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

Abstract iii

Table of Contents v

Foreword ix

1 *Introduction* 1

2 *Machine Learning Theory* 5

2.1 *Statistical Learning Theory* 5

2.2 *Supervised Learning* 6

2.3 *Generalization Bound* 7

2.3.1 *Generalization Bound for infinite hypotheses* 9

2.4 *Avoiding overfitting* 10

2.4.1 *Model Regularization* 10

2.4.2 *Cross Validation* 12

2.4.3 *Early Stopping* 14

2.5 *Loss functions* 14

2.5.1 *Evaluation Function* 16

2.6 *Decision Trees* 16

2.6.1 *Ensembles of Decision Trees* 17

2.7 *Hyperparamater Optimization* 19

2.7.1 *Grid Search* 20

2.7.2 *Random Search* 20

2.7.3 *Bayesian Optimization* 21

2.8 *Feature Importance* 23

3 Danish Housing Prices 27
3.1 Data Preparation and Exploratory Data Analysis 28
3.1.1 Correlations 30
3.1.2 Validity of input variables 32
3.1.3 Cuts 33
3.2 Feature Augmentation 33
3.2.1 Time-Dependent Price Index 34
3.3 Evaluation Function 35
3.4 Initial Hyperparameter Optimization 36
3.5 Hyperparameter Optimization 38
3.6 Results 40
3.7 Model Inspection 43
3.8 Multiple Models 45
3.9 Discussion 47

4 Particle Physics and LEP 53
4.1 The Standard Model 53
4.2 Quark Hadronization 55
4.3 The ALEPH Detector and LEP 56
4.4 Jet clustering 58
4.5 The variables 58

5 Quark Gluon Analysis 63
5.1 Data Preprocessing 63
5.2 Exploratory Data Analysis 64
5.3 Loss and Evaluation Function 67
5.4 b-Tagging Analysis 68
5.4.1 b-Tagging Hyperparameter Optimization 68
5.4.2 b-Tagging Results 69
5.4.3 b-Tagging Model Inspection 70
5.5 Truncated Uniform PDF 72
5.6 g-Tagging Analysis 72
5.6.1 Permutation Invariance 73
5.6.2 g-Tagging Hyperparameter Optimization 73
5.6.3 PermNet 74
5.6.4 1D Comparison of LGB and PermNet 74
5.6.5 g-Tagging Results 75

6 *Discussion and Outlook* 81

7 *Conclusion* 83

7.1 *Tufte-LaTeX Website* 83

7.2 *Tufte-LaTeX Mailing Lists* 83

7.3 *Getting Help* 83

A *Housing Prices Appendix* 85

B *Quarks vs. Gluons Appendix* 113

List of Figures 123

List of Tables 126

Index 135

Foreword

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.

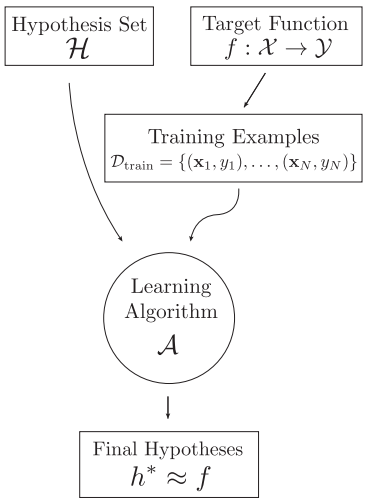

Figure 2.1: Schematic overview of the learning problem and how to find the optimal hypothesis h^* to approximate f given the training data $\mathcal{D}_{\text{train}}$.

[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.

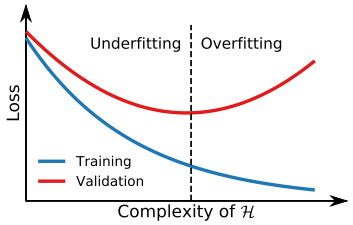

Figure 2.2: Illustration of the empirical loss as a function of model complexity. The training error is shown in blue and validation error in red.

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.

