CHRISTIAN MICHELSEN
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

Christian Michelsen

https://github.com/christianmichelsen

First printing, December 2019

Abstract

Here will be a decent abstract at some point^{TM}.

Contents

Foreword

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. 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. The aim of this section is to provide an initial overview of the scope and relationship of the two sub-projects.

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 [13, 37]) 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 market[1] and have been very helpful in the continuos process of providing data. It 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 Gudiksen[2] 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 physics, was the main part of the project. Not only was most of the

[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.

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 [5]. During LEP1 it was primarily the decay channels of the Z-boson that were probed where especially the $Z \to q\bar{q}g$ and $Z \to q\bar{q}gg$ were examined in this thesis. The distributions of these gluon jets and the difference between Data[3] and Monte-Carlo (MC) that are of interests to the theoreticians that develop the MC-models. At first an improved b-tagging algorithm was developed. 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 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!**.

[3] Where "Data" with capital D refers to the actual, measured data and "data" refers to any arbitrary selection of data.

The thesis is structured such that chapter 2 introduces the needed theoretical Machine Learning (ML) background needed for understanding the methods used throughout the thesis, chapter 3 describes the housing prices subproject 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, chapter 5 explains analysis of the main project in this thesis, i.e. the quark gluon analysis, and finally the two chapters chapter 6 and chapter 7 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 complementary instances of same underlying problem: teaching computers how to find patterns automatically in high-dimensional data and should thus not be seen as two independent projects. This also highlights 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.

Part I

The first part of this thesis deals with the introductory theory of machine learning and its predictive power in estimating Danish housing prices.

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[79] such as NumPy, Matplotlib, Pandas, Scikit-Learn, Scipy has been used extensively and should thus also be mentioned.

2. 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 collection of different subjects 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 [77] which has seen a plethora of use cases in recent years.

2.1 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 [11] and the graduate course Advanced Topics in Machine Learning [1] at the computer science institute[1] 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.

[1] Datalogisk Institut Københavns Universitet, DIKU.

Overall there are two main branches within machine learning: *supervised* and *unsupervised*[2]. 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.

[2] Also known as "self-supervised" or "predictive" learning.

2.2 *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 $\mathbf{x}_i = [x_1, x_2, \ldots, x_M]^T \in \mathcal{X}$ for the ith 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 $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \ldots, \mathbf{x}_N^T]$ 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 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* problem[3] e.g. predicting whether or not a particle is a quark or not.

Without any loss of 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 for each observation $\mathbf{x}$. 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 $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$. The optimal hypothesis, h^*, is chosen among a set of K candidate hypotheses $\mathcal{H} = \{h_1, h_2, \ldots, h_K\}$, and hopefully h^* will be a good approximation of f: $h^* \approx f$. A schematic overview of this process can be seen in Figure 2.1.

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 good as possible.

To quantify the statement "as good 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 [78] and the expected loss[5]:

$$L(h) = \mathbb{E}\left[\ell(h(\mathbf{x}), y)\right] = \int \ell(h(\mathbf{x}), y)\, dP(\mathbf{x}, y). \tag{2.1}$$

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 loss[6] of h on $\mathcal{D}_{\text{train}}$:

$$\hat{L}(h, S) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(\mathbf{x}_i), y_i), \tag{2.2}$$

which is an approximation of $L(h)$ based on the training data available. Now the optimal hypothesis h^* can defined:

$$h^* = \arg\min_{h \in \mathcal{H}} \hat{L}(h, S). \tag{2.3}$$

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

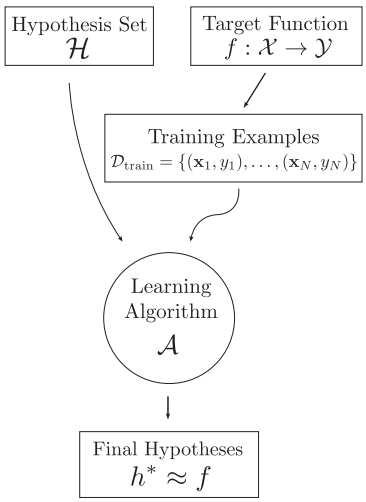


Figure 2.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

[4] This is one of the two key assumptions of statistical learning theory, the other being that future events are coming from the same distrubution 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.

2.3 Generalization Bound

In section 2.2 the method of selecting the optimal hypotheses 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:

$$G(h) = \hat{L}(h, S) - L(h). \tag{2.4}$$

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 $E_1, E_2, \ldots$ (not necessarily independent):*

$$\mathbb{P}\left\{\bigcup_{i\leq 1} E_i\right\} \leq \sum_{i\leq 1} \mathbb{P}\left\{E_i\right\}. \tag{2.5}$$

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 $Z_1, \ldots, Z_N$ 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 $\epsilon > 0$:*

$$\mathbb{P}\left\{\frac{1}{N}\sum_{i=1}^{N} Z_i - \mu \geq \epsilon\right\} \leq e^{-2N\epsilon^2} \tag{2.6}$$

and

$$\mathbb{P}\left\{\mu - \frac{1}{N}\sum_{i=1}^{N} Z_i \geq \epsilon\right\} \leq e^{-2N\epsilon^2}. \tag{2.7}$$

When using the union bound on equation (2.6) and equation (2.7) we arrive at Hoeffding's (two) sided inequality:

Lemma 3 (The two-sided Hoeffding's inequality). *Let $Z_1, \ldots, Z_N$ 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 $\epsilon > 0$:*

$$\mathbb{P}\left\{\left|\frac{1}{N}\sum_{i=1}^{n} Z_i - \mu\right| \geq \epsilon\right\} \equiv \mathbb{P}\left\{|\hat{\mu} - \mu| \geq \epsilon\right\} \leq 2e^{-2N\epsilon^2}, \tag{2.8}$$

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

Assuming that the loss $\ell(\hat{y}, y)$ is bounded in the $[0,1]$ interval[7], $\ell(\hat{y}, y) \in [0,1]$ 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 3 and equation (2.2) we see that $\hat{\mu} = \hat{L}(h, S)$, and similar for equation (2.1): $\mu = L(h)$. We then see that the generalization error is bounded:

$$\mathbb{P}\left\{|G(h)| \geq \epsilon\right\} = \mathbb{P}\left\{\left|\hat{L}(h,S) - L(h)\right| \geq \epsilon\right\} \leq 2e^{-2N\epsilon^2}. \tag{2.9}$$

This equation provides a bound on the difference between the empirical loss and the expected loss. The generalization bound can be rewritten in terms of δ:

$$\delta \equiv 2e^{-2N\epsilon^2} \in (0,1) \quad \Rightarrow \quad \epsilon = \sqrt{\frac{\ln\frac{2}{\delta}}{2N}} \quad \Rightarrow \tag{2.10}$$

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:*

$$\mathbb{P}\left\{\left|\hat{L}(h,S) - L(h)\right| \geq \sqrt{\frac{\ln\frac{2}{\delta}}{2N}}\right\} \leq \delta. \tag{2.11}$$

Equation (2.11) can be read as the probability of the generalization error being larger than $\sqrt{\frac{\ln\frac{2}{\delta}}{2N}}$ is δ or less, or, similarly, that with probability greater than $1-\delta$:

$$\left|\hat{L}(h,S) - L(h)\right| \leq \sqrt{\frac{\ln\frac{2}{\delta}}{2N}}. \tag{2.12}$$

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, inversely[8] related to on the certainty[9] 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 K hypotheses: $|\mathcal{H}| = K$. We thus have $[h_1, h_2, \ldots, h_K]$ 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 $|\mathcal{H}| = K$. Then for any $\delta \in (0,1)$ we have:*

$$\mathbb{P}\left\{\exists h \in \mathcal{H} : \left|\hat{L}(h,S) - L(h)\right| \geq \sqrt{\frac{\ln\frac{2K}{\delta}}{2N}}\right\} \leq \delta. \tag{2.13}$$

[7] Which it is for classification, however, it can be extended in a similar fasion 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:

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

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

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

By making the substitution $\delta = K\delta'$ we arrive at equation (2.13). □

As we did in equation (2.12), equation (2.13) can also be read as with probability greater than $1 - \delta$ then for all $h \in \mathcal{H}$:

$$\left|\hat{L}(h, S) - L(h)\right| \leq \sqrt{\frac{\ln \frac{2K}{\delta}}{2N}}. \tag{2.14}$$

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

2.3.1 *Generalization Bound for infinite hypotheses*

Section 2.3 dealt with the case of a single hypotheses h and a finite set of candidate hypotheses $h \in \mathcal{H}, |\mathcal{H}| = K$. When K goes towards infinity the generalization bound goes to infinity and the bound becomes useless. However, even simple models such as a linear classifier[10] that predicts $\hat{y} = 1$ when the dot product $\mathbf{w}^T\mathbf{x}$ is positive and $\hat{y} = -1$ when it is negative, has $|\mathcal{H}| = \infty$. Since an infinite number of hypotheses $h(\mathbf{w})$ exists, 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 introduce[11] the Vapnik-Chervonenkis (VC) generalization bound. The VC-bound is based on the so-called VC-dimension of the hypothesis space $\mathcal{H}$: $d_{\text{VC}}(\mathcal{H}) = d_{\text{VC}}$. 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 M-dimensional linear classifier defined above[12] is $d_{\text{VC}} = M$ for $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^M$.

[10] Also known as the perceptron. Often includes a constant offset b as well which is omitted here for brevity. The functional form of this function is $f(x) = \text{sign}(\mathbf{w}^T\mathbf{x})$.

[11] For proof, see: Abu-Mostafa et al. [11]

[12] In the general case when the offset b is included: $d_{\text{VC}} = M + 1$.

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 $1-\delta$:*

$$L(h) \leq \hat{L}(h,S) + \sqrt{\frac{8}{N}\ln\left(\frac{4}{\delta}\left((2N)^{d_{\text{VC}}}+1\right)\right)}. \tag{2.15}$$

Equation (2.15) 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 $\Omega(N,\mathcal{H},\delta)$:

$$\Omega(N,\mathcal{H},\delta) = \sqrt{\frac{8}{N}\ln\left(\frac{4}{\delta}\left((2N)^{d_{\text{VC}}(\mathcal{H})}+1\right)\right)}. \tag{2.16}$$

As the hypothesis space complexity d_{VC} grows, the model complexity penalty increases 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 2.2. 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.

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

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

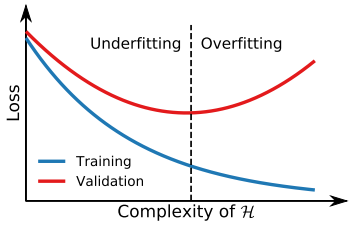


Figure 2.2: Illustration of the empirical loss as a function of model complexity. The training error is shown in blue and validation error in red.

2.4 *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 complementary of each other but can be taken advantage of in a combination. In this section model regularization will be introduced in subsection 2.4.1, cross validation in subsection 2.4.2, and early stopping in subsection 2.4.3.

2.4.1 *Model Regularization*

One of the earliest methods developed for preventing overfitting was model regularization. A. N. Tikhonov [73] 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:

$$\begin{aligned}\hat{\boldsymbol{\beta}}_{\text{LS}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y}-\mathbf{f}(\mathbf{X})\|_2^2 \\ \mathbf{f}(\mathbf{X}) = \mathbf{X}\boldsymbol{\beta},\end{aligned} \tag{2.17}$$

where $\mathbf{y}$ is the vector of values we are trying to predict[15], $\mathbf{X} \in \mathbb{R}^{N\times M}$ the matrix of input variables, $\hat{\boldsymbol{\beta}}_{\text{LS}}$ the vector of unknown coefficients[16] 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:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{N} |x_i|^p\right)^{1/p}. \tag{2.18}$$

Differentiating the objective $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$ with respect to (w.r.t.) $\boldsymbol{\beta}$ and setting the derivative equal to 0 to find the minimum[17] yields the solution for $\boldsymbol{\beta}$:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = 0 \Rightarrow \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \Rightarrow \\ \hat{\boldsymbol{\beta}}_{\text{LS}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned} \tag{2.19}$$

However, this solution for $\hat{\boldsymbol{\beta}}_{\text{LS}}$ is only valid when $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. $\mathbf{X}$ has to be full rank [43]. 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:

$$\hat{\boldsymbol{\beta}}_{\text{L}_2} = \arg\min_{\boldsymbol{\beta}} \left\{ \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \right\}, \tag{2.20}$$

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 regression[18] the corresponding solution for $\boldsymbol{\beta}$ is:

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}, \tag{2.21}$$

where $\mathbf{I}$ is the identity matrix[19]. This extra term, $\Omega = \lambda\|\boldsymbol{\beta}\|_2^2$, acts as a shrinkage factor on the coefficients of $\boldsymbol{\beta}$. Looking at equation (2.20), we see that this is the Lagrangian form of the equivalent problem:

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{\text{L}_2} &= \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{f}(\mathbf{X})\|_2^2 \\ \text{subject to :} \quad &\sum_{i=1}^{N} \beta_i^2 = \|\boldsymbol{\beta}\|_2^2 \leq t, \end{aligned} \tag{2.22}$$

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 $\boldsymbol{\beta}$ to have some maximal norm. The effect of the regularization is controlled by λ, where $\hat{\boldsymbol{\beta}}_{\text{L}_2} \to \hat{\boldsymbol{\beta}}_{\text{LS}}$ for $\lambda \to 0$ and $\hat{\boldsymbol{\beta}}_{\text{L}_2} \to \mathbf{0}$ for $\lambda \to \infty$. An example of this can be seen in Figure 2.3.

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 equation (2.20) for different values of λ. Here we see the regularizing

[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. $\boldsymbol{\beta}$ it is seen that this really is a minimum and not a maximum (or saddle point).

[18] Here $\hat{\boldsymbol{\beta}}_{\text{L}_2}$ is the solution for any general function $\mathbf{f}$ and $\hat{\boldsymbol{\beta}}_{\text{ridge}}$ is the specific solution for a linear function $\mathbf{f}$, where linear is w.r.t. to the model parameters $\boldsymbol{\beta}$.

[19] The $\lambda\mathbf{I}$ in equation (2.21) 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.

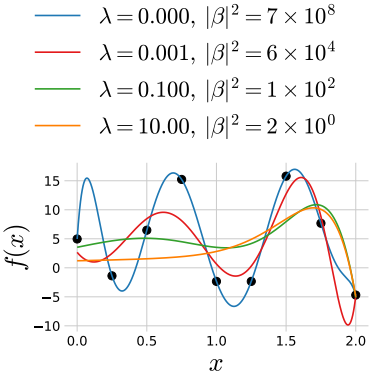


Figure 2.3: Effect of tuning the regularization strength λ in ridge regression.

effect of λ, going from $\lambda = 0$ in blue which fits all points[20] 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 subsection 2.3.1, 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.

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

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

$$\Omega = \lambda \|\boldsymbol{\beta}\|_1, \tag{2.23}$$

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 Tibshirani [72] in 1996. As with the L_2-penalty, the L_1-penalty also regularizes the coefficients of $\boldsymbol{\beta}$, however, this loss leads to sparse[21] solutions. An illustration of this can be seen in Figure 2.4 and Figure 2.5 where the constraint regions of $\boldsymbol{\beta}$ is 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 $p \leq 1$ [43].

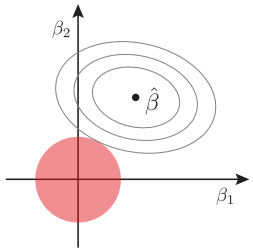


Figure 2.4: Sketch of the minimization problem defined in equation (2.22), i.e. for a L_2-penalty. The constrain region shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in $2D$-space and the contours of the unconstrained solution is shown with grey, dashed lines.

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

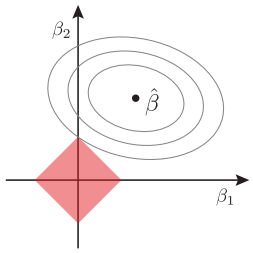


Figure 2.5: Sketch of the similar minimization problem defined in Figure 2.4 for the L_1-penalty. The constrain region shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in $2D$-space and the contours of the unconstrained solution is shown with grey, dashed lines.

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:

$$\mathcal{L}(h) = \hat{L}(\ell, h, S) + \Omega(h) \tag{2.24}$$

where $\hat{L}$ 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 via cross validation is discussed in subsection 2.4.2 and the choice of the training loss function ℓ in section 2.5.

2.4.2 *Cross Validation*

In general we want to be able to estimate the performance[22] 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

[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 – e.g. with different values of regularization strength λ – 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 [43]. For an illustration of this, see Figure 2.6. 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 gauged[23]. 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 machine learning 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, eg. all houses sold before 2018, is used for training and then the model is evaluated on the performance of samples after the event, e.g. houses sold in 2018. For an illustration of this, see Figure 2.7.

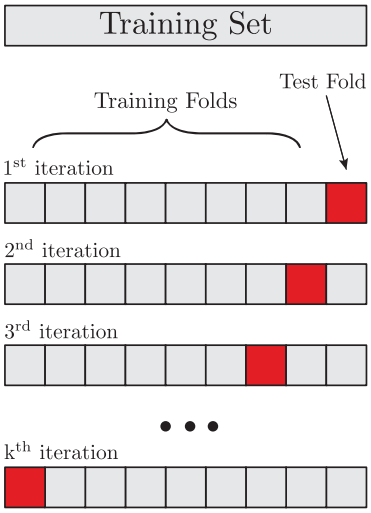


Figure 2.6: k-fold cross validation.

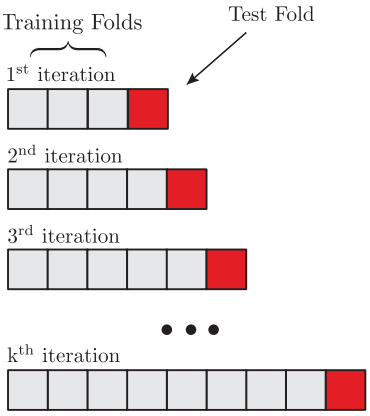


Figure 2.7: k-fold cross validation for time series data.

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

2.4.3 *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 and then by looking at the data "learns" a new and better set of values. The question then become: for how long should the model be allowed to continue training?

This is a another 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 2.2 was just an illustration of the bias variance tradeoff, it is also something that is seen in real data and can taken advantage of through *early stopping*. Early stopping is the process of monitoring the loss for the training set $\mathcal{D}_{\text{train}}$ and validation set $\mathcal{D}_{\text{val}}$. As mentioned in subsection 2.4.2, the model is only fitted on $\mathcal{D}_{\text{train}}$ but the performance on $\mathcal{D}_{\text{val}}$ is also measured. Whenever the validation loss starts to increase, the training of the model should be terminated.

To avoid stopping due to a single noise-induced outlier which terminated 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.

2.5 *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 a classification problem or a regression problem is dealth with. Let us for know 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 or bad? The first case was a factor of 2 off, whereas the prediction in the second case was only 33% off compared to the true value. The first case underestimated the price whereas the second overestimated; should this have any importance?

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, it 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 section 2.7. The most common choice of loss function is by far the Squared Error (SE):

$$\ell_{\text{SE}}(y, \hat{y}) = (y - \hat{y})^2 , \tag{2.25}$$

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:

$$\ell_{\text{AE}}(y, \hat{y}) = |y - \hat{y}|. \tag{2.26}$$

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 [18]. Six different loss functions (for regression problems) have been investigated in this project. In addition to SE and AE also the LogCosh, Cauchy [18], Welsch [18] and Fair [2] loss functions are used:

$$\begin{aligned}
\ell_{\text{LogCosh}}(y, \hat{y}) &= \log\left(\cosh\left(y - \hat{y}\right)\right) \\
\ell_{\text{Cauchy}}(y, \hat{y}) &= \log\left(\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2 + 1\right) \\
\ell_{\text{Welsch}}(y, \hat{y}) &= 1 - \exp\left(-\frac{1}{2}\left(\frac{y - \hat{y}}{c}\right)^2\right) \\
\ell_{\text{Fair}}(y, \hat{y}) &= c^2\left(\frac{|y - \hat{y}|}{c} - \log\left(\frac{|y - \hat{y}|}{c} + 1\right)\right).
\end{aligned} \tag{2.27}$$

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 2.8. 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 one when $y - \hat{y}$ goes towards infinity, whereas it goes to zero 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.

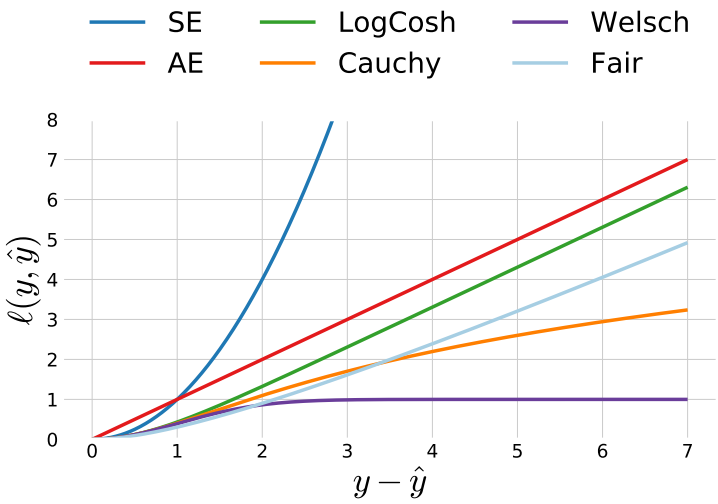


Figure 2.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$, see equation (2.27). In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair functions c is set to 1. For a zoom in of the inner region where $y - \hat{y} < 2$ see Figure 2.9. All six graphs are symmetric in $y - \hat{y}$ which is why they are only shown for positive values of $y - \hat{y}$.

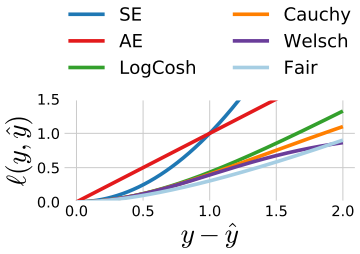


Figure 2.9: Zoom in of Figure 2.8.

2.5.1 *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 f_{eval} which is the overall metric. For the loss functions defined in section 2.5 the objective function is based on the mean of the individual losses:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \hat{y}_i) + \Omega. \tag{2.28}$$

2.6 *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 [43]. A simple example of this can be seen in Figure 2.10 and Figure 2.11. 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 top "box" is called the *root* of the tree, any

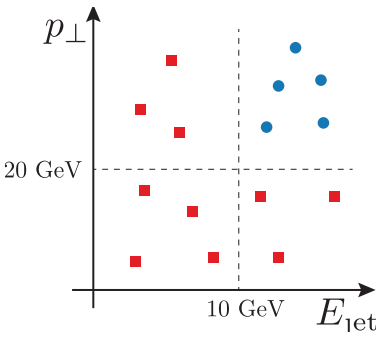


Figure 2.10: Illustration of the cuts a decision tree model make for signal in blue circles and background in red squares. This is an visualization in the feature space of the decision tree seen in Figure 2.11.

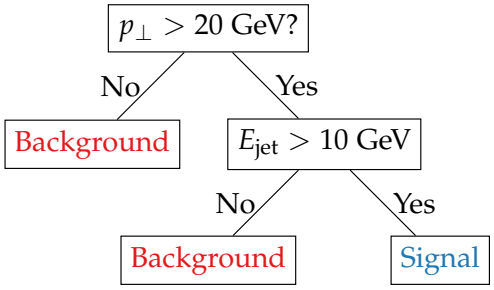


Figure 2.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables $p_\perp$ and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure 2.10.

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 momentum[24] $p_\perp$: 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 overfitting , 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 Hastie et al. [43].

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

2.6.1 *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 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 high variance.

Random Forests

Random Forests were first introduced in 2001 by Breiman [25]. Random Forests are a collection of B decision trees where each tree is trained on bootstrapped versions of the training data and the average[25] the individual trees' predictions T_b:

$$f_{\text{RF}}(\mathbf{x}) = \hat{y}_{\text{RF}} = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathbf{x}). \tag{2.29}$$

[25] In the 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* [43]. 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:*

$$\mathrm{Var}(\bar{\mu}_B) = \frac{1-\rho}{B}\sigma^2 + \rho\sigma^2, \tag{2.30}$$

where $\bar{\mu}_B$ is the average of the i.d. variables [43].

Equation (2.30) 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)[26]. This is called column-subsampling (in contrast to row subsampling).

[26] Throughout this project all data is assumed to be *tidy data* unless otherwise explicitly mentioned. Tidy data was a concept formalized in 2003 by Wickham [82] 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.

[27] Kaggle is an online platform where people compete with machine learning models.

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 [31] revolutionized the ML world by winning numerous Kaggle[27] competitions [3] including the Higgs Machine Learning competition in 2014 [4].

Boosting is the process of sequentially applying weak learner models to repeatedly modified versions of data [43]. In an iterative fashion this combines many weak learners into a single strong learner. The final prediction is thus a weighted sum over the K different weak learners F_k:

$$f_{\mathrm{BDT}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k F_k(\mathbf{x}) \tag{2.31}$$

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 $F_k(x) = \hat{y}$. In boosting we want the next iteration to be a better model than the previous one, so imagine that the perfect addition to the model

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

$$F_{k+1}(x) = F_k(x) + h(x) = y \Leftrightarrow h(x) = y - F_k(x). \tag{2.32}$$

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: $L(y, F_k(x)) = \frac{1}{2}(y - F_k(x))^2$. The gradient of the loss w.r.t. to $F_k(x)$ is:

$$\frac{\partial L(y, F_k(x))}{\partial F_k(x)} = F_k(x) - y. \tag{2.33}$$

This is exactly the negative of the r.h.s. of equation (2.32). 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:

$$h(x) = -\frac{\partial L}{\partial F_k}. \tag{2.34}$$

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

$$F_{k+1}(x) = F_k(x) + h(x) = F_k - \frac{\partial L}{\partial F_k}. \tag{2.35}$$

AdaBoost [38] 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 function[28] 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 [31] 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.

[28] Exponential loss for classification.

2.7 Hyperparamater 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 the internal optimization process. This could be the 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 BTDs.

In general we say that we have the ML model $\mathcal{A}$ with K hyperparameters. Each of these hyperparameters have a domain Λ_k. 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 categorical[29]. We define the hyperparameter configuration space as: $\mathbf{\Lambda} = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_K$. Within this space we are searching for a vector of hyperparameters $\boldsymbol{\lambda} \in \mathbf{\Lambda}$ which defines the optimal model $\mathcal{A}_{\boldsymbol{\lambda}^*}$. 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 $\boldsymbol{\lambda}^*$ is known as *hyperparameter optimization* (HPO.)

[29] Could e.g. be the choice of loss function.

The first naive approach would simply to manually[30] try out different combinations of $\boldsymbol{\lambda}$ and see performance on the validation set[31]. This is of course too cumbersome for advanced ML models, but it should be noted that it is a good place to start. In subsection 2.7.1 the HPO method called Grid Search is introduced which is further generalized and optimized in subsection 2.7.2 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 subsection 2.7.3 which allows for "smart" guesses.

[30] Also known as jokingly as "Grad Student Descent".

[31] Remember only to use the test set on the final model.

2.7.1 *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 $\mathbf{\Lambda}$. 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 $3 \times 4 = 12$ combinations of these two sets:

$$(x_i, y_i) \in \{(1,1),(1,2),\ldots,(x_i,y_i),\ldots,(3,4)\}, \tag{2.36}$$

as visualized in Figure 2.12. The advantage of GS is that it is an exhaustive search over all combinations[32] of hyperparameters, however, the total number of combinations grows exponentially and GS as a method thus suffers the curse of dimensionality[33].

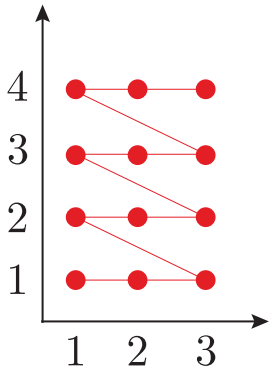


Figure 2.12: Visualization of grid search run on the two hyperparameters x and why y with the domains $x = \{1,2,3\}$ and $y = \{1,2,3,4\}$.

[32] Note that the user has to provide the values for each hyperparamater to be tried out manually.

[33] Not the dimensionality of the input feature space, but of the hyperparameter configuration space.

2.7.2 *Random Search*

To circumvent the problems of grid search, Bergstra and Bengio [22] 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 hyper-parameter optimization"* [22]. Instead of searching through all possible values of $\boldsymbol{\lambda}$ like in GS, RS makes B runs where each $\boldsymbol{\lambda}_i$ is given by:

$$\boldsymbol{\lambda}_i \sim \sum_{j=1}^{K} \mathrm{PDF}_j(\Lambda_j) \cdot \hat{\boldsymbol{e}}_j. \tag{2.37}$$

Equation (2.37) 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 $\boldsymbol{\lambda}$ be the vector of those N random numbers. In a 2D-space $\boldsymbol{\lambda}_i$ could thus be:

$$\boldsymbol{\lambda}_i \sim = \begin{bmatrix} \mathcal{N}(100, 4) \\ \mathcal{U}(0, 1) \end{bmatrix}, \tag{2.38}$$

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 tunable[34], but also due to the fact that often some hyperparameter dimensions are more important than other. Even though the hyperparameter configuration space $\boldsymbol{\Lambda}$ might be high-dimensional, it often exhibits *low effective dimensionality* [22]. 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 2.13.

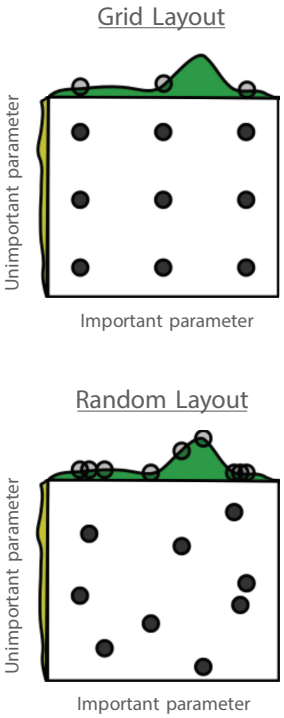


Figure 2.13: Visualization of the difference between grid search and random search. Adapted from Bergstra and Bengio [22].

[34] Compared to grid search which tries *all* possible combinations.

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 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 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 [22].

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

2.7.3 *Bayesian Optimization*

When performing hyperparameter optimization it often takes a lot of time to evaluate the individual hyperparameters. Remember, that each evaluation consists of fitting $\mathcal{A}_\lambda$ 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 expensive[35] to evaluate then "smart" guesses are worth spending a bit of time on developing. The hope

[35] With respect to time.

is that the 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 $\boldsymbol{\lambda}$ is independent of the value of the evaluation performance.

In Bayesian Optimization (BO) [26], the evaluation function as a function of hyperparameter is unknown. This unknown 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 is a manually chosen function which 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 $\boldsymbol{\lambda}$. 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 2.14. 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 unknown hyperparameter-dependent 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 $\boldsymbol{\lambda}$ 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 $\boldsymbol{\lambda}$ 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 $\boldsymbol{\lambda}$ according the acquisition function given the GPs.

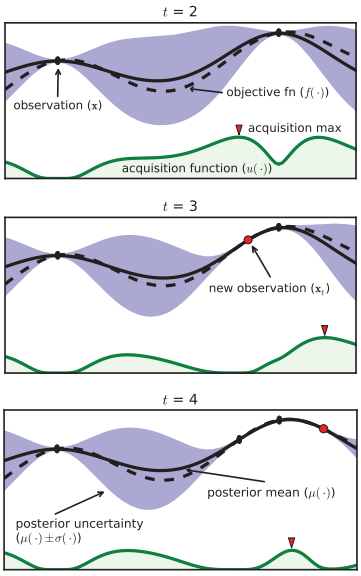


Figure 2.14: XXX **TODO!**. Adapted from Brochu et al. [26].

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 Brochu et al. [26]. 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 2.14 described above. The acquisition function used in this project is the Upper Confidence Bound (UCB):

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \tag{2.39}$$

where $\kappa \geq 0$ is the parameter[36] controlling the exploration-exploitation tradeoff.

Bayesian Optimization has the great benefit of slowly learning

[36] Here κ can thus be regarded as a hyper-hyperparameter since it controlls how the other hyperparamaters are optimized.

the hyperparameter space and making smarter and more educated guesses over time. However, it also comes with the cost of being harder to numerically implement compared to GS and RS[37], and parallelization is non-trivial to implement since it by definition is a sequential process. The performance boost is also not guaranteed.

[37] Which are basically plug-and-play with Scikit-Learn [59].

2.8 Feature Importance

Having first established in section 2.2 that machine learning algorithms are indeed able to not only learn from data but also to generalize well without overfitting (section 2.4), modern ML algorithms such as decision trees, random forests and boosted decision trees were introduced in section 2.6 and they were hyperparameter optimized in section 2.7, 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 Lundberg and Lee [51] showed that six different previously used methods were all specific instances of a universal underlying method[38] and proposed SHaplay Additive exPlanation (SHAP) values as a unified measure of feature importance. In 2018 they developed a fast algorithm for computing SHAP values for tree ensemples and showed that previous measures of feature importance heavily used for trees, e.g. *gain*, were *inconsistent* [53]. 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*-based feature importances are both consistent feature importance measure, however, only SHAP allows for individualized[39], or local, feature importances.

[38] The class of *additive feature attribution methods* [51].

[39] 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.

