Stockholms universitet

An Interpretable and Comprehensive Machine Learning Study of ADHD Symptom Severity from Cognitive Tasks and Chronotype

Abir Myllymäki

Masteruppsats i matematisk statistik
Master Thesis in Mathematical Statistics

Masteruppsats 2025:15
Matematisk statistik
Juni 2025

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2025:15**
http://www.math.su.se

An Interpretable and Comprehensive Machine Learning Study of ADHD Symptom Severity from Cognitive Tasks and Chronotype

Abir Myllymäki*

June 2025

Abstract

This thesis investigates whether cognitive task performance, together with age and chronotype, can predict adult ADHD symptom severity as mea- sured by the ASRS questionnaire. Data from 356 participants was ana- lyzed using Gaussian mixture models (GMM) for clustering, and eXtreme Gradient Boosting (XGBoost) for both regression and classification. To interpret model behavior, SHapley Additive exPlanations (SHAP) and counterfactual analysis were applied.

Clustering showed weak separation, with significant overlap between components. The XGBoost regression model achieved a test RMSE of 11.70 (approximately 16atic bias toward mid-range scores. Classification performance was limited by class imbalance and feature overlap, resulting in a balanced accuracy of 0.51 (sensitivity 0.99, specificity 0.04). Interpreting the regression model, SHAP analysis found age, chronotype and Tower of London performance as the most influential features, though overall contributions were modest. Counterfactual analysis showed that lowering predicted symptom levels typically required changes to multiple features, especially those related to participants' response time.

The results show the potential of combining cognitive task features with chronotype in ADHD prediction. However, the models faced great limitations in both accuracy and interpretability, likely due to sample size and data imbalance. Future work should focus on larger, more balanced datasets and consider alternative ways of aggregating the cognitive task data before such models can be applied in clinical screening or intervention.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: abir.myllymaki@outlook.com. Supervisor: Chun-Biu Li.

Acknowledgments

First and foremost, I would like to thank my supervisor Chun-Biu Li, as his guidance and expertise made this project possible. Over the past semester, I have received more advice and encouragement from Chun than I could have hoped for, and I'm truly grateful for his support.

I am also thankful to John Axelsson and Leonie Balter, researchers at Karolinska Institutet and Stockholm University, for generously providing the data and for their input on the psychological aspects of the project. Their collaboration made this thesis meaningful.

Lastly, I am forever grateful to my husband Matias Myllymäki, whose love and faith in me have been my strength throughout my studies. Him cheering me on during the most challenging moments, his quiet presence when I needed to think, and his celebration of every small victory have carried me through not only this project but every part of my life.

Contents

1 Introduction

"Wait, what was I doing again?". This familiar question may be a minor everyday annoyance for many, but for those with Attention Deficit Hyperactivity Disorder (ADHD), such distractions can be a constant challenge. Characterized by inattention, impulsivity and restlessness, ADHD affects approximately 3-5% of adults worldwide and remains a challenge not only for clinical diagnosis but also for computational modeling (Ayano et al., 2023; Polanczyk et al., 2007). When left untreated, symptoms can negatively affect school and work performance, relationships and overall quality of life (Kim et al., 2023; Spencer, 2006).

Currently, diagnosis mostly relies on questionnaires and clinical interviews, but there is a growing interest in finding data-driven ways to quantify, predict and better understand ADHD severity (Bzdok and Meyer-Lindenberg, 2018; Dwyer et al., 2018).

Traditional statistical methods often struggle to model the complex and high-dimensional relationships in psychiatric measures. On the contrary, machine learning (ML) can handle large feature sets and find nonlinear interactions given enough data (Tai et al., 2019). Recent applications of ML to mental health research have been increasing, with data collected from brain imaging to measurements from wearable devices (Kim et al., 2023).

Motivated by these advances, this thesis investigates whether ADHD symptom severity can be predicted using data from cognitive tasks assessing attention, memory and related functions, in combination with age and chronotype (morning/evening preference). These task scores provide numeric measures of cognitive function that we hypothesize relate to each participant's ADHD symptom level, as measured by the Adult ADHD Self-Report Scale (ASRS). Beyond predictive accuracy, particular emphasis is placed on model interpretability: which features influence these predictions and how small changes might shift a participants classification.

To address these questions, the analysis implements both unsupervised and supervised ML methods, namely Gaussian Mixture Models (GMM) for clustering and eXtreme Gradient Boosting (XGBoost) for both regression and classification. These are complemented by interpretable AI techniques, including SHapley Additive exPlanations (SHAP) and counterfactual analysis.

The broader goal of this project is not to develop a clinical diagnostic tool, but rather to find data-driven approaches to model behavioral data. In doing so, it draws attention to both the potential and limitations of machine learning methods in the context of analyzing psychiatric data. This thesis investigates the following questions:

1. Can ADHD symptom severity (ASRS score) be accurately predicted from cognitive task performance, age and chronotype?

2. Which features contribute most to these predictions?

3. How do minimal changes in feature values affect ADHD severity classifications?

Through this analysis, we hope to better understand the structure (or lack thereof) in cognitive task performance as it relates to ADHD.

1.1 Thesis Structure

This thesis will proceed as follows: In Section 2, the methodology is detailed, describing the models and tools used, which lay the theoretical groundwork for the analysis. In Section 3, the results of the exploratory analysis and predictive modeling are presented, demonstrating the strengths and limitations of the approach. In Section 4, these results are discussed in relation to the research questions, with emphasis on challenges related to data structure and feature overlap. In Section 5, the main conclusions and implications are summarized. Finally, the appendices support the main text with mathematical derivations, data descriptions and supplementary figures.

2 Methods

This section presents the theoretical foundations and methods underlying the models used in this thesis. It introduces Gaussian mixture models (GMM) for probabilistic clustering, followed by XGBoost for supervised learning and concludes with model interpretation techniques, including SHAP value analysis and counterfactual explanations.

2.1 Gaussian Mixture Models

Clustering is a method in machine learning for grouping observations based on shared patterns or features. Among the most widely used methods is k-means clustering, which assigns each point exclusively to a single cluster, resulting in hard boundaries between groups. However, in many real-world settings, particularly those involving noisy or overlapping data, such strict assignments may be inappropriate.

Gaussian mixture models (GMM) provide a more flexible and probabilistic approach for such cases. Instead of assigning each observation to a single cluster, this method assumes that the data is being generated from a mixture of multivariate Gaussian distributions, each corresponding to a latent subpopulation or component. Unlike hard clustering methods, GMM allows for soft assignments, where each observation has a probability of belonging to multiple clusters. This is especially useful when modeling data where boundaries between clusters may be gradual rather than discrete.

The following subsections present the mathematical formulation of Gaussian mixture models, including parameter estimation using the Expectation-Maximization (EM) algorithm.

2.1.1 Model Definition

Let $\mathbf{x} \in \mathbb{R}^D$ denote an observed D-dimensional continuous random variable. A Gaussian mixture model assumes that $\mathbf{x}$ is generated in two steps: first, a "hidden" label, or component index, $k \in \{1, \ldots, K\}$ is drawn from a categorical distribution, determining which Gaussian component to use. The data point $\mathbf{x}$ is then sampled from the Gaussian distribution associated with that component, see Figure 1 for an illustration.

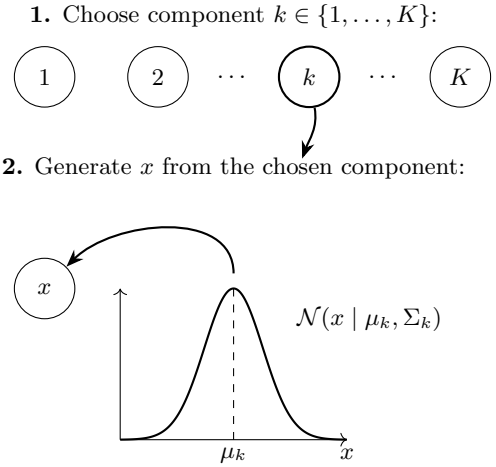


Figure 1: A Gaussian mixture model generates a data point x by first choosing a component $k \in \{1, \ldots, K\}$, then drawing x from the Gaussian distribution with parameters (μ_k, Σ_k).

Formally, let $p(k)$ denote the prior probability of choosing component k, satisfying $\sum_k p(k) = 1$ (Bishop, 2006, p. 111). Then, the marginal distribution over $\mathbf{x}$ is given by:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(k)\, \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{1}$$

where the density $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ corresponds to the k-th multivariate Gaussian distribution with mean $\boldsymbol{\mu}_k \in \mathbb{R}^D$ and covariance matrix $\boldsymbol{\Sigma}_k \in \mathbb{R}^{D \times D}$. Each term in (1) corresponds to one component in the mixture.

The covariance matrices $\boldsymbol{\Sigma}_k$ must be positive definite to ensure a valid Gaussian distribution and control the shape, size and orientation of the component. This makes GMMs very flexible, as they can represent clusters that are spherical, elongated, rotated, or vary in volume. By allowing a full covariance matrix for each component, the model can find cluster structures that are not well approximated by simpler methods like k-means or models with constrained covariance assumptions.

While this flexibility can be an advantage, it introduces a complexity trade-off. An unconstrained GMM with K components in D dimensions involves

estimating

$$\underbrace{K \times D}_{\text{means}} + \underbrace{K \times \frac{D(D+1)}{2}}_{\text{covariances}} + \underbrace{K-1}_{\text{priors}} \tag{2}$$

parameters. With this many free parameters, which grow rapidly with D and K, the model is more prone to overfitting in high dimensions or with small sample sizes. This explains why it is common to restrict covariance structures (diagonal, spherical or ellipsoidal) to reduce the number of parameters; a constraint that reduces flexibility but improves generalization.

To formalize the generative process, we introduce a K-dimensional latent binary variable $\mathbf{z}$ with a 1-of-K ("one-hot") encoding (Bishop, 2006, p. 430). The element $z_k = 1$ indicates that observation $\mathbf{x}$ belongs to component k, with all other $z_j = 0$ for $j \neq k$. For N observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, each $\mathbf{x}_n$ is paired with a latent variable $\mathbf{z}_n = (z_{n1}, \ldots, z_{nK})^\top$. The marginal distribution over $\mathbf{z}_n$ is given by:

$$p(z_{nk} = 1) = p(k), \quad \text{where } \sum_{k=1}^{K} z_{nk} = 1,$$

which ensures that exactly one component is active per observation. The joint distribution over $\mathbf{z}_n$ can then be expressed as:

$$p(\mathbf{z}_n) = p\left(z_1 = 1\right)^{z_1} p\left(z_2 = 1\right)^{z_2} \ldots p\left(z_K = 1\right)^{z_K} = \prod_{k=1}^{K} p(k)^{z_{nk}}. \tag{3}$$

Conditioned on $\mathbf{z}_n$, the observation $\mathbf{x}_n$ follows a Gaussian distribution:

$$p(\mathbf{x}_n \mid \mathbf{z}_n) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}. \tag{4}$$

Due to the one-hot encoding of $\mathbf{z}_n$, only the term corresponding to the active component k contributes to the product.

From the product rule of probability, the joint distribution over $\mathbf{x}_n$ and $\mathbf{z}_n$ is obtained:

$$p(\mathbf{x}_n, \mathbf{z}_n) = p(\mathbf{z}_n)p(\mathbf{x}_n \mid \mathbf{z}_n) = \prod_{k=1}^{K} \left[p(k)\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right]^{z_{nk}}. \tag{5}$$

Marginalizing over $\mathbf{z}_n$ recovers the mixture density by summing over all possible latent states:

$$p(\mathbf{x}_n) = \sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n) = \sum_{k=1}^{K} p(k)\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{6}$$

which directly corresponds to the GMM formulation in Equation (1).

To estimate the model parameters from data, maximum likelihood is implemented. Consider a dataset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of i.i.d. observations, and let $\mathbf{p} := (p(1), \ldots, p(K))$ denote the vector of prior probabilities over components. Then, the likelihood function for the GMM parameters is

$$p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \sum_{k=1}^{K} p(k)\, \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{7}$$

and the log-likelihood is

$$\ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} p(k)\, \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right). \tag{8}$$

The expression in (8) is difficult to optimize directly due to the summation inside the logarithm. As a result, closed-form solutions for parameter estimates are not available (Bishop, 2006, p. 435). Instead, we turn to an iterative method, the *Expectation-Maximization (EM) algorithm*, which is applicable to models with latent variables.

Before presenting the EM algorithm, it is useful to derive the posterior distribution over the latent component index. This posterior, often referred to as the *responsibility* that component k takes for explaining $\mathbf{x}_n$, is denoted by $p(k \mid \mathbf{x}_n)$. Using Bayes' rule, it can be expressed as

$$\underbrace{p(k \mid \mathbf{x}_n)}_{\text{Posterior responsibility}} = \frac{\overbrace{p(k)}^{\text{Prior weight}} \cdot \overbrace{p(\mathbf{x}_n \mid k)}^{\text{Gaussian likelihood}}}{\underbrace{p(\mathbf{x}_n)}_{\text{Marginal likelihood}}} = \frac{p(k) \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} p(j) \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{9}$$

The posterior $p(k \mid \mathbf{x}_n)$ can be equivalently expressed as $p(z_{nk} = 1 \mid \mathbf{x}_n)$ and represents a soft assignment of the data point $\mathbf{x}_n$ to component k. It forms the core of the E-step in the EM algorithm, which is described in Section 2.1.1.2.