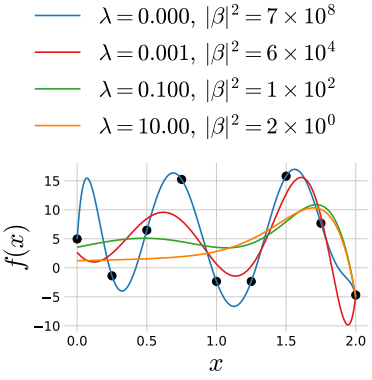

Figure 2.3: Effect of tuning the regularization strength λ in ridge regression.

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].

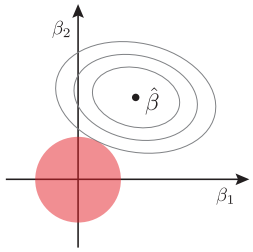

Figure 2.4: Sketch of the minimization problem defined in equation (2.22), i.e. for a L_2-penalty. The constrain region shown in red is defined as $\beta_1^2 + \beta_2^2 \leq t$ for L_2 in $2D$-space and the contours of the unconstrained solution is shown with grey, dashed lines.

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

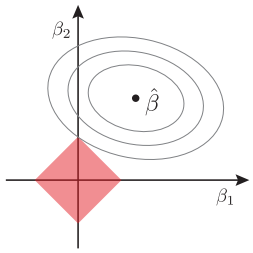

Figure 2.5: Sketch of the similar minimization problem defined in Figure 2.4 for the L_1-penalty. The constrain region shown in red is defined as $|\beta_1| + |\beta_2| \leq t$ for L_1 in $2D$-space and the contours of the unconstrained solution is shown with grey, dashed lines.

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.

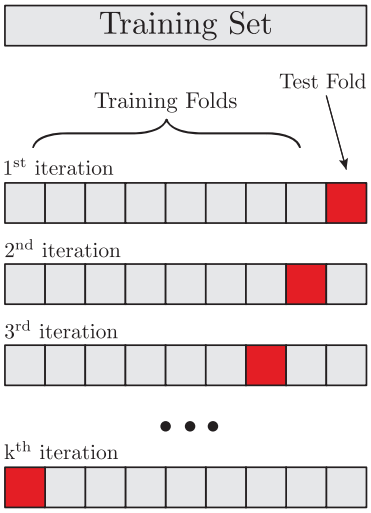

Figure 2.6: k-fold cross validation.

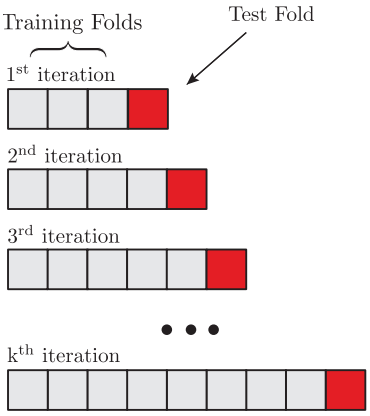

Figure 2.7: k-fold cross validation for time series data.

[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.

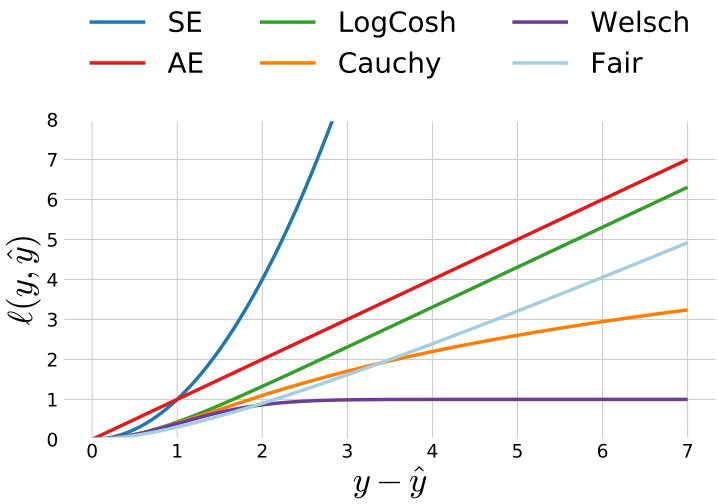

Figure 2.8: Comparison of the the six loss functions SE, AE, LogCosh, Cauchy, Welsch, and Fair as a function of $y - \hat{y}$, see equation (2.27). In the plot SE is shown in blue, AE in red, LogCosh in green, Cauchy in orange, Welsch in purple, and Fair in light blue. For the Cauchy, Welsch, and Fair 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 $y - \hat{y}$.