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:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i. \tag{2.40}$$

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 [68]. 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:

$$f(x) = g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i. \tag{2.41}$$

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:*

$$z'_i = 0 \Rightarrow \phi_i = 0. \tag{2.42}$$

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, Lundberg and Lee [51] show that the only solution to equation (2.40) is:

$$\phi_i = \sum_{S \subseteq \widetilde{M} \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} \left[f_x(S \cup \{i\}) - f_x(S) \right], \tag{2.43}$$

which one can simplify to:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq \widetilde{M} \setminus \{i\}} w(S) \cdot \Delta_{f_x}(S) \\ w(S) &\equiv \frac{|S|!(M - |S| - 1)!}{M!} \\ \Delta_{f_x}(S) &\equiv \left[f_x(S \cup \{i\}) - f_x(S) \right]. \end{aligned} \tag{2.44}$$

In equation (2.43) $\widetilde{M}$ is the *set* of all input features[40], $S \subseteq \widetilde{M} \setminus \{i\}$ means a subset S of $\widetilde{M}$ 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 (2.44) 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.

[40] Compared to $M = |\widetilde{M}|$ which is the *number* of all input features.

To get a better understanding of the different sets in the summation, one could look at the decision tree shown in Figure 2.11. Here $\widetilde{M}$ would be $\widetilde{M} = \{p_\perp, E_{\text{jet}}\}$. For the feature $i = p_\perp$ one would thus have:

$$\begin{aligned} \phi_{p_\perp} &= \sum_{S \subseteq \widetilde{M} \setminus \{p_\perp\}} w(S) \cdot \Delta_{f_x}(S) \\ &= \sum_{S \in [\{\}, \{E_{\text{jet}}\}]} w(S) \cdot \Delta_{f_x}(S) \\ &= \frac{0!(2 - 0 - 1)!}{2!} \cdot \left[f_x(\{p_\perp\}) - f_x(\{\}) \right] \\ &\quad + \frac{1!(2 - 1 - 1)!}{2!} \cdot \left[f_x(\{E_{\text{jet}}, p_\perp\}) - f_x(\{E_{\text{jet}}\}) \right]. \end{aligned} \tag{2.45}$$

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 Lundberg et al. [53] did was to develop an efficient algorithm that could solve this for trees in polynomial time[41].

[41] 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).

SHAP values allows one to explain for a single prediction why it got the prediction that it got. When applied to the entire data set $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations each with M features, one gets the matrix $\mathbf{\Phi}$. When summing over the absolute value of each column, one gets the global impact ϕ_i^{tot} of the i^{th} feature [53]:

$$\phi_i^{\text{tot}} = \sum_{j=1}^{N} |\mathbf{\Phi}_{i,j}|. \tag{2.46}$$

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 correlated[42] to an already existing feature, the feature importance of the previous feature will decrease. This is due to the axiom of symmetry:

[42] Not necessarily linearly correlated.

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

$$f_x(S \cup \{i\}) = f_x(S \cup \{j\}) \tag{2.47}$$

then $\phi_i = \phi_j$.

It can be shown that axiom 4 is implied by axiom 3 [51, Supp. Material]. Two identical features, as seen by the model, will thus "share" the feature importance.

3. *Danish Housing Prices*

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

— Mark Twain

Housing markets have always been a playing field for economists, property speculators, real estate agents, 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 as seen in papers from the Danish National Bank developing a regional model of the Danish housing market [46] to an analysis of the financial crisis in 2008-2009 and its effects on the Copenhagen metropolitan area [57].

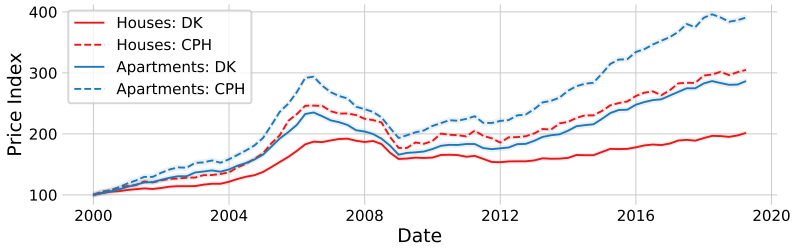


Figure 3.1: Price Index of the Danish housing market. Prince index of Danish one-family houses and owner-occupied apartments where houses are shown in red and apartments in blue, where full lines are for the entirety of Denmark and dashed lines are only for Copenhagen. Errorbars (scaled up with a factor of 2) are shown as colored bands. The price index and its uncertainty is based on numbers from DST [34], however, rescaled to 100 in 2000 (instead of 2006 as it was in the data).

If one takes a look at the time development of the Danish housing market, the Danish governmental organization for statistics, Statistics Denmark, releases a price index [34] for both one-family houses (OFH) and owner-occupied apartments (OOA) quarterly, see Figure 3.1. Here it is easy to see the effect of the financial crisis around 2008, but also the steady increase in the housing market in both Copenhagen and the entire country since then. Housing in this context means both actual houses and privately owned apartments, and will be called residences in general in this project.

The goal of this subproject 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 computer to automatically find these patterns and see if we can improve this model[1].

In section 3.1 the data will be introduced and feature augmented in section 3.2. The evaluation functions will be discussed in section 3.3 and the choice of loss function decided in section 3.4. The

[1] In contrary to Hviid [46], Mulalic et al. [57] and others who base their models on macro-economic principles.

model is fitted and optimized in section 3.5 and the results presented in section 3.6. Finally the model will be further understood in section 3.7, some additional models presented in section 3.8 and at last the models will be discussed in section 3.9.

3.1 *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 which 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 columns which only contain internal information for Boligsiden[2]. To get a better understanding of all of the variables, histograms showing the one-dimensional distributions of all of the variables were made.

[2] The variables `Sag_Kvhx`, `Enhed_GOP_BoligtypeKode`, and `Bygning_GOP_Matrikelnr`.

Four particularly interesting ones are seen in Figure 3.2: the distribution of the date of the sale `SalgsDato` in subfigure (a), the distribution of the type of residence `SagtypeNr` in subfigure (b), the distribution of the longitude of the residence `GisX_WGS84` in subfigure (c), and the distribution of the area of the residence `ArealBolig` in subfigure (d).

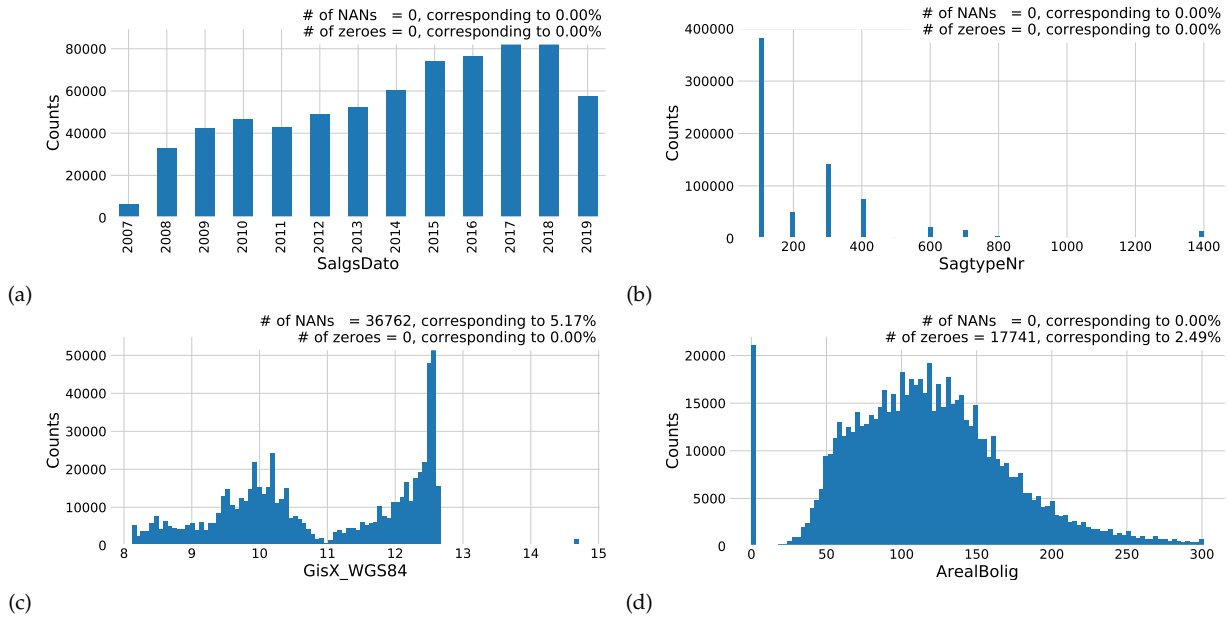


Figure 3.2: Distributions of four out of the 168 input variables. Subplot (a) shows the date of the sale, Subplot (b) shows the type of residence, Subplot (c) shows the longitude, Subplot (d) shows the area og the house.

The distribution of the date of sale, Figure 3.2 (a), is an interest-

ing 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 `SagTypeNr` 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 3.1.

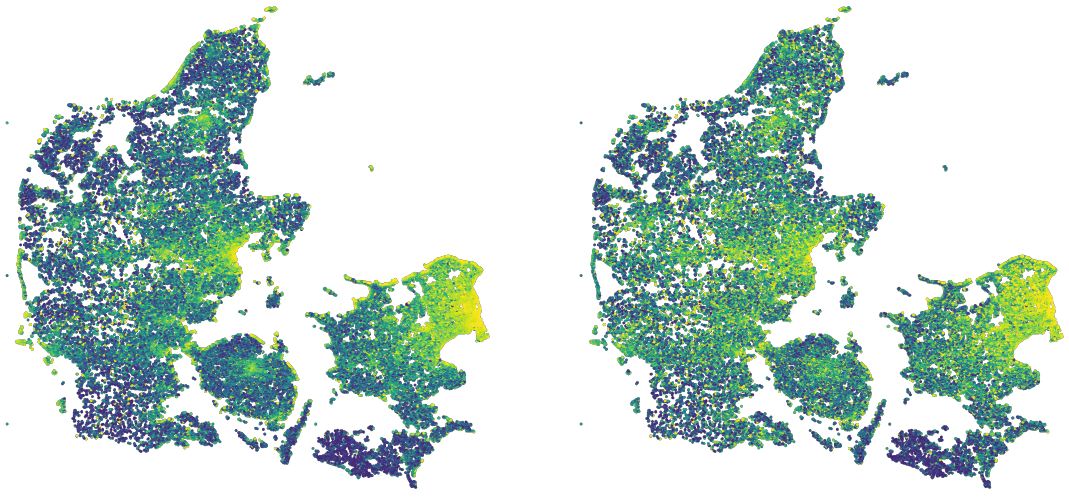
In this project only one-family houses – "Villas" in Danish – with code 100 and owner-occupied apartments – "Ejerlejlighed" in Danish – with code 300 are considered. As can be seen from Figure 3.2 (b) these two types of residences are also the most frequent sales with close to 400 000 and 150 000 sales in total. The longitude distribution, Figure 3.2 (c), 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, so-called "Not A Number"s (NANs). The distribution of the area, Figure 3.2 (d), shows that most residences are between 50 m^2 and 200 m^2, as expected in Denmark. However, a relatively large part of the residences, 2.5 %, are listed as having an area of 0 m^2 which are obviously erroneous entries. All of the 1D-distributions can be seen in Figure A.2–A.15.

The geographic distribution of sales can be seen in Figure 3.3. The residences are coloured according the square meter price in Figure 3.3 (a) and according to the sales price in Figure 3.3 (b). 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 points[3], overplotting quickly becomes an issue. To circumvent this the software package called DataShader [8] was used which in a simple, consistent, and not at least computationally efficient manner allows one to plot big data.

Type	Name
100	Villa
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 3.1: Mapping between the code in `SagTypeNr` and the type of residence. The two important types of residences are villa (one-family houses) and ejerlejlighed (owner-occupied apartments).

[3] Only sales with a valid GPS-coordinate and area of residence are shown



(a) (b)

Figure 3.3: Geographic distribution of the sold residences. In subplot (a) the sales are colored according to their square meter price and in subplot (b) according to the sales price.

The most important of the features is the sales price, called

`SalgsPris` in the dataset. Its distribution is shown in Figure 3.4. This is a positively skewed distribution that shares visual similarities with a log-normal distribution. The mode[4] of the all sales prices is 1.1 M.kr. and the median is 1.6 M.kr. The mean is 2.0 M.kr. but this value is heavily influenced by a few very high values.

[4] Measured in millions DKK, M.kr.

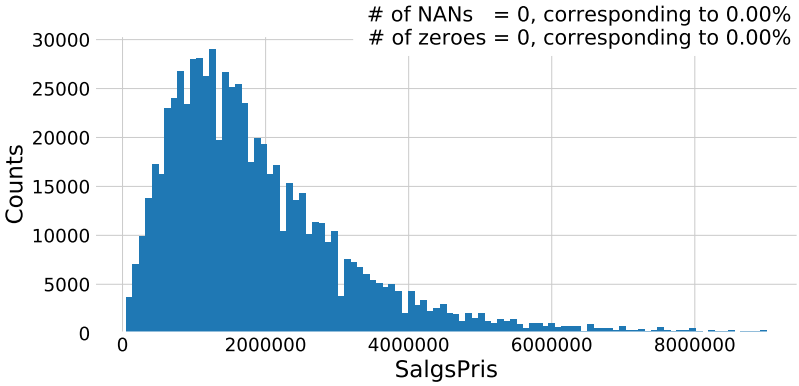


Figure 3.4: Histogram of prices of houses and apartments sold in Denmark.

3.1.1 *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 A.16 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 lattitude-longitude and ETRS89[5] 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.

[5] European Terrestrial Reference System 1989.

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 variable; the sales price. This is shown in Figure 3.5 for the variables where $|\rho| > 10\,\%$. It is the previous property evaluation, `EjdVurdering_EjendomsVaerdi` that is correlated the most with the sales price, which does not come as any huge surprise. Other positively correlated variables are the cost of ownership[6], 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.

[6] This a great example of the fact that correlation does not imply causation

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

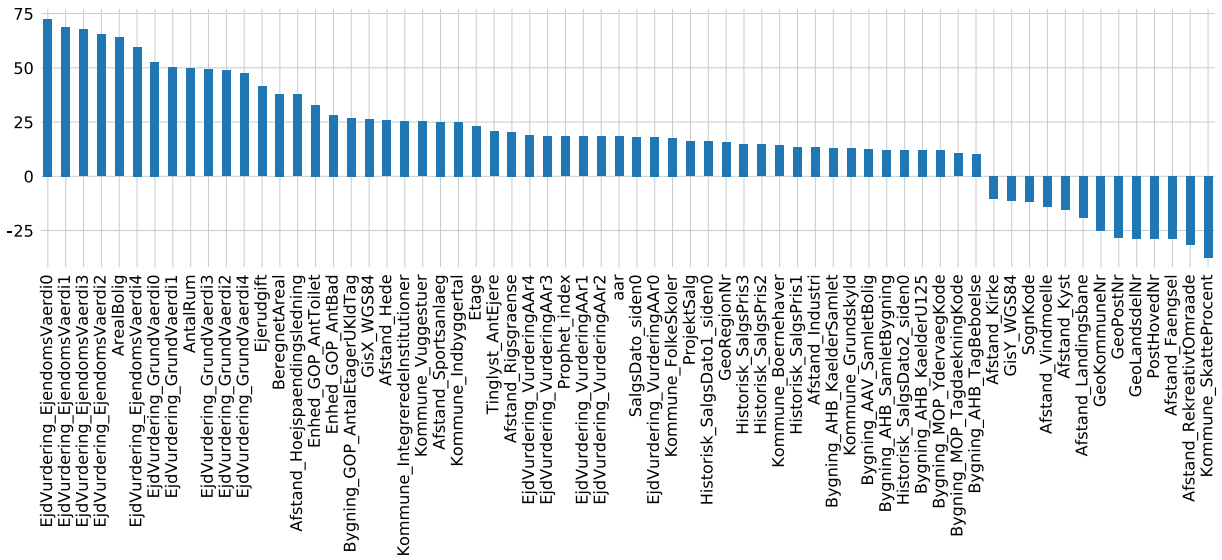


Figure 3.5: Linear correlation between variables and price for variables where the correlation coefficient ρ is $|\rho| > 10\,\%$.

relation 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 [64]. This is in comparison to two uncorrelated variables which would simply display noisy behavior and only have few grids with many points in. Albanese et al. [12] extended on this idea and developed the computationally efficient algorithm called MICtools which computes the estimator for MIC: MIC_e. An example of this non-linear correlation is seen in Figure 3.6. 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 A.17 in the appendix.

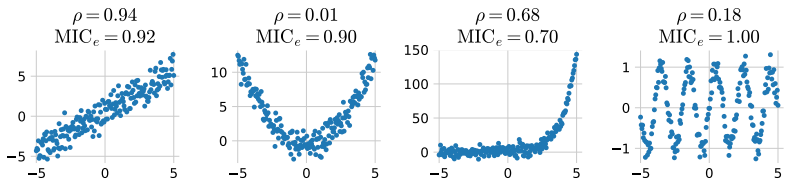


Figure 3.6: Comparison of the linear correlation ρ and the non-linear MIC for a straight line, a parabola, an exponential, and a sine wave, all with noise added. See also Figure A.17 in the appendix.

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 3.7. Again, the previous property evaluations are the most correlated features to the sales price followed by the parish code, `SogneKode`. In general the geographical variables score high here with also the post code, municipality number, and longitude.

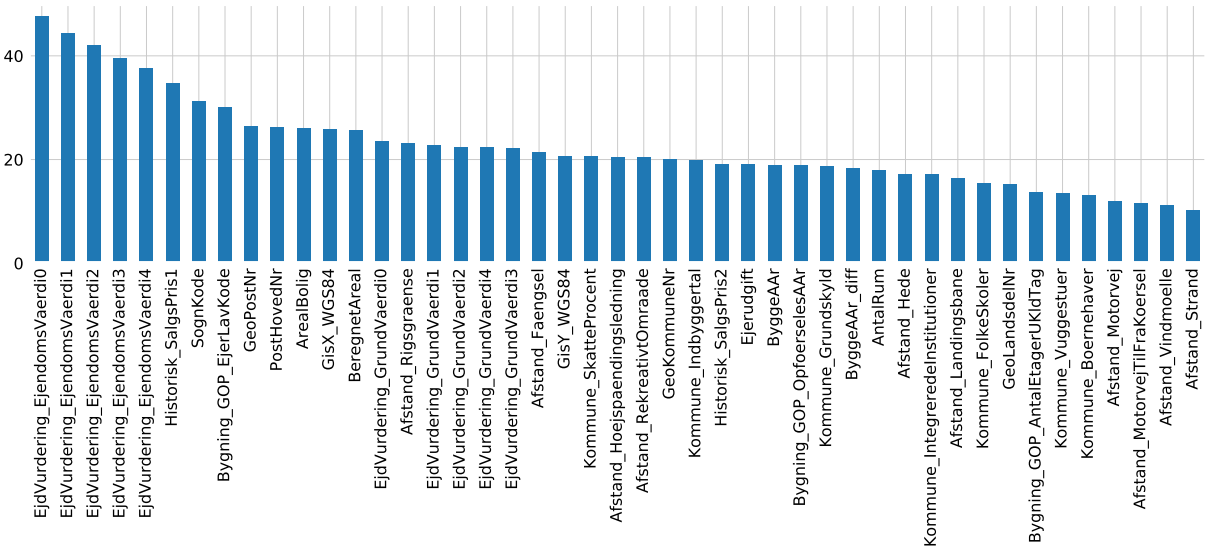


Figure 3.7: Non-linear correlation between variables and price using Maximal Information Coefficient (MIC) for variables where MIC $> 10\%$.

3.1.2 *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 3.8. 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-variables[7]. 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 %.

[7] Distance to: prison, heath, high-voltage transmission line, industri, visible railroad, church, churchyard, coast, landing strip, motorway, access to motorway, recreational area, border, sports centre, beach, and windmill.

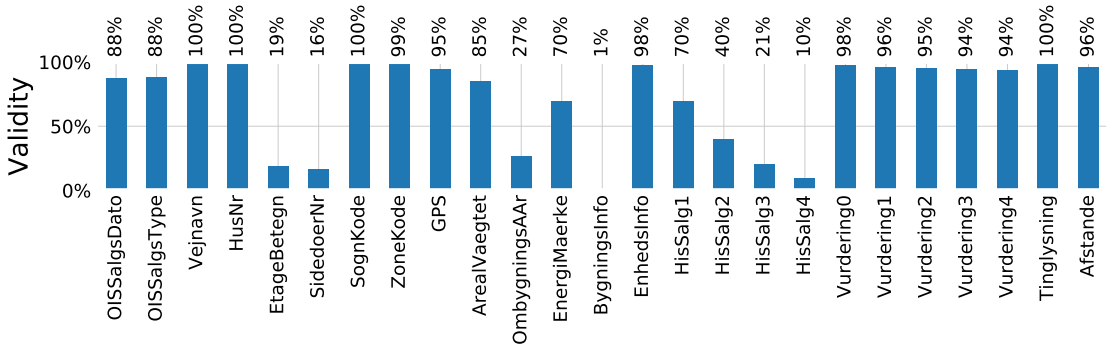


Figure 3.8: Percentage of valid counts for each variable grouped together in categories.

To see how closely related the different validity groups are, one can look at the dendrogram in Figure 3.9. The dendrogram is based on a hierarchical clustering algorithm [79] where the different groups are clustered according to the linear correlation of their validity. 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.

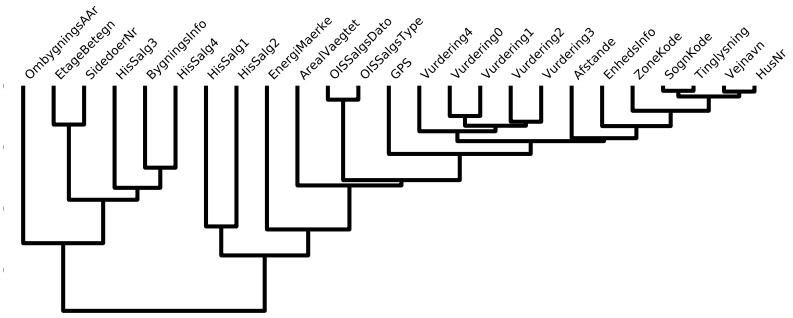


Figure 3.9: Validity Dendrogram based on hierarchical clustering of the linear correlation of validity for the housing price variables clustered together.

To see a heatmap of the inter-variable correlations, see Figure A.1 in the appendix.

3.1.3 *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 3.2. Sales type, `OISSalgsType`, is a OIS[8] code which describes what type of sale it is: when it is 1 it is considered a normal sale, compared to 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.

[8] OIS is short for "Den Offentlige Informationsserver", the Danish public information server, and it collects information about Danish residences [9].

	Description	Remaining	Removed
Area	$20\,\mathrm{m}^2 \leq$ Area $\leq 500\,\mathrm{m}^2$	689 140	23 666
Price	0.1 M.kr. $\leq$ Price $\leq$ 100 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

Table 3.2: Overview of the basic cuts which define the minimum information needed to preditct the price of a sale.

3.2 *Feature Augmentation*

Until now 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 existing features. The day of the month, the month, and the year are extracted from the sales date and the sales date is also represented as the numbers of days since January 1st, 2009. From the number of the house, `HusNr`, the number is extracted along with a boolean

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

Table 3.3: Side door mapping. If the side door string contains e.g. "TH" this gets the code 11.

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

Table 3.4: Street mapping. If the street name contains e.g. "Vej" this gets the code 0.

flag indicating whether or not it includes a letter (eg. "27B"). The number of the side door, `SidedoerNr`, is formatted according to Table 3.3 and the road name is according to Table 3.4. The age of the house is added[9] and the amount of time (in years) since last major modification. The energy rating label, `EnergiMaerke`, is also converted from strings to values according to Table A.2 in the appendix.

[9] In addition to only having the year the house was was built.

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. They are dropped since we do not want the model to learn the price of the residence using these variables.

3.2.1 *Time-Dependent Price Index*

In addition to the manual data augmentation in section 3.2, a time-dependent price index is also added. We make use of the open source package called Prophet made by Taylor and Letham [70] at Facebook. It is based on a decomposable time series model [41] with two[10] components; trend $g(t)$ and seasonality $s(t)$:

$$y_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t, \tag{3.1}$$

[10] In their paper, Taylor and Letham [70] include a holiday component in their analysis as well which is not included in this project.

where ϵ_t is a normally distributed error term. Taylor and Letham [70] fit this equation with a generalized additive model (GAM) [42] which they argue has several practical advantages compared to ARIMA [11] models which commonly used in economics [56].

[11] AutoRegressive Integrated Moving Average.

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 (owner-occupied) apartments are seen in Figure 3.10 and Figure 3.11. In Figure 3.10 the weakly median price pr. square meter for apartments 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 3.11 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. The Prophet model plots for one-family houses can be seen in Figure A.18 and A.19 in the appendix.

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:

$$\text{PI}(t) = \frac{y_{\text{Prophet}}(t)}{\langle y_{\text{Prophet}}(t) \rangle}, \tag{3.2}$$

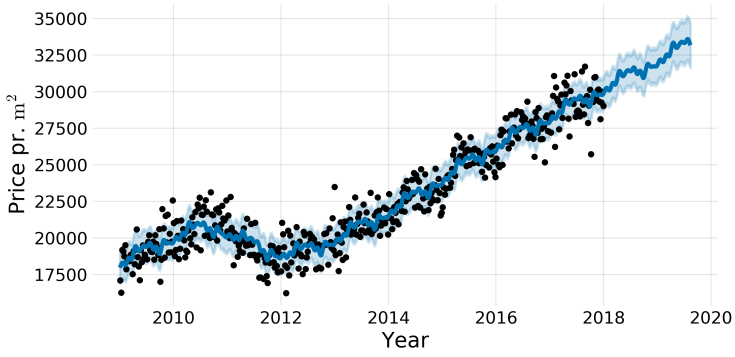


Figure 3.10: The predictions of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. The data is down-sampled to weekly bins where the median of each week is used as in input to the Prophet model. This can be seen as black dots in the figure. The model's forecasts for 2018 and 2019 are shown in blue with a light blue error band showing the 1σ confidence interval.

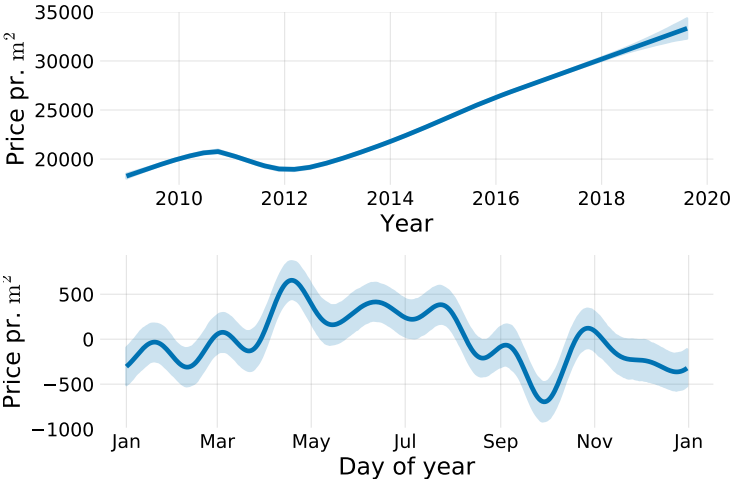


Figure 3.11: The trends of the Facebook Prophet model trained on square meter prices for owner-occupied apartments sold before January 1st, 2018. In the top plot is the overall trend as a function of year and in the bottom plot is the yearly variation as a function of day of year. It can be seen that 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 ($< 10\%$).

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.

3.3 *Evaluation Function*

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

$$z = \frac{\hat{y} - y}{y}, \tag{3.3}$$

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 $\mathbf{z}$ is expected to approximately follow a normal distribution[12]. Initially the mean of $\mathbf{z}$ was considered as the choice of evaluation function $f_{\text{eval}} \stackrel{?}{=} \text{mean}(\mathbf{z})$ 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 $f_{\text{eval}} \stackrel{?}{=} \text{std}(\mathbf{z})$. 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 fraction[13] of bad observations that can cause

[12] Where $\mathbf{z}$ is the vector of all relative predictions $\mathbf{z} \in \mathbb{R}^N$.

[13] Where the "asymptotic" in "asymptotic breakdown point" refers to when the number of samples goes to infinity.

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

$$\mathrm{MAD}(\mathbf{x}) = c \cdot \mathrm{median}\left(|\mathbf{x} - \mathrm{median}(\mathbf{x})|\right), \quad c = \frac{1}{\Phi^{-1}(\frac{3}{4})}, \tag{3.4}$$

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 function[14] [50]. 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 $\mathbf{z}$ as narrow as possible, we also want it centered around 0. We thus continue with the following evaluation function:

[14] Inverse of the cumulative distribution function.

$$\begin{aligned} \mathrm{MAD}_0(\mathbf{x}) &\equiv c \cdot \mathrm{median}\left(|\mathbf{x} - 0|\right) = c \cdot \mathrm{median}\left(|\mathbf{x}|\right) \\ f_{\mathrm{eval}}(\mathbf{z}) &\equiv \mathrm{MAD}_0(\mathbf{z}) = c \cdot \mathrm{median}\left(|\mathbf{z}|\right). \end{aligned} \tag{3.5}$$

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: $f_{\mathrm{eval}}(\mathbf{z}_{\mathrm{OFH}}) = 11.35\,\%$ for houses and $f_{\mathrm{eval}}(\mathbf{z}_{\mathrm{OOA}}) = 5.72\,\%$ for apartments. In some cases f_{eval} will still be referred to as MAD and it will be mentioned explicably if the form in equation (3.4) is meant.

The MAD is assuming symmetric distributions and thus non-symmetric robust measures of the variability of a sample have been developed. See Rousseeuw and Croux [65] for more details.

3.4 *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 A.1 in the appendix. The data are split intro 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 3.5. 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 all[15] numeric variables with more than 3 unique values. The number of observations for the different tight sets can be seen in Table 3.6.

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

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

Table 3.5: train test split XXX **TODO!**.

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

Table 3.6: train test split tight XXX **TODO!**.

[15] Except the variables that contains the words: "aar" (year), "dato" (date), and "prophet".

assigning them a lower weight depending on time. It was investigated whether or not the model would perform better if samples got the time-dependent weight $w(t)$ given by:

$$w'(t) = e^{k \cdot t}, \quad k = \frac{\log 2}{T_{\frac{1}{2}}}$$
$$w(t) = \frac{w'(t)}{\langle w'(t) \rangle} \tag{3.6}$$

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

This is illustrated in Figure 3.12 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 section 2.5. A grid search[16] was performed for:

$$\begin{aligned} T_{\frac{1}{2}} &\in \{2.5, 5, 10, 20, \infty\} \text{ years} \\ \log_{10} &\in \{\text{True, False}\} \\ \ell &\in \{\ell_{\text{Cauchy}}, \ell_{\text{Fair}}, \ell_{\text{LogCosh}}, \ell_{\text{SE}}, \ell_{\text{Welsch}}\} \end{aligned} \tag{3.7}$$

The GS is run on the training set with 5-fold CV, early stopping is applied with a patience of 100, and XGBoost [31] (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 3.7. 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 3.8. It is interesting to note that both HPO for houses and apartments choose the same half-life and not to do any transformation[17]. All of the results for the apartments can be seen in Table A.3–A.7 in the appendix along with all of results for the houses in Table A.8–A.12.

A visualization of the HPO results can be seen as the parallel coordinate plot in Figure 3.13. 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 hyperparamater `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 A.20 in the appendix.

What can be concluded from Figure 3.13 is that there is a clear preference from not log-transforming the data, that the BDTs with

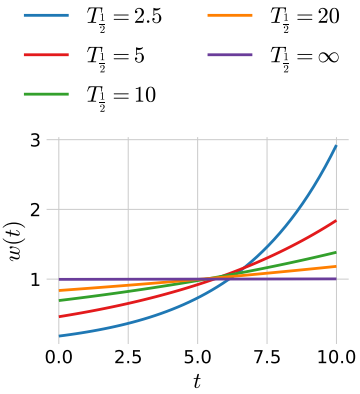


Figure 3.12: Time here is years after January 1st, 2009. XXX **TODO!**.

[16] 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 3.7: 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 3.8: Cauchy-villa.

[17] 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..

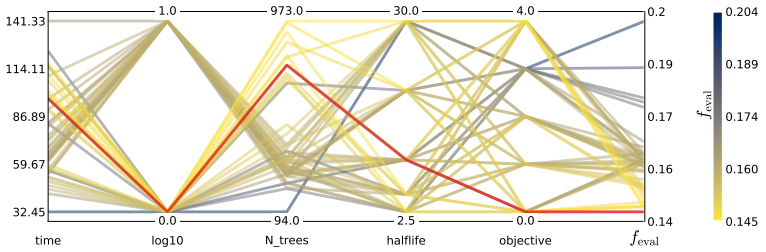


Figure 3.13: Hyperparameter optimization results of the housing model for apartments. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance 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.

many trees[18] generally performed better than the ones with fewer trees, that it is difficult to see a clear pattern for the half-life, and 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 uncertainties[19] of the different iterations also matter.

For the half-life and the choice of objective function these can be seen in Figure 3.14 and 3.15. 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 archives 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 hyperparamaters:

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

[18] Remember that the number of trees where selected by early stopping.

[19] The uncertainties here are the standard deviation (not of the mean) of the 5 folds in the cross validation.

