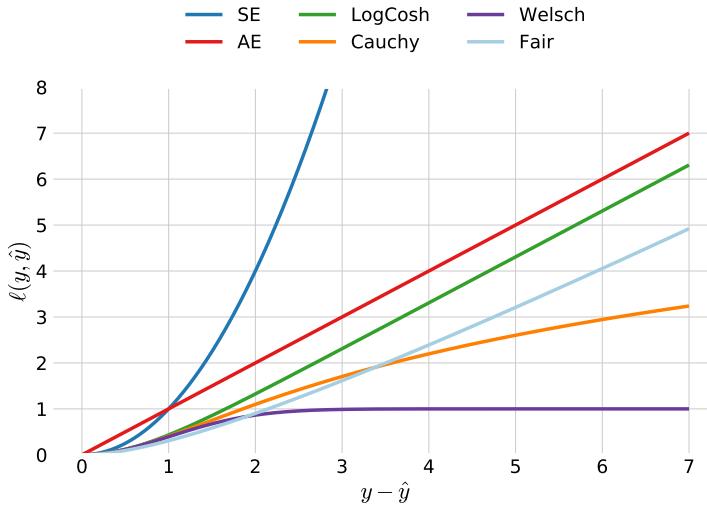


Figure 23.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

23.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (23.28)$$

23.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

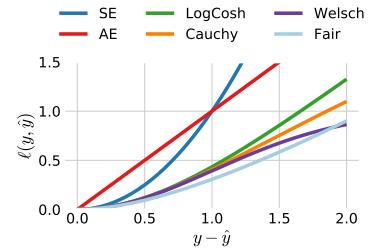


Figure 23.9: Zoom in of Figure ??.

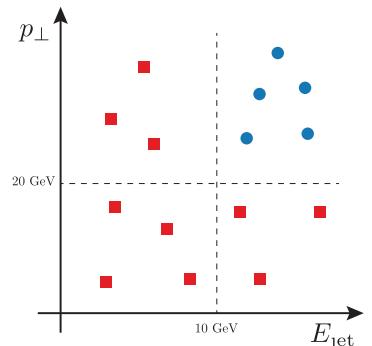


Figure 23.10: Illustration of the cuts a decision tree model make for *signal* in blue circles and *background* in red squares. This is an visualization in the feature space of the decision tree seen in Figure ??.

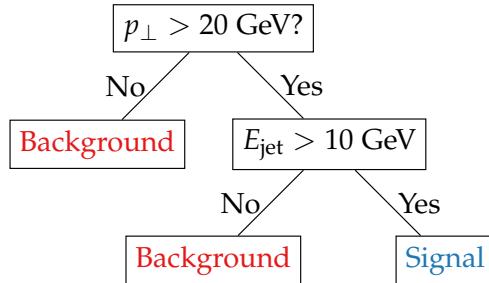


Figure 23.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

23.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (23.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 24 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (23.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (23.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (23.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (23.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (23.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (23.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

23.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

23.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (23.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

23.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (23.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

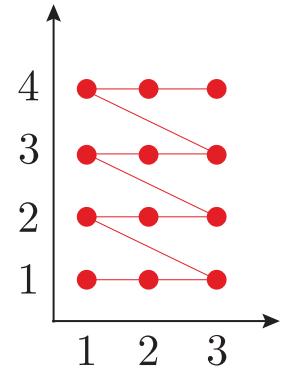


Figure 23.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (23.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

23.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

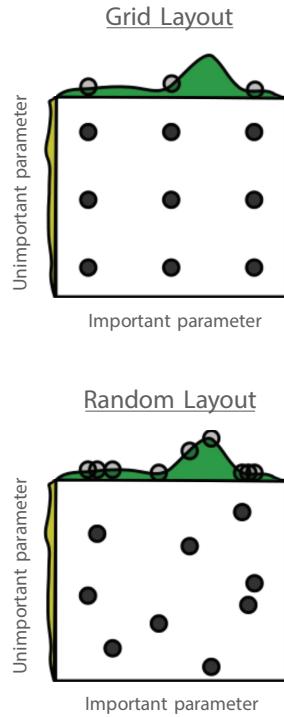


Figure 23.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the data-dependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (23.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

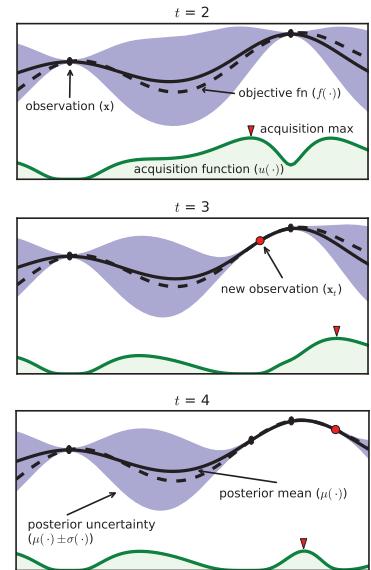


Figure 23.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

23.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (23.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 21 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (23.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 22 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (23.42)$$

Axiom 23 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (23.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (23.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (23.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (23.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

⁴³ Not necessarily linearly correlated.

Axiom 24 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (23.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

24. Danish Housing Prices

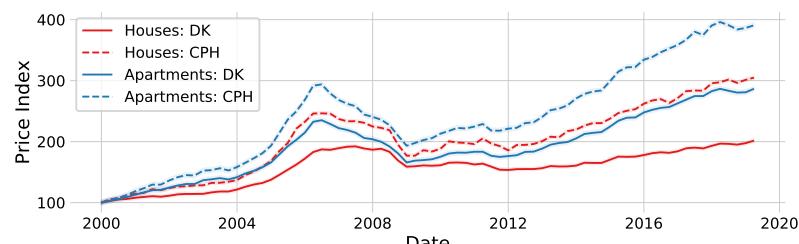
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 24.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

24.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

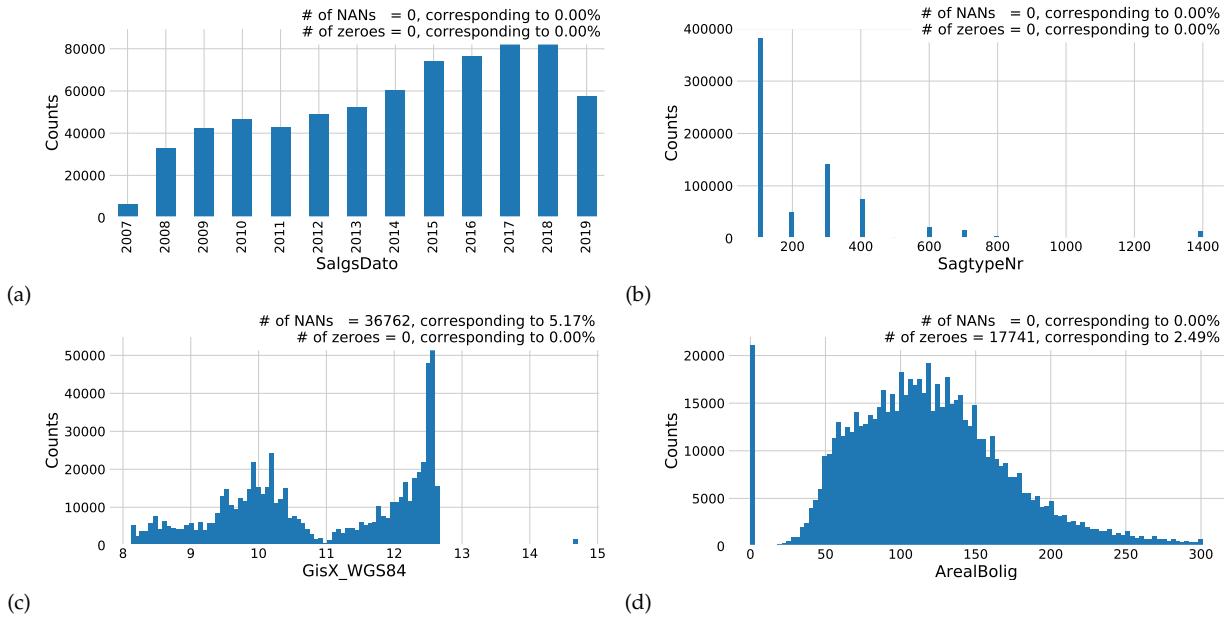
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 24.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehøus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 24.1: XXX TODO!

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

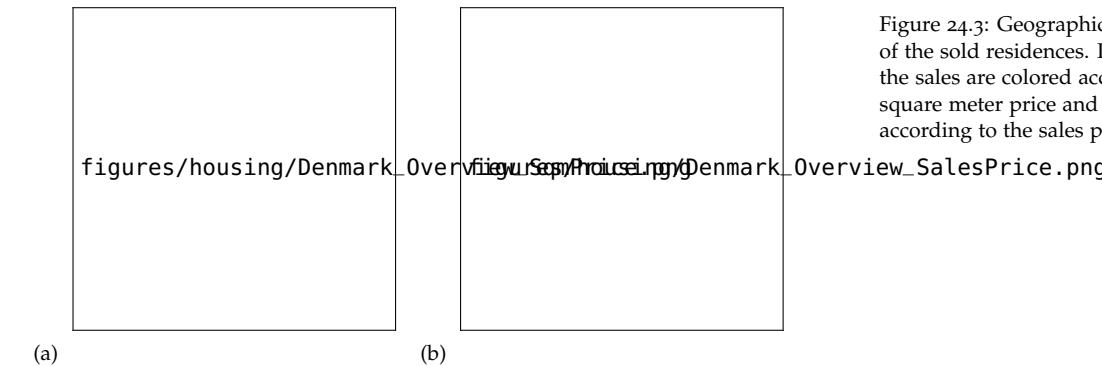


Figure 24.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

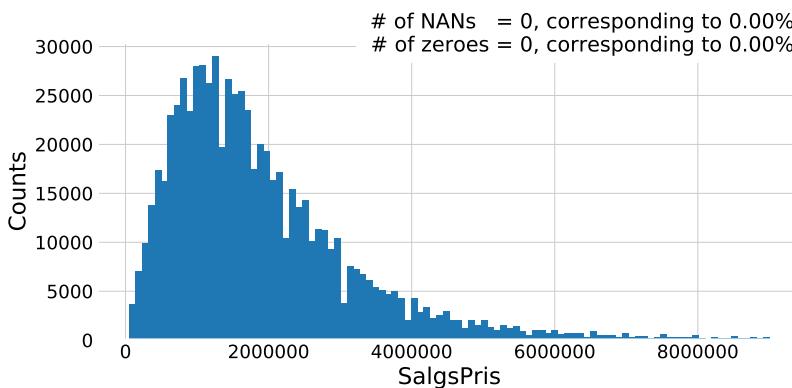


Figure 24.4: Histogram of prices of houses and apartments sold in Denmark.

24.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

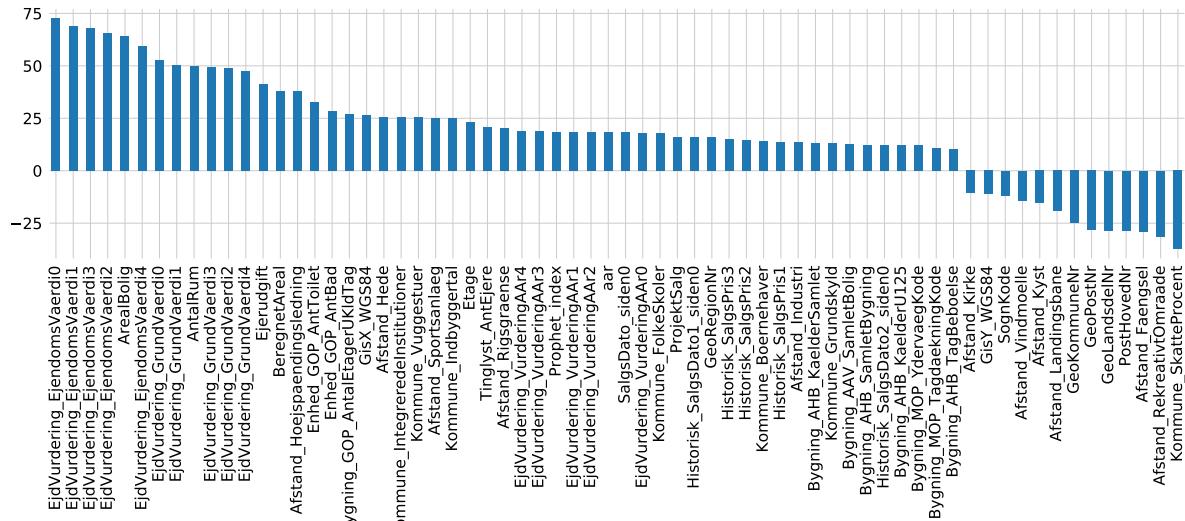


Figure 24.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. ?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

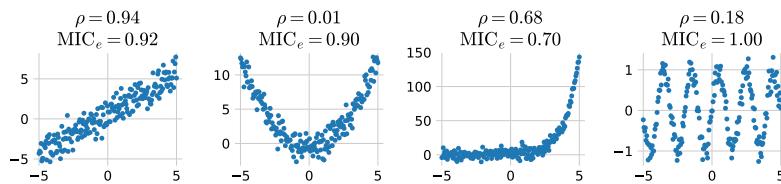


Figure 24.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SognKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

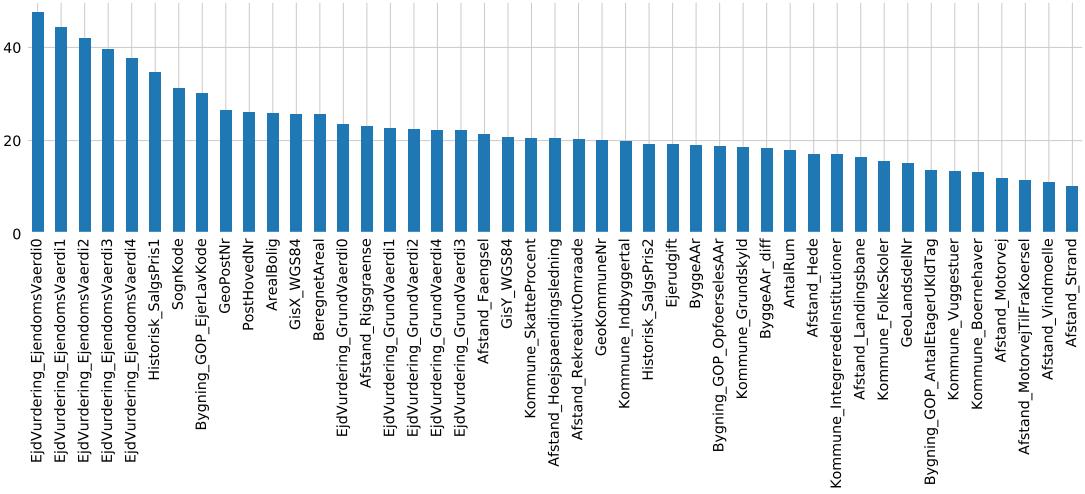
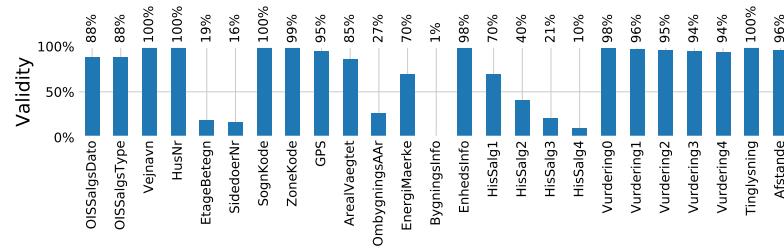


Figure 24.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where $\text{MIC} > 10\%$.

24.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaedningsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 24.8: Percentage of valid counts for each variable grouped together in categories.



Figure 24.9: Validity Dendrogram
TODO!

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

24.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 24.2: XXX TODO!

24.2 Feature Augmentation

Until now all of the analysis have dealt with different types of residences all together. From now on, the rest of the analysis will be applied on single-family houses and owner-occupied apartments independently.

First invalid counts are dropped, such that variables which contain more than 10 % NaNs are dropped, and duplicate rows are also removed. Then some manual features are added based on the day of the month, the month, and the year are extracted from the sales date and the sales date is also represented as both a float and as the numbers of days since January 1st, 2009. From the number of the house, `HusNr`, the number is extracted along with a boolean flag indicating whether or not it includes a letter (eg. “27B”). The number of the side door, `SidedoerNr`, is formatted according to

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 24.3: XXX TODO!

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 24.4: XXX TODO!

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ?? . The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, EnergiMaerke , is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

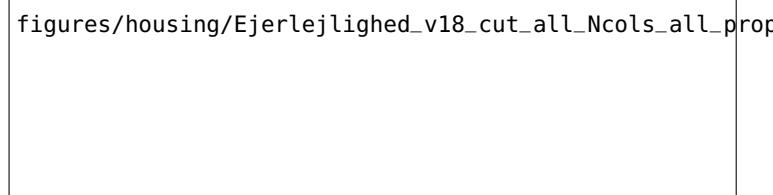
24.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by ?] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (24.1)$$

where ϵ_t is a normally distributed error term. ?] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ?? . In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png

¹¹ In addition to only having the year the house was built

¹² In their paper, ?] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 24.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb`

Figure 24.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (24.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

24.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (24.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (24.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (24.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

24.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 24.6: train test split XXX **TODO!**

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 24.7: train test split tight XXX **TODO!**

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (24.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (24.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

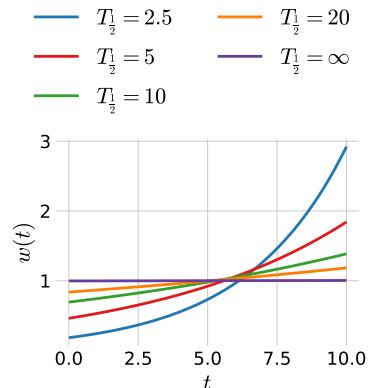


Figure 24.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

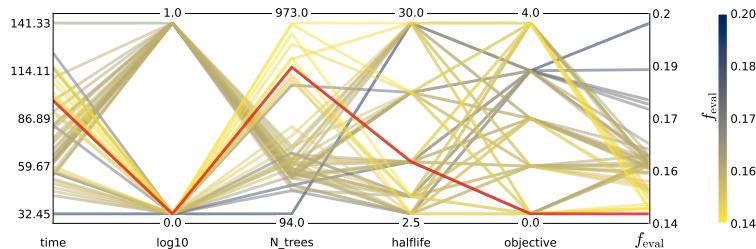
Table 24.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 24.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

24.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 24.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The standard deviations (and MAD_0) and the 5-folds in the cross validation's `Cauchy` (0), `Fair` (1), `LogCosh` (2) `SquaredError` (3), and `Welsch` (4) are mapped to the integers in the parentheses.

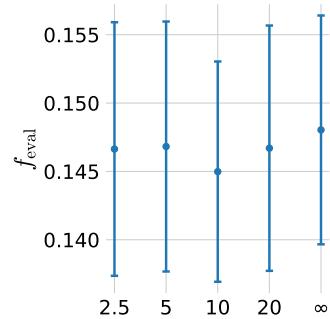


Figure 24.14: XXX Halflife $T_{\frac{1}{2}}$.

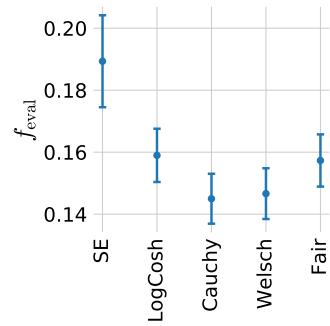


Figure 24.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 24.10: XXX

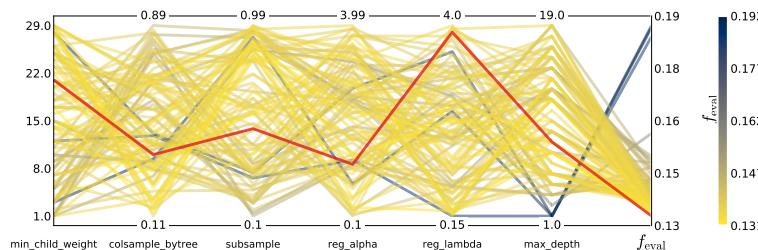


Figure 24.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

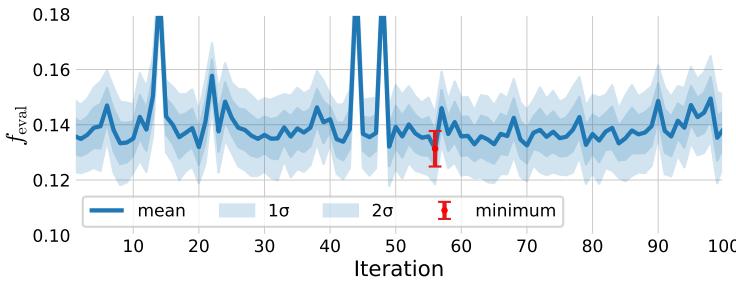


Figure 24.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

24.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 24.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

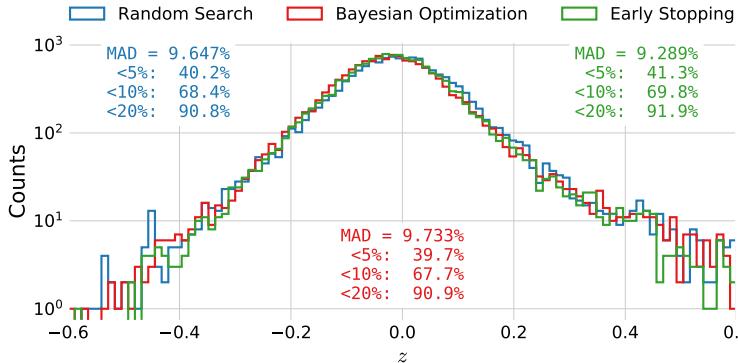


Figure 24.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

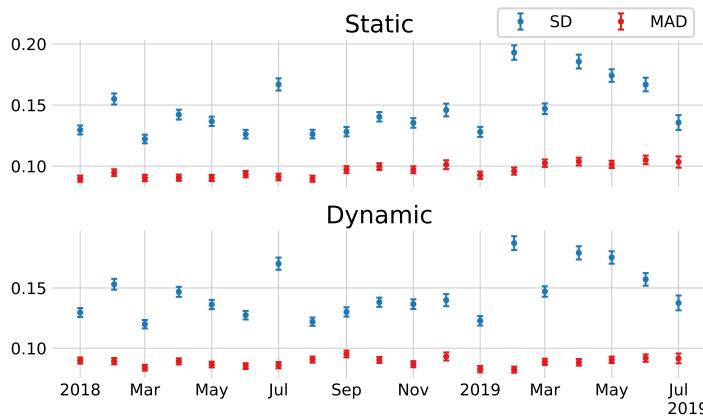


Figure 24.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month's sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (24.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (24.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predictions are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

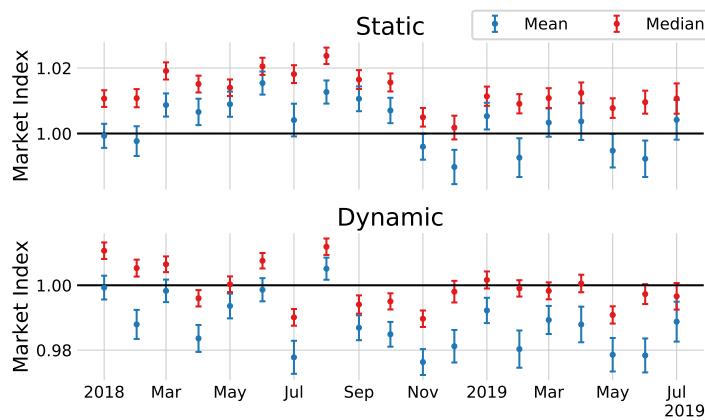


Figure 24.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002

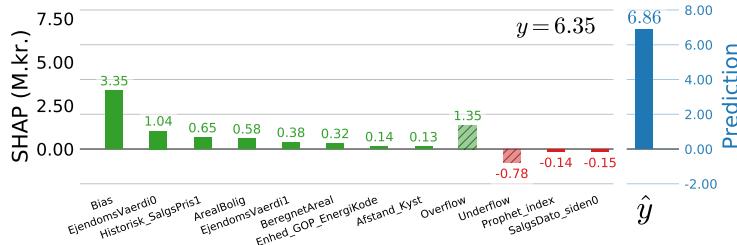
Table 24.12: XXX ejer

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002

Table 24.13: XXX villa

24.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 24.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~On the right hand side of the plot the model prediction is shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.~~

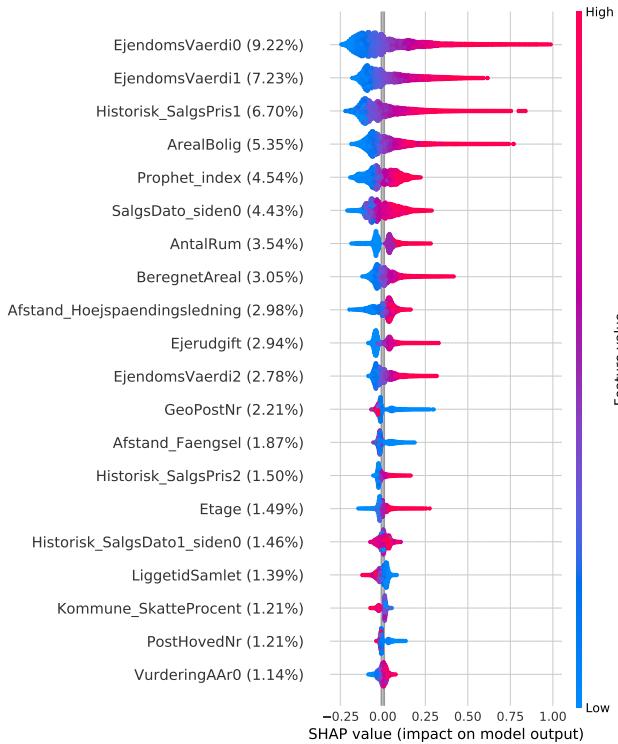


Figure 24.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.

`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 24.24: Feature importance of apartment prices using the XGB-model. XXX

24.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

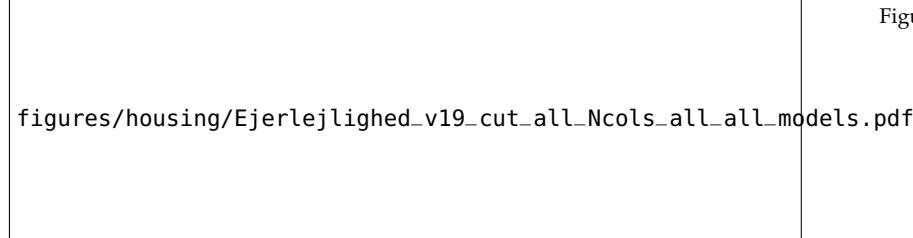


Figure 24.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (24.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (24.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

24.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

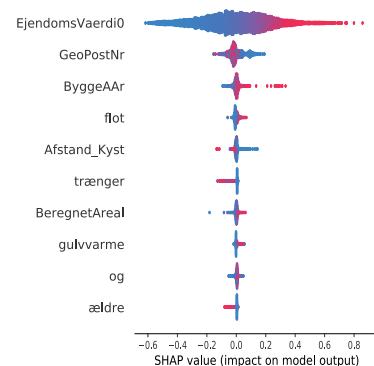


Figure 24.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
*arg min arg min
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

25. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

26. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

27. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

27.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

27.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (27.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (27.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

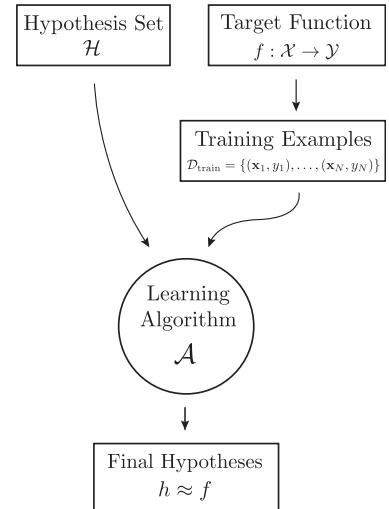


Figure 27.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (27.3)$$

27.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (27.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(\mathbf{x}, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 19 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (27.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 20 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (27.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (27.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 21 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (27.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (27.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (27.10)$$

Theorem 25 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (27.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (27.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 26 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (27.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (27.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

27.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: ?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 27 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (27.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (27.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

27.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

27.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (27.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

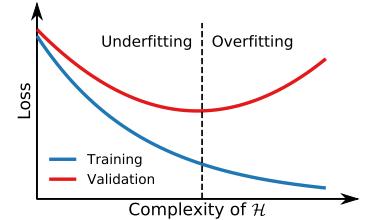


Figure 27.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (27.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (27.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (27.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (27.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \quad (27.22) \end{aligned}$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

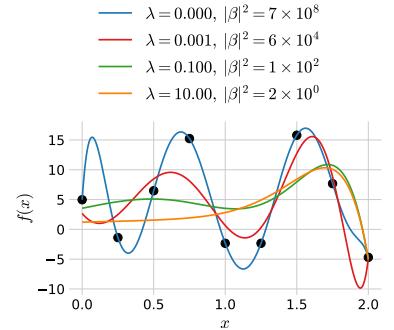


Figure 27.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (27.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by ?] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (27.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

27.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

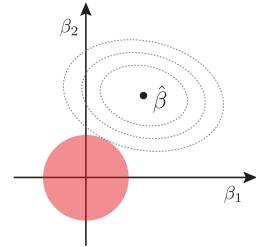


Figure 27.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

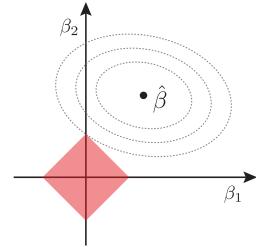


Figure 27.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

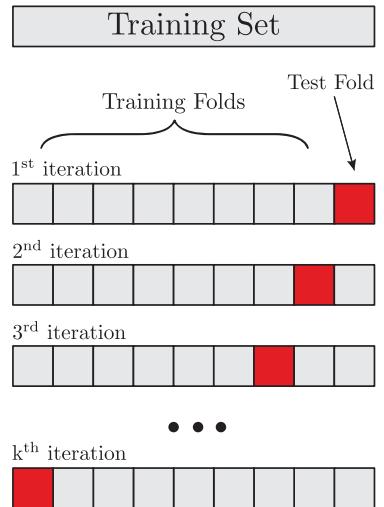


Figure 27.6: k -fold cross validation.

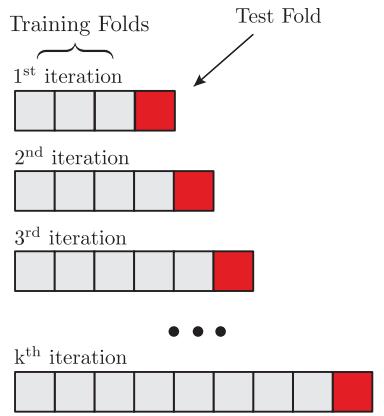


Figure 27.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

27.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

27.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (27.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (27.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (27.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

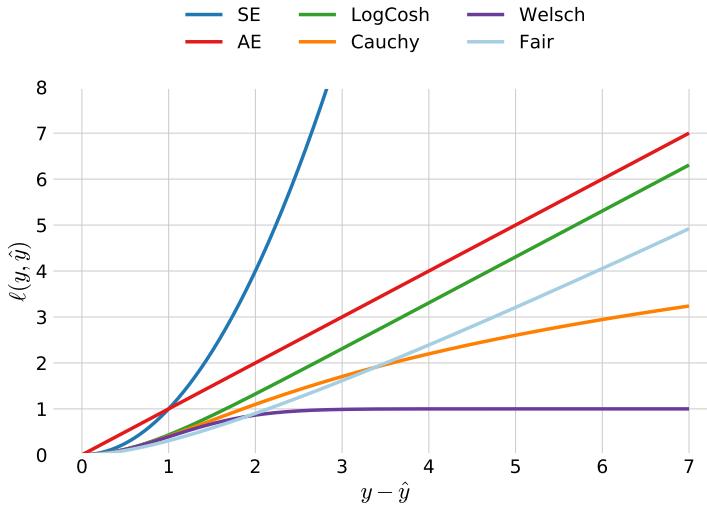


Figure 27.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

27.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (27.28)$$

27.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

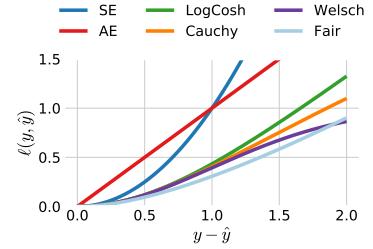


Figure 27.9: Zoom in of Figure ??.

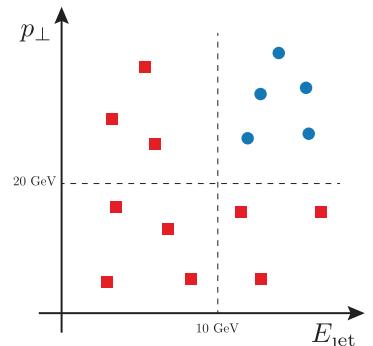


Figure 27.10: Illustration of the cuts a decision tree model make for signal in blue circles and background in red squares. This is a visualization in the feature space of the decision tree seen in Figure ??.

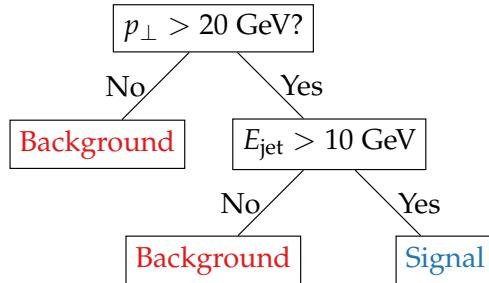


Figure 27.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

27.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (27.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 28 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (27.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (27.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (27.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (27.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (27.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (27.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

27.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO.)

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

27.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (27.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

27.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (27.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

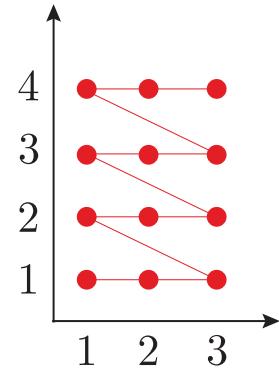


Figure 27.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (27.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

27.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

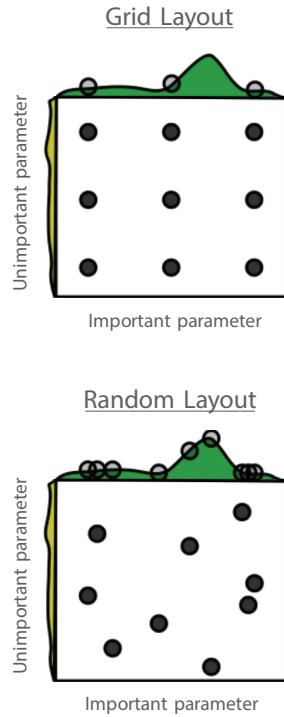


Figure 27.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the datadependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (27.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

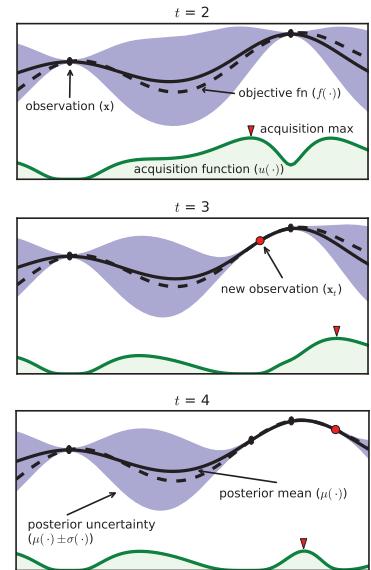


Figure 27.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

27.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (27.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 25 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (27.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 26 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (27.42)$$

Axiom 27 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (27.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (27.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (27.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (27.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

Axiom 28 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (27.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

⁴³ Not necessarily linearly correlated.

28. Danish Housing Prices

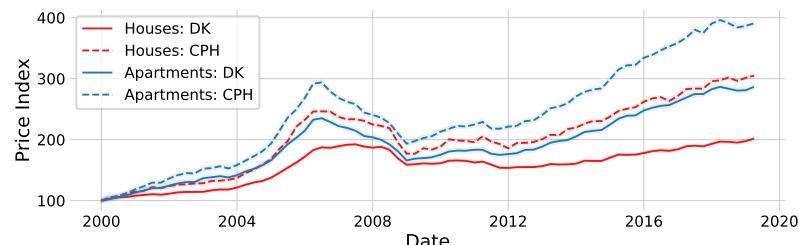
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 28.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

28.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

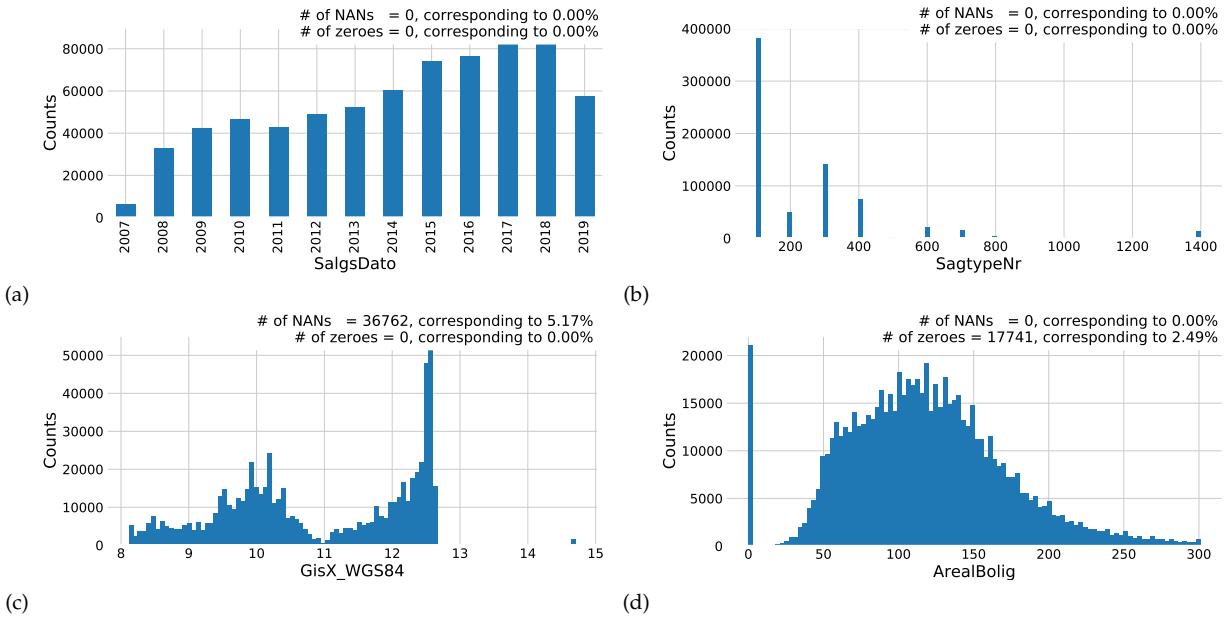
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 28.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehøus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 28.1: XXX TODO!

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

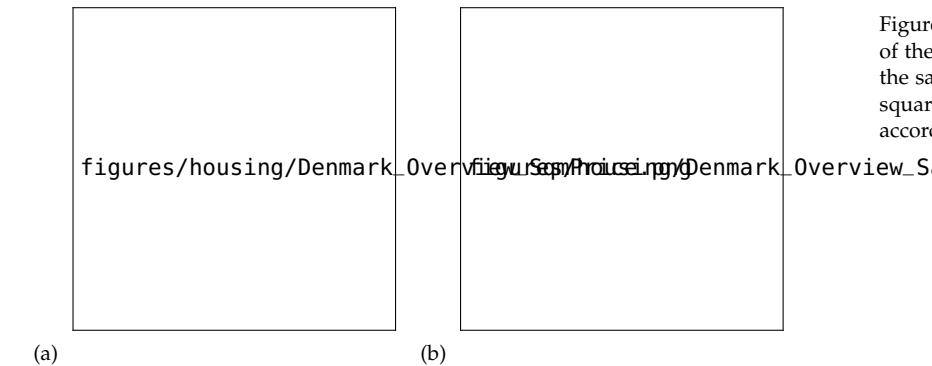


Figure 28.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

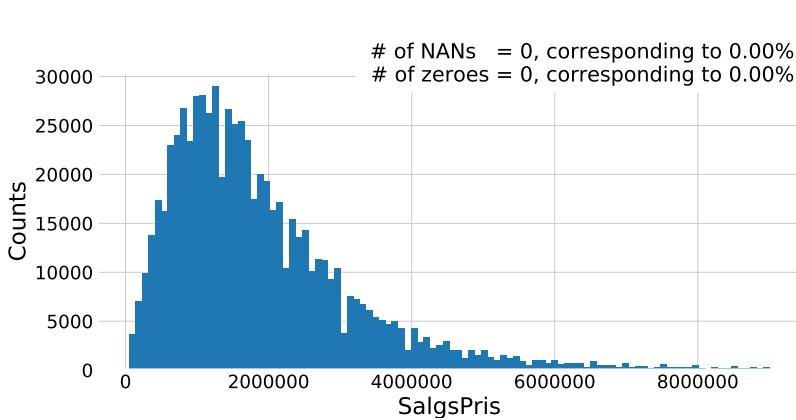


Figure 28.4: Histogram of prices of houses and apartments sold in Denmark.