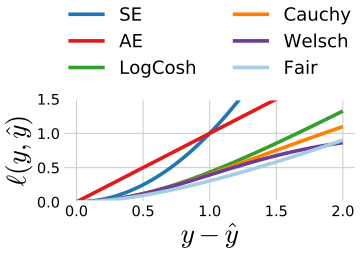

Figure 2.9: Zoom in of Figure 2.8.

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

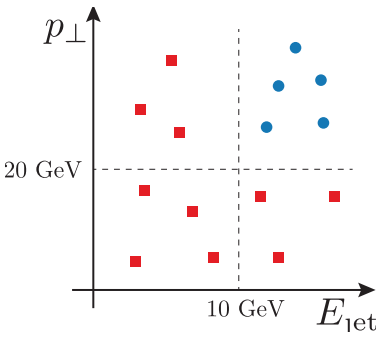

Figure 2.10: Illustration of the cuts a decision tree model make for signal in blue circles and background in red squares. This is an visualization in the feature space of the decision tree seen in Figure 2.11.

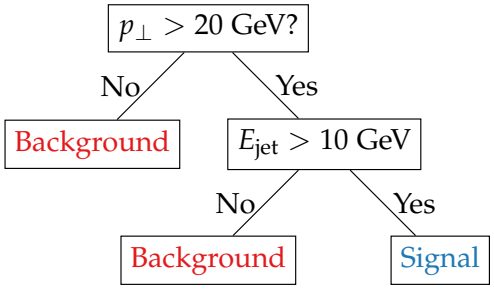

Figure 2.11: Illustration of a simple decision tree. Here the tree partitions the input feature space consisting of the two variables $p_\perp$ and E_{jet} into two categories; either signal or background. A visualization of the cuts in the feature space can be seen in Figure 2.10.

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].

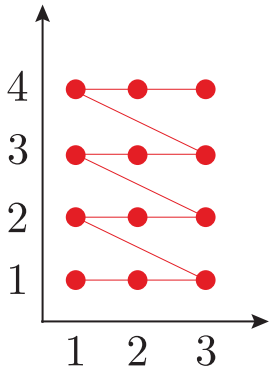

Figure 2.12: Visualization of grid search run on the two hyperparameters x and why y with the domains $x = \{1,2,3\}$ and $y = \{1,2,3,4\}$.

[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.

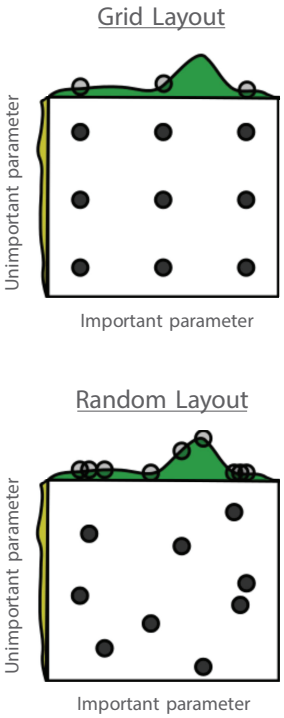

Figure 2.13: Visualization of the difference between grid search and random search. Adapted from Bergstra and Bengio [22].

[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.

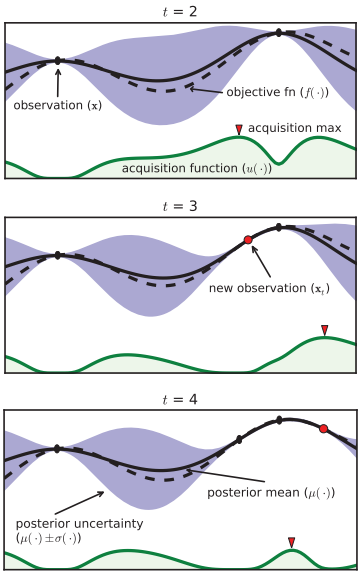

Figure 2.14: XXX **TODO!**. Adapted from Brochu et al. [26].