In the remainder, the following Bayesian notation is used: $p(k)$ for the prior over components, and $p(k \mid \mathbf{x}_n)$ denotes the posterior responsibility.

2.1.1.1 Maximum Likelihood Estimation

Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the log-likelihood of a K-component Gaussian mixture is

$$\ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} p(k)\, \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right). \tag{10}$$

Maximizing (10) with respect to $\mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$ leads to three closed-form solutions, respectively:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} p(k \mid \mathbf{x}_n) \mathbf{x}_n, \tag{11a}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T, \tag{11b}$$

$$p(k) = \frac{N_k}{N}. \tag{11c}$$

where

$$N_k = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n). \tag{12}$$

Full algebraic derivations of equations (11a)-(12) are given in Appendix A.1. Here, the mean of component k is a weighted average of all data points, where the weight $p(k \mid \mathbf{x}_n)$ quantifies how strongly $\mathbf{x}_n$ belongs to component k. Points with high membership probability for component k contribute more to its mean. Similarly, the covariance matrix is estimated using weighted squared deviations, $(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$, ensuring that each component adapts its shape to the data it is most responsible for, with greater influence from points assigned more confidently to k.

The prior $p(k)$ represents the average responsibility of component k across the entire dataset. Since $N_k = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)$, the prior $p(k) = \frac{N_k}{N}$ is the total membership probability for component k, normalized by the dataset size. A large N_k means component k explains a larger share of the data probabilistically.

Note that, unlike hard clustering, N_k is not an integer count but a sum of probabilities quantifying the likelihood that all points belong to k. It represents the total contribution of the dataset to component k in a probabilistic sense, allowing overlapping clusters and uncertainty in assignments.

These parameter equations present a circular problem: to compute the responsibilities $p(k \mid \mathbf{x}_n)$ we need the parameters $\mathbf{p}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, but to estimate these we need the responsibilities. This mutual dependence calls for an iterative approach to the solution.

Additionally, maximum likelihood estimation for GMM encounters a significant problem due to singularities (Bishop, 2006, p. 433). If a Gaussian "collapses" onto a specific data point (i.e., $\boldsymbol{\mu}_k = \mathbf{x}_n$ and $\boldsymbol{\Sigma}_k \to 0$), the likelihood function becomes unbounded. These singularities lead to severe overfitting, making the maximization of the log-likelihood unstable. To address this, heuristics can be used, such as resetting the mean and covariance of a collapsing component (Bishop, 2006, p. 434). The singularity problem does not occur in a single Gaussian distribution, where if the Gaussian collapses onto a data point, the likelihood of the other data points decreases exponentially, causing the overall likelihood to approach zero.

2.1.1.2 Expectation-Maximization Algorithm

The *expectation-maximization* (EM) algorithm provides a neat solution to the circular problem described in the previous section. Introduced in 1977, EM is an iterative method suitable for finding maximum likelihood solutions in models with latent variables (Dempster et al., 1977). The core focus of EM is to work with the *complete-data log-likelihood* (the joint distribution of the observed data $\mathbf{X}$ and the latent variables $\mathbf{Z}$) rather than just the marginal distribution of observed data (McLachlan et al., 2012, p. 143). For a single observation $\mathbf{x}_n$, the joint distribution is given in (5). Extending to N i.i.d. observations and taking the logarithm gives the complete-data log-likelihood

$$\ln p(\mathbf{X}, \mathbf{Z} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left\{ \ln p(k) + \ln \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}. \tag{13}$$

By expressing the log-likelihood as a linear sum over z_{nk} (via $\sum_{n,k} z_{nk}(\cdot)$), we avoid the problematic logarithm of a sum as seen in the *incomplete-data log-likelihood* in Equation (8). If we knew the values of z_{nk}, maximizing this function would be straightforward and similar to fitting separate Gaussian distributions to disjoint subsets of the data.

However, since we do not observe the latent variables $\mathbf{Z}$, the EM algorithm alternates between estimating their expected values and maximizing the expected complete-data log-likelihood (Bishop, 2006, p. 440). This alternation can be summarized in the following steps:

- **Initialize:** Choose initial values for the parameters $p(k), \boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$.
- **E-step:** For each data point $\mathbf{x}_n$ and each component k, compute the posterior probabilities:

$$p(k \mid \mathbf{x}_n) = \frac{p(k)\,\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} p(j)\,\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.$$

- **M-step:** Update the parameters using the current posterior probabilities:

$$\begin{aligned} p^{\text{new}}(k) &= \frac{N_k}{N} \\ \boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} p(k \mid \mathbf{x}_n) \mathbf{x}_n \\ \boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T \end{aligned}$$

  where

$$N_k = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n).$$

- Convergence check: Evaluate the log-likelihood

$$\ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} p(k) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right),$$

  and check for convergence. If the change in log-likelihood is below a predetermined threshold, or if a maximum number of iterations is reached, stop. Otherwise, return to Step 2.

Each iteration of EM is guaranteed to increase the log-likelihood function (or leave unchanged). This ensures that the algorithm converges to a local maximum or saddle point of the likelihood function (Bishop, 2006, pp. 453–454). However, this does not equal finding the global maximum, which makes the initialization strategy important.

Several common approaches are used to initialize the parameters. One of the simplest is *random initialization*, where K data points are randomly selected as initial means, the covariance matrices are set to the sample covariance of the entire dataset, and the priors are set uniformly as $p(k) = \frac{1}{K}$. Another strategy is to apply the K-means algorithm to the data and to use the resulting cluster centroids as initial means (Bishop, 2006, p. 427). The covariance matrices are then computed from the within-cluster scatter, and the initial priors are set according to the relative sizes of the clusters.

One can also apply multiple restarts, where the EM algorithm is run several times with different initializations. The solution with the highest final log-likelihood is then chosen as the final model. This helps avoid convergence to poor local optima and is useful in noisy or high-dimensional datasets.

2.1.2 Validation

Choosing the optimal number of components K in a Gaussian mixture model can be a challenge. The most common model selection methods are likelihood-based critera, such as the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC). These criteria attempt to balance model fit against complexity:

$$\text{BIC} = -2 \ln p(\mathbf{X} \mid \hat{\mathbf{p}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) + \kappa \ln(N) \tag{14}$$

$$\text{AIC} = -2 \ln p(\mathbf{X} \mid \hat{\mathbf{p}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) + 2\kappa, \tag{15}$$

where $\hat{\mathbf{p}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}$ are the maximum likelihood estimates of the GMM parameters, κ is the number of parameters and N the number of observations. While BIC and AIC are widely used and easy to compute, their assumptions lead to great limitations (Bishop, 2006, p. 217). The BIC, in particular, assumes both large sample sizes and correct model specification, which are assumptions that are not always held in practice. When working with real-world data, the true model may not belong the specified parametric family (Gaussian in our case), resulting in the likelihood function itself being potentially misleading. Additionally, both

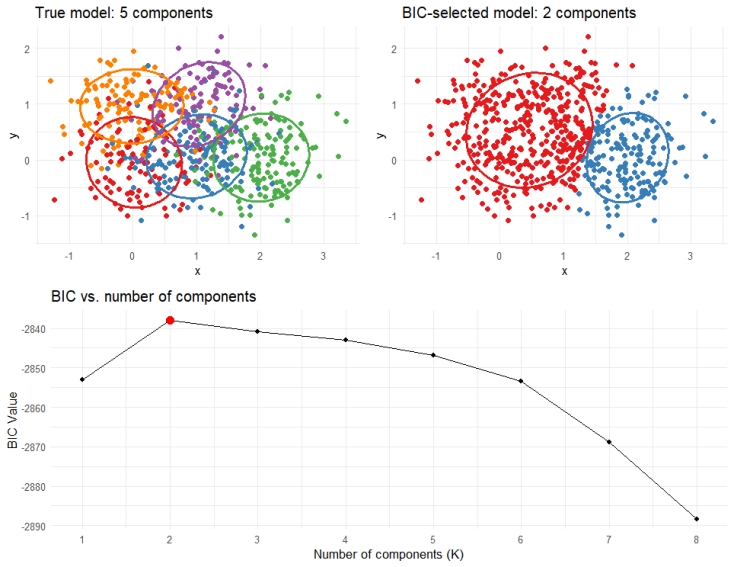


Figure 2: Illustration of BIC's conservative behavior under overlapping components. Although the true number of clusters is $K = 5$, BIC selects $K = 2$ due to its strong penalty for model complexity. This often results in merged clusters and underfitting when cluster separation is low.

BIC and AIC assume that the data is i.i.d., an assumption often not fulfilled in dependent or structured data such as time series, hierarchical groupings, etc.

Moreover, the strong penalty term of BIC often leads to the selection of models with fewer components than is truly present in the data (Bishop, 2006, p. 33; Hastie et al., 2001, p. 235). This conservative bias can lead to underfitting, especially in datasets where the true number of clusters is large or the components are overlapping. See Figure 2 for an illustration of this. Note that while the standard BIC and AIC formulations in (14) and (15) are designed to be minimized, the `mclust` package in R, which is used for BIC computation throughout this thesis and in Figure 2, reports BIC using an inverted sign (Fraley & Raftery, 2007, p. 5). Specifically, it defines

$$\mathrm{BIC}_{\mathrm{mclust}} = 2 \ln p(\mathbf{X} \mid \hat{\mathbf{p}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) - \kappa \ln(N), \tag{16}$$

such that larger values indicate a better model. This explains why Figure 2 selects $K = 2$ at the maximum BIC value rather than the minimum.

Practical issues also arise when analyzing smaller-sized datasets, where sampling variability can create overlapping confidence intervals in AIC and BIC values across candidate models. This makes it difficult to find a clear local minimum (or maximum in the `mclust` sign convention) when plotting the BIC or AIC values against the number of components K.

Due to these limitations, alternative validation metrics are implemented that

instead rely on cluster geometry, such as compactness and separation, rather than likelihood.

2.1.2.1 Alternative Validation Metrics

The AIC and BIC criteria evaluate models based on their likelihood and complexity, penalizing the number of free parameters to avoid overfitting. While effective, they depend heavily on the likelihood function of the assumed parametric family (e.g., Gaussian distributions). In contrast, the indices introduced in this subsection focus on the geometry of the fitted clusters, measuring properties like within-cluster compactness and between-cluster separation directly from the data. These metrics are model-agnostic, meaning they can still be applied even when the Gaussian assumption is doubtful, and they are more interpretable in settings where cluster geometry matters more than probabilistic fit.

Xie-Beni Index

The *Xie-Beni index* (XB) evaluates cluster quality as a ratio of within-cluster compactness to between-cluster separation (Xie et al., 1991, p. 843). Let N be the total number of observations, K the number of components and $m \geq 1$ a fuzzification parameter controlling the influence of probabilistic assignments. The Xie-Beni index is defined as

$$V_{XB} = \frac{\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}[p(k \mid \mathbf{x}_n)]^m \left\|\mathbf{x}_n - \boldsymbol{\mu}_k\right\|^2}{\min_{i\neq j}\left\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right\|^2} \in [0, \infty), \tag{17}$$

where $p(k \mid \mathbf{x}_n)$ is the membership probability that observation $\mathbf{x}_n$ belongs to component k, and $\boldsymbol{\mu}_k$ is the mean of component k, i.e., the cluster center.

The numerator represents the total fuzzy within-cluster variance (where each squared distance term is weighted by $[p(k \mid x_n)]^m$), while the denominator is the squared distance between the two closest cluster centers. Smaller values of V_{XB} indicate better clustering, with compact clusters and well-separated clusters. The optimal number of components K corresponds to the minimum value of the Xie-Beni index across different K values.

The fuzzifier $m \in [1, \infty)$ controls the degree of fuzziness in the membership weights. Values of m closer to 1 make the clustering assignments harder (more certain), while larger m make them softer (more fuzzy). In Gaussian mixture modeling, soft memberships are inherently probabilistic, with $p(k \mid \mathbf{x}_n)$ summing to 1 for each observation. When applying the XB index to GMMs, it is therefore common to set $m = 1$ and insert those raw probabilities directly without additional fuzzification.

With $m = 1$, the numerator simplifies to the expected squared distance from points to their component means. If one were to choose $m > 1$ on a GMM, the effect would be to make the index more forgiving of ambiguous points (since $p(k \mid \mathbf{x}_n)^m < p(k \mid \mathbf{x}_n)$ for $0 < p(k \mid \mathbf{x}_n) < 1$ when $m > 1$), adding fuzziness to already fuzzy assignments. While this aligns the XB index more closely with

its original fuzzy clustering intent (e.g., Fuzzy C-means that requires $m > 1$ to introduce softness), GMMs do not include a fuzzifier parameter. Thus, $m = 1$ is a natural choice and is used throughout this thesis.