28.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

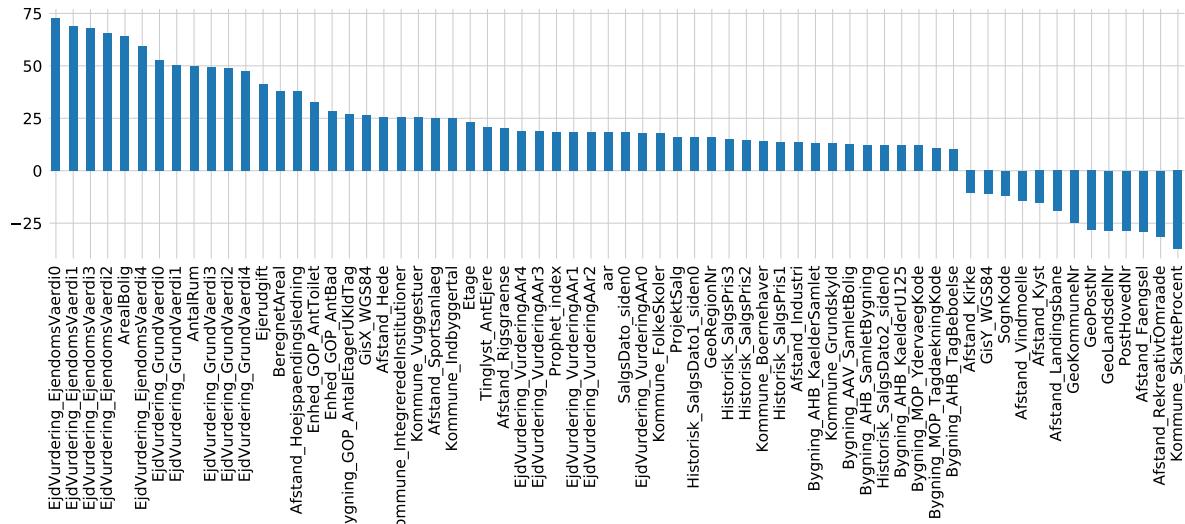


Figure 28.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. [?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

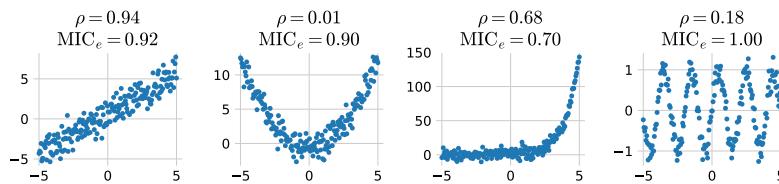


Figure 28.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

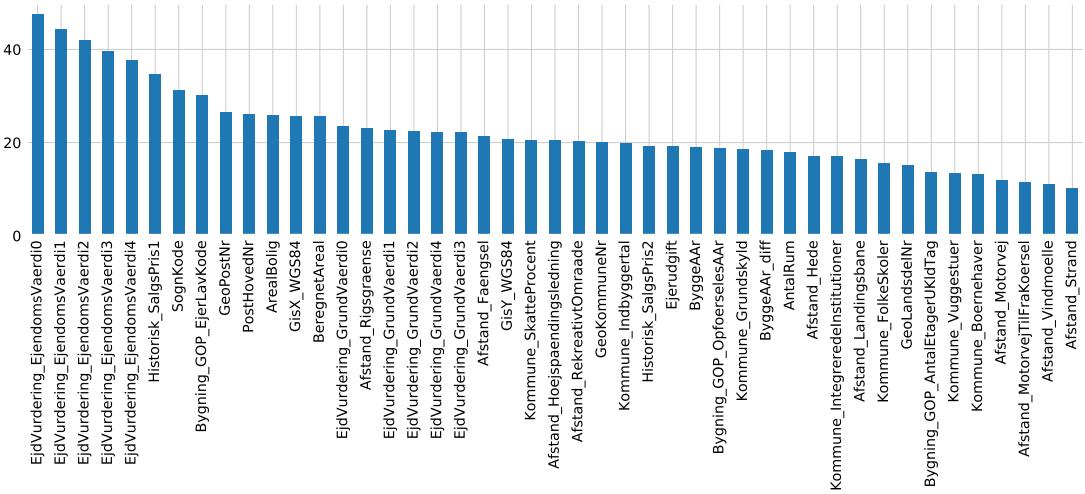
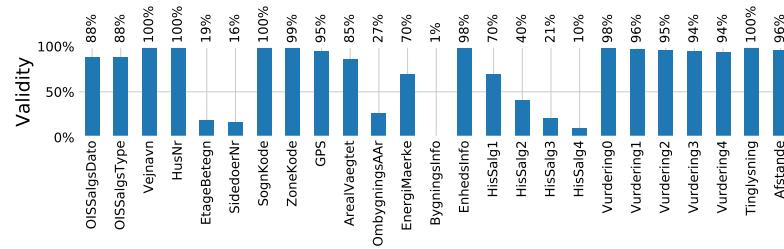


Figure 28.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where $\text{MIC} > 10\%$.

28.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaedningsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 28.8: Percentage of valid counts for each variable grouped together in categories.

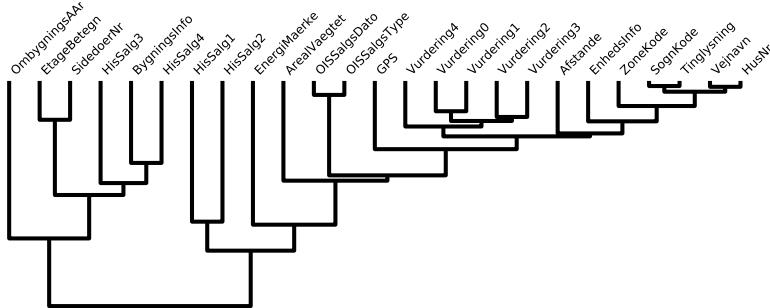


Figure 28.9: Validity Dendrogram
TODO!

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

28.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 28.2: XXX TODO!

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 28.3: XXX TODO!

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 28.4: XXX TODO!

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ?? . The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, EnergiMaerke , is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

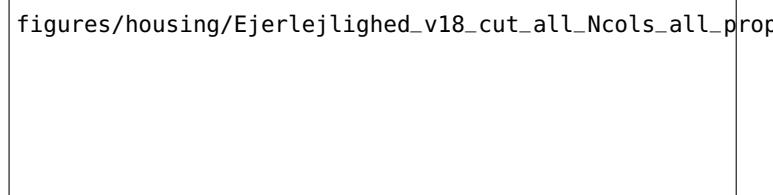
28.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by ?] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (28.1)$$

where ϵ_t is a normally distributed error term. ?] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ?? . In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png

¹¹ In addition to only having the year the house was built

¹² In their paper, ?] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 28.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

```
figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb
```

Figure 28.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (28.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

28.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (28.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (28.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (28.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

28.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 28.6: train test split XXX **TODO!**

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 28.7: train test split tight XXX **TODO!**

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (28.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (28.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??–?? in the appendix along with all of results for the houses in Table ??–??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

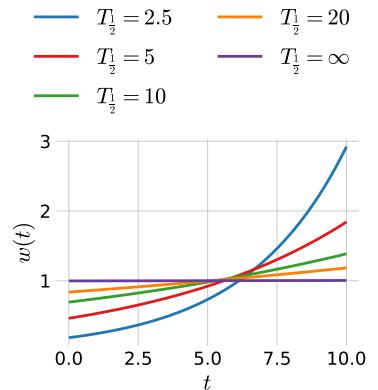


Figure 28.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

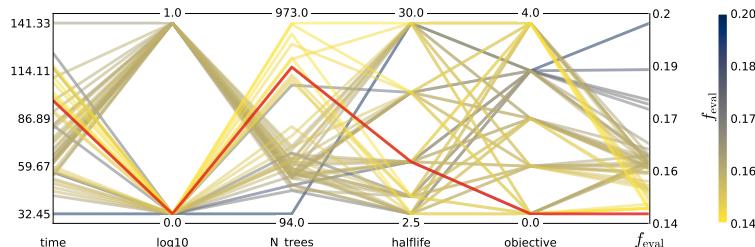
Table 28.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 28.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

28.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 28.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD_0 in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The standard deviations (and propagation) and the 5-folds in the cross validation's `Cauchy` (0), `Fair` (1), `LogCosh` (2) `SquaredError` (3), and `Welsch` (4) are mapped to the integers in the parentheses.

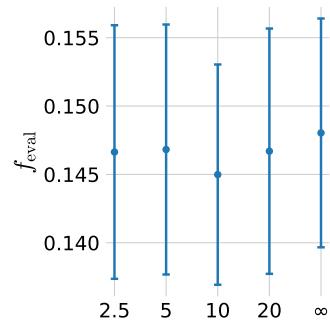


Figure 28.14: XXX Halflife $T_{\frac{1}{2}}$.

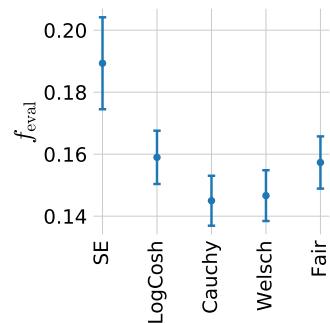


Figure 28.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 28.10: XXX

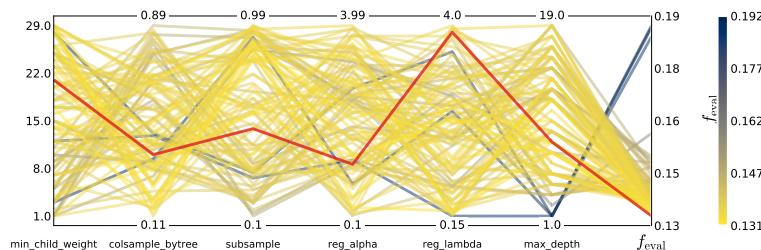


Figure 28.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

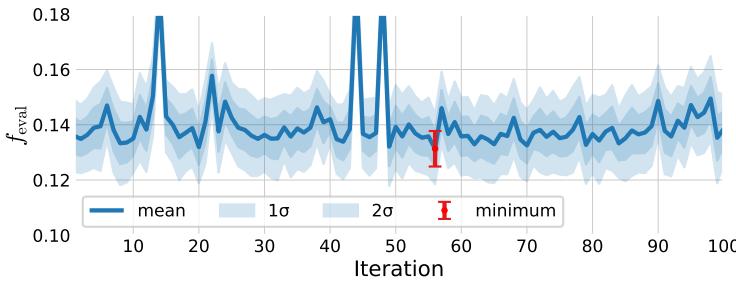


Figure 28.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

28.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 28.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

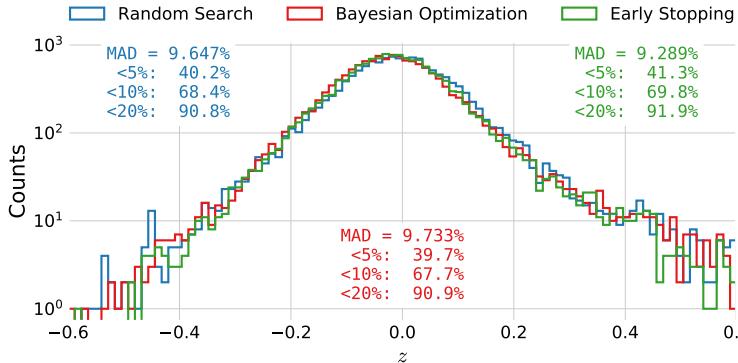


Figure 28.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

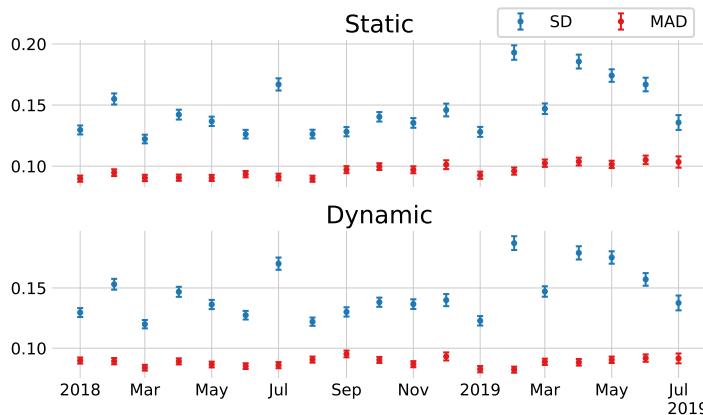


Figure 28.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month's sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (28.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (28.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predicitons are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

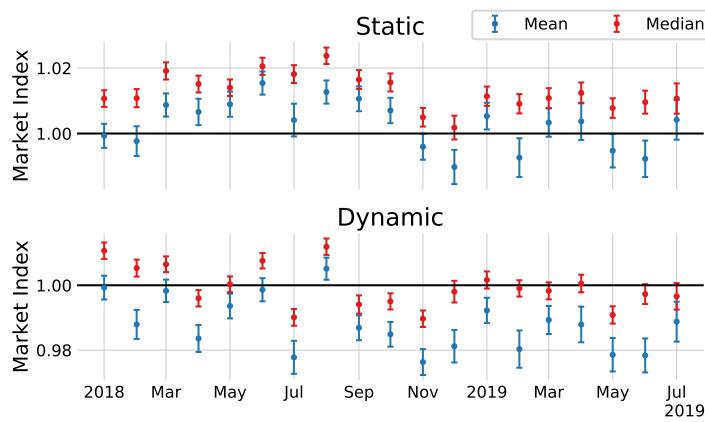


Figure 28.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002

Table 28.12: XXX ejer

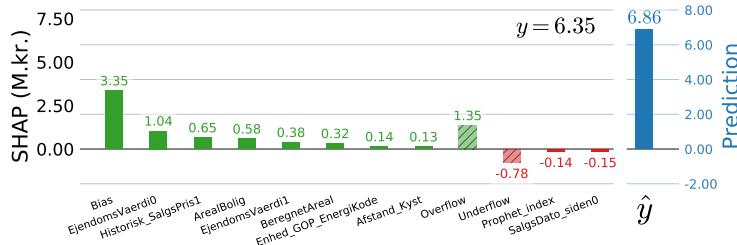
	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002

Table 28.13: XXX villa

28.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ??.

Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 28.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~On the right hand side of the plot the entire dataset model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.~~

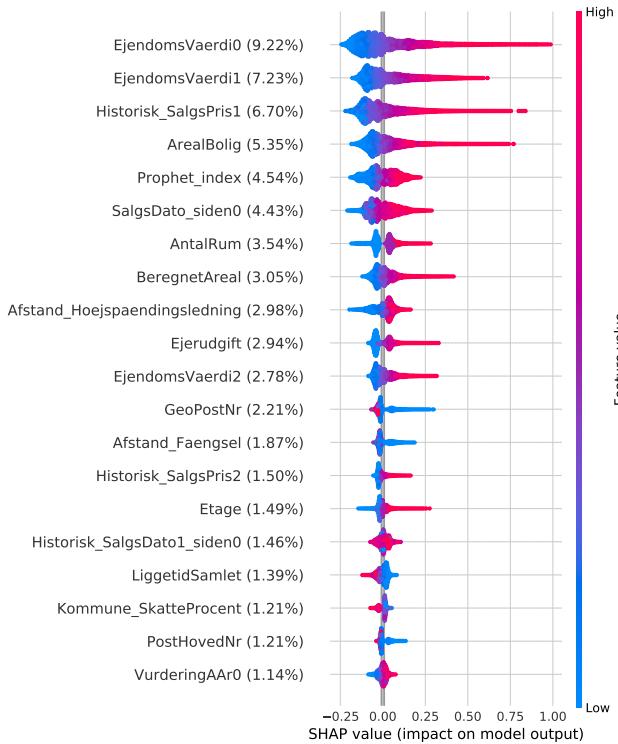


Figure 28.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.

`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 28.24: Feature importance of apartment prices using the XGB-model. XXX

28.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

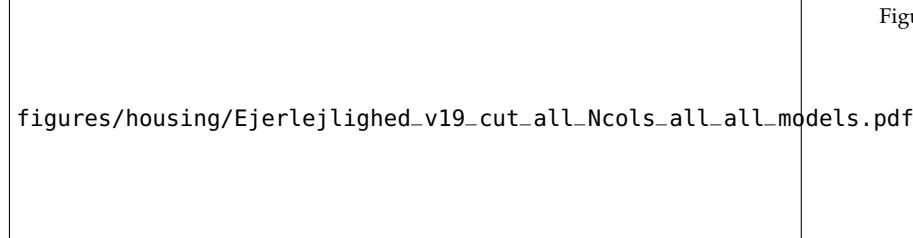


Figure 28.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (28.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (28.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

28.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

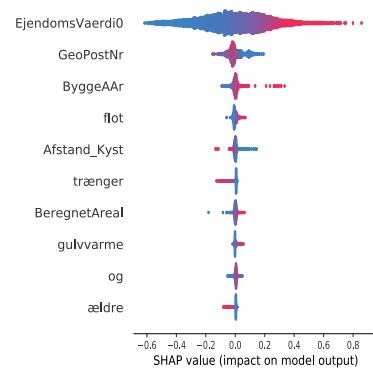


Figure 28.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
 $\arg\min$ 
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

29. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

30. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

31. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

31.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

31.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (31.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (31.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

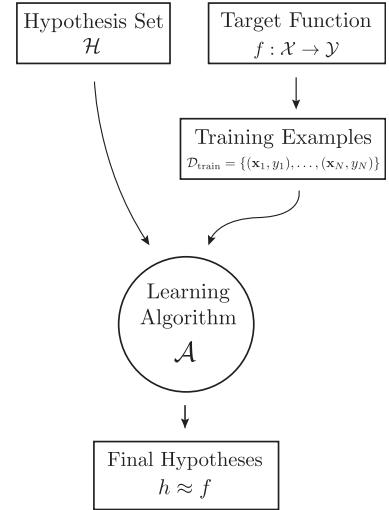


Figure 31.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (31.3)$$

31.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (31.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(x, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 22 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (31.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 23 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (31.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (31.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 24 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (31.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (31.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (31.10)$$

Theorem 29 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (31.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (31.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 30 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (31.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (31.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

31.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: [?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 31 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (31.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (31.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

31.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

31.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (31.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

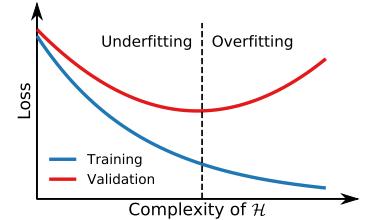


Figure 31.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (31.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (31.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (31.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (31.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \quad (31.22) \end{aligned}$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

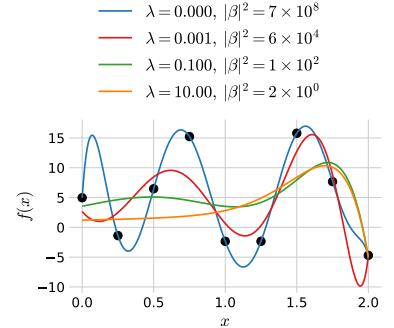


Figure 31.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (31.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by ?] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (31.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

31.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

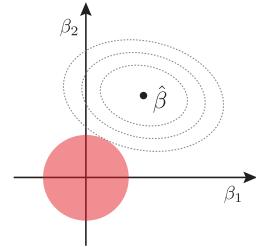


Figure 31.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

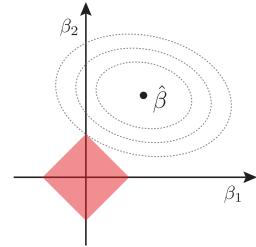


Figure 31.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

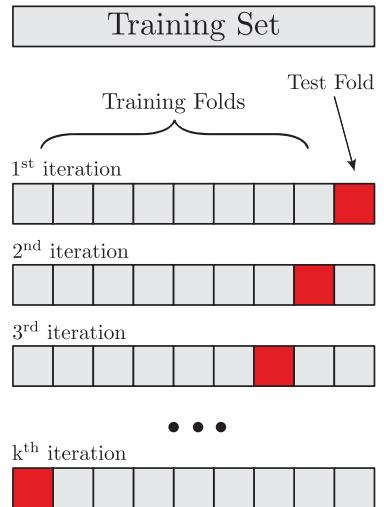


Figure 31.6: k -fold cross validation.

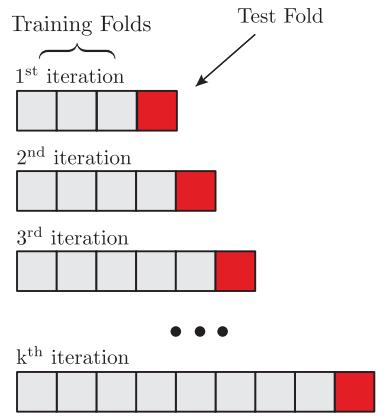


Figure 31.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

31.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

31.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (31.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (31.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (31.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

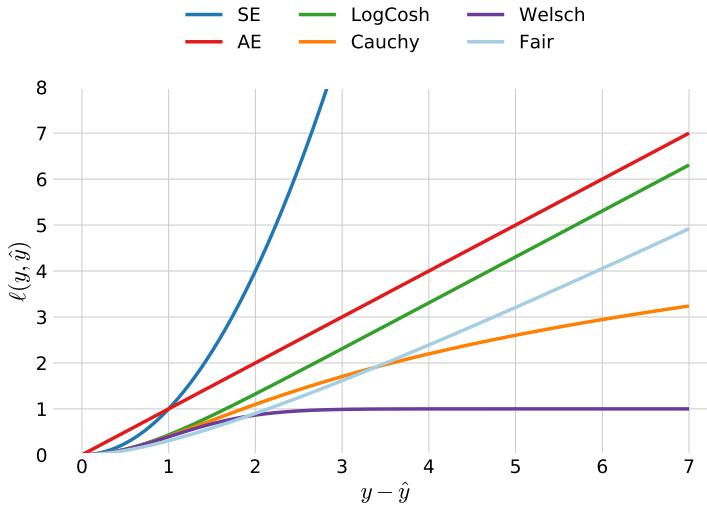


Figure 31.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

31.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (31.28)$$

31.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

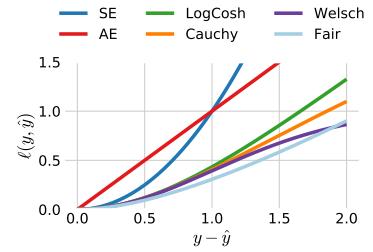


Figure 31.9: Zoom in of Figure ??.

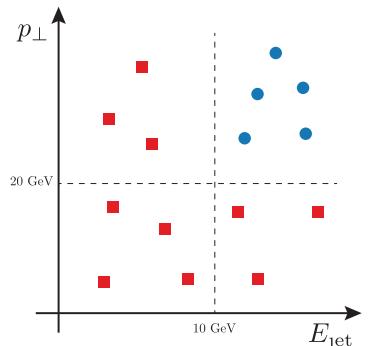


Figure 31.10: Illustration of the cuts a decision tree model make for *signal* in blue circles and *background* in red squares. This is an visualization in the feature space of the decision tree seen in Figure ??.

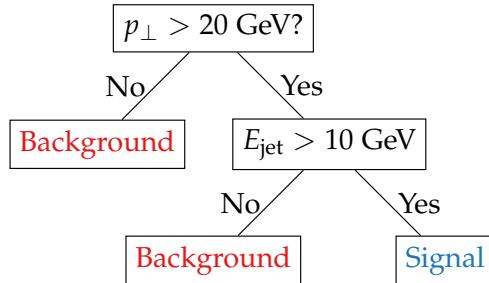


Figure 31.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

31.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (31.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 32 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (31.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (31.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (31.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (31.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (31.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (31.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

31.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

31.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (31.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

31.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (31.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

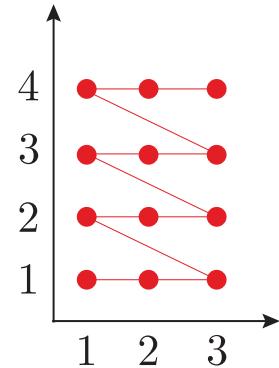


Figure 31.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (31.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

31.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

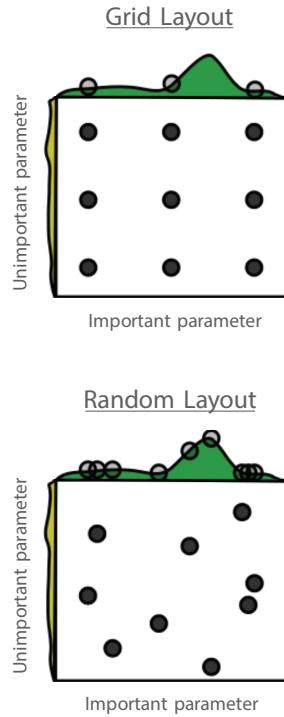


Figure 31.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the datadependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (31.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

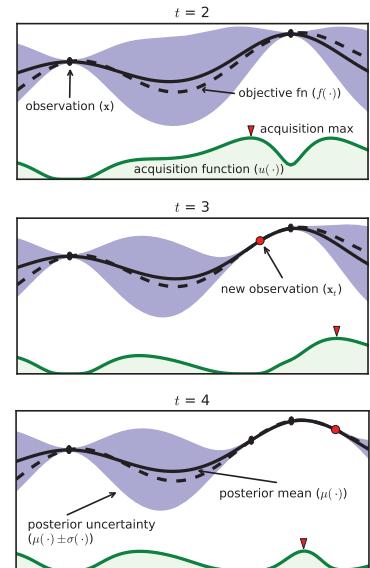


Figure 31.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

31.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (31.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 29 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (31.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 30 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (31.42)$$

Axiom 31 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (31.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (31.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (31.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (31.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

Axiom 32 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (31.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

⁴³ Not necessarily linearly correlated.

32. Danish Housing Prices

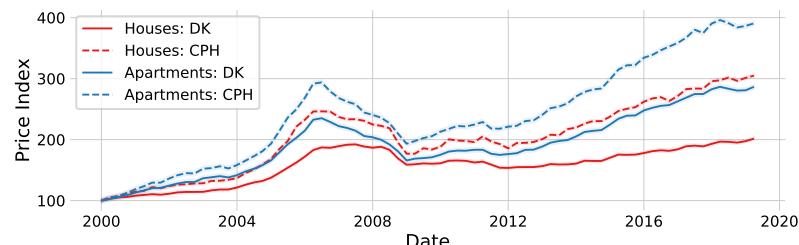
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 32.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

32.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

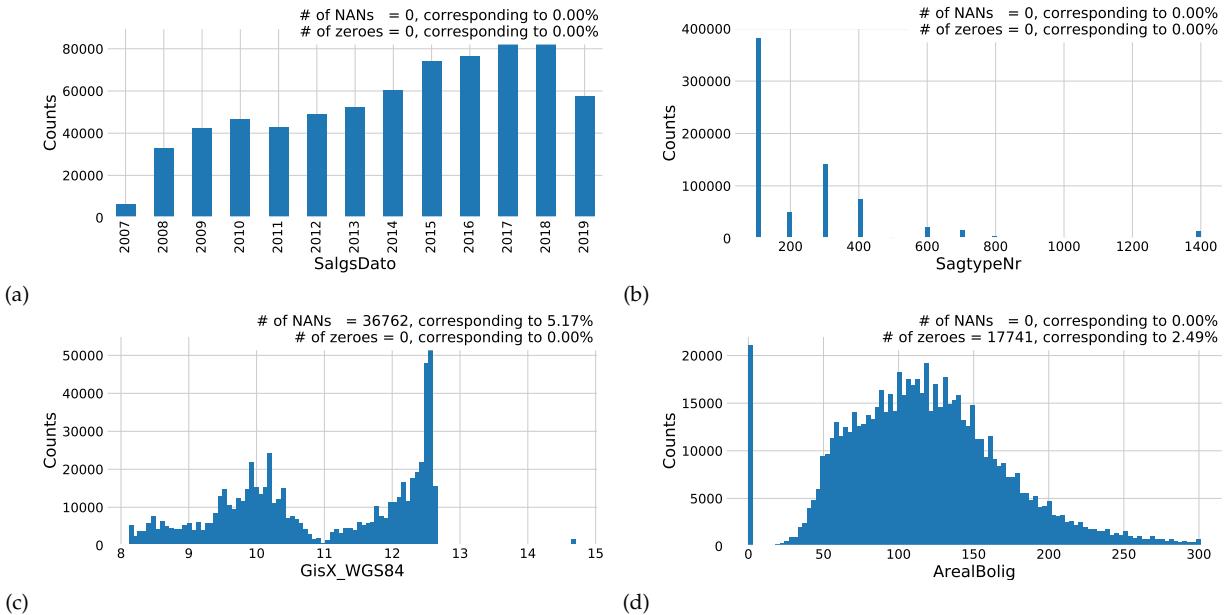
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 32.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 32.1: XXX TODO!

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

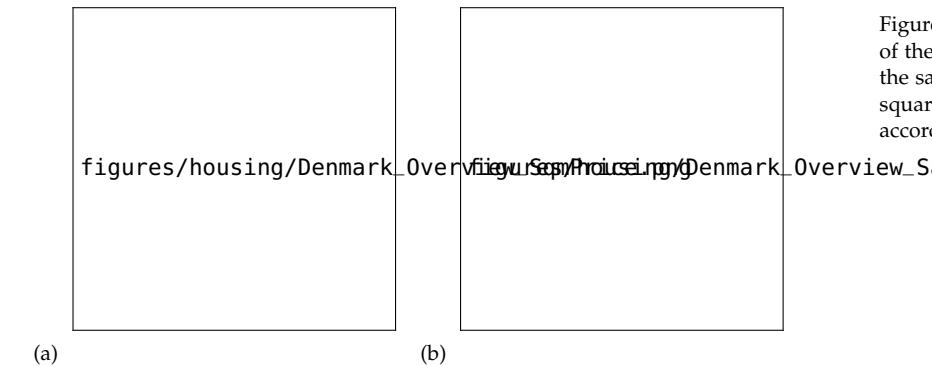


Figure 32.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

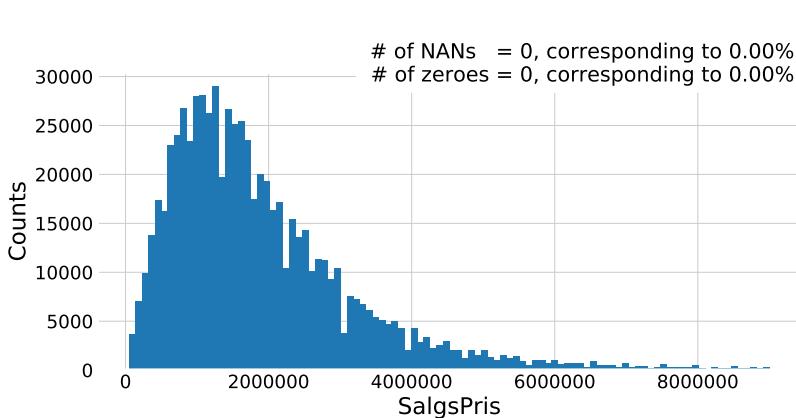


Figure 32.4: Histogram of prices of houses and apartments sold in Denmark.

32.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

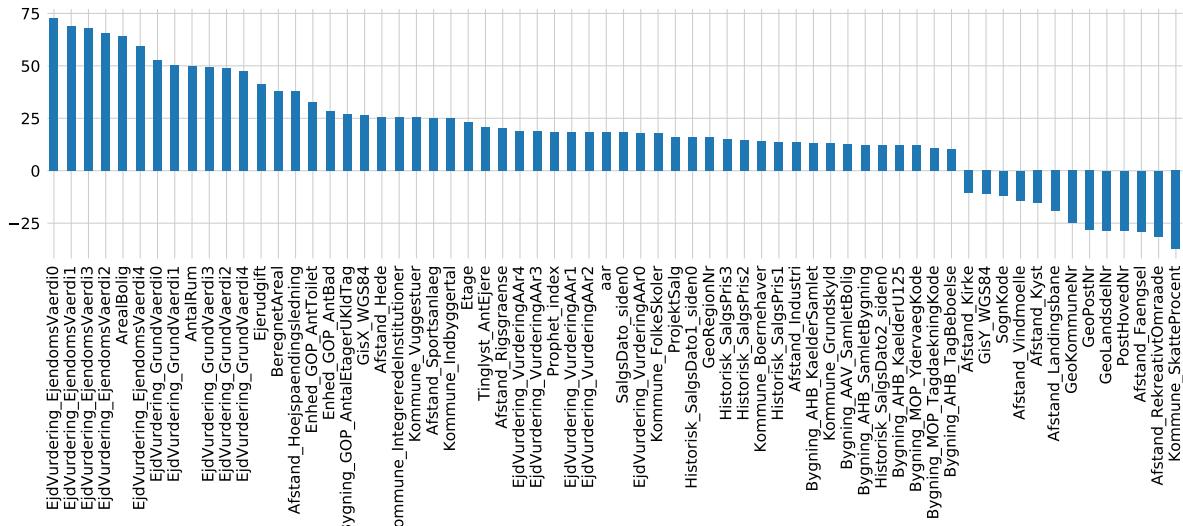


Figure 32.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. [?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

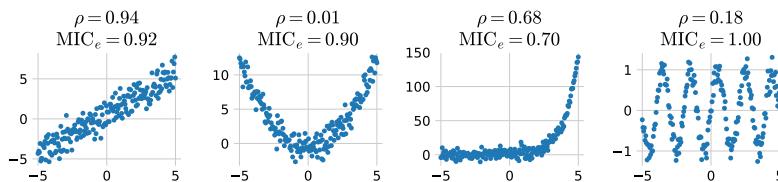


Figure 32.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

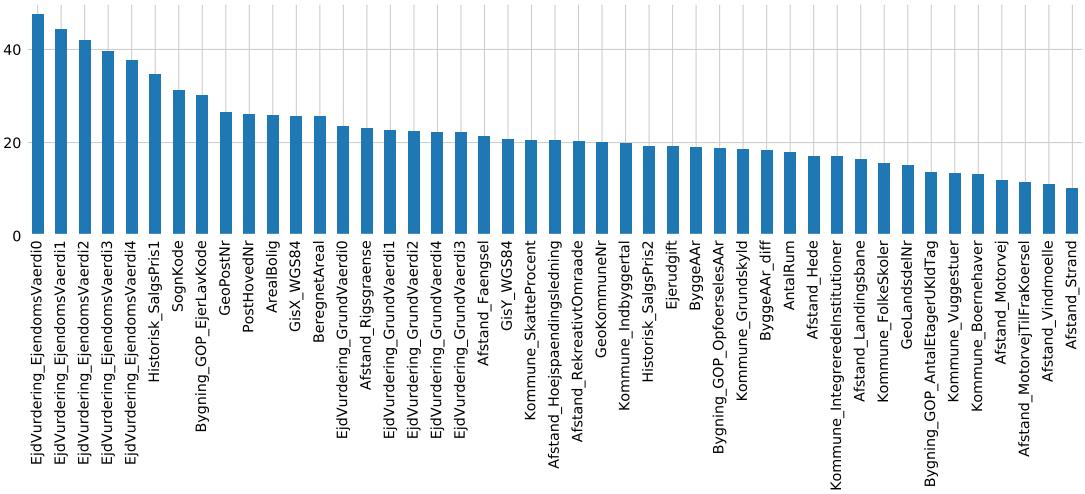
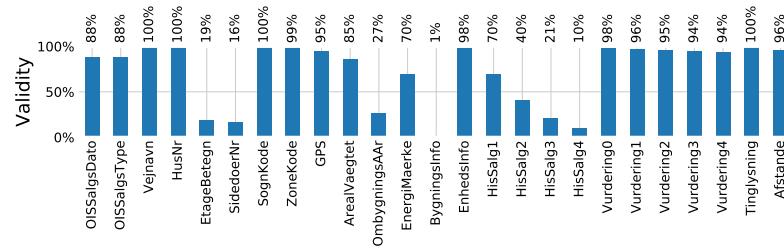


Figure 32.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where MIC > 10%.

32.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningsAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaerdingsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 32.8: Percentage of valid counts for each variable grouped together in categories.

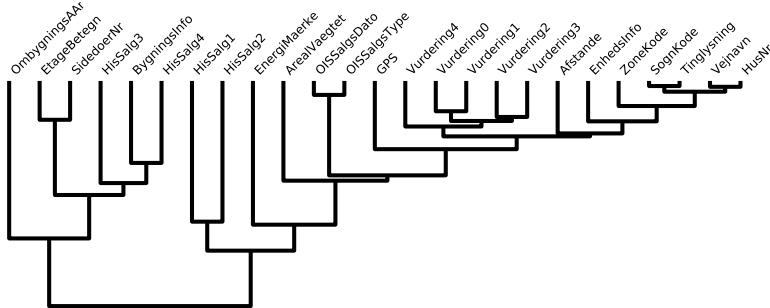


Figure 32.9: Validity Dendrogram
TODO!

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

32.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 32.2: XXX TODO!

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 32.3: XXX TODO!

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 32.4: XXX TODO!

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ???. The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, `EnergiMaerke`, is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

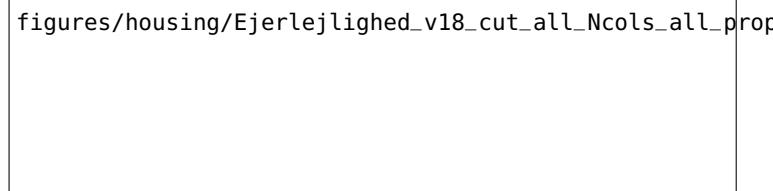
32.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by [] at Facebook. It is based on a decomposable time series model [] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (32.1)$$

where ϵ_t is a normally distributed error term. [] fit this equation with a generalized additive model (GAM) [] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ???. In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png`

¹¹ In addition to only having the year the house was built

¹² In their paper, [] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 32.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb`

Figure 32.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (32.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

32.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (32.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (32.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (32.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

32.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 32.6: train test split XXX **TODO!**

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 32.7: train test split tight XXX **TODO!**

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (32.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (32.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

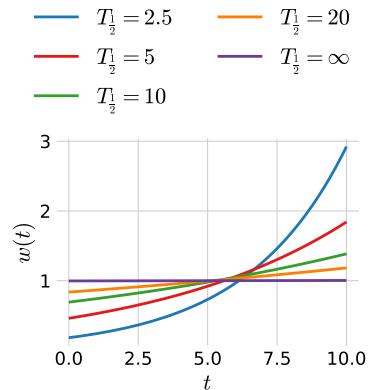


Figure 32.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

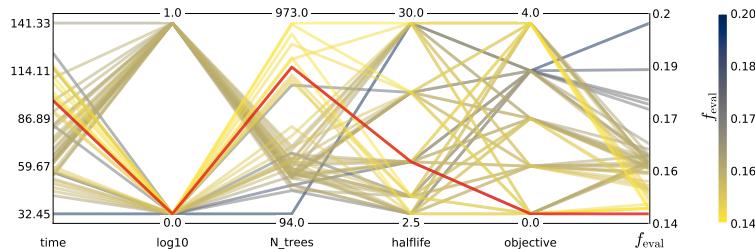
Table 32.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 32.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

32.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 32.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The standard deviations (and MAD_0) and the 5-folds in the cross validation's `Cauchy` (0), `Fair` (1), `LogCosh` (2) `SquaredError` (3), and `Welsch` (4) are mapped to the integers in the parentheses.

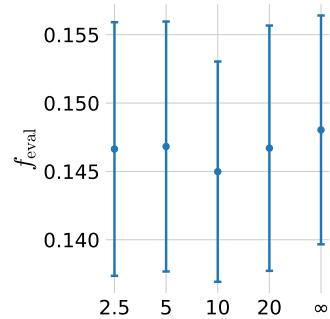


Figure 32.14: XXX Halflife $T_{\frac{1}{2}}$.

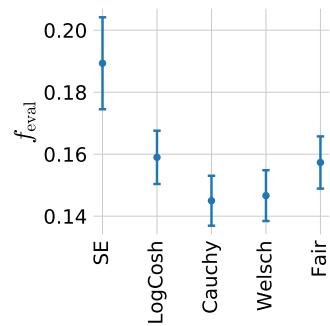


Figure 32.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 32.10: XXX

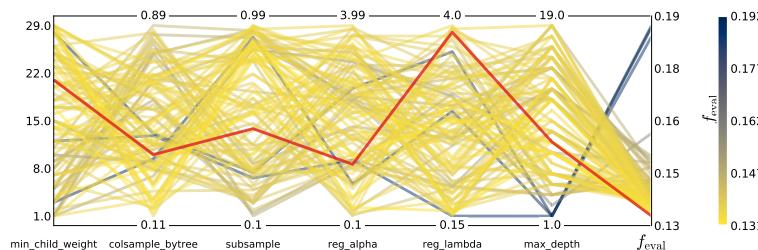


Figure 32.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

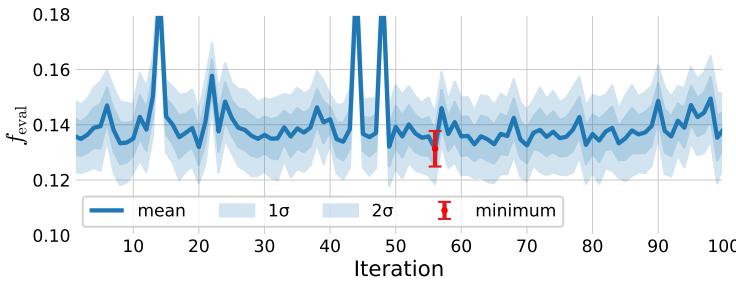


Figure 32.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

32.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 32.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

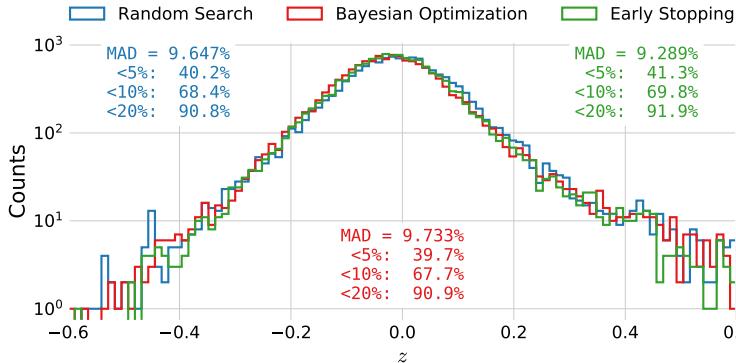


Figure 32.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

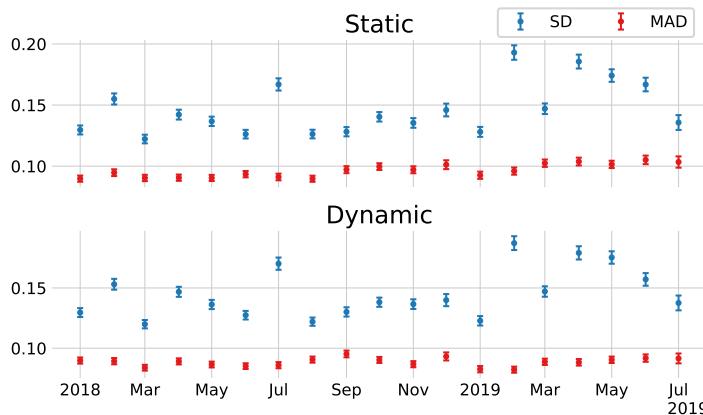


Figure 32.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month's sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

Table 32.11: XXX

	Train	Test	2019
Normal	5.80 %	4.97 %	6.19 %
Tight	5.69 %	4.94 %	6.19 %

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (32.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (32.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predictions are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

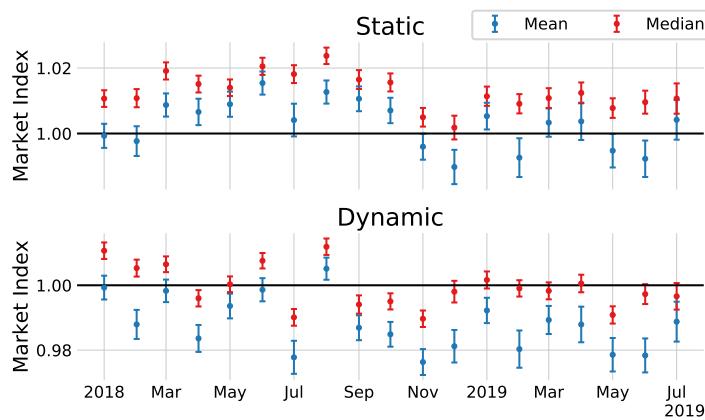


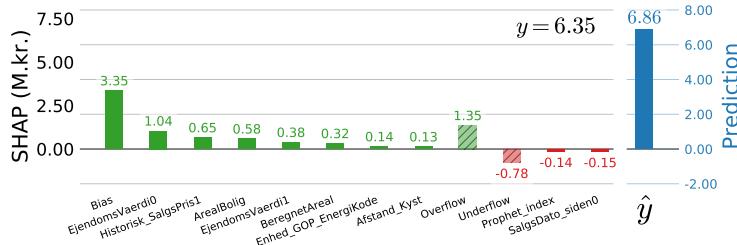
Figure 32.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z	Table 32.12: XXX ejer
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$	
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012	
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002	

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z	Table 32.13: XXX villa
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007	
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016	
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002	

32.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 32.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~Of the total number of the entire dataset~~ model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

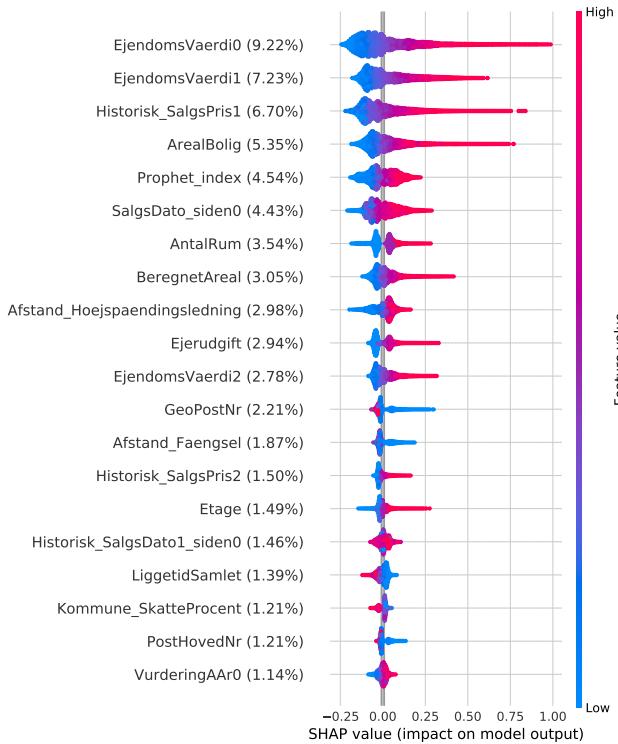


Figure 32.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (DOM) described by the variable `LiggetidSamlet` where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.