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.

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.

B. Quarks vs. Gluons Appendix

List of Figures

2.1 The learning problem. 6
2.2 Approximation-Estimation Tradeoff 10
2.3 Regularization Strength 11
2.4 Regularization Effect of L_2 12
2.5 Regularization Effect of L_1 12
2.6 k-Fold Cross Validation 13
2.7 k-Fold Cross Validation for Time Series Data 13
2.8 Objective Functions. 16
2.9 Objective Functions Zoom In. 16
2.10 Decision Tree Cuts In Feature Space 16
2.11 Decision Tree 17
2.12 Grid Search 20
2.13 Random Search 21
2.14 Bayesian Optimization 22

3.1 Danish Housing Price Index 27
3.2 Distributions for the housing price dataset 28
3.3 Distributions for the housing price dataset 29
3.4 Histogram of prices of houses and apartments sold in Denmark 30
3.5 Linear correlation between variables and price 31
3.6 Comparison of the Linear Correlation ρ and the Non-Linear MIC. 31
3.7 Non-linear correlation between variables and price 32
3.8 Validity of input features 32
3.9 Validity Dendrogram 33
3.10 Prophet Forecast for apartments 35
3.11 Prophet Trends 35
3.12 XXX 37
3.13 Overview of initial hyperparamater optimization of the housing model for apartments 38
3.14 XXX 38
3.15 XXX 38
3.16 XXX 39
3.17 Hyperparameter optimization: random search results 40
3.18 Early Stopping results 40
3.19 Performance of XGB-model on apartment prices 41
3.20 2018 XGB Forecast 41
3.21 2018 XGB Forecast 43
3.22 SHAP Prediction Explanation for apartment 44
3.23 Feature importance of apartments prices using XGB 45

3.24 Feature importance of apartments prices using XGB XXX 45
3.25 Multiple Models XXX 46
3.26 SHAP plot villa TFIDF XXX 48

4.1 The Standard Model 54
4.2 Feynman diagram for the jet production at LEP 55
4.3 Quark splitting 55
4.4 Hadronization process 56
4.5 The ALEPH detector 57
4.6 Polar angle 57
4.7 Azimuthal angle 57

5.1 Histograms of the vertex variables 65
5.2 UMAP visualization of vertex variables for 4-jet events 66
5.3 UMAP visualization of vertex variables for 3-jet events 66
5.4 UMAP visualization of vertex variables for 2-jet events 66
5.5 Correlation of Vertex Variables 67
5.6 Plot of the log-loss $\ell_{\log}$ 68
5.7 Hyperparameter Optimization of b-tagging 69
5.8 Parallel Plot of HPO Results for 4-Jet b-Tagging 69
5.9 b-Tag Scores in 4-Jet Events 70
5.10 ROC curve for 4-jet b-tagging 70
5.11 Global Feature Importances for the LGB b-Tagging Algorithm on 4-Jet Events 71
5.12 The expit Function 71
5.13 The logit Function 71
5.14 SHAP 3-Jet Model Explanation for b-like Jet 72
5.15 Hyperparameter Optimization of g-tagging 74
5.16 1D Sum Models Predictions and Signal Fraction for 4-jets events 75
5.17 g-Tag Scores in 4-Jet Events 76
5.18 g-tag scores in 4-jet events for signal and background 76
5.19 ROC curve for g-tag in 4-jet events 77
5.20 Monte Carlo – Data bias for b-tags and jet energy 77
5.21 b-Tagging Efficiency $\varepsilon_b^{b-\text{sig}}$ as a function of jet energy 77
5.22 b-Tagging Efficiency $\varepsilon_b^{g-\text{sig}}$ as a function of jet energy 78
5.23 b-Tagging Efficiency $\varepsilon_g^{g-\text{sig}}$ as a function of jet energy 78
5.24 b-Tagging Efficiency $\varepsilon_g^{b-\text{sig}}$ as a function of jet energy 78
5.25 g-Tagging proxy efficiency for $b\bar{b}g$-events as function of the mean invariant mass 78
5.26 g-Tagging proxy efficiency for $b\bar{b}g$-events as function of g-tag 79
5.27 g-Tagging efficiency for 4-jet events in MC as a function of normalized gluon gluon jet energy difference 79
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 79
5.29 R kt CA overview XXX **TODO!** 79
5.30 R kt CA cut region A XXX **TODO!** 80