To improve robustness against outliers or unstable, overlapping clusters (e.g., due to random initialization), the minimum in the denominator can be replaced with the median:

$$\text{median}_{i \neq j} \left\| \boldsymbol{\mu}_i - \boldsymbol{\mu}_j \right\|.$$

Silhouette Score

For hard cluster assignments, the *silhouette score* evaluates how similar a point $\mathbf{x}_n$ is to other points in the same cluster (cohesion) versus points in the nearest different cluster (separation). (Rousseeuw, 1987). Let $C_{k(n)}$ denote the cluster assigned to $\mathbf{x}_n$, and let $d(\cdot, \cdot)$ be a distance function (e.g., Euclidean). Then: [1]

$$\begin{aligned} a(n) &= \frac{1}{|C_{k(n)}| - 1} \sum_{\substack{\mathbf{x}_m \in C_{k(n)}, \\ m \neq n}} d(\mathbf{x}_n, \mathbf{x}_m), \\ b(n) &= \min_{j \neq k(n)} \frac{1}{|C_j|} \sum_{\mathbf{x}_m \in C_j} d(\mathbf{x}_n, \mathbf{x}_m). \end{aligned} \tag{18}$$

Here, $a(n)$ is the average distance from $\mathbf{x}_n$ to other points in its own cluster, while $b(n)$ is the smallest average distance to any other cluster. The silhouette value for $\mathbf{x}_n$ is then defined as

$$s(n) = \frac{b(n) - a(n)}{\max\{a(n), b(n)\}} \in [-1, 1], \tag{19}$$

where:

- $s(n) \approx 1$: well assigned, far from neighboring clusters,
- $s(n) \approx 0$: near a cluster boundary,
- $s(n) < 0$: potentially misassigned.

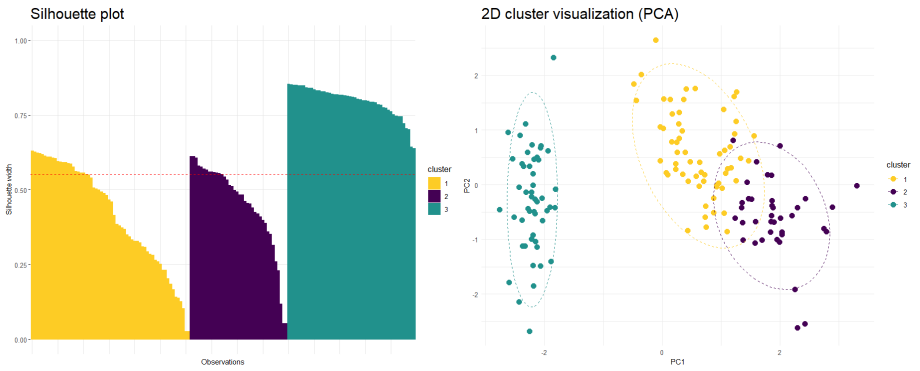
Unlike criteria such as AIC and BIC, that evaluate models globally, the silhouette score provides validation at three levels. Because $s(n)$ is defined for every individual point, one can measure validation element-wise through single $s(n)$ values, cluster-wise by averaging $s(n)$ over all points in cluster k, and globally by averaging over the whole data set.
The global (hard) silhouette score is therefore:

$$V_S = \frac{1}{N} \sum_{n=1}^{N} s(n). \tag{20}$$

Traditionally, silhouette scores are visualized in a *silhouette plot*, where each cluster is represented by a horizontal "blade" sorted by $s(n)$ (Rousseeuw, 1987,

[1] Note: If $|C_{k(n)}| = 1$, $a(n)$ is undefined. In practice, such cases are excluded or assigned $s(n) = 0$.

p. 11). The width of a blade corresponds to cluster size, and the height reflects individual $s(n)$ values. This plot aids in understanding cluster quality, where wide and uniformly high blades indicate well-separated clusters and narrow, or low, blades suggest overlapping clusters or misassignments. See Figure 3 for an example.



(a) Silhouette plot for three clusters.

(b) Cluster assignments projected onto first two principal components with. Ellipses represent 95% confidence regions for each cluster.

Figure 3: Example of silhouette visualization for three clusters. (a) Silhouette plot showing individual silhouette scores $s(n)$ within each cluster. Wider blades and higher values indicate better-defined clusters. The red dashed line represents the average silhouette score across clusters. (b) The same clusters displayed in two-dimensional PCA space for geometric comparison.

Fuzzy Silhouette Score

For soft clustering algorithms like GMM, the standard silhouette score needs adaptation to account for probabilistic cluster memberships. The *fuzzy silhouette score* weighs each $s(n)$ by how confidently the point belongs to its most-likely cluster versus the second-most likely one (Campello and Hruschka, 2006). Let $p_{(1)}(n)$ and $p_{(2)}(n)$ denote the highest and second-highest posterior probabilities for $\mathbf{x}_n$:

$$p_{(1)}(n) = \max_k p(k \mid \mathbf{x}_n), \quad p_{(2)}(n) = \text{second largest } p(k \mid \mathbf{x}_n).$$

Then the fuzzy silhouette score is defined as

$$V_{FS} = \frac{\sum_{n=1}^{N} [p_{(1)}(n) - p_{(2)}(n)]^{\alpha} s(n)}{\sum_{n=1}^{N} [p_{(1)}(n) - p_{(2)}(n)]^{\alpha}}, \tag{21}$$

where $\alpha > 0$ controls how strongly confident assignments are weighted. The default value $\alpha = 1$ is used in this thesis (Campello & Hruschka, 2006, p. 2865).

This score gives more importance to observations with certain cluster assignments. When a point lies "softly" on a boundary ($p_{(1)} \approx p_{(2)}$), it gets a smaller weight and therefore contributes less. Larger values of V_{FS} indicate better separated and more confidently assigned fuzzy clusters.

Model Selection in GMMs
Model selection for Gaussian mixture models typically involves the following procedure:

- Fit GMMs for a range of components K.
- Compute multiple validation metrics, such as $V_{XB}(K)$ and $V_{FS}(K)$.
- Choose the value of K that minimizes V_{XB} and/or maximizes V_{FS}, with preference for V_{XB} for a direct fuzzy measure of compactness / separation.

2.2 Extreme Gradient Boosting (XGBoost)

Gradient boosting is a powerful ensemble method widely used for supervised learning tasks. Among its various implementations, *eXtreme Gradient Boosting* (XGBoost) has become one of the most effective and widely used tools, consistently achieving top performance across applications ranging from regression to classification (Chen & Guestrin, 2016, p. 1). Introduced by Chen and Guestrin (2016), XGBoost improves upon standard gradient boosting through algorithmic enhancements such as regularization, second-order optimization and efficient parallelized computation.

At its core, XGBoost constructs an ensemble prediction model by sequentially adding regression trees. Each tree attempts to correct the prediction error (residuals) of the previously built trees, gradually improving accuracy. The trees predict continuous scores for all tasks, making the model adaptable to both regression and classification through link functions (e.g., sigmoid or softmax).

Let $\{\mathbf{x}_i, y_i\}_{i=1}^N$ represent a dataset with N observations, where each $\mathbf{x}_i \in \mathbb{R}^D$ is a D-dimensional feature vector, and $y_i \in \mathbb{R}$ is the corresponding target value. The ensemble prediction is modeled as the sum of M additive functions

$$\hat{y}_i = \sum_{m=1}^{M} \eta \; t_m(\mathbf{x}_i), \quad t_m \in \mathcal{T}, \tag{22}$$

where each t_m is a regression tree mapping input $\mathbf{x}_i$ to a scalar prediction, and $\mathcal{T}$ is the space of regression trees, i.e. piece-wise-constant functions that partition $\mathbb{R}^D$ into a finite set of leaf regions (Chen & Guestrin, 2016, p. 2). The learning rate $\eta \in [0, 1]$ shrinks each tree's contribution, allowing more trees to be added, reducing the risk of overfitting. This shrinkage technique is standard in boosting literature (Hastie et al., 2001, p. 364).

Each tree assigns an input $\mathbf{x}_i$ to one of its leaves using a function

$$s_m : \mathbb{R}^D \to \{1, 2, ..., L_m\},$$

where L_m is the number of leaves in tree m. The function s_m is the defining tree structure that encodes the decision rules (e.g., "if $x_1 < 0.5$, then left branch") [2]. Each leaf j is associated with an output value $O_j^{(m)} \in \mathbb{R}$, which is the prediction that tree m returns for any observation falling into that leaf. Let $\mathbf{O}^{(m)} = [O_1^{(m)}, ..., O_L^{(m)}]$ denote the vector of all leaf outputs for tree t_m. Thus, the output of the tree is

$$t_m(\mathbf{x}_i) = O^{(m)}_{s_m(\mathbf{x}_i)}. \tag{23}$$

This process is illustrated in Figure 4.

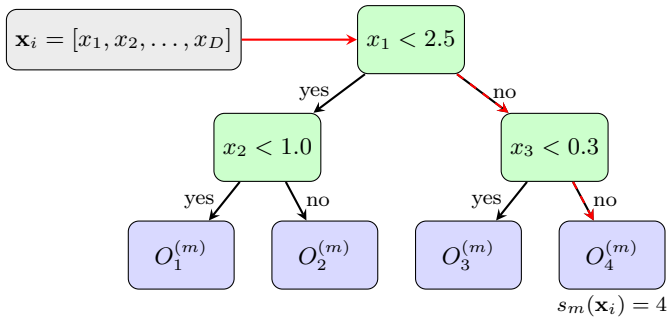


Figure 4: Illustration of a single decision tree in XGBoost. An input instance $\mathbf{x}_i = [x_1, x_2, \ldots, x_D]$ traverses the tree according to split decisions on the input features, as defined by the structure function $s_m(\cdot)$, reaching a leaf node with output value $O_j^{(m)}$.

The model is trained by minimizing a *regularized objective function* that combines the total loss across all training examples

$$\mathcal{L}_{\text{obj}} = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i) + \sum_{m=1}^{M} \Psi(t_m), \tag{24}$$

where ℓ is a twice differentiable convex loss function and $\Psi(t_m)$ is a regularization term that penalizes overly complex trees:

$$\Psi(t_m) = \gamma L_m + \frac{1}{2}\lambda ||O^{(m)}||^2 = \gamma L_m + \frac{1}{2}\lambda \sum_{j=1}^{L_m} \left(O_j^{(m)}\right)^2. \tag{25}$$

The term Ψ consists of two components: (1) a penalty on the number of leaves L_m, controlled by the parameter γ, and (2) an L2 penalty on the size of the leaf outputs, controlled by λ. Together, these components encourage trees to remain shallow and their outputs to stay small unless justified by strong gradients, thus improving generalization.

[2] *Notation:* Boldface lowercase letters such as $\mathbf{x}_i \in \mathbb{R}^D$ denote column vectors; their j-th coordinate is written x_{ij} (or x_j when the instance index i is clear).

2.2.1 Second-Order Objective and Update

To efficiently optimize the objective, XGBoost employs a second-order Taylor expansion of the loss around the current model predictions. This enables the use of both first and second derivatives of the loss, leading to more accurate and stable updates than standard first-order boosting methods.

Let $\hat{y}_i^{(0)}$ denote the initial prediction (typically set to minimize the loss over the entire dataset, such as the mean of the target values for squared error loss or log-odds for logistic loss), and let $\hat{y}_i^{(m)}$ represent the ensemble's prediction after m boosting rounds. The model is constructed additively by sequentially updating predictions based on newly fitted trees

$$\hat{y}_i^{(m)} = \hat{y}_i^{(m-1)} + \eta\ t_m(\mathbf{x}_i). \tag{26}$$

At each iteration, the regularized objective is minimized

$$\mathcal{L}_{\text{obj}}^{(m)} = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i^{(m-1)} + t_m(\mathbf{x}_i)) + \Psi(t_m). \tag{27}$$

This is equivalent to traditional gradient descent, where t_m approximates the negative gradient of the loss, but the regularization term $\Psi(t_m)$ makes XGBoost unique, since most boosting algorithms only minimize the empirical loss (Hastie et al., 2001, p. 358). In (27), each new tree is designed to correct the residual error of the current ensemble. However, directly optimizing (27) for arbitrary loss functions is challenging, as it would require finding the optimal tree structure and leaf values simultaneously. To make the problem manageable, XGBoost applies a second-order Taylor approximation of the loss around the current predictions $\hat{y}_i^{(m-1)}$:

$$\mathcal{L}^{(m)} \approx \sum_{i=1}^{N} \left[\ell(y_i, \hat{y}_i^{(m-1)}) + g_i t_m(\mathbf{x}_i) + \frac{1}{2} h_i t_m^2(\mathbf{x}_i) \right] + \Psi(t_m), \tag{28}$$

where g_i and h_i are the first and second derivatives of the loss w.r.t. $\hat{y}_i^{(m-1)}$:

$$g_i = \frac{\partial \ell(y_i, \hat{y}_i^{(m-1)})}{\partial \hat{y}_i^{(m-1)}} \quad \text{and} \quad h_i = \frac{\partial^2 \ell(y_i, \hat{y}_i^{(m-1)})}{\partial (\hat{y}_i^{(m-1)})^2}. \tag{29}$$

The gradient g_i indicates the direction in which the current prediction should be updated, while the Hessian h_i provides local information on the curvature of the loss function, and thus sets the size of the step taken. In practice, XGBoost assumes that the Hessians of the loss are strictly positive, ensuring convexity and the existence of a unique minimum at each boosting iteration.

Minimizing the quadratic form (28) with respect to the leaf outputs yields the closed-form solution

$$O_j^{*(m)} = -\frac{G_j}{H_j + \lambda}, \qquad G_j = \sum_{i \in I_j} g_i,\ H_j = \sum_{i \in I_j} h_i, \tag{30}$$

and the corresponding tree-structure score

$$\tilde{\mathcal{L}}^{(m)}(s) = -\frac{1}{2}\sum_{j=1}^{L_m}\frac{G_j^2}{H_j+\lambda} \; + \; \gamma L_m. \tag{31}$$

The algebraic derivations are given in Appendix A.2.

Having derived the main elements of the XGBoost framework, the full training procedure can be summarized. Algorithm 1 in Appendix A.3 outlines the high-level steps performed during boosting, while the tree-building method is detailed separately in Section 2.2.2.