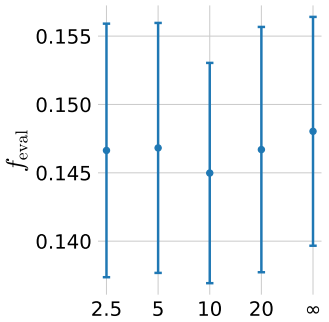


Figure 3.14: XXX Halflife $T_{\frac{1}{2}}$.

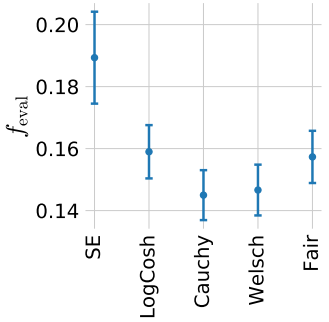


Figure 3.15: Objective function XXX.

3.5 *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 iteratations with 5-fold cross validation and early stopping[20]. The hyperparameters to be optimzed 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.

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

The `min_child_weight` variable controls when the decision tree algorithm should stop splitting a node into further nodes (and 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 3.9. 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.

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

Table 3.9: XXX

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 prepocessing 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 packagages provide the same flexibility as a custom implementation that works fully automatically.

The results of the RS and BO can be seen in Figure 3.16 and A.22 (in the appendix). The correpsonding plots for houses can be seen in Figure A.23 and A.24 in the appendix.

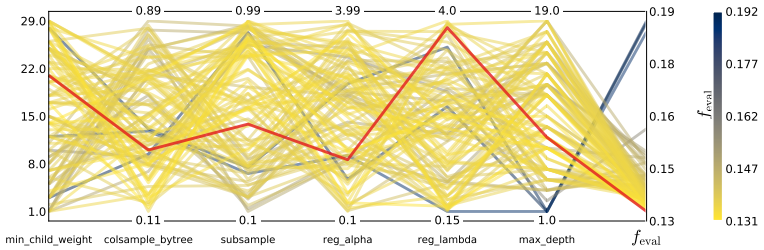


Figure 3.16: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

As with the initial HPO, it is important to compare the results in Figure 3.16 with their uncertainties. This can be seen in Figure 3.17. 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 minumum value, notice how flat of a minimum this is: most of the other iterations are within 1σ. The plot for BO in Figure A.26 shows the similar characteristic and does not show any improvement over time as was otherwise expected for BO compared to RS.

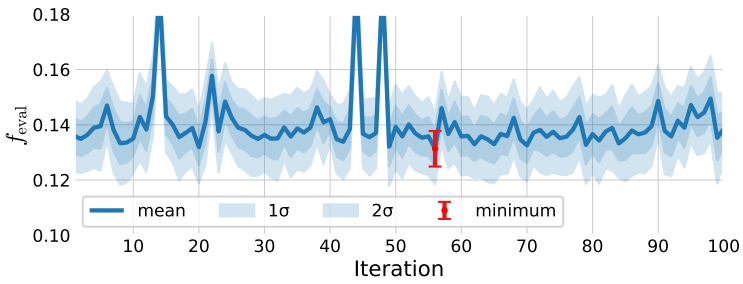


Figure 3.17: The results of running random search (RS) as hyperparameter optimization (HPO) on apartments using the XGB-model. The minimum (mean) loss along with its uncertainty is shown in red, the means for the different iterations of RS in blue, and as light blue bands are the one (and two) standard deviation(s) of the means (SDOM), all as a a function of iteration number.

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 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 3.18. This curve is the realisation of Figure 2.2 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 predition time – we keep the model with the lowest number of estimators that are still within 1σ of the minimimum: 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.

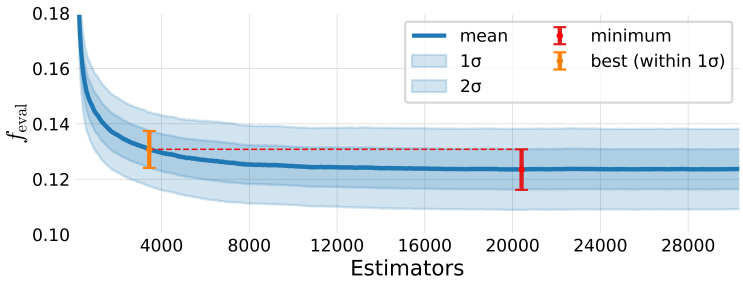


Figure 3.18: The results of early stopping on apartments using the XGB-model. The minimum (mean) loss along with its uncertainty is shown in red, the means for the different iterations of RS in blue, and as light blue bands are the one (and two) standard deviation(s), all as a a function of number of 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.

3.6 *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 3.19. Here the distribution of the relative price prediction $\mathbf{z}$ 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 function[21] 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 ±5 % wrong, 69.8 % within ±10 %, and 91.9 % within ±20 %. Note that the difference between the three models are minor and that they distributions

[21] Written as `MAD`

almost cannot be distinguished between each other.

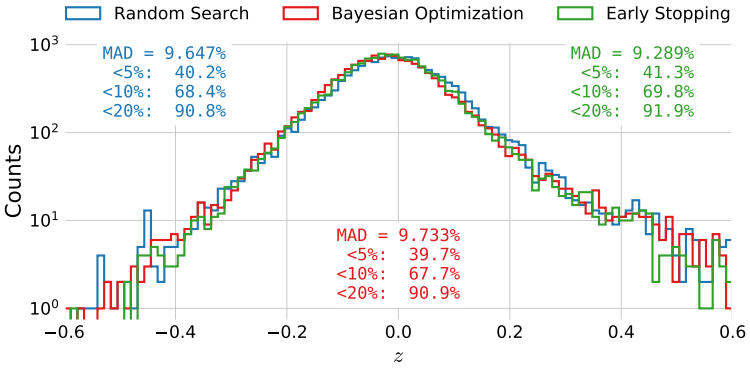


Figure 3.19: Histogram of z-values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using Random Search (RS) is shown in blue, for Bayesian Optimization (BO) in red. After finding the best model, BO in this case, the model is retrained using early stopping, the performance of which is shown in green.

An interesting observation from the performance metrics of the test set, is the low value of $f_{\text{eval}} = 0.9289$ (`MAD`). By looking at Figure 3.18 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 3.10. 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 A.27 in the appendix, where it can be seen that the final model has $f_{\text{eval}} = 0.8383$.

	Train	Test	2019
Normal	5.80 %	4.97 %	6.19 %
Tight	5.69 %	4.94 %	6.19 %

Table 3.10: XXX

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 $\mathbf{z}$ to be calculated and the MAD and the standard deviation (SD) of $\mathbf{z}$ are shown in Figure 3.20.

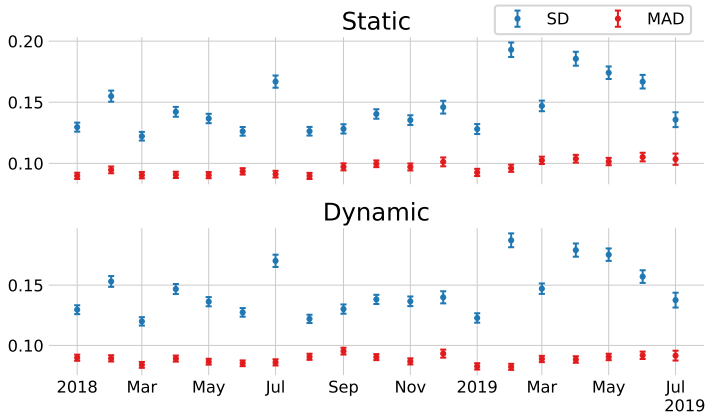


Figure 3.20: Performance of 1-month forecasts for apartments sold in 2018 and 2010. For both plots the XGB model is trained on data up to (but excluding) 2018. Top) The performance of the static model's prediction for both the standard deviation (SD) and MAD of the z-scores. Bottom) Same as above, however, 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:

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}, \quad \sigma_\sigma = \frac{\sigma}{\sqrt{2N}}, \tag{3.8}$$

where σ_μ is the standard deviation of the mean and σ_σ is the standard deviation of the standard deviation[22] [17]. 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.

[22] 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:

$$\begin{aligned} z_{\text{mi}} &\equiv \frac{\hat{y}}{y} = 1 - \frac{\hat{y} - y}{y} = 1 - z \\ \text{MI}_{\text{mean}} &= \text{mean}(\mathbf{z}_{\text{mi}}) \\ \text{MI}_{\text{median}} &= \text{median}(\mathbf{z}_{\text{mi}}). \end{aligned} \tag{3.9}$$

Here $\mathbf{z}_{\text{mi}}$ 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 $\text{MI}_{\text{median}}$. The market indices for every month of the forecast described in the previous figure can be seen in Figure 3.21. In the top panel the market index for the static model is shown where it is visible for $\text{MI}_{\text{median}}$ how it cosistently overshoots. Compare this to the dynamic $\text{MI}_{\text{median}}$ whic fluctuates around 0. BLABLA mere analyse her XXX. That the MI_{mean} is consistently lower than $\text{MI}_{\text{median}}$ 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 3.11 for owner-occupied apartments and in in Table 3.12 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 predicitions are within ± 5 % and more than 90 % are within ± 20 % which is similar to the performance of the professional automated property evaluations from e.g. Bolighed [24]. Also shown in the tables are the mean of the relative predictions $\mu_{\mathbf{z}}$ which shows that the The per-

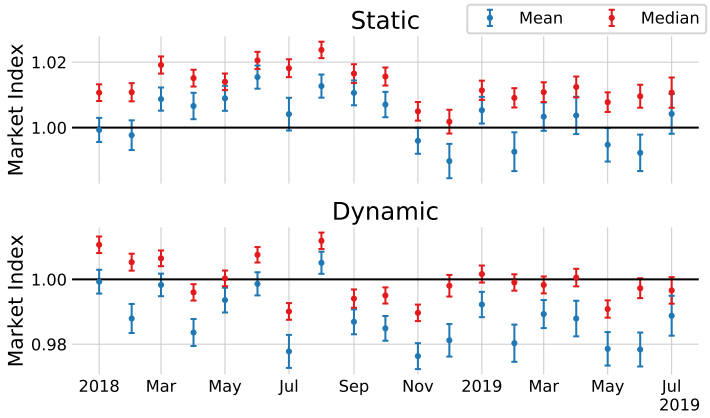


Figure 3.21: XXX **TODO!**.

formance on the tight cuts can be seen in Table A.13 and A.14 in the appendix.

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	$\mu_{\mathbf{z}}$
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 3.11: XXX ejer

	MAD (%)	$\leq 5\%(\%)$	$\leq 10\%(\%)$	$\leq 20\%(\%)$	$\mu_{\mathbf{z}}$
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 3.12: XXX villa

3.7 *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 section 2.8, 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 3.22. 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 (2.40). 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 use can use this tool to better understand why the prediction was estimated as it was.

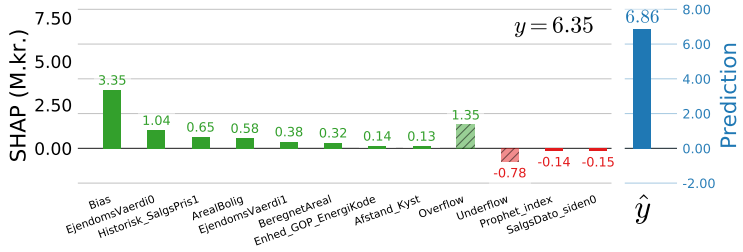


Figure 3.22: Model explanation for XGB model for a specific apartment. The bars are the variables in the dataset that the model found most important sorted after their importance for this particular apartment. The bias bar refers to the expected value of the model, which is simply the mean of the training 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 is the model prediction shown. The model prediction is the sum of all of the bars in the left par (6.86 M.kr. in this example). The negative values are shown in red, positive ones in green, and the prediction value in blue.

To get an overview of which variables are most important on a global[23] scale, ϕ_i^{tot}, see Figure 3.23. 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 $\phi^{\text{tot}}_{\text{ArealBolig}} = 5.35\,\%$. 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 the total days on market (DOM) (`LiggetidSamlet`) is large it pushes the prediction in the negative direction.

The SHAP-package[52] not only allow for 1D dependences to be gauged, it also allows for so-called *interaction plots*. These plots shows the 2D-dependece between the variable and the SHAP value colored according to a second variable. Since the previous public property valuation (PPPV) (`EjendomdsVaerdi0`) is the most important of the features, the interaction plot of this variable is seen in Figure 3.24. Here the SHAP value of the `EjendomdsVaerdi0` is plotted as a function of `EjendomdsVaerdi0` . 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 1$^{\text{st}}$ 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 then 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.

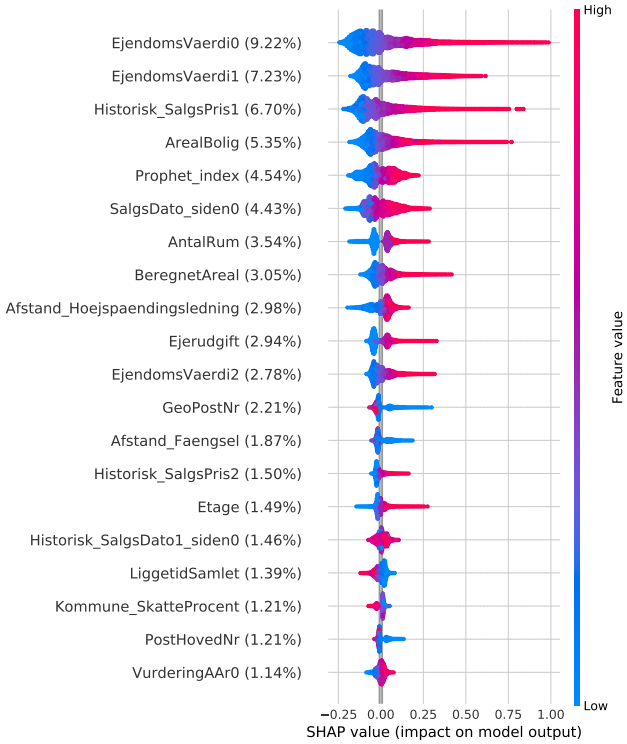


Figure 3.23: Feature importance of apartment prices using the XGB-model. The feature importance is measured using SHAP values. The variables are sorted top to bottom according to their overall feature importance, i.e. the previous public property valuation `EjendomdsVaerdi0` is the most important single 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.

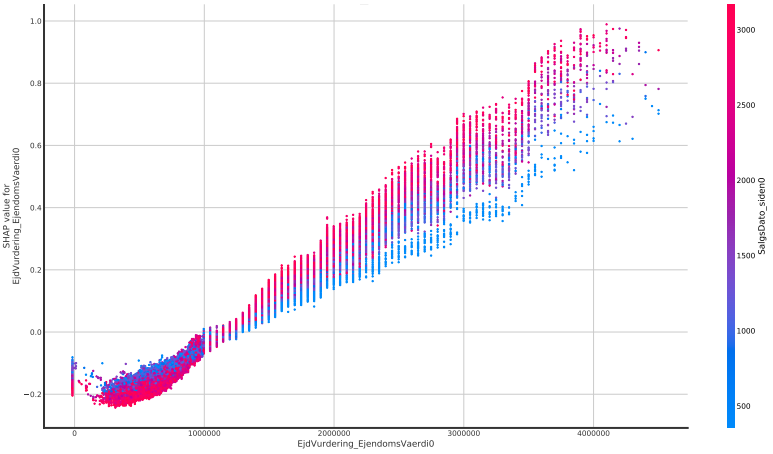


Figure 3.24: Feature importance of apartment prices using the XGB-model. XXX

3.8 *Multiple Models*

In addition to the XGBoost [31] (XGB) BDT model, several other models were also tested. During the sub-project the LightGBM [48] (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-imputed[24] and the input features were scaled[25] with a robust scaler from Scipy [79] 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.

[24] Which means that all invalid values were replaced with the median along each column-

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

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, Eucludian, see also subsection 2.4.1).

Finally, support vector machines[26] (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.

[26] For regression, also known as support vector regression[16].

The results for the five different models are shown in Figure 3.25, 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 test set, which confirms the visual clue: that XGBoost performs best, followed by the ensemble model.

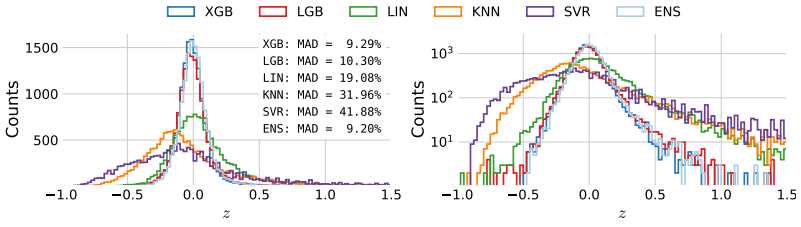


Figure 3.25: Multiple models. XXX

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 would 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 [60, 75]. 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 Polley and van der Laan [60]. 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 untill 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 data[27] this prevents overfitting. The meta learner then fits Z to the actual predictions of the training data y in the ususal way. The combination of a meta learner fitted to the predictions of

[27] The predictions for the very first fold is based on training data.

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:

$$\Psi_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^{5} \alpha_i \Psi_i(\mathbf{x}), \tag{3.10}$$

where $\boldsymbol{\alpha}$ 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 regresson model a simple squared error loss. Since a simple weighted average should work as the meta learner [60], a custom algorithm for finding $\boldsymbol{\alpha}$ according to MAD was implemented.

Given the training data, the evaluation function as a function of $\boldsymbol{\alpha}$ was minimized using of the MINUIT algorithm[47] via the iminuit[71] Python interface. It yielded decent result, yet they were 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 a the new initial parameter in the minimization routine. This yielded the following result:

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_{\text{LIN}} \\ \alpha_{\text{KNN}} \\ \alpha_{\text{SVR}} \\ \alpha_{\text{XGB}} \\ \alpha_{\text{LGB}} \end{bmatrix} = \begin{bmatrix} 0.202\,\% \\ 0.002\,\% \\ 0.001\,\% \\ 81.302\,\% \\ 20.002\,\% \end{bmatrix}. \tag{3.11}$$

The fact that it sums to more than 1 just corresponds to an overall scaling. When using the found $\boldsymbol{\alpha}$ in equation (3.10), one gets the ensemble model (ENS) shown in Figure 3.25 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.

3.9 *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 it[28]. With the dataset at hand, decent results were made, however, nowhere near the performance of the realtors'. There are two main resons 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 acces to a lot more data than ML models have. We are not in possesion 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 varibles that are in the data set but none of them describe the overall quality of the house, the maintenance level, the the age of the kitchen or bathroom. These features are invisble to the ML model.

During the project it was investigated how to get acces 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 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 3.26 where a housing-based model was trained with the five numerical variables `Ejendomsvaerdi0` (PPPV), `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 3.26. 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 approching, 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.

[28] Not only user-time programming it, but also computational ressources used.

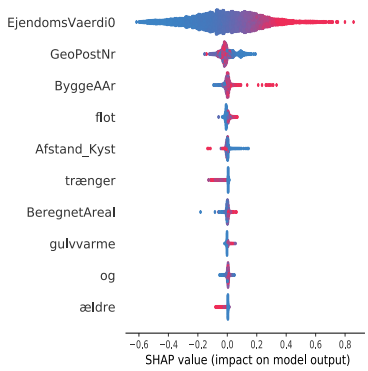


Figure 3.26: SHAP plot villa TFIDF XXX.

Another step would be to apply more modern deep learning[29] 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 data[30] [**?**]-]klambauerSelfNormalizingNeuralNetworks2017. 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.

[29] Basically advanced neural networks with many layers.

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

Part II

The second part of this thesis deals with particle physics and the discriminatory power of machine learning for quark-gluon identification and subsequent analysis.

4. Particle Physics and LEP

> *"Not only is the Universe stranger than we think, it is stranger than we can think."*
>
> — Werner Heisenberg

The aim of this chapter is to introduce the reader to the level of particle physics required for understanding the following chapter, in particular introducing the Standard Model in section 4.1, the theory behind quark hadronization in section 4.2, and the ALEPH detector at LEP in section 4.3. The goal is not to make a deep and thorough introduction to the field as this is not needed for the following analysis along with the fact that the author is no particle physicist himself.

4.1 The Standard Model

The *Standard Model* (SM) [39, 66, 81] of particle physics is the currently best known description of the elementary particles and thus describes the fundamental building blocks of our Universe. An overview of the particles explained by the Standard Model is shown in the typical tabular form seen in Figure 4.1. In general, particles comes in two categories: *bosons* and *fermions*.

The fermions, the left part of the figure, are particles with half-integer spin that obey Fermi-Dirac statistics and are further subdivided into *quarks* (upper left in figure) and *leptons* (lower left). The quarks interact with all of the four known forces[1], including the strong force. In contrary the leptons do not interact with the strong force. Quarks are never observed freely but are always combined into *hadrons* due to *color confinement* which is further explained in section 4.2. An example of this are protons which consists of two up-quarks and a down-quark. Leptons exist as either the charged leptons[2] or as neutral leptons, the so-called neutrinos[3]. The fermions come in three generations with increasing mass.

The bosons, the right part of the figure, are the force-carrying particles (with integer spin and which obey Bose-Einstein statistics) where the gluon g mediates the strong nuclear force (color charge), the photon γ mediates the electromagnetic force (charge), and the two $W^{\pm}$ and the Z bosons the weak nuclear force (weak isospin). The Higgs boson H, experimentally discovered in 2012 [32], does

[1] Gravity, electromagnetism, and the strong and weak force.

[2] The electron e, the muon μ, and the tau τ.

[3] The electron neutrine ν_e, the muon neutrino ν_μ, and the tau neutrino ν_τ.

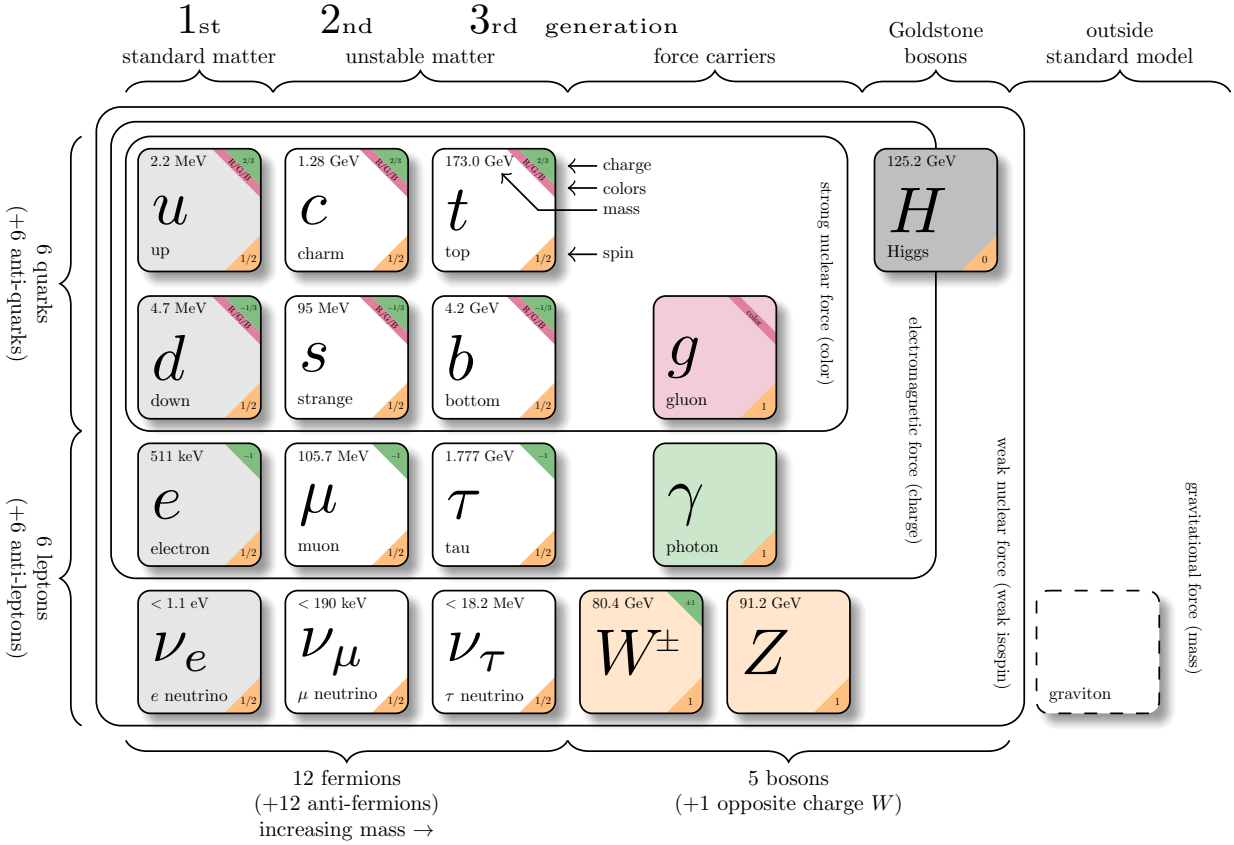


Figure 4.1: The Standard Model. Inspired by Purcell [62] using the template by Burgard [29] with manually updated masses according to Particle Data Group et al. [58].

not mediate any forces but interacts with all massive particles and explains why particles have mass.

All particles have antiparticles which are particles with opposite charge but the same mass. Some particles are their own antiparticles[4], such as the Z. At the Large Electron Positron collider (LEP), see section 4.3, electrons e^- and their antiparticles positrons e^+ were collided at an energy of around 91 GeV. This particular energy was chosen since this is at the resonance peak of the Z which mass distribution follows a Cauchy distribution (also known as Breit-Wigner) with mean[5] $m_Z = (91.1876 \pm 0.0021)\,\text{GeV}$ and a full width of $\Gamma_Z = (2.4952 \pm 0.0023)\,\text{GeV}$: LEP was as such a Z-factory. The Z, however, is only very short-lived with a half-life of $1/\Gamma_Z \sim 2.6 \times 10^{-25}\,\text{s}$. The decay mode for this unstable Z particle is primarily to hadrons ($(69.91 \pm 0.06)\,\%$) where the ratio (R) for b-quarks is $R_b = (Z \to b\bar{b}) = (15.12 \pm 0.05)\,\%$ and $R_g = (Z \to ggg) = (15.12 \pm 0.05)\,\%$ for gluons [58]. The fact that the Z is its own anti-particle forces its decay to be a particle–anti-particle decay (due to charge-conservation) where antiparticles are written with a a bar on top, e.g. the $\bar{b}$-quark is the antiparticle of the b-quark.

[4] The photon, the Z, and the Higgs.

[5] Calculated in natural units where $c = \hbar = 1$ which will also be used throughout this thesis.

4.2 *Quark Hadronization*

The electron-positron e^+e^- annihilations at LEP are complicated events that require advanced high-energy particle physics theory to be properly understood. Most of the aspects of the process is well-described by now, however, especially the hadronization process is still an area of active research. To better get an overview of the different stages of the e^+e^- annihilations, see the Feynman diagram in Figure 4.2.

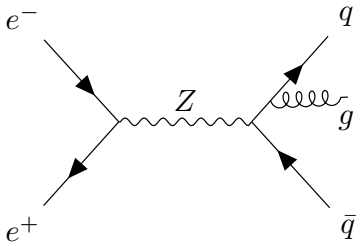


Figure 4.2: Feynman diagram showing the $e^+e^- \to Z^0$ production at LEP. The Z has several decay modes where the $Z \to q\bar{q}g$ is shown here.

Reading from left to right, the electron and the positron annihilates to a Z. This interaction is well-described by quantum electrodynamics (QED), a theory that has been around for more than 60 years by now. As mentioned in the previous section, the Z has several decays modes, yet most of these are background processes of no interest in this project and the focus for now will be the decay mode $Z \to q\bar{q}$ (Z to quark–anti-quark) as seen in the Feynman diagram. The particles produced by the Z-decay are called primary *partons*. Since this process involves quarks, and thus color charge, QED is no longer an adequate theory: quantum chromodynamics (QCD) is needed [15]. The $q\bar{q}$ pairs in this example acts as (color) dipoles from which a gluon can radiate. It can be shown with QCD that the gluon can only be radiated inside the cone that the $q\bar{q}$ pairs spans [23]. As mentioned in the introduction, quarks cannot exist freely (due to *confinement*) and we therefore cannot observe the $q\bar{q}g$ event produced in the Feynman diagram. Confinement is basically the QCD principle saying that quarks are always confined or bound inside hadrons. The initial partons (carrying color charge) are converted to (color-neutral) hadrons by non-perturbative QCD processes in what is called *hadronization*, and these hadrons can be measured.

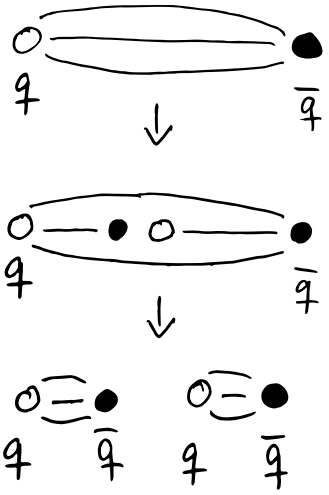


Figure 4.3: Illustration of the quarks splitting as explained by the Lund string model. For large charge separation the (color) field lines seem to be compressed to a tube-like region, where the strong interactions are mediated by the massless gluons (that couple to the color charge of quarks). When the two quarks are separated enough, the potential energy is released by the production of a new $q\bar{q}$ pair.

The hadronization process is not yet fully developed and currently two competing models for predicting the hadronization pattern exists: the Lund string model and the cluster model. In this project only the former of the models will be used. The Lund string model [14] is the theoretical framework underlying the widely used Monte Carlo event generator PYTHIA [69]. The string model is based on the observation that (color) field lines between quarks seem to compress into a tube-like region mediated by gluons, see the top part of Figure 4.3. The field can be described by a linearly rising potential $V(r) = \kappa r$ at large distances[6], where r is the distance and κ is the strength of potential [28]. This field is similar to the (constant) force of a string: $V(r) = \kappa r \Rightarrow F(r) = -\kappa$ where κ is the to be regarded as the spring tension. As quarks move apart, the potential energy stored in the "string" increases until it is large enough to "snap" and convert its potential energy into mass energy. This mass energy is released with the production of a new $q\bar{q}$ pair, see the rest of Figure 4.3.

An example of the hadronization process, or the transition from initial partons to final hadrons is sketched in Figure 4.4. Here the

[6] At small distances a Coulumb term has to be included, however, this term is assumed to be negligible by the Lund string model.

production of two kaons K^- and K^+, and two pions π^- and π^0 are shown. Since particles are created by "splits" in the "string", and the fact that there is energy-momentum conservation, they all have to share the total energy stored in the string. This is described by the fragmentation function:

$$f(z) \propto \frac{(1-z)^a}{z} \exp\left(-\frac{bm^2}{z}\right), \tag{4.1}$$

where $0 \leq z \leq 1$ is the remaining momentum that the new hadron takes, a and b are constants, and m is the mass[7] [23]. When the system runs out of available momentum, it will stop producing new hadrons and the fragmentation function thus explains the distribution of final state particles. The Lund string model can be extended from only $q\bar{q}$ events to $q\bar{q}g$ events where it predicts cones spanning the angular regions qg and $\bar{q}g$ should receive enhanced particle production compared to the $q\bar{q}$ region. This prediction by the Lund string model is also measured in e^+e^- collisions [28].

[7] Where $m \to m_\perp$ for particles with transverse momentum.

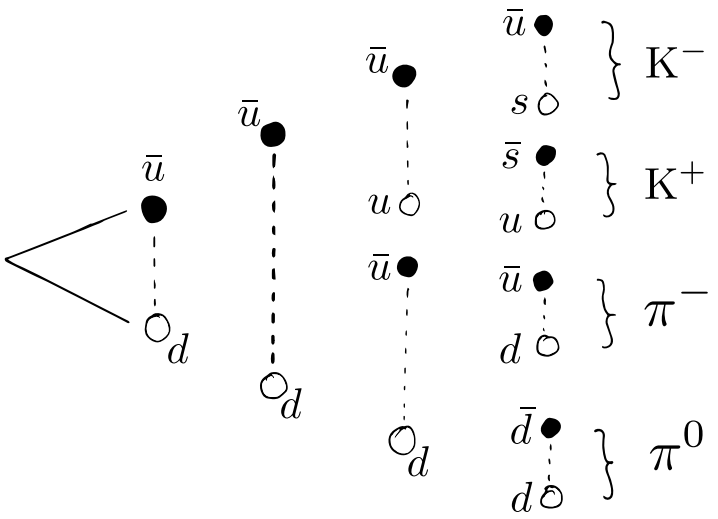


Figure 4.4: Illustration of the hadronization process by which $\bar{u}$- and d-quarks decay into four different mesons. The theoretical strings are shown as dashed lines and particles as circles, where filled circles are antiparticles.

The initial partons produced as Z decay therefore decay themselves to final state hadrons[8] which create a whole "shower" in the direction of the initial parton: this is called a *parton shower* and it is this parton shower, a *jet*, that is measured in the detector. The reverse computation from tracks measured in the detector is done with the use of *jet clustering* algorithms. The detector and the clustering algorithms are described in the following section.

[8] To either mesons which consist of two quarks (color–anti-color) or baryons (r-g-b) which consist of three quarks.

4.3 The ALEPH Detector and LEP

The Large Electron Positron collider (LEP) was a particle collider at CERN in Switzerland operating from 1989 to 2000. It collided counter-rotating bunches of electrons and positrons in a giant ring with a circumference of more than 26 km. The first phase, LEP1, ran from 1989 to 1995 at the Z resonance 91 GeV and the second phase, LEP2, continued afterwards closer to 200 GeV for W^+W^-

pair production [15], however, it is only the data collected at the energy $\sqrt{s} = 91.3\,\text{GeV}$ called the *Z peak data* that is used throughout the rest of this project. There were four independent detectors at the LEP experiment, one of them ALEPH[9].