A.1 Validity Heatmap 85
A.2 Distributions for the housing price dataset 86

A.3 Distributions for the housing price dataset 87
A.4 Distributions for the housing price dataset 88
A.5 Distributions for the housing price dataset 89
A.6 Distributions for the housing price dataset 90
A.7 Distributions for the housing price dataset 91
A.8 Distributions for the housing price dataset 92
A.9 Distributions for the housing price dataset 93
A.10 Distributions for the housing price dataset 94
A.11 Distributions for the housing price dataset 95
A.12 Distributions for the housing price dataset 96
A.13 Distributions for the housing price dataset 97
A.14 Distributions for the housing price dataset 98
A.15 Distributions for the housing price dataset 99
A.16 Linear Correlations 101
A.17 MIC non-linear correlation 102
A.18 Prophet Forecast for apartments 103
A.19 Prophet Trends 103
A.20 Overview of initial hyperparamater optimization of the housing model for houses 107
A.21 XXX 108
A.22 XXX 108
A.23 XXX 108
A.24 XXX 109
A.25 XXX 109
A.26 XXX 109
A.27 Performance of XGB-model on apartment prices 110

B.1 UMAP Parameter Grid Search 115
B.2 Visualization of the t-SNE algorithm 115
B.3 Parallel Plot of HPO results for 3-jet *b*-Tagging 116
B.4 b-tag scores in 3-jet events 116
B.5 ROC curve for 3-jet *b*-tagging 117
B.6 Global Feature Importances for the LGB *b*-Tagging Algorithm on 3-Jet Events 117
B.7 Parallel Plot of HPO Results for 3-Jet *g*-Tagging for Energy Ordered Jets 117
B.8 Parallel Plot of HPO Results for 3-Jet *g*-Tagging for Shuffled Jets 117
B.9 Parallel Plot of HPO Results for 4-Jet *g*-Tagging for Energy Ordered Jets 118
B.10 Parallel Plot of HPO Results for 4-Jet *g*-Tagging for Shuffled Jets 118
B.11 PermNet Architecture 118
B.12 1D LGB Model Cuts for 4-jets events 119
B.13 1D Sum Models Predictions and Signal Fraction for 3-jets events 119
B.14 1D LGB Model Cuts for 3-jets events 119

List of Tables

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). 29
3.2 Basic Cuts 33
3.3 Side Door Mapping. 33
3.4 Street Mapping 33
3.5 train test split XXX **TODO!**. 36
3.6 train test split tight XXX **TODO!**. 36
3.7 Cauchy-ejerlejlighed. 37
3.8 Cauchy-villa. 37
3.9 XXX 39
3.10 XXX 41
3.11 XXX ejer 43
3.12 XXX villa 43

5.1 Dimensions of dataset for Data 64
5.2 Dimensions of dataset for MC and MCb 64
5.3 Number of different types of jets for MC and MCb. See also Table B.1 in the appendix for relative numbers. 65
5.4 Random Search PDFs for LGB 68

A.1 XXX **TODO!**. 100
A.2 Energy Rating Mapping 102
A.3 Rmse-ejerlejlighed-appendix. 104
A.4 Logcosh-ejerlejlighed-appendix. 104
A.5 Cauchy-ejerlejlighed-appendix. 104
A.6 Welsch-ejerlejlighed-appendix. 105
A.7 Fair-ejerlejlighed-appendix. 105
A.8 Rmse-villa-appendix. 105
A.9 Logcosh-villa-appendix. 105
A.10 Cauchy-villa-appendix. 106
A.11 Welsch-villa-appendix. 106
A.12 Fair-villa-appendix. 106
A.13 XXX ejer tight 111
A.14 XXX villa tight 111

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. 114
B.2 Random Search PDFs for XGB 116

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

license, ii