2.2.2 Split Evaluation and Gain Function

Searching exhaustively through all possible tree structures is computationally expensive, due to the combinatorial explosion of potential splits. Instead, XGBoost grows trees using a greedy split-finding strategy, evaluating one split at a time. At each node, the algorithm selects the split that yields the greatest reduction in the objective function. This approach is referred to as the *Exact Greedy Algorithm* and is the default when the dataset is small enough to permit evaluating all split candidates efficiently.

Consider a parent node containing a subset of training instances I. If this node is split into two disjoint subsets I_L (left child) and I_R (right child), the *split gain* quantifies the improvement in the regularized objective function (Chen & Guestrin, 2016, p. 3). The gain $\mathcal{G}$ is defined as:

$$\mathcal{G} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right] - \gamma, \tag{32}$$

where G and H denote the total gradients and Hessians in the parent node, and G_L, H_L and G_R, H_R those of the children. The gain expression compares the loss reduction from assigning two separate output values (one for each child) to that of using a single value for the parent node. Subtracting γ reflects the cost of adding a new leaf, which discourages unnecessary growth and ensures each split must yield sufficient improvement to justify the added model complexity.

The gain formula can be interpreted as a generalization of the concept of variance reduction in classic regression tree algorithms. In the case where all h_i's are equal to 1 (as we will see is the case with squared error loss), the gain becomes proportional to the reduction in squared gradients after the split. In other words, the algorithm favors splits that separate the data into regions with more coherent gradients, leading to more confident and effective updates.

To illustrate the gain calculation, consider a node with three training instances $I = \{1, 2, 3\}$, with the following gradients and Hessians:

i	g_i	h_i
1	2.0	1.0
2	1.0	0.5
3	-3.0	2.0

Now, consider a split after instance 2, resulting in two children (see Figure 5):

- Left child $I_L = \{1, 2\}$, with $G_L = 3.0$, $H_L = 1.5$,
- Right child $I_R = \{3\}$, with $G_R = -3.0$, $H_R = 2.0$,
- Parent node: $G = 0$, $H = 3.5$.

With $\lambda = 1$ and $\gamma = 0$, the gain is:

$$\mathcal{G} = \frac{1}{2}\left[\frac{3^2}{1.5+1} + \frac{(-3)^2}{2+1} - \frac{0^2}{3.5+1}\right] = \frac{1}{2}\left[\frac{9}{2.5} + \frac{9}{3}\right] = 3.3.$$

Since the gain is positive, the algorithm would accept this split, assigning separate leaf values to each child node. The opposing gradient signs ($G_L > 0$ vs. $G_R < 0$) suggest a high potential for loss reduction, justifying the split.

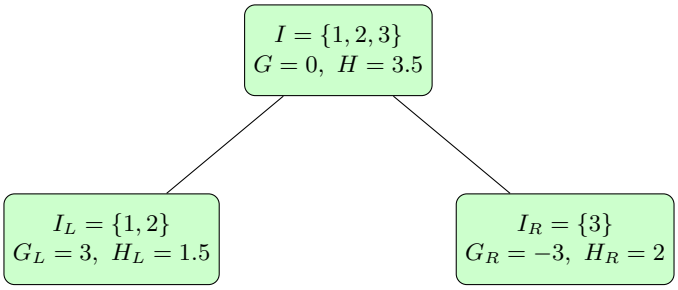


Figure 5: Illustration of a gain-based split. A parent node with three instances $I = \{1, 2, 3\}$ is split into two children. The gain is computed using the aggregated gradients and Hessians of each node.

The above gain computation forms the core of the exact greedy algorithm, outlined in Algorithm 2 in Appendix A.3. At each node, the algorithm searches through all features and split points to find the one that gives the highest gain. If the best gain exceeds zero, the split is applied; otherwise, the node is declared a leaf.

While the exact method is effective on small datasets, it becomes computationally expensive as data size grows. The XGBoost framework therefore includes alternative strategies for split finding:

Approximate algorithm: Proposes candidate splits using weighted percentiles (Hessians as weights) (Chen & Guestrin, 2016, p. 3)

Sparsity-aware algorithm: Handles missing values and sparse input by learning default split directions during training. (Chen & Guestrin, 2016, p. 4)

These alternatives are especially useful when working with large-scale or high-dimensional data. However, for this thesis we focus on the exact greedy approach.

While this algorithm lays the foundation for split finding, its application depends heavily on the choice of loss function. We now examine how these concepts specialize to specific learning objectives, starting with regression.

2.2.3 XGBoost for Regression

For regression tasks with continuous target variables, the most common loss function is the squared error (SE) loss:

$$\ell_{\text{SE}}(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2. \tag{33}$$

The first and second derivatives of this loss function w.r.t. the predicted value are

$$g_i = \frac{\partial}{\partial \hat{y}_i^{(m-1)}} \left[\frac{1}{2}(y_i - \hat{y}_i^{(m-1)})^2 \right] = \hat{y}_i^{(m-1)} - y_i, \tag{34}$$

$$h_i = \frac{\partial^2}{\partial (\hat{y}_i^{(m-1)})^2} \left[\frac{1}{2}(y_i - \hat{y}_i^{(m-1)})^2 \right] = 1. \tag{35}$$

The gradient g_i is simply the residual error, directing the new tree to predict in the opposite direction of the error. The constant Hessian $h_i = 1$ implies that the curvature of the loss function is uniform regardless of the prediction error, meaning all residuals contribute equally in the update step.

With this constant Hessian value, the optimal leaf output simplifies to

$$\begin{aligned} O_j^* &= \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \\ &= \frac{\sum_{i \in I_j} (y_i - \hat{y}_i^{(m-1)})}{\sum_{i \in I_j} 1 + \lambda} \\ &= \frac{\sum_{i \in I_j} (y_i - \hat{y}_i^{(m-1)})}{|I_j| + \lambda}, \end{aligned} \tag{36}$$

where λ is the L2-shrinkage parameter from the regularization term $\Psi(t_m)$ (see (25)). This shows that the optimal leaf prediction is the average residual within that leaf, modified by the regularization term. Here, λ shrinks predictions toward zero when the number of observations $|I_j|$ in the leaf is small (preventing overfitting to sparse data). For larger leaves, the regularization becomes negligible and the output approaches the unregularized average.

For example, consider a leaf with three instances and residuals $\{2, -1, 3\}$, for which the unregularized average is $\frac{2-1+3}{3} \approx 1.33$. However, with $\lambda = 4$, the regularized output becomes $\frac{2-1+3}{3+4} \approx 0.57$, which is significantly closer to zero. In comparison, for a leaf with 100 such residuals summing to 200, the regularized output becomes $\frac{200}{100+4} = 1.92$, very close to the unregularized value of 2.

The corresponding gain formula for the squared error loss simplifies to

$$\mathcal{G}_{SE} = \frac{1}{2}\left[\frac{G_L^2}{|I_L| + \lambda} + \frac{G_R^2}{|I_R| + \lambda} - \frac{G^2}{|I| + \lambda}\right] - \gamma, \tag{37}$$

where $G = \sum_{i \in I}(y_i - \hat{y}_i^{(m-1)})$ is the total residual for each node. This expression reflects the reduction in squared residuals obtained by the split, penalized by γ. It aligns with the traditional variance reduction principle used in regression trees, but with added regularization.

To illustrate, consider the classic example of predicting house prices based on features like size, location and age. Initially, the model might predict the global average price. The first tree would then identify important factors driving price variations, such as location, with leaves containing the average price deviation for houses in different neighborhoods. The following trees would build on this foundation: perhaps the second tree might focus on house size, adjusting predictions differently for small apartments versus large family homes. Each new tree addresses the residual error left by the previous ones, allowing the model to find complex interactions, such as how the impact of an extra bedroom varies between urban and suburban locations.

To avoid overfitting, the parameter λ discourages the model from growing trees that are too specific to the training data. Without such constraints, the model might create separate leaves for homes that differ only in trivial ways, such as the color of the front door.

2.2.3.1 Alternative Loss Functions

While the squared error loss is the most widely used in regression due to its smooth derivatives and computational convenience, other loss functions may be better suited to specific situations. One alternative is the Absolute Error (AE) loss,

$$\ell_{\text{AE}}(y_i, \hat{y}_i) = |y_i - \hat{y}_i|, \tag{38}$$

which is more robust to outliers, as it penalizes large deviations linearly rather than quadratically (Hastie et al., 2001, p. 349). However, AE comes with a challenge: its gradient is undefined at zero and its second derivative is zero almost everywhere (except when $y_i = \hat{y}_i$, then it's undefined). This makes it incompatible with XGBoost's second-order optimization, which relies on nonzero Hessians.

To handle this, recent versions of XGBoost implement a *line search* strategy when using absolute error as the loss function (Developers, 2022). After building each tree, the algorithm chooses a step size η that minimizes the loss in the tree's predicted direction. This avoids issues with division by zero and ensures that the training procedure still converges, although typically more slowly than with second-order methods (Developers, 2022). Line search strategies are used frequently in optimization when second-order derivatives do not exist or are unreliable (Boyd et al., 2004, Sec. 9.2; Wright & Wright, 2018, p. 280; Wright & Nocedal, 2006, Algorithm 3.5).

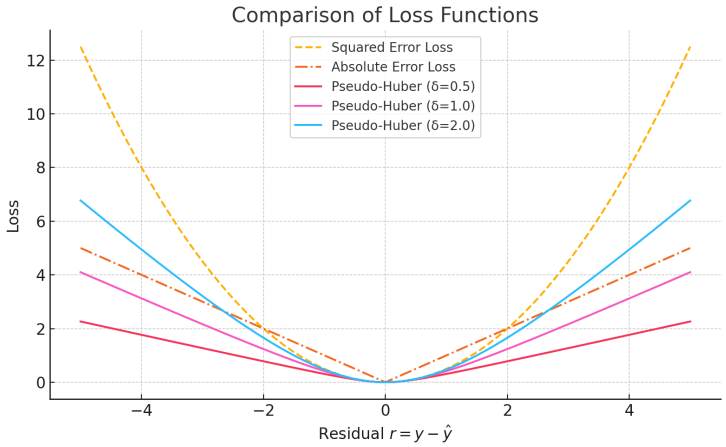


Figure 6: Comparison of squared error loss, absolute error loss and pseudo-Huber loss for different values of the transition parameter δ. Squared loss grows quadratically and is highly sensitive to outliers, while the absolute loss grows linearly and is more robust. The pseudo-Huber loss smoothly interpolates between the two, behaving quadratically near zero and linearly for large residuals.

A smoother alternative is the *Pseudo-Huber* loss, which behaves like a squared error loss near zero and like AE for large residuals, while remaining fully differentiable (Guo et al., 2024, p. 2):

$$\ell_{\text{PH}}(y_i, \hat{y}_i) = \delta^2 \left(\sqrt{1 + \left(\frac{y_i - \hat{y}_i}{\delta} \right)^2} - 1 \right), \tag{39}$$

where $\delta > 0$ controls the transition from quadratic to linear behavior. This loss is compatible with second-order boosting, as its Hessian is always positive:

$$\frac{\partial^2 \ell_{\text{PH}}}{\partial \hat{y}_i^2} = \frac{\delta^2}{(\delta^2 + (y_i - \hat{y}_i)^2)^{3/2}} > 0. \tag{40}$$

Figure 6 compares these three loss functions.

The ideas behind gradient boosting carry over naturally from regression to classification, with a few modifications to the loss function and its derivatives. We now turn to the binary classification setting.

2.2.4 XGBoost for Classification

In binary classification, the model produces continuous predictions $\hat{y}_i$ that are transformed into class probabilities. This is typically achieved using the logistic sigmoid function

$$\pi_i = \frac{1}{1 + e^{-\hat{y}_i}}, \tag{41}$$

where π_i is the predicted probability that instance i belongs to the positive class ($y_i = 1$). The standard loss function in this setting is the binary logistic (log-loss, LL) function

$$\ell_{\text{LL}}(y_i, \hat{y}_i) = -\left[y_i \ln \pi_i + (1 - y_i) \ln(1 - \pi_i)\right], \tag{42}$$

where $y_i \in \{0, 1\}$ is the true class label. Substituting π_i into the loss gives

$$\begin{aligned} \ell_{\text{LL}}(y_i, \hat{y}_i) &= -\left[y_i \ln\left(\frac{1}{1 + e^{-\hat{y}_i}}\right) + (1 - y_i) \ln\left(\frac{e^{-\hat{y}_i}}{1 + e^{-\hat{y}_i}}\right)\right] \\ &= y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i)\left[\hat{y}_i + \ln(1 + e^{-\hat{y}_i})\right] \\ &= \ln(1 + e^{-\hat{y}_i}) + (1 - y_i)\hat{y}_i. \end{aligned} \tag{43}$$

Alternatively, by switching signs and combining terms, the loss can also be written as

$$\ell_{\text{LL}}(y_i, \hat{y}_i) = \ln(1 + e^{\hat{y}_i}) - y_i \hat{y}_i. \tag{44}$$

This equivalent form is preferred for differentiation, and it also shows the asymmetric behavior of the loss: it penalizes false positives ($y_i = 0$ but $\hat{y}_i \gg 0$) exponentially, while penalizing false negatives ($y_i = 1$ but $\hat{y} \ll 0$) approximately linearly.