[9] Together with DELPHI, L3, and OPAL.

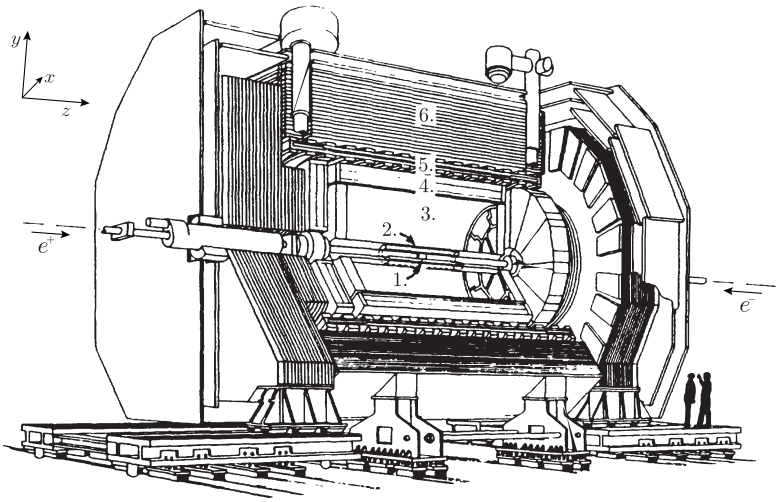


Figure 4.5: The ALEPH detector at LEP. 1) Vertex detector (VDET). 2) Drift chamber (ITC). 3) Time projection chamber (TPC). 4) Electromagnetic calorimeter (ECAL). 5) Superconducting magnet coil. 6) Hadron calorimeter (HCAL). Adapted from Buskulic et al. [30].

The *apparatus for LEP physics* (ALEPH) was a particle detector at LEP with a wide coverage, almost 4π, consisting of cylindrical subdetectors, see Figure 4.5, with the coordinate system shown in the upper left corner[10]. The polar angle θ is illustrated in Figure 4.6 together with the transverse (longitudinal) momentum $p_\perp$ (p_L) and the azimuthal angle ϕ in Figure 4.7. The goal of ALEPH was to measure the energy deposited in calorimeters by charged and neutral particles, measure the momenta of charged particles, measure the distance of travel of short-lived particles, and to identify the three lepton flavors (electron, muon, tau) [30]. As can be seen in Figure 4.5, ALEPH consisted of three detectors (the vertex detector (VDET), the drift chamber (ITC), and the time projection chamber (TPC)) and two calorimeters (the electromagnetic (ECAL) and the hadronic calorimeters (HCAL)).

[10] The z-axis pointing along the beam direction, the y-axis pointing upwards, and the x-axis pointing towards the center of LEP.

The detectors allow for precise tracking of the charged particles produced in the parton shower and the calorimeters of precise energy measurements for both charged and neutral particles going through the detector.

A hadronic event from a parton shower may leave a score of charged tracks resulting in hundreds of hits in the detectors (VDET, ITC, and TPC) which are fitted[11] with Kalman filters to obtain global track fits, of which bad charged tracks are discarded for further analysis. The tracks are helical due to the presence of a 1.5 T magnetic field which curves the charged particles according to their $p_\perp$.

The energy resolution σ of the calorimeters, or the *calorimeter performance*, is expected to increase with $\sqrt{E}$. In fact, it was found at ALEPH that the energy dependence of the resolution follows the

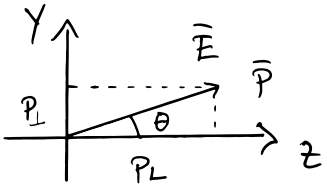


Figure 4.6: The polar angle θ defined in the zy coordinate system

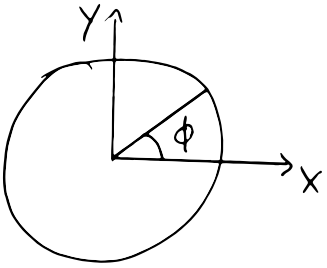


Figure 4.7: The azimuthal angle ϕ defined in the xy coordinate system.

[11] the process of fitting tracks is called *track reconstruction* in high energy particle physics.

parametrization [30]:

$$\sigma(E) = \Big((0.59 \pm 0.03) \cdot \sqrt{E/\mathrm{GeV}} + (0.6 \pm 0.3)\Big)\,\mathrm{GeV}. \tag{4.2}$$

Even though $\sigma(E)$ increases with E, the relative resolutions improves with higher energies. Since one never measures Nature directly, the results one obtains in a measurement are thus products of both model and experimental uncertainties folded together. To unfold the measurements to obtain experiment-independent results, the uncertainties are important to understand. Of course there are dozens of other uncertainties in an advanced experiment like LEP, however, the energy dependence is the primary focus in this project.

4.4 *Jet clustering*

Since the initial partons created as decay products from the Z are unstable themselves, what is measured in the detector is a whole shower of hadrons seen as charged tracks in the detectors and energy deposits in the calorimeters. However, say that the Z decayed to a $b\bar{b}$ event. In this case the two b's would be back-to-back and the final hadrons would be observed approximately in the same direction as the b's were created. The interest of the experiment is not to measure the final hadrons, but rather to infer information about the initial quarks and gluons. This is done via the reverse-engineering process called *jet clustering*. Over the years many clustering algorithms have been developed, however, most of these are younger than LEP. In the ALEPH experiment the JADE algorithm was used [19]. JADE is a sequential recombination algorithm where final state particles are initially described as individual so-called pseudo-jets which are then recursively merged to larger jets according to their inter-jet distance d_{ij}^2. The distance measure for JADE is:

$$d_{ij}^2 = \frac{2E_iE_j(1-\cos\theta_{ij})}{E_{\mathrm{vis}}^2}, \tag{4.3}$$

where E_{vis} is the visible energy[12] and θ_{ij} is the angle between jet i and j. The JADE algorithm computes d_{ij}^2 for all combinations of jets and merges the two jets with the lowest d_{ij}^2, continuing like that recursively until $\min(d_{ij}^2) > d_{\mathrm{cut}}^2$ for some predefined value of d_{cut}^2. In the dataset at hand, only the final jets were available and not the jet constituents, unfortunately.

[12] The total sum of energies in the event.

4.5 *The variables*

The overall goal of the project is to be able to discriminate quarks and gluons using only vertex variables. The reason for the last condition is that the goal is to better understand the shape distributions of gluons in which there is still significant differences between Monte Carlo (MC) simulations and Data. Therefore only vertex

variables will be used to avoid any biases introduced by using shape-related variables to detect differences in shape-distributions. The vertex variables are a subset of all variables which include the three variables `projet`, `bqvjet`, and `ptlrel`. These three particular variables have each shown discriminatory power in separating b-quarks from light quarks and gluons.

`projet`: For each track in the jet an impact parameter δ is computed. This parameter is the minimum distance between the estimated Z decay point and the track itself and its sign depends on whether or not the point of closest approach is in front of or behind the Z decay point (relative to the momentum). From δ the significance $\mathcal{S}$ – which is δ/σ_δ – is computed and is thus a measure of the certainty of a measured track. High values of $\mathcal{S}$ is typically an indicator of b jets, since long-lived particles typically decay in front of the Z relative to the jet direction, while uds-jets might as well have negative values of $\mathcal{S}$. From $\mathcal{S}$ the track probability $\mathcal{P}_{\text{track}}$ of a track originating at the decay point of the Z can be computed, which can further be aggregated across all tracks within a jet to form the jet probability $\mathcal{P}_{\text{jet}}$ which `projet` is a function of [35]. Whether or not $\mathcal{P}_{\text{jet}}$ is strictly a probability can be discussed but it is related to the probability of all tracks within a jet to originate from long-lived particles, which itself is a good indicator of being a b- (or c-) jet. This variable further has the advantage of being independent of any vertex algorithm.

`bqvjet`: For any jet with good[13] charged tracks, a fit with a (hypothetical) secondary vertex is performed. The difference in χ^2 between the null hypothesis that all good tracks originate from the same primary vertex and the alternative hypothesis that a secondary vertex exists in addition to the primary one is calculated. For the long-lived massive b and c quarks this typically results in large differences in χ^2 compared to uds- and gluon jets which have much lower $\Delta\chi^2$-values [15]. The `bqvjet` is related to the $\Delta\chi^2$-value from the secondary vertex algorithm. This value is dependent of the vertex algorithm, but still explores other areas of phase space than `projet`, however, they are still very correlated[14].

[13] Meaning that there are at least four TPC hits and the fit has a reduced χ^2 of less than four [15]

[14] With a linear correlation of $\rho = 0.82$ in the dataset.

`ptlrel`: If any leptons (in the case of $e^\pm$ or $\mu^\pm$) are measured by the detector, this is a good sign of the jet originating from a b-quark. The high mass of the b quark leads to high $p_\perp$ for the leptons relative to the jet axis which is exactly measured by `ptlrel`.

The fact that the heavy b-quarks have much longer lifetimes than the lighter uds-quarks stems from their much lower inter-coupling

magnitudes written as the CKM matrix $\mathbf{V}$ [58]:

$$\mathbf{V} = \begin{array}{c} \\ u \\ c \\ t \end{array} \begin{array}{c} \begin{array}{ccc} d & s & b \end{array} \\ \begin{pmatrix} 0.97446 & 0.22452 & 0.00365 \\ 0.22438 & 0.97359 & 0.04214 \\ 0.00896 & 0.04133 & 0.99911 \end{pmatrix} \end{array}. \tag{4.4}$$

The matrix element $|V_{ij}|^2$ is proportional to the transition-probability of quark i transitioning to quark j. From the CKM matrix it can be seen that u and d quarks couples strongly together, likewise with c-s and b-t quark pairs. When a Z decays into a b-quark, this quark couples strongly with the top quark, however, due to the high mass of the top quark compared to the center-of-mass energies at LEP1, the b-quark cannot decay into a t-quark but must (almost always) decay to a c, however, still with low probability, $V_{bc} \ll 1$. This, together with the fact that $V_{bu} \ll V_{bc}$ explains the long life-time of b quarks. This is also why the three variables above are very common variables for b-tagging algorithms. That c-quarks also have relative long life-times are not due to the CKM elements, as for b-quarks, but rather due to the c-decay being governed by the weak force through virtual W^* bosons, a force that is much slower than the strong force. This also happens for b-quarks which further explains why c-quarks share many similarities with b-quarks but also resembles resembles light-quarks (which are very short-lived.).

The rest of the non-vertex variables are:

`ejet` : The energy of the jet E_{jet}

`costheta` : The cosine of the θ angle defined in Figure 4.6: $\cos\theta$.

`phijet` : The angle ϕ of defined in Figure 4.7: ϕ.

`sphjet` : The sphericity tensor $\mathbf{S}$ is defined as:

$$S^{(\alpha\beta)} = \frac{\sum_{i=1}^{N} p_i^{(\alpha)} p_i^{(\beta)}}{\sum_{i=1}^{N} |p_i|^2} \quad \alpha, \beta \in \{x, y, z\}, \tag{4.5}$$

and the sphericity is determined as $S = \frac{3}{2}(\lambda_2 + \lambda_3)$ where $\lambda_1 \geq \lambda_2 \geq \lambda_3$, $\lambda_1 + \lambda_2 + \lambda_3 = 1$ are the three eigenvalues of the sphericity tensor. The sphericity $0 \leq S \leq 1$ is a measure of the angular distribution of the tracks in a jet. When $S = 0$ the jets form a perfect sphere, compared to $S = 1$ for a perfect jet. The `sphjet` variable is the sphericity of the jet when calculated in its boosted rest frame, also known as *boosted sphericity*.

`pt2jet` : The sum of the square of transverse momentum w.r.t. the jet axis: $\sum_i p_{\perp,i}^2$.

`muljet` : The multiplicity of the jet.

For further details about the variables, see Armstrong [15].

The variables explained above are all used in the following analysis where the machine learning model is trained on onluy the vertex variables to probe differences in the shape-distributions of the shape-variables. The goal of this is to better understand the gluon hadronization process to minimize differences in MC simulations and ultimately get a better understanding of the rules governed by Nature.

5. Quark Gluon Analysis

As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena.

5.1 Data Preprocessing

The data consists of 43 data files taken between 1991 and 1995 totalling 3.5 GB (Data). Along with this comes 125 files based on Monte Carlo (MC) simulations (8.4 GB) and additional 42 MC-files with only b-quark events (MCb) simulated (2.1 GB). The data files which are in the form of *Ntuples*, ROOT's data format [27], are converted to HDF5-files by using uproot [7]. While iterating over the Ntuples, some basic cuts are applied before exporting the data to HDF5. The first one being that the (center of mass) energy E in the event has to be within $90.8\,\text{GeV} \leq E \leq 91.6\,\text{GeV}$ to only use the Z peak data. The second one being that the sum of the momenta p_{sum} in each event is $32\,\text{GeV} \leq p_{\text{sum}}$ to remove any $Z \to \tau^+\tau^-$ events. To ensure a primary vertex, at least two good tracks are required where a good track is defined as having 7 TPC hits and $\geq$ 1 silicon hit. Finally it is required that the cosine of the thrust axis polar angle, which is the angle between the trust axis and the beam, is less than or equal to 0.8 to avoid any low angle events since the detector performance worsens significantly in that region. These cuts were standard requirements for the ALEPH experiment.

One last cut which was experimented with was the threshold value for *jet matching*. The jet matching is the process of matching the jet with one of the final state quarks. The jet is said to be matched if the dot product of between the final quark momentum and the jet momentum is more than then threshold value. Higher thresholds means cleaner jets but at the expense of less statistics. A jet matching threshold of 0.90 was found to be a good compromise between purity and quantity where 97.8 % of all 2-jet events are

matched and 96.7 % of all other jets were matched[1].

The data structure is quite differently structured in the Ntuples compared to normal structured data in the form of tidy data [82]. The data is organized such that one iterates over each event where the variables are variable-length depending on the number of jets in the events; this is also known as *jagged* arrays. The data is unjagged[2] before exporting to HDF5-format and only the needed variables are kept. This reduces the total output file to a 2.9 GB HDF5-file for both Data, MC, and MCb.

The number of events for each number of jets can be seen in Table 5.1 for the Data and in Figure 5.2 for the MC and MCb.

[1] Compare this to 98.5 % and 97.8 % for a threshold of 0.85 or 95.9 % and 93.9 % for a threshold of 0.95.

[2] Such that e.g. a 3-jet event will figure as three rows in the dataset.

	jets	events
2	2 359 738	1 179 869
3	3 619 290	1 206 430
4	854 336	213 584
5	52 775	10 555
6	510	85
Total	6 886 649	2 610 523

Table 5.1: The dimensions of the dataset for the actual Data. The numbers in the jet columns are the number of events multiplied with the number of jets; e.g. $85 \cdot 6 = 510$.

	jets	events
2	7 293 594	3 646 797
3	10 780 890	3 593 630
4	2 241 908	560 477
5	103 820	20 764
6	588	98
Total	20 420 800	7 821 766

Table 5.2: The dimensions for the MC and MCb datasets.

5.2 *Explanatory Data Analysis*

Since the machine learning models are only trained on the three vertex variables `projet`, `bqvjet`, and `ptljet` – see chapter 4 for a deeper introduction to these variables – these variables will be the primary focus of this section. Given the fact that MC-simulated data exists, the truth of each simulated event is also known. This allows us visualize the difference between the different types of quarks. In the MC simulation each event are generated such that the type of quark, or *flavor*, is known and assigned the variable `flevt`. The mapping from flavor to `flevt` is:

Flavor:	*bb*	*cc*	*ss*	*dd*	*uu*
`flevt`:	5	4	3	2	1

In addition to knowing the correct flavor, we define that an event is *q-matched* if one, and only one, of the jets are assigned to one of the quarks, one, and only one, of the jets are assigned to the other quark, and no other jets are matched to any of the quarks. We can then define what constitutes a *b*-jet: if it has `flevt` $= 5$, the entire event is *q*-matched, and the jet is matched to one of the quarks. Similarly we define *c*-jets only with the change that `flevt` $= 5$, and *uds*-jets with `flevt` $\in \{1, 2, 3\}$. A gluon jet is defined as an any-flavor event which is *q*-matched but the jet is not assigned to any of the quarks. Strictly speaking, this means that *g*-jet is not 100 % certain of being a gluon, however, since the MC simulation does not contain this information this is the only option. Due to the *q*-match criterion this also means that some jets are assigned the label "non-*q*-matched" which is regarded as background. The distribution of different types of jets can be seen in Table 5.3 and shown as relative numbers in Table B.1 in the appendix.

With the criteria defined above for what constitutes a specific type of jet the 1D-distributions for the three vertex variables is plotted in Figure 5.1. For all three subplots the histograms are show with a logarithmic *y*-axis, all *b*-jets in blue, *c*-jets in red, *g*-jets in green and all jets in orange. In fully opaque color are shown the distributions for 2-jet events, in dashed (and lighter color) 3-jet

	b	c	uds	g	non-q-matched
2	2 713 454	944 380	2 125 900	0	1 509 860
3	2 433 878	964 212	2 129 218	3 365 969	1 887 613
4	326 264	156 332	336 548	1 012 198	410 566
5	10 332	5960	12 668	54 525	20 335
6	42	26	52	320	148
Total	5 483 970	2 070 910	4 433 012	4 604 386	3 828 522

Table 5.3: Number of different types of jets for MC and MCb. See also Table B.1 in the appendix for relative numbers.

events, and in semi-transparent 4-jet events. In the left subplot the `projet` variable is plotted where it can be seen that high values of `projet` tend to indicate b-jets. In the middle subplot `bqvjet` is plotted which shares many similarities with the `projet`-variables, including that high values indicate b-jets. In the right subplot the `ptljet` is plotted. This variable has many zeros in it which correlates with mostly with gluon[3] and large values are mostly due to b-jets. In general it is clear to se how the differences in distribution between the 2-, 3-, and 4-jet events are minor, with the one exception of 2-jet events which does not contain any gluons at all.

[3] Around 98 % of all g-jets are zeros compared to $\sim$ 82 % for c-jets and $\sim$ 70 % for b-jets.

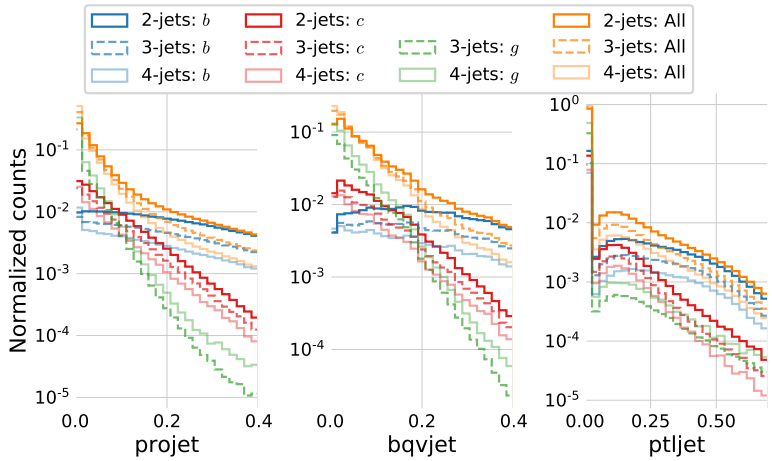


Figure 5.1: Normalized histograms of the three vertex variables: `projet`, `bqvjet`, and `ptljet`. In blue colors the variables are shown for true b-jets, in red for true c-jets, in green for true g-jets, and in orange for all of the jets (including non q-matched). In fully opaque color are shown the 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.

Even though there are only three vertex variables, it is difficult to properly get an intuition about how easily separated they different types of jets are. Since there are millions of points a single 3D scatter plot quickly becomes overcrowded in one wants to plot all jets. We apply dimensionality reduction from the three dimensions down to two dimensions by using the UMAP algorithm [55]. Within recent years the field of dimensionality reduction algorithms has grown a lot from just the typical (linear) principal component analysis to also include non-linear algorithms. The t-SNE algorithm [76] deserves an honorable mention since this algorithm revolutionized the usage of (nonlinear) dimensionality reduction algorithms in e.g. bioinformatics [74, 80] yet its mathematical foundation has strongly been improved with the never, faster UMAP algorithm [55] which usage is also expanding [20, 21, 33].

The aim of UMAP, short for Uniform Manifold Approximation and Projection, is to correctly identify and preserve the structure, or topology, of the high-dimensional feature space in a lower-dimensional output space. It does so by trying to stitch together local manifolds in the high-dimensional feature space such that the difference between the high- and low-dimensional representations is minimized according to the cross-entropy such that both global structure and local structure is preserved [55]. Compared to t-SNE the approach in UMAP has an algebraic topological background compared to the more heuristic approach taken by t-SNE. Note that the UMAP algorithm is not provided any information about which jets are which types.

The UMAP algorithm has several hyperparameters, where two of the most important ones are the number of neighbors `n_neighbors` which controls the priority between correctly preserving the global versus the local structure, and the `min_dist` which defines how tightly together UMAP is allowed to cluster the points in the low-dimensional representation. To properly choose the best combination of `n_neighbors` and `min_dist` a grid search with `n_neighbors` $\in \{10, 50, 100, 250\}$ and `min_dist` $\in \{0, 0.2, 0.5\}$ is performed. This is shown for 4-jet events in Figure B.1 in the appendix. In this case the choice of best combination of `n_neighbors` and `min_dist` is subjective at best, but it was judged by the author that `n_neighbors` $= 250$ and `min_dist` 0.2 gave the best compromise between preserving local and global structure. The results of running UMAP on 4-jet events can be seen in Figure 5.4. Here the millions of points are plotted using Datashader [8] to avoid overplotting and colored according to the jet type. From the figure it is seen how there are some clear, blue b-jet clusters, however, most of the data seem to be a mix of g- and uds-jets. The plots with the same UMAP parameters for 3-jet and 2-jet events are seen in Figure 5.3 and 5.4.

These figures suggests that it should be possible to discriminate the b-jets from the other jets somewhat, however, no clear separation is expected. The t-SNE algorithm was also tested but showed inferior performance compared to UMAP, see Figure B.2 in the appendix for an example of this.

The correlation between the vertex variables can be seen in Figure 5.5, where the upper diagonal shows the linear correlation ρ and the lower diagonal shows the (estimate of the) MIC non-linear correlation MIC_e. Here it ca be seen that `projet` and `bqvjet` correlate mostly whereas the other variables correlate a lot less. Had they all correlated a lot, it would be more difficult to extract any meaningful insights from the system at it would contain less information.

figures/quarks/df_UMAP-X=1120952-n_neighbors=250

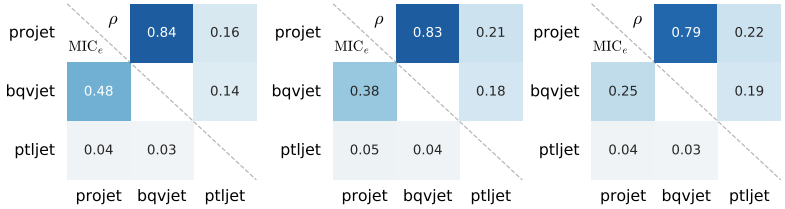
Figure 5.2: Visualization of the vertex variables for the different categories: true b-jets in blue, true c-jets in red, true uds-jets in green, true g-jets in orange, and non q-matched events in purple. The clustering is performed with the UMAP algorithm which outputs a 2D-projection. This projection is then visualized using the Datashader which takes takes care of point size, avoids over and underplotting, and color intensity.

figures/quarks/df_UMAP-X=1078089-n_neighbors=250

Figure 5.3: UMAP visualization of vertex variables for 3-jet events.

figures/quarks/df_UMAP-X=729358-n_neighbors=250

Figure 5.4: UMAP visualization of vertex variables for 2-jet events.



(a) 2-jet events (b) 3-jet events (c) 4-jet events

Figure 5.5: Correlation of the three vertex variables for 2-, 3- and 4-jet events.

5.3 *Loss and Evaluation Function*

In contrary to the housing prices subproject the goal in this project is to predict the class of particles, or the types of jets, where the so-called *signal* observations[4] are often assigned the label 1 and *background* observations 0. The combination of this being a *classification* problem (compared to a regression problem) along with the fact all the variables are actual measurements from a particle physics accelerator means that the issue of outliers is negligible. This also means that the problem of finding a robust loss function is non-existent since the in classification loss is already bounded in the $[0, 1]$-interval.

[4] Often called *signal events*, however, this term would require that each event constitutes a single data point in the dataset which it does not here.

Classically *accuracy* is often used as loss function for classification which is simply the fraction of correct predictions, however, accuracy as a metric suffers a lot when handling *imbalanced* data: when the ratio between the number of instances of each class is not approximately $(50 : 50)\%$. The problem is that if the sample contains 90 % background and only 10 % signal, then a simple model which simply predicts everything to be background will have a 90 % accuracy.

To circumvent this issue, the area under the ROC curve (AUC) is used, where the ROC[5] curve is the the *signal efficiency* ε_{sig} of the ML model plotted as a function of the *background efficiency* ε_{bkg}. The definition of these two measures are:

[5] Receiver Operating Characteristic.

$$\varepsilon_{\text{sig}} = \frac{S_{\text{sel}}}{S_{\text{tot}}}, \qquad \varepsilon_{\text{bkg}} = \frac{B_{\text{sel}}}{B_{\text{tot}}}, \tag{5.1}$$

where S_{sel} are signal events that were also selected (predicted) as signal by the ML model, $S_{\text{tot}} = S_{\text{sel}} + S_{\text{rej}}$ is the total number of signal events (the selected and rejected), and likewise for background events B. Within the machine learning community the signal efficiency is called the true positive rate (TPR) and the background efficiency the false positive rate (FPR). For the rest of this project, the AUC will be the evaluation function $f_{\text{eval}} = \text{AUC}$, however, since this metric does not work on single observations it cannot be used as the loss function. Instead we will use the *log-loss* as the loss function[6] which not not only is differentiable for single predictions, compared to AUC, but also takes the certainty of the prediction into account. When using tree-based algorithms or neural networks one can extract not only whether or not a single observation is classified as signal or background but also a prediction score. This is a number in the $[0, 1]$-interval and the closer to 1 the score is, the

[6] In the context of machine learning this is the same as the *cross entropy*.

more certain the model is of the prediction being signal. Given the prediction score $\hat{y}$ and the true label y, the log-loss $\ell_{\log}$ is calculated as:

$$\ell_{\log} = -y \log \hat{y} - (1 - y) \log (1 - \hat{y}). \tag{5.2}$$

This is visualized in Figure 5.6. Here it can be seen how the loss changes as a function of the prediction score. Notice that when $y = 0$ the loss for $\hat{y} = 1$ diverges towards ∞ and likewise with $y = 1$ and $\hat{y} = 0$ (since $\log 0$ diverges to $-\infty$).

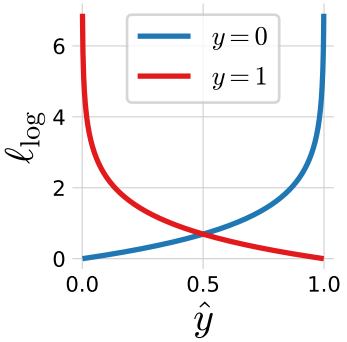


Figure 5.6: Plot of the log-loss $\ell_{\log}$.

5.4 *b-Tagging Analysis*

The ability to discriminate between the different types of particles produced in a collision is obviously import to understand the results. Today much work go into tagging algorithms from b-tagging in ATLAS and CMS [67] but this work started even decades ago. That b-quarks are tagged specifically is both due to b-quarks having more unique characteristics compared to e.g. c-quarks and are thus easier to tag, but also the fact that b-quarks are the second-heaviest of the quarks and are measured to better understand CP[7]-violation at LHC-b, contributes to the choice of tagging b-quarks. In ALEPH Proriol et al. [61] started the work of comparing different methods for b-tagging already in 1991. They concluded that a neural network had the best performance compared to e.g. a linear (Fisher) discriminant. The neural network used was a 3-layer neural network (NN) trained on nine variables and the output `nnbjet`. For this of this project this pre-trained network will be called NNB.

[7] Short for charge-parity.

The data are split[8] into training and test sets in such a way that the individual jets in an event are not split. As such, the splitting is performed at event-scale in a $(80 : 20)\%$ train-test ratio.

[8] After removing all low-energy jets such that all events that contain any jets with an energy of less than 4 GeV are removed.

5.4.1 *b-Tagging Hyperparameter Optimization*

Compared to the housing prices dataset, the number of observations N is a lot larger, although the dimensionality M is much smaller ($3 \ll 143$). Therefore both XGBoost (XGB) and LightGBM (LGB) were included as models initially since their performance in the housing dataset was very similar but LightGBM was expected to quite a lot faster on this dataset, which also turned out to be the case. The models were hyperparameter optimized (HPO) using random search (RS) since the Bayesian optimization (BO) did not show any performance gains compared to RS. They were run with 5-fold cross validation and early stopping with a patience of 100. The PDFs for the random search for the LightGBM model can be seen in Table 5.4, and the ones for XGBoost in Table B.2 in the appendix. The random search has been run with 100 iterations for LightGBM and only 10 for XGBoost since XGBoost is slow at fitting datasets of this size[9]. The results of the HPO for 3-jet and 4-jet events can be seen in Figure 5.7. For 3-jets it can be seen how most of the iterations share about the same performance within 1σ, however some

Hyperparameter	Range
`subsample`	$\mathcal{U}(0.4, 1)$
`colsample_bytree`	$\mathcal{U}_{\text{trunc}}(0.4, 1, 2)$
`max_depth`	$\mathcal{U}_{\text{int}}(-5, 63)$
`num_leaves`	$\mathcal{U}_{\text{int}}(7, 4095)$

Table 5.4: Probability Density Functions for the random search hyperparameter optimization process for the LightGBM model. For an explanation of $\mathcal{U}_{\text{trunc}}$, see section 5.5. All negative values of `max_depth` are interpreted as no max depth by both LGB and XGB.

[9] See page 70 for a discussion of the timings.

iterations have a significantly decrease in performance. For 4-jets there are not any iterations which share the same bad performance relative to the others as some of the 3-jets.

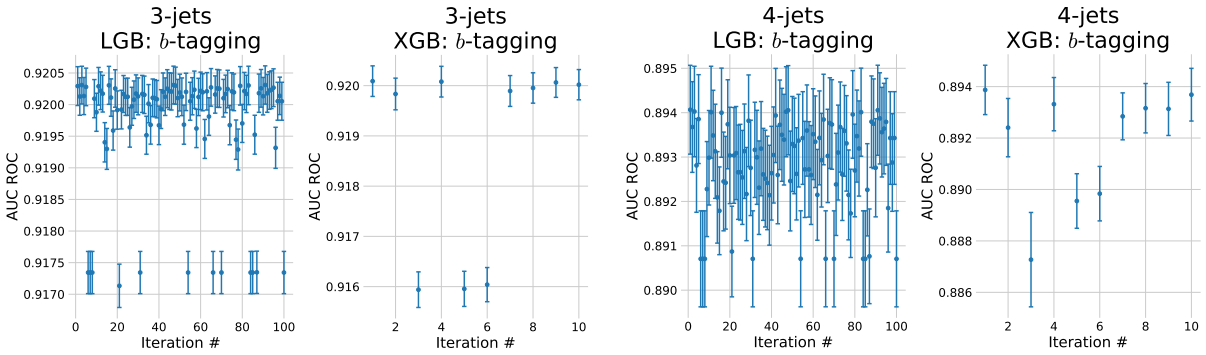


Figure 5.7: Hyperparameter Optimization results of b-tagging with random search. From left to right, we have A) 100 iterations of RS with LGB on 3-jets, B) 10 iterations of RS with XGB on 3-jets, C) 100 iterations of RS with LGB on 4-jets, D) 10 iterations of RS with XGB on 4-jets. Notice the different ranges on the y-axes.

The relationship between the different hyperparameters in 4-jet events can be seen in the parallel coordinate plot in Figure 5.8. First of all the importance of the column downsampling `colsample_bytree` variable is significant: all of the low-performing hyperparameter sets have a low value of this hyperparameter. Since $M = 3$ for the vertex variables this makes logical sense; using only $\text{int}(\sim 0.5 \cdot 3) = 1$ variable[10] the model cannot properly learn the structure in the data. Compared to the column downsampling, the other hyperparameters are notably less important. The same overall conclusion can be inferred in the 3-jet case, see Figure B.3 in the appendix.

[10] See section 5.5 for a deeper discussion about the `colsample_bytree` hyperparameter.

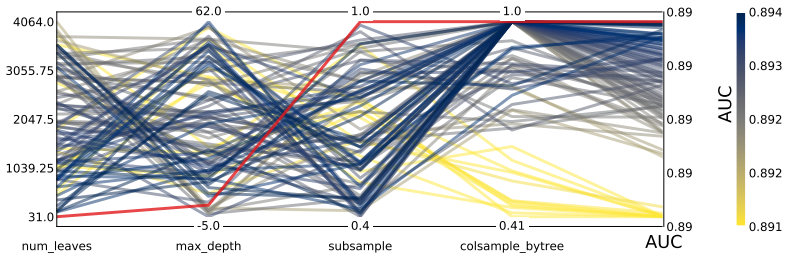


Figure 5.8: Hyperparameter optimization results of b-tagging for 4-jet events. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of 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.