`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 32.24: Feature importance of apartment prices using the XGB-model. XXX

32.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

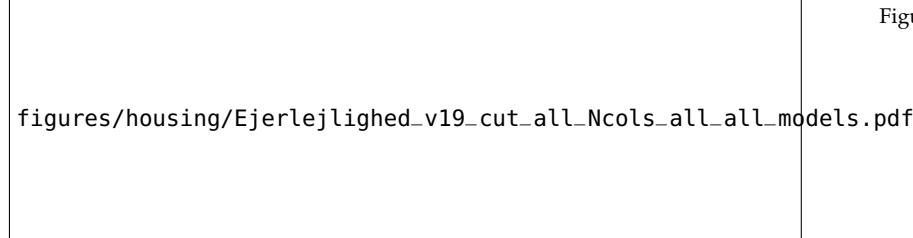


Figure 32.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (32.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (32.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

32.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

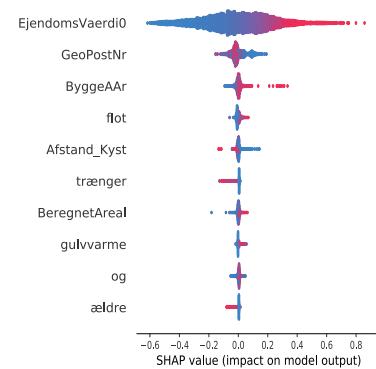


Figure 32.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
*arg min arg min
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

33. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

34. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

35. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

35.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

35.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (35.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (35.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

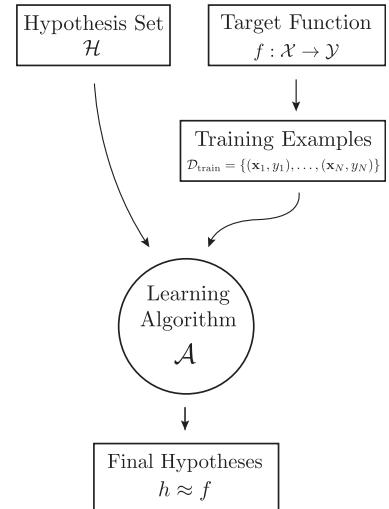


Figure 35.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (35.3)$$

35.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (35.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(\mathbf{x}, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 25 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (35.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 26 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (35.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (35.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 27 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (35.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (35.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (35.10)$$

Theorem 33 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (35.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (35.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 34 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (35.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (35.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

35.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: ?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 35 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (35.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (35.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

35.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

35.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (35.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

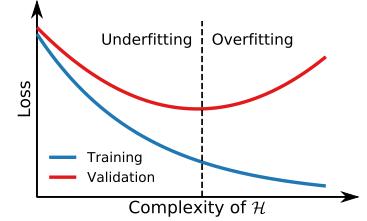


Figure 35.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (35.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (35.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (35.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (35.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \quad (35.22) \end{aligned}$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

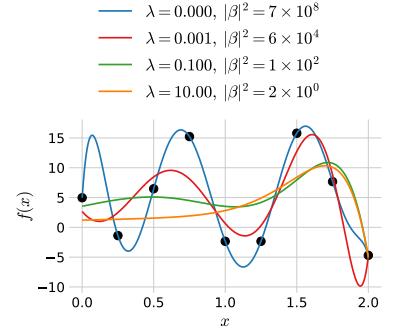


Figure 35.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (35.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by [1] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (35.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

35.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

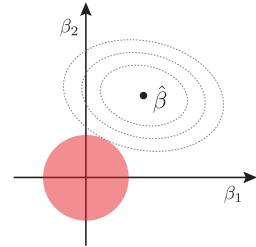


Figure 35.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

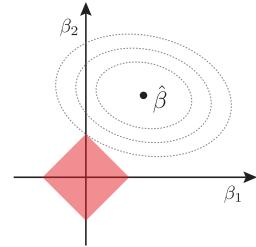


Figure 35.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

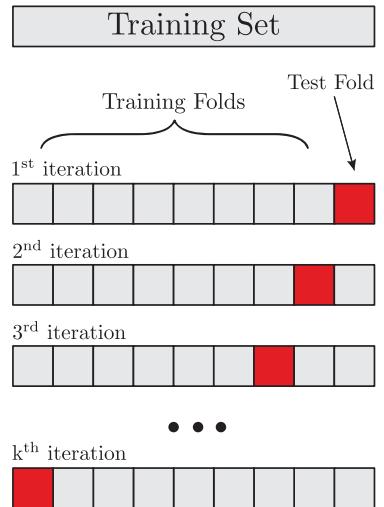


Figure 35.6: k -fold cross validation.

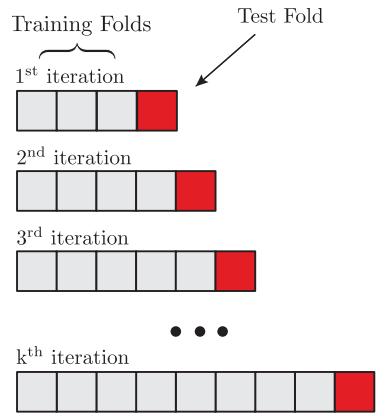


Figure 35.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

35.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

35.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (35.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (35.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (35.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

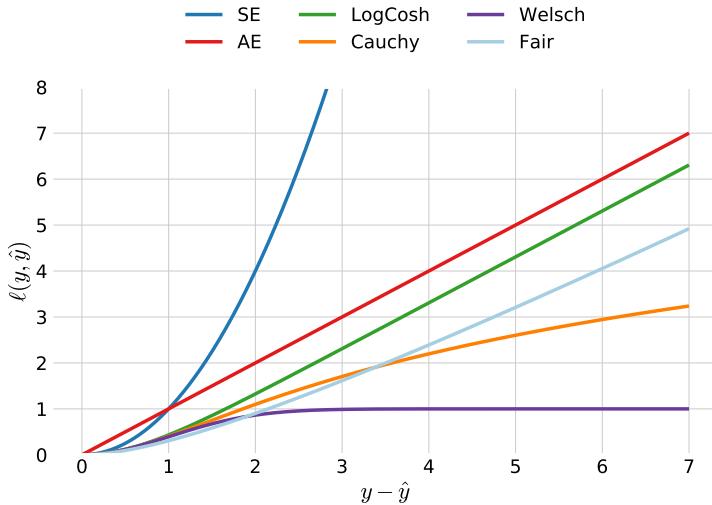


Figure 35.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

35.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (35.28)$$

35.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

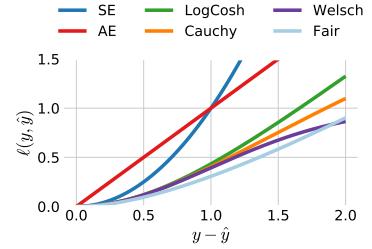


Figure 35.9: Zoom in of Figure ??.

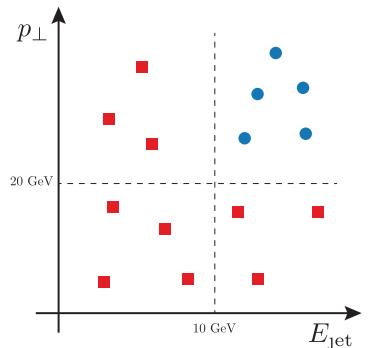


Figure 35.10: Illustration of the cuts a decision tree model make for *signal* in blue circles and *background* in red squares. This is an visualization in the feature space of the decision tree seen in Figure ??.

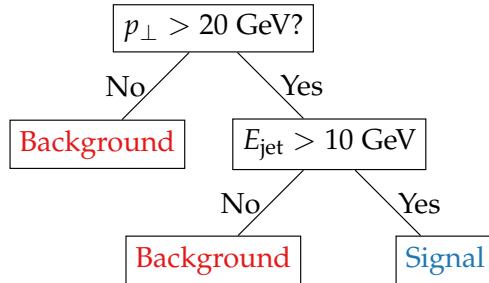


Figure 35.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

35.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (35.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 36 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (35.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (35.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (35.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (35.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (35.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (35.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

35.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

35.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (35.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

35.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (35.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

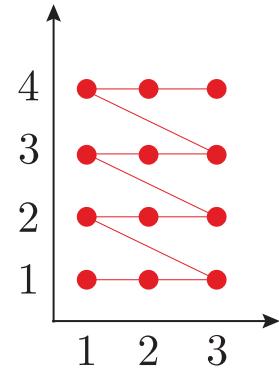


Figure 35.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (35.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

35.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

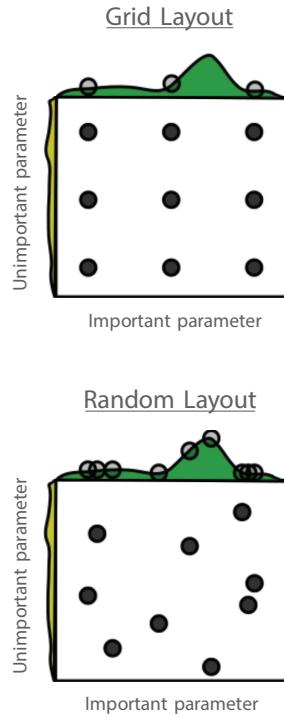


Figure 35.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries all possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the datadependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (35.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

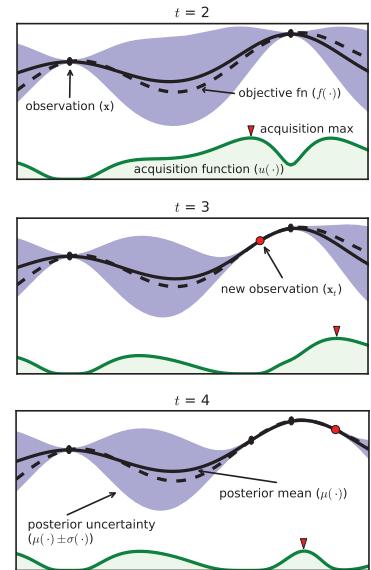


Figure 35.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

35.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (35.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 33 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (35.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 34 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (35.42)$$

Axiom 35 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (35.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (35.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (35.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (35.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

Axiom 36 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (35.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

⁴³ Not necessarily linearly correlated.

36. Danish Housing Prices

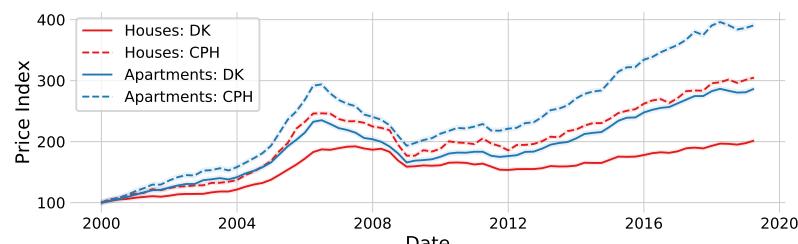
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 36.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

36.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

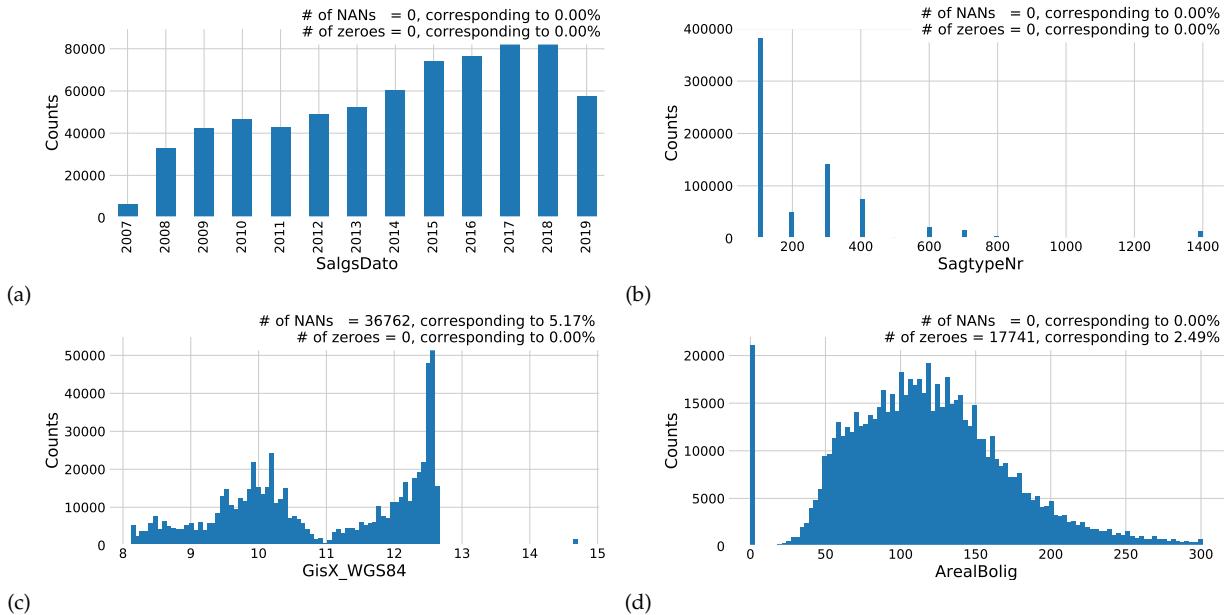
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 36.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 36.1: XXX TODO!

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

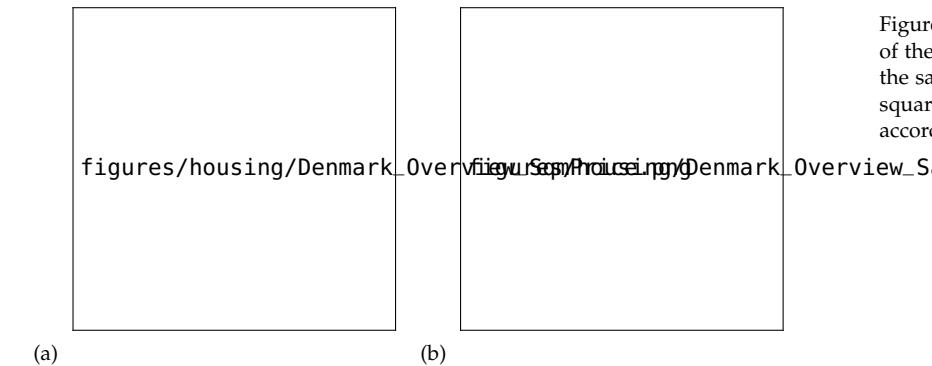


Figure 36.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

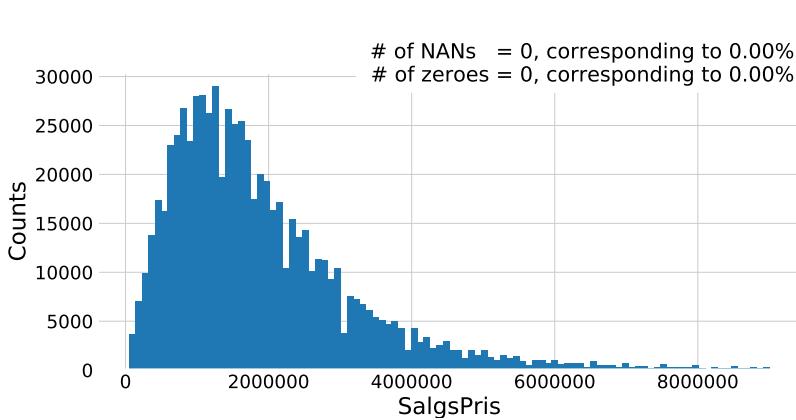


Figure 36.4: Histogram of prices of houses and apartments sold in Denmark.

36.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

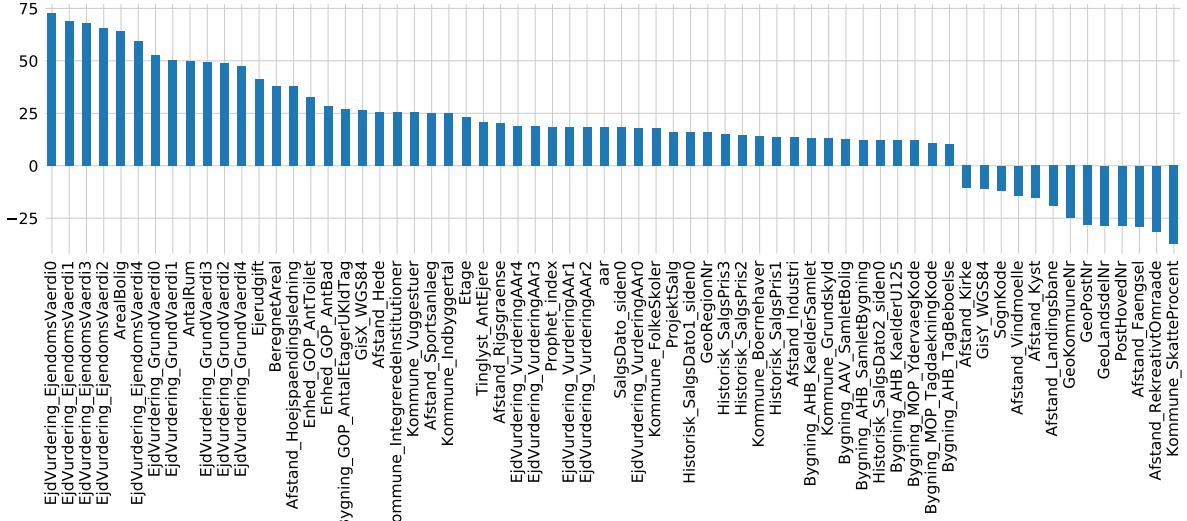


Figure 36.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. [?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

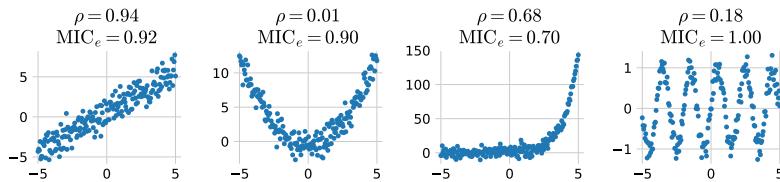


Figure 36.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

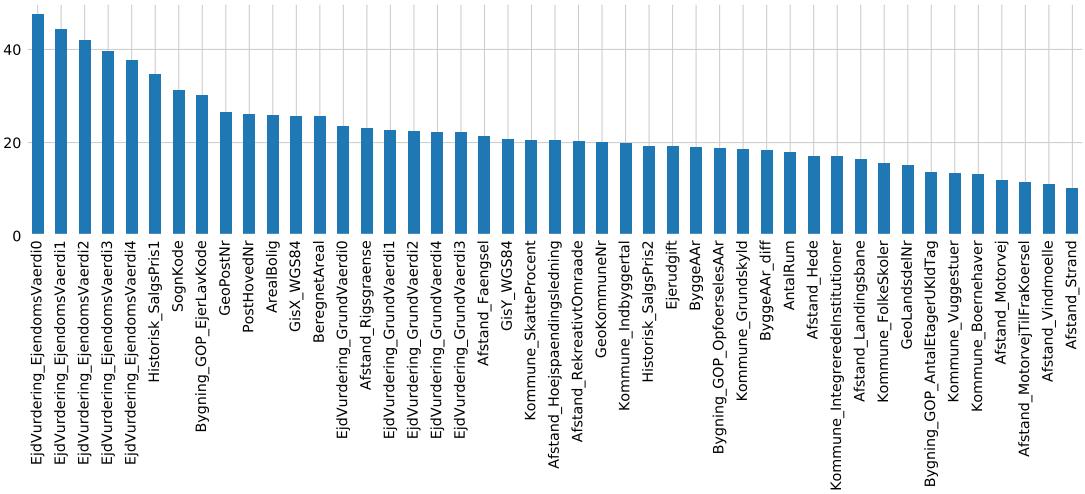
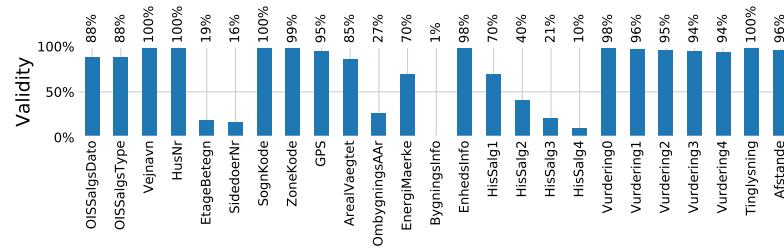


Figure 36.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where MIC > 10%.

36.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningsAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaerdingsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 36.8: Percentage of valid counts for each variable grouped together in categories.

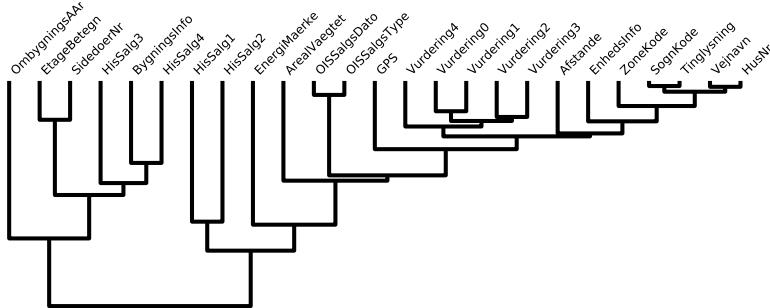


Figure 36.9: Validity Dendrogram
TODO!

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

36.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 36.2: XXX TODO!

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 36.3: XXX TODO!

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 36.4: XXX TODO!

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ???. The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, `EnergiMaerke`, is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

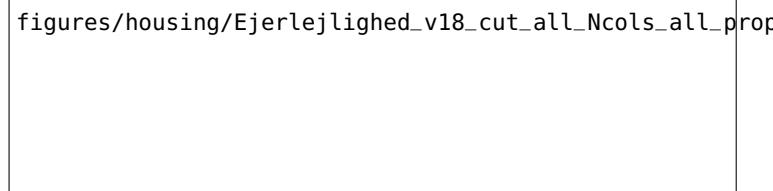
36.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by [] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (36.1)$$

where ϵ_t is a normally distributed error term. [] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ???. In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png`

¹¹ In addition to only having the year the house was built

¹² In their paper, [] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 36.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb`

Figure 36.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (36.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

36.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (36.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (36.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (36.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

36.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 36.6: train test split XXX **TODO!**

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 36.7: train test split tight XXX **TODO!**

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (36.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (36.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

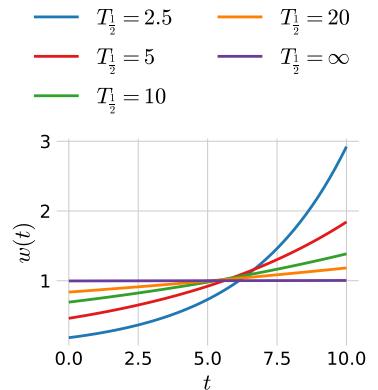


Figure 36.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

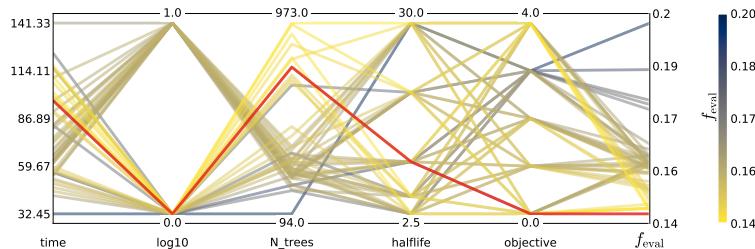
Table 36.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 36.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

36.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 36.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The standard deviations (and propagation) and the 5-folds in the cross validation's `Cauchy` (0), `Fair` (1), `LogCosh` (2) `SquaredError` (3), and `Welsch` (4) are mapped to the integers in the parentheses.

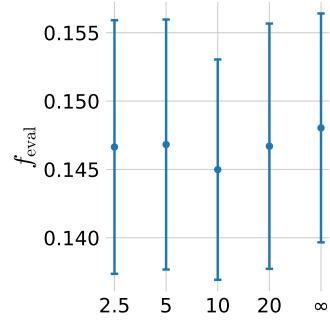


Figure 36.14: XXX Halflife $T_{\frac{1}{2}}$.

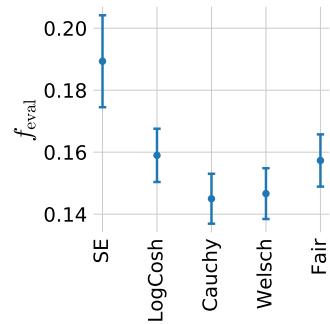


Figure 36.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 36.10: XXX

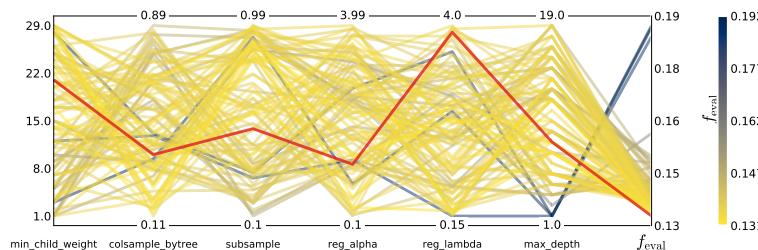


Figure 36.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

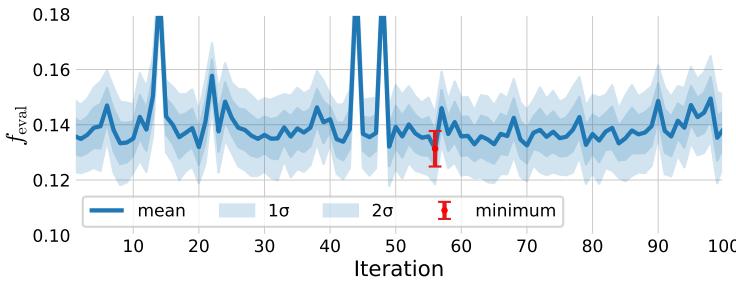


Figure 36.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

36.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 36.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

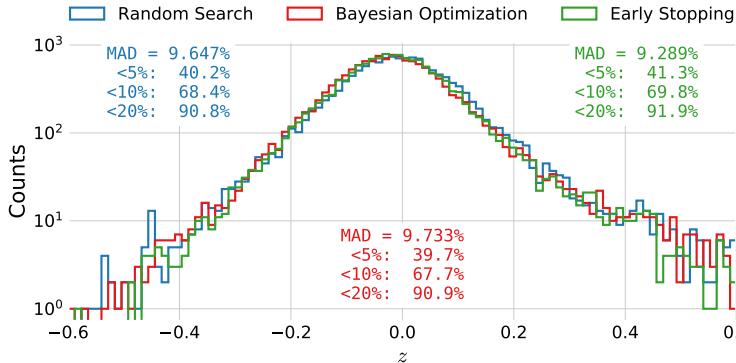


Figure 36.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

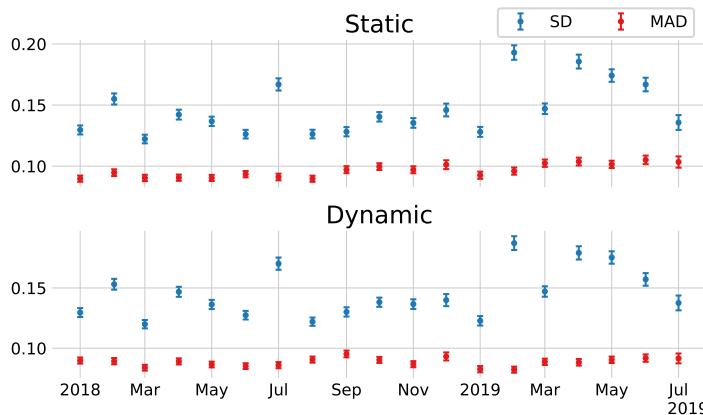


Figure 36.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month's sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

Table 36.11: XXX

	Train	Test	2019
Normal	5.80 %	4.97 %	6.19 %
Tight	5.69 %	4.94 %	6.19 %

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (36.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (36.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predictions are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

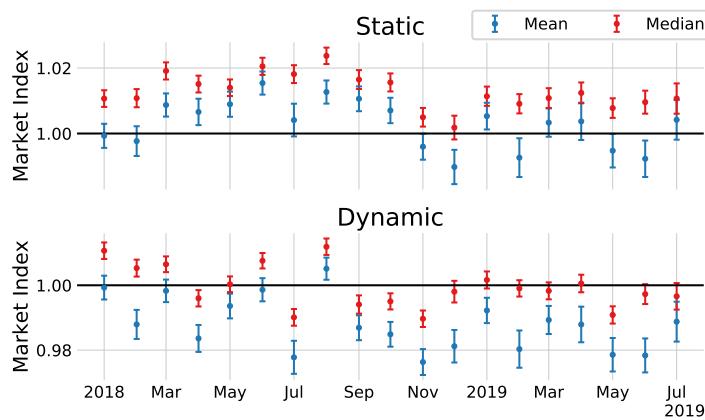


Figure 36.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002

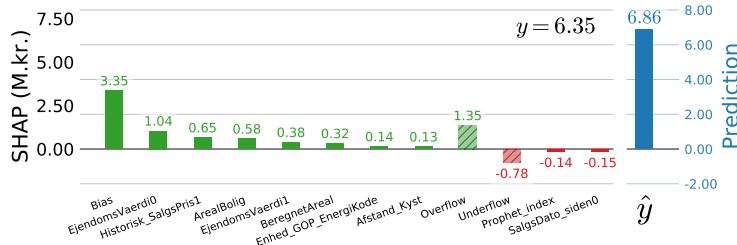
Table 36.12: XXX ejer

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002

Table 36.13: XXX villa

36.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 36.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~On the right hand side of the plot~~ model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

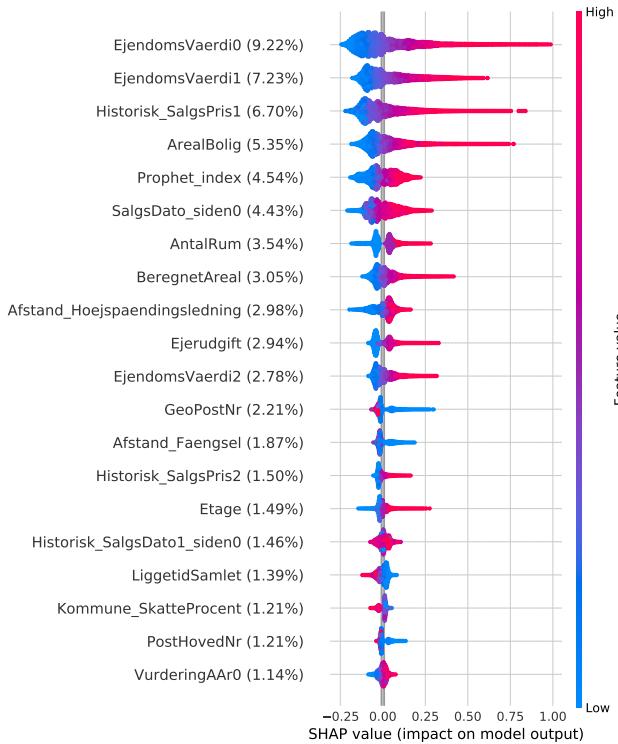


Figure 36.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.

`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 36.24: Feature importance of apartment prices using the XGB-model. XXX

36.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

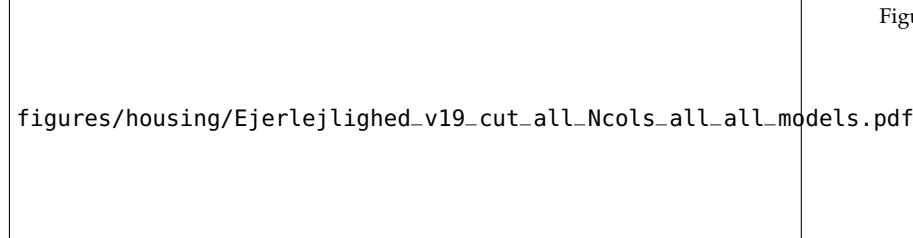


Figure 36.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (36.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (36.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

36.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

```
University of Copenhagen
[square, numbers, nonamebreak]natbib
lipsum
epigraph
booktabs
amsthm,amssymb,amsmath [mathscr]eucal
graphicx
grffile
tikz [subpreambles=true]standalone
[caption=false]subfig
```

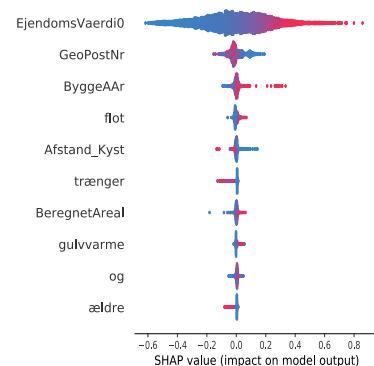


Figure 36.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

```
fancyvrb
xspace
Lemma Lemma[lemma] Corollary Theorem Axiom
bm
[super]nth
import xifthen pdfpages transparent
[binary-units=true]siunitx kr.kr. kr.kr. M.kr.M.kr. year yearsyears
yryr
mathtools
[dvipsnames]xcolor
*arg min arg min
units
makeidx
```


CHRISTIAN MICHELS
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

A PHYSICIST'S
APPROACH TO
MACHINE LEARNING
—
UNDERSTANDING
THE BASIC BRICKS

SUPERVISOR:
TROELS PETERSEN
NIELS BOHR INSTITUTE
UNIVERSITY OF COPENHAGEN

Copyright © 2019

Christian Michelsen

[HTTPS:/ / GITHUB.COM / CHRISTIANMICHELSEN](https://github.com/CHRISTIANMICHELSEN)

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019

Contents

List of Figures

List of Tables

37. Abstract

This sample book discusses the design of Edward Tufte's books and the use of the `tufte-book` and `tufte-handout` document classes.

38. Introduction

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

— Lewis Carroll, *Alice in Wonderland*

NOT ONLY is the title of this project fairly broad, so are the subjects covered in this thesis. The overall goal of this project is to apply machine learning to different datasets and see how well these comparatively new tools might improve classical statistical methods. The project have dealt with two (seemingly) very different datasets: Danish housing prices and Quark-Gluon discrimination in particle physics, and the aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects; two sub-projects which are covered in each part of this book.

The first part of the thesis deals with the problem of estimating housing prices as precisely and accurately as possible. This was the sub-project that was worked on in the beginning of the overall project and worked as an initial introduction to the application of machine learning to real-life datasets. The housing prices dataset thus became the playground in which the subtleties of these new modern tools were examined, where the difference between real life datasets with all its quirks, outliers and bad formatting, and curated toy datasets that works out of the box (such as the famous Iris dataset [? ?]) were experienced first hand. Since the project started the dataset changed due to a new collaboration with the Danish housing agency **Boligsiden** where the agreement was, stated shortly, that we would get their data and they would get our results. Boligsiden is a natural collaborator since they are the biggest on the market¹ and have been very helpful the continuos process of providing data but it also should also be noted that they have had no say on the results presented in this thesis. During this initial stage, the author sparred with Simon Gudiksen² who also worked on the same dataset, however, both projects were done independently. Where Gudiksen focussed on the prediction of the time evolution of the housing prices using Recurrent Neural Networks (RNN), my work was mostly on the different levels and methods of hyperparameter optimization with some smaller detours into Natural Language Processing (NLP) as an example.

The second part, the Quark-Gluon discrimination in particle

The background for this masters's thesis, is that it is part of a so-called 4+4 Ph.D. project (also known as an integrated Ph.D.). The Ph.D. dissertation is about the use of machine learning and deep learning in the field of ancient genomics. Here ancient DNA is sampled and analysed with the hope of finding patterns, structure, in the genome which were previously unknown. The overall goal is two-fold. On the big scale it is the better understand human history in the broadest sense of the word history. Where did we come from, where did we go. On a much smaller scale, the goal is to understand local history and migration patterns; how did we end up where we did. It is with this background that this project should be seen: as an introduction to the general use of applied machine learning.

¹ Due to being owned by the "Dansk Ejendomsmæglerforening", The Danish Association of Chartered Estate Agents.

² Who afterwards went on to get a job at Boligsiden.

physics, was the main part of the project. Not only was most of the time focussed on this sub-project, it was also the work that generated the highest academic output; an article based on this is in the making. This part dealt with data from the Large Electron Positron collider (LEP) which was an underground particle accelerator at CERN built in 1989 and was discontinued in 2000, where the first phase (LEP1), from 1989-1995, is the sole source of data. As the name suggests it collided electrons and positrons together in what is still the largest electron-positron accelerator ever built [?]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \rightarrow q\bar{q}g$ and $Z \rightarrow q\bar{q}gg$ were examined in this thesis. It is especially the distributions of these gluon jets and the difference between Data³ and MC that are of interests to the theoreticians that develop the MC-models. At first an improved *b*-tagging algorithm was developed based on only the vertex variables since the primary variables of interest to the theoreticians are the shape variables. Here methods and code developed in the hyperparameter optimization process from the housing prices part were used. After the improvement in the *b*-tagging model, an event-based *g*-tag model – in comparison to the jet-based *b*-tagging model – was implemented which (hopefully) allows one to extract useful events of interest. Having found these useful events, one can start looking at how the distributions in the relevant variables differ between Data and MC. Finally XXX **TODO!**

The thesis is structured such that ?? introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, ?? describes the housing prices part of the project as mentioned above, ?? introduces the basic physics in the standard model and the Lund string model which is used throughout the rest of the theses, ?? explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters ?? and ?? discusses the overall work in this thesis and concludes on it.

The work presented in this thesis is split up into two parts as presented above, however, it should be noted that during the analysis part of the project they were treated not as two different projects but rather as two different instances of same underlying problem: teaching computers how to find patterns in high-dimensional data automatically and should thus not be seen as two independent projects. This also highlight another key aspect of this project, that the author does not have any background in particle physics other than rudimentary knowledge stemming from an undergraduate education in general physics.

All of the work presented here is performed by the author unless otherwise noted.

³ Where “Data” with capital D refers to the actual, measured data and “data” refers to any arbitrary selection of data.

39. Machine Learning Theory

“People worry that computers will get too smart and take over the world, but the real problem is that they’re too stupid and they’ve already taken over the world.”

— Pedro Domingos

MACHINE LEARNING is the method of teaching computers how to automatically find patterns in (often high-dimensional) data. According to some sceptics machine learning (ML) is just glorified statistics, however, by the same logic physics is just glorified mathematics. In contrary, machine learning is a set of subject located somewhere along the hypothetical line from simple, classical statistics to futuristic artificial intelligence. It includes methods ranging from the well-known statistical methods such as linear regression to the modern, advanced zoo of different neural networks [?] which has seen a plethora of use cases in recent years.

39.1 Statistical Learning Theory

This chapter deals with the theory of ML which Statistical Learning Theory is a subcategory of. Many books are written on the subject where this thesis especially follows the overall notation used in the very accessible introduction in the book Learning From Data [?] and the graduate course Advanced Topics in Machine Learning [?] at the computer science institute¹ at the faculty of Science, University of Copenhagen. Statistical learning theory is the analysis of how to not only find the function, or *hypothesis*, that matches the data best, but also bounding the difference in performance between this hypothesis and the hidden, underlying data generation distribution often only known by Nature.

Overall there are two subfields within machine learning: supervised and unsupervised². The difference depends on whether or not the data that is trained on is labelled or not. Classic linear regression is an example of the former and linear dimensionality reduction using PCA of the latter. Since unsupervised learning techniques are only used sparsely throughout this project, the main focus will be on supervised learning.

¹ Datalogisk Institut Københavns Universitet, DIKU.

² Also known as “self-supervised” or “predictive” learning.

39.2 Supervised Learning

In supervised learning we are given a set of N different samples of which we for each one knows M different variables written as the column-vector $\mathbf{x}_i = [x_1, x_2, \dots, x_M] \in \mathcal{X}$ for the i th observation and \mathcal{X} denotes the sample space. All of these samples as a whole is written as the matrix \mathbf{X} with the individual observations \mathbf{x}_i transposed and stacked on top of each other $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$ such that row i in \mathbf{X} corresponds to observation i . In the case of supervised learning we also have the label y of each sample $y \in \mathcal{Y}$ where \mathcal{Y} denotes the sample or label space. In the case where y is a real number $\mathcal{Y} = \mathbb{R}$ the problem is said to a *regression* problem, e.g. predicting the price of a house. On the contrary, if \mathcal{Y} is binary such that $\mathcal{Y} = \{0, 1\}$ then the problem is said to be a (binary) *classification* problem³ e.g. predicting whether or not a particle is a quark or not.

Without any loss of generality let the focus for now be on classification. The goal is to find the underlying “true” function $f : \mathcal{X} \mapsto \mathcal{Y}$ that gives the correct label $y \in \mathcal{Y}$ for each observation $\mathbf{x} \in \mathcal{X}$. This function, however, is unknown and cannot perfectly be found. Although it is impossible to find f it is possible to learn an approximation of it h based on some training observations $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^* , is chosen among a set of candidate hypotheses $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$, and hopefully h^* will be a good approximation of f , $h^* \approx f$. A schematic overview of this process can be seen in Figure ??.

How can one make sure that h^* really is a good approximation of f ? That is where Statistical learning theory comes into play. From a statistical standpoint we are interested in modelling the unknown joint probability $P(\mathbf{x}, y)$ over \mathcal{X} and \mathcal{Y} . We assume that $\mathcal{D}_{\text{train}}$ is independent and identically distributed (*iid*)⁴ from $P(\mathbf{x}, y)$ and thus want to find the hypothesis whose predictions $h(\mathbf{x}) = \hat{y}$ matches the conditional probability distribution $P(y|\mathbf{x})$ as best as possible.

To quantify the statement “as best as possible” in the previous paragraph we define the loss function ℓ which measures the loss for predicting \hat{y} instead of y : $\ell(\hat{y}, y) = \ell(h(\mathbf{x}), y) \in \mathbb{R}^+$. Given ℓ we now introduce the method of (empirical) risk minimization [?] and the expected loss⁵:

$$L(h) = \mathbb{E} [\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) dP(\mathbf{x}, y). \quad (39.1)$$

The optimal hypotheses h^* is the hypothesis which minimizes the expected loss $L(h)$. However, the joint probability distribution $P(\mathbf{x}, y)$ is unknown and we are thus left with the empirical loss⁶ of h on \mathcal{D} :

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^N \ell(h(\mathbf{x}_i), y_i), \quad (39.2)$$

which is an approximation of $L(h)$ based on the training data avail-

³ If $\mathcal{Y} \in \mathbb{Z}$ then it would be a multi-class classification problem.