To fit a new tree at boosting step m, the first and second derivatives of the loss with respect to the current prediction $\hat{y}_i^{(m-1)}$ are computed. Using

$$\pi_i^{(m-1)} = \frac{1}{1 + e^{-\hat{y}_i^{(m-1)}}},$$

we get

$$g_i = \frac{\partial \ell_{\text{LL}}(y_i, \hat{y}_i^{(m-1)})}{\partial \hat{y}_i^{(m-1)}} = \pi_i^{(m-1)} - y_i, \tag{45}$$

$$h_i = \frac{\partial^2 \ell_{\text{LL}}(y_i, \hat{y}_i^{(m-1)})}{\partial (\hat{y}_i^{(m-1)})^2} = \pi_i^{(m-1)}(1 - \pi_i^{(m-1)}) \tag{46}$$

The gradient is the difference between the predicted probability and the actual class label, representing the classification error. It is positive when the model is overestimating the probability of class 1 (e.g., predicting $\pi_i = 0.9$ when $y_i = 0$), and negative when it is underestimating it (e.g., $\pi_i = 0.2$ when $y_i = 1$). A gradient of zero indicates a perfect prediction. Further, the Hessian reaches its maximum when the prediction is most uncertain ($\pi_i = 0.5$), leading to

conservative updates, and decreases as the prediction becomes more confident in either direction (i.e., it approaches 0 as $\pi_i \to 0$ or 1).

Following the same procedure as for regression, the optimal output value for a leaf becomes

$$O_j^{*(m)} = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} = -\frac{\sum_{i \in I_j} (\pi_i^{(m-1)} - y_i)}{\sum_{i \in I_j} \pi_i^{(m-1)}(1 - \pi_i^{(m-1)}) + \lambda}, \tag{47}$$

where λ is the same L2-shrinkage constant that appears in the regularization term $\Psi(t_m)$ (see (25)). The gain formula is obtained by inserting the logistic-loss derivatives into the generic split-gain expression derived in Section 2.2.2, Eq. (32):

$$\mathcal{G}_{\text{LL}} = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma, \tag{48}$$

where again $g_i = \pi_i^{(m-1)} - y_i$ and $h_i = \pi_i^{(m-1)}(1 - \pi_i^{(m-1)})$.

As before, the gain measures how well a split aligns gradients in opposite directions between the left and right children. A high gain often occurs when one subset contains mostly positive examples and the other mostly negative, indicating a clear decision boundary.

2.3 Explainable Machine Learning

What if your loan was denied, your job application filtered out, or your medical treatment plan altered by an algorithm you couldn't question? As algorithms become more involved in decisions that affect our lives, it's no longer enough for models to be accurate. They must also be understandable (Lundberg & Lee, 2017, p. 1).

Ensemble methods like boosted trees often function as black boxes, making it difficult to trace how specific input features influence the outcome. These models often sacrifice transparency for performance; a trade-off that becomes problematic in any setting where decisions need to be justified (Molnar, 2019, p. 15). Interpretability isn't just a convenience - it's essential for trust, fairness and accountability (Doshi-Velez & Kim, 2017, p. 2).

This is the main motivation behind the field of *explainable machine learning* (XML), that is, to make models more transparent and trustworthy by identifying *how* and *why* a prediction was made.

2.3.1 Shapley Values

A widely used tool in XML is the *Shapley value*, a concept borrowed from cooperative game theory and introduced by Lloyd Shapley (1953). It provides a way to assign importance to each feature in a prediction, based on their marginal contributions across all possible feature combinations, or *coalitions*.

To illustrate, imagine a streaming service that predicts how much a user will enjoy the new Star Wars movie. The average predicted enjoyment across all users is 60%, but for a specific user, the model outputs 90%. The input features for this user include:

- Age: 26 years
- Favorite genre: Sci-Fi
- Recently watched: 5 comedy movies
- Time of day: Evening
- Subscription level: Premium

The question becomes: why was this particular prediction so high? Is it because they love Sci-Fi? Does it matter that they recently watched five movies in a row? More generally, how can the individual contribution of each feature be fairly and consistently measured?

The difficulty in answering that question is that in many ML models, especially nonlinear ones, features do not act independently. Their combined effect may not equal the sum of their individual effects. For example, "Favorite genre: Sci-Fi" might only push the prediction higher when paired with "Time of day: Evening", which would be an interaction that a linear model would miss unless explicitly encoded.

Shapley values address this issue. They aim to distribute the difference between a specific prediction and the average (or *baseline*) prediction across all features (Molnar, 2019, p. 177). The idea is to evaluate every possible coalition (i.e., subset of features) and measure how the prediction changes as each feature joins the group. By doing so, the full prediction gap (e.g., $90\% - 60\% = 30\%$) is fairly distributed among the input features, taking into account interactions and avoiding arbitrary assumptions. A great advantage is that this approach is model-agnostic, as it can be applied to any predictive model regardless of its architecture, from simple linear regressions to complex neural networks.

Worth noting is that the baseline for comparison does not need to be the global average prediction. Instead, Shapley values can be computed relative to any reference point, allowing us to ask not just "why is this prediction high?", but also "why is it different from another specific instance?".

Mathematical Definition

In cooperative game theory, the setup is around a game and players. A prediction for an instance $\mathbf{x} \in \mathbb{R}^D$ is interpreted as a "payout" arising from a collaborative game played by the D input features. Each feature x_j is a "player", and the model's prediction $f(\mathbf{x})$ is the total reward to be distributed (Molnar, 2019, p. 178). The goal is to determine how much of the excess prediction $f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})]$ is due to each feature x_j alone.

Let $\mathcal{D} = \{1, 2, ..., D\}$ be the index set of all features. For any subset $S \subseteq \mathcal{D} \setminus \{j\}$, the marginal contribution of feature x_j is the change in prediction when

feature j is added to the subset S (Hart, 1989, p. 211):

$$\Delta_j(S) = v(S \cup \{j\}) - v(S), \tag{49}$$

where $v(S)$ denotes the expected model output when only the features in S (written $\mathbf{x}_S$) are known. The Shapley value $\varphi_j(\mathbf{x})$ is then defined as the average marginal contribution of feature j over all possible subsets $S \subseteq \mathcal{D} \setminus \{j\}$ (Shapley, 1953, pp. 311–312):

$$\varphi_j(\mathbf{x}) = \sum_{S \subseteq \mathcal{D} \setminus \{j\}} \frac{|S|!(D - |S| - 1)!}{D!} [v(S \cup \{j\}) - v(S)]. \tag{50}$$

This expression assumes that all permutations of feature orderings are equally likely, weighting each marginal contribution accordingly (Hart, 1989, p. 211). The combinatorial weight $\frac{|S|!(D-|S|-1)!}{D!}$ corresponds to the number of permutations in which the subset S appears before feature j, normalized by the total number of permutations. In each such ordering, the marginal contribution of j to coalition S is computed and then averaged. By averaging, the calculation ensures equal treatment for features with identical impact and fairness (no coalition bias).

In practice, the value function $v(S)$ is estimated by marginalizing over the unknown features (Lundberg & Lee, 2017, p. 5):

$$v(S) = \mathbb{E}[f(\mathbf{x}) \mid \mathbf{X}_S = \mathbf{x}_S]. \tag{51}$$

That is, the remaining features $\mathbf{x}_{\mathcal{D} \setminus S}$ are integrated out, often under the assumption of independence or using empirical data distributions (Molnar, 2019, pp. 183–184). While the assumption of independence simplifies computation, it can introduce bias in scenarios where features are correlated. The resulting explanations could lead to misinterpretations when analyzing the results (Aas et al., 2021, p. 2).

For example, say Sci-Fi preference (x_1) and Evening viewing (x_2) are strongly correlated. Marginalizing over x_2 as if it were independent might over- or underestimate x_1's contribution, since $\mathbb{E}[f(\mathbf{x}) \mid \mathbf{x}_1]$ would sample unrealistic combinations (e.g., Sci-Fi lovers who never watch movies in the evening).

The expectation in (51) can be taken under different distributions for the remaining features; Section 2.3.2.2 makes this explicit for tree models.

Further, the Shapley value is the only solution that satisfies four axioms that formalize what is considered a fair distribution (Shapley, 1953, pp. 309, 312; Molnar, 2019, p. 184):

- **Symmetry:** If two features contribute equally in every coalition, they get the same value, i.e.,

$$\text{If } v(S \cup \{j\}) = v(S \cup \{k\}) \text{ for all } S \subseteq \mathcal{D} \setminus \{j, k\}, \text{ then } \varphi_j = \varphi_k. \tag{52}$$

- **Efficiency:** The sum of Shapley values equals the total difference:

$$\sum_{j=1}^{D} \varphi_j(\mathbf{x}) = f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})]. \tag{53}$$

- **Additivity:** The Shapley values for a sum of models equal the sum of the Shapley values from each model, i.e.,

$$\text{For two models f and g, } \varphi_j^{(f+g)} = \varphi_j^{(f)} + \varphi_j^{(g)}. \tag{54}$$

- **Dummy:** A feature that does not affect the prediction in any subset gets a value of zero.

$$\text{If } v(S \cup \{j\}) = v(S) \text{ for all } S \subseteq \mathcal{D}, \text{ then } \ \varphi_j = 0. \tag{55}$$

A helpful intuition for understanding Shapley values is to imagine features entering a room in random order (Molnar, 2019, p. 185). Initially, the room is empty and the prediction equals the baseline $\mathbb{E}[f(\mathbf{x})]$ (60% in the movie streaming example). As each feature x_j enters the room, it contributes to shifting the prediction from this average by an amount $\Delta_j(S) = v(S \cup \{j\}) - v(S)$, where S is the subset of features entered previously. The Shapley value for a feature is its average contribution across all possible entry orders. This ensures that the assignment is fair regardless of the order in which features are considered.

Figure 7 demonstrates this for our streaming model:

- Baseline (60%): Empty room ($S = \emptyset$, no features).
- First feature (Sci-Fi genre): Enters and contributes +15% (from 60% to 75%), reflecting its marginal contribution when added to $S = \emptyset$
- Second feature (Evening time): Increases the prediction further by +10% (to 85%), but this value accounts for its interaction with Sci-Fi.
- Third feature (Age 26): Adds a final +5%, which shows how later features may have diminishing marginal contributions.

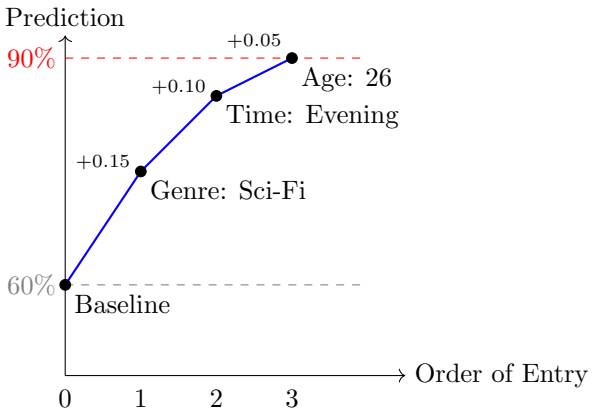


Figure 7: Shapley value intuition: features enter one by one and shift the prediction from the baseline. Contributions are averaged over all orderings.

The dashed lines in Figure 7 emphasize that the total prediction shift (30%) equals the sum of all Shapley values $\sum_{j=1}^{D} \varphi_j = f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})] = 90\% - 60\% = 30\%$, satisfying the efficiency axiom. The weights $\frac{|S|!(D-|S|-1)!}{D!}$ ensure symmetry: for example, if Sci-Fi and Evening contributed equally in all coalitions, they would receive identical values regardless of entry order.

2.3.1.1 Computation and Approximation

Computing exact Shapley values requires evaluating all possible subsets of features, which is exponential in the number of features. For a model with D features, there are 2^{D-1} possible coalitions per feature, making exact computation very expensive for high-dimensional datasets. In total, computing all Shapley values requires evaluating $D \cdot 2^{D-1}$ times, once for each coalition per feature. This exponential scaling makes the approach impractical for models with more than a dozen features unless structure-specific optimizations are used.

To address this challenge, many methods have been proposed, with *SHapley Additive exPlanations* (SHAP) being one of the more common ones. As will be discussed in Section 2.3.2, the SHAP framework introduces optimizations that enable efficient computation of Shapley values for certain model types, such as decision trees.

2.3.2 SHapley Additive exPlanations (SHAP)

While Shapley values offer a theoretically plausible method for attributing feature contributions to individual predictions, they are rarely practical to compute directly. As seen in Section 2.3.1, their implementation faces several challenges, including computational complexity and the handling of feature dependencies. SHapley Additive exPlanations (SHAP), developed by Lundberg and Lee (2017), solves these problems by combining multiple explanation methods into a single framework that keeps the important benefits of Shapley values (Lundberg and Lee, 2017).

2.3.2.1 Mathematical Framework

SHAP builds upon the game-theoretic foundation of Shapley values, but adds practical improvements to work faster and with more types of ML models. Instead of interpreting the complex original model directly, SHAP constructs a simplified *surrogate explanation model* – typically a linear model that approximates the original model's behavior for a specific prediction (Molnar, 2019, p. 163) [3]. This surrogate is designed to be both additive (where each feature's contribution can be summed) and locally accurate (faithful to the original model's output for that particular input) (Lundberg & Lee, 2017, p. 2). The Shapley values then become the coefficients of this interpretable surrogate

[3] Molnar's ironic observation expresses this well: *"Solving machine learning interpretability by using more machine learning!"* (Molnar, 2019, p. 163).

model, representing how much each feature contributes to the deviation from a baseline prediction.