5.4.2 *b-Tagging Results*

The prediction score for the b-tagging models is usually called the b-tag and will be written as β_{tag}. The distribution of β_{tag} for the two HPO-optimized models, LGB and XGB, together with the pre-trained neural network NNB can be seen in Figure 5.9 for 4-jet events and in B.4 in the appendix for 3-jet events. Notice the strong match between the NNB and LGB models. The XGB model has almost no high b-tags $\beta_{\text{tag}} > 0.8$, but a majority of b-tags in the very low end. This indicates that the XGBoost has focussed on the background events compared to the signal events, whereas the NNB and LGB models have focused more on the signal events.

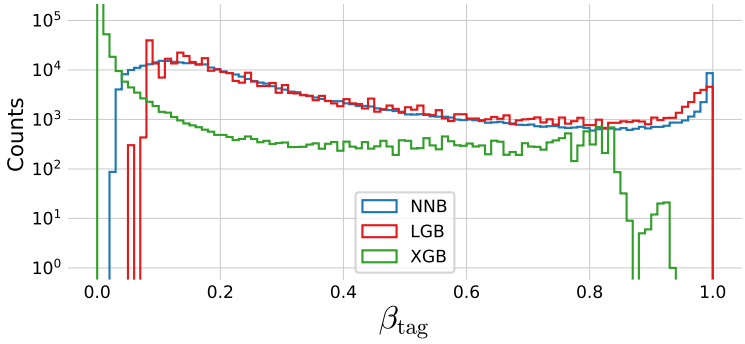


Figure 5.9: Histogram of b-tag scores β_{tag} in 4-jet events for NNB (the neural network pre-trained by ALEPH, also called `nnbjet`) in blue, LGB in red, and XGB in green.

Even though the distributions of b-tags are different between the three models, the real performance plot for classification is the ROC curve seen in Figure 5.10 for 4-jet events. Here the signal efficiency ε_{sig} is plotted as a function of the background efficiency ε_{sig} with the AUC shown in the bottom right corner. The LGB and XGB models performs similarly well with an AUC $= 0.896$ compared to the NNB with AUC $= 0.884$. The differences between the models are even smaller for 3-jet events seen in Figure B.5 in the appendix. In general the LGB and XGB models are so similar that they cannot be distinguished from another in any of the plots and their difference in AUC is on the forth decimal point. However, the LGB model is several times faster than the XGB model. In comparison, 10 iterations of HPO using RS on 3-jet events with XGB took more almost 34 hours on HEP[11] compared to just 23 hours for 100 iterations for LGB. The same performance difference was seen in 4-jet events where the timings were 4 hours for XGB compared to 2.5 hours for LGB, and thus XGB is dropped in all subsequent analysis.

[11] The local computing cluster.

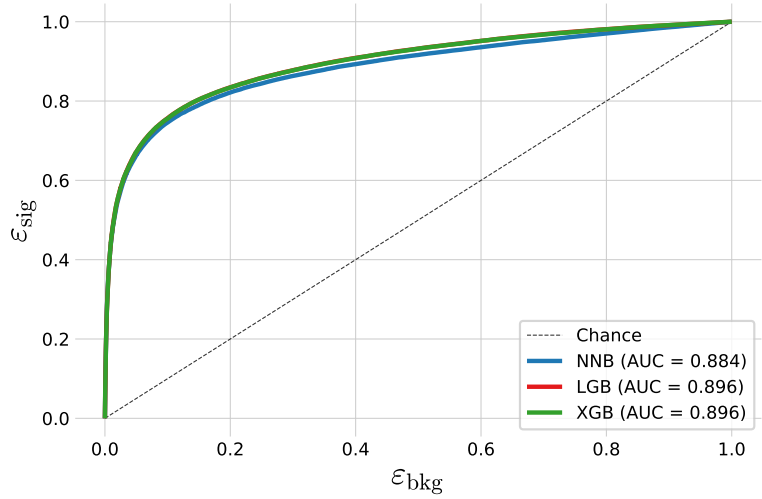


Figure 5.10: ROC curve of the three b-tag models in 4-jet events for NNB (the pre-trained neural network trained by ALEPH, also called `nnbjet`) in blue, LGB in red, and XGB in green. In the legend the area under curve (AUC) is also shown. Notice that the LGB and XGB models share performance and it is thus due to overplotting that only the green line for XGB can be seen. In the machine learning community the background efficiency ε_{bkg} is sometimes know as the false positive rate (FPR) and the signal efficiency ε_{sig} as the true positive rate (TPR).

5.4.3 *b-Tagging Model Inspection*

To get a better understanding of the trained LGB model, the global SHAP feature importances can be seen in Figure 5.11 for 4-jet events. First of all it is noted that the `projet` has global feature

importance of 57.32 %, `bqvjet` 29.16 %, and `ptljet` 13.52 %. For all three variables it is seen how most of the points have many small feature values which has a negative impact on the model output however small. Especially the `ptljet` has many features with a low value (0 in fact) yet this does not pull the model too much towards background events compared to if a jet has a high value of `ptljet` which has a strong, positive impact on the output prediction.

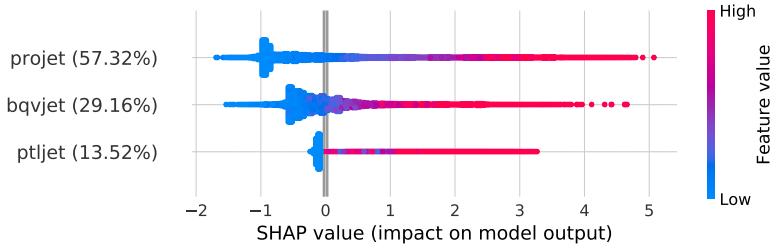


Figure 5.11: Global feature importances for the LGB b-tagging algorithm on 4-jet events. The normalized feature importance is shown in the parenthesis and shov ıe SHA .

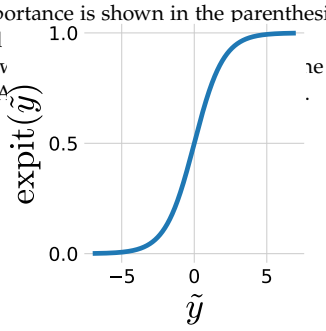


Figure 5.12: The expit function.

In regression, the model output is a continuous prediction $\hat{y}_{\text{reg}} \in \mathbb{R}$. In classification what is actually happening under the hood is that the model predicts a value $\tilde{y} \in \mathbb{R}$ which is transformed to a number in the $[0, 1]$-interval via the *expit* function:

$$\text{expit}(\tilde{y}) = \frac{e^{\tilde{y}}}{1 + e^{\tilde{y}}} \equiv p, \tag{5.3}$$

where p is a number in the $[0, 1]$-interval. The expit function is also sometimes known as the logistic function and is visualized in Figure 5.12. Its inverse is the *logit* function:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \tilde{y}, \tag{5.4}$$

which is visualized in Figure 5.13. The fraction in equation (5.4) is called the *odds* and the logit-transformed value of p, $\text{logit}(p) = \tilde{y}$, is thus sometimes called the *log-odds*. It is in this log-odds space that LightGBM makes its predictions and the SHAP values in Figure 5.11 are also in log-odds space. The additivity[12] of SHAP is in this log-odds space.

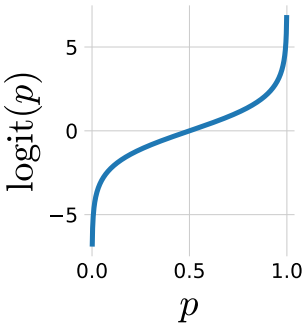


Figure 5.13: The logit function.

With this in mind, single predictions of the LGB b-tagging model can be understood with SHAP which Figure 5.14 is an example of. This figure shows the logic behind the models prediction for this particular jet. That the bias is negative reflects that there is a majority of background compared to signal[13]. This particular event has `projet` $= 1.003$, `bqvjet` $= 0.529$, and `ptljet` $= 0$. In the plot it is seen how this high value of `projet` has the greatest impact on the model prediction, while the medium value of `bqvjet` also pushes the model prediction towards a signal-prediction. The four bars in the left part of the plot are all in log-odds space and their sum is shown as the blue bar to right, where the right y-axis shows the value in probability space $p \in [0, 1]$. This jet was in fact a b-jet.

[12] See also section 2.8.

[13] There are 22.1 % b-jets in the 3-jet training set.

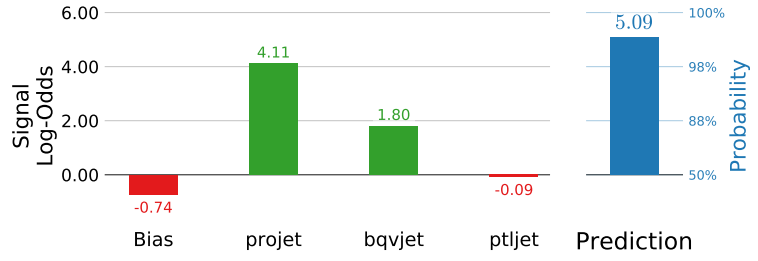


Figure 5.14: Model explanation for the 3-jet b-tagging LGB model for a b-like jet. The first column is the bias of the training set which acts as the naive prediction baseline, the rest are the input data variables. On the right hand side of the plot is the model prediction shown. The left part of the plot is shown in log-odds space, the right part in probability space. The negative log-odd values are shown in red, positive ones in green, and the prediction value in blue.

5.5 *Truncated Uniform PDF*

Initially when plotting the HPO performance as a function of iteration, it was seen how there were three significant plateaus, where the highest plateau (i.e. highest AUC value and thus best score) was only seen in the very first iteration. It was quickly realized that this was due to the very first iteration was being run with the default values of the LGB and XGB models in the custom implementation by the author. However, what was not understood was why this value was performing so much better than random sets of hyperparameters[14]. During the debugging process the column downsampling `colsample_bytree` was diagnosed to be the culprit. The default value is `colsample_bytree` $= 1$, however, the probability density function (PDF) used in RS for this parameter was $\mathcal{U}(0.4, 1)$ which was expected to give the same performance as the default value for large values of `colsample_bytree`. By inspecting the source code of LightGBM it was realized that the model takes the integer of the column downsampling multiplied with the total number of features if the column downsampling is less than 1 [6]. This means that no matter how close to 1 the column downsampling get, the integer value of the total number of columns get floored to 2 at max, compared to when the column downsampling is exactly 1 which it only is for the default values.

To deal with this problem a new PDF was developed on top of the existing ones in Scipy, the truncated uniform PDF: $\mathcal{U}_{\text{trunc}}(a, b, c)$. This PDF first generates a random number x from a uniform distribution between a and c. Then if x is larger than b it is floored to b. In this way, it is possible to both get values of x in the interval $[a, b]$ but also values exactly equal to b. The value of c controls how often these "overflow" values of x are generated.

[14] LightGBM and XGBoost of course have chosen their default parameters smartly, however, it one would not expect them to outperform other sets of hyperparameters that clearly.

5.6 *g-Tagging Analysis*

The trained b-tagging LGB model is a jet-based model which provides a b-tag score β_{tag} to a jet. This also means that each of the jets e.g. a 4-jet event can get a b-tag: $\boldsymbol{\beta}_{\text{tag}} = [\beta_{\text{tag}_1}, \beta_{\text{tag}_2}, \beta_{\text{tag}_3}, \beta_{\text{tag}_4}]$. Using $\boldsymbol{\beta}_{\text{tag}}$ one can train a new model on the events, compared to individual jets, where signal events are defined to be q-matched events where the gluons are assigned the $n-2$ lowest b-tag scores for n-jet events; e.g. $\boldsymbol{\beta}_{\text{tag}} = [0.95, 0.89, 0.15, 0.07]^\top$ for the four jets

$[b, \bar{b}, g, g]$. This event-based process will be called g-tagging and the trained model will return a g-tag score written as γ_{tag}. Compared to the b-tagging LGB model, this model will allow one to extract entire events which contains a clear identification of gluons versus non-gluons.

5.6.1 *Permutation Invariance*

Since the b-tags are only based on the vertex variables, the goal of the g-tag is to also be constructed in an un-biased way with respect to the jet energy E_{jet}. However, even though $\beta_{\text{tag}} \perp\!\!\!\perp E_{\text{jet}}$ and $\gamma_{\text{tag}} = f(\beta_{\text{tag}})$, it turned out that $\gamma_{\text{tag}} \not\perp\!\!\!\perp E_{\text{jet}}$, where $a \perp\!\!\!\perp b$ is defined to mean that a is independent[15] of b and f is an unknown function. This was because the ordering of the jets within the event was energy-dependent: they sorted according to their E_{jet}.

[15] And $\not\perp\!\!\!\perp$ means not independent.

This meant that the different components in $\boldsymbol{\beta}_{\text{tag}}$ had different importances, even though they should be equally important. Instead of defining $\boldsymbol{\beta}_{\text{tag}}$ as a vector it should instead be seen as a set[16] $\boldsymbol{\beta}_{\text{tag}} = \{\beta_{\text{tag}_1}, \ldots, \beta_{\text{tag}_n}\}$. The g-tagging model trained on the events should thus be *permutation invariant*[17] with regards to the input variables. The category of permutation invariant (and equivariant[18]) neural networks in the deep learning community has seen an huge development within recent years where the paper from Zaheer et al. [83] in 2017 was quite influential, however also other examples exists [63, 40]. Yet, the same development cannot be said to have happened within the more classic machine learning field.

[16] Since sets have no inherent order.

[17] $f(\mathbf{x}) = f(\tau(\mathbf{x}))$ for any permutation τ on an input vector $\mathbf{x}$.

[18] $\tau(f(\mathbf{x})) = f(\tau(\mathbf{x}))$ for any permutation τ on an input vector $\mathbf{x}$.

Although not being a novel software-technical solution, the problem was circumvented by two simple, different approaches: 1) by simply shuffling the inputs variables independently for each observation (row) in the dataset, and 2) training on all possible permutations of the variables in the dataset. The second approach can be seen as a feature augmentation technique where the data is artificially increased with factor of n factorial: $N \to n! \cdot N$ where N is the number of observations (rows) and n is the number of jets. These two methods were tested along with the original order of the dataset.

5.6.2 *g-Tagging Hyperparameter Optimization*

Four LightGBM models, two for 3-jet events and two for 4-jet events, were trained and hyperparameter optimized for both the the energy ordered and shuffled[19] data sets with 100 iterations of random search with the same PDFs as for the b-tagging, see Table 5.4, and 5-fold cross validation and early stopping with a patience of 100. The results of the HPO can be seen in Figure 5.15. Here the two 3-jets models are seen in the two plots to the left, and the two 4-jets to the right. The very left plot shows the 3-jet energy-ordered (no permutation) performance as a function of iteration number, which was also where the issues mentioned in section 5.5 were first discovered. Here the difference between the how many

[19] The method with all permutations was trained using the same hyperparameters as the best ones for the shuffled model.

of the three variables, the three b-tags, are included is seen as three clear plateaus. The three plateaus are also seen in the 3-jet events that were shuffled, however, with more variation in each plateau, along with a drop in performance. For the 4-jet events the plateaus are not as apparent but it can still be seen how some of the iterations how a significantly lower score than others. The parallel plots for the four fits can be seen in Figure B.7–B.10 in the appendix.

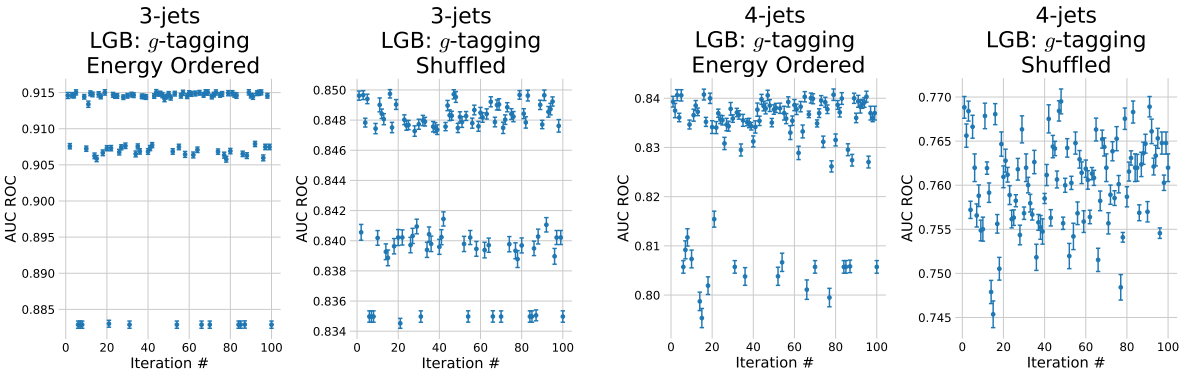


Figure 5.15: Hyperparameter Optimization results of g-tagging with 100 iterations of random search with LGB. From left to right, we have A) 3-jet events energy-ordered (no permutations), B) 3-jet events row-shuffled, C) 4-jet events energy-ordered, D) 4-jet events row-shuffled. Notice the different ranges on the y-axes.

5.6.3 *PermNet*

In addition to the LGB models, a permutation invariant neural network called PermNet based on the Deep Sets paper [83] implemented in Tensorflow [10] by Faye [36] was also tested. Zaheer et al. [83] showed that $f(X)$ is permutation invariant if and only if it can be decomposed in the following way:

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right). \tag{5.5}$$

for suitable transformations ρ and ϕ (which the neural network learns[20]). The PermNet was trained using three layers[21] with leaky ReLU [54] as the activation function and ADAM [49] as the optimizer optimizing the log-loss. The network was trained with early stopping with a patience of 50 epochs and a batch size of 128. A visual overview of the PermNet architecture can be seen in Figure B.11 in the appendix.

[20] This is possible since neural networks are universal function approximators [44].

[21] Where the two hidden layers have 128 and 64 neurons in each.

5.6.4 *1D Comparison of LGB and PermNet*

To better understand the difference between the difference between the LGB and PermNet models, a small comparison was made. This comparison was constructed by summing the b-tag scores in the n-jet event together $\sum_i^n \beta_{\text{tag}_i}$. The β_{tag_i} are summed together since this turns the problem into a 1D problem that is easy to visualize, the sum of numbers is a permutation invariant function, and is similar to the simplest functions of ρ and ϕ in equation (5.5): the identity function. The 1D models are fit to the training events and

then a linear scan from $\sum_i^n \beta_{\text{tag}_i} = 0.4$ to 3.1 is made to see how the predicted g-tags γ_{tag} distribute. This is shown in Figure 5.16 for 4-jet events. Here the value of γ_{tag} is shown for the two models together with the fraction of signal to background in each bin. If the g-tag score should resemble a true probability it would be expected to follow the signal ratio, e.g. a model should predict $\gamma_{\text{tag}} = 0.9$ if there is 90 % signal in that bin. In the figure it is seen how the PermNet does a great job at fitting the signal fraction, however, the LGB model also does a decent job. Remember that none of these models were shown the signal fraction explicitly, only the b-tag sum and a signal-or-background label. The distribution of signal and background together with the distribution of cuts made by the LGB model can be seen in Figure B.12 in the appendix. The similar plots for 3-jet events are plotted in Figure B.13 and B.14, both in the appendix.

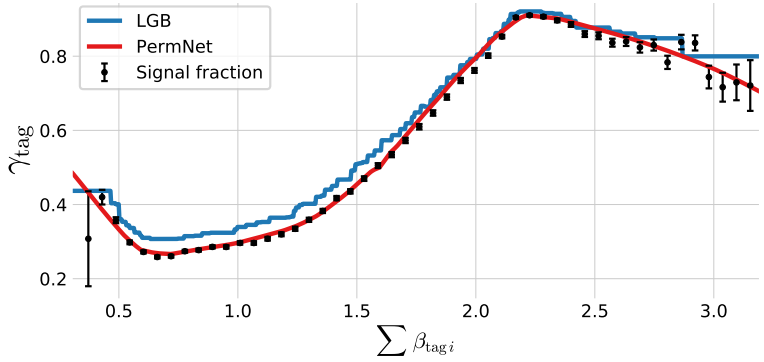


Figure 5.16: Plot of the (1D) g-tag scores for 4-jet events as a function of $\sum \beta_i$ for the LGB model in blue and the PermNet model in red. The signal fraction (based on the signal and background histograms in Figure B.12) is plotted as black error bars where the size of the error bars is based on the propagated uncertainties of the signal and background histogram assuming Poissonian statistics.

It can be concluded, at least in 1D, that both LGB and PermNet are able to capture the inherent structure in the (1D) data.

5.6.5 *g-Tagging Results*

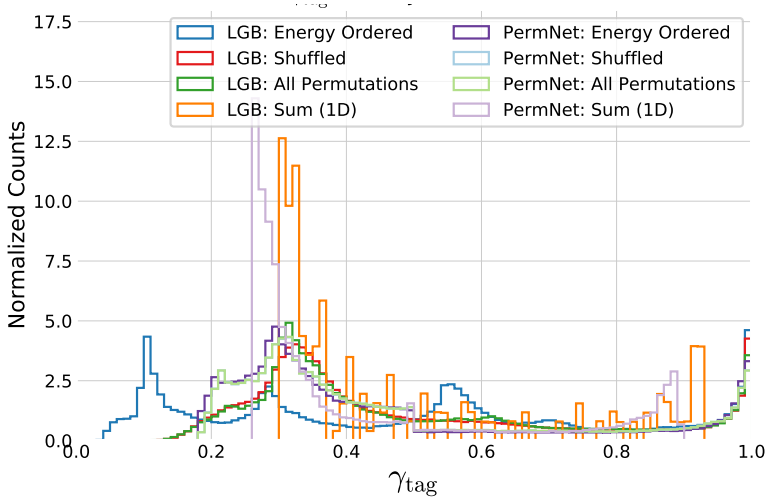


Figure 5.17: Histogram of g-tag scores (model prediction) in 4-jet events for XGB: Energy Ordered in blue, XGB: Shuffled in red, XGB: All Permutations in green, XGB: Sum 1D in orange, PermNet: Energy Ordered in purple, PermNet: Shuffled in light-blue, PermNet: All Permutations in light-green, PermNet: 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.

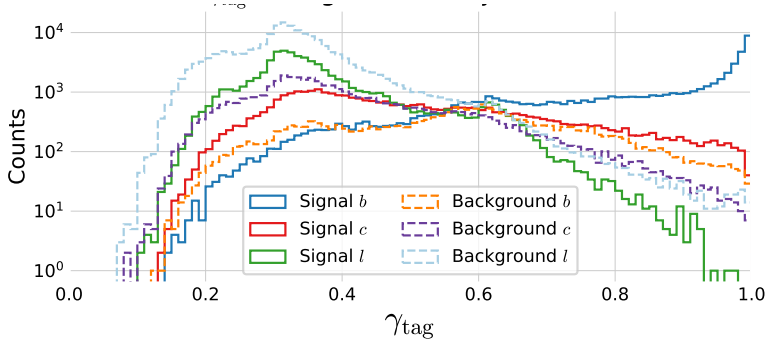


Figure 5.18: Histogram of g-tag scores (model prediction) from the XGB-model in 4-jet events for b signal in blue, c signal in red, l signal in green, b background in orange, c background in purple, l background in light-blue.

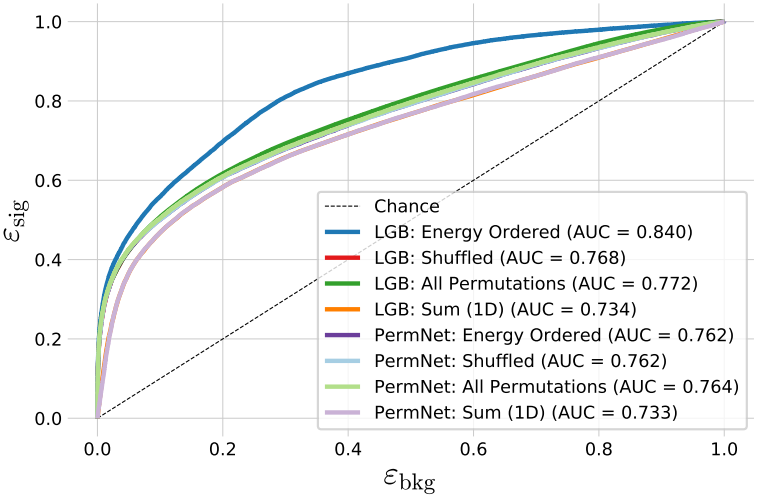


Figure 5.19: ROC curve of the eight g-tag models in 4-jet events. First one in dashed black is the ROC curve that you get by random chance. The colors are the same as in Figure 5.17 and in the legend also the Area Under the ROC curve (AUC) is shown. Notice that the XGB model which uses the energy ordered 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.

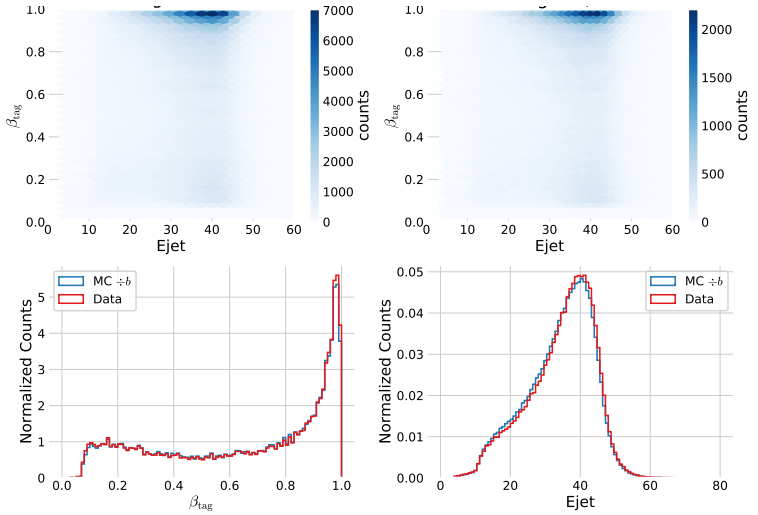


Figure 5.20: Comparison of the b-tag and jet energy (`Ejet`) distributions for Monte Carlo (MC) versus data. In the top row the 2D-distributions are shown for MC on the left (without the extra MCb samples) and data on the right. In the bottom row the 1D marginal distrubtions are shown for the b-tag and the jet energy with data in red and Monte Carlo ones in blue. Notice the the almost identical distributions in b-tag.

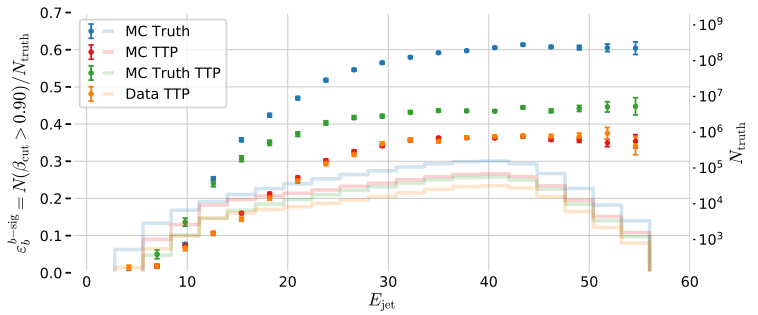


Figure 5.21: Efficiency of the b-tags for b-jets in the b-signal region for 3-jet events, $\varepsilon_b^{b-\text{sig}}$, as a function of jet energy `Ejet` . The b-signal region is defined as $\beta > 0.9$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in green, 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. Notice how both MC TTP and Data TTP follow each other closely.

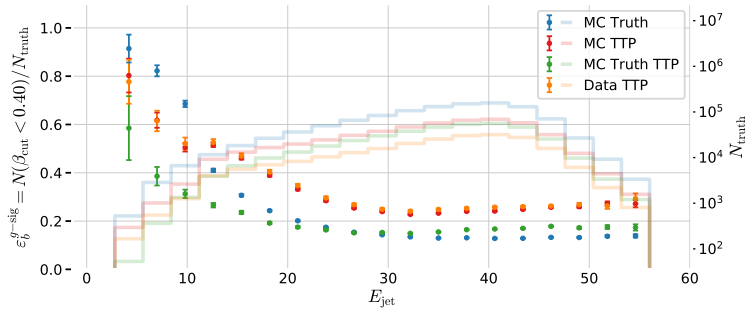


Figure 5.22: Efficiency of the b-tags for b-jets in the g-signal region for 3-jet events, $\varepsilon_b^{g-\mathrm{sig}}$, as a function of jet energy `Ejet`. The g-signal region is defined as $\beta < 0.4$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in 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.

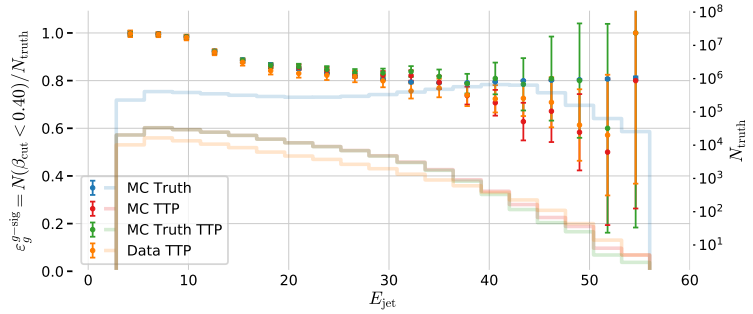


Figure 5.23: Efficiency of the b-tags for g-jets in the g-signal region for 3-jet events, $\varepsilon_g^{g-\mathrm{sig}}$, as a function of jet energy `Ejet`. The g-signal region is defined as $\beta < 0.4$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in 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.

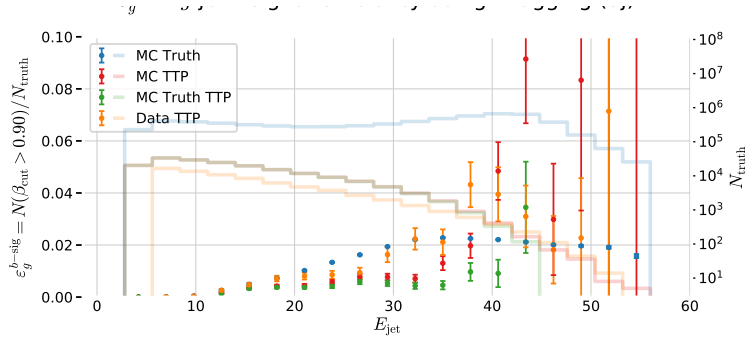


Figure 5.24: Efficiency of the b-tags for g-jets in the b-signal region for 3-jet events, $\varepsilon_g^{b-\mathrm{sig}}$, as a function of jet energy `Ejet`. The b-signal region is defined as $\beta > 0.9$. In the plot the efficiencies are shown for MC Truth in blue, MC TTP in red, MC Truth TTP in 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.

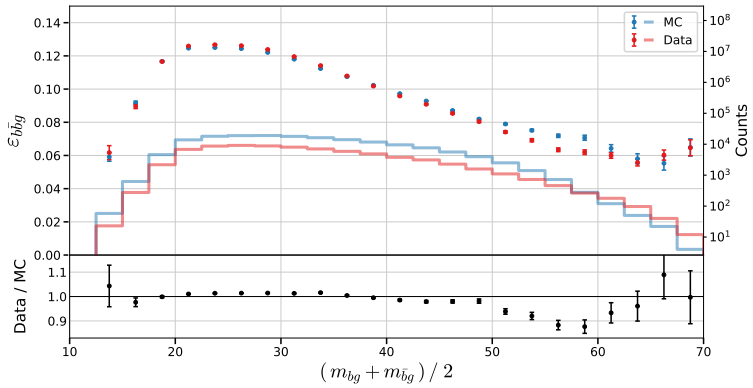


Figure 5.25: Proxy efficiency of the g-tags for $b\bar{b}g$ 3-jet events as a function of the mean of the two invariant masses m_{bg} and $m_{\bar{b}g}$. The proxy efficiency $\varepsilon_{b\bar{b}g}$ is measured by finding $b\bar{b}g$-events where $\beta_b > 0.9$, $\beta_{\bar{b}} > 0.9$, and $\beta_g < 0.4$. and then calculating $\varepsilon_{b\bar{b}g} = \varepsilon_b^{b-\mathrm{sig}} \cdot \varepsilon_{\bar{b}}^{b-\mathrm{sig}} \cdot \varepsilon_g^{g-\mathrm{sig}}$. In the top plot $\varepsilon_{b\bar{b}g}$ 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.

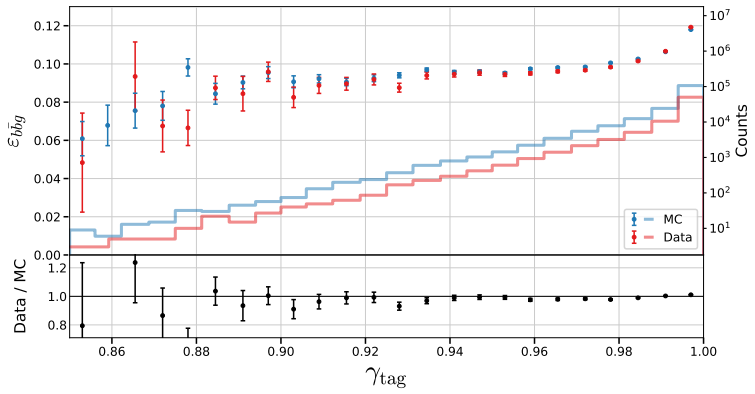


Figure 5.26: Proxy efficiency of the g-tags for $b\bar{b}g$ 3-jet events as a function of the event's g-tag. The proxy efficiency $\varepsilon_{b\bar{b}g}$ is measured by finding $b\bar{b}g$-events where $\beta_b > 0.9$, $\beta_{\bar{b}} > 0.9$, and $\beta_g < 0.4$. and then calculating $\varepsilon_{b\bar{b}g} = \varepsilon_b^{b-\text{sig}} \cdot \varepsilon_{\bar{b}}^{b-\text{sig}} \cdot \varepsilon_g^{g-\text{sig}}$. In the top plot $\varepsilon_{b\bar{b}g}$ 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.