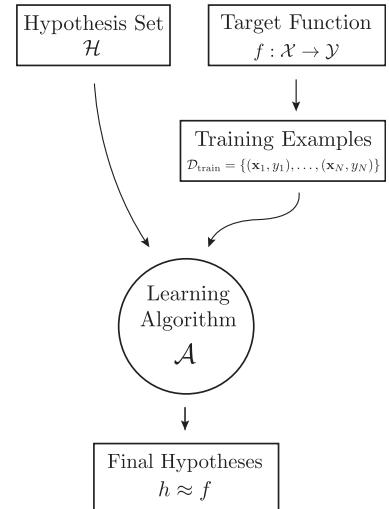


Figure 39.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

⁴ This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distribution as the one that generated the past events. These assumptions are sometimes called the PAC assumptions where PAC is an abbreviation for Probably, Approximately Correct.

⁵ Also called expected error or the out-of-sample error.

⁶ Also called empirical error.

able. Now the optimal hypothesis h^* can be defined:

$$h^* = \arg \min_{h \in \mathcal{H}} \hat{L}(h, S). \quad (39.3)$$

39.3 Generalization Bound

In ?? the method of selecting the optimal hypothesis h^* out of the total set of candidate hypotheses \mathcal{H} was sketched. However, there is still no guarantee that h^* will work well, that is to say that the *generalization error* $G(h)$ might be big:

$$G(h) = \hat{L}(h, S) - L(h). \quad (39.4)$$

The generalization error is thus the difference between the expected error $L(h)$ and the empirical error $\hat{L}(h, S)$. It describes the loss in performance of our chosen model compared to the optimal, yet hidden, model. Since $P(\mathbf{x}, y)$ is unknown, $G(h)$ cannot be computed, however, it is possible to bound this error using statistical learning theory. To do so, the union bound and Hoeffding's (one-sided) inequalities are introduced.

Lemma 28 (The Union Bound). *For any finite or countably infinite sequence of events E_1, E_2, \dots (not necessarily independent):*

$$\mathbb{P} \left\{ \bigcup_{i \leq 1} E_i \right\} \leq \sum_{i \leq 1} \mathbb{P} \{E_i\}. \quad (39.5)$$

The union bound, in simple terms, states that the probability of any one of n events happening is less than or equal to the sum of the individual probabilities of the events happening. As an example, let $E_1 = \{2, 4, 6\}$ be the event that a die rolls an even number and $E_2 = \{4, 5, 6\}$ be the event that a die rolls a number larger than or equal to 4. Then $\mathbb{P} \{E_1 \cup E_2\} = \mathbb{P} \{2, 4, 5, 6\} \leq \mathbb{P} \{E_1\} + \mathbb{P} \{E_2\}$.

Lemma 29 (The one-sided Hoeffding's inequalities). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \frac{1}{N} \sum_{i=1}^N Z_i - \mu \geq \epsilon \right\} \leq e^{-2n\epsilon^2} \quad (39.6)$$

and

$$\mathbb{P} \left\{ \mu - \frac{1}{N} \sum_{i=1}^N Z_i \geq \epsilon \right\} \leq e^{-2n\epsilon^2}. \quad (39.7)$$

When using the union bound on equation ?? and equation ?? we arrive at Hoeffding's (two) sided inequality:

Lemma 30 (The two-sided Hoeffding's inequality). *Let Z_1, \dots, Z_n be independent random variables each belonging to the $[0, 1]$ interval such that $\mathbb{P} \{Z_i \in [0, 1]\} = 1$ and $\mathbb{E}[Z_i] = \mu$ for all i , then for every $\epsilon > 0$:*

$$\mathbb{P} \left\{ \left| \frac{1}{N} \sum_{i=1}^n Z_i - \mu \right| \geq \epsilon \right\} \equiv \mathbb{P} \{|\hat{\mu} - \mu| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}, \quad (39.8)$$

where we have defined the (empirical) average of Z to be $\hat{\mu} = \frac{1}{n} \sum_{i=1}^N Z_i$

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0, 1]$ interval⁷, $\ell(\hat{y}, y) \in [0, 1]$ for all \hat{y}, y , we can bound the generalization error $G(h)$ by letting $Z_i = \ell(\hat{y}_i, y_i) = \ell(h(\mathbf{x}_i), y_i)$ be the loss of h in sample (\mathbf{x}_i, y_i) . By comparing Lemma ?? and equation (??) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (??): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\{|G(h)| \geq \epsilon\} = \mathbb{P}\{|\hat{L}(h, S) - L(h)| \geq \epsilon\} \leq 2e^{-2N\epsilon^2}. \quad (39.9)$$

This equation provides a bound on the difference between the empirical loss and the expected loss. Say the probability of the generalization error being larger than $\epsilon = 0.1$ is needed for $N = 100$ samples, we find that the this probability is $P = 27\%$. The generalization bound can be rewritten in terms of δ :

$$\delta = 2e^{-2N\epsilon^2} \in (0, 1) \Rightarrow \epsilon = \sqrt{\frac{\ln \frac{2}{\delta}}{2N}} \Rightarrow \quad (39.10)$$

Theorem 37 (Hoeffding's inequality for a single hypothesis). *Assume that ℓ if bounded in the $[0, 1]$ interval, then for a single hypothesis h and any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{|\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right\} \leq \delta. \quad (39.11)$$

Equation (??) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}$ is δ or similarly that with probability greater than $1 - \delta$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2N}}. \quad (39.12)$$

This is a powerful result relating the performance for a (fixed) hypothesis h with the number of samples, N . We see that a higher N yields a tighter bound on the generalization error, however, inversely⁸ related to on the certainty⁹ δ of this bound.

There is a big assumption of this derivation although: that the hypothesis h cannot depend on the sample S and thus has to be chosen before seeing the data. We say that h has to be *fixed*. Of course the term machine learning indicates that some kind of learning is taking place: exactly as seen previously where we wanted to find the optimal hypotheses h^* out of all the possible ones \mathcal{H} . For now assume that \mathcal{H} is finite and consists of M hypotheses: $|\mathcal{H}| = M$. We thus have $[h_1, h_2, \dots, h_M]$ hypotheses which we test simultaneously and where Hoeffding's inequality is true for each of them leading to the following theorem:

Theorem 38 (Hoeffding's inequality for a finite set of hypotheses candidates). *Assume that ℓ is bounded in the $[0, 1]$ interval and that $|\mathcal{H}| = M$. Then for any $\delta \in (0, 1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}\right\} \leq \delta. \quad (39.13)$$

⁷ Which it is for classification, however, it can be extended in a similar fashion for regression.

⁸ Not to be understood as $1/x$ in this context.

⁹ Technically the certainty of the model is $1 - \delta$.

Proof. The proof begins by denoting H_i as the event where:

$$|\hat{L}(h_i, S) - L(h_i)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}},$$

and then taking the union bound (the first inequality) followed by applying Hoeffding's inequality to each part in the sum (the second inequality):

$$\begin{aligned} \mathbb{P} \left\{ \exists h \in \mathcal{H} : |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ = \mathbb{P} \left\{ \bigcup_{h \in \mathcal{H}} |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \mathbb{P} \left\{ |\hat{L}(h, S) - L(h)| \geq \sqrt{\frac{\ln \frac{2}{\delta'}}{2N}} \right\} \\ \leq \sum_{h \in \mathcal{H}} \delta' \\ = M\delta'. \end{aligned}$$

By making the substitution $\delta = M\delta'$ we arrive at equation (??). \square

As we did in equation (??), equation (??) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$|\hat{L}(h, S) - L(h)| \leq \sqrt{\frac{\ln \frac{2M}{\delta}}{2N}}. \quad (39.14)$$

This bound is looser than the one for only a single hypothesis by a factor $\ln M$, however, this holds for the optimal hypothesis h^* .

39.3.1 Generalization Bound for infinite hypotheses

Section ?? dealt with the case of a single hypothesis h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = M$. When M goes towards ∞ the generalization bound goes to ∞ and the bound becomes useless. However, even simple models such as a linear classifier¹⁰ that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T \mathbf{x}$ is positive and $\hat{y} = 0$ when it is negative has $|\mathcal{H}| = \infty$. Since there are an infinite number of hypotheses $h(\mathbf{w})$, assuming we allow \mathbf{w} to take any real values as is almost always the case, \mathcal{H} is infinite.

To solve this obvious problem with the Hoeffding inequality, we introduce¹¹ the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space \mathcal{H} : $d_{VC}(\mathcal{H}) = d_{VC}$. The VC-dimension is a measure of the complexity of the hypothesis space, the degrees of freedom of the model so to say. For example the VC-dimension of the d -dimensional linear classifier defined above¹² is $d_{VC} = d$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^d$.

¹⁰ Also known as the perceptron. Often includes a constant offset b as well which is omitted for brevity.

¹¹ For proof, see: ?]

¹² In the general case when the offset b is included: $d_{VC} = d + 1$.

Theorem 39 (VC Generalization Bound). *Let \mathcal{H} be a hypotheses class with VC-dimension: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. Then with probability at least $1 - \delta$:*

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}} + 1 \right) \right)}. \quad (39.15)$$

Equation (??) states that the out of sample error $L(h)$ is bounded from above by the empirical error $\hat{L}(h, S)$ and the $\sqrt{\cdot}$ which is related to the complexity of the hypothesis space \mathcal{H} , the number of samples N and the certainty δ . We will call this model complexity penalty $\Omega(N, \mathcal{H}, \delta)$:

$$\Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln \left(\frac{4}{\delta} \left((2N)^{d_{\text{VC}}(\mathcal{H})} + 1 \right) \right)}. \quad (39.16)$$

As the hypothesis space complexity d_{VC} grows, the generalization bound loosens but it is more likely that \mathcal{H} contains a strong hypothesis. This relationship is called the approximation-estimation or the *bias-variance* tradeoff. When the model is too simple to properly fit the complexity in the data it is called *underfitting*¹³, when the model is so complex that it starts fitting the inherent noise in the data it is called *overfitting*¹⁴. The loss as a function of model complexity gives the characteristic curve illustrated in Figure ???. As the model complexity increases the training loss decreases. Initially also the validation loss decreases, but at some point the behavior of model on the validation set worsens and the loss increases; overfitting happens.

39.4 Avoiding overfitting

Avoiding overfitting is one of the most important issues in machine learning. By now, most modern machine learning algorithms have the inherent model complexity needed for overfitting and thus it has to be managed. Due to the importance of the issue, a number of different methods preventing or reducing overfitting exists. Most of them are not complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in ??, cross validation in ??, and early stopping in ??.

39.4.1 Model Regularization

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [?] was one of the first to describe this method in 1943. In particular regularization was used to solve *ill posed* linear regression problems. Regular linear regression problems refer to minimizing the residual sum of squares written in matrix form as:

$$\hat{\beta}_{\text{LS}} = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \quad (39.17)$$

$$\mathbf{f}(\mathbf{X}) = \mathbf{X}\beta,$$

¹³ Here the error from $\hat{L}(h, S)$ dominates.

¹⁴ Here the error from $\Omega(N, \mathcal{H}, \delta)$ dominates.

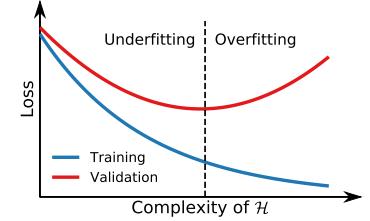


Figure 39.2: Illustration of the empirical loss as a function of model complexity. The **training error** is shown in blue and **validation error** in red.

where \mathbf{y} is vector of values we are trying to predict¹⁵, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix of input parameters such that each observation $\mathbf{x}_i \in \mathbb{R}^M$ is a M -dimensional vector stacked in \mathbf{X} as rows with N total observations, $\hat{\beta}_{LS}$ is the vector of unknown coefficients¹⁶ of the linear least squares (LS) model \mathbf{f} , and $\|\cdot\|_2$ is the normal Euclidean norm. In general, the p -norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}. \quad (39.18)$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\beta\|_2^2$ with respect to (w.r.t.) β and setting the derivative equal to 0 to find the minimum¹⁷ yields the solution for β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta &\Rightarrow \quad (39.19) \\ \hat{\beta}_{LS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

However, this solution for $\hat{\beta}_{LS}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. \mathbf{X} has to be full rank [?]. If this is not the case, the problem is said to be ill posed. Tikhonov solved this problem by adding an extra term to the minimization problem, which we will call Ω for simplicity. For a specific choice of Ω , one gets:

$$\hat{\beta}_{L_2} = \arg \min_{\beta} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\beta\|_2^2 \right\}, \quad (39.20)$$

where $\lambda \geq 0$ is the regularization strength. This is the so-called L_2 -regularization, also known as ridge regression for linear problems. For ridge regression¹⁸ the corresponding solution for β is:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (39.21)$$

where \mathbf{I} is the identity matrix¹⁹. This extra term, $\Omega = \lambda \|\beta\|_2^2$, acts as a shrinkage factor on the coefficients of β . Looking at equation (??), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\beta}_{L_2} &= \arg \min_{\beta} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to : } \sum_{i=1}^N \beta_i^2 &= \|\beta\|_2^2 \leq t, \end{aligned} \quad (39.22)$$

for some $t \geq 0$ with a one-to-one mapping between λ and t . We thus see that L_2 regularizes the coefficients of β to have some maximal norm. The effect of the regularization is controlled by λ , where $\hat{\beta}_{L_2} \rightarrow \hat{\beta}_{LS}$ for $\lambda \rightarrow 0$ and $\hat{\beta}_{L_2} \rightarrow \mathbf{0}$ for $\lambda \rightarrow \infty$. An example of this can be seen in Figure ??.

Here $N = 9$ datapoints were randomly generated such that the x -values are evenly spaced from 0 to 1 and $y \sim \mathcal{N}(\mu = 0, \sigma = 10)$. They were then fit with a 9-order polynomial by minimizing

¹⁵ E.g. the prices of a collection of houses.

¹⁶ Here excluding the constant offset β_0 which can be included trivially.

¹⁷ When checking the double derivative wrt. β it is seen that this really is a minimum and not a maximum (or saddle point).

¹⁸ Here $\hat{\beta}_{L_2}$ is the solution for any general function f and $\hat{\beta}_{\text{ridge}}$ is the specific solution for a linear function f , where linear is w.r.t. to the model parameters β .

¹⁹ The $\lambda \mathbf{I}$ in equation (??) also acts as a conditioner on the problem in the sense that it reduces the condition number of the matrix to be inverted turning the ill-posed problem to a well-behaved one.

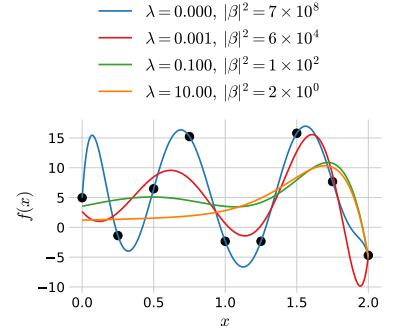


Figure 39.3: Effect of tuning the regularization strength λ in ridge regression.

equation (??) for different values of λ . Here we see the regularizing effect of λ , going from $\lambda = 0$ in blue which fits all points²⁰ with a high degree of variance and a wildly oscillatory pattern to $\lambda = 10$ in orange which is mostly flat in most of the interval. Also note in the legend how the norm of the fit parameters also decrease with λ as expected. This is a great example of the *bias-variance* tradeoff mentioned in ??, where bias refers the error the model makes when it is not advanced enough to fit the overall trend in the data (underfitting) and variance refers to the error the model makes when it starts to fit spurious noisy fluctuations in the training set which are not present in the validation set (overfitting). In this example $\lambda = 0$ is clearly overfitting the data whereas $\lambda = 10$ is an example of underfitting.

Other values of the regularization function Ω exists, for example the L_1 -penalty:

$$\Omega = \lambda \|\beta\|_1, \quad (39.23)$$

where the 1 -norm, also known as Manhattan norm, is used. In the case of linear problems the L_1 -penalty leads to Lasso regression introduced by ?] in 1996. As with the L_2 -penalty, the L_1 -penalty also regularizes the coefficients of β , however, this loss leads to sparse²¹ solutions. An illustration of this can be seen in Figure ?? and Figure ?? where the constraint regions of β os shown in red and the grey ellipses are the contours of the non-constrained problem. Notice how the intersection of the contour lines and the constrain region leads to $\beta_1 \neq 0, \beta_2 \neq 0$ for the L_2 -penalty whereas it leads to the sparse solution $\beta_1 = 0, \beta_2 \neq 0$ for the L_1 -penalty. This is a general pattern seen for L_p -penalties for $p \leq 1$.

Overall, model regularization is heavily used in modern machine learning algorithms. In general, the function or the so-called *objective function* \mathcal{L} they are trying to minimize is:

$$\mathcal{L}(h) = \hat{\mathcal{L}}(\ell, h, S) + \Omega(h) \quad (39.24)$$

where $\hat{\mathcal{L}}$ is the empirical loss and Ω is the regularization penalty. As can be seen from the above discussion, choosing the right value for the regularization strength is fundamental problem in model regularization. How to choose a suitable value for λ is discussed in ?? and the choice of the training loss function ℓ in ??.

39.4.2 Cross Validation

In general we want to be able to estimate the performance²² of the developed model. Since evaluation the model on the data it was already trained on would give a biased estimate of the performance, we need an unbiased method of doing so. The easiest way of doing so would be to set a fraction of the data aside, e.g. 20 %, train on the remaining part and then evaluate the performance on the data set aside. Splitting the data up like this would provide us with a training (data)set and a test set in a 80 : 20 ratio. This would then

²⁰ Since the order of the polynomial is the same as the number of datapoints.

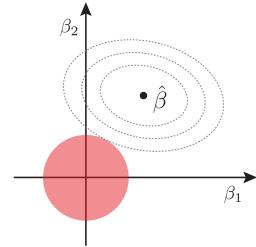


Figure 39.4: Sketch of the minimization problem defined in equation (??), i.e. for a L_2 -penalty. The **constrain region** shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

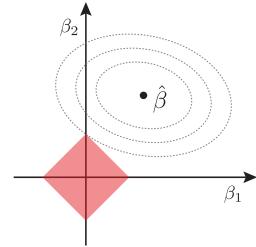


Figure 39.5: Sketch of the similar minimization problem defined in Figure ?? for the L_1 -penalty. The **constrain region** shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in 2D-space and the contours of the unconstrained solution is shown with grey, dashed lines.

²¹ Meaning that a number of the β_i coefficients are 0, the number depending on λ .

²² "Performance" is used here as the word for the general metric to be optimized for, no matter whether or not this metric should be maximized or minimized.

yield an unbiased performance estimate when evaluation the performance on the test set. However, if one then needed to compare two different models and choose the best one, as is often the case, this method would not work since we would choose the model with the best performance on the test set which can be seen as training on the test set and thus it has “tainted” the purity of the test set. To avoid this, an additional split is made such that we get a training set, a validation set, and a test set, where you often see a 80 : 10 : 10 ratio. The two models can then be compared on the validation set and the performance of the chosen model can be estimated from the test set.

This way of splitting up the data has some clear benefits and is thus also often used. There is a drawback, however, and that is that we are not fully utilizing a lot of the data in this way. Basically 20% of the data are only used to provide a single number of performance and does not necessarily allow an uncertainty or confidence interval of this measurement to be calculated. Thus other methods of estimating model performance are developed where one of the most used and well-known are the k -fold cross validation (CV). Here the entire dataset are split up into k chunks which are randomly drawn subsamples (without replacement). In the first iteration, the model is trained on the first $k - 1$ subsamples and evaluated on the last k subsample. In the second iteration the evaluation subsample is a new one. This process is continued k times until all samples in the dataset have been trained and evaluated on [?]. For an illustration of this, see Figure ???. The process yields k estimates of the performance of the model which can then be averaged to form a single performance number and the variability of the performance can even be gauged²³. The disadvantage of k -fold CV is that the performance estimate is now slightly biased, however, this effect is generally very small. The biggest disadvantage is the computational burden related to doing k -fold CV where $k \gg 1$. A compromise often used in applied ML is $k = 5$ which is also what is used in this project.

Special care has to be taken when dealing with time series data. Here the problem of “data leakage” is often introduced inadvertently. Data leakage is when the model is exposed to information from the test set that it was not supposed to be exposed to. In the case of time series data, if the data is split by the usual k -fold CV, then each subsample contains events from all times and the model does not learn how to predict future events. To circumvent this problem, a special type of k -fold CV for time series data has to be used. Here all samples up to a specific time, e.g., all houses sold before 2018, is used for training and then it is evaluated on the performance of samples after the event, e.g., houses sold in 2018. For an illustration of this, see Figure ??.

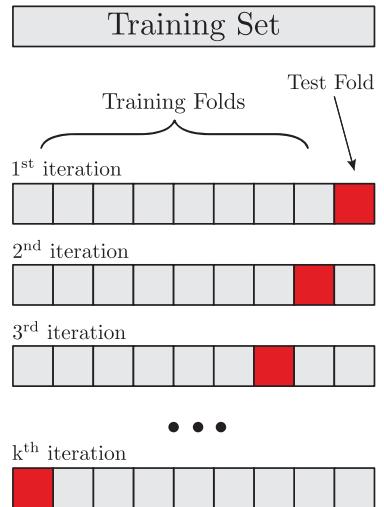


Figure 39.6: k -fold cross validation.

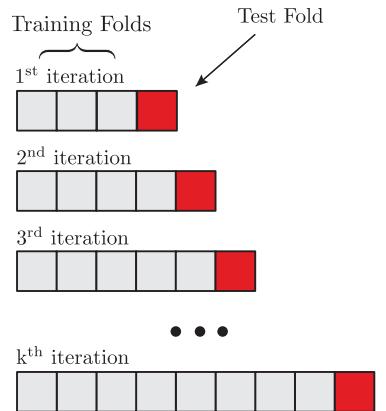


Figure 39.7: k -fold cross validation for time series data.

²³ Special care has to be taken here since the k different performance values are not independent.

39.4.3 Early Stopping

Most modern machine learning models are trained iteratively. This is the case for both (boosted) decision trees and neural networks, both of which are used in this project. Iteratively here means that the model starts off with an initial guess of the parameters of the model to learn and then by looking at the data “learns” a new set of values for the parameters. The question then becomes: for how long should the model be allowed to continue training.

This is a prime example of the bias-variance tradeoff. The model should be trained long enough to be able to capture the complexity inherent in the data but also should not train for so long that it starts to overfit the data. Even though Figure ?? was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can be taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for training set and validation set $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} . As mentioned in ??, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on \mathcal{D}_{val} is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping just because of a single outlier due to noise that terminates the process, one often uses *patience* in the early stopping process: if the loss has not decreased since the last minimum after *patience* number of iterations, then terminate the process. Early stopping is thus an easy way of avoiding overfitting for iteratively trained models only requiring a validation set.

39.5 Loss functions

How we evaluate the performance of a model is of course very important since it defines the metric for the problem; what is good and what is bad? Obviously this depends on whether or not we are dealing with a classification problem or a regression problem. Let us focus on the latter. Say that a house is estimated to cost 2 million DKK (M.kr.) but was sold for 4 M.kr.. Compare this to a house that was estimated to cost 8 M.kr. but was sold for 6 M.kr. In both cases the price was 2 M.kr. wrong, but does this mean that both predictions are equally good? The first case was a factor of 2 off, whereas the second was only 25% off. The first case underestimated the price whereas the second overestimated.

As it should be clear by now the choice of loss function ℓ is of utmost importance. It is also not a problem that can be solved by computers, is problem-specific, and has to be defined manually. The choice of loss function is what is called a *hyperparameter*, the optimization of which is further discussed in ???. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2, \quad (39.25)$$

where y is the true value and \hat{y} is the predicted one. Squared Error has the advantage that it is differentiable everywhere, an effect

that is both needed for many statistical derivations but also a requirement for some machine learning models. The disadvantage is that it gives too much weight to outliers since every deviation away from the truth is squared. In contrast to this, there is the Absolute Error (AE) defined as:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \quad (39.26)$$

For AE, outliers have a lot smaller weight since it deals with the absolute value of the deviation and not the squared deviation. However, this comes at a price; AE is not differentiable at every point: at $y - \hat{y} = 0$ the derivative of the absolute value function is un-defined. Many functions have been invented trying to deal with these problems. For a more general discussion of loss functions, see e.g. Barron [?]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [?], Welsch [?] and Fair [?] loss functions are used:

$$\begin{aligned} \ell_{\text{LogCosh}}(y, \hat{y}) &= \log(\cosh(y - \hat{y})) \\ \ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\ \ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\ \ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right). \end{aligned} \quad (39.27)$$

The above loss functions share some similarities with AE, and in addition to this they are all (twice) differentiable functions. They are shown in Figure ???. They are shown for only positive values of $y - \hat{y}$ since they are symmetric in $y - \hat{y}$. Notice how SE quickly grows very large compared to the others. Absolute Error has a kink at $y - \hat{y} = 0$ as the only one of the functions. Welsch is bounded in the interval $[0, 1]$. The derivative of both LogCosh and Fair goes toward 1 when $y - \hat{y}$ goes towards ∞ , whereas it goes to 0 for the Cauchy loss. A priori it is almost impossible to know which one of these loss functions performs best for a specific data set, so they have to be treated as hyperparameters.

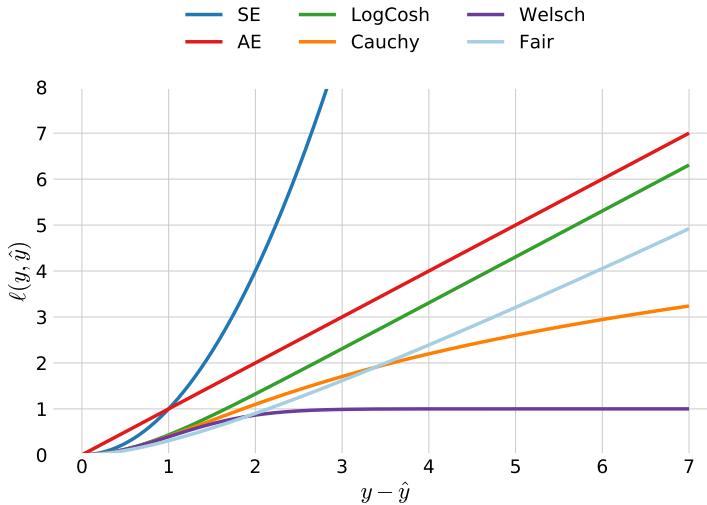


Figure 39.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$. In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure ???. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

39.5.1 Evaluation Function

Since some machine learning models require an analytic expression for the derivative and second derivative, it not always possible to use a custom objective function if it is non-differentiable. The Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) is such a measure. Another example would be the width of the distribution of all $y_i - \hat{y}_i$. In this thesis this overall performance metric will be called the *evaluation function* f_{eval} compared to the differentiable proxy for this function, the objective function. To sum up: the loss function $\ell(y, \hat{y})$ measures the loss for an individual prediction and the objective function \mathcal{L} is the aggregated version of the individual losses. The objective function is assumed to be a good proxy for the evaluation function.

For the loss functions defined in ?? the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell(y_i, \hat{y}_i) + \Omega. \quad (39.28)$$

39.6 Decision Trees

Decision Trees are a simple machine learning method that works by partitioning the feature space into smaller subspaces, high-dimensional rectangles basically, and then fit each subspace with a constant for regression problems or a single label for classification problems [?]. A simple example of this can be seen in Figure ?? and Figure ???. In the first figure we see an illustration of how the signal and background distributions look in the 2D feature space. The dashed lines in the figure indicate the cuts made by the decision tree (DT), cuts which are shown on the second figure as a typical DT plot. Here the first “box” is called the *root* of the tree,

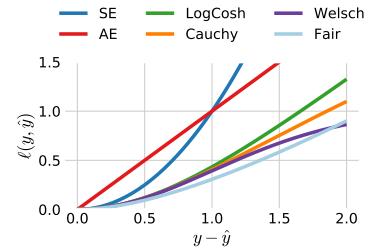


Figure 39.9: Zoom in of Figure ??.

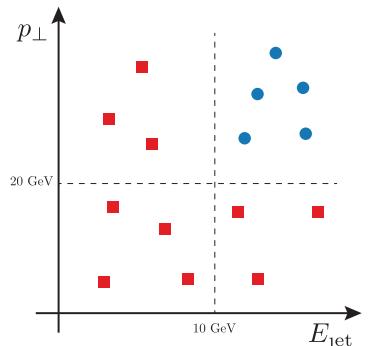


Figure 39.10: Illustration of the cuts a decision tree model make for *signal* in blue circles and *background* in red squares. This is an visualization in the feature space of the decision tree seen in Figure ??.

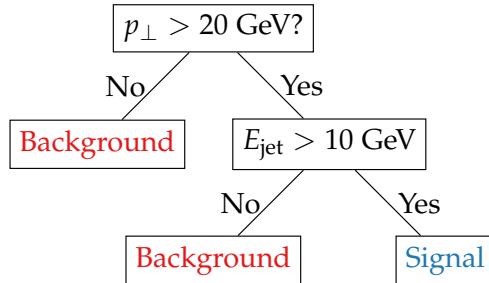


Figure 39.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables p_{\perp} and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure ??.

any subsequent boxes are *nodes* except the final ones that are not split any further which are *leaves*.

At first the DT partitions the space according the the value of the transverse momentum²⁴ p_{\perp} : if it is lower than (or equal to) 20 GeV then it classified as background. If the value is higher than 20 GeV then an extra split is made, this time on the energy of the jet E_{jet} : if it is higher than 10 GeV it is classified as signal and otherwise as background. Training a DT on this data allows us to predict a new unseen event $(p_{\perp}, E_{\text{jet}}) = (24 \text{ GeV}, 11 \text{ GeV})$ to be a signal-like event. This DT is said to be a shallow tree since it only has a depth of 2. The maximum depth allowed for the model is an important hyperparameter since it clearly controls under- and overfitting by changing how many cuts and partitions in the feature space are allowed; the deeper the tree, the more complex the model becomes. Single DTs are very prone to overfitting²⁵, however, they are also extremely inspectable. They are even referred to as “white-box models” compared to black-box models such as neural networks. For a more thorough introduction to decision trees and how they are internally optimized (for finding the best cut values), see [?].

39.6.1 Ensembles of Decision Trees

Single decision trees are prone to overfitting and generally suffer from high variance. Today especially two different methods exist to alleviate these problems: Random Forests (RFs) and Boosted Decision Trees (BDTs). Both methods are examples of so-called ensemble methods where a finite set of ML methods are combined into a single model. Typically ensemble methods are based on *weak learners*: simple, often fast, methods that individually show relatively poor generalization performance typically due to variance.

Random Forests

Random Forests were first introduced in 2001 by [?]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average²⁶ the individual trees’ predictions T_b :

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x}). \quad (39.29)$$

²⁴ All units in this project are in natural units such that both momentum and energy are in units of eV.

²⁵ With a solution to this problem given in ??.

²⁶ In case of classification, it is the majority vote which is taken.

The method of making artificial extra samples and training on them is in general called bootstrap aggregation or *bagging*[?]. It works by averaging out noisy estimates of the individual models hence reducing variance.

Theorem 40 (Variance of average of correlated i.d. variables). *Given B identically distributed (but not necessarily independent) variables each with variance σ^2 and positive pairwise correlation ρ , the variance of the average is:*

$$\text{Var}(\bar{\mu}_B) = \frac{1 - \rho}{B} \sigma^2 + \rho \sigma^2, \quad (39.30)$$

where $\bar{\mu}_B$ is the average of the i.d. variables [?].

Equation (??) is the main idea behind RFs. As the number of trees B in the forest increases, the first term goes towards zero. Thus, the more the correlation ρ between the individual trees is reduced, the more the variance is reduced (which is the main problem in the case of DTs to begin with).

In addition to training the trees on bagged samples, one more technique is used to further decrease the correlation between the individual trees: each bootstrapped sample of the dataset not only contains a only subset of the observations (rows), but also only a subset of the variables (columns)²⁷. This called column-subsampling (in contrast to row subsampling).

Boosted Decision Trees

In the overall family of ensemble models, *boosting* might be the most successful of them all where especially the specific algorithm called XGBoost [?] revolutionized the ML world by winning numerous Kaggle²⁸ competitions [?] including the Higgs Machine Learning competition in 2014 [?].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [?]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the M different weak learners F_m :

$$f_{\text{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{m=1}^M \alpha_m F_m(\mathbf{x}) \quad (39.31)$$

Boosting thus works by reducing both bias and variance since it iteratively fits weak learners. The variance is not reduced as much as for RFs since the weak learners are more correlated, however, their bias is lower.

The term gradient boosting comes from the observation that repeatedly minimizing the residuals of the current model is similar to minimizing the gradient of the loss function for a specific choice of the loss function.

Imagine that we start off with an imperfect model $F_m(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so image that the perfect addition to the model

²⁷ Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by [?] which basically states that each variable forms a columns, each observation forms a row, and that each type of observation unit forms a table. This also means that the term variable and column will be used interchangeable (together with *feature*), and likewise with observation and row.

²⁸ XXX TODO!

that we needed to make was $h(x)$. For it to be perfect, the following would have to be true:

$$F_{m+1}(x) = F_m(x) + h(x) = y \Leftrightarrow h(x) = y - F_m(x). \quad (39.32)$$

The r.h.s. is the residual of the model, so at each stage the model is trying to fit the residuals of the current iteration of model. The “gradient” in “gradient boosting” comes from the following. Assume the loss function: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$. The gradient of the loss w.r.t. to $F_m(x)$ is:

$$\frac{\partial L(y, F_m(x))}{\partial F_m(x)} = F_m(x) - y. \quad (39.33)$$

This is exactly the negative of the r.h.s. of equation (??). Instead of looking at the model as trying to minimize the residual at each iteration, it can instead be generalized as trying to fit the negative gradients of the loss function:

$$h(x) = -\frac{\partial L}{\partial F_m}. \quad (39.34)$$

We thus end up with the following iterative model which is basically a *gradient descent* algorithm.

$$F_{m+1}(x) = F_m(x) + h(x) = F_m - \frac{\partial L}{\partial F_m}. \quad (39.35)$$

AdaBoost [?] was the first major algorithm to make use of boosting. It was seen as a way of iteratively giving wrongly predicted observations higher weight, however, this is just a result of a “lucky” choice of loss function²⁹ which was not realized until much later. AdaBoost could be used with many different weak learners, however mostly DTs were used to form BDTs. XGBoost [?] is a fast, computationally efficient implementation of gradient boosting with DTs as base learners. It implements several model regularization techniques which makes it less prone to overfitting than other BDTs.

²⁹ Exponential loss for classification.

39.7 Hyperparameter Optimization

By now linear models with L_1 and L_2 regularization have been introduced along with decision trees (DTs), random forests (RFs) and gradient boosted trees (GBTs). All of these ML models tries to optimize their parameters according to some objective function. In addition to the parameters of the model, each one has a specific set of *hyperparameters* that cannot directly be optimized in internal optimization process. This could be amount of regularization λ for linear models, the maximum tree depth for DTs, the number of trees for RFs, or the column (or row) subsampling fraction for BDTs.

In general we say that we have the ML model \mathcal{A} with K hyperparameters. Each of these hyperparameters have a domain Λ_n . The

domain can be either real numbers $\Lambda_k \in \mathbb{R}$, integers $\Lambda_k \in \mathbb{Z}$, binary $\Lambda_k \in \{0, 1\}$, or categorical³⁰. We define the hyperparameter configuration space as: $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\lambda \in \Lambda$ which defines the optimal model \mathcal{A}_{λ^*} . Here the optimal model is the defined as the model which gives the best generalization performance according to some evaluation function. The goal of finding the best hyperparameter λ^* is known as *hyperparameter optimization* (HPO).

The first naive approach would simply to manually³¹ try out different combinations of λ and see performance on the validation set³². This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In ?? the HPO method called Grid Search is introduced which is further generalized and optimized in ?? with Random Search. Both these methods are easily parallelizable since they do not have any inherent history in its guesses. This is in contrary to Bayesian Optimization introduced in ?? which allows for “smart” guesses.

39.7.1 Grid Search

Grid Search (GS) is a HPO method also known as full factorial design. It is called this because it tries out all possible combinations of the hyperparameter configuration space: the so-called cartesian product of Λ . Imagine a 2D space where the two domains are respectively $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$. Then GS runs through all 12 combinations of these two sets:

$$\{(1, 1), (1, 2), \dots, (x_i, y_i), \dots, (3, 4)\}, \quad (39.36)$$

as visualized in Figure ???. The advantage of GS is that it is an exhaustive search over all combinations³³ of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality³⁴.

39.7.2 Random Search

To circumvent the problems of grid search, ?] developed the Random Search (RS) algorithm in 2012. Regarding the effect of the curse of dimensionality on grid search they wrote: “*This failure of grid search is the rule rather than the exception in high dimensional hyperparameter optimization*” [?]. Instead of searching through all possible values of λ like in GS, RS makes B runs where each λ_i is given by:

$$\lambda_i \sim \sum_{j=1}^K \text{PDF}_j(\Lambda_j) \cdot \hat{e}_j. \quad (39.37)$$

Equation (??) should be understood in the following way. For each hyperparameter draw a random number from a user-defined Probability Density Function (PDF) and then let λ be the vector of those

³⁰ Could e.g. be the choice of loss function.

³¹ Also known as jokingly as “Grad Student Descent”.

³² Remember only to use the test set on the final model!

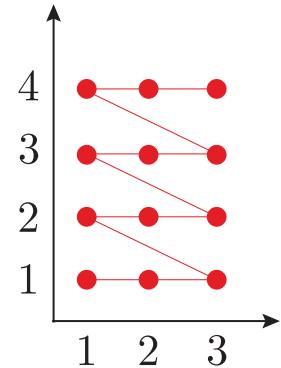


Figure 39.12: Visualization of grid search run on the two hyperparameters x and y with the domains $x = \{1, 2, 3\}$ and $y = \{1, 2, 3, 4\}$.

³³ Note that the user has to provide the values for each hyperparameter to be tried out manually.

³⁴ Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

N random numbers. In a 2D-space λ_i could thus be:

$$\lambda_i \sim \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \quad (39.38)$$

where $\mathcal{N}(100, 2)$ is normal (Gaussian) distribution with mean $\mu = 100$ and standard deviation $\sigma^2 = 4$ and $\mathcal{U}(0, 1)$ is the uniform distribution in the interval $[0, 1]$. Of course the PDF can be a PMF in the case of discrete hyperparameter domains.

The reason why random search is so powerful is not only because the number of function evaluations B is easily tunable³⁵, but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space Λ might be high-dimensional, it often exhibits *low effective dimensionality* [?]. In the simplest 2D-case this can be written as the following example. Imagine that we want to maximize some evaluation function, e.g. accuracy of the predictions, and the model depends on the two independent hyperparameters x and y : $f(x, y)$. In this example assume that f is almost insensitive to y and thus has an effective dimensionality of 1. Then $f(x, y) = g(x) + h(y) \approx g(x)$. For a visualization of this example, see Figure ??.

Here GS is run with a grid of $3 \times 3 = 9$ points and RS is similarly run with 9 points drawn from uniform PDFs in the same interval. It is easy to see that when the hyperparameter configuration space has a lower effective dimensionality than the actual dimensionality RS is far better at probing the space due to the projections into the sensitive dimensions cover more of these axes than for GS. In general in ML the hyperparameter configuration space has lower effective dimension than its actual dimension, but the different hyperparameters matter in different datasets

In general only a fraction of all hyperparameters matter for any dataset but different hyperparameters matter in different datasets and thus generally RS performs as well as GS or better [?].

Note that RS can be seen as a generalization of GS, where GS is the specific example of RS if one uses a multidimensional hypergeometric distribution as PDF where the PDF is reevaluated after each run.

39.7.3 Bayesian Optimization

When optimizing the hyperparameters it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting \mathcal{A}_λ to the training data and then measure the performance on the validation set. Fitting the model on the training set can often take minutes, if not hours. This process is even slower when using cross validation. The idea behind Bayesian Optimization is that when the ML model, or any other black box function, is expensive³⁶ to evaluate then “smart” guesses are worth spending a bit of time on developing. The hope is that the time

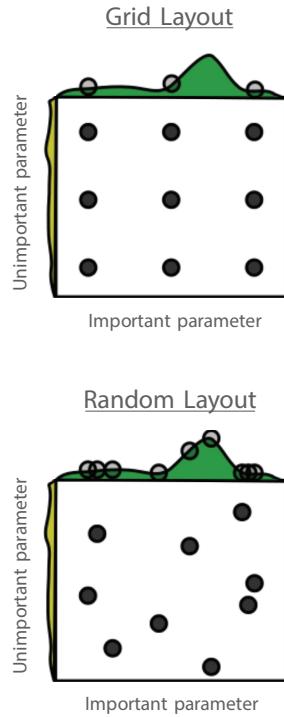


Figure 39.13: Visualization of the difference between grid search and random search. Adapted from [?].

³⁵ Compared to grid search which tries *all* possible combinations.

³⁶ With respect to time.

taken to come up with smart guesses is negligible compared to the overall function evaluation time. This is contrary to both GS and RS where each new set of hyperparameters λ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [?], the (unknown) evaluation function is iteratively fitted with a probabilistic surrogate model, most often by Gaussian processes (GPs). Given the fitted surrogate model, an acquisition function is computed. This function is cheap to evaluate and is a measure of where in the hyper-dimensional hyperparameter space there is a highest chance of finding a new good value of λ . The acquisition function has to be chosen manually and especially the tradeoff between *exploitation versus exploration* is particularly important. This value decides how “adventurous” or conservative the BO algorithm should be when exploring the evaluation space.

Bayesian Optimization is better explained by looking at Figure ???. First look at the top plot. This is a plot of the surrogate function in black with uncertainties shown in blue. This is a result of fitting GPs to the two previous points in black, $t = 2$. This surrogate function is supposed to fit the evaluation function (called objective in the figure) shown as a dashed black line. Below we see the acquisition function in green. This is a function of the blue curve and the position of its maximum decides where the next guess of λ should be. With the chosen acquisition function and exploration willingness, we see that the next guess should be slightly to the left of the right-most point. This is a simple 1D toy problem, but one should imagine this happening in a high-dimensional space. After making a new guess, $t = 3$ in the middle plot, the acquisition function changes since it learnt that this gave a worse evaluation value than the right-most point. Therefore, the next proposal for λ is slightly to the right of the right-most point. The process continues like this in an iterative fashion: first fitting GPs to the previous evaluation values and then choosing the next λ according the acquisition function given the GPs.

Gaussian Processes provide a posterior distribution given some prior distribution and the data-dependent likelihood. The process of BO is quite technical and mathematical, especially if GPs are new material. For a more in-depth explanation of the topic, see ?]. The important thing to note is that GPs return not only a posterior mean $\mu(\mathbf{x})$ but also an uncertainty $\sigma(\mathbf{x})$, as seen in Figure ?? described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad (39.39)$$

where $\kappa \geq 0$ is the parameter³⁷ controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being

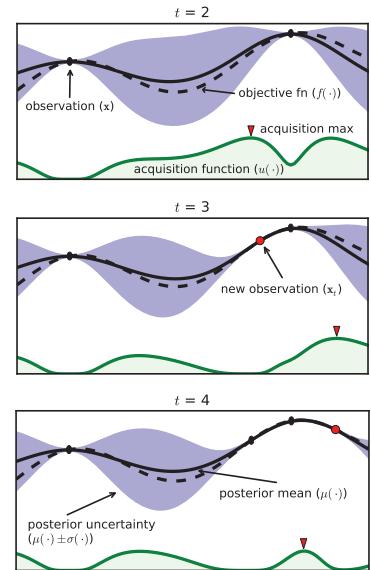


Figure 39.14: XXX TODO!. Adapted from ?].