To formalize this, let $f(\mathbf{x})$ be the original prediction model, say a trained XGBoost regression model. The surrogate captures how the original model behaves around a specific input, using a *coalition vector* $\mathbf{z}' \in \{0,1\}^D$ to represent which features are "present" ($z'_j = 1$) or "absent" ($z'_j = 0$) (Lundberg & Lee, 2017, p. 2; Molnar, 2022a). Since most models cannot handle "missing" features directly, SHAP uses a mapping function $h_{\mathbf{x}} : \{0,1\}^D \to \mathbb{R}^D$ that replaces absent features ($z'_j = 0$) with a value from a reference distribution, usually the marginal distribution of the dataset (e.g., mean/median) (Lundberg & Lee, 2017, p. 2). For present features, $h_{\mathbf{x}}$ simply uses the original value x_j:

$$h_{\mathbf{x}}(\mathbf{z}')_j = \begin{cases} x_j & \text{if } z'_j = 1 \quad \text{(feature present)}, \\ \mathbb{E}[x_j] \text{ or } x_{j,\text{ref}} & \text{if } z'_j = 0 \quad \text{(feature absent)}, \end{cases} \tag{56}$$

where x_j is the j-th coordinate of the explained instance $\mathbf{x}$. Evaluating the original model on a coalition vector $\mathbf{z}'$ defines the *coalition game*

$$g_{\mathbf{x}}(\mathbf{z}') \;=\; f\big(h_{\mathbf{x}}(\mathbf{z}')\big), \tag{57}$$

such that $g_{\mathbf{x}} : \{0,1\}^D \to \mathbb{R}$ maps coalitions to model outputs. This mapping is necessary since most models require complete input vectors, so some strategy is needed to impute or marginalize over missing features for SHAP to compute feature contributions (see Figure 8).

SHAP then constructs a surrogate model, constrained by the Shapley axioms, that matches the coalition game wherever the axioms allow. Constraining the surrogate to be linear in the presence indicators yields (Lundberg & Lee, 2017, p. 2)

$$\hat{f}_{\text{SHAP}}(\mathbf{z}') = \underbrace{\varphi_0}_{\text{baseline}} + \sum_{j=1}^{D} \varphi_j z'_j, \tag{58}$$

where φ_j are the Shapley values and $\varphi_0 = \mathbb{E}[f(\mathbf{X})]$ is the baseline prediction. The Shapley axioms make the coefficients φ_j the unique solution satisfying

$$\hat{f}_{\text{SHAP}}(\mathbf{z}') = g_{\mathbf{x}}(\mathbf{z}') \quad \text{for all } \mathbf{z}' \in \{0,1\}^D \tag{59}$$

if and only if the coalition game $g_{\mathbf{x}}$ is itself linear in $\mathbf{z}'$ (e.g., when the original model is additive in the input features). For nonlinear models such as XGBoost, SHAP guarantees an exact match at $\mathbf{z}' = \mathbf{1}$ (as seen in the local accuracy property introduced below) and provides the unique Shapley-consistent linear approximation elsewhere.

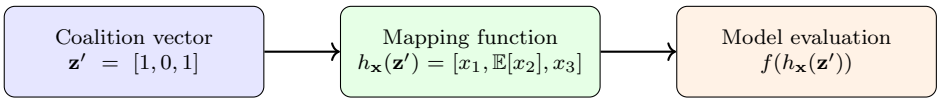


Figure 8: SHAP evaluates the model on synthetic inputs constructed from feature coalitions. The coalition vector $\mathbf{z}' = [1, 0, 1]$ includes features 1 and 3 but excludes feature 2. The mapping function $h_{\mathbf{x}}$ replaces the missing x_2 with its expected value $\mathbb{E}[x_2]$, allowing evaluation of the original model f.

The Shapley values are estimated by computing the average marginal contribution of each feature j across all coalitions in Equation (50). Because exact evaluation is exponential in D, SHAP relies on faster algorithms:

- **KernelSHAP** (model-agnostic): approximates Shapley values by solving a weighted least-squares problem over sampled coalitions. It uses a kernel that assigns higher weights to very small or very large coalitions, ensuring the solution converges to the true Shapley values in expectation (Lundberg & Lee, 2017, p. 6; Molnar, 2022a).
- **TreeSHAP** (decision-tree models): uses tree structure to compute exact Shapley values (Lundberg et al., 2020).

SHAP extends beyond just approximating Shapley values; it reframes them as *local explanations* of individual predictions. To ensure that these explanations are meaningful and fair, SHAP imposes three axiomatic properties that the explanation model $\hat{f}_{\text{SHAP}}$ must satisfy. Let $f(\mathbf{x})$ again denote the original model prediction (e.g., XGBoost regression). The following properties are then required (Lundberg & Lee, 2017, p. 4; Molnar, 2022a; Lundberg et al., 2020, p. 64):

1. Local accuracy

$$\hat{f}_{\text{SHAP}}(\mathbf{z}' = \mathbf{1}) = f(\mathbf{x}) \quad \Rightarrow \quad f(\mathbf{x}) = \varphi_0 + \sum_{j=1}^{D} \varphi_j \tag{60}$$

This property ensures that when all features are present ($\mathbf{z}' = 1$), the explanation model exactly reproduces the prediction of the original model. In other words, the SHAP values φ_j together explain the entire deviation of the prediction from the baseline

$$f(\mathbf{x}) - \mathbb{E}[f(\mathbf{X})] = \sum_{j=1}^{D} \varphi_j. \tag{61}$$

This mirrors the efficiency axiom (see Equation (53)) of Shapley values and guarantees that the explanation is "complete", i.e., no part of the prediction is left unexplained (Molnar, 2022a).

2. Missingness
Missingness states that a feature which is not included in the coalition ($z'_j = 0$) must receive an attribution of zero:

$$z'_j = 0 \Rightarrow \varphi_j = 0. \tag{62}$$

In other words, absent features contribute nothing to the explanation. This aligns with our intuitive expectation: if a feature was not in the surrogate model input, then it should not be credited for the resulting prediction.

To clarify further, SHAP distinguishes between a feature being absent and a feature having a value of zero. The coalition vector $\mathbf{z}' \in \{0,1\}^D$ is used to encode presence or absence, where $z'_j = 0$ denotes that feature j is missing and its value is replaced via the mapping function $h_{\mathbf{x}}(\mathbf{z}')$. This property ensures that the explanation respects this absence. If a feature is not included in the model input, its Shapley value must be zero[4].

3. Consistency
If the marginal contribution of a feature increases (or stays the same) in a new model, then its SHAP value should not decrease (Lundberg & Lee, 2017, p. 4). More formally, let $f_x(\mathbf{z}') = f(h_{\mathbf{x}}(\mathbf{z}'))$ be the model output when evaluated on the coalition $\mathbf{z}' \in \{0,1\}^D$, and let $\mathbf{z}' \setminus j$ denote a copy of $\mathbf{z}'$ where feature j has been removed (i.e., $z'_j = 0$). Then, for any two models f and f', if

$$f'_x(\mathbf{z}') - f'_x(\mathbf{z}' \setminus j) \geq f_x(\mathbf{z}') - f_x(\mathbf{z}' \setminus j) \quad \text{for all } \mathbf{z}' \in \{0,1\}^D, \tag{63}$$

then the SHAP value for feature j must not decrease

$$\varphi_j^{(f')} \geq \varphi_j^{(f)}. \tag{64}$$

This formalizes the intuition that if a feature becomes more important in a new model, its attribution (SHAP value) in the explanation should reflect that increase (Molnar, 2022a, Section. SHAP theory). Consistency guarantees that the feature's SHAP value under the new model will be at least as large as under the original model. In simpler terms:

- If a feature becomes more important, its SHAP value doesn't shrink.
- If a feature stays equally important, its SHAP value doesn't change (assuming all other marginal contributions unchanged).

A simple analogy: Imagine two chefs f and f', who cook using the same ingredients (features). Chef f' relies more heavily on salt (feature j) than Chef f in every dish, no matter the combinations of other ingredients. The consistency

[4] Although the original SHAP paper uses $x'_j = 0 \Rightarrow \varphi_j = 0$ to state this property (Lundberg & Lee, 2017, Equation 6), this is slightly misleading in notation. As clarified in the *Interpretable Machine Learning* book (Molnar, 2022a, Section. SHAP theory), the value x'_j in that context refers to a binary coalition indicator - what we denote here as z'_j - not the actual feature value $x_j \in \mathbb{R}$.

property ensures that salt's SHAP value under Chef f' will be greater than or equal to its SHAP value under Chef f.

Although SHAP values are mathematically equivalent to Shapley values, their interpretation and computation are grounded in model behavior. The following example revisits the earlier movie recommendation scenario (see Section 2.3.1), now illustrating both SHAP computation and its axiomatic properties.

Consider a model predicting how much a user will enjoy a movie using these two binary features:

- Genre: Sci-Fi ($x_1 = 1$ if favorite, else 0)
- Time: Evening ($x_2 = 1$ if watching in the evening, else 0).

For a user $\mathbf{x} = (1, 1)$, the model predicts $f(\mathbf{x}) = 0.90$ and baseline $\mathbb{E}[f(\mathbf{X})] = 0.60$. SHAP evaluates the model on synthetic inputs formed from all feature coalitions as follows:

Coalition S	**Mapped Input** $h_{\mathbf{x}}(S)$	**Model Output** $f(h_{\mathbf{x}}(S))$
$\emptyset$	$[\mathbb{E}[x_1], \mathbb{E}[x_2]]$	0.60
{Sci-Fi}	$[1, \mathbb{E}[x_2]]$	0.75
{Evening}	$[\mathbb{E}[x_1], 1]$	0.70
{Sci-Fi, Evening}	$[1, 1]$	0.90

The four coalition payouts above are inputs to the Shapley formula (Equation (50)) for computing the exact Shapley values, and should not be confused as four independent equations to be solved. For $D = 2$, the Shapley formula uses all four coalitions with the appropriate combinatorial weights $\frac{|S|!(D-|S|-1)!}{D!}$, avoiding the need to solve a linear system. The efficiency axiom then ensures $\varphi_1 + \varphi_2 = f(\mathbf{x}) - \varphi_0$, which follows from the formula rather than acting as an extra algebraic constraint. Hence there is no conflict between the four coalition values and the three unknown φ's.

Now, using Equation (50), the resulting Shapley values are:

$$\varphi_{\text{Sci-Fi}} = \frac{(0.75 - 0.60) + (0.90 - 0.70)}{2} = 0.175,$$
$$\varphi_{\text{Evening}} = \frac{(0.70 - 0.60) + (0.90 - 0.75)}{2} = 0.125.$$

One might wonder why we use Eq. (50) rather than Eq. (58)? It may seem reasonable to treat $\hat{f}_{\text{SHAP}}(\mathbf{z}') = \varphi_0 + \sum_{j=1}^{D} \varphi_j z'_j$ as a system of equations and solve for φ_j directly by using coalition payouts. However, this approach fails for two reasons. First, with D features there are $D + 1$ unknowns but 2^D coalitions, creating an over-determined system. Even if only $D + 1$ coalitions are used (e.g., the baseline and the D single-feature coalitions), the resulting coefficients violate the local accuracy axiom. Second, unweighted least-squares solutions using (58) ignore the combinatorial fairness of Shapley values, leading to violations of efficiency ($\sum \varphi_j \neq f(\mathbf{x}) - \varphi_0$) or symmetry.

The Shapley formula solves both these issues by (i) weighting every coalition with the combinatorial weights, derived from the fairness axiom, and (ii) enforcing efficiency so that the coefficients sum to the exact prediction gap. In practice, algorithms such as KernelSHAP and TreeSHAP replicate the same result by solving a weighted linear system whose kernel reproduces these weights. Hence, Eq. (50) is used and not Eq. (58) to compute φ_j in the example.

The three SHAP properties can now be verified:

Local accuracy:
The SHAP values together explain the difference between the prediction and baseline, such that

$$f(\mathbf{x}) = \mathbb{E}[f(\mathbf{X})] + \varphi_{\text{Sci-Fi}} + \varphi_{\text{Evening}} = 0.60 + 0.175 + 0.125 = 0.90. \quad \checkmark$$

Missingness:
If the coalition excludes *Evening* ($z'_2 = 0$), the input becomes $[1, \mathbb{E}[x_2]]$ and the model output is 0.75. Even though the actual input has $x_2 = 1$, the SHAP value for *Evening* is set to zero in this coalition: $\varphi_{\text{Evening}} = 0$. $\checkmark$

Consistency:
Now suppose a new model f' increases the contribution of *Sci-Fi* in all coalitions, for example

$$f'(1, \mathbb{E}[x_2]) - f'(0, \mathbb{E}[x_2]) = 0.20 > 0.15.$$

Then by consistency,

$$\varphi^{(f')}_{\text{Sci-Fi}} \geq \varphi^{(f)}_{\text{Sci-Fi}} = 0.175. \quad \checkmark$$

Still, computing exact SHAP values remains computationally expensive, especially for complex models. In the next section, we introduce TreeSHAP, an efficient algorithm that enables exact SHAP value computation for tree-based models such as XGBoost.