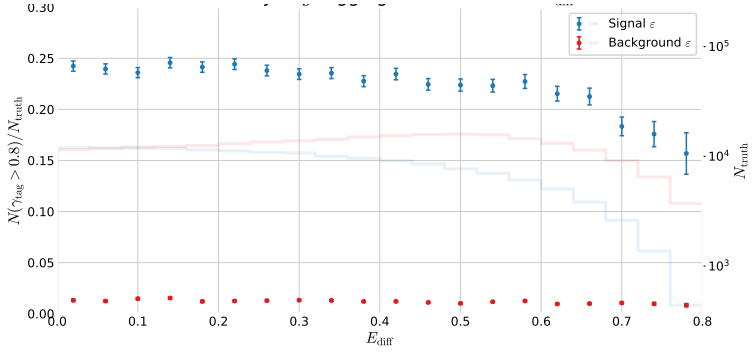


Figure 5.27: Efficiency of the g-tags for 4-jet events as a function of normalized gluon gluon jet energy difference in Monte Carlo. The efficiency is measured as the number of events with a g-tag higher than 0.8 ($\gamma > 0.8$) out of the total number and the normalized gluon gluon jet energy difference A is $A = \frac{E_{g\text{max}} - E_{g\text{min}}}{E_{g\text{max}} + E_{g\text{min}}}$ where $E_{g\text{max}}$ ($E_{g\text{min}}$) 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.

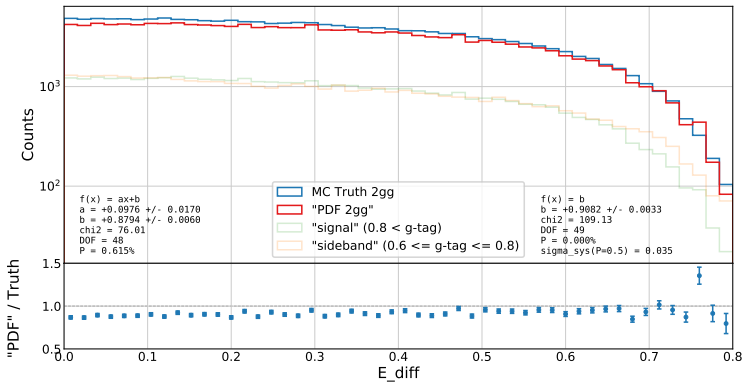


Figure 5.28: Closure plot between MC Truth and the corrected g-tagging model in 4-jet events for the normalized gluon gluon jet energy difference. The corrected g-taggingg model is described in further detail in section XXX **TODO!**. In the top part of the plot the MC Truth is shown in blue, the corrected 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 $A = \frac{E_{g\text{max}} - E_{g\text{min}}}{E_{g\text{max}} + E_{g\text{min}}}$ where $E_{g\text{max}}$ ($E_{g\text{min}}$) refers to the energy of the gluon with the highest (lowest) energy.

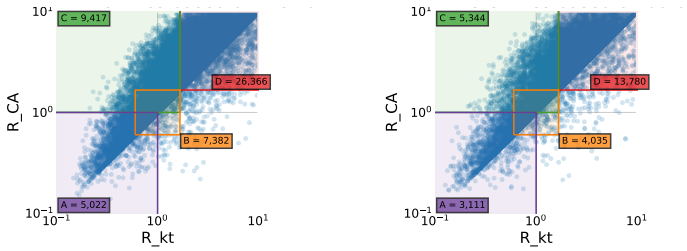


Figure 5.29: R_kt CA overview XXX **TODO!**

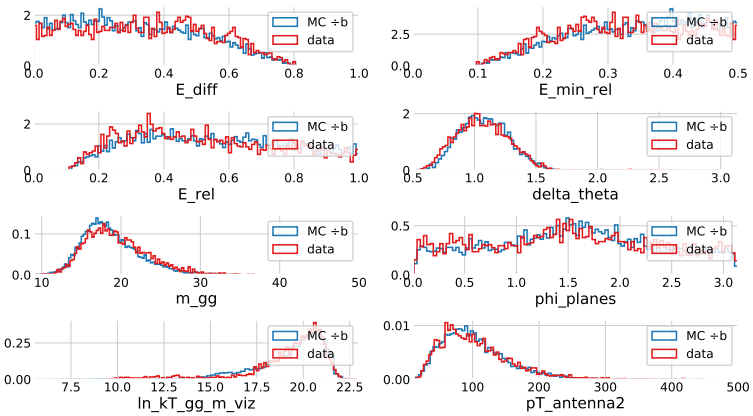


Figure 5.30: R kt CA cut region A XXX **TODO!**

6. *Discussion and Outlook*

The Tufte-LaTeX 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-LaTeX document classes to better fit your needs.

7. Conclusion

7.1 Tufte-LaTeX Website

The website for the Tufte-LaTeX 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.

7.2 Tufte-LaTeX Mailing Lists

There are two mailing lists for the Tufte-LaTeX 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-LaTeX 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.

7.3 Getting Help

If you've encountered a problem with one of the Tufte-LaTeX 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.

A. Housing Prices Appendix

Figure A.1: XXXX **TODO!**.