³⁷ Here κ can thus be regarded as a hyper-hyperparameter since it controls how the other hyperparameters are optimized.

harder to numerically implement compared to GS and RS³⁸, and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

39.8 Feature Importance

Having first established in ?? that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (??), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in ?? and they were hyperparameter optimized in ??, one would expect that one would have a well performing model by now.

Now comes one of the most important issues in ML today: model inspection. Actually trying to make sense of the learnt model. Why does it predict as it does? Which features or variables are most important according to the model? Model interpretation is still very much active research today with no universally accepted methods. Some methods are model dependent and accurate, others might be model agnostic but slow or only approximations. In this project the focus will be on the so-called *SHAP* values.

In 2017 ?] showed that six different previously used methods were all specific instances of a universal underlying method³⁹ and propose SHapley Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018-2019 they developed a fast algorithm for computing SHAP values for tree ensembles and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [?]. That a measure for feature importance is inconsistent means that a model could rely more on feature *A* than *B*, however, the feature importance would indicate opposite. SHAP values and *permutation* feature importance are both consistent feature importance measure, however, only SHAP allows for individualized⁴⁰, or local, feature importances.

SHAP values are within the class of additive feature attribution methods, which are functions where the explanation model g is a linear combination of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (39.40)$$

Here $\phi_i \in \mathbb{R}$ are the feature importances and z' is a binary variable such that $z'_i = 1$ if the feature is present and otherwise $z'_i = 0$. SHAP values are based on Shapley regression values known from cooperative game theory [?]. These values are based on the three axioms:

Axiom 37 (Local Accuracy). *Local accuracy says that the sum of the feature importances should equal the total reward:*

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i. \quad (39.41)$$

³⁸ Which are basically plug-and-play with Scikit-Learn [?].

³⁹ The class of *additive feature attribution methods* [?].

⁴⁰ Meaning that you can get the feature importances for a single observation compared to only the global, overall feature importances as seen across the entire data set.

Here f is the ML model, g is the explanation model, x is an observation in input feature space, z' is an observation in the binary space as described above, and ϕ_i is the feature importance.

Axiom 38 (Missingness). *Missingness means that features missing in the original input feature space (such that $z'_i = 0$) should be attributed no importance:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \quad (39.42)$$

Axiom 39 (Consistency). *Consistency states if a model is changed such that it relies more on a certain feature, the feature importance of that feature should never decrease.*

Given these three axioms, [?] show that the only solution to equation (??) is:

$$\phi_i = \sum_{S \subseteq \tilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (39.43)$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \tilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv [f_x(S \cup \{i\}) - f_x(S)]. \end{aligned} \quad (39.44)$$

In equation (??) \tilde{M} is the set of all input features⁴¹, $S \subseteq \tilde{M} \setminus \{i\}$ means a subset S of \tilde{M} without feature i , $S \cup \{i\}$ means the set S with feature i and $f_x(S) = f(h_x(z'))$ where $h_x(z')$ is the mapping function from the binary z' space to the input feature space x . In equation (??) the function is simplified to its basic constituents: the difference in performance between including feature i and not including it, Δ_{f_x} , and its weight w .

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure [?]. Here \tilde{M} would be $\tilde{M} = \{p_\perp, E_{jet}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \tilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{jet}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} [f_x(\{p_\perp\}) - f_x(\{\})] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} [f_x(\{E_{jet}, p_\perp\}) - f_x(\{E_{jet}\})]. \end{aligned} \quad (39.45)$$

Whereas $w(S)$ are easily calculated, Δ_{f_x} depends on the data. As the number of features grows, the number of terms in the sum grows exponentially. What [?] did was to develop an efficient algorithm that could solve this for trees in polynomial time⁴².

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data

⁴¹ Compared to $M = |\tilde{M}|$ which is the number of all input features.

⁴² Specifically they managed to improve the time complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$ where T is the number of trees, L is the maximum number of leaves in any tree, M is the number of features, and D is the maximum depth of any tree (where $D \approx \log L$ for balanced trees).

set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each of with M features, one gets the matrix Φ . When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [?]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^N |\Phi_{i,j}|. \quad (39.46)$$

The global feature importance ϕ_i^{tot} is thus a measure of the overall importance of feature i .

Note that if one introduces a new feature to the dataset, correlated⁴³ to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

Axiom 40 (Symmetry). *If for all subsets S that do not contain i or j :*

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \quad (39.47)$$

then $\phi_i = \phi_j$.

It can be shown that axiom ?? is implied by axiom ?? [? , Supp. Material]. Two identical features, as seen by the model, will thus “share” the feature importance.

⁴³ Not necessarily linearly correlated.

40. Danish Housing Prices

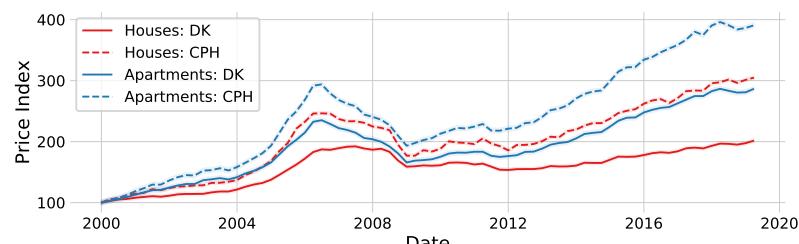
"Buy land, they're not making it anymore."

— Mark Twain

"It's tangible, it's solid, it's beautiful. It's artistic, from my standpoint, and I just love real estate."

— Donald Trump

HOUSING MARKETS have always been a playing field for economists, property speculator, real estate developers, and realtors. The author of this thesis is by no metric any of these, not even close to, yet, when the issue of estimating Danish housing prices came up, it was too much of a challenge just lying there to let it go. Estimating housing prices is a classical economical discipline with papers from the Danish National Bank developing a regional model of the Danish housing market [?] to analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [?].



If one takes a look at the time development of the Danish housing¹ market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [?] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, this is shown in Figure ???. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market since then in both Copenhagen and the entire country.

The goal of this subproject² is not to predict any future collapse as the financial markets as we saw upwards of 10 years ago. Instead, it is to learn patterns in the price of houses in steady times. The goal is training a computer³ to automatically to find these patterns and see if we can improve this model.

Figure 40.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where **houses** are shown in red and **apartments** in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from [?], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

¹ "Housing" means both actual houses and privately owned apartments in this project.

² The housing price estimation sub-project.

³ In contrary to [?] and others who base their models on macro-economic principles.

In chapter XXX I will introduce the data, do EDA and so on. Then I will introduce the models in XXX, start the HPO in chapter XXX, and talk about the results in XXX, the SHAP values in XXX and the time predictions in XXX.

This subproject was done in collaboration with Boligsiden without whom it would not have been possible. During this project, common python data science tools from the SciPy ecosystem[?] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

40.1 Data Preparation and Exploratory Data Analysis

"80 % of data science is cleaning the data and 20 % is complaining about cleaning the data."

— Anthony Goldbloom, Kaggle

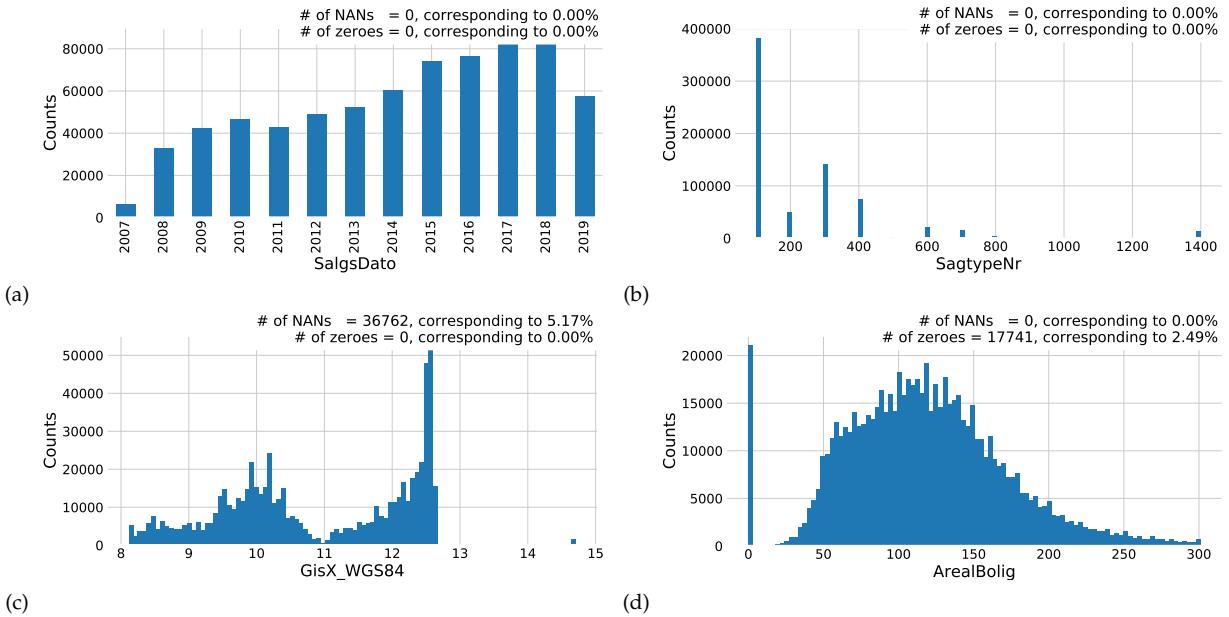
The first part of any data science project is actually getting the data and being able to read it. This has been an iterative process that has improved over time. The last data transfer we got was September 3rd 2019 and consisted of a 522.4 MB CSV file with dimensions (711 212, 171). This section will go through the data cleaning process.

Before any further data analysis is performed, all of the data is loaded, except three columns which only contain internal information for Boligsiden⁴. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

Four particularly interesting ones are seen in Figure ??: the distribution of the date of the sale `SalgsPris` in subfigure ??, the distribution of the type of residence `SagtypeNr` in subfigure ??, the distribution of the longitude of the residence `GisX_WGS84` in subfigure ??, and the distribution of the area of the residence `ArealBolig` in subfigure ??.

The distribution of the date of sale, Figure ?? ??, is an interesting variable because it shows how Boligsiden has been collecting more and more data over time. Here 2007 and 2019 are clear outliers since their current database only contains sales from the end of 2007, and 2019 only contains data from the first eight months of the year. The type of residence is a discrete code that Boligsiden uses to differentiate between different types of residences. The mapping between code and description can be seen in Table ??.

⁴ The variables are `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.



In this project only one-family houses – “Villas” in Danish – with code 100 and owner-occupied apartments – “Ejerlejlighed” in Danish – with code 300 are considered. As can be seen from Figure ?? ?? these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure ?? ??, is mostly interesting due to fact that it clearly shows how the Great Belt and especially the Baltic Sea separates Denmark into three parts; the Western part, the Eastern part, and then Bornholm. Note that more than 5 % of the residences’ locations are unknown values: Not A Number “NANs”. The distribution of the area, Figure ?? ??, shows that most residences are between 50 m² and 200 m², as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m². All of the 1D-distributions can be seen in Figure ??-??.

The geographic distribution of sales can be seen in Figure ???. The residences are coloured according the square meter price in Figure ?? ?? and according to the sales price in Figure ?? ???. Notice the strong correlation between the distance to water and the square meter price, a correlation that is less visible when looking at the sales price. Since these plots each contain 674 647 points⁵, over-plotting quickly becomes an issue. To circumvent this, the great software package called DataShader was used [?] which in a simple, consistent, and not at least fast manner allows one to plot big data.

The most important of the features is the sales price, called `SalgsPris` in the dataset, and its distribution shown in Figure ???. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode of the distribution is 1.1 M.kr. and the median⁶ is 1.6 M.kr., but remember that this variables is shown for all types of residences.

Figure 40.2: Distributions of four out of the 168 input variables. Subplot ?? shows the date of the sale, Subplot ?? shows the type of residence, Subplot ?? shows the longitude, Subplot ?? shows the area of the house.

200	Rækkehus
300	Ejerlejlighed
400	Fritidsbolig
401	Kolonihave
500	Andelsbolig
600	Landejendom
700	Helårsgrund
800	Fritidsgrund
900	Villalejlighed
1000	Kvæggård
1100	Svinegård
1200	Planteavlsgård
1300	Skovejendom
1400	Lystejendom
1500	Specialejendom

Table 40.1: XXX TODO!

⁵ Only sales with a valid GPS-coordinate and area of residence are shown

⁶ The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

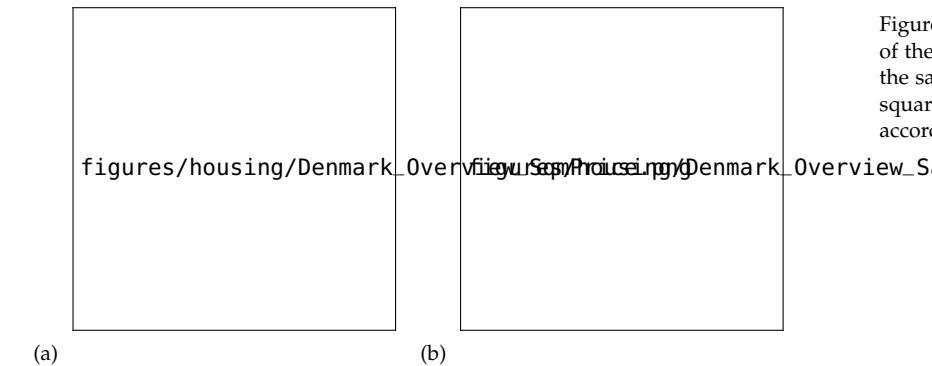


Figure 40.3: Geographic distribution of the sold residences. In subplot ?? the sales are colored according to their square meter price and in subplot ?? according to the sales price.

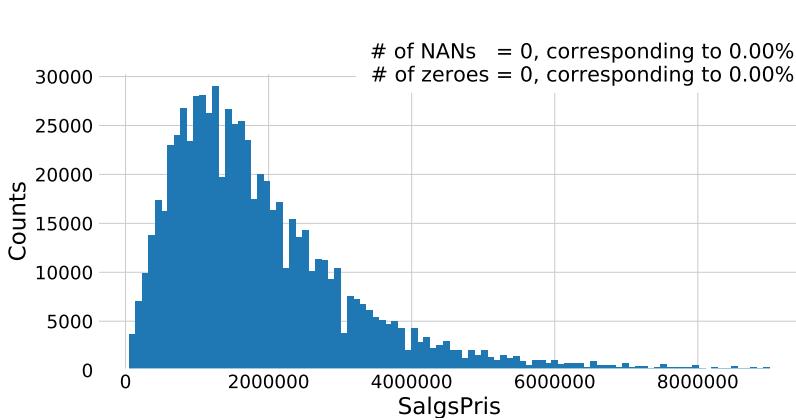


Figure 40.4: Histogram of prices of houses and apartments sold in Denmark.

40.1.1 Correlations

Having shown the 1D-distributions of all the different variables in the previous section, the next step would be to look at the correlations between the variables. Since there are 168 input variables, it is almost impossible to understand every inter-variable correlation, however, it is tried in Figure ?? in the appendix. Here the correlation between all numerical variables that are not obviously related to other variables (like the GPS-coordinates that are in both latitude-longitude and ETRS89⁷ format), and with the condition that it has to have one inter-variable correlation higher than $|\rho| > 30\%$, are plotted as a (86×86) -dimensional heatmap.

Even though inter-variable correlations are important in the exploratory data analysis (EDA) phase, what is more important is to get a better understanding of how the input variables correlate with the output variables; the sales price. This is shown in Figure ?? for the variables where $|\rho| > 10\%$. It is the previous property evaluations, `EjdVurdering_EjendomsVaerdi` that are positively correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership⁸, area, number of bathrooms, its longitude, and distance to nearest wind mill. In the other end, the local income tax, `Kommune_SkatteProcent`, is the variable that is the most negatively correlated to the sales price, followed by the geographical variables related to province, municipality, and postcode.

The correlation ρ used above is the linear correlation which thus only captures linear relationships between variables. All modern

⁷ European Terrestrial Reference System 1989.

⁸ This is a great example of the fact that correlation does not imply causation

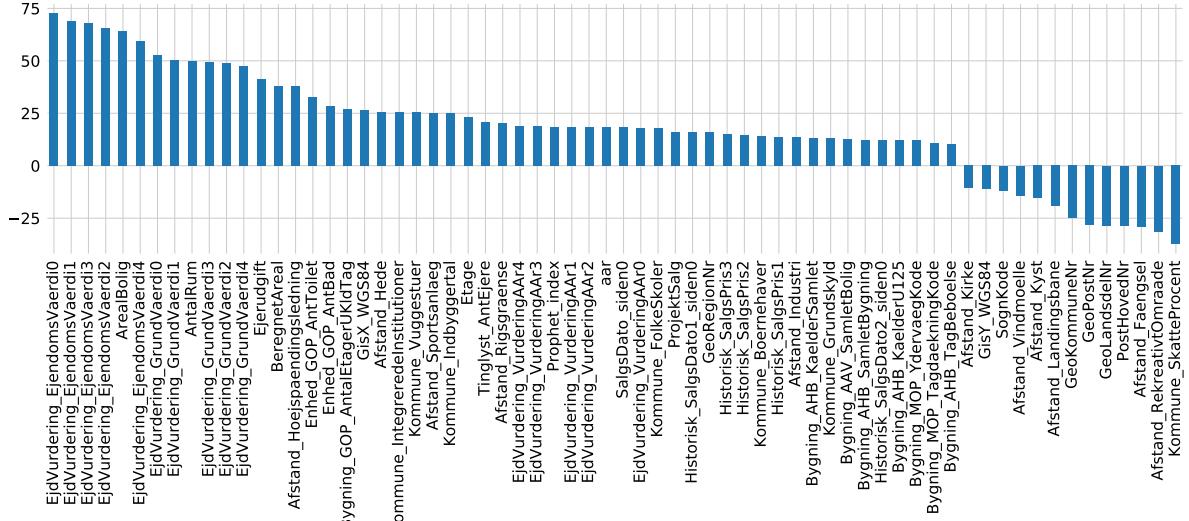


Figure 40.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\%$.

machine learning algorithms, however, are also able to capture higher-order correlations and thus a higher-order correlation measure is needed. The maximal information coefficient (MIC), which is a value between 0 and 1, is such a non-linear measure. It finds correlation based on the intuition that if two variables are correlated, it should be possible to split the data up into smaller grids where, if they are correlated, the grid that contain points should contain many points and the rest of the grids should be (relatively) empty [?]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. [?] extended on this idea and developed the computationally efficient algorithm called `MICtools` which computes the estimator for MIC: MIC_e . An example of this tool non-linear correlation is seen in Figure ???. Here the relationship between the normal linear correlation ρ and MIC_e can be seen for four synthetic datasets. Notice that MIC_e does it particularly well for the sine wave, and decent for the line and parabola, but only slightly captures the relationship for the exponential growth. The influence of noise on ρ and MIC_e can be see in Figure ?? in the appendix.

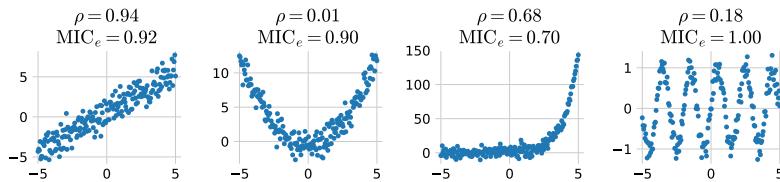


Figure 40.6: MIC non-linear correlation. XXX **TODO!**

Using MIC_e as the correlation measure between the numerical variables and the sales prices, the variables with a MIC_e -score higher than 10 % can be seen in Figure ???. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geo-

graphical variables score high here, with also the post code, municipality number, and longitude.

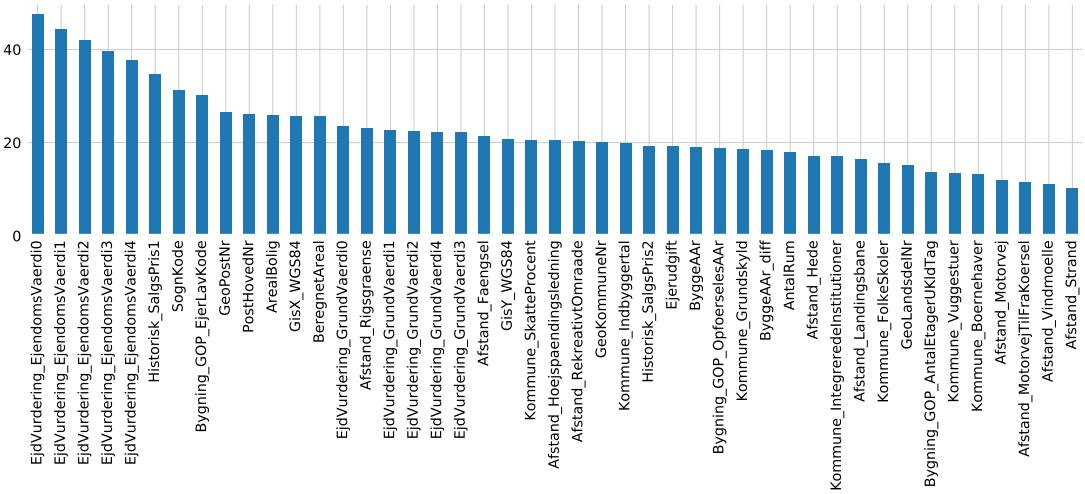
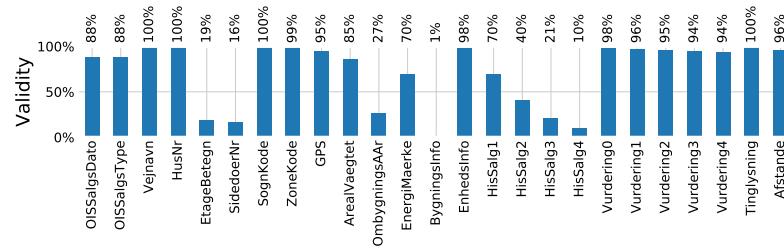


Figure 40.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where MIC > 10%.

40.1.2 Validity of input variables

The fact that some of the variables contains considerable amount of invalid values, NaNs, requires this to be taken into account before any further analysis. The validity, defined as the percentage of valid observations, of every variable is shown in Figure ???. Here the 168 variables variables are grouped together into 25 variables where each group share the same validity. An example of this are all of the 16 different distance-variables⁹. We see that most of the variables have validities around more than 85 %, however, a few of the variables, especially information about the building, `BygningsInfo`, have validities less than 20 %.



To see how closely related the different validity groups are, one can look at the dendrogram in Figure ???. The dendrogram is based on a hierarchical clustering algorithm [?] where the different groups are clustered according to their linear correlation. This diagram is supposed to be read in a top-down approach, where it can be see that the name of the street, `Vejnavn`, and the number of the residence, `HusNr`, correlate a lot and are thus clustered very early. The year of the last time the residence was rebuilt or greatly modified, `OmbygningsAAr`, does not correlate strongly with any of the other variables and is thus the last variable to be clustered together.

⁹ Distance to: Fængsel, Hede, Højspaerdingsledning, Industri, JernbaneSynlig, Kirke, Kirkegård, Kyst, Landingsbane, Motorvej, MotorvejTilFraKørsel, RekreativtOmråde, Rigsgrense, Sportsanlæg, Strand, Vindmølle

Figure 40.8: Percentage of valid counts for each variable grouped together in categories.

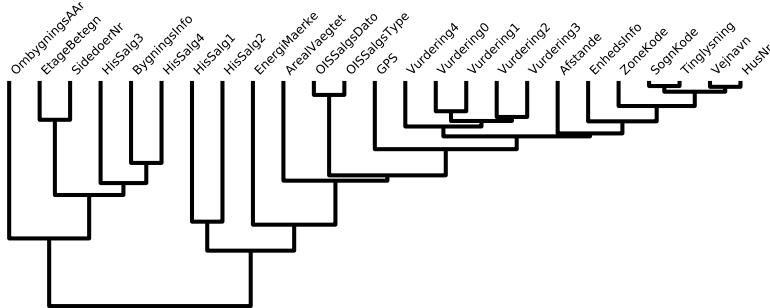


Figure 40.9: Validity Dendrogram
TODO!

To see a heatmap of the inter-variable correlations, see Figure ?? in the appendix.

40.1.3 Cuts

Given the 1D input variable distributions and their validity, we apply some very basic cuts before any further analysis. These cuts are seen in Table ?? . Sales type, `OISSalgsType`, is a OIS¹⁰ code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared e.g. forced sales. These cuts are seen as the minimum requirements for what constitutes a curated dataset with no obvious outliers. The reason why the time requirement is applied, is to reduce the effect of the financial crisis to creep into the model.

	Description	Remaining	Removed
Area	$20 \text{ m}^2 \leq \text{Area} \leq 500 \text{ m}^2$	689 140	23 666
Price	$0.1 \text{ M.kr.} \leq \text{Price} \leq 100 \text{ M.kr.}$	687 546	1 594
Type	Has sales type 1	605 415	82 131
GPS	Has valid GPS coordinates	578 860	26 555
Private	Only non-business sales	549 140	29 720
Time	Sold in 2009 or later	520 548	28 592

¹⁰ OIS is short for “Den Offentlige Informationsserver”, the Danish public information server, and it collects information about Danish residences [?].

Table 40.2: XXX TODO!

40.2 Feature Augmentation

Until now all of the analysis have dealt with different types of residences all together. From now on, the rest of the analysis will be applied on single-family houses and owner-occupied apartments independently.

First invalid counts are dropped, such that variables which contain more than 10 % NaNs are dropped, and duplicate rows are also removed. Then some manual features are added based on the day of the month, the month, and the year are extracted from the sales date and the sales date is also represented as both a float and as the numbers of days since January 1st, 2009. From the number of the house, `HusNr`, the number is extracted along with a boolean flag indicating whether or not it includes a letter (eg. “27B”). The number of the side door, `SidedoerNr`, is formatted according to

String	Explanation	Code
NAN	No side door	0
TH, TV	Right, left	11
MF	Center	12
-	The rest	15

Table 40.3: XXX TODO!

Contains	Explanation	Code
Vej	Road	0
Gade	Street	1
Alle, Allé	Avenue	2
Boulevard	Boulevard	3
-	The rest	-1

Table 40.4: XXX TODO!

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50

Table ?? and the road name is according to Table ???. The age of the house is added¹¹ and the amount of time (in year) since last major modification. The energy rating label, `EnergiMaerke`, is also converted from strings to values according to Table ??

Finally, some of the variables in the dataset are not suitable for machine learning or simply transformations of other variables. These are variables such as the ID, when the house was deleted at Boligsiden, or the cash price, since we want the model to learn the price of the house and not simply the efficiency of the realtor.

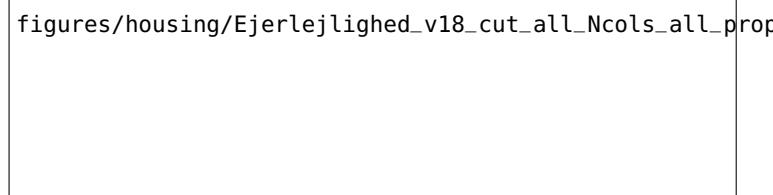
40.2.1 Time-Dependent Price Index

In addition to the manual data augmentation in ??, a time-dependent price index is also added. We make use of the open source package called Prophet made by [] at Facebook. It is based on a decomposable time series model [?] with two¹² components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \quad (40.1)$$

where ϵ_t is a normally distributed error term. [] fit this equation with a generalized additive model (GAM) [?] which they argue has several practical advantages compared to ARIMA models¹³.

We fit the Prophet model on the weekly median price pr. square meter (PPSM) up until (and including) 2017. The results of the prophet model fitted on OOAs are seen in Figure ?? and Figure ???. In Figure ?? the weekly median price pr. square meter for OOAs are shown as black dots with the fitted Prophet model shown in blue. The 1σ uncertainty intervals are shown as the transparent blue band. The Prophet model not only allows predicting previous and future PPSMs, it also return the uncertainty of this prediction. The future predictions for the PPSM are the values after 2018. The trend and seasonality of the model are shown in Figure ?? where the top plot is the overall trend $g(t)$ and the bottom plot is the seasonality $s(t)$. Whereas the trend just continues to rise, the seasonality shows that residences are generally sold for a higher price in the Summer months compared to the Winter months.



`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_forecast.png`

¹¹ In addition to only having the year the house was built

¹² In their paper, [] include a holiday component in their analysis as well which is not included in this project.

¹³ AutoRegressive Integrated Moving Average

Figure 40.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median price each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1-\sigma$ confidence interval.

The Prophet model plots for OFHs can be seen in Figure ?? and Figure ?? in the appendix.

`figures/housing/Ejerlejlighed_v18_cut_all_Ncols_all_prophet_trends.ipynb`

Figure 40.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

Using the Prophet model, we define the price index (PI) to be the Prophet-predicted PPSM, y_{Prophet} , for each residence normalized by the mean to give values around 1:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \quad (40.2)$$

where $\langle \cdot \rangle$ refers to the average. The price index thus works as a measure of the national price for houses or apartments at a given time and is added as a variable to the dataset.

40.3 Evaluation Function

The choice of evaluation function f_{eval} is an important decision. The evaluation function will be based on the relative prediction z :

$$z = \frac{\hat{y} - y}{y}, \quad (40.3)$$

where y is true price and \hat{y} the predicted one. The relative prediction is defined such that it is positive when $\hat{y} > y$, due to the outlier cuts made earlier the denominator is made sure to always be positive (and never 0), and z is expected to approximately follow a normal distribution. Initially the mean of z was considered as the choice of f_{eval} though this only ensures a minimum of bias, not necessarily a low spread. This lead the discussion on to look at the standard deviation of z as f_{eval} . The mean and the standard deviation, however, are not a very robust estimators since they are heavily influenced by outliers. The mean (and thus also the standard deviation) has a *asymptotic breakdown point* at 0 %, where the breakdown point is defined as the smallest fraction¹⁴ of bad observations that can cause an estimator to become arbitrarily small or large: a single outlier with an arbitrarily large value may cause the mean to diverge to that large value [?]. In comparison, the median has an asymptotic breakdown point of 50 % and is thus a more robust estimator of centrality. A robust measure of the variability or

¹⁴ Where the “asymptotic” in “asymptotic breakdown point” refers to when the number of samples goes to infinity.

dispersion of a sample \mathbf{x} – compared to e.g. the standard deviation σ – is the median absolute deviation (MAD) written as:

$$\text{MAD}(\mathbf{x}) = c \cdot \text{median}(|\mathbf{x} - \text{median}(\mathbf{x})|), \quad c = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)}, \quad (40.4)$$

where c is a normalization constant to make MAD a consistent estimator of the standard deviation σ assuming normally distributed data and Φ^{-1} is the percent point function¹⁵ [?]. The MAD is thus the median of the absolute differences between the data and the median of the data. We are, however, not just interested in having the distribution of the (relative) predictions as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

$$\begin{aligned} \text{MAD}_0(\mathbf{x}) &\equiv c \cdot \text{median}(|\mathbf{x} - 0|) = c \cdot \text{median}(|\mathbf{x}|) \\ f_{\text{eval}}(\mathbf{z}) &\equiv \text{MAD}_0(\mathbf{z}) = c \cdot \text{median}(|\mathbf{z}|). \end{aligned} \quad (40.5)$$

Here $\mathbf{z} \in \mathbb{R}^N$ is the vector of all relative price predictions. To get an intuition about the size of a “good” value of MAD_0 , one could calculate it comparing the asking price with the actual sales price. Doing so, one finds: $f_{\text{eval}}(\mathbf{z}_{\text{OFH}}) = 11.35\%$ for houses and $f_{\text{eval}}(\mathbf{z}_{\text{OOA}}) = 5.72\%$ for apartments. In some cases f_{eval} will still be referred to as MAD, however, it will be mentioned explicitly if the form in equation (??) is meant.

¹⁵ Inverse of the cumulative distribution function.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See ?] for more details.

40.4 Initial Hyperparameter Optimization

With the initial cleaning and feature adding done, the shapes of the ML-ready datasets are: (291 317, 144) for houses (OFHs) and (114 166, 144) for apartments (OOA), both sharing the same variables. All of the variables which are used from this point on can be seen in Table ?? in the appendix. The data are split into training and test sets such that training is defined as every sale from before 2018, every sale from 2018 is the test set, and since more data came since the project started, 2019 is a small extra test set.

The number of observations for the different sets can be seen in Table ???. Since the dataset has been shown to be quite noisy with a lot invalid counts, a “tight” selection of the data is also applied. The tight selection is defined as residences which are within the 1 % to 99 % quantiles of all¹⁶ numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table ??.

Before the data is properly fitted and the model is trained, a small study into the effect of some various hyperparameters was performed. This study investigated the effect of the old sales by assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples

	Houses	Apartments
Train	240 070	93 115
Test	34 628	14 183
2019	16 619	6868

Table 40.6: train test split XXX **TODO!**

Tight	Houses	Apartments
Train	143 179	57 795
Test	20 338	8376
2019	9683	4030

Table 40.7: train test split tight XXX **TODO!**

¹⁶ Except the variables that contains the words: “aar” (year), “dato” (date), and “prophet”.

got the time-dependent weight $w(t)$ given by:

$$\begin{aligned} w'(t) &= e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}} \\ w(t) &= \frac{w'(t)}{\langle w'(t) \rangle} \end{aligned} \quad (40.6)$$

where $T_{\frac{1}{2}}$ is the half-life.

This is illustrated in Figure ?? for the different values of $T_{\frac{1}{2}}$ used in the study. In addition to the weight, it was also investigated whether or not a \log_{10} transformation of the price would increase performance. The reasoning behind this would be that some machine learning methods assume that the dependent variable, y , is normally distributed. Finally the choice of loss function was also added to the study for the five different loss functions defined in ???. A grid search¹⁷ was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \quad (40.7)$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [?] (XGB) is used as the ML model. For apartments the loss function with the lowest f_{eval} was the Cauchy loss with $T_{\frac{1}{2}} = 20$ years and no \log_{10} transformation. This BDT terminated by early stopping after 932 trees, see Table ???. For houses the loss function with the lowest f_{eval} was the Fair loss (also) with $T_{\frac{1}{2}} = 20$ years and (also) no \log_{10} transformation. This BDT terminated by early stopping after 1996 trees, see Table ???. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation¹⁸. All of the results for the apartments can be seen in Table ??-?? in the appendix along with all of results for the houses in Table ??-??.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure ???. Here the hyperparameters (along the time taken and the number of trees) of the HPO are plotted along the abscissa (x-axis) and the value of the hyperparameter on the ordinate (y-axis). Every iteration of the HPO is thus a line on the plot. The lines are colored according to their evaluation score; the best hyperparameter is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses. The same plot for houses can be seen in Figure ?? in the appendix.

What can be concluded from Figure ?? is that there is a clear preference from not log-transforming the data, that the BDTs with many trees¹⁹ generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and

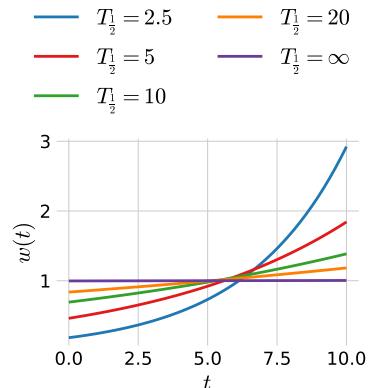


Figure 40.12: Time here is years after January 1st, 2009. XXX **TODO!**.

¹⁷ Grid search was acceptable since the parameter space is small and two of the three dimensions is non-numerical.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	293	0.1598
2.5	False	814	0.1466
5	True	304	0.1610
5	False	923	0.1468
10	True	266	0.1610
10	False	770	0.1450
20	True	288	0.1613
20	False	967	0.1467
∞	True	340	0.1601
∞	False	807	0.1480

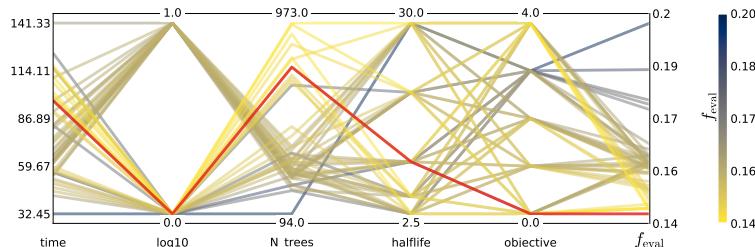
Table 40.8: Cauchy-ejerlejlighed.

Half-life	\log_{10}	N_{trees}	f_{eval}
2.5	True	434	0.1991
2.5	False	1007	0.1872
5	True	350	0.1999
5	False	1130	0.1858
10	True	436	0.1992
10	False	1183	0.1850
20	True	397	0.2003
20	False	1514	0.1833
∞	True	449	0.1992
∞	False	1351	0.1844

Table 40.9: Cauchy-villa.

¹⁸ The reason why the \log_{10} transformation is included even to begin with, was that initially it should better results than no transformation. However, this turned out to be a numerical consequence which was alleviated by dividing all prices with a million, such that y had units of M.kr. instead of kr..

¹⁹ Remember that the number of trees where selected by early stopping.



that there seems to be a tendency for the Cauchy loss to be the best, however, it is still ambiguous. What is not seen in the figure, however, are how the uncertainties²⁰ of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure ?? and ???. In the first of the two figures it is easily seen that even though $T_{\frac{1}{2}} = 10$ yr is the minimum, the uncertainties are so large that nothing can be concluded with regards to the half-life parameter related to the weights $w(t)$. In contrary, in the second figure there is a clear performance difference between the different loss functions where the Cauchy loss achieves the best (lowest) value of f_{eval} and especially the Squared Error is disregarded.

The rest of the machine learning models thus continue with the following hyperparameters:

	\log_{10}	Half-life	Loss function
Apartments	False	20 yr	Cauchy
Houses	False	20 yr	Cauchy

40.5 Hyperparameter Optimization

With the initial HPO set, the actual training of the models was started. An XGBoost model was fitted to the each of the two training sets, one for apartments and one for houses, where the hyperparameters were optimized using both random search and Bayesian optimization each run for 100 iterations with 5-fold cross validation and early stopping²¹. The hyperparameters to be optimized where the following:

The `subsample` varible controls the row-subsampling and is a number between 0 and 1.

The hyperparameter `colsample_bytree` controls the column-downsampling for each tree, so how many columns (or variables) are each tree allowed to fit to. Is a number between 0 and 1.

The `max_depth` controls the maximum depth of every tree. Is a positive integer.

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and

Figure 40.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD_0 in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True. The uncertainties here are the standard deviations (one standard deviation) and the 5-folds in the cross validation. The objective functions (XXX) (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses.

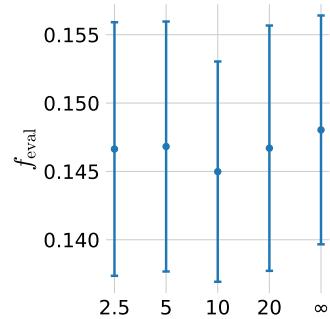


Figure 40.14: XXX Halflife $T_{\frac{1}{2}}$.

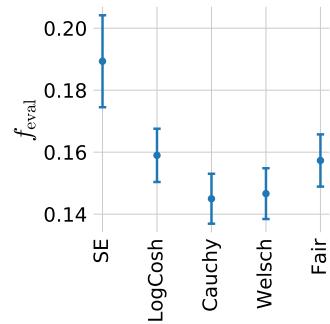


Figure 40.15: Objective function XXX.

²¹ This takes around 6 hours for both RS and GS for the apartments when run with 15 cores on HEP. XXX

will thus turn it into a leaf). Is a positive integer

`reg_lambda` controls the L₂ regularization term. Is a positive number.

`reg_alpha` controls the L₁ regularization term. Is a positive number.

The ranges of the hyperparameters were chosen by a manual, iterative process of fitting a subset of the data (1 %–10 %) and making sure that the best hyperparameter is not sufficiently close to the range; if it is, then the range is extended. The final HPO ranges chosen can be seen in Table ???. Here $\mathcal{U}(a, b)$ refers to a uniform distribution from a to b , and $\mathcal{U}_{\text{int}}(a, b)$ is the same only an integer distribution.

The fitting pipeline for this subproject is to first run both RS and BO as HPOs to compare their results. The best of the two models are chosen and used afterwards where the learning rate η is reduced from $\eta = 0.1$ to $\eta = 0.01$ and is finally optimised by early stopping. In the end one ends up with a model that has been HPO optimized for preprocessing optimizations (log-transformations), loss function, sample weights, normal XGBoost parameters and finally the learning rate. This pipeline has been manually implemented in Python since no other packages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure ?? and ?? (in the appendix). The corresponding plots for houses can be seen in Figure ?? and ?? in the appendix.

Hyperparameter	Range
<code>subsample</code>	$\mathcal{U}(0.5, 0.9)$
<code>colsample_bytree</code>	$\mathcal{U}(0.1, 0.99)$
<code>max_depth</code>	$\mathcal{U}_{\text{int}}(1, 20)$
<code>min_child_weight</code>	$\mathcal{U}_{\text{int}}(1, 30)$
<code>reg_lambda</code>	$\mathcal{U}(0.1, 4)$
<code>reg_alpha</code>	$\mathcal{U}(0.1, 4)$

Table 40.10: XXX

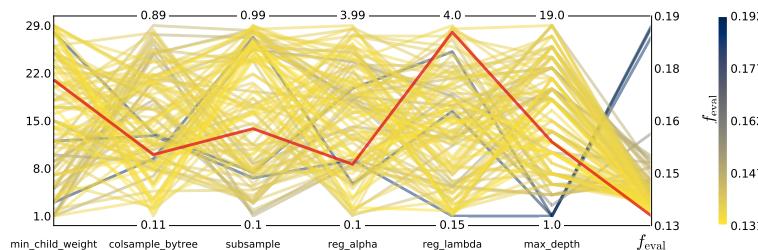


Figure 40.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure ?? with their uncertainties. This can be seen in Figure ???. Here the value of the evaluation function along with its 1σ and 2σ uncertainties are plotted as a function of the iteration along the abscissa. The minimum value of f_{eval} is shown in red. Even though this is the minimum value, notice how flat of a minimum this is: most of the other iterations are within 1σ . The plot for BO in Figure ?? shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

The best of the RS and GS models are chosen for subsequent analysis by first reducing the learning rate to the $\eta = 0.01$ and then

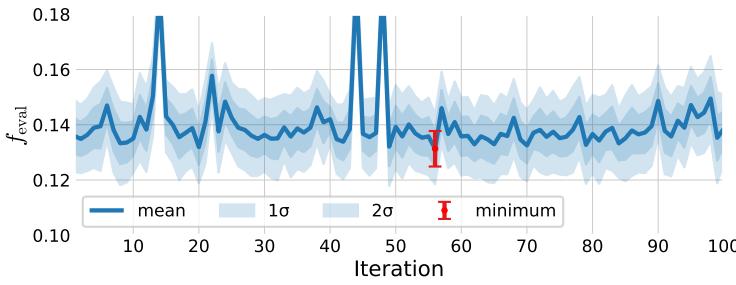


Figure 40.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s) of the means (SDOM)**, all as a function of iteration number.

find the best number of estimators by early stopping. The evaluation function as a function of number of estimators, also known as the *learning curve*, is seen in Figure ???. This curve is the realisation of Figure ?? in real data, where it first improves a lot and then finds a stable plateau. The minimum is shown in red with its uncertainty. To reduce the risk of overfitting and model complexity – with the further advantage of resulting in faster model at prediction time – we keep the model with the lowest number of estimators that are still within 1σ of the minimum: see the orange point in the figure. This results in a model that contains less than a fifth of the number of trees and is thus also significantly simpler and faster at inference time. This is the final hyperparameter optimized model that will be used for the further analysis.

```
figures/housing/Ejerlejlighed_v19_cut_all_Ncols_all_xgb_early_stopping.ipynb
```

40.6 Results

The performances of the different models, the RS-optimized, the BO-optimised and the best of the two η -optimized with ES, are shown in Figure ???. Here the distribution of the relative price prediction z of the model evaluated on the test set, apartments sold in 2018, is shown for the three models. In addition to the distributions, also the performance metrics are shown: the value of the evaluation function²² along with the fraction of z that are within the specified percentage. In this particular instance it is seen that 41.3 % of the predictions by the final ES model are less than $\pm 5\%$ wrong, 69.8 % within $\pm 10\%$, and 91.9 % within $\pm 20\%$. Note that the difference between the three models are minor and that they distributions almost cannot be distinguished between each other.