2.3.2.2 TreeSHAP

As discussed in Section 2.3.1, evaluating all 2^D coalitions quickly becomes intractable as the number of features grows. *TreeSHAP*, introduced by Lundberg et al. (2020), addresses this bottleneck for tree-based models such as decision trees, random forests and gradient-boosted ensembles. It is a model-specific algorithm that uses the tree structure to compute exact SHAP values in polynomial time (Lundberg et al., 2020, p. 56). This means that the computation time grows as a polynomial function of model size, rather than exponentially with the number of features. Here, model size refers to the number of trees M, the maximum tree depth Υ and the number of leaves L. As discussed below, TreeSHAP's complexity depends polynomially on these parameters, avoiding the exponential dependence on the feature count D as in the case with previous

exact Shapley computations. Consequently, TreeSHAP can be applied to even large datasets with many dimensions.

Notably, TreeSHAP satisfies the same three SHAP axioms (local accuracy, missingness and consistency) while being an order of magnitude faster than model-agnostic alternatives.

There are two main variants of TreeSHAP: *interventional* and *tree-path dependent* (Lundberg & Lee, 2017, p. 65) (Molnar, 2022a, Section. TreeSHAP). The interventional variant computes classical Shapley values by marginalizing over a background dataset, by assuming independence between features. In contrast, the tree-path dependent method uses conditional expectations inferred from the tree's split structure, which preserves dependencies found during training.

While the interventional approach aligns more closely with causal interpretations, the path-dependent variant is computationally faster and more faithful to the model's internal logic. In this thesis, we focus exclusively on the tree-path dependent method, which is natively implemented in the `xgboost` R-package used in the analysis.

Tree-based models are made up of piecewise constant functions, where each input $\mathbf{x} \in \mathbb{R}^D$ follows a unique path from root to leaf, and each leaf outputs a fixed value. The idea of TreeSHAP is to compute, for each feature, the expected change in model output when that feature is excluded from the input—that is, when it is treated as "missing" from the coalition.

When a feature is present, TreeSHAP follows the actual path that the instance $\mathbf{x}$ takes through the tree. On the other hand, when a feature is considered missing (i.e., not in the coalition), TreeSHAP does not follow the actual decision path for that feature. Instead, it branches at every split node involving that feature and computes a weighted average over both left and right subtrees. These weights, or probabilities, assigned to each branch are determined by the training data *coverage*. The coverage (also called cover) is the proportion of training instances that followed each path during training.

For example, if a split on feature x_j sent 70% of the training samples to the left and 30% to the right, TreeSHAP uses these as probabilities when computing the expected output in the absence of x_j (0.7 to left and 0.3 to the right), see Figure 9. These coverage statistics are recorded during training of the tree model and are used as path weights for simulating feature splits. This allows TreeSHAP to compute the expectations analytically, without requiring sampling or external reference data.

How TreeSHAP works

Recall the generic Shapley value function in (51). It can be rewritten as

$$v(S) = \mathbb{E}\big[f(\mathbf{x}_S, \mathbf{X}_{S^C}) \mid \mathbf{X}_S = \mathbf{x}_S\big], \tag{65}$$

where S^C is the complement of S, and the expectation is taken over the features that are *not* in the coalition S. For the path-dependent TreeSHAP variant we specialize this expectation to the empirical conditional distribution encoded by the tree structure:

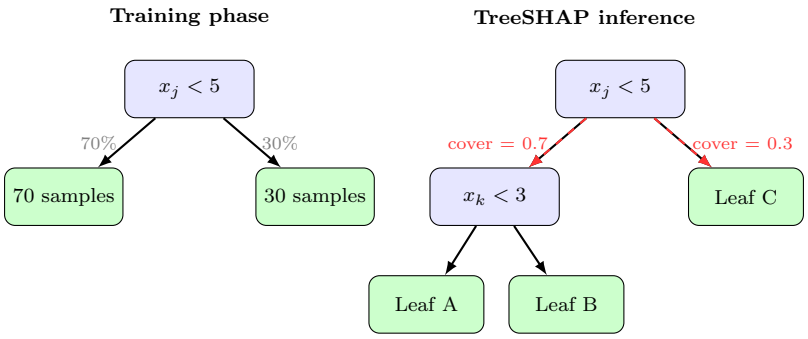


Figure 9: TreeSHAP explanation using training-derived coverages. Left: During training, a split on feature x_j sends 70% of samples left and 30% right. These proportions are recorded as coverage statistics. Right: When x_j is present in a coalition, TreeSHAP follows a single path (normal execution). When x_j is missing from the coalition, TreeSHAP follows both paths simultaneously, using the training coverage proportions (70%/30%) as weights to compute the expected model output.

$$v_{\text{path}}(S) = \mathbb{E}_{\mathbf{X}_{S^C} \sim P\left(\mathbf{X}_{S^C} | \mathbf{X}_S = \mathbf{x}_S,\ \text{tree path}\right)} \Big[f(\mathbf{x}_S, \mathbf{X}_{S^C}) \Big]. \tag{66}$$

The conditional density $P(\mathbf{X}_{S^C} \mid \mathbf{X}_S, \text{tree path})$ is approximated by the cover statistics recorded during training. Cover thus provides an empirical estimate of the conditional probability that a held-out instance would traverse each branch. The goal of TreeSHAP is to approximate (66).

Now, for a single decision tree, TreeSHAP traverses the tree recursively. At each node, it checks whether the splitting feature is in the current coalition S:

- If *present*, the traversal continues down the path determined by $\mathbf{x}$.
- If *absent*, both subtrees are explored, and weighted by training data coverage.

During this traversal, the algorithm saves a *path object*, γ, that keeps track of current coalition features S, the weighted probability of reaching the current node, and the partial contributions of each feature based on Shapley values (Lundberg et al., 2020, p. 64). These intermediate objects are stored and updated dynamically to avoid redundant computations.

Taking advantage of the sparsity and hierarchical structure of decision trees, TreeSHAP avoids evaluating every possible subset and therefore computes SHAP values in polynomial time. The overall complexity becomes

$$\mathcal{O}(M \cdot L \cdot \Upsilon^2),$$

where M is the number of trees, L is the maximum number of leaves and Υ is the maximum tree depth. In comparison, a direct implementation of the

Shapley formula has complexity $\mathcal{O}(M \cdot L \cdot D \cdot 2^D)$, where D is the number of features (Lundberg et al., 2020, p. 64).

The path-dependent TreeSHAP algorithm is summarized in Algorithm 3 (see Appendix A.3) and follows the structure of Algorithm 2 in Lundberg et al. (2020), with notation adapted for consistency with this thesis.

With the SHAP values derived for tree-based methods, these can be computed and interpreted.

2.3.2.3 Visualizing SHAP Explanations

Once SHAP values are computed, each observation in the dataset receives one SHAP value per feature. This results in a matrix of size $N \times D$, which can seem like too many to easily interpret. These values can be interpreted either locally (to explain an individual prediction) or globally (to understand model behavior across the dataset). However, the sheer volume of values can be difficult to interpret directly.

A commonly used visualization tool is the *beeswarm plot*, which displays the distribution of SHAP values across all instances and features. Figure 10 illustrates an example of a movie enjoyment prediction model. In a beeswarm plot, the x-axis shows the SHAP value and the y-axis lists all features, ordered by their mean absolute SHAP value (i.e., overall importance). The higher the value, the more impact the feature had on the predictions. Each point in the plot corresponds to a single observation for a given feature, and is placed horizontally according to its SHAP value. The color of the point represents the original feature value, where in Figure 10 yellow represents low values and purple indicates high values.

Points that lie to the right of the vertical zero line indicate that the feature increased the model's prediction for that instance; those to the left decreased it. The color provides further information: if high feature values consistently push predictions upward (right side), the model has learned a positive relationship between the feature and the target.

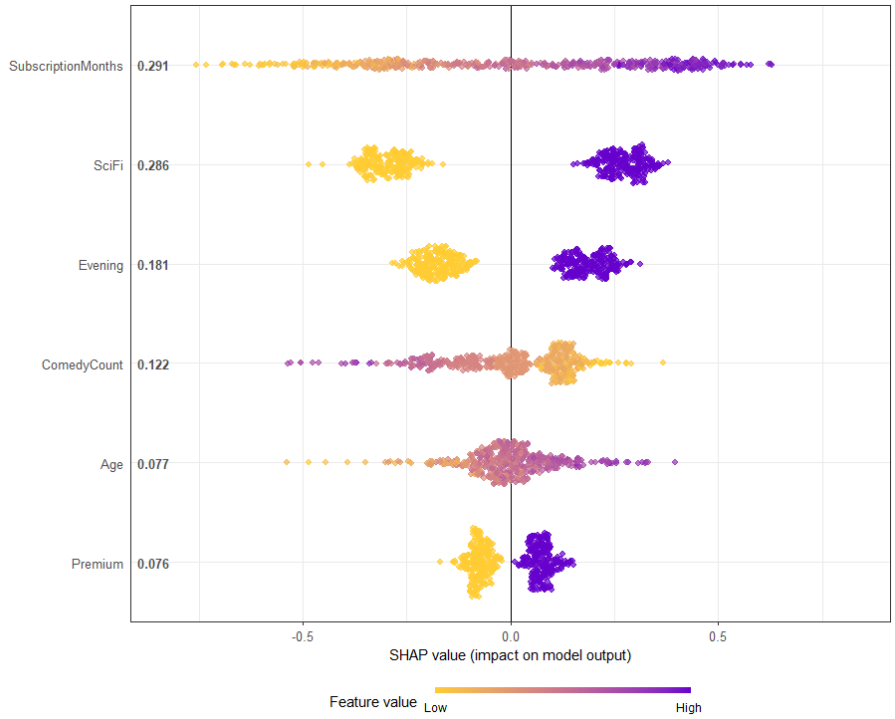


Figure 10: Beeswarm plot for a movie preference model. Features are ranked by impact (mean |SHAP|). Each dot corresponds to a SHAP value for one instance and one feature, colored by its original value (yellow = low, purple = high). For example, high `SubscriptionMonths` (purple dots on the right) strongly increase predicted enjoyment.

In Figure 10, we can interpret each feature's impact pattern in more detail. The feature `SubscriptionMonths` shows a relatively uniform distribution across the x-axis. High feature values (purple) are mostly located to the right, indicating that users with long subscription histories tend to receive higher predicted enjoyment scores. Conversely, lower values (yellow) tend to push the prediction downward. This suggests a roughly linear and monotonic relationship between the number of subscription months and predicted enjoyment.

Binary features like `SciFi`, `Evening` and `Premium`, form two tight clusters, corresponding to values 0 (yellow) and 1 (purple). For all these features, instances have separated on low and high feature values, which indicate a strong connection between how the feature affects the model prediction. For example, users with `SciFi` = 1 typically receive a high positive SHAP value, meaning that Sci-Fi preference strongly increases predicted enjoyment.

For `ComedyCount`, low values are associated with small positive contributions, while higher ones push the prediction strongly downward. This suggests that watching too many comedy movies tends to reduce predicted enjoyment,

perhaps due to content fatigue.

Lastly, `Age`, a continuous variable, forms a wide cloud of points centered near zero, with both low and high values having different effects on the prediction. This implies that `Age` contributes less consistently than the other features, and can be interpreted as minimal or a context-dependent impact (e.g., interactions with other features).

Taken together, the beeswarm plot gives both a global view of feature importance (via vertical ranking) and local information about how each feature affects predictions, including the direction and magnitude of its influence and whether the relationship is consistent or varies across observations.

2.3.3 Counterfactual Analysis

While SHAP explains why a prediction was made, it does not answer how it could have been different. Suppose a machine learning model is used to classify a patient based on their risk of developing a certain condition. For a given individual, the model predicts high risk, though the person expected a low-risk outcome. A natural question arises: what would need to change in the input features for the person to no longer be classified as high-risk? *Counterfactual explanations* try to answer this by identifying the *minimal* changes in the input that would result in a desired prediction - in this case, a low-risk classification (Molnar, 2019, p. 191).

This way of thinking aligns with human reasoning, as we often think through "what if" scenarios when trying to understand complicated situations and make decisions. Counterfactual explanations formalize this intuition by asking: What is the closest possible input to the current one, such that the model output changes in a meaningful way? These explanations are local, instance-based and contrastive (Molnar, 2019, p. 192). That is, they compare the original input to a hypothetical alternative, the "counterfactual", which leads to a different output. This is especially useful when understanding *how* to achieve a different result is more relevant than understanding *why* the current prediction was made.

Traditional explanation methods such as feature importance may give valuable information on a global level, but often fall short of providing actionable recommendations for individual cases. Counterfactuals aim to close this gap by producing alternative instances that are both close to the original and aligned with a specific target outcome. This target can be a different class label in classification problems or a target range in regression.

Mathematically, counterfactual analysis involves finding an alternative feature vector $\mathbf{x}^* \in \mathbb{R}^D$ that is minimally different from the original instance $\mathbf{x}$, yet results in a different predicted outcome (Wachter et al., 2018, p. 9). This search can be framed as a constrained optimization problem with multiple competing objectives: (i) ensuring the prediction changes, (ii) remaining close to the original input ($\mathbf{x}^*$ near $\mathbf{x}$), (iii) altering as few features as possible, and (iv) choosing feature values that are likely in the feature space (Molnar, 2019, p. 193).

2.3.3.1 Mathematical Framework

Let $\hat{f}\colon \mathbb{R}^D \to \mathbb{R}$ be a trained predictive model, and $\mathbf{x} \in \mathbb{R}^D$ a feature vector with output $\hat{f}(\mathbf{x}) \in \mathbb{R}$. A counterfactual instance $\mathbf{x}^* \in \mathbb{R}^D$ is a modified input that yields a prediction within a desired outcome set $Y^* \subset \mathbb{R}$, while remaining as similar as possible to $\mathbf{x}$. The target set Y^* may be a specific value or a continuous interval.