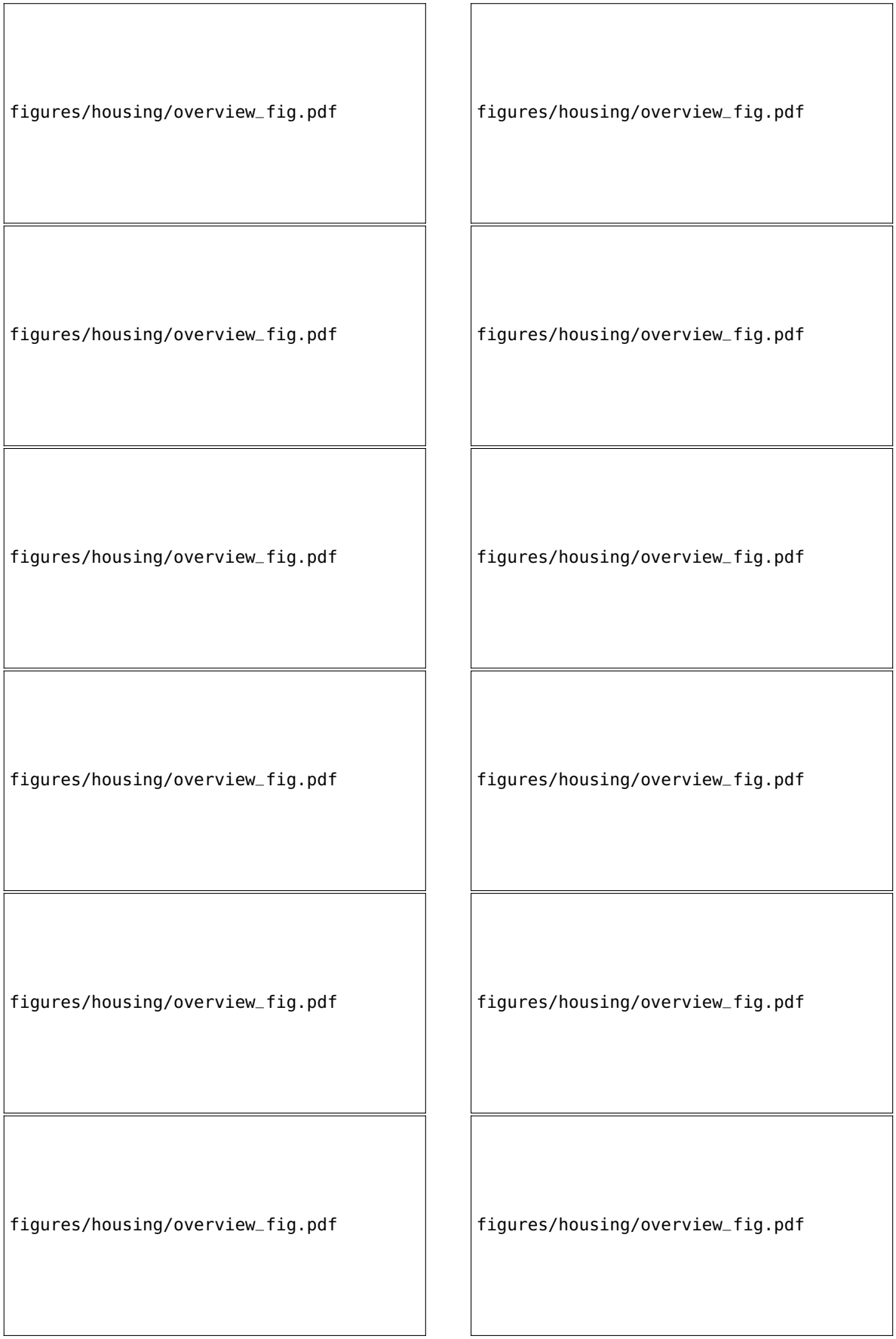


Figure A.2: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

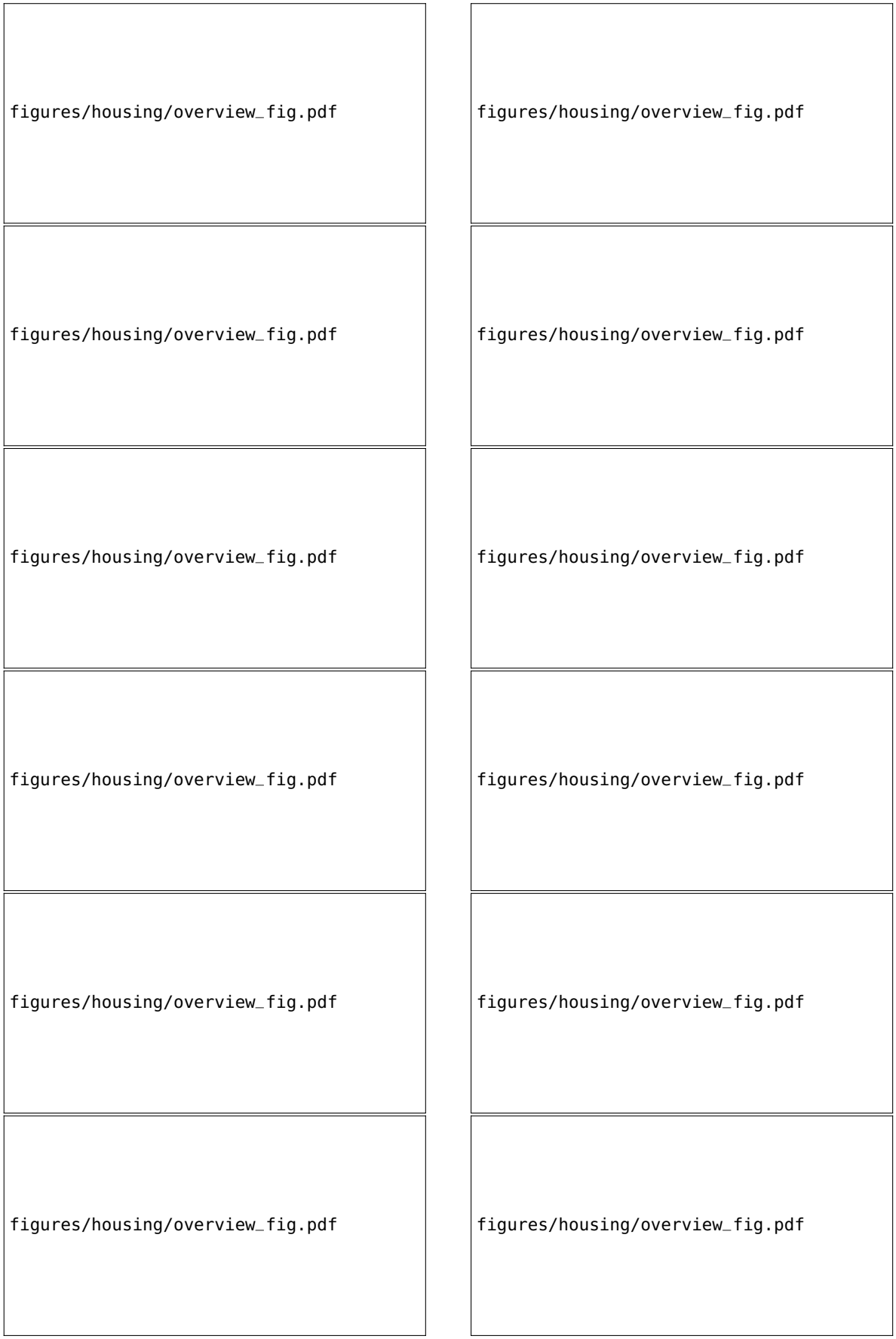


Figure A.3: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

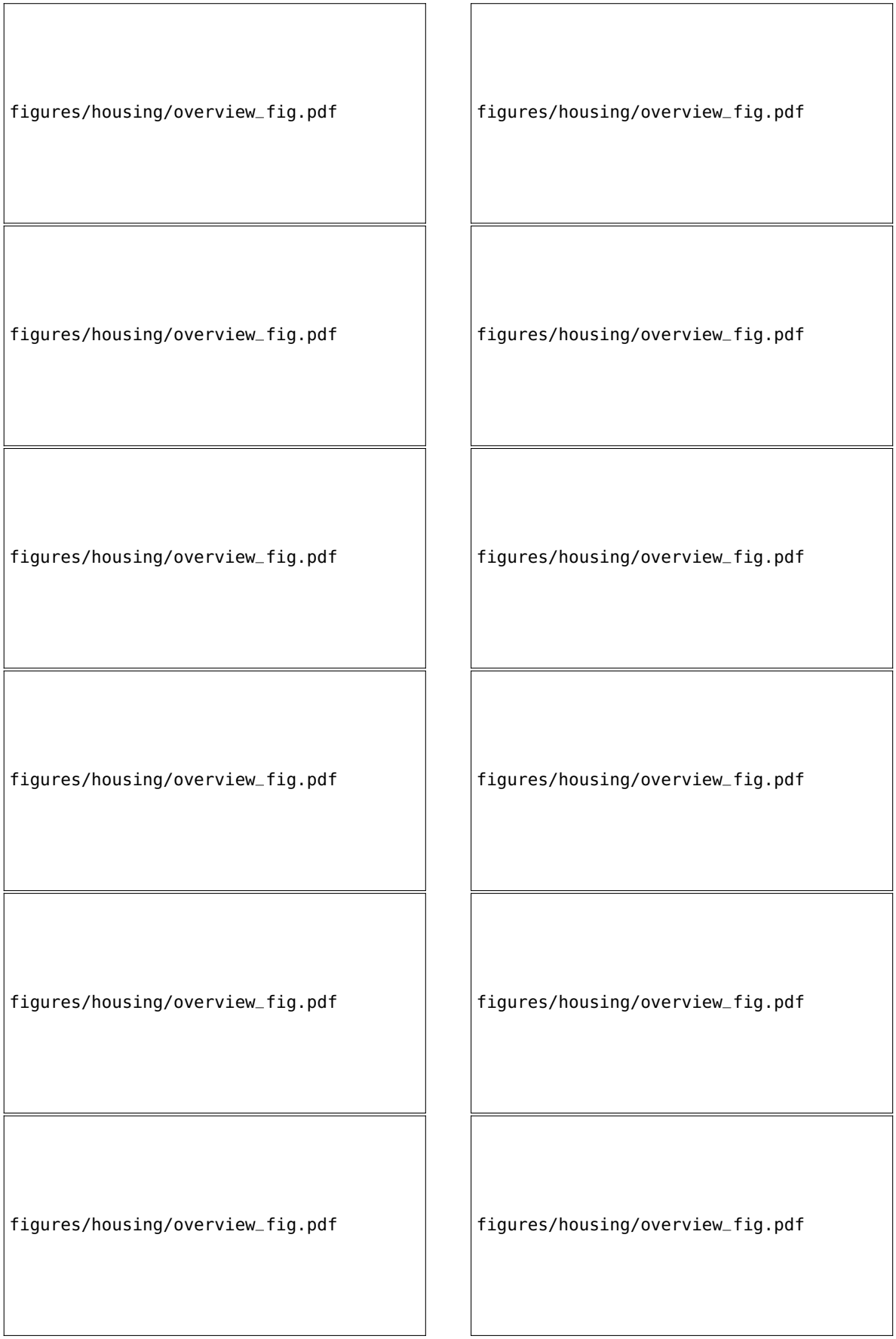


Figure A.4: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

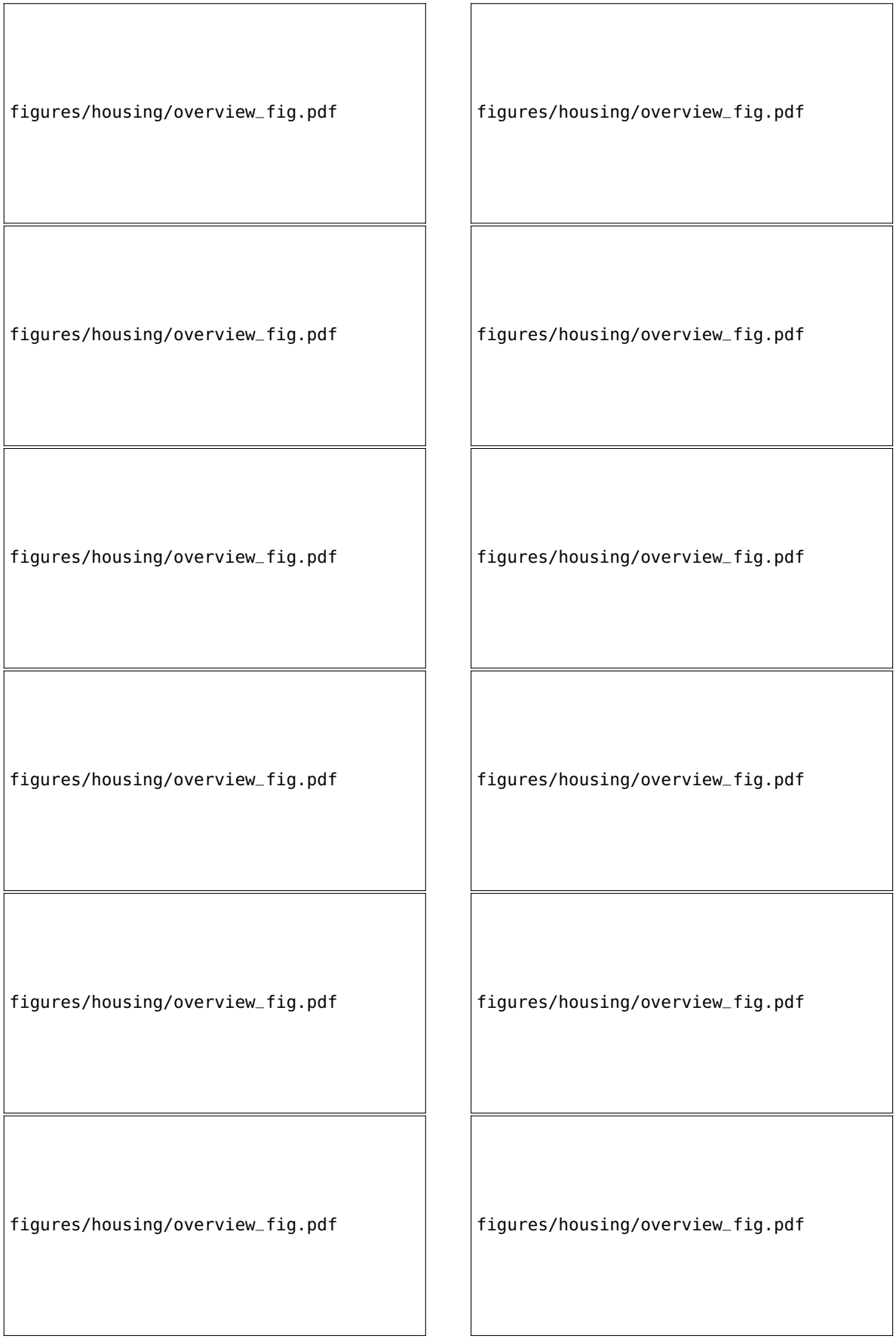


Figure A.5: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

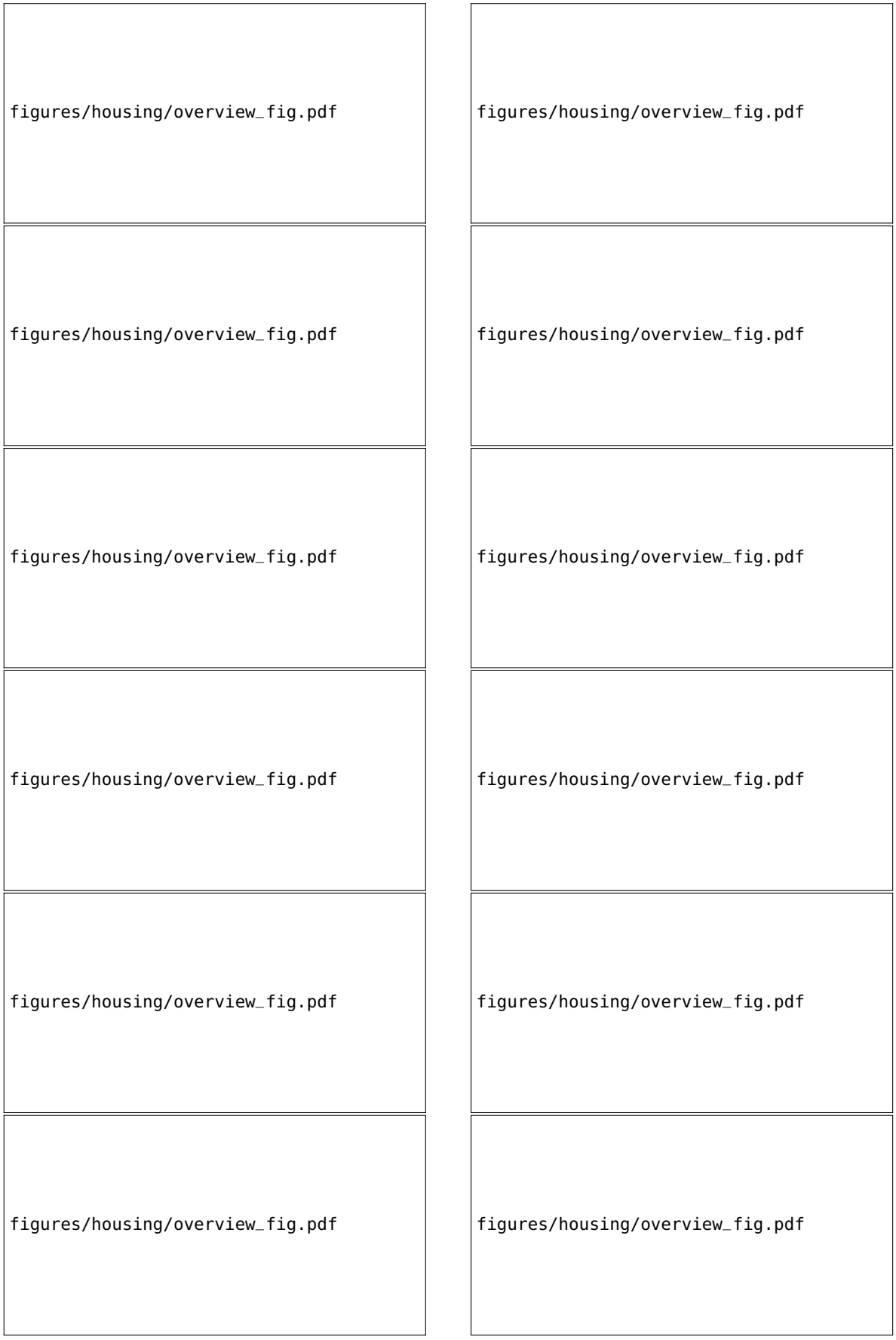


Figure A.6: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

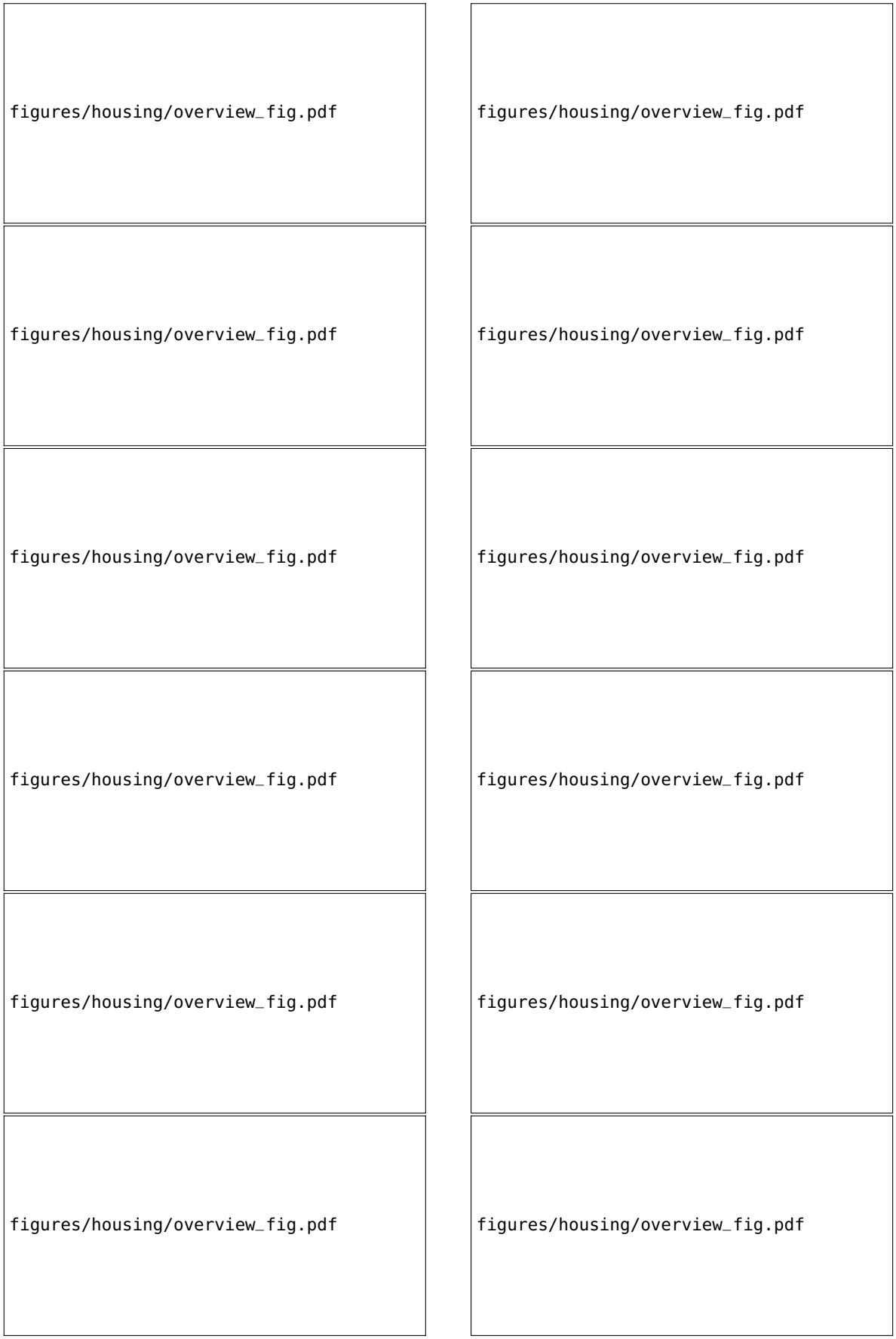


Figure A.7: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

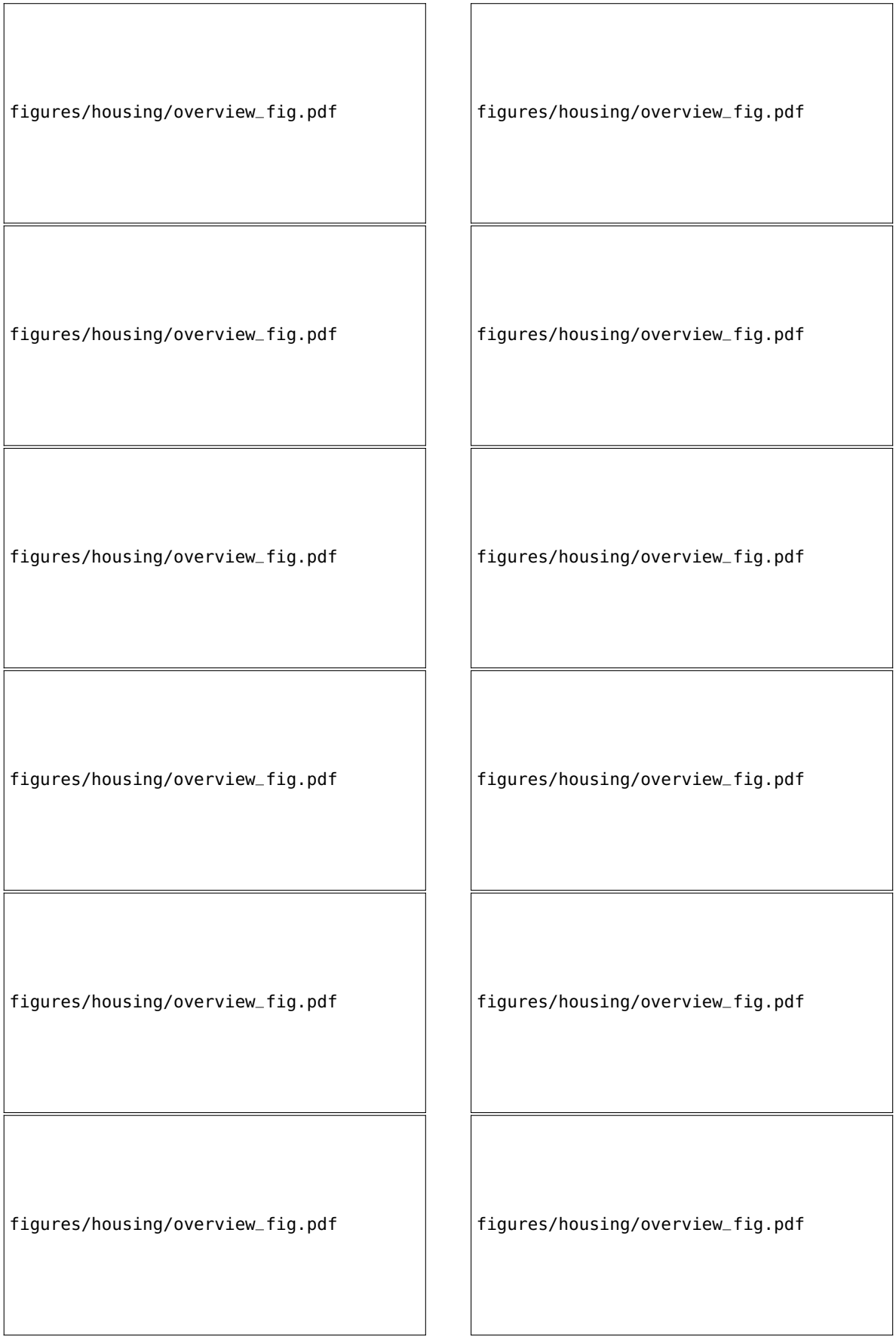


Figure A.8: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

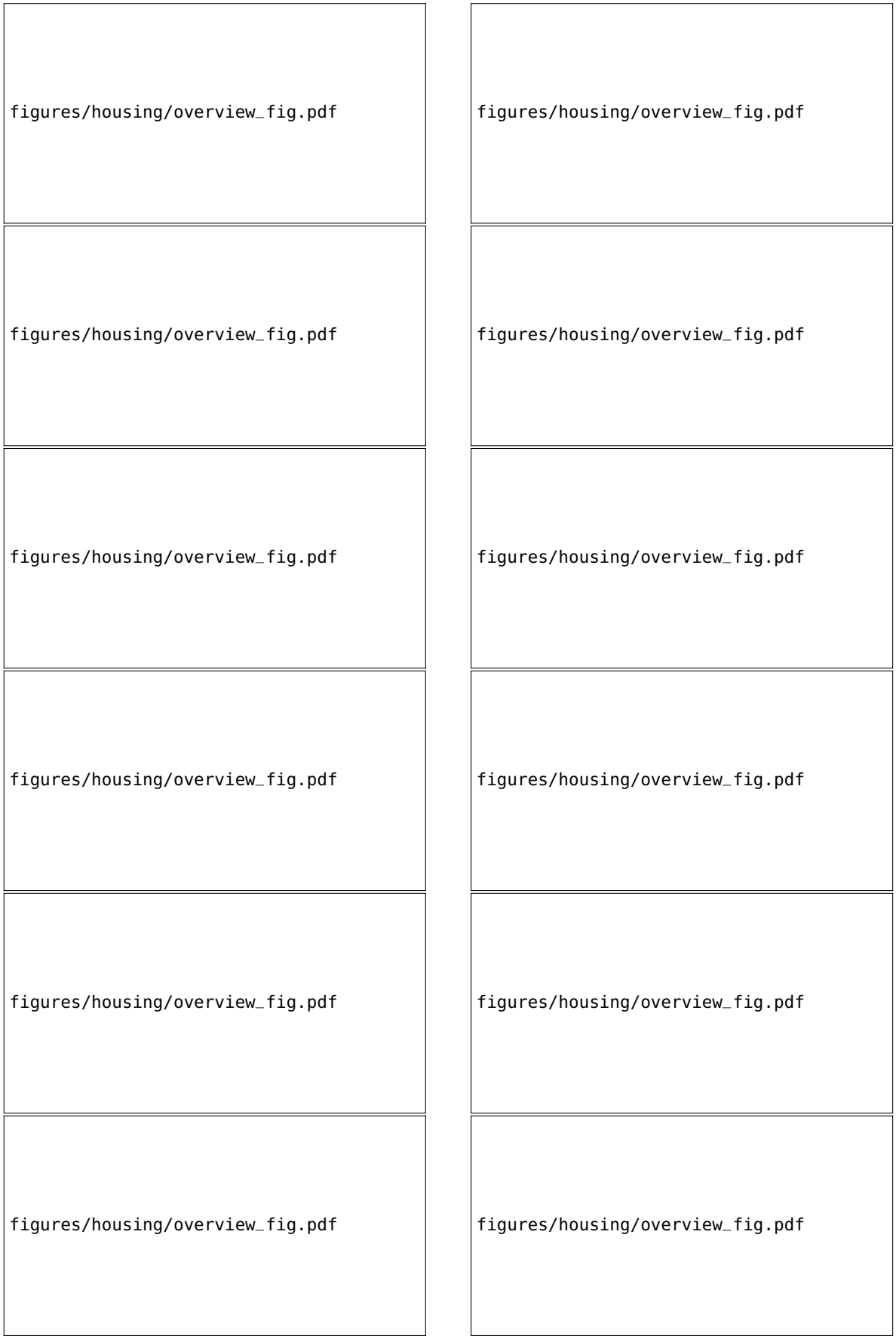


Figure A.9: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

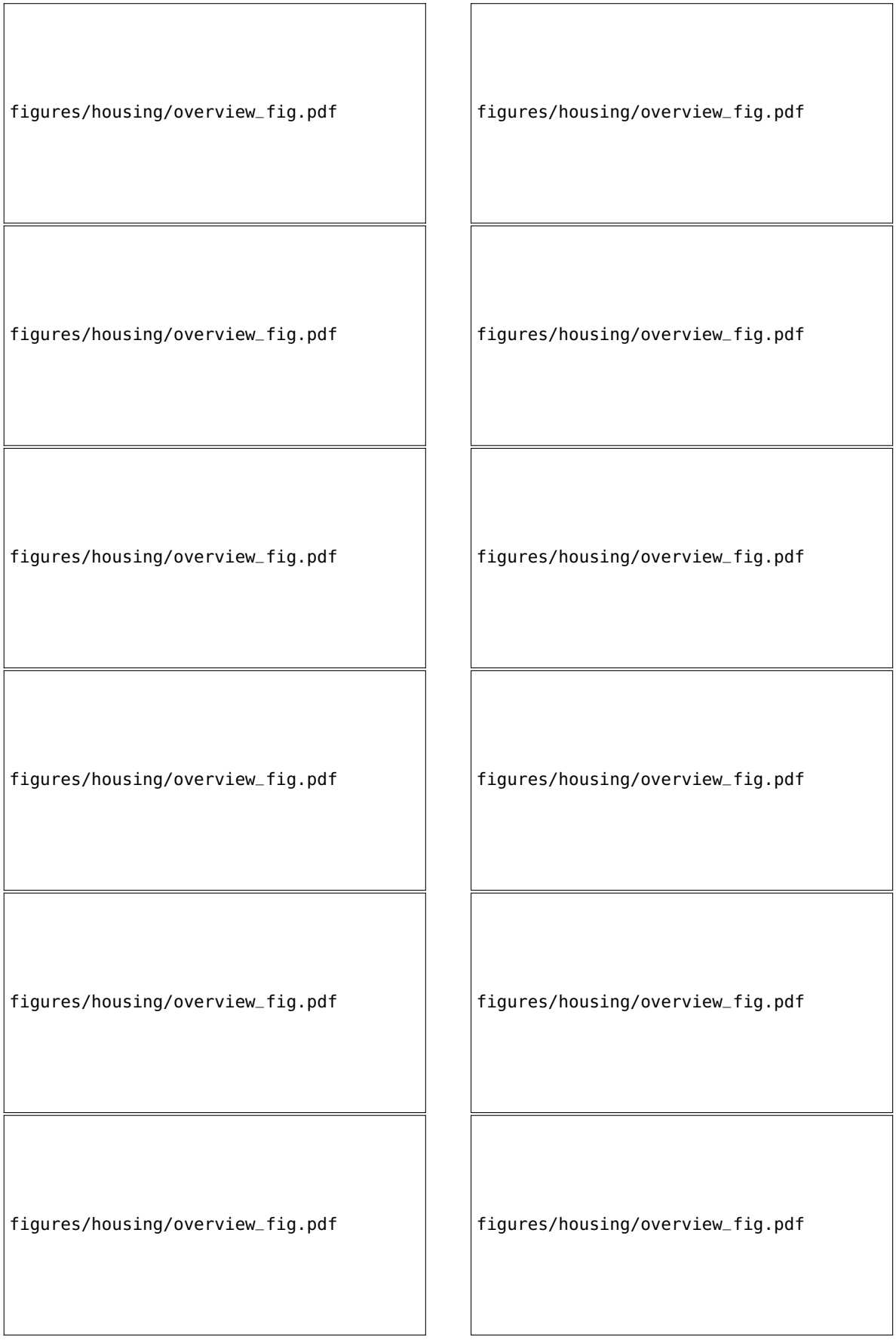


Figure A.10: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

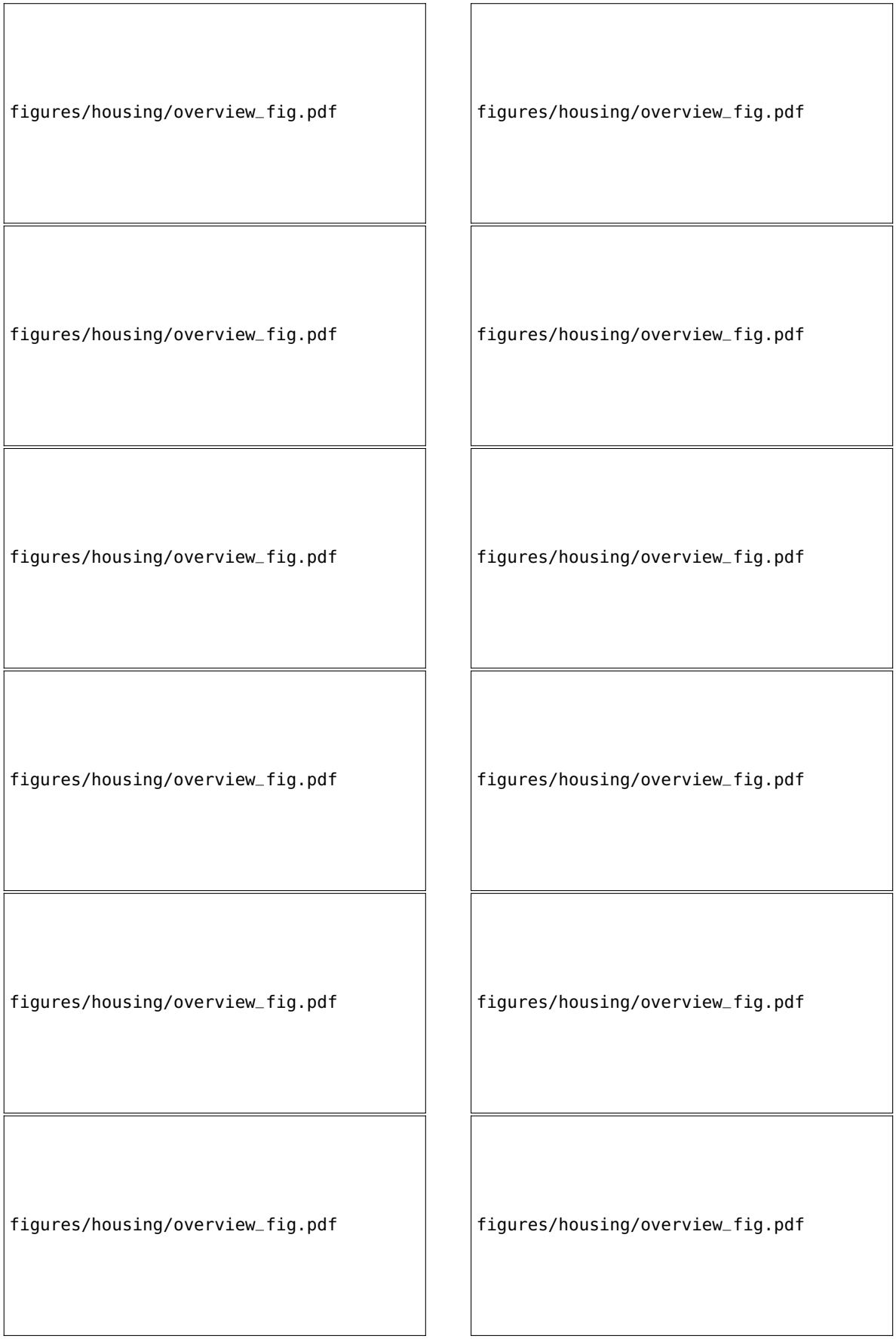


Figure A.11: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

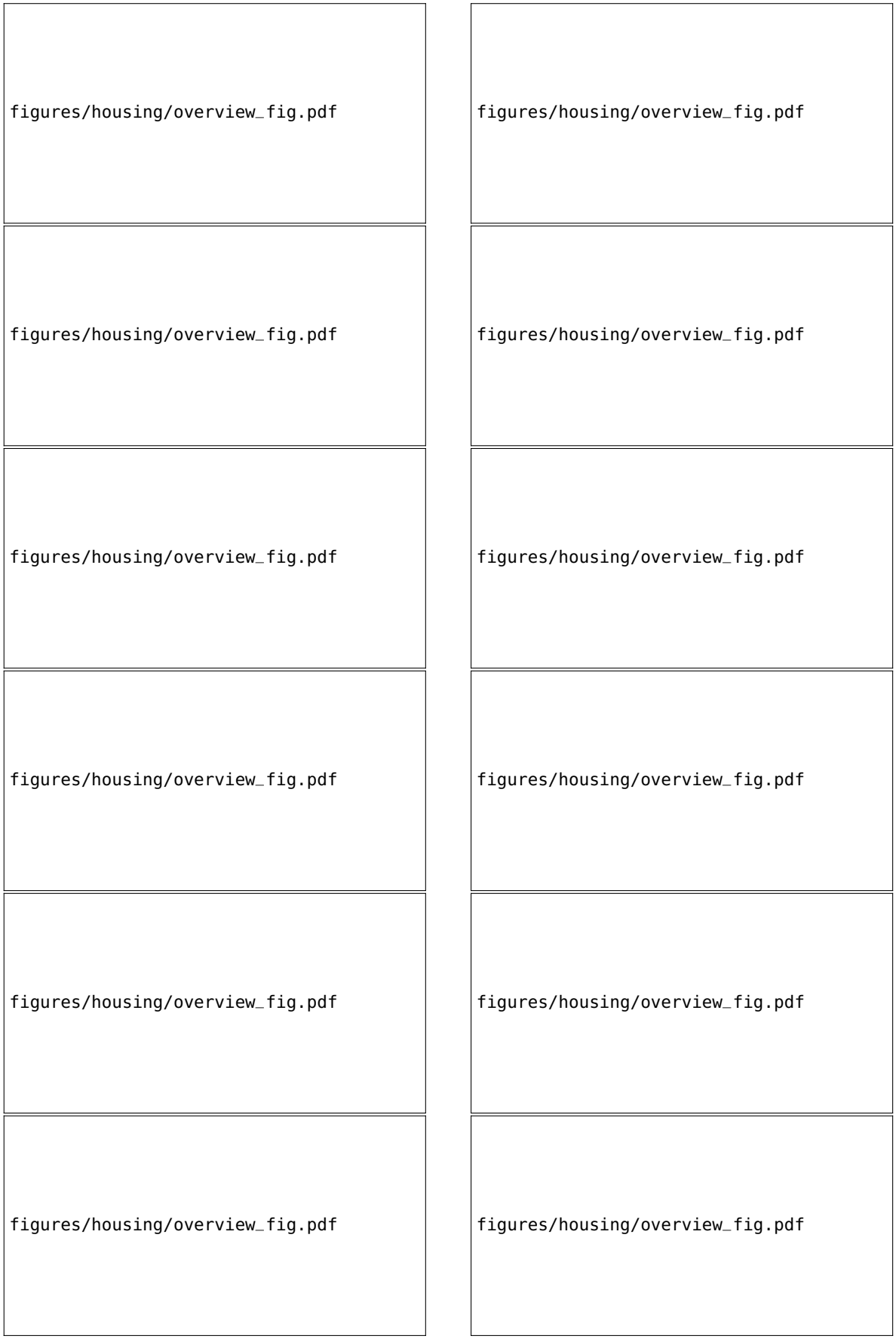


Figure A.12: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

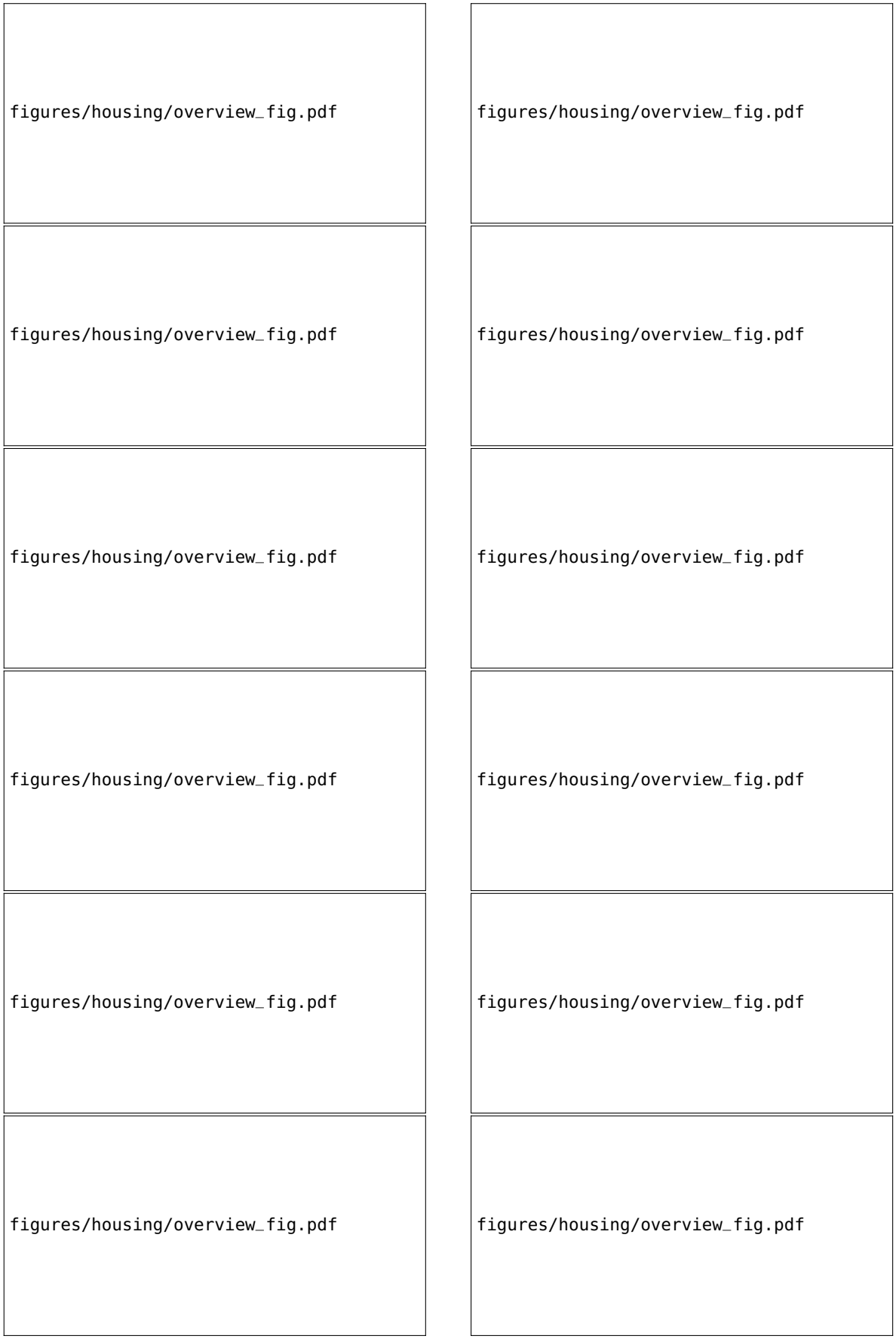


Figure A.13: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

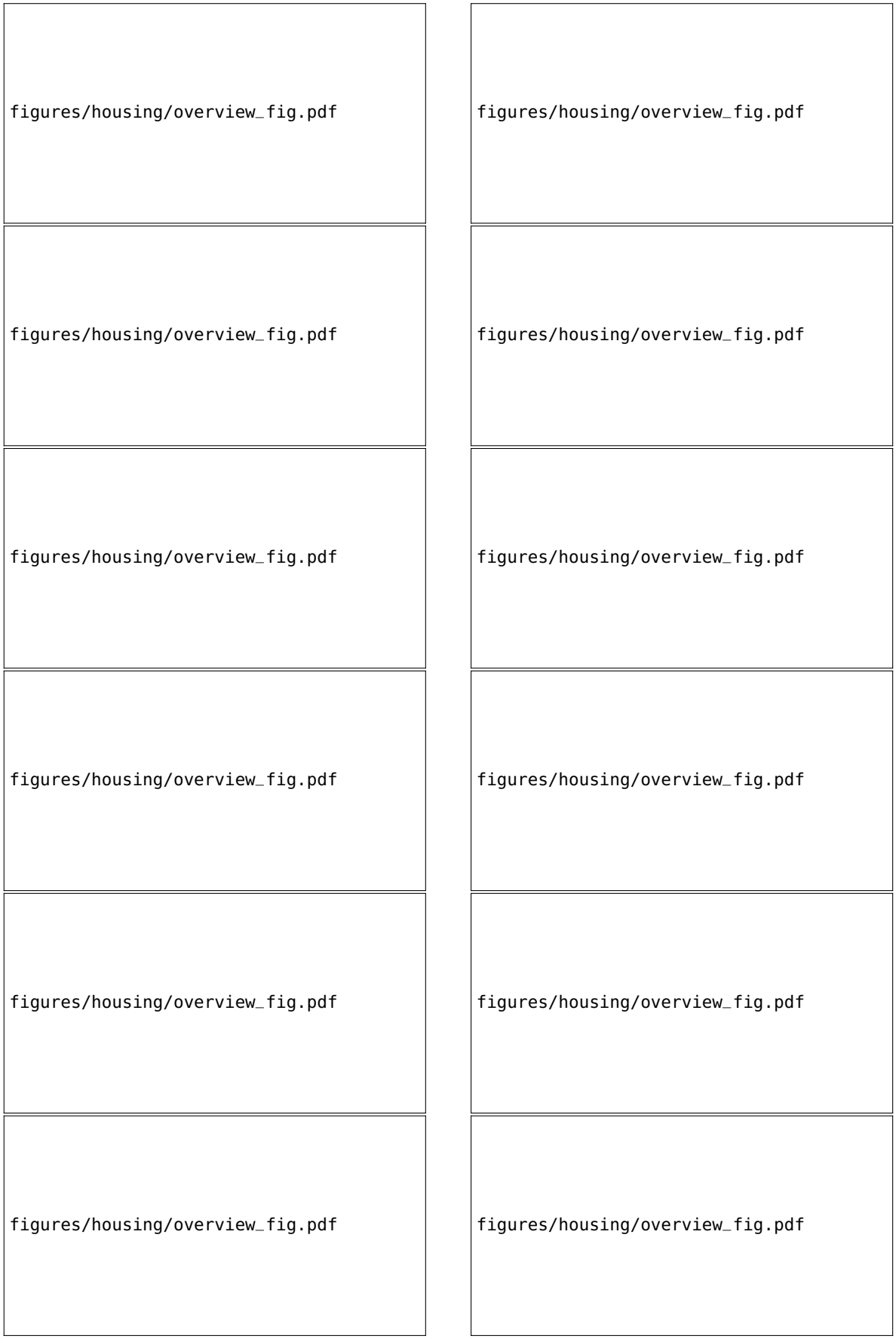


Figure A.14: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

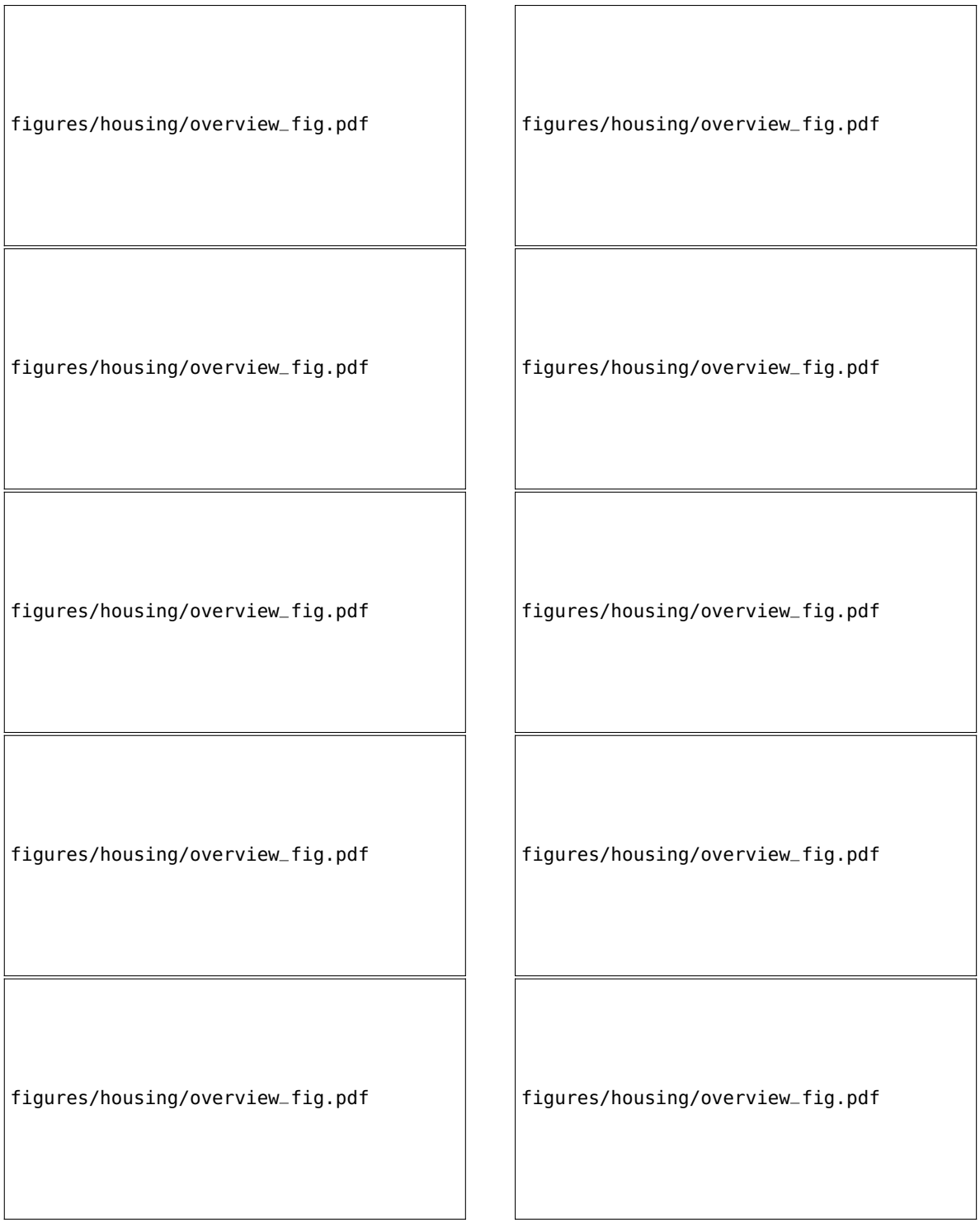


Figure A.15: Distributions the 168 input variables (excluding `ID` and `Vejnavn`).

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_HerafAffaldsrum
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_Rigsgraense
Afstand_Sportsanlaeg	Afstand_Strand	Afstand_Vindmoelle
Kommune_Indbyggertal	Kommune_SkatteProcent	Kommune_Vuggestuer
Kommune_Boernehaver	Kommune_IntegreredeInstitutioner	Kommune_FolkeSkoler
Kommune_Grundskyld	dag_i_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!**.

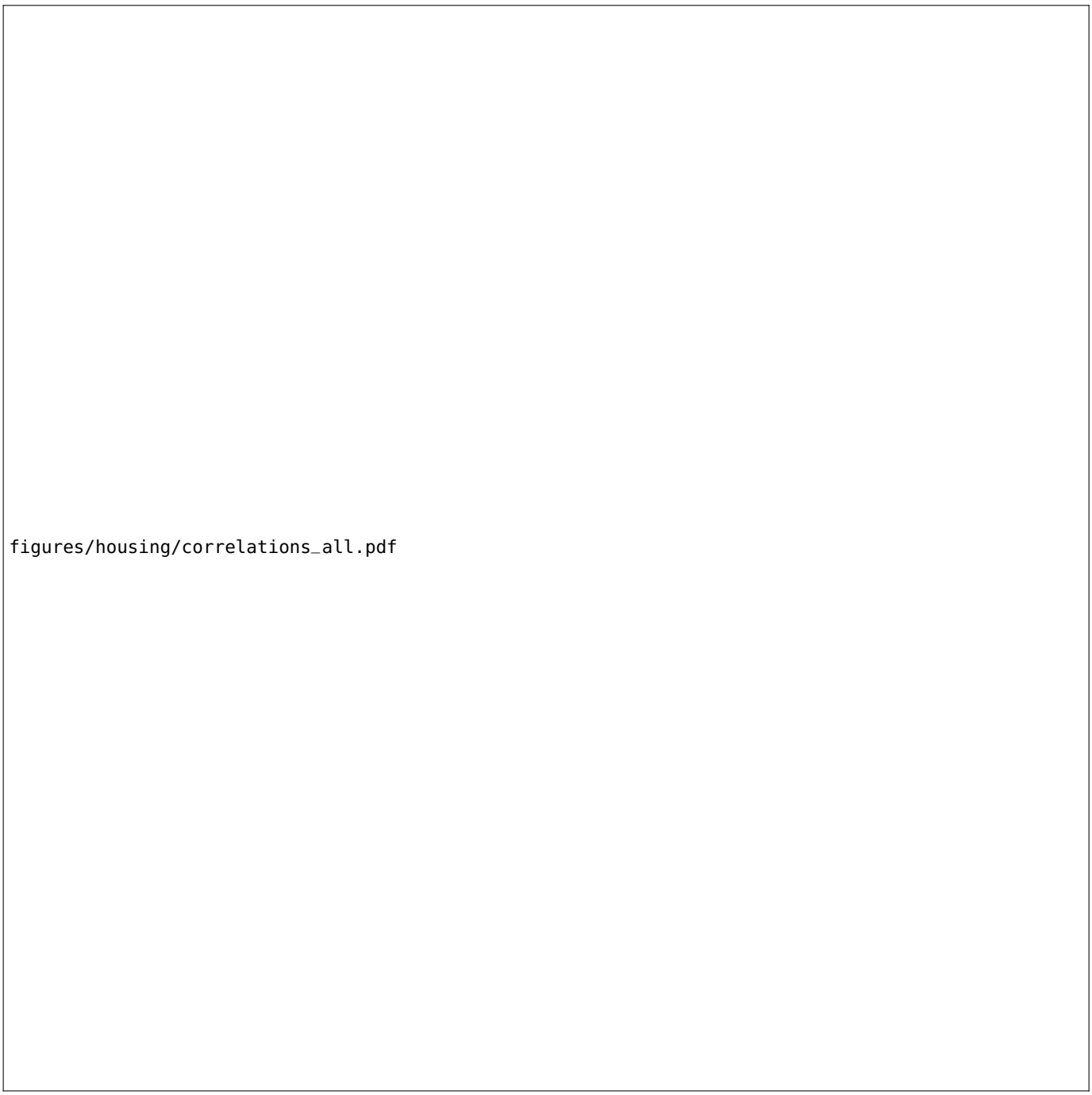


Figure A.16: XXXX **TODO!**.

Figure A.17: MIC non-linear correlation.

Energy rating label	Code
A2020	2
A2015	4
A2010	6
A2	8
A1	10
A	12
B	20
C	30
D	40
E	50
F2	60
F1	62
F	64
G2	70
G1	72
G	74
H, I, J, K, M	90
NAN	100

Table A.2: Energy rating mapping. If the energy rating is e.g. "A2" this gets the code 8.

figures/housing/Villa_v18_cut_all_Ncols_all_prophet_forecast.png

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 in 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.

figures/housing/Villa_v18_cut_all_Ncols_all_prophet_trends.pdf

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 ($< 10\%$).

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.3: 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.4: 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.5: 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.6: 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.7: 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.8: 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.9: 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.10: 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.11: 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.12: Fair-villa-appendix.

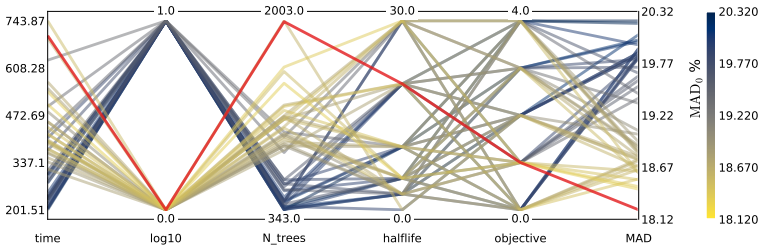


Figure A.20: Hyperparameter optimization results of the housing model for houses. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance 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 hyperparamater `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.

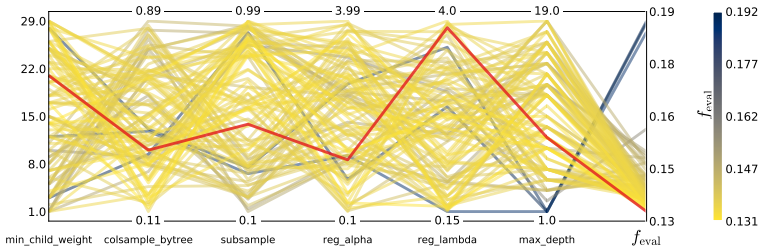


Figure A.21: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

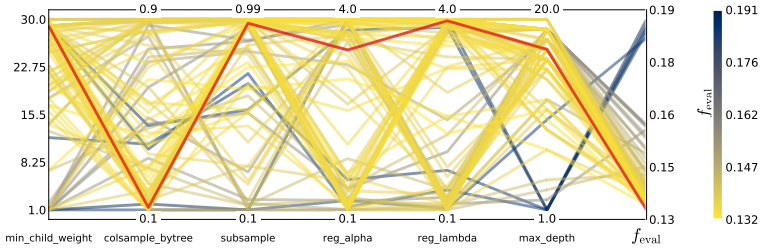


Figure A.22: Hyperparameter optimization results of XGBoost parameters of the housing model for apartments shown as parallel coordinates. Here shown for Bayesian optimization as hyperparameter optimization.

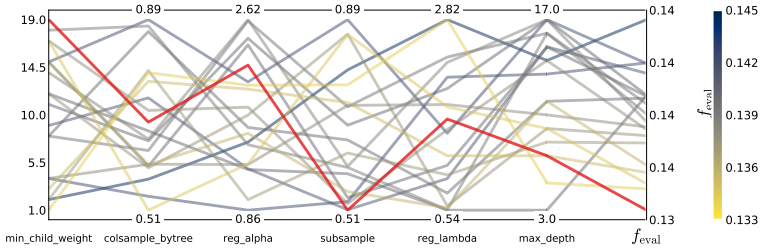


Figure A.23: Hyperparameter optimization results of XGBoost parameters of the housing model for houses shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

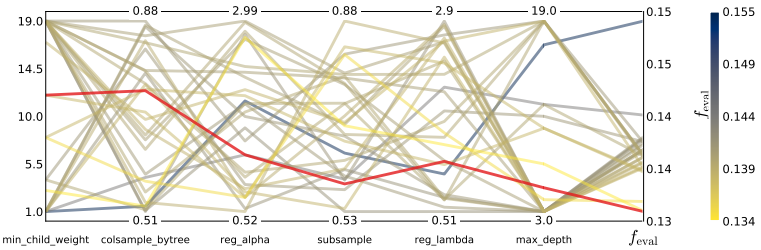


Figure A.24: Hyperparameter optimization results of XGBoost parameters of the housing model for houses shown as parallel coordinates. Here shown for Bayesian optimization as hyperparameter optimization.