An interesting observation from the performance metrics of the

Figure 40.18: The results of early stopping on apartments using the XGB-model. The **minimum (mean) loss** along with its uncertainty is shown in red, the **means** for the different iterations of RS in blue, and as light blue bands are the **one (and two) standard deviation(s)**, all as a function of number of estimators (trees). In orange the **"best" number of estimators** is shown, defined as the lowest number of estimators which are still within 1σ of the minimum value. This leads to a model that has a performance that is within 1σ of the best model, but a lot simpler and faster.

²² Written as **MAD**

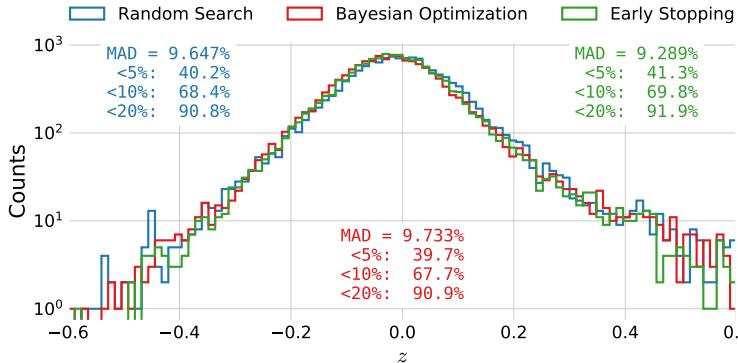


Figure 40.19: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

test set, is the low value of $f_{\text{eval}} = 0.9289$ (MAD). By looking at Figure ?? one would expect a test loss of around 0.12 assuming iid. samples. However, this assumption does not seem to be fulfilled for the test set. The performances of the realtors are also better for the test set than the training set, and by comparing the test set with the extra 2019 set it seems to be 2019 that was an extra easy year without too many outliers, see Table ???. The “Tight” in the the table corresponds to the realtors performance on the tight version of the different datasets. The performance of the XGB models on the tight test set can be seen in Figure ?? in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

To gauge the predictive power of the model over time, we applied the model to the next months data, evaluated the results for that month and continued like that for all the months in the test set (2018) and 2019. We apply two different methods of forecasting: “static” forecasting where the model is only trained once, and “dynamic” forecasting where the model is retrained after each month on all of the previous sales. These predictions allows the relative predictions z to be calculated and the MAD and the standard deviation (SD) of z are shown in Figure ??.

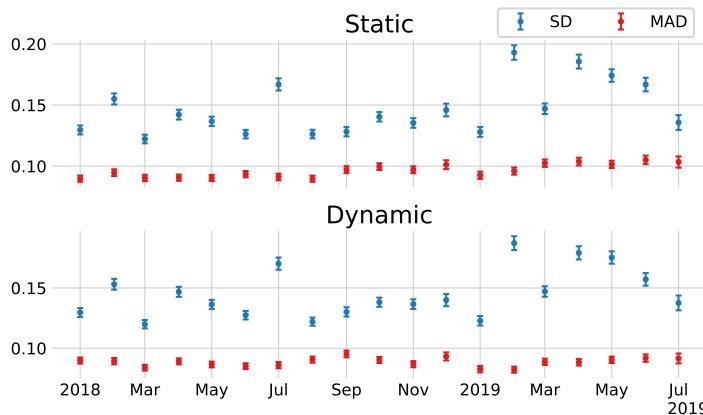


Figure 40.20: Performance of 1-month forecasts for apartments sold in 2018 and 2019. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the **standard deviation (SD)** and **MAD** of the z -scores. Bottom) Same as above, however, based a dynamic model, i.e. a model which is retrained after every month to include the previous month's sales.

In the top subplot the results for the static forecast are shown, whereas the dynamic results are shown in the bottom subplot. The errorbars are calculated using the usual variance of the standard

deviation:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \quad (40.8)$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation²³ [?]. Notice the large fluctuations in SD over time compared to MAD which an effect of MAD being a robust estimator. What is also interesting to note is that the MAD seems to increase over time for the static model, albeit slowly. In comparison, for the dynamic model this seem less pronounced. This figure not only shows the time dependence of the performance of the model, but also that the model is quite stable over time, at least for the dynamic model.

²³ That this estimator is biased does not matter since we are in the large N limit, $N \sim 1000$

Using the relative price predictions \mathbf{z} , we construct the Market Index, MI. This is an index which measures the overall level of the Danish housing market based on the assumption that if the houses sold in a month are generally sold at a higher price than was predicted by the model, there can be two reasons for it: either the model was wrong or the market simply changed in the time span. With the latter assumption, the ratio between the prediction and the actual price of a residence is thus a measure of the market index:

$$\begin{aligned} z_{mi} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ MI_{mean} &= \text{mean}(\mathbf{z}_{mi}) \\ MI_{median} &= \text{median}(\mathbf{z}_{mi}). \end{aligned} \quad (40.9)$$

Here \mathbf{z}_{mi} is the vector of ratios between prediction and actual price, and the market index can then be estimated using either the mean MI_{mean} or the median MI_{median} . The market indices for every month of the forecast described in the previous figure can be seen in Figure ???. In the top panel the market index for the static model is shown where it is visible for MI_{median} how it cosistently overshoots. Compare this to the dynamic MI_{median} whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than MI_{median} is an indication of a low tail in z_{mi} (*negative skewness*) which means that some of the the predictions are much lower than the actual salesprice. BLABLA, XXX.

The final results for the model is seen in Table ?? for owner-occupied apartments and in in Table ?? for one family houses. The MAD is around 9 % for apartments and 16 % for houses, which is still worse than the realtors' prediction, yet still acceptable for a model that does not have any variables describing the indoor conditions. For apartments around 40 % of all the predicitons are within $\pm 5\%$ and more than 90 % are within $\pm 20\%$ which is similar to the performance of the professional automated property evaluations from e.g. ?]. Also shown in the tables are the mean of the relative predictions μ_z which shows that the The performance on the tight cuts can be seen in Table ?? and ?? in the appendix.

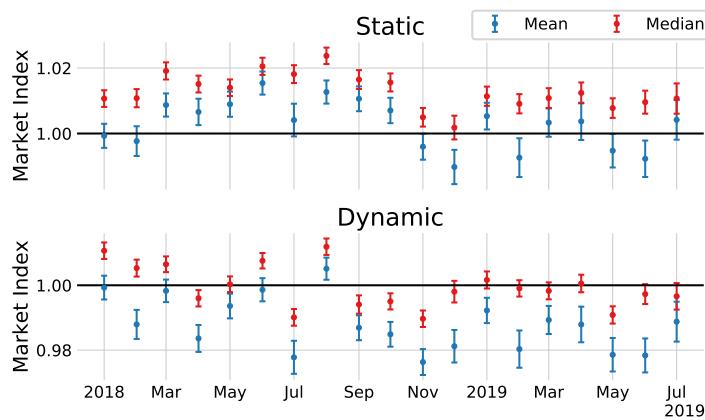


Figure 40.21: XXX TODO!

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	7.83	47.90	75.74	93.97	$+0.0128 \pm 0.0008$
Test	9.29	41.33	69.77	91.91	-0.0067 ± 0.0012
2019	9.89	38.85	66.76	90.04	-0.000 ± 0.002

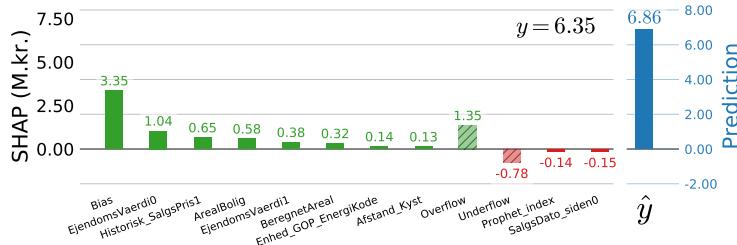
Table 40.12: XXX ejer

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ_z
Train	14.12	28.95	51.89	78.04	0.0449 ± 0.0007
Test	15.79	25.61	47.52	75.14	0.0310 ± 0.0016
2019	16.50	24.01	45.89	74.22	0.031 ± 0.002

Table 40.13: XXX villa

40.7 Model Inspection

One of the most important aspects of applying advanced machine learning methods, in addition to getting accurate predictions, is understanding the model. As mentioned in ??, it is possible to use SHAP values to inspect the trained model for both local predictions and for global feature importances ϕ_i^{tot} . An example of how to use SHAP to better understand a local prediction is seen in Figure ???. Here the SHAP values for a particular sale are visualized as a bar chart where the green colors are positive values, red values negative values and the blue is the final prediction \hat{y} . Remember that for SHAP values the prediction is the sum of all of the SHAP values, see equation (??). Here ϕ_0 is the expectation value denoted as Bias in the plot. To show all 143 variables would make the figure excessively large, so two extra bins have been added: the Overflow bar which is the sum of the remaining positive SHAP values and likewise with Underflow for negative values. The sum of all the green and red bars adds up to $\hat{y} = 6.86 \text{ M.kr.}$ in this particular instance and the actual sold value was $y = 6.35 \text{ M.kr.}$ Thus, in cases where there is a large discrepancy between the predicted and actual sales prices, one can use this tool to better understand why the prediction was estimated as it was.



To get an overview of which variables are most important on a global²⁴ scale, ϕ_i^{tot} , see Figure ???. Here the variables are sorted according to the normalised (i.e. summing to one) ϕ_i^{tot} which is shown in parentheses after each variable name. In the center of the plot is shown a dot-plot of the dataset plotted with the SHAP value on the abscissa and colored according to the feature value. The way to interpret this plot is as follows. Take a variable of interest, e.g. the area of the apartment ArealBolig with $\phi_{\text{ArealBolig}}^{\text{tot}} = 5.35\%$. Each dot is a sample in the dataset plotted as a function of their SHAP value with a spread such that the height corresponds to the SHAP-distribution of that specific feature. For the area it can be seen that there is a long tail towards high SHAP-values, however, most of the samples have a slightly negative SHAP value. The dots are colored according to their feature value and it can thus be seen that large apartments (red) are given a higher SHAP value than small apartments; precisely as expected from the model. In contrary, when the total days on market (DOM) (LiggetidSamlet) is large it pushes the prediction in the negative direction.

Figure 40.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training set which acts as the naive prediction baseline. The “cutoff positive (negative)” bars are the sum of the remaining positive (negative) values that are not shown. ~~Of the total number of the entire dataset~~ model prediction shown. The model prediction is the sum of all of the bars in the left part (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

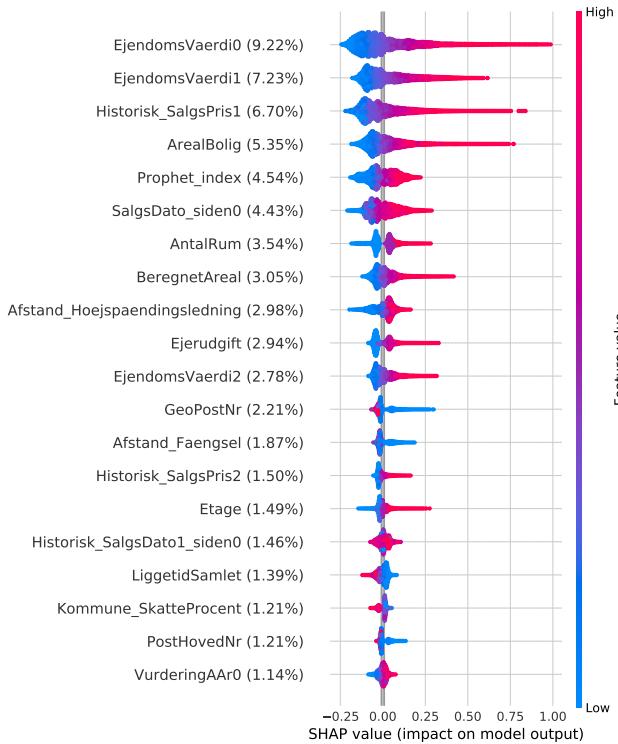
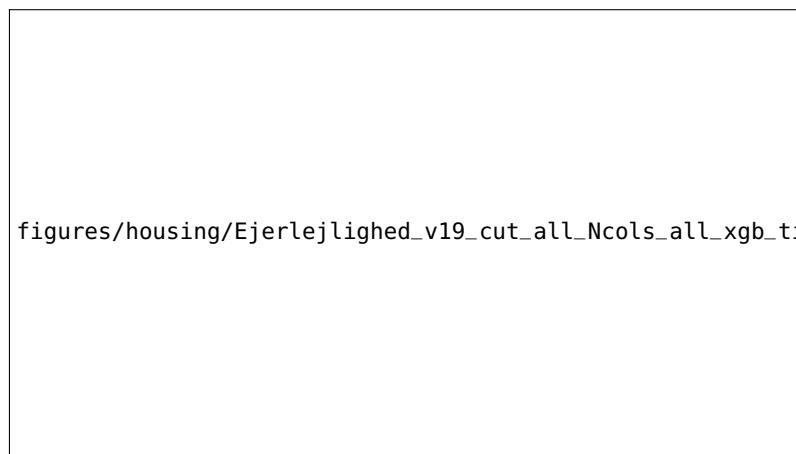


Figure 40.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomsVaerdi0` is the most important single feature. Along the x-axis is the impact on model output, in this example the price in XXX. This axes is colored by the value of the feature, from **low** (blue) to **high** (red). In this particular example we see that high values of the previous public property valuation has high, positive impact on the model prediction – exactly as expected. This is exactly opposite the total days on market (`LiggetidSamlet`) where a high value has a negative impact.

The SHAP-package[?] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependence between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure ???. Here the SHAP value of the `EjendomsVaerdi0` is plotted as a function of `EjendomsVaerdi0`. Ignoring the colors for a second, this plot shows that the higher the PPPV, the higher the model output. The colors on the other hand shows how this trend depends on time by the variable `SalgsDato_siden0` which is the number of days since January 1st 2009 that the apartment was. The plot shows that if the apartment has a high PPPV then the newer sales has an even higher SHAP value than older sales, agreeing with the fact that the market has gone up since 2009. On the other hand, for low PPPV apartments the relationship with time is inverse, however, the effect is much smaller here. The SHAP-package chose to color by `SalgsDato_siden0` since this is the variable which explains most of the variation for a given PPPV.



`figures/housing/Ejerlejliged_v19_cut_all_Ncols_all_xgb_tight_SHAP_vals_interaction.pdf`

Figure 40.24: Feature importance of apartment prices using the XGB-model. XXX

40.8 Multiple Models

In addition to the XGBoost [?] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [?] (LGB), also a BDT model, was released and started gaining traction in the ML community, especially for large-scale data analysis (Big Data). In comparison to XGBoost, LightGBM implements some extra binning and categorical assumptions that greatly speeds up the fitting process. It was trained in the same was as the XGBoost model with the same HPO-process and range for the hyperparameters.

To compliment the BDTs models, a simple linear model (LIN) with L_2 loss, so-called *ridge* regression, was fitted where all of the NaNs were median-imputed²⁵ and the input features were scaled²⁶ with a robust scaler from Scipy [?] in the (25, 75) quantile range and the regularization parameter was hyperparameter optimized. This linear model is quick and easy to both implement and fit, and can be seen as the simplest baseline model.

The K -nearest neighbors (KNN) algorithm was fitted to the data with a data preprocessing pipeline similar to the linear model with median imputation and robust scaling of the input features. For the HPO the number of neighbors, K , was optimized for together with the p -norm of the metric, $p \in \{1, 2\}$ (Manhattan, Euclidian, see also ??).

Finally, support vector machines²⁷ (SVR) were used with the same preprocessing pipeline as the previous two methods and hyperparameter optmized for the L_2 regularization parameter C and the kernel coefficient γ for the radial basis function (RBF) kernel.

The results for the five different models are shown in Figure ??, where the two subplots are the same up to a logarithmic scaling of the ordinate axis in the right plot. It is easily seen how XGBoost and LightGBM are the best-performing models together with the ensemble (ENS) of the different models, see paragraph below for further explanation. In the figure is also shown the MAD on the

²⁵ Which means that all invalid values were replaced with the median along each column-

²⁶ Scaling of input features is generally an important preprocessing step for non-tree based ML models.

²⁷ For regression, also known as support vector regression[?].

test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

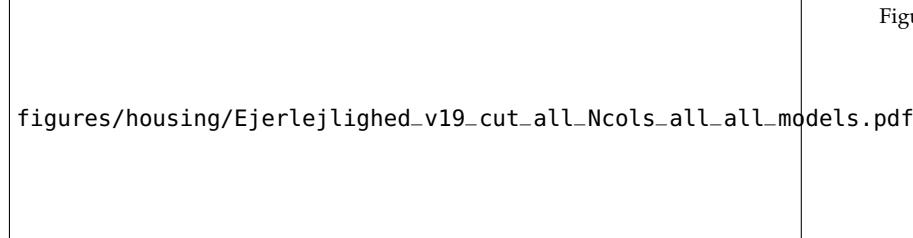


Figure 40.25: Multiple models. XXX

The five different models – XGB, LGB, LIN, KNN, SVR – should be able to each capture different parts of the hyperdimensional phase space and an ensemble of these models should thus be expected to be as good or better as the best of the individual models. This kind of ensemble model is called sometimes called *super learner* in the statistics community [? ?]. To make sure that the ensemble model is not just retraining on the training set, and thus end up overfitting, we follow the process from [?]. Using cross-validation for time-series data with $k = 10$, the training data is split up into folds sorted by time. Each fold is fitted with all five models, and the prediction of the next fold is made for all five models. This is repeated for the remaining folds until one ends up with a matrix of predictions $Z \in \mathbb{R}^{(N \times 5)}$ for N training samples. Since all the folds in Z consists of predictions on unseen data²⁸ this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the usual way. The combination of a meta learner fitted to the predictions of individual models is called an ensemble model.

At first an XGBoost model was used as the meta learner yielding decent results, however, still performing worse than the single XGB model ($MAD = 9.57\%$). To better understand the issue, the meta learner was changed to a linear model which would basically just compute a weighted average of the different models:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^5 \alpha_i \Psi_i(\mathbf{x}), \quad (40.10)$$

where α is a vector of the weights for the meta learner and Ψ is an ML-model. The linear model performed even worse ($MAD = 10.48\%$) than the XGB model, yet it was more transparent. During the debugging process it was realised that none of these models actually optimize the evaluation function, MAD, directly. The XGBoost model was using the Cauchy loss found earlier and the linear regression model a simple squared error loss. Since a simple weighted average should work as the meta learner [?], a custom algorithm for finding α according to MAD was implemented.

Given the training data, the evaluation function as a function of α was minimized using of the MINUIT algorithm[?] via the iminuit[?] Python interface. It yielded decent result, yet they were

²⁸ The predictions for the very first fold is based on training data.

all very dependent on the initial parameter of the fit indicating many local minima. A scan over the 5-dimensional hyperspace in steps of 0.01 was thus conducted and the result of this scan was used as the new initial parameter in the minimization routine. This yielded the following result:

$$\alpha = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202 \% \\ 0.002 \% \\ 0.001 \% \\ 81.302 \% \\ 20.002 \% \end{bmatrix}. \quad (40.11)$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found α in equation (??), one gets the ensemble model (ENS) shown in Figure ?? with a MAD = 9.20 %. This value is the evaluation loss on the test set based on only the training data and thus outperforms all of the individual models.

The paragraphs above refer to apartments, however, the intermediate results for houses showed the same pattern. The combined model along with their performance can be seen in Fig XXX in the appendix.

40.9 Discussion

The subproject of estimating housing prices has focussed a lot on experimenting with different machine learning models and how to optimize them. As it can be seen in the previous sections, the choice of ML model is by far the most important. Actually, the gain from HPO is quite small, especially considering the amount of time spent on it²⁹. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main reasons for this, the first being that realtors are educated within this field and thus has developed the skills required for estimating the price of a house over many years of hard work. The second reason is the fact that the realtor has access to a lot more data than ML models have. We are not in possession of any *indoor* variables as we call it. The area of the house, the number of rooms, the name of the street, and the distance to a highway are all variables that are in the data set but none of them describe the overall quality of the house, the maintenance level, the age of the kitchen or bathroom. These features are invisible to the ML model.

During the project it was investigated how to get access to these variables. At first the online images from each sale was suggested, however, it turned out that Boligsiden only have the right to use them while a residence is for sale; when it is sold all rights return to the photographer. The images are not the only thing that provide more information about the condition of the residence, also the descriptions does that. They turned out to be available for most of the sales and was investigated for a short period. At that time of the project, the MAD for (a tight subset of the) apartments was

²⁹ Not only user-time programming it, but also computational resources used.

around $\pm 10\%$ and $\pm 20\%$ for houses. By using methods from the big natural language processing (NLP) community with in the field of machine learning, it was possible to reduce the MAD to around $\pm 8\%$ and $\pm 15\%$ for apartments and houses respectively. From the improvement in performance it is visible how apartments in general are much more uniform compared to houses where the “inside” is more decisive regarding the price.

The methods for translating the text to numerical variables decipherable by classical ML models where for instance simple *bag of words* (BOW) models and term *Term Frequency, Inverse Document Frequency* (TF-IDF) but also slightly more advanced statistical tools such as *Latent Dirichlet Allocation* (LDA). An old example of the learnt text model is seen in Figure ?? where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPP), `GeoPostNr` (postal code), `ByggeAar` (year of construction), `Afstand_Kyst` (distance to shore), and `BeregnetAreal` (weighted area) and the text descriptions (encoded with TF-IDF). The summary of the trained model is as a SHAP plot in Figure ???. As expected, the most important features are the numerical ones, however, the word `flot` (“pretty”) was in top five. The model also learnt that `flot` has a positive impact on the price compared to `trænger` (“requires”) which has a negative impact.

The descriptions turned out to be more time-consuming to extract for Boligsiden and along with the fact that overall deadline was quickly approaching, the remaining time was focussed on the main part of the project, the quark gluon discrimination. Given more time, the text analysis would definitely be first step for further improving the accuracy and precision of the price predictions.

Another step would be to apply more modern deep learning³⁰ methods. These methods were briefly experimented with in the intial stages of this subproject but showed inferior performance compared to BDTs. It is a generally accepted truth (with modifications) in the ML community that neural networks underperform, or at least not outperform, classical ML methods on structured data³¹. Most often they have the inherent complexity to perform as well as ML methods, however, this requires extensive architecture optimization, or, in short; the hypothesis space for neural networks is much larger than for classical ML methods and thus requires more care to avoid overfitting.

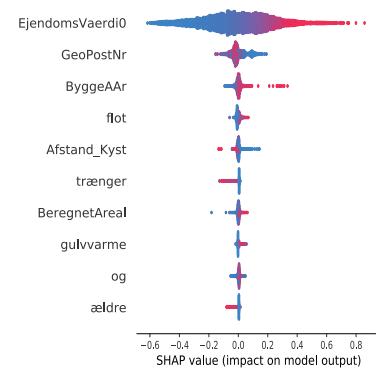


Figure 40.26: SHAP plot villa TFIDF XXX.

³⁰ Basically advanced neural networks with many layers.

³¹ In general data that can be described by a spread sheet, i.e. has a well-defined number of variables and observations

41. Particle Physics and LEP

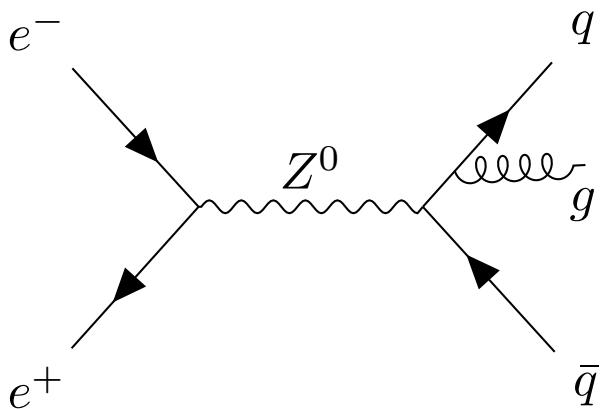


Figure 41.1: Feynman diagram showing the $e^+e^- \rightarrow Z^0$ production at LEP. The Z^0 has several decay modes where the $Z \rightarrow q\bar{q}g$ is shown here.

The Tufte-L^AT_EX document classes define a style similar to the style Edward Tufte uses in his books and handouts. Tufte's style is known for its extensive use of sidenotes, tight integration of graphics with text, and well-set typography. This document aims to be at once a demonstration of the features of the Tufte-L^AT_EX document classes and a style guide to their use.

41.1 Page Layout

41.1.1 Headings

This style provides A- and B-heads (that is, `\section` and `\subsection`), demonstrated above.

If you need more than two levels of section headings, you'll have to define them yourself at the moment; there are no pre-defined styles for anything below a `\subsection`. As Bringhurst points out in *The Elements of Typographic Style*, you should "use as many levels of headings as you need: no more, and no fewer."

The Tufte-L^AT_EX classes will emit an error if you try to use `\subsubsection` and smaller headings.

IN HIS LATER BOOKS, Tufte starts each section with a bit of vertical space, a non-indented paragraph, and sets the first few words of the

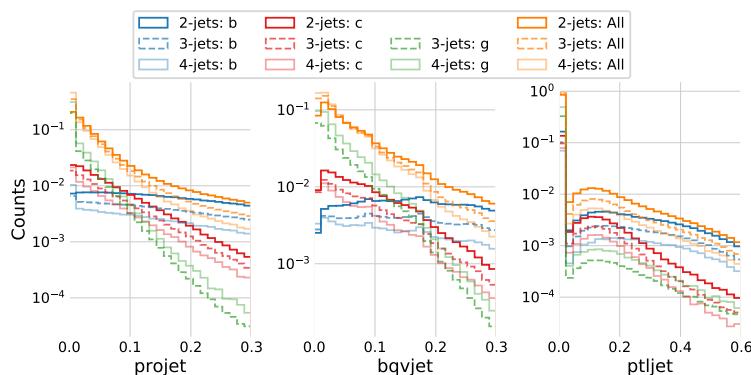
sentence in **SMALL CAPS**. To accomplish this using this style, use the `\newthought` command:

```
\newthought{In his later books}, Tufte starts...
```

42. Quark Gluon Analysis

42.1 Sidenotes

One of the most prominent and distinctive features of this style is the extensive use of sidenotes. There is a wide margin to provide ample room for sidenotes and small figures. Any `\footnotes` will automatically be converted to sidenotes.¹ If you'd like to place ancillary information in the margin without the sidenote mark (the superscript number), you can use the `\marginnote` command.



The specification of the `\sidenote` command is:

```
\sidenote[<number>][<offset>]{Sidenote text.}
```

Both the `<number>` and `<offset>` arguments are optional. If you provide a `<number>` argument, then that number will be used as the sidenote number. It will change of the number of the current sidenote only and will not affect the numbering sequence of subsequent sidenotes.

Sometimes a sidenote may run over the top of other text or graphics in the margin space. If this happens, you can adjust the vertical position of the sidenote by providing a dimension in the `<offset>` argument. Some examples of valid dimensions are:

```
1.0in    2.54cm    254mm    6\baselineskip
```

If the dimension is positive it will push the sidenote down the page; if the dimension is negative, it will move the sidenote up the page.

While both the `<number>` and `<offset>` arguments are optional, they must be provided in order. To adjust the vertical position of the sidenote while leaving the sidenote number alone, use the following syntax:

¹ This is a sidenote that was entered using the `\footnote` command.

This is a margin note. Notice that there isn't a number preceding the note, and there is no number in the main text. Figure 42.1: Histograms of the three vertex variables, `projet`, `bqvjet`,

and `ptljet`, used as input variables in the b-tagging models. In blue colors the variables are shown for **true b-jets**, in red for **true c-jets**, in green for **true g-jets**, and in orange for **all of the jets** (including non q-matched). In fully opaque color are shown the distributions for 2-jet events, in dashed (and lighter color) 3-jet events, and in semi-transparent 4-jet events. Notice the logarithmic y-axis, that there are no g-jets for 2-jet events (as expected), and that all of the distributions are very similar not matter how many jets.

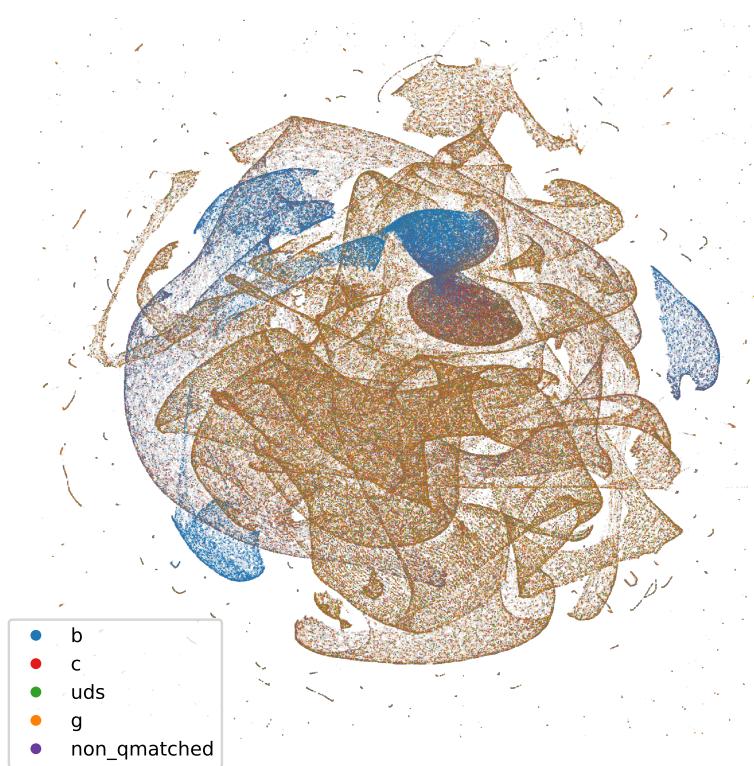


Figure 42.2: Vizualisation of the vertex variables for the different categories: **true b-jets** in blue, **true c-jets** in red, **true uds-jets** in green, **true g-jets** in orange, and **non q-matched**. The clustering is performed with the UMAP algorithm which outputs a 2D-projection. This projection is then visualized using the Datashader which takes care of point size, avoids over- and under-plotting, and color intensity.

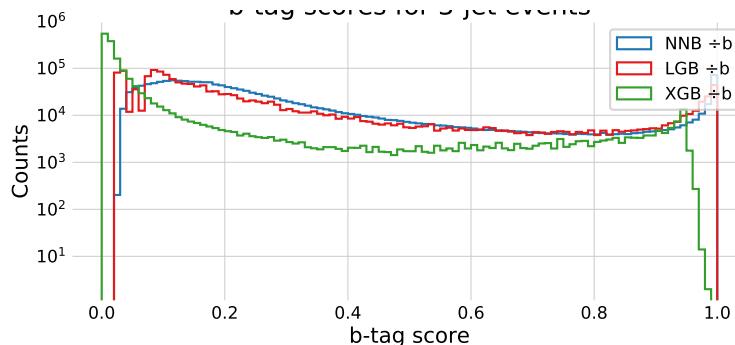


Figure 42.3: Histogram of b-tag scores (model prediction) in 3-jet events for **NNB** (the neural network trained by ATLAS, also called `nbnbjet`) in blue, **XGB** in red, and **XGB** in green. We see that the XGB predictions closely match those of NNB which is a good confirmation of a successful fit.

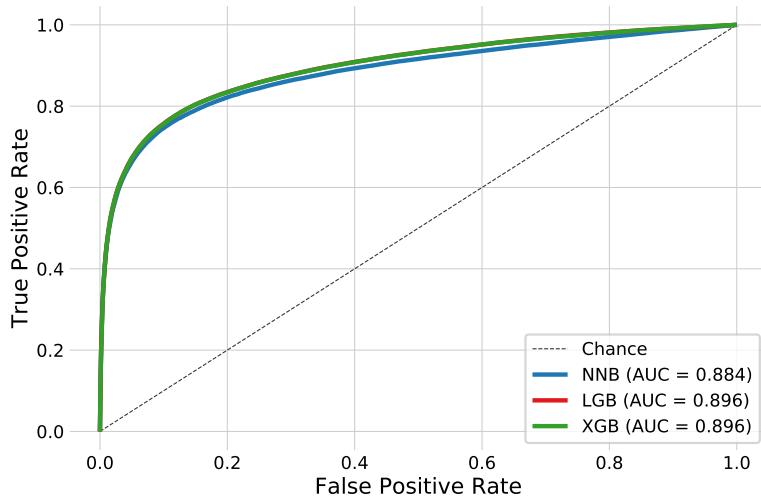


Figure 42.4: ROC curve of the three b-tag models in 3-jet events for **NNB** (the neural network trained by ATLAS, also called `nbnbjet`) in blue, **XGB** in red, and **XGB** in green. In the legend the Area Under Curve (AUC) is also shown. Notice that the XGB and XGB models share performance and it is thus due to overplotting that only the green line for XGB can be seen. In the particle physics community False Positive Rate (FPR) is sometimes better known as background efficiency and True Positive Rate (TPR) as signal efficiency.

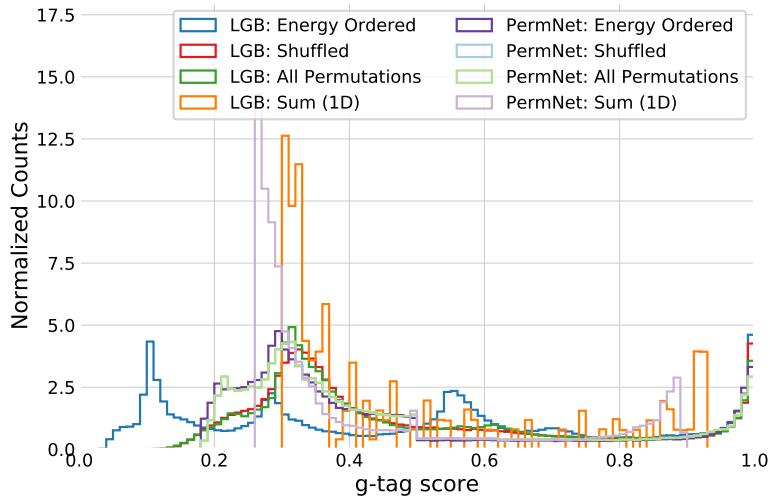


Figure 42.5: Histogram of g-tag scores (model prediction) in 4-jet events for XGB: Energy Ordered in blue, XGB: Shuffled in red, XGB: All Permutations in green, XGB: Sum 1D in orange, PermNet: Energy Ordered in purple, PermNet: Shuffled in light-blue, PermNet: All Permutations in light-green, PermNet: Sum 1D in light-purple. Here XGB and PermNet are the two different type of models and “Energy Ordered”, “Shuffled”, “All Permutations”, and “Sum 1D” are the different methods used for making the input data permutation invariant.

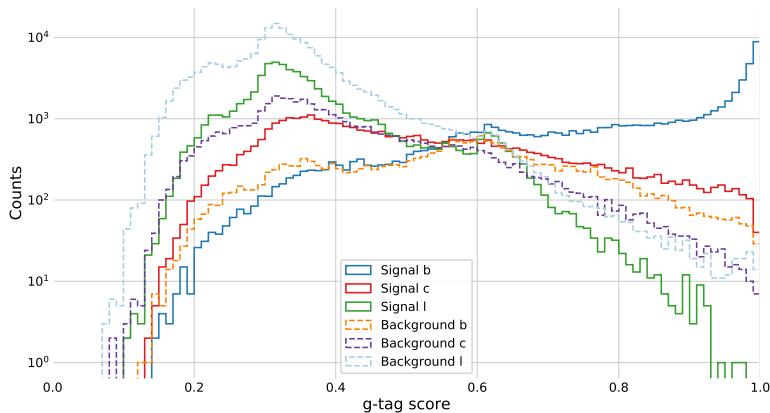


Figure 42.6: Histogram of g-tag scores (model prediction) from the XGB-model in 4-jet events for b signal in blue, c signal in red, I signal in green, b background in orange, c background in purple, I background in light-blue.

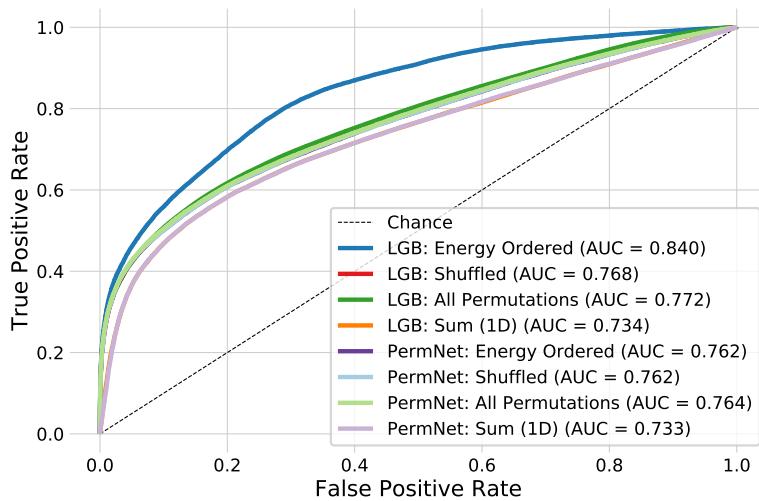


Figure 42.7: ROC curve of the eight g-tag models in 4-jet events. First one in dashed black is the ROC curve that you get by random chance. The colors are the same as in Figure ?? and in the legend also the Area Under the ROC curve (AUC) is shown. Notice that the XGB model which uses the energy ordered data produced the best model, however, this model is not permutation invariant. Of the permutation invariant models (the rest), the XGB model trained on all permutations of the b-tags performs highest. The lowest performing models are the two models trained only on the 1-dimensional sum of b-tags, as expected, however, still with a better performance than expected by the author.

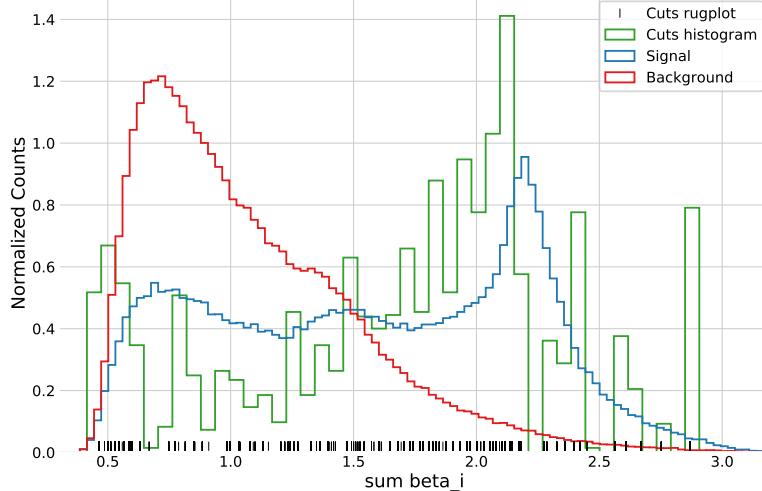


Figure 42.8: Histogram of the distribution of **signal** in blue and **background** in red for 1-dimensional sum of b-tags training data. A histogram of the **cut values** from the XGB model trained on this data is shown in green together with a rug plot of the cut values in black. Notice how most of the cuts match up with the signal peak at around a $\sum \beta_i \sim 2.1$, however, there are also quite a lot of cuts around $\sum \beta_i \sim 0.5$.

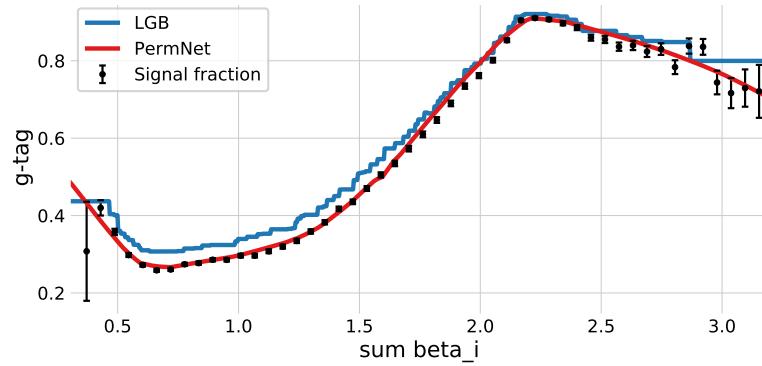


Figure 42.9: Plot of the (1D) g-tag scores as a function of $\sum \beta_i$ for the **XGB** model in blue and the **PermNet** model in red. Here the g-tag scores are just the models' output values when input a uniformly spaced grid of $\sum \beta_i$ values between 0 and 4. The signal fraction (based on the signal and background histograms in Figure ??) is plotted as black error bars where the size of the error bars is based on the propagated uncertainties of the signal and background histogram assuming Poissonian statistics. Notice how both models capture the overall trend of the signal fraction with the PermNet being **Figure 42.10: Hyperparameter Optimization (HPO) results after running 100 iterations of Random Search (only 10 for XGB).**

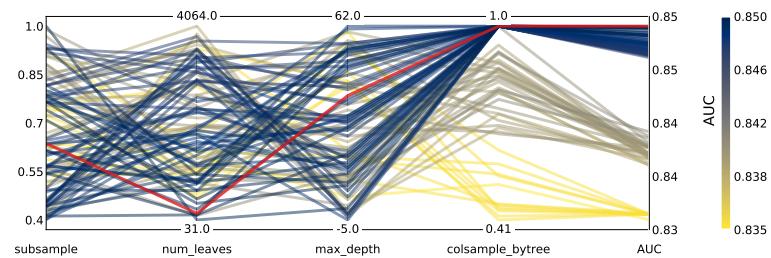
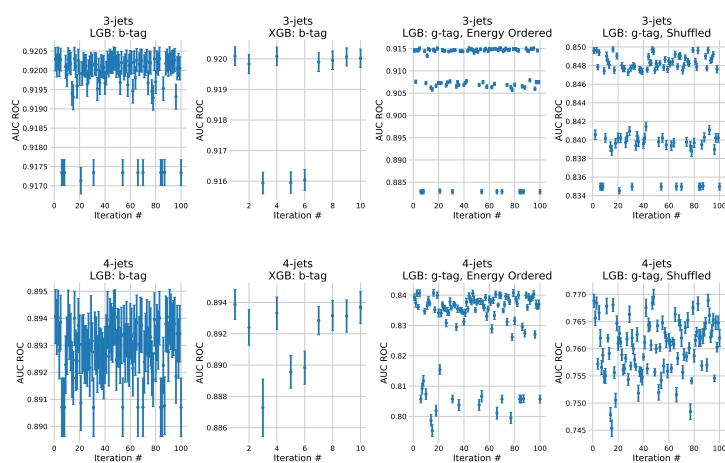


Figure 42.11: Hyperparameter optimization results of g-tagging for 3-jet shuffled events. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by AUC from highest AUC in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red.

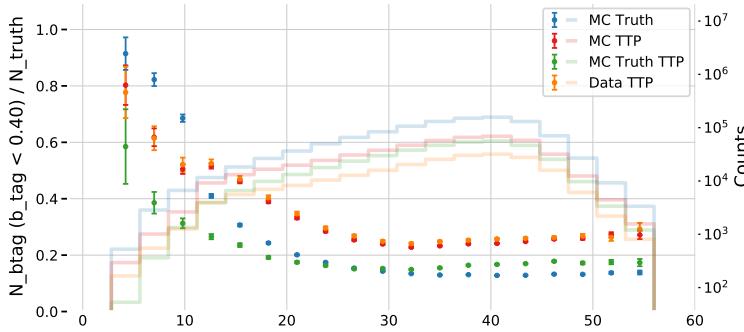
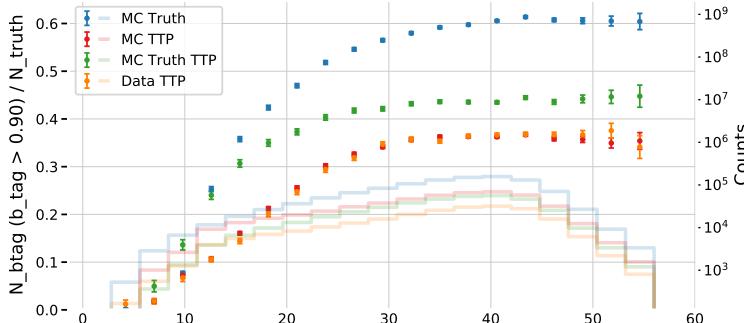
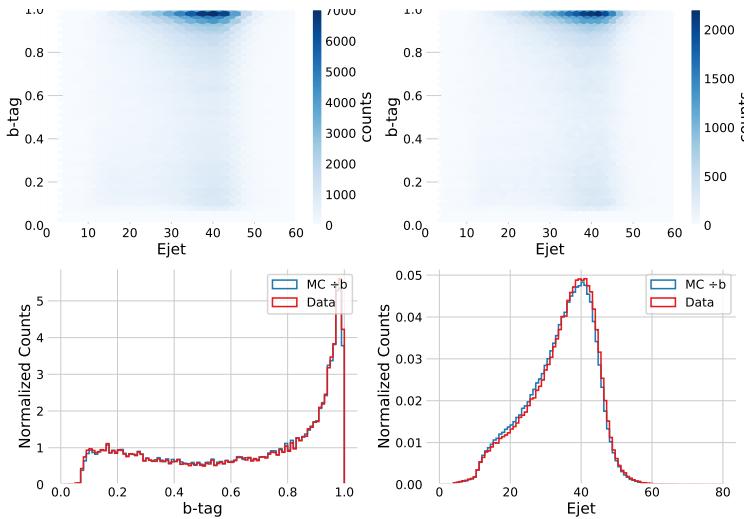
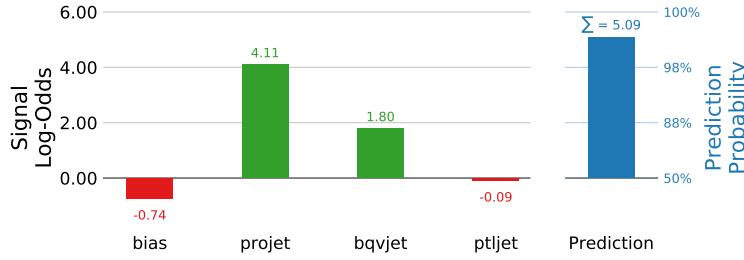


Figure 42.12: Model explanation for the 3-jet b-tagging model for a b-like jet. The first column is the bias of the training set which acts as the naive prediction baseline, the rest are the input data variables. On the right hand side of the plot is the model prediction shown. The left part of the plot is shown in log-odds space, the right part in probability space. The model prediction is the sum of the log-odds (5.09 in this example) transformed into probability space. The negative log-odd values are shown in red, positive ones in green, a prediction value (E_{jet}) distributions in blue. Figure 42.13: Comparison of the b-tag and energy (E_{jet}) distributions for Monte Carlo (MC) versus data. In the top row the 2D-distributions are shown for MC on the left (without the extra MC_b samples) and data on the right. In the bottom row the 1D marginal distributions are shown for the b-tag and the jet energy with data in red and Monte Carlo ones in blue. Notice the almost identical distributions in b-tag.

Figure 42.14: Efficiency of the b-tags for b-jets in the b-signal region for 3-jet events, ϵ_b^{b-sig} , as a function of jet energy E_{jet} . The b-signal region is defined as $\beta > 0.9$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in green, and Data TTP in orange. The efficiencies (the errorbars) can be read off on the left y-axis and the counts (histograms) on the right y-axis. The abbreviation TTP is short for “Tag, Tag, Probe” where two jets in an event are used as tags and the probe is then used for further analysis. Notice how both MC TTP and Data TTP follow each other closely. Figure 42.14: Efficiency of the b-tags for b-jets in the g-signal region for 3-jet events, ϵ_b^{g-sig} , as a function of jet energy E_{jet} . The g-signal region is defined as $\beta < 0.4$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in green, and Data TTP in orange. The efficiencies (the errorbars) can be read off on the left y-axis and the counts (histograms) on the right y-axis. The abbreviation TTP is short for “Tag, Tag, Probe” where two jets in an event are used as tags and the probe is then used for further analysis. Notice how both MC TTP and Data TTP follow each other closely.

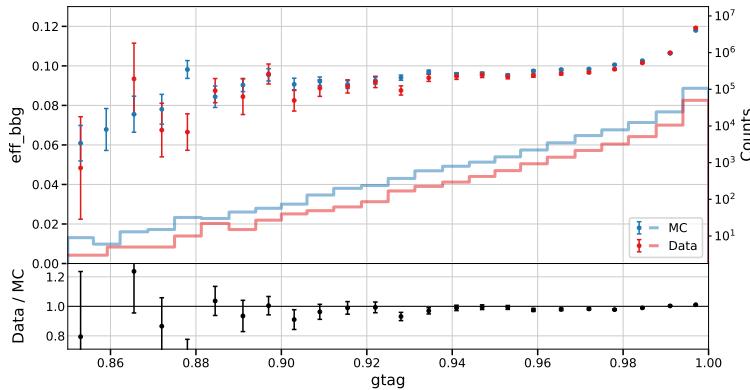
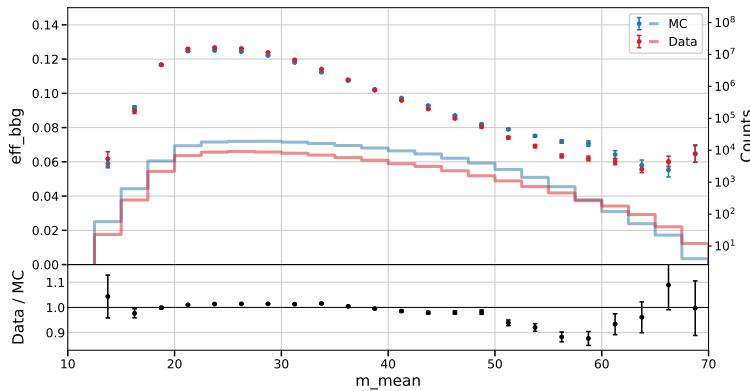
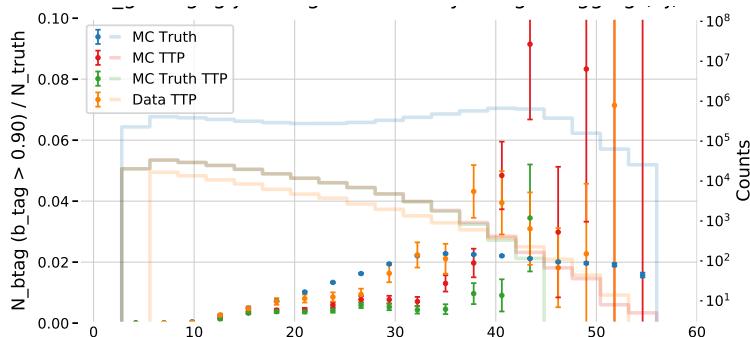
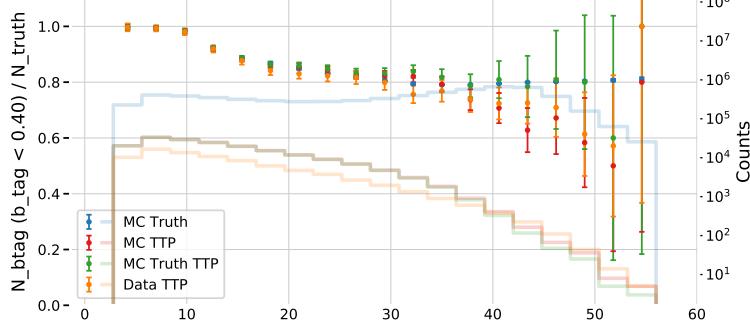


Figure 42.16: Efficiency of the b-tags for g-jets in the g-signal region for 3-jet events, $\varepsilon_g^{g\text{-sig}}$, as a function of jet energy E_{jet} . The g-signal region is defined as $\beta < 0.4$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in green, and Data TTP in orange. The efficiencies (the errorbars) can be read off on the left y-axis and the counts (histograms) on the right y-axis. The abbreviation TTP is short for “Tag, Tag, Probe” where two jets in a event are used as tags and the probe is then used for further analysis. Notice how both MC TTP and Data TTP follow each other closely.

Figure 42.17: Efficiency of the b-tags for g-jets in the b-signal region for 3-jet events, $\varepsilon_g^{b\text{-sig}}$, as a function of jet energy E_{jet} . The b-signal region is defined as $\beta > 0.9$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in green, and Data TTP in orange. The efficiencies (the errorbars) can be read off on the left y-axis and the counts (histograms) on the right y-axis. The abbreviation TTP is short for “Tag, Tag, Probe” where two jets in a event are used as tags and the probe is then used for further analysis.

Figure 42.18: Proxy efficiency of the g-tags for $bb\bar{g}$ 3-jet events as a function of the mean of the two invariant masses m_{bg} and $m_{b\bar{g}}$. The proxy efficiency $\varepsilon_{bb\bar{g}}$ is measured by finding $bb\bar{g}$ -events where $\beta_b > 0.9$, $\beta_{\bar{b}} > 0.9$, and $\beta_g < 0.4$, and then calculating $\varepsilon_{bb\bar{g}} = \varepsilon_b^{b\text{-sig}} \cdot \varepsilon_{\bar{b}}^{b\text{-sig}} \cdot \varepsilon_g^{g\text{-sig}}$. In the top plot $\varepsilon_{bb\bar{g}}$ is shown for MC in blue and Data in red where the counts in each bin can be read on right y-axis. In the bottom plot the ratio between Data and MC is shown.

Figure 42.19: Proxy efficiency of the g-tags for $bb\bar{g}$ 3-jet events as a function of the event’s g-tag. The proxy efficiency $\varepsilon_{bb\bar{g}}$ is measured by finding $bb\bar{g}$ -events where $\beta_b > 0.9$, $\beta_{\bar{b}} > 0.9$, and $\beta_g < 0.4$. and then calculating $\varepsilon_{bb\bar{g}} = \varepsilon_b^{b\text{-sig}} \cdot \varepsilon_{\bar{b}}^{b\text{-sig}} \cdot \varepsilon_g^{g\text{-sig}}$. In the top plot $\varepsilon_{bb\bar{g}}$ is shown for MC in blue and Data in red where the counts in each bin can be read on right y-axis. In the bottom plot the ratio between Data and MC is shown.

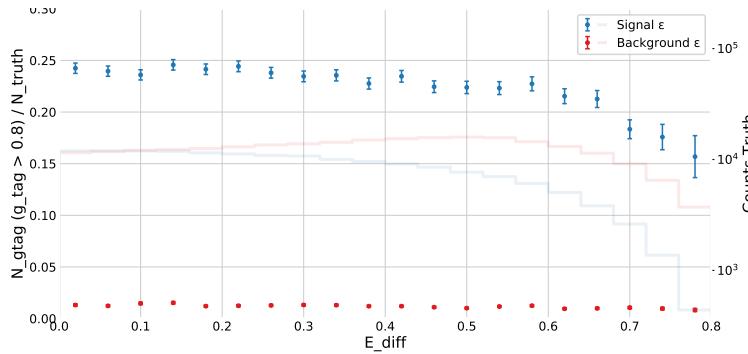


Figure 42.20: Efficiency of the g-tags for 4-jet events as a function of normalized gluon gluon jet energy difference in Monte Carlo. The efficiency is measured as the number of events with a g-tag higher than 0.8 ($\gamma > 0.8$) out of the total number and the normalized gluon gluon jet energy difference A is $A = \frac{E_{g_{\max}} - E_{g_{\min}}}{E_{g_{\max}} + E_{g_{\min}}}$ where $E_{g_{\max}}$ ($E_{g_{\min}}$) refers to the energy of the gluon with the highest (lowest) energy. The efficiency is plotted for **signal events** according to MC Truth in blue and **background events** according to MC Truth in red.

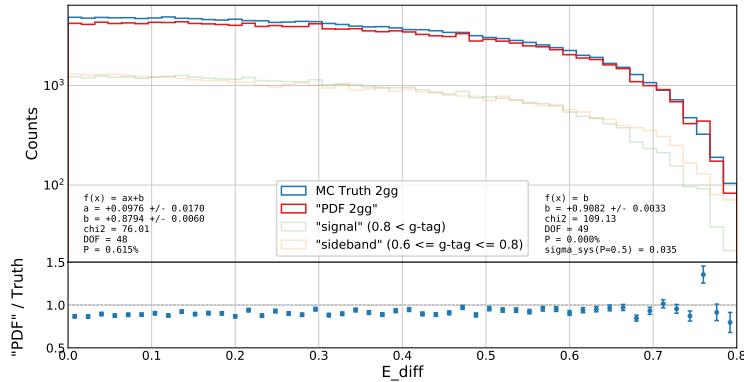


Figure 42.21: Closure plot between MC Truth and the corrected g-tagging model in 4-jet events for the normalized gluon gluon jet energy difference. The corrected g-tagging model is described in further detail in section XXX **TODO!**. In the top part of the plot the MC Truth is shown in blue, the corrected g-tagging model "PDF 2gg" in red, the g-signal distribution in semi-transparent green and the g-sideband distribution in semi-transparent orange. In the bottom part of the plot the ratio between MC Truth and the output of the corrected g-tagging model is shown. The normalized gluon gluon jet energy difference A is $A = \frac{E_{g_{\max}} - E_{g_{\min}}}{E_{g_{\max}} + E_{g_{\min}}}$ where $E_{g_{\max}}$ ($E_{g_{\min}}$) refers to the energy of the gluon with the highest (lowest) energy.

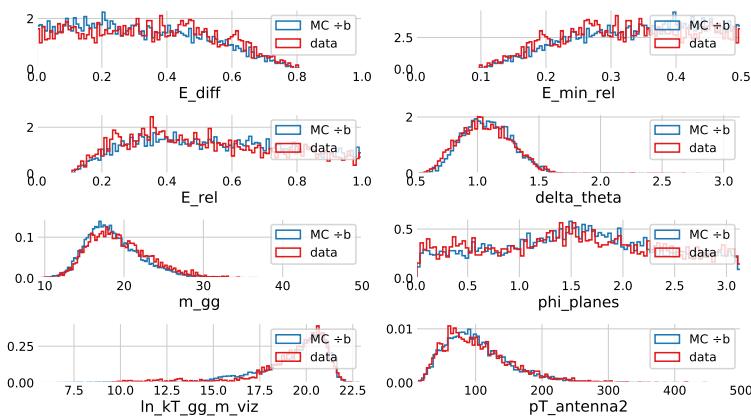
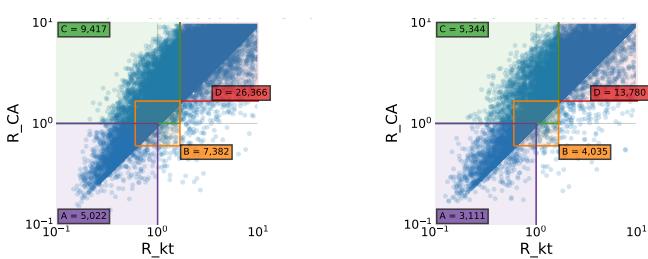


Figure 42.23: R_{kt} CA cut region A XXX **TODO!**

```
\sidenote[][\langle offset\rangle]{Sidenote text.}
```

The empty brackets tell the `\sidenote` command to use the default sidenote number.

If you *only* want to change the sidenote number, however, you may completely omit the `\langle offset\rangle` argument:

```
\sidenote[<number>]{Sidenote text.}
```

The `\marginnote` command has a similar `offset` argument:

```
\marginnote[\langle offset\rangle]{Margin note text.}
```

42.2 References

References are placed alongside their citations as sidenotes, as well. This can be accomplished using the normal `\citep` command.²

The complete list of references may also be printed automatically by using the `\bibliography` command. (See the end of this document for an example.) If you do not want to print a bibliography at the end of your document, use the `\nobibliography` command in its place.

² The first paragraph of this document includes a citation.

42.3 Figures and Tables

Images and graphics play an integral role in Tufte's work. In addition to the standard `figure` and `tabular` environments, this style provides special figure and table environments for full-width floats.

Full page-width figures and tables may be placed in `figure*` or `table*` environments. To place figures or tables in the margin, use the `marginfigure` or `margitable` environments as follows (see figure ??):

```
\begin{marginfigure}
\includegraphics{helix}
\caption{This is a margin figure.}
\label{fig:marginfig}
\end{marginfigure}
```

The `marginfigure` and `margitable` environments accept an optional parameter `\langle offset\rangle` that adjusts the vertical position of the figure or table. See the “??” section above for examples. The specifications are:

```
\begin{marginfigure}[\langle offset\rangle]
...
\end{marginfigure}

\begin{margitable}[\langle offset\rangle]
...
\end{margitable}
```

Figure ?? is an example of the `figure*` environment and figure ?? is an example of the normal `figure` environment.

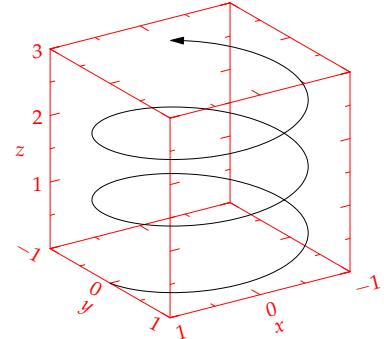


Figure 42.24: This is a margin figure. The helix is defined by $x = \cos(2\pi z)$, $y = \sin(2\pi z)$, and $z = [0, 2.7]$. The figure was drawn using Asymptote (<http://asymptote.sourceforge.net/>).

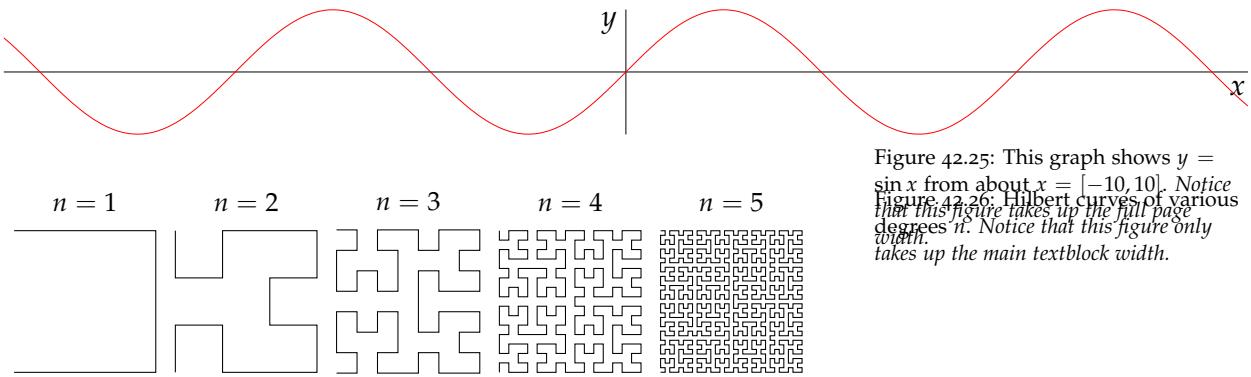


Figure 42.25: This graph shows $y = \sin x$ from about $x = [-10, 10]$. Notice that this figure takes up the full page width. Notice that this figure only takes up the main textblock width.

As with sidenotes and marginnotes, a caption may sometimes require vertical adjustment. The `\caption` command now takes a second optional argument that enables you to do this by providing a dimension $\langle offset \rangle$. You may specify the caption in any one of the following forms:

```
\caption{long caption}
\caption[short caption]{long caption}
\caption[][\langle offset \rangle]{long caption}
\caption[short caption][\langle offset \rangle]{long caption}
```

A positive $\langle offset \rangle$ will push the caption down the page. The short caption, if provided, is what appears in the list of figures/tables, otherwise the “long” caption appears there. Note that although the arguments $\langle short\ caption \rangle$ and $\langle offset \rangle$ are both optional, they must be provided in order. Thus, to specify an $\langle offset \rangle$ without specifying a $\langle short\ caption \rangle$, you must include the first set of empty brackets `[]`, which tell `\caption` to use the default “long” caption. As an example, the caption to figure ?? above was given in the form

```
\caption[Hilbert curves...]{Hilbert curves...}
```

Table ?? shows table created with the `booktabs` package. Notice the lack of vertical rules—they serve only to clutter the table’s data.

Margin	Length
Paper width	8½ inches
Paper height	11 inches
Textblock width	6½ inches
Textblock/sidenote gutter	¾ inches
Sidenote width	2 inches

Table 42.1: Here are the dimensions of the various margins used in the Tufte-handout class.

OCCASIONALLY \LaTeX will generate an error message:

```
Error: Too many unprocessed floats
```

\LaTeX tries to place floats in the best position on the page. Until it’s finished composing the page, however, it won’t know where those positions are. If you have a lot of floats on a page (including

sidenotes, margin notes, figures, tables, etc.), L^AT_EX may run out of “slots” to keep track of them and will generate the above error.

L^AT_EX initially allocates 18 slots for storing floats. To work around this limitation, the Tufte-L^AT_EX document classes provide a `\morefloats` command that will reserve more slots.

The first time `\morefloats` is called, it allocates an additional 34 slots. The second time `\morefloats` is called, it allocates another 26 slots.

The `\morefloats` command may only be used two times. Calling it a third time will generate an error message. (This is because we can't safely allocate many more floats or L^AT_EX will run out of memory.)

If, after using the `\morefloats` command twice, you continue to get the `Too many unprocessed floats` error, there are a couple things you can do.

The `\FloatBarrier` command will immediately process all the floats before typesetting more material. Since `\FloatBarrier` will start a new paragraph, you should place this command at the beginning or end of a paragraph.

The `\clearpage` command will also process the floats before continuing, but instead of starting a new paragraph, it will start a new page.

You can also try moving your floats around a bit: move a figure or table to the next page or reduce the number of sidenotes. (Each sidenote actually uses *two* slots.)

After the floats have placed, L^AT_EX will mark those slots as unused so they are available for the next page to be composed.

42.4 Captions

You may notice that the captions are sometimes misaligned. Due to the way L^AT_EX's float mechanism works, we can't know for sure where it decided to put a float. Therefore, the Tufte-L^AT_EX document classes provide commands to override the caption position.

Vertical alignment To override the vertical alignment, use the `\setfloatalignment` command inside the float environment. For example:

```
\begin{figure}[btp]
  \includegraphics{sinewave}
  \caption{This is an example of a sine wave.}
  \label{fig:sinewave}
  \setfloatalignment{b}% forces caption to be bottom-aligned
\end{figure}
```

The syntax of the `\setfloatalignment` command is:

```
\setfloatalignment{\langle pos\rangle}
```

where $\langle pos \rangle$ can be either `b` for bottom-aligned captions, or `t` for top-aligned captions.

Horizontal alignment To override the horizontal alignment, use either the `\forceversofloat` or the `\forcerectofloat` command inside of the float environment. For example:

```
\begin{figure}[btp]
\includegraphics{sinewave}
\caption{This is an example of a sine wave.}
\label{fig:sinewave}
\forceversofloat% forces caption to be set to the left of the float
\end{figure}
```

The `\forceversofloat` command causes the algorithm to assume the float has been placed on a verso page—that is, a page on the left side of a two-page spread. Conversely, the `\forcerectofloat` command causes the algorithm to assume the float has been placed on a recto page—that is, a page on the right side of a two-page spread.