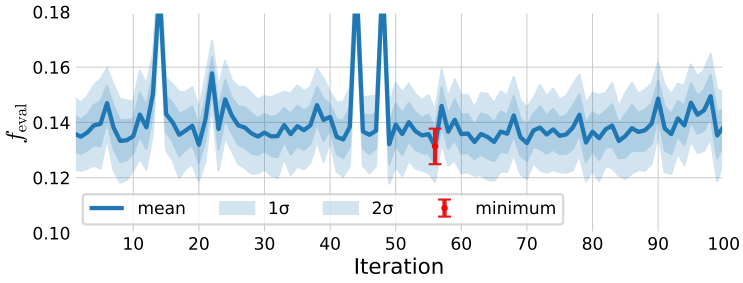


Figure A.25: XXX of the housing model for apartments shown as parallel coordinates. Here shown for random search as hyperparameter optimization.

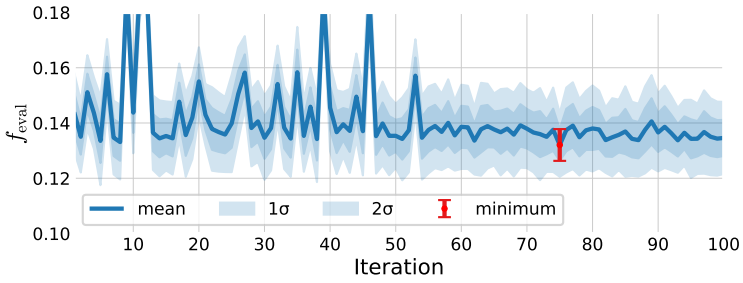


Figure A.26: XXX of the housing model for apartments shown as parallel coordinates. Here shown for Bayesian optimization as hyperparameter optimization.

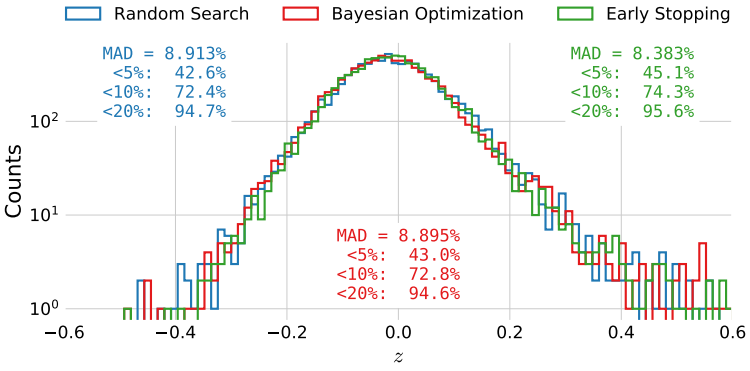


Figure A.27: Histogram of z-values of the XGB-model trained on apartments. The performance after hyperparameter optimization (HPO) using Random Search (RS) is shown in blue, for Bayesian Optimization (BO) in red. After finding the best model, BO in this case, the model is retrained using early stopping, 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.13: 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.14: XXX villa tight

B. Quarks vs. Gluons Appendix

	b	*c*	*uds*	*g*	non-*q*-matched
2	37.2 %	12.9 %	29.1 %	0.0 %	20.7 %
3	22.6 %	8.9 %	19.7 %	31.2 %	17.5 %
4	14.6 %	7.0 %	15.0 %	45.1 %	18.3 %
5	10.0 %	5.7 %	12.2 %	52.5 %	19.6 %
6	7.1 %	4.4 %	8.8 %	54.4 %	25.2 %

Table B.1: Number of different types of jets for MC and MCb written in relative numbers such that each row sum to 100 %. See also Table 5.3.

Figure B.1: Grid search of the two parameters `n_neighbors` and `min_dist` for the UMAP algorithm run on 4-jet events. For an explanation of these, see section 5.2.

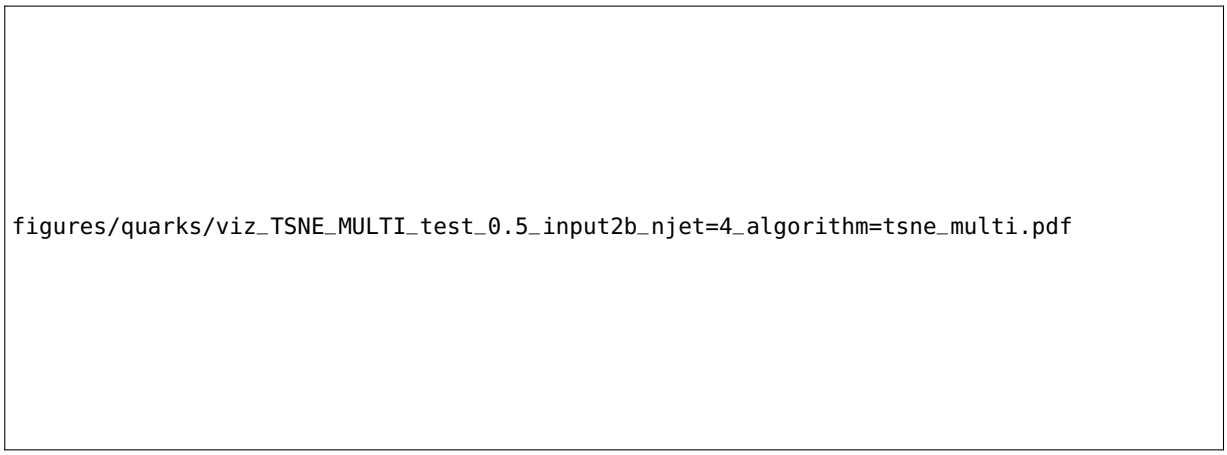


Figure B.2: Visualization of the t-SNE algorithm as a function of the `perplexity` parameters for 4-jet events.

Hyperparameter	Range
`subsample`	$\mathcal{U}(0.4, 1)$
`colsample_bytree`	$\mathcal{U}_{\text{trunc}}(0.4, 1, 2)$
`max_depth`	$\mathcal{U}_{\text{int}}(1, 20)$
`min_child_weight`	$\mathcal{U}_{\text{int}}(0, 10)$

Table B.2: Probability Density Functions for the random search hyperparameter optimization process for the XGBoost model.

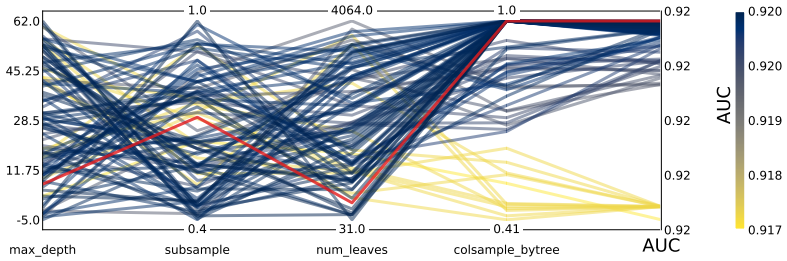


Figure B.3: Hyperparameter optimization results of b-tagging for 3-jet events. The results are shown as parallel coordinates with each hyperparameter along the x-axis and the value of that parameter on the y-axis. Each line is an event in the 4-dimensional space colored according to the performance of 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.

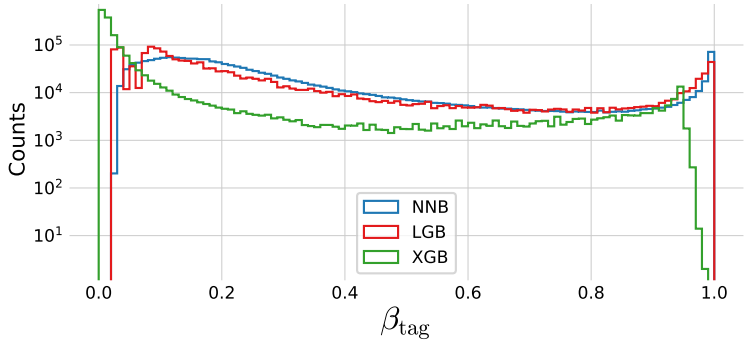


Figure B.4: Histogram of b-tag scores β_{tag} in 3-jet events for NNB (the neural network pre-trained by ALEPH, also called `nnbjet`) in blue, LGB in red, and XGB in green.

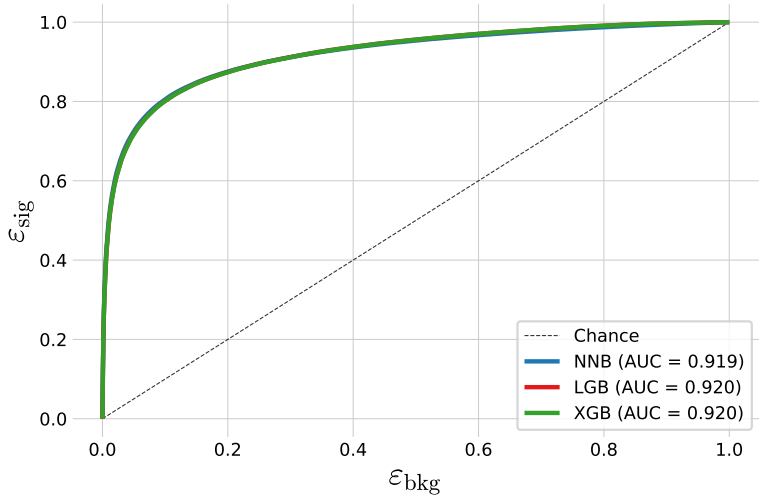


Figure B.5: ROC curve of the three *b*-tag models in 3-jet events for NNB (the pre-trained neural network trained by ALEPH, also called `nnbjet`) in blue, LGB in red, and XGB in green. In the legend the area under curve (AUC) is also shown. Notice that the LGB and XGB models share performance and it is thus due to overplotting that only the green line for XGB can be seen. In the machine learning community the background efficiency ε_{bkg} is sometimes know as the false positive rate (FPR) and the signal efficiency ε_{sig} as the true positive rate (TPR).

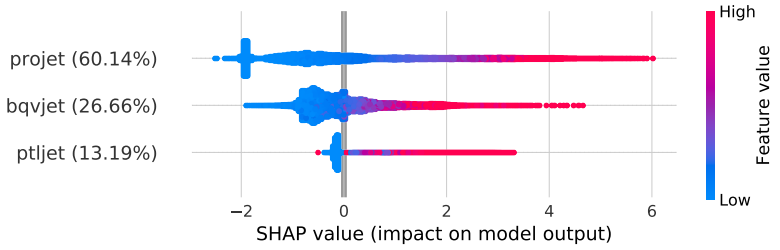


Figure B.6: Global feature importances for the LGB *b*-tagging algorithm on 3-jet events. The normalized feature importance is shown in the parenthesis and the each dot is an observation showing the dependance between the SHAP value and the feature's value.

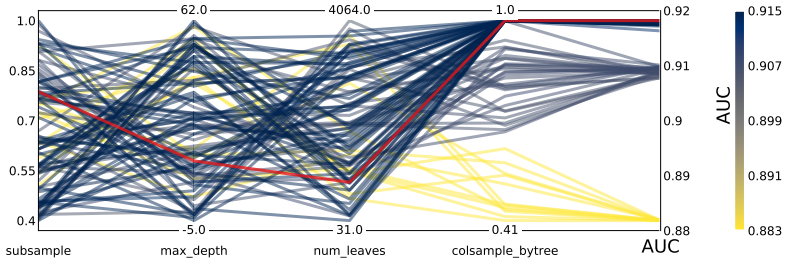


Figure B.7: Hyperparameter optimization results of *g*-tagging for 3-jet events for energy ordered jets.

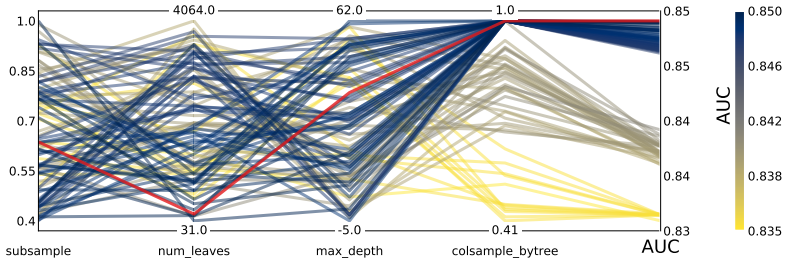


Figure B.8: Hyperparameter optimization results of *g*-tagging for 3-jet events for (row) shuffled jets.

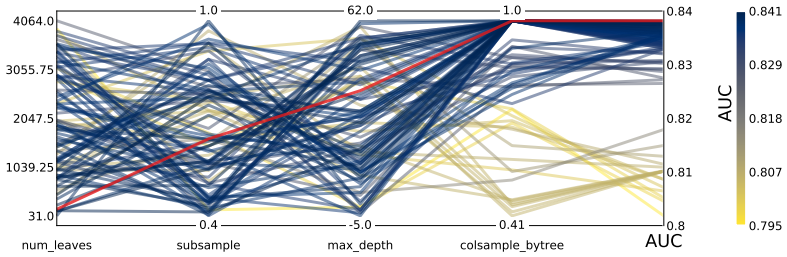


Figure B.9: Hyperparameter optimization results of g-tagging for 4-jet events for energy ordered jets.

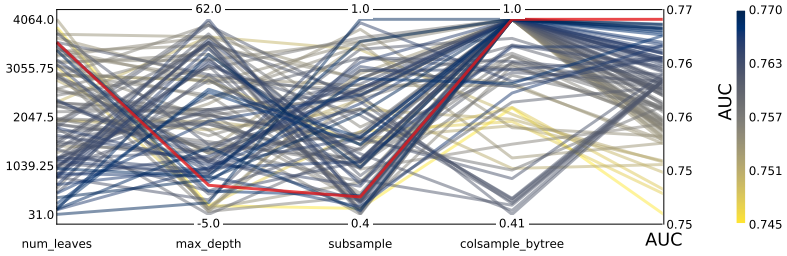


Figure B.10: Hyperparameter optimization results of g-tagging for 4-jet events for (row) shuffled jets.

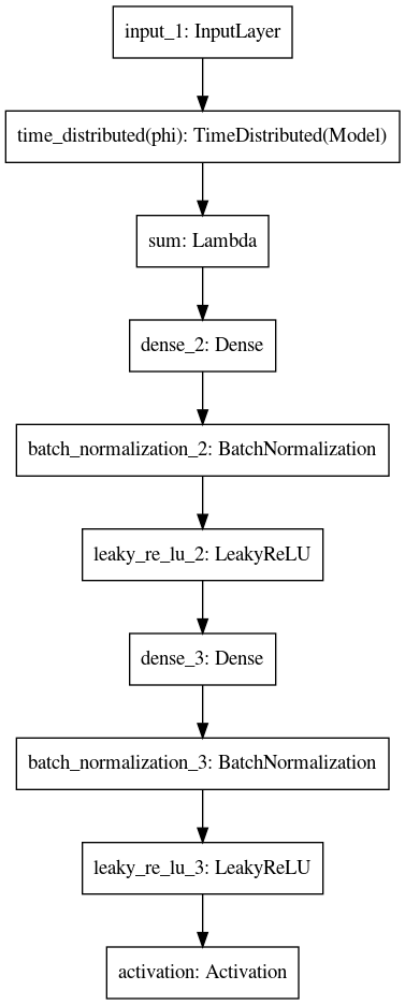


Figure B.11: Architecture of the PermNet neural network.

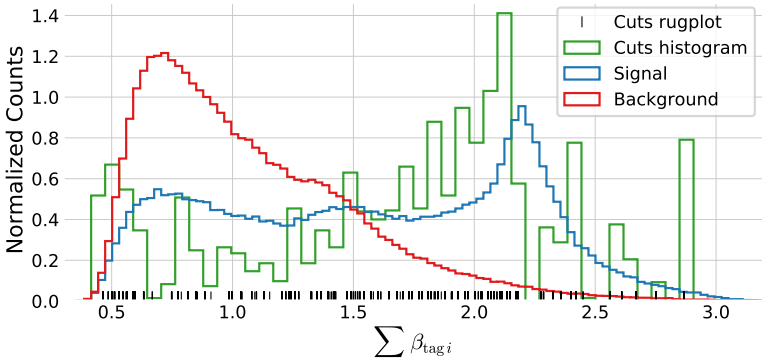


Figure B.12: Histogram of the distribution of signal in blue and background in red for the 1-dimensional sum of b-tags for 4-jet events. A histogram of the cut values from the LGB model trained on this data is shown in green together with a rug plot of the cut values in black. Notice how most of the cuts match up with the signal peak at around a $\sum \beta_i \sim 2.1$.

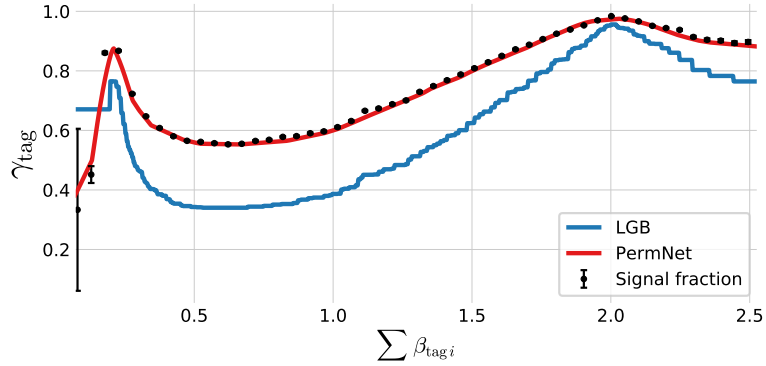


Figure B.13: Plot of the (1D) g-tag scores for 3-jet events as a function of $\sum \beta_i$ for the LGB model in blue and the PermNet model in red. The signal fraction (based on the signal and background histograms in Figure B.14) is plotted as black error bars where the size of the error bars is based on the propagated uncertainties of the signal and background histogram assuming Poissonian statistics.

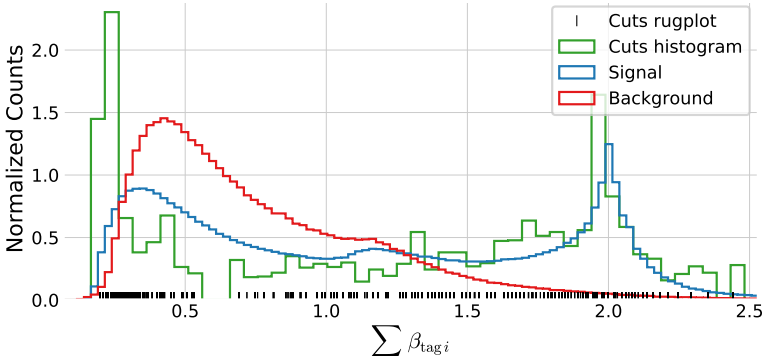


Figure B.14: Histogram of the distribution of signal in blue and background in red for the 1-dimensional sum of b-tags for 3-jet events. A histogram of the cut values from the LGB model trained on this data is shown in green together with a rug plot of the cut values in black. Notice how most of the cuts match up with the signal peak at around a $\sum \beta_i \sim 2.1$.

List of Figures

List of Tables

Bibliography

[1] Advanced Topics in Machine Learning (ATML). URL https://kurser.ku.dk/course/ndak15014u.

[2] Allstate Claims Severity - Fair Loss. URL https://kaggle.com/c/allstate-claims-severity.

[3] Dmlc/xgboost. URL https://github.com/dmlc/xgboost.

[4] HEP meets ML award | The Higgs Machine Learning Challenge. URL https://higgsml.lal.in2p3.fr/prizes-and-award/award/.

[5] The Large Electron-Positron Collider | CERN. URL https://home.cern/science/accelerators/large-electron-positron-collider.

[6] Microsoft/LightGBM. URL https://github.com/microsoft/LightGBM/blob/b3975555d7023fd05f8e56326905fe7b185109de/src/treelearner/serial_tree_learner.cpp#L282.

[7] Scikit-hep/uproot. URL https://github.com/scikit-hep/uproot.

[8] Datashader: Revealing the Structure of Genuinely Big Data. URL https://github.com/holoviz/datashader.

[9] O. . Www.OIS.dk - Din genvej til ejendomsdata. URL https://www.ois.dk/.

[10] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. URL http://tensorflow.org/.

[11] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning From Data*. AMLBook. ISBN 978-1-60049-006-4.

[12] D. Albanese, S. Riccadonna, C. Donati, and P. Franceschi. A practical tool for maximal information coefficient analysis. 7. ISSN 2047-217X. doi: 10.1093/gigascience/giy032. URL https://doi.org/10.1093/gigascience/giy032.

[13] E. Anderson. The Species Problem in Iris. 23(3):457–509. ISSN 00266493. doi: 10.2307/2394164. URL www.jstor.org/stable/2394164.

[14] B. Andersson, G. Gustafson, G. Ingelman, and T. Sjöstrand. Parton fragmentation and string dynamics. 97(2):31–145. ISSN 0370-1573. doi: 10.1016/0370-1573(83)90080-7. URL http://www.sciencedirect.com/science/article/pii/0370157383900807.

[15] S. R. Armstrong. A Search for the standard model Higgs boson in four jet final states at center-of-mass energies near 183-GeV with the ALEPH detector at LEP. URL http://wwwlib.umi.com/dissertations/fullcit?p9910371.

[16] M. Awad and R. Khanna. Support Vector Regression. In M. Awad and R. Khanna, editors, *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*, pages 67–80. Apress. ISBN 978-1-4302-5990-9. doi: 10.1007/978-1-4302-5990-9_4. URL https://doi.org/10.1007/978-1-4302-5990-9_4.

[17] R. J. Barlow. *Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences (Manchester Physics Series)*. WileyBlackwell, reprint edition. ISBN 0-471-92295-1. URL http://www.amazon.co.uk/Statistics-Statistical-Physical-Sciences-Manchester/dp/0471922951%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0471922951.

[18] J. T. Barron. A General and Adaptive Robust Loss Function. URL http://arxiv.org/abs/1701.03077.

[19] W. Bartel et al. Experimental study of jets in electron-positron annihilation. 101(1):129–134. ISSN 0370-2693. doi: 10.1016/0370-2693(81)90505-0. URL http://www.sciencedirect.com/science/article/pii/0370269381905050.

[20] E. Becht, C.-A. Dutertre, I. W. H. Kwok, L. G. Ng, F. Ginhoux, and E. W. Newell. Evaluation of UMAP as an alternative to t-SNE for single-cell data. page 298430, . doi: 10.1101/298430. URL https://www.biorxiv.org/content/10.1101/298430v1.

[21] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I. W. H. Kwok, L. G. Ng, F. Ginhoux, and E. W. Newell. Dimensionality reduction for visualizing single-cell data using UMAP. 37 (1):38–44, . ISSN 1546-1696. doi: 10.1038/nbt.4314. URL https://www.nature.com/articles/nbt.4314.

[22] J. Bergstra and Y. Bengio. Random Search for Hyper-parameter Optimization. 13:281–305. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2188385.2188395.

[23] C. Bierlich. Rope hadronization, geometry and particle production in pp and pA Collisions. URL https://lup.lub.lu.se/search/ws/files/18474576/thesis.pdf.

[24] Bolighed. Bolighed - usikkerhed i data-vurderingen. URL https://bolighed.dk/om-bolighed/spoergsmaal-og-svar/#boligvaerdi.

[25] L. Breiman. Random Forests. 45(1):5–32. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[26] E. Brochu, V. M. Cora, and N. de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. URL http://arxiv.org/abs/1012.2599.

[27] R. Brun and F. Rademakers. ROOT — An object oriented data analysis framework. 389(1):81–86. ISSN 0168-9002. doi: 10.1016/S0168-9002(97)00048-X. URL http://www.sciencedirect.com/science/article/pii/S016890029700048X.

[28] A. Buckley, J. Butterworth, S. Gieseke, D. Grellscheid, S. Hoche, H. Hoeth, F. Krauss, L. Lonnblad, E. Nurse, P. Richardson, S. Schumann, M. H. Seymour, T. Sjostrand, P. Skands, and B. Webber. General-purpose event generators for LHC physics. 504(5):145–233. ISSN 03701573. doi: 10.1016/j.physrep.2011.03.005. URL http://arxiv.org/abs/1101.2599.

[29] C. Burgard. Standard model of physics | TikZ example. URL http://www.texample.net/tikz/examples/model-physics/.

[30] D. Buskulic et al. An investigation of Bdo and Bso oscillation. 322(4):441–458. ISSN 0370-2693. doi: 10.1016/0370-2693(94)91177-0. URL http://www.sciencedirect.com/science/article/pii/0370269394911770.

[31] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. pages 785–794. doi: 10.1145/2939672.2939785. URL http://arxiv.org/abs/1603.02754.

[32] T. A. Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. 716(1):1–29. ISSN 03702693. doi: 10.1016/j.physletb.2012.08.020. URL http://arxiv.org/abs/1207.7214.

[33] A. Diaz-Papkovich, L. Anderson-Trocmé, C. Ben-Eghan, and S. Gravel. UMAP reveals cryptic population structure and phenotype heterogeneity in large genomic cohorts. 15(11). ISSN 1553-7390. doi: 10.1371/journal.pgen.1008432. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6853336/.

[34] S. D. DST. Price Index (EJ14) - Statistics Denmark. URL https://www.dst.dk/en/Statistik/emner/priser-og-forbrug/ejendomme.

[35] D. et al. Buskulic. A precise measurement of hadrons. 313(3): 535–548. ISSN 0370-2693. doi: 10.1016/0370-2693(93)90028-G. URL http://www.sciencedirect.com/science/article/pii/037026939390028G.

[36] F. Faye. Frederik Faye / deepcalo. URL https://gitlab.com/ffaye/deepcalo.

[37] R. A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. 7(2):179–188. ISSN 2050-1439. doi: 10.1111/j.1469-1809.1936.tb02137.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x.

[38] Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In P. Vitányi, editor, *Computational Learning Theory*, pages 23–37. Springer Berlin Heidelberg. ISBN 978-3-540-49195-8. AdaBoost.

[39] S. L. Glashow. Partial-symmetries of weak interactions. 22(4): 579–588. ISSN 0029-5582. doi: 10.1016/0029-5582(61)90469-2. URL http://www.sciencedirect.com/science/article/pii/0029558261904692.

[40] N. Guttenberg, N. Virgo, O. Witkowski, H. Aoki, and R. Kanai. Permutation-equivariant neural networks applied to dynamics prediction. URL http://arxiv.org/abs/1612.04530.

[41] A. E. Harvey and S. Peters. Estimation Procedures for Structural Time Series Models. doi: 10.1002/for.3980090203.

[42] T. Hastie and R. Tibshirani. Generalized Additive Models: Some Applications. 82(398):371–386. ISSN 01621459. doi: 10.2307/2289439. URL www.jstor.org/stable/2289439.

[43] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer-Verlag, 2 edition. ISBN 978-0-387-84857-0. URL //www.springer.com/la/book/9780387848570.

[44] K. Hornik. Approximation capabilities of multilayer feedforward networks. 4(2):251–257. ISSN 0893-6080. doi: 10.1016/0893-6080(91)90009-T. URL http://www.sciencedirect.com/science/article/pii/089360809190009T.

[45] P. Huber and E. Ronchetti. *Robust Statistics*. Wiley Series in Probability and Statistics. Wiley. ISBN 978-1-118-21033-8. URL https://books.google.dk/books?id=j1OhquR_j88C.

[46] S. Hviid, Juul. Working Paper: A regional model of the Danish housing market. URL http://www.nationalbanken.dk/en/publications/Pages/2017/11/Working-Paper-A-regional-model-of-the-Danish-housing-market.aspx.

[47] F. James and M. Roos. Minuit – a system for function minimization and analysis of the parameter errors and correlations. 10:343–367. doi: 10.1016/0010-4655(75)90039-9.

[48] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc. URL http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

[49] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. URL http://arxiv.org/abs/1412.6980.

[50] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. 49(4): 764–766. ISSN 0022-1031. doi: 10.1016/j.jesp.2013.03.013. URL http://www.sciencedirect.com/science/article/pii/S0022103113000668.

[51] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 4768–4777. Curran Associates Inc. ISBN 978-1-5108-6096-4. URL http://dl.acm.org/citation.cfm?id=3295222.3295230.

[52] S. M. Lundberg, G. G. Erion, and S.-I. Lee. Consistent Individualized Feature Attribution for Tree Ensembles - SHAP. . URL http://arxiv.org/abs/1802.03888.

[53] S. M. Lundberg, G. G. Erion, and S.-I. Lee. Consistent Individualized Feature Attribution for Tree Ensembles. . URL http://arxiv.org/abs/1802.03888.

[54] A. L. Maas. Rectifier nonlinearities improve neural network acoustic models.

[55] L. McInnes and J. Healy. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. URL http://arxiv.org/abs/1802.03426.

[56] T. C. Mills. *Time Series Techniques for Economists / Terence c. Mills*. Cambridge University Press Cambridge [England] ; New York. ISBN 0-521-34339-9 0-521-40574-2. URL http://www.loc.gov/catdir/toc/cam031/89007187.html.

[57] I. Mulalic, H. Rasmussen, J. Rouwendal, and H. H. Woltmann. The Financial Crisis and Diverging House Prices: Evidence

from the Copenhagen Metropolitan Area. ISSN 1556-5068. doi: 10.2139/ssrn.3041272. URL https://www.ssrn.com/abstract=3041272.

[58] Particle Data Group et al. Review of Particle Physics. 98 (3):030001. doi: 10.1103/PhysRevD.98.030001. URL https://link.aps.org/doi/10.1103/PhysRevD.98.030001.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. 12:2825–2830.

[60] E. Polley and M. van der Laan. Super Learner In Prediction. URL https://biostats.bepress.com/ucbbiostat/paper266.

[61] J. Proriol, J. Jousset, C. Guicheney, A. Falvard, P. Henrard, D. Pallin, P. Perret, and B. Brandl. TAGGING B QUARK EVENTS IN ALEPH WITH NEURAL NETWORKS (comparison of different methods : Neural Networks and Discriminant Analysis). page 27.

[62] A. Purcell. Go on a particle quest at the first CERN webfest. URL https://cds.cern.ch/record/1473657.

[63] S. Ravanbakhsh, J. Schneider, and B. Poczos. Deep Learning with Sets and Point Clouds. URL http://arxiv.org/abs/1611.04500.

[64] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti. Detecting Novel Associations in Large Data Sets. 334(6062):1518–1524. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.1205438. URL http://science.sciencemag.org/content/334/6062/1518.

[65] P. J. Rousseeuw and C. Croux. Alternatives to the Median Absolute Deviation. 88(424):1273–1283. ISSN 0162-1459. doi: 10.1080/01621459.1993.10476408. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1993.10476408.

[66] A. Salam. Weak and electromagnetic interactions. In *Selected Papers of Abdus Salam*, volume Volume 5 of *World Scientific Series in 20th Century Physics*, pages 244–254. WORLD SCIENTIFIC. ISBN 978-981-02-1662-7. doi: 10.1142/9789812795915_0034. URL https://www.worldscientific.com/doi/abs/10.1142/9789812795915_0034.

[67] L. Scodellaro. B tagging in ATLAS and CMS. URL http://arxiv.org/abs/1709.01290.

[68] L. Shapley. A value for n-person games. In *The Shapley Value*, volume 28 of *Annals of Math Studies*, pages 307–317. doi: 10.1017/CBO9780511528446.003.

[69] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands. An Introduction to PYTHIA 8.2. 191:159–177. ISSN 00104655. doi: 10.1016/j.cpc.2015.01.024. URL http://arxiv.org/abs/1410.3012.

[70] S. J. Taylor and B. Letham. Forecasting at scale. doi: 10.7287/peerj.preprints.3190v2. URL https://peerj.com/preprints/3190.

[71] i. team. Iminuit – A python interface to minuit. URL https://github.com/scikit-hep/iminuit.

[72] R. Tibshirani. Regression Shrinkage and Selection via the Lasso. 58(1):267–288. ISSN 0035-9246. URL www.jstor.org/stable/2346178.

[73] A. Tikhonov. *On the Stability of Inverse Problems*, volume vol. 39 of *Doklady Akademii Nauk SSSR*.

[74] S. Toghi Eshghi, A. Au-Yeung, C. Takahashi, C. R. Bolen, M. N. Nyachienga, S. P. Lear, C. Green, W. R. Mathews, and W. E. O'Gorman. Quantitative Comparison of Conventional and t-SNE-guided Gating Analyses. 10. ISSN 1664-3224. doi: 10.3389/fimmu.2019.01194. URL https://www.frontiersin.org/articles/10.3389/fimmu.2019.01194/full.

[75] d. L. M. J. van, E. C. Polley, and A. E. Hubbard. Super Learner. 6(1). ISSN 1544-6115. doi: 10.2202/1544-6115.1309. URL https://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml.

[76] L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. 9:2579–2605. ISSN ISSN 1533-7928. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[77] F. van Veen. The Neural Network Zoo. URL http://www.asimovinstitute.org/neural-network-zoo/.

[78] V. Vapnik. Principles of Risk Minimization for Learning Theory. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pages 831–838. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-222-9. URL http://dl.acm.org/citation.cfm?id=2986916.2987018.

[79] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors.

SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. page arXiv:1907.10121. URL https://ui.adsabs.harvard.edu/abs/2019arXiv190710121V/abstract.

[80] I. Wallach and R. Lilien. The protein–small-molecule database, a non-redundant structural resource for the analysis of protein-ligand binding. 25(5):615–620. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp035. URL https://academic.oup.com/bioinformatics/article/25/5/615/183421.

[81] S. Weinberg. A Model of Leptons. 19(21):1264–1266. doi: 10.1103/PhysRevLett.19.1264. URL https://link.aps.org/doi/10.1103/PhysRevLett.19.1264.

[82] H. Wickham. Tidy data. 59(10):1–23. ISSN 1548-7660. doi: 10.18637/jss.v059.i10. URL https://www.jstatsoft.org/v059/i10.

[83] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep Sets. URL http://arxiv.org/abs/1703.06114.

Index