42.5 Full-width text blocks

In addition to the new float types, there is a `fullwidth` environment that stretches across the main text block and the sidenotes area.

```
\begin{fullwidth}
Lorem ipsum dolor sit amet...
\end{fullwidth}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

42.6 Typography

42.6.1 Typefaces

If the Palatino, Helvetica, and Bera Mono typefaces are installed, this style will use them automatically. Otherwise, we'll fall back on the Computer Modern typefaces.

42.6.2 Letterspacing

This document class includes two new commands and some improvements on existing commands for letterspacing.

When setting strings of ALL CAPS or SMALL CAPS, the letter-spacing—that is, the spacing between the letters—should be strings of FULL CAPITAL LETTERS, and the `\smallcaps` command has letterspacing for SMALL CAPITAL LETTERS. These commands will also automatically convert the case of the text to upper- or lowercase, respectively.

The `\textsc` command has also been redefined to include letterspacing. The case of the `\textsc` argument is left as is, however. This allows one to use both uppercase and lowercase letters: THE INITIAL LETTERS OF THE WORDS IN THIS SENTENCE ARE CAPITALIZED.

42.7 Document Class Options

The `tufte-book` class is based on the L^AT_EX book document class. Therefore, you can pass any of the typical book options. There are a few options that are specific to the `tufte-book` document class, however.

The `a4paper` option will set the paper size to A4 instead of the default us letter size.

The `sfsidenotes` option will set the sidenotes and title block in a sans serif typeface instead of the default roman.

The `twoside` option will modify the running heads so that the page number is printed on the outside edge (as opposed to always printing the page number on the right-side edge in `oneside` mode).

The `symmetric` option typesets the sidenotes on the outside edge of the page. This is how books are traditionally printed, but is contrary to Tufte's book design which sets the sidenotes on the right side of the page. This option implicitly sets the `twoside` option.

The `justified` option sets all the text fully justified (flush left and right). The default is to set the text ragged right. The body text of Tufte's books are set ragged right. This prevents needless hyphenation and makes it easier to read the text in the slightly narrower column.

The `bidi` option loads the `bidi` package which is used with X_EL^AT_EX to typeset bi-directional text. Since the `bidi` package needs to be loaded before the `sidenotes` and `citep` commands are defined, it can't be loaded in the document preamble.

The `debug` option causes the Tufte-L^AT_EX classes to output debug information to the log file which is useful in troubleshooting bugs. It will also cause the graphics to be replaced by outlines.

The `nofonts` option prevents the Tufte-L^AT_EX classes from automatically loading the Palatino and Helvetica typefaces. You should use this option if you wish to load your own fonts. If you're using X_EL^AT_EX, this option is implied (*i.e.*, the Palatino and Helvetica fonts aren't loaded if you use X_EL^AT_EX).

The `nols` option inhibits the letterspacing code. The Tufte-L^AT_EX classes try to load the appropriate letterspacing package (either pdfL^AT_EX's `letterspace` package or the `soul` package). If you're using X_EL^AT_EX with `fontenc`, however, you should configure your own letterspacing.

The `notitlepage` option causes `\maketitle` to generate a title block instead of a title page. The `book` class defaults to a title page and the `handout` class defaults to the title block. There is an analogous `titlepage` option that forces `\maketitle` to generate a full title

page instead of the title block.

The `notoc` option suppresses Tufte-L^AT_EX's custom table of contents (toc) design. The current toc design only shows unnumbered chapter titles; it doesn't show sections or subsections. The `notoc` option will revert to L^AT_EX's toc design.

The `nohyper` option prevents the `hyperref` package from being loaded. The default is to load the `hyperref` package and use the `\title` and `\author` contents as metadata for the generated PDF.

43. Discussion and Outlook

The Tufte-L^AT_EX document classes are designed to closely emulate Tufte's book design by default. However, each document is different and you may encounter situations where the default settings are insufficient. This chapter explores many of the ways you can adjust the Tufte-L^AT_EX document classes to better fit your needs.

43.1 File Hooks

If you create many documents using the Tufte-L^AT_EX classes, it's easier to store your customizations in a separate file instead of copying them into the preamble of each document. The Tufte-L^AT_EX classes provide three file hooks: `tufte-common-local.tex`, `tufte-book-local.tex`, and `tufte-handout-local.tex`.

`tufte-common-local.tex` If this file exists, it will be loaded by all of the Tufte-L^AT_EX document classes just prior to any document-class-specific code. If your customizations or code should be included in both the book and handout classes, use this file hook.

`tufte-book-local.tex` If this file exists, it will be loaded after all of the common and book-specific code has been read. If your customizations apply only to the book class, use this file hook.

`tufte-common-handout.tex` If this file exists, it will be loaded after all of the common and handout-specific code has been read. If your customizations apply only to the handout class, use this file hook.

44. Conclusion

44.1 Tufte-L^AT_EX Website

The website for the Tufte-L^AT_EX packages is located at <http://code.google.com/p/tufte-latex/>. On our website, you'll find links to our SVN repository, mailing lists, bug tracker, and documentation.

44.2 Tufte-L^AT_EX Mailing Lists

There are two mailing lists for the Tufte-L^AT_EX project:

Discussion list The tufte-latex discussion list is for asking questions, getting assistance with problems, and help with troubleshooting. Release announcements are also posted to this list. You can subscribe to the tufte-latex discussion list at <http://groups.google.com/group/tufte-latex>.

Commits list The tufte-latex-commits list is a read-only mailing list. A message is sent to the list any time the Tufte-L^AT_EX code has been updated. If you'd like to keep up with the latest code developments, you may subscribe to this list. You can subscribe to the tufte-latex-commits mailing list at <http://groups.google.com/group/tufte-latex-commits>.

44.3 Getting Help

If you've encountered a problem with one of the Tufte-L^AT_EX document classes, have a question, or would like to report a bug, please send an email to our mailing list or visit our website.

To help us troubleshoot the problem more quickly, please try to compile your document using the debug class option and send the generated .log file to the mailing list with a brief description of the problem.

44.4 Errors, Warnings, and Informational Messages

The following is a list of all of the errors, warnings, and other messages generated by the Tufte-L^AT_EX classes and a brief description of their meanings.

Error: \subparagraph is undefined by this class.

The \subparagraph command is not defined in the Tufte-L^AT_EX document classes. If you'd like to use the \subparagraph command, you'll need to redefine it yourself. See the "Headings" section on page ?? for a description of the heading styles available in the Tufte-L^AT_EX document classes.

Error: \subsubsection is undefined by this class.

The \subsubsection command is not defined in the Tufte-L^AT_EX document classes. If you'd like to use the \subsubsection command, you'll need to redefine it yourself. See the "Headings" section on page ?? for a description of the heading styles available in the Tufte-L^AT_EX document classes.

Error: You may only call \morefloats twice. See the Tufte-LaTeX documentation for other workarounds.

L^AT_EX allocates 18 slots for storing floats. The first time \morefloats is called, it allocates an additional 34 slots. The second time \morefloats is called, it allocates another 26 slots.

The \morefloats command may only be called two times. Calling it a third time will generate this error message. See page ?? for more information.

Warning: Option '*<class option>*' is not supported -- ignoring option.

This warning appears when you've tried to use *<class option>* with a Tufte-L^AT_EX document class, but *<class option>* isn't supported by the Tufte-L^AT_EX document class. In this situation, *<class option>* is ignored.

Info: The 'symmetric' option implies 'twoside'

You specified the symmetric document class option. This option automatically forces the twoside option as well. See page ?? for more information on the symmetric class option.

44.5 Package Dependencies

The following is a list of packages that the Tufte-L^AT_EX document classes rely upon. Packages marked with an asterisk are optional.

- xifthen
- ifpdf*
- ifxetex*
- hyperref
- geometry
- ragged2e
- chngpage or changepage
- paralist
- textcase
- soul*

- letterspace*
- setspace
- natbib *and* bibentry
- optparams
- placeins
- mathpazo*
- helvet*
- fontenc
- beramono*
- fancyhdr
- xcolor
- textcomp
- titlesec
- titletoc

A. Housing Prices Appendix

`figures/housing/missing_heatmap.pdf`

Figure A.1: XXXX **TODO!**

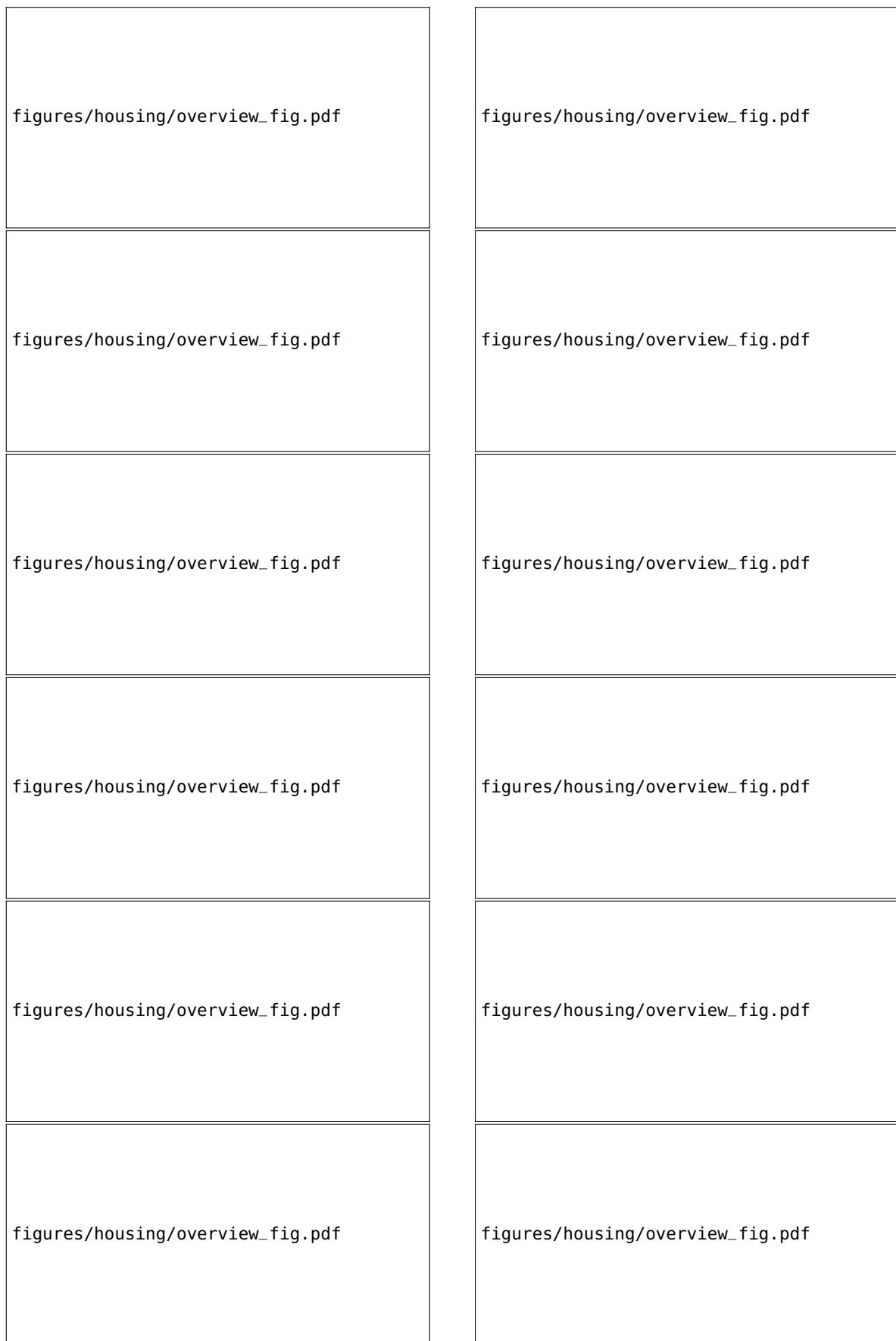


Figure A.2: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

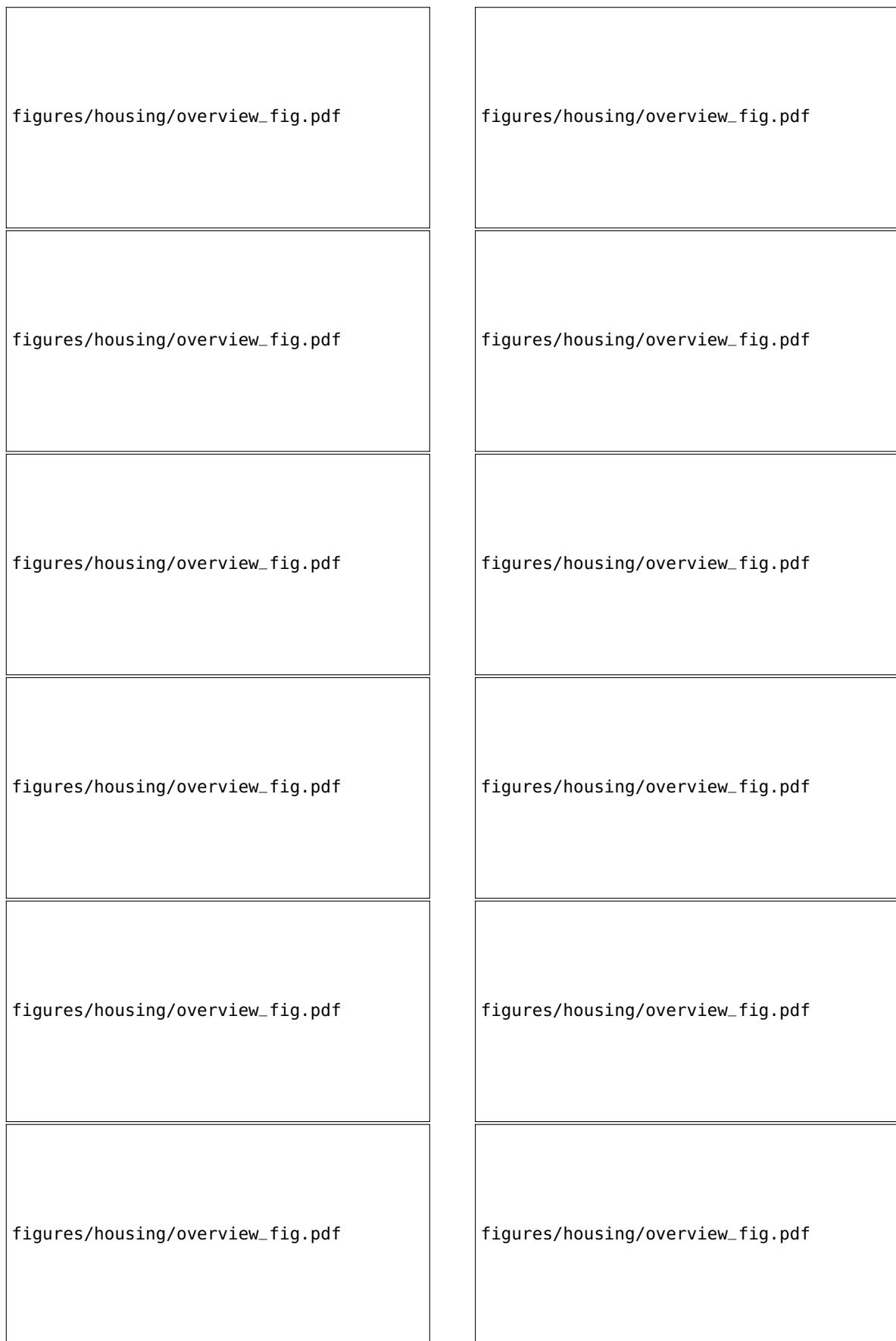


Figure A.3: Distributions the 168 input variables (excluding ID and Vejnavn).



Figure A.4: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).



Figure A.5: Distributions the 168
input variables (excluding `ID` and
`Vejnavn`).

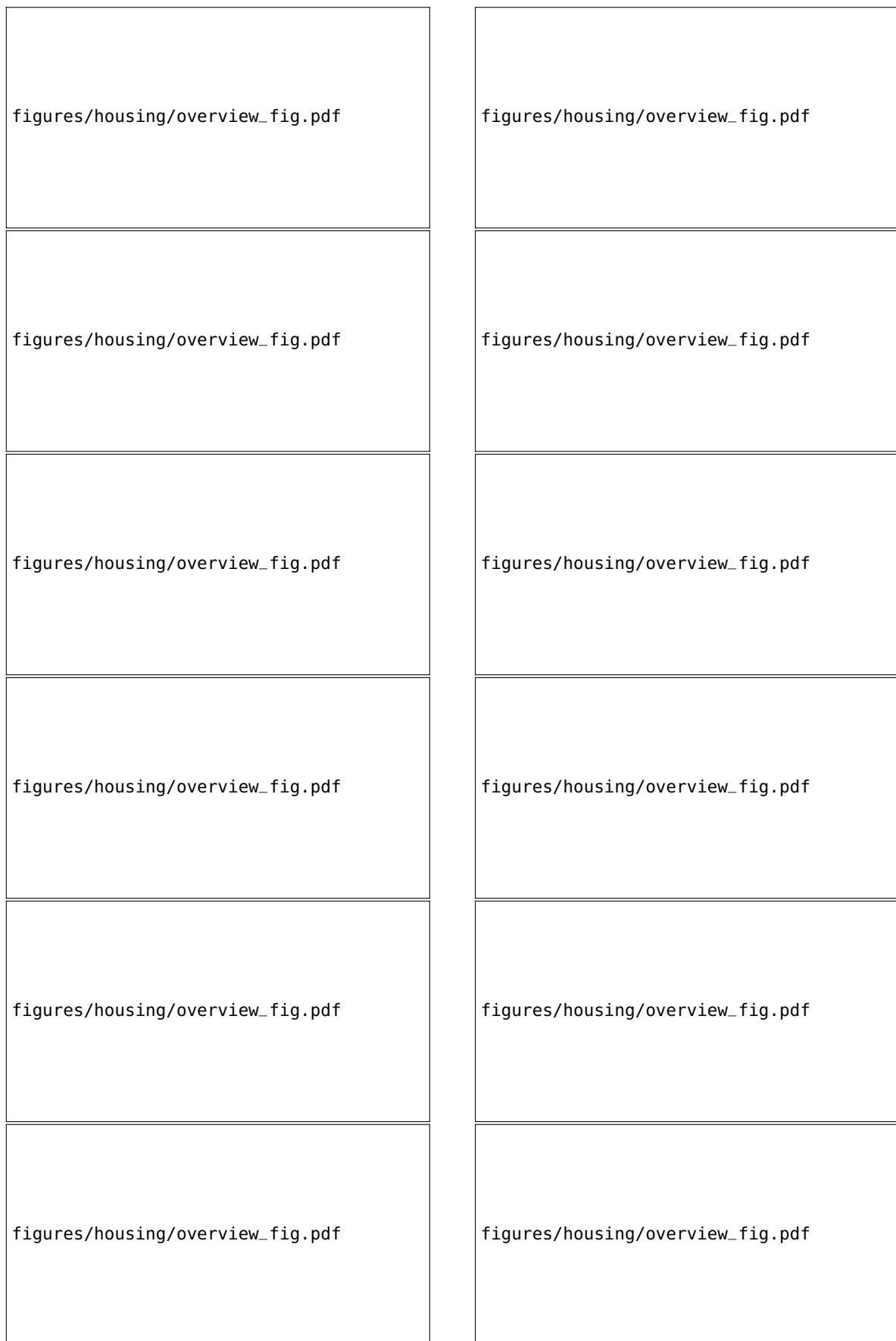


Figure A.6: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

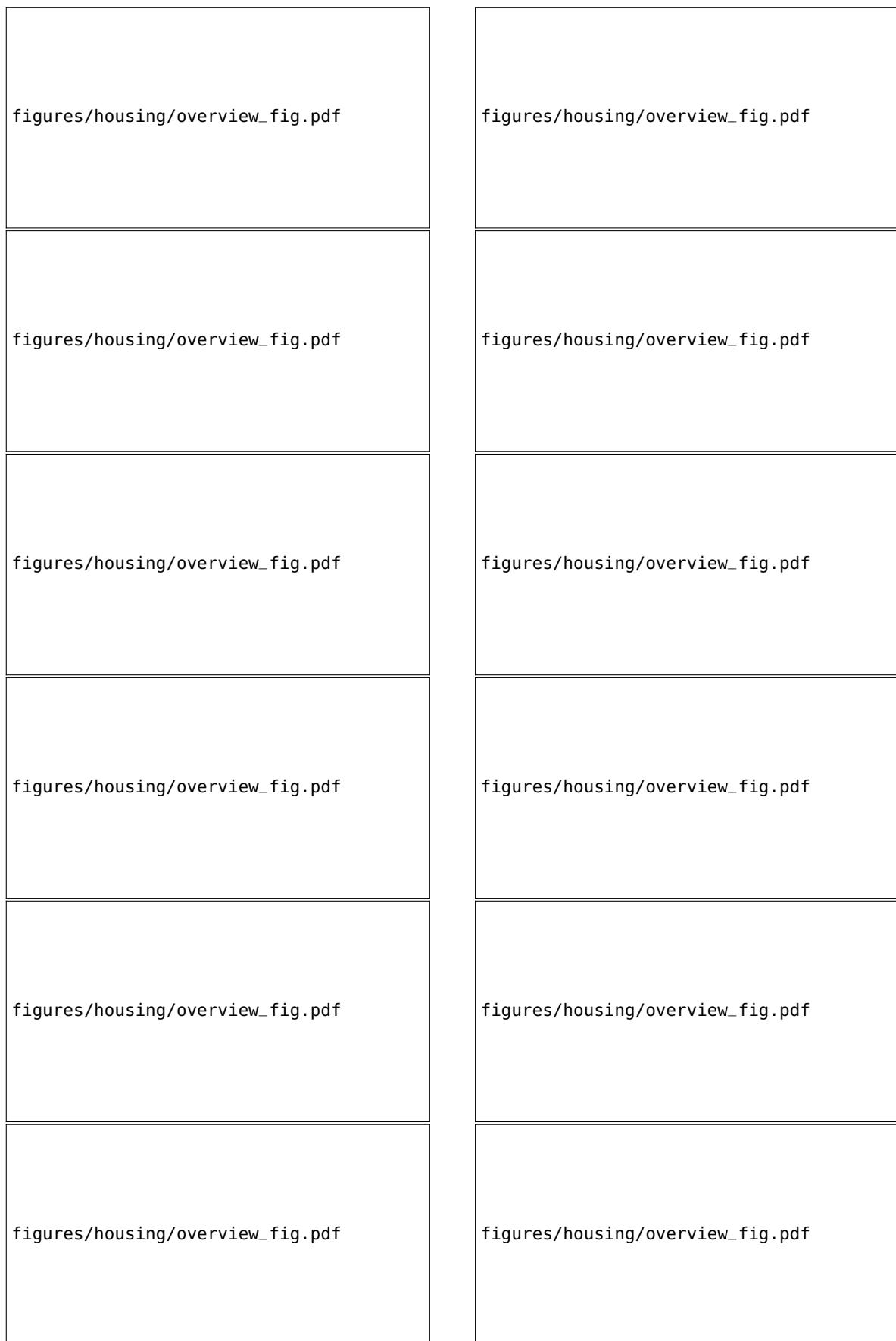


Figure A.7: Distributions the 168 input variables (excluding ID and Vejnavn).



Figure A.8: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

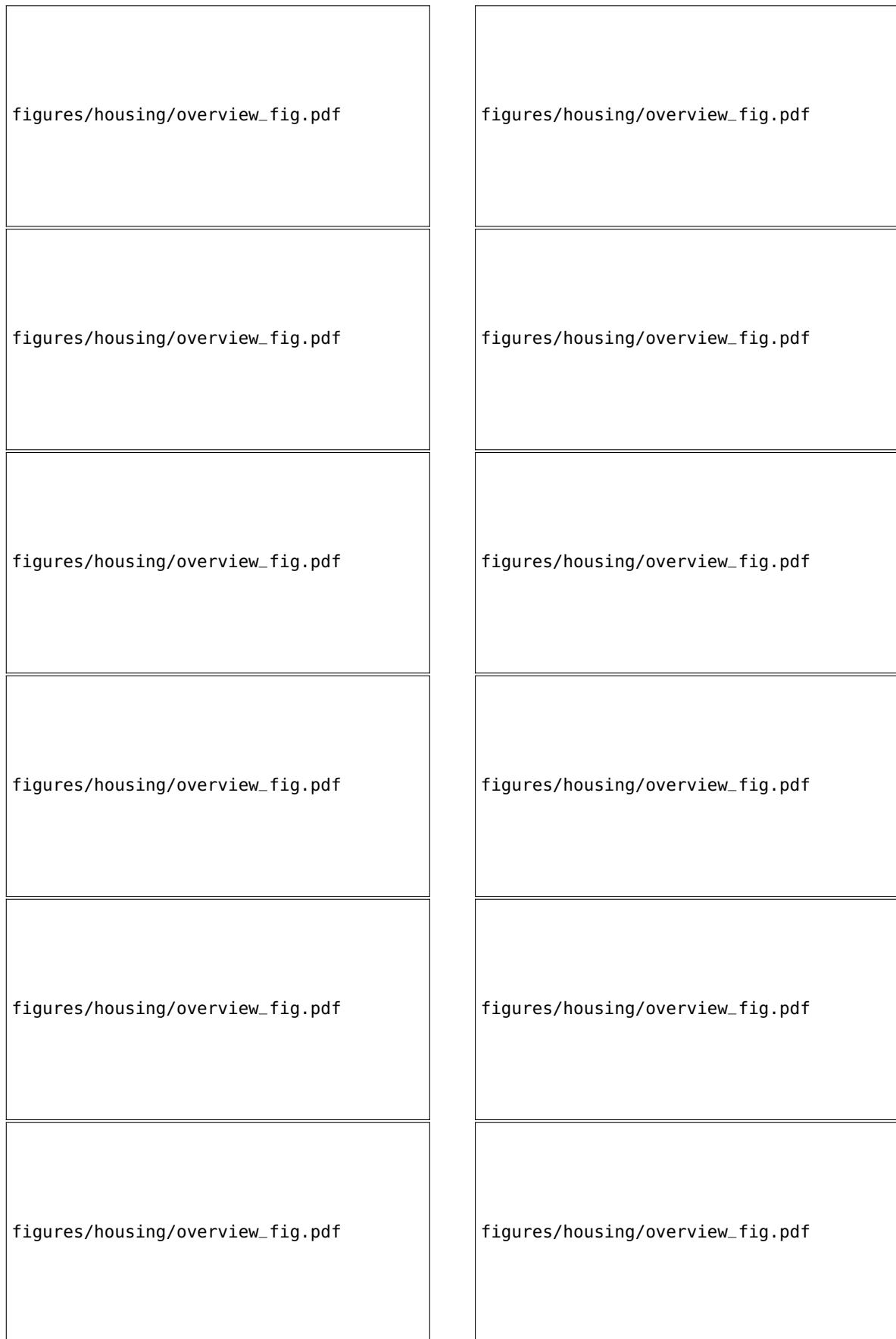


Figure A.9: Distributions the 168 input variables (excluding ID and Vejnavn).

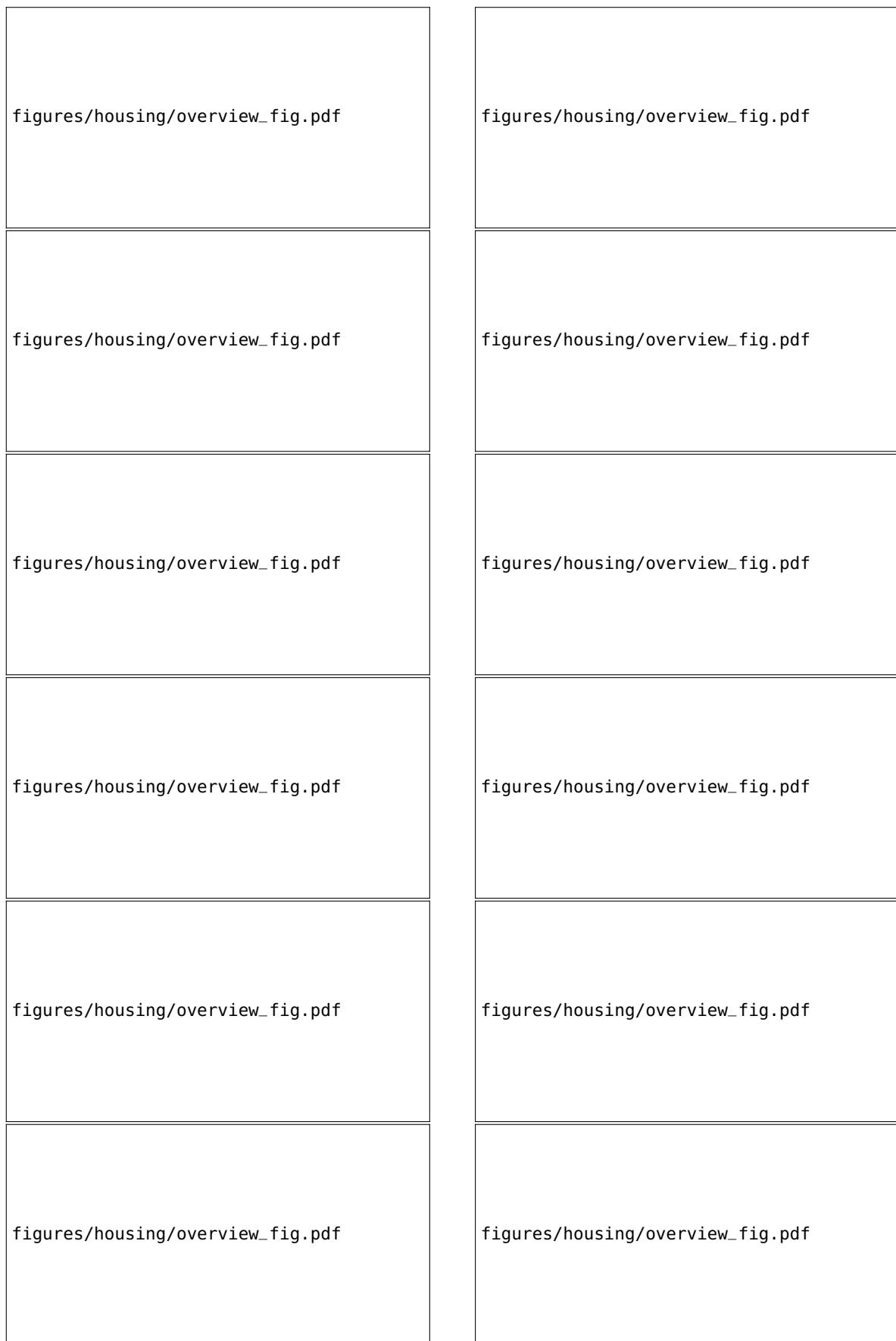


Figure A.10: Distributions the 16 input variables (excluding `ID` and `Vejnavn`).

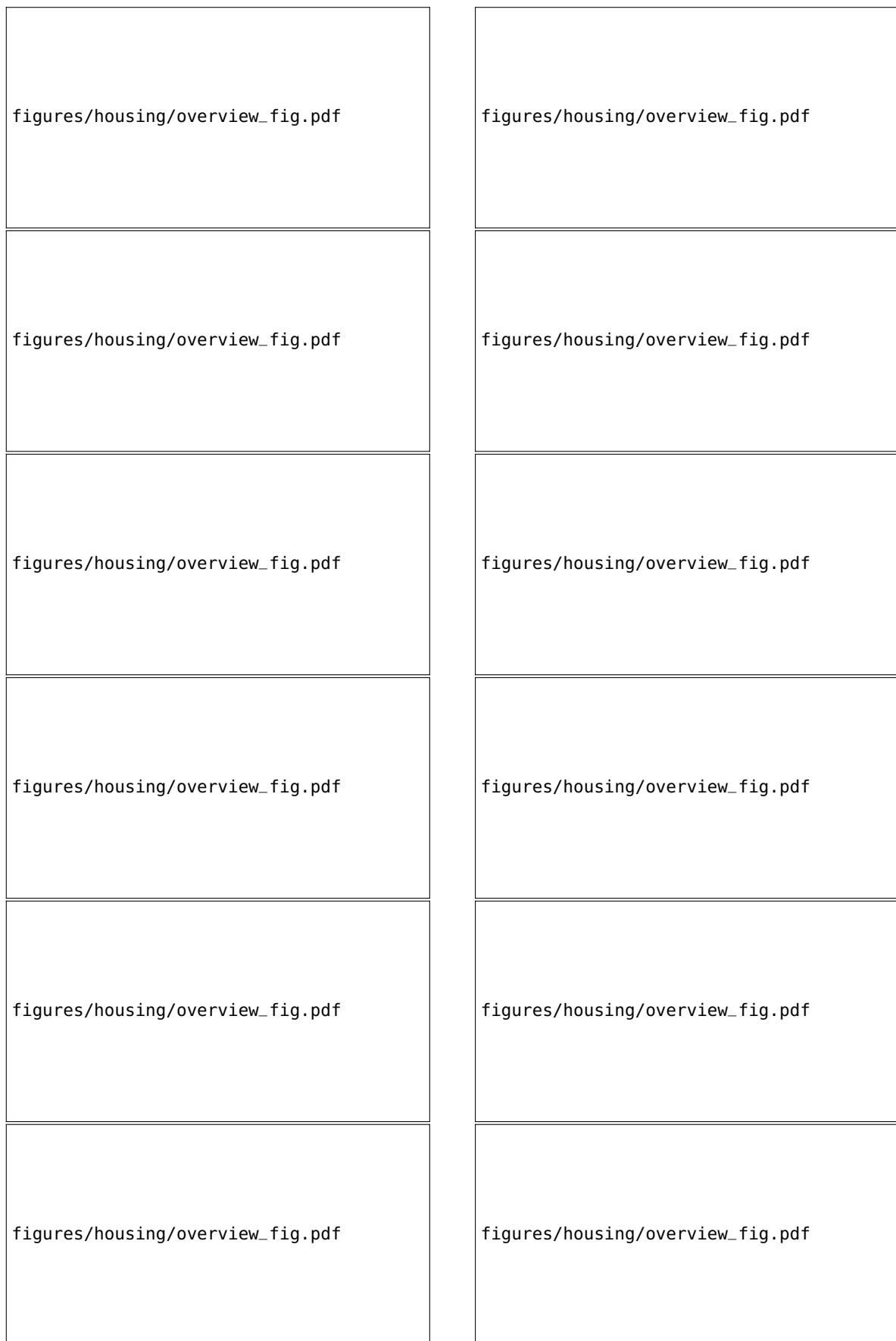


Figure A.11: Distributions the 168 input variables (excluding ID and Vejnavn).

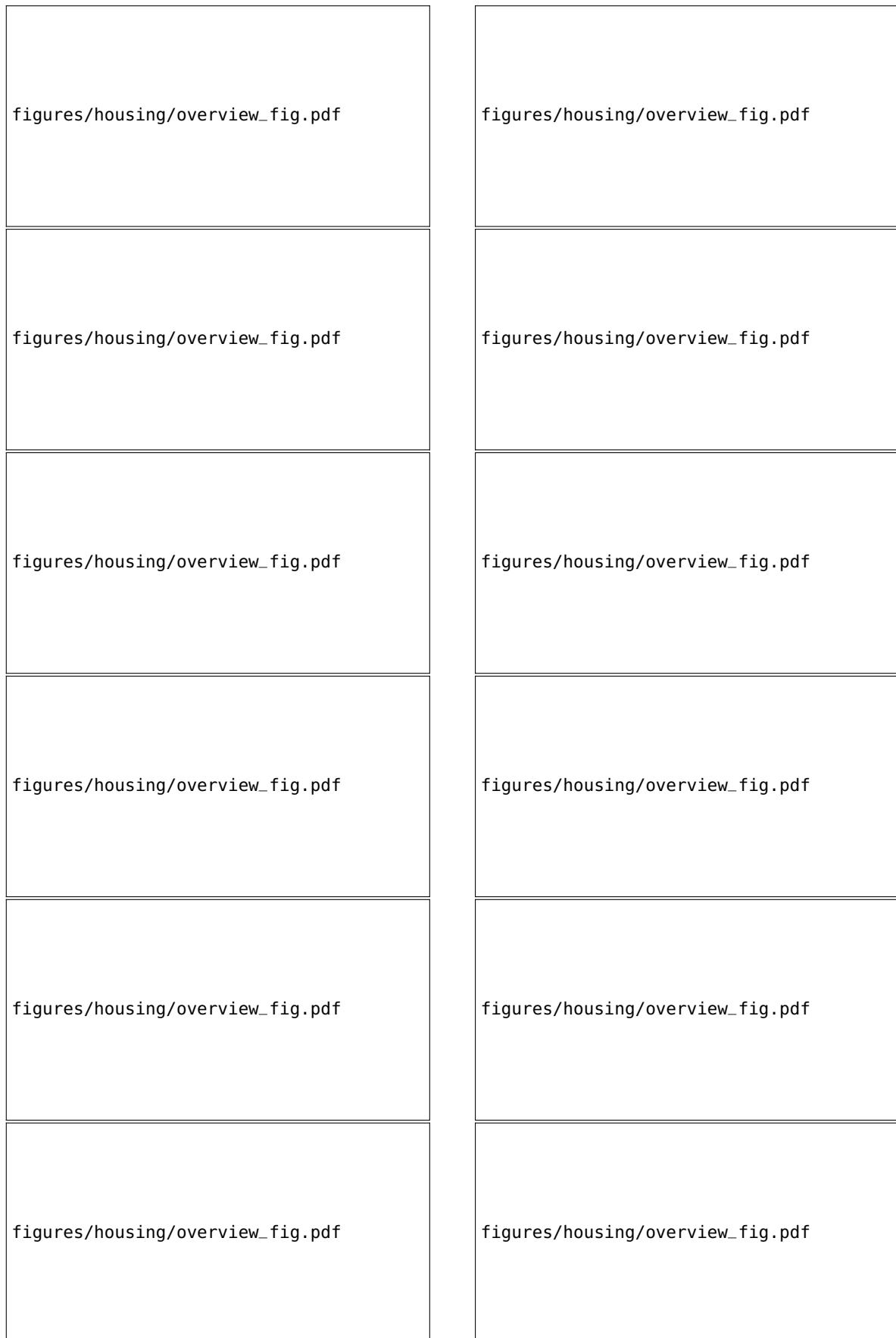


Figure A.12: Distributions the 16 input variables (excluding `ID` and `Vejnavn`).



Figure A.13: Distributions the 168 input variables (excluding ID and Vejnavn).



Figure A.14: Distributions the 16 input variables (excluding `ID` and `Vejnavn`).

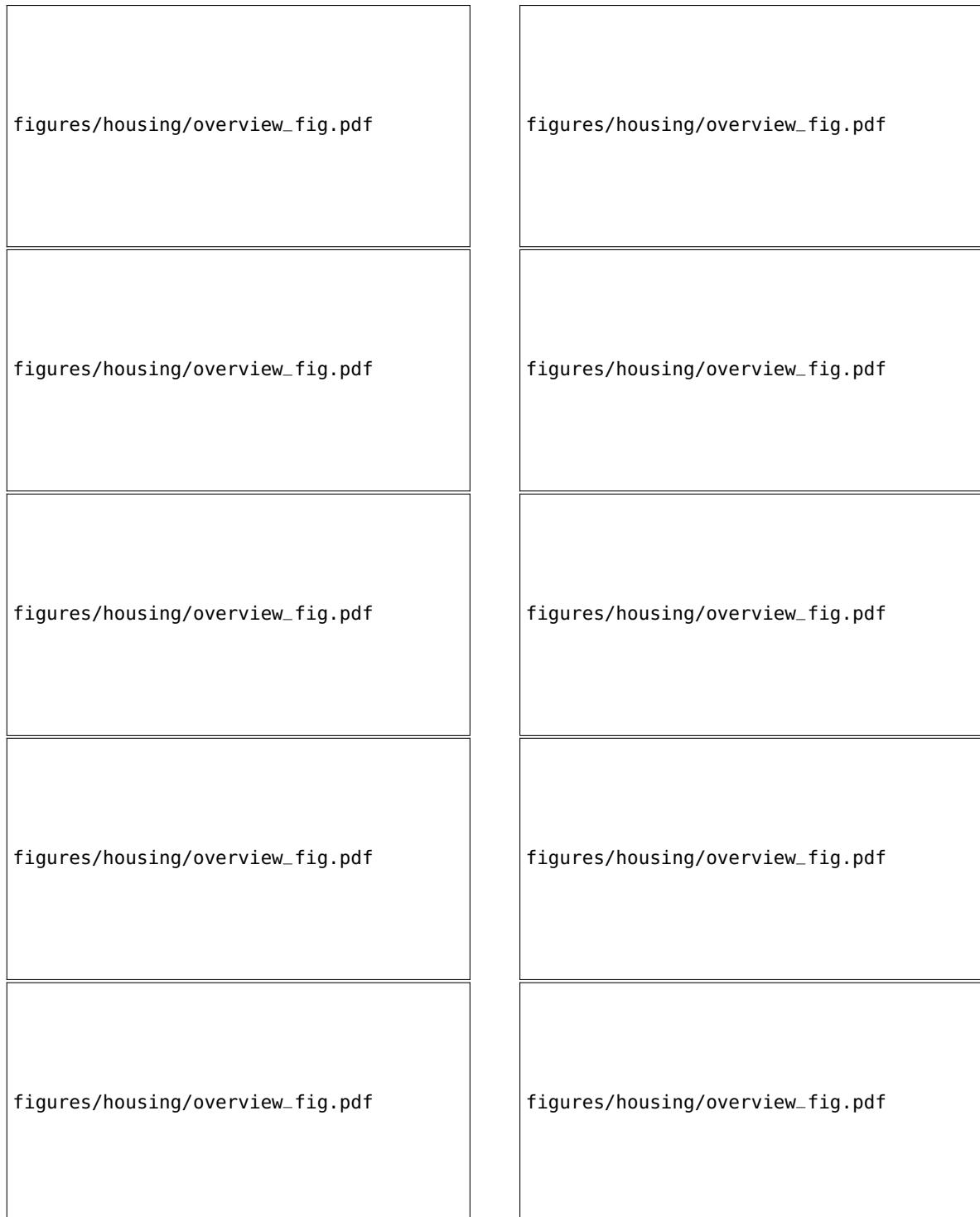


Figure A.15: Distributions the 168 input variables (excluding ID and Vejnavn).

OISSalgsType	GeoLandsdelNr	GeoRegionNr
GeoKommuneNr	GeoPostNr	PostHovedNr
Etage	SognKode	ZoneKode
GisX_WGS84	GisY_WGS84	EjendomsNr
ArealBolig	ArealKaelder	ArealGrund
BeregnetAreal	AntalRum	AntalEtager
ByggeAAr	OmbygningsAAr	EnergiLov
UdenAnnoncering	ProjektSalg	LiggetidAktuel
LiggetidSamlet	Ejerudgift	Bygning_GOP_OpfoerselesAAr
Bygning_GOP_OmbygningsAAr	Bygning_GOP_EjerLavKode	Bygning_GOP_EjerForholdKode
Bygning_GOP_AntalEtagerUKldTag	Bygning_GOP_AntalBoligMedKoekken	Bygning_GOP_AntalBoligUdenKoekken
Bygning_MOP_YdervaegKode	Bygning_MOP_TagdaekningKode	Bygning_MOP_KildeMatrKode
Bygning_IOP_VarmeinstalKode	Bygning_IOP_OpvarmningKode	Bygning_IOP_SuppVarmeKode
Bygning_VOP_VandforsyningKode	Bygning_AGP_Bebygget	Bygning_AGP_HerafAffaldsrsum
Bygning_AGP_HerafIndbGarage	Bygning_AGP_HerafIndbCarport	Bygning_AGP_HerafIndbUdhus
Bygning_AGP_HerafUdstue	Bygning_AGP_Overdaekket	Bygning_AHB_SamletBygning
Bygning_AHB_HerafUdvIsolering	Bygning_AHB_KaelderSamlet	Bygning_AHB_KaelderU125
Bygning_AHB_TagSamlet	Bygning_AHB_TagBeboelse	Bygning_AHB_LukkedeOverdaekning
Bygning_AAV_SamletBolig	Bygning_AAV_HerafKaelder	Bygning_AAV_Adgangs
Bygning_AAV_Andet	Bygning_AAV_AABenOverdaekning	Enhed_Ejendomsnr
Enhed_GOP_AnvendelseKode	Enhed_GOP_AntVaerelseErv	Enhed_GOP_AntToilet
Enhed_GOP_AntBad	Enhed_GOP_EnergiKode	Enhed_AAV_Erhverv
Enhed_AAV_Andet	Enhed_AAV_FeallesAdg	Enhed_AAV_AabenOverdaek
Enhed_AAV_LukketOverdaek	Historisk_SalgsType1	Historisk_SalgsPris1
Historisk_SalgsType2	Historisk_SalgsPris2	Historisk_SalgsType3
Historisk_SalgsPris3	EjdVurdering_VurderingAAr0	EjdVurdering_EjendomsVaerdi0
EjdVurdering_GrundVaerdi0	EjdVurdering_StuehusVaerdi0	EjdVurdering_StueGrundVaerdi0
EjdVurdering_VurderingAAr1	EjdVurdering_EjendomsVaerdi1	EjdVurdering_GrundVaerdi1
EjdVurdering_StuehusVaerdi1	EjdVurdering_StueGrundVaerdi1	EjdVurdering_VurderingAAr2
EjdVurdering_EjendomsVaerdi2	EjdVurdering_GrundVaerdi2	EjdVurdering_StuehusVaerdi2
EjdVurdering_StueGrundVaerdi2	EjdVurdering_VurderingAAr3	EjdVurdering_EjendomsVaerdi3
EjdVurdering_GrundVaerdi3	EjdVurdering_StuehusVaerdi3	EjdVurdering_StueGrundVaerdi3
EjdVurdering_VurderingAAr4	EjdVurdering_EjendomsVaerdi4	EjdVurdering_GrundVaerdi4
EjdVurdering_StuehusVaerdi4	EjdVurdering_StueGrundVaerdi4	Tinglyst_AntEjere
Tinglyst_MindsteAndel	Tinglyst_StoersteAndel	Afstand_Faengsel
Afstand_Hede	Afstand_Hoejspaendingsledning	Afstand_Industri
Afstand_JernbaneSynlig	Afstand_Kirke	Afstand_Kirkegaard
Afstand_Kyst	Afstand_Landingsbane	Afstand_Motorvej
Afstand_MotorvejTilFraKoersel	Afstand_RekreativtOmraade	Afstand_Rigsgrænse
Afstand_Sportsanlaeg	Afstand_Strand	Afstand_Vindmoelle
Kommune_Indbyggertal	Kommune_SkatteProcent	Kommune_Vuggestuer
Kommune_Boernehaver	Kommune_IntegreredeInstitutioner	Kommune_Folkeskoler
Kommune_Grundskyld	dag_maaned	maaned
aar	SalgsDato_siden0	Historisk_SalgsDato1_siden0
Historisk_SalgsDato2_siden0	Historisk_SalgsDato3_siden0	HusNr_tal
HusNr_bogstav	SidedoerNummer	Vej
ArealVaegtet_same_as_BeregnetAreal	ByggeAAr_diff	OmbygningsAAr_diff
Energi	Prophet_index	

Table A.1: XXX TODO!

figures/housing/correlations_all.pdf

Figure A.16: XXXX **TODO!**.

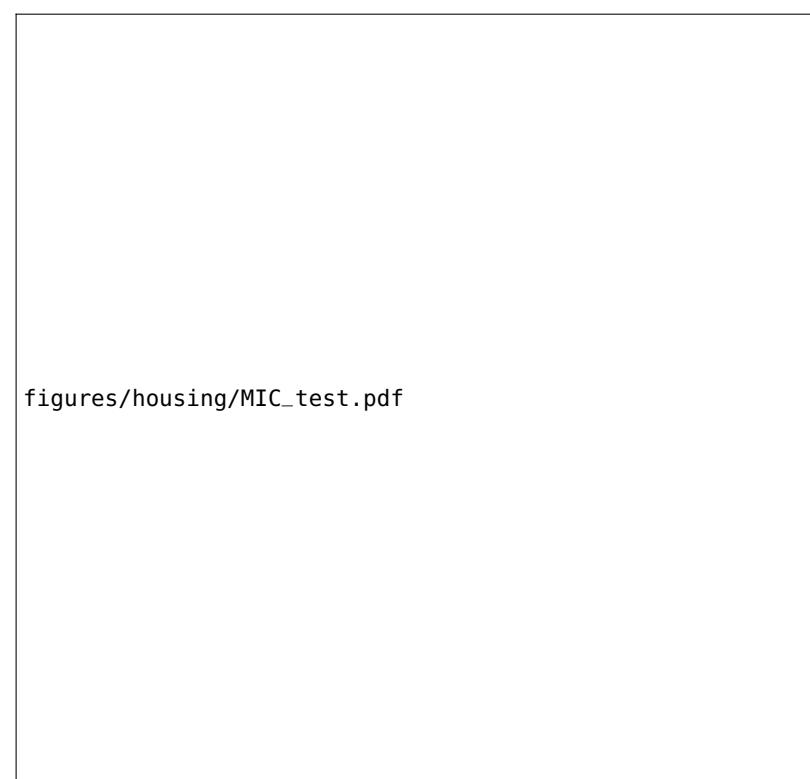


Figure A.17: MIC non-linear correlation.

`figures/housing/MIC_test.pdf`

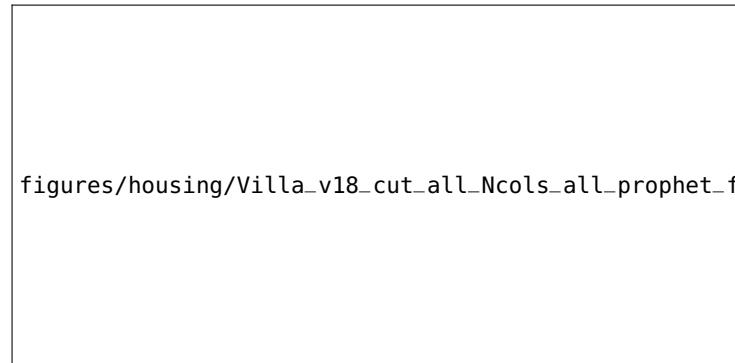


Figure A.18: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median of each week is used as input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the $1 - \sigma$ confidence interval.

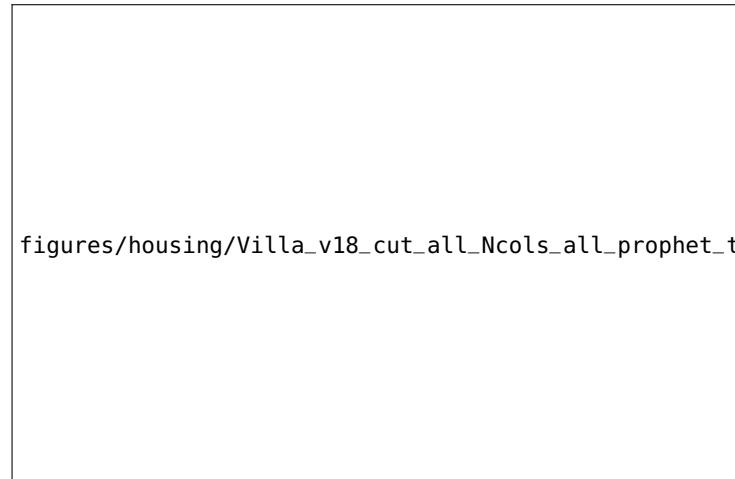


Figure A.19: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that the square meter price is higher during the Summer months compared to the Winter months, however, compared to the overall trend this effect is minor (< 10%).

`figures/housing/Villa_v18_cut_all_Ncols_all_prophet_trends.pdf`

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	226	141	0.1664
2.5	False	201	115	0.1770
5	True	301	90	0.1623
5	False	375	82	0.1786
10	True	318	97	0.1618
10	False	226	56	0.1893
20	True	265	81	0.1626
20	False	687	124	0.1799
∞	True	405	110	0.1600
∞	False	94	32	0.2036

Table A.2: Rmse-ejerlejlighed-
appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	333	75	0.1595
2.5	False	496	57	0.1523
5	True	280	66	0.1606
5	False	734	96	0.1513
10	True	367	83	0.1618
10	False	351	52	0.1590
20	True	269	62	0.1609
20	False	333	49	0.1587
∞	True	388	83	0.1595
∞	False	268	42	0.1648

Table A.3: Logcosh-ejerlejlighed-
appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	293	56	0.1598
2.5	False	814	101	0.1466
5	True	304	68	0.1610
5	False	923	110	0.1468
10	True	266	62	0.1610
10	False	770	97	0.1450
20	True	288	65	0.1613
20	False	967	117	0.1467
∞	True	340	72	0.1601
∞	False	807	99	0.1480

Table A.4: Cauchy-ejerlejlighed-
appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	285	64	0.1628
2.5	False	718	90	0.1517
5	True	257	62	0.1600
5	False	702	91	0.1499
10	True	272	62	0.1601
10	False	771	99	0.1466
20	True	260	61	0.1603
20	False	876	107	0.1486
∞	True	310	69	0.1584
∞	False	973	115	0.1459

Table A.5: Welsch-ejerlejlighed- appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	229	54	0.1601
2.5	False	304	45	0.1577
5	True	205	54	0.1629
5	False	343	51	0.1549
10	True	257	61	0.1596
10	False	332	47	0.1573
20	True	272	62	0.1608
20	False	403	56	0.1537
∞	True	344	74	0.1578
∞	False	453	59	0.1527

Table A.6: Fair-ejerlejlighed- appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	458	339	0.1983
2.5	False	844	439	0.1913
5	True	733	478	0.1968
5	False	1126	541	0.1888
10	True	434	310	0.1999
10	False	917	444	0.1884
20	True	398	286	0.2013
20	False	1206	575	0.1867
∞	True	730	505	0.1977
∞	False	1264	625	0.1876

Table A.7: Rmse-villa- appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	346	223	0.2018
2.5	False	1095	415	0.1877
5	True	618	331	0.1976
5	False	1601	546	0.1847
10	True	506	280	0.1990
10	False	1160	400	0.1873
20	True	445	269	0.2011
20	False	1313	497	0.1876
∞	True	432	258	0.1982
∞	False	2151	739	0.1842

Table A.8: Logcosh-villa- appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	434	244	0.1991
2.5	False	1007	356	0.1872
5	True	350	208	0.1999
5	False	1130	389	0.1858
10	True	436	240	0.1992
10	False	1183	394	0.1850
20	True	397	242	0.2003
20	False	1514	542	0.1833
∞	True	449	257	0.1992
∞	False	1351	470	0.1844

Table A.9: Cauchy-villa-appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	867	440	0.1960
2.5	False	835	300	0.1897
5	True	301	184	0.2035
5	False	893	312	0.1878
10	True	341	200	0.2014
10	False	1113	390	0.1869
20	True	338	209	0.2022
20	False	1212	424	0.1875
∞	True	579	321	0.1970
∞	False	1497	509	0.1837

Table A.10: Welsch-villa-appendix.

Half-life	\log_{10}	N_{trees}	Time [s]	f_{eval}
2.5	True	508	278	0.1956
2.5	False	862	301	0.1882
5	True	506	278	0.1957
5	False	1357	462	0.1835
10	True	875	436	0.1946
10	False	954	325	0.1861
20	True	763	402	0.1943
20	False	1256	435	0.1840
∞	True	535	303	0.1973
∞	False	1337	456	0.1844

Table A.11: Fair-villa-appendix.

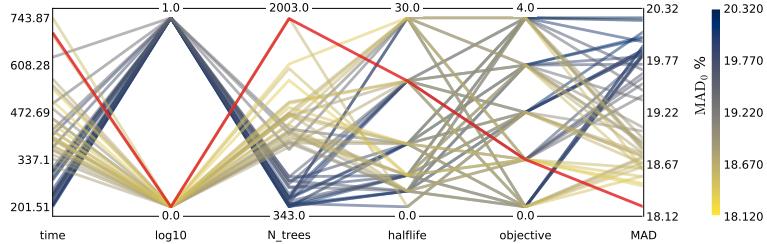


Figure A.20: Hyperparameter optimization results of the housing model for houses. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of that hyperparameter as measured by MAD_0 from highest MAD in dark blue to lowest AUC in yellow. The **single best hyperparameter** is shown in red. For the hyperparameter `log10` 0 means False and 1 means True, for `Halftime` ∞ is mapped to 30, and for `objektive` the functions Cauchy (0), Fair (1), LogCosh (2) SquaredError (3), and Welsch (4) are mapped to the integers in the parentheses.

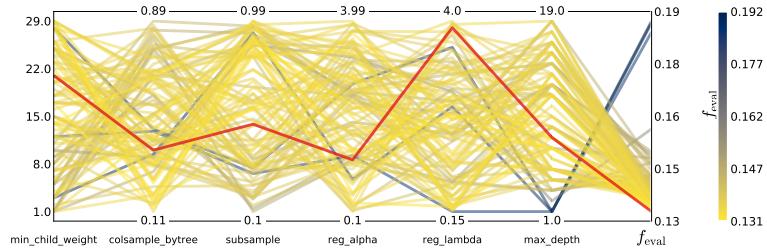


Figure A.21: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

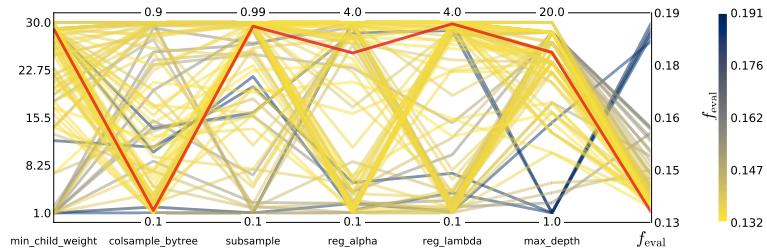


Figure A.22: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for Bayesian optimization as hyperparameter optimization.

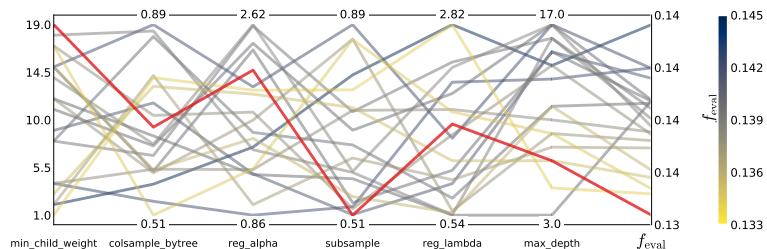


Figure A.23: Hyperparameter optimization results of XGBoost parameters of the housing model for houses shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

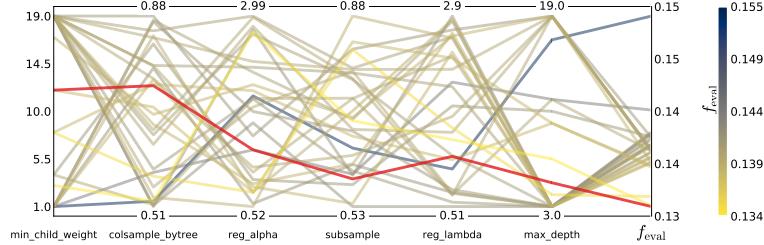


Figure A.24: Hyperparameter optimization results of XGBoost parameters of the housing model for houses shown as parallel coordinates. Here shown for Bayesian optimization as hyperparameter optimization.

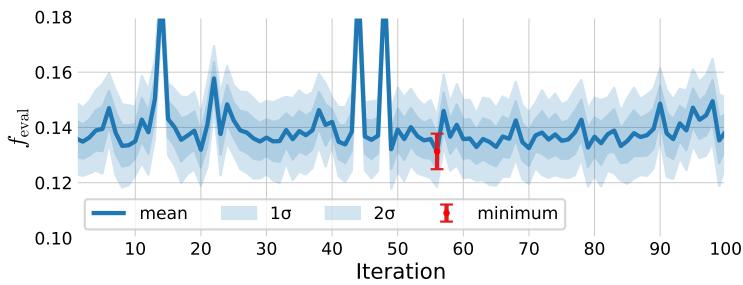


Figure A.25: XXX of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

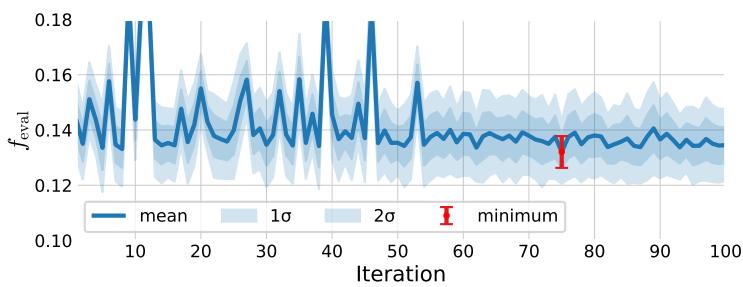


Figure A.26: XXX of the housing model for apartments shown as parallel coordinates. Here shown for Bayesian optimization as hyperparameter optimization.

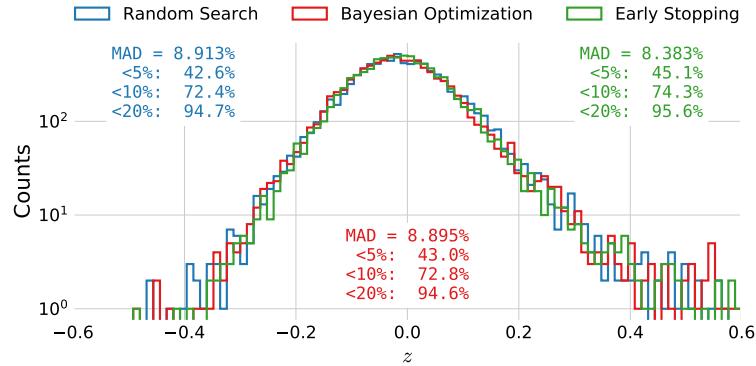


Figure A.27: Histogram of z -values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using **Random Search** (RS) is shown in blue, for **Bayesian Optimization** (BO) in red. After finding the best model, BO in this case, the model is retrained using **early stopping**, the performance of which is shown in green.

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ
Train	6.35	56.22	83.41	97.08	0.00902 ± 0.00068
Test	8.38	45.06	74.32	95.58	-0.00820 ± 0.00115
2019	9.12	42.63	71.36	93.65	0.00297 ± 0.00235

Table A.12: XXX ejer tight

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	μ
Train	15.63	25.65	47.89	75.82	0.04543 ± 0.00080
Test	16.49	24.30	45.77	75.19	0.01686 ± 0.00194
2019	17.17	23.67	44.25	73.54	0.02056 ± 0.00279

Table A.13: XXX villa tight

B. Quarks vs. Gluons Appendix