The counterfactual loss is defined as a four-dimensional objective (Dandl et al., 2020, p. 451):

$$\mathcal{L}_C(\mathbf{x}, \mathbf{x}^*, Y^*, \mathbf{X}^{\text{obs}}) = \big(c_1, c_2, c_3, c_4\big), \tag{67}$$

where each component $c_1, .., c_4$ correspond to the four criteria (i)-(iv) mentioned above, and $\mathbf{X}^{\text{obs}}$ is the training data.

The first term c_1 ensures that the prediction for $\mathbf{x}^*$ is sufficiently close to the desired target. This is typically defined as

$$c_1\big(\hat{f}(\mathbf{x}^*), Y^*\big) = \begin{cases} 0 & \text{if } \hat{f}(\mathbf{x}^*) \in Y^* \\ \inf\limits_{y^* \in Y^*} |\hat{f}(\mathbf{x}^*) - y^*| & \text{else} \end{cases}, \tag{68}$$

penalizing candidates that fail to reach the target region. This term can be relaxed to allow soft thresholds in regression.

To encourage similarity, the second term c_2 measures the distance between $\mathbf{x}$ and $\mathbf{x}^*$. Gower distance, δ_G, is commonly used here, as it handles mixed feature types well (Dandl et al., 2020, p. 451):

$$c_2(\mathbf{x}, \mathbf{x}^*) = \frac{1}{D} \sum_{j=1}^{D} \delta_G(x_j, x_j^*) \in [0, 1], \tag{69}$$

with D being the number of features. The distance calculation δ_G varies according to feature type (Gower, 1971):

$$\delta_G(x_j, x_j^*) = \begin{cases} \frac{1}{\widehat{R}_j} |x_j - x_j^*| & \text{for numerical features} \\ \mathbb{I}_{x_j \neq x_j^*} & \text{for categorical features,} \end{cases} \tag{70}$$

where $\widehat{R}_j$ is the range of values for feature j in the observed dataset (Dandl et al., 2020, p. 451).

Since two instances could be close in terms of raw distance but differ in many features, the third objective c_3 promotes sparsity by counting how many features change using the L_0 norm:

$$c_3(\mathbf{x}, \mathbf{x}^*) = \|\mathbf{x} - \mathbf{x}^*\|_0 = \sum_{j=1}^{D} \mathbb{I}_{x_j \neq x_j^*}. \tag{71}$$

This favors simpler and more interpretable changes.

Finally, c_4 ensures plausibility by penalizing unrealistic counterfactuals, meaning those lying in low-density regions of the training data. It computes the average Gower distance between $\mathbf{x}^*$ and its k nearest neighbors $\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[k]} \in \mathbf{X}^{\text{obs}}$ from the training data (Dandl et al., 2020, p. 451):

$$c_4(\mathbf{x}^*, \mathbf{X}^{\text{obs}}) = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{D} \sum_{j=1}^{D} \delta_G(x_j^*, x_j^{(i)}), \tag{72}$$

where $x^{(i)}$ are the nearest neighbors and $\mathbf{X}^{\text{obs}}$ is the training data. This objective acts as a soft constraint, keeping solutions in regions that are supported by the data.

Together, these objectives sum up the goal of counterfactual analysis: to identify alternative scenarios that are not only reaching a specific target, but also being realistic and interpretable.

Now, the challenge lies in optimizing all four objectives in (67) simultaneously. This can be done using the *Nondominated Sorting Genetic Algorithm* (NSGA-II), an *evolutionary algorithm* designed to find a set of Pareto-optimal solutions (Dandl et al., 2020, p. 452; Deb et al., 2002). The Pareto-optimal set refers to solutions for which no single objective can be improved without worsening at least one of the others (Deb et al., 2002, p. 182). Put differently, these can be interpreted as the local minima of the objectives.

In the context of counterfactual explanations, this corresponds to the alternatives that each balance prediction accuracy, similarity, sparsity and plausibility in different ways. The goal of the algorithm is to find multiple valid options, rather than producing a single counterfactual. A detailed description of the NSGA-II algorithm, including its steps and workflow, is provided in Appendix A.3.1.

While NSGA-II builds a great foundation, Dandl et al. (2020) have modified it and developed the *Multi-objective Counterfactual (MOC)*, that better suits specific demands of counterfactual explanations (Dandl et al., 2020). For instance, MOC gives the option to fix certain features (e.g., age, gender), which are then held constant throughout the optimization to reflect real-world constraints on what changed. Additionally, among other things, different crossover and mutation strategies are implemented for continuous, ordinal and categorical features, improving the search in mixed-feature spaces (Dandl et al., 2020, p. 452).

To conclude, we summarize this section with an example. Consider a heart disease risk prediction model based on these features:

- Age
- Blood pressure (BP)
- Cholesterol level
- Smoking status (yes/no)

We are interested in finding counterfactual explanations for an individual with feature values:

Age	**BP**	**Cholesterol**	**Smoking**
50	160	240	Yes

For this instance, the model predicts a high risk with 85% probability and we want to reduce this probability to 50%. An initial population might contain candidate counterfactuals such as those presented in Table 1

Table 1: Initial counterfactual candidates (generation 1)

Candidate	**Age**	**BP**	**Cholesterol**	**Smoking**
1	50	140	240	Yes
2	50	160	200	Yes
3	50	160	240	No
4	45	160	240	Yes
5	50	150	220	No

Each candidate is evaluated according to the four objectives. For example, candidate 3 who is not smoking, may result in a desired risk prediction of 50% (good c_1 value), changes a binary feature (moderate c_2), changes only one feature (good c_3), and lies well within the training distribution (good c_4).

NSGA-II might identify candidates 3 and 5 as Pareto-optimal (front 1), while the rest fall into subsequent fronts. Suppose candidates 3 and 5 are selected as parents, with a crossover resulting in a child with values:

Age	**BP**	**Cholesterol**	**Smoking**
50	150	240	No

Mutation might refine this to:

Age	**BP**	**Cholesterol**	**Smoking**
50	145	240	No

After several generations, the algorithm converges toward a diverse set of counterfactuals along the Pareto front, presented in Table 2.

Table 2: Converged Pareto-optimal counterfactuals

Candidate	**Age**	**BP**	**Cholesterol**	**Smoking**
A	50	140	240	Yes
B	50	150	220	No
C	45	150	230	No

Each solution in Table 2 represents a different balance between the objectives. Candidate A changes only BP but needs a relatively large reduction. We have candidate B that modifies two features with moderate changes, and candidate C who proposes more changes but may better match the distribution of realistic instances.

This diversity of solutions enables us to choose counterfactuals that align with real-life usefulness. For example, a doctor may prefer candidate B if moderate adjustments across multiple features are more practical than extreme changes to a single variable.

3 Results

3.1 Data

The initial dataset consisted of 428 participants (mean age = 45.1, SD = 14.0, range = $18 - 70$), recruited via the online platform Prolific.co. Participants were part of a convenience sample, meaning that there were no specific inclusion criteria regarding psychiatric diagnoses. All measures were collected via participants' own mobile smartphones between 11:00–17:00 to account for circadian rhythms. The study was approved by the Swedish Ethical Review Authority (dnr: 2020-03250 and 2021-01695).

After excluding extreme values (typically beyond 4 standard deviations) and applying task-specific outlier removal procedures described in Appendix B.3, the final analysis sample included 356 participants.

Self-Report Measures
Participants completed the Reduced Morningness-Eveningness Questionnaire (rMEQ) to assess their chronotype, which is a measure of individual preference for morning or evening activity. Theoretical scores range from 4 (strong evening preference) to 25 (strong morning preference), with a sample range: between 4–23.

Additionally, participants completed the Adult ADHD Self-Report Scale (ASRS v1.1), an 18-item questionnaire divided into Part A and B. Two outcomes were derived: (1) a binary ASRS screener flag (TRUE/FALSE) based on Part A responses, and (2) a total symptom score (theoretical range: $0 - 72$; sample range: $0 - 64$) summing responses across all 18 items. See Appendix B.1 for scoring details.

Cognitive Task Data
All participants completed the following seven cognitive tasks assessing domains such as attention, cognitive control, memory and planning: Design fluency, Go, Go/No-Go, Stroop, Stroop switch, Tower of London and Grid. Descriptions of each task are provided in Appendix B.2. Tasks were completed under time constraints, resulting in a varying number of trials per participants for each task.

Raw trial-level data were aggregated into summary features based on guidelines from the data providers. Table 3 presents these cognitive features. For the Go and Go/No-Go tasks, the participants' *response time variability* (RTV) was computed instead of raw response time. This was done because all participants completed the tasks on their own smartphones, with different processing speeds. To obtain a measure that is not too affected by those differences, the RTV is used instead and is calculated as:

$$\text{RTV} = \frac{\text{MAD(response time)}}{\text{median(response time)}} \times 100\%,$$

where MAD denotes the median absolute deviation.
The final dataset included the cognitive performance features presented in Table 3, rMEQ (chronotype) scores, age and ADHD symptom measures. These formed the basis for the following exploratory analysis, clustering, and predictive modeling.

Table 3: Summary features derived from cognitive task performance

Feature name	**Description**	**Cognitive task**
`correct_trials_c3`	Number of correct trials under condition 3	Design Fluency
`gono_rtv`	RTV in go-trials	Go/No-Go
`mean_switch_cost`	Average RT difference for switch vs. non-switch trials	Stroop Switch
`go_rtv`	RTV in go-trials	Go
`congruencyeffectRT`	RT difference between incongruent and congruent trials	Stroop
`reverse_max`	Maximum span achieved in backward condition	Grid
`total_moves`	Total number of moves across all trials	Tower of London

Note. RT = Response time. RTV = Response time variability.

3.2 Statistical Analysis

3.2.1 Exploratory Analysis and Clustering

Prior to modeling, exploratory analyses were conducted to examine the feature distributions and space. Histograms were generated for each feature, in which uniform noise $U(-0.5, 0.5)$ was added to discrete-valued features (`correct_trials_c3`, `reverse_max`, `chronotype_score`) to create smoother distributions for visualization, see Figure 11. Most features showed roughly symmetric distributions, with mild right skewness observed in some variables, particularly those related to response time measures (e.g., `go_rtv` and `gono_rtv`). This skewness is expected in cognitive performance data.

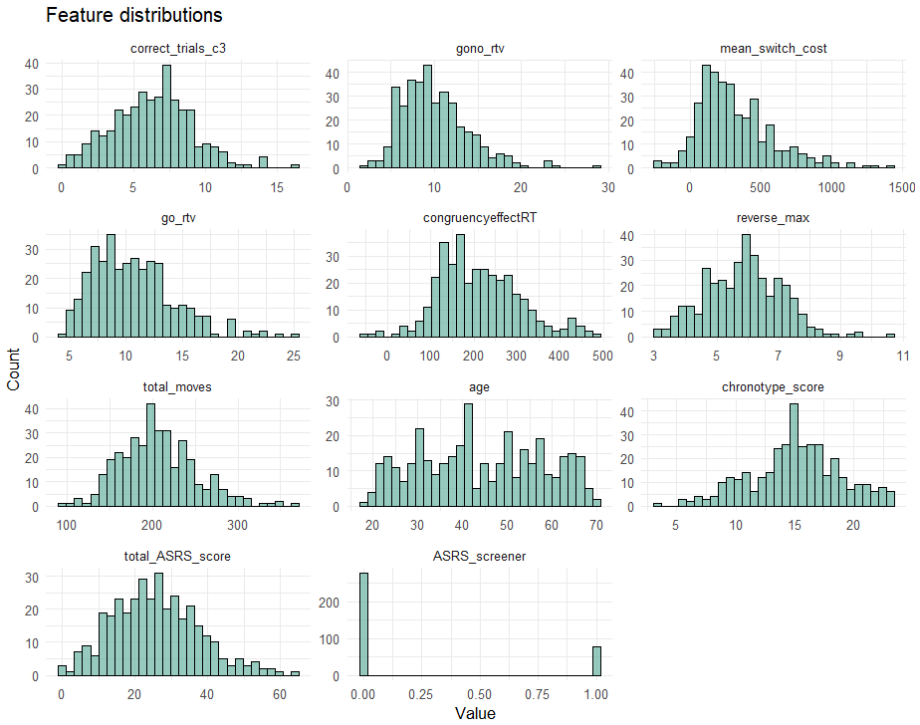


Figure 11: Histograms of all cognitive and demographic features used in modeling. Uniform noise was added to discrete variables for visualization. Distributions are approximately symmetric, with some skewness in reaction time-related features.

The total ASRS symptom score was relatively symmetrically distributed across the sample, while the binary ADHD screener flag showed a clear class imbalance, with more participants screening negative (0) than positive (1). Age and chronotype were broadly spread across their respective ranges, with some local peaks but no extreme clustering or gaps.

Overall, no anomalies were observed that would interfere with modeling, although the imbalance in the screener flag may pose challenges for classification models.

Empirical cumulative distribution functions (ECDFs) were generated for each feature to further assess the smoothness and continuity of the distributions (Figure 26 in Appendix C). The ECDFs confirmed the overall shapes observed in the histograms, with smooth curves and no visible discontinuities or multimodal behavior, indicating no clear evidence of latent clustering in individual features.

A correlation heatmap for all features is presented in Figure 12. Although XGBoost handles correlated features well, this analysis was done to verify that no extreme multicollinearity or redundancy was present in the predictors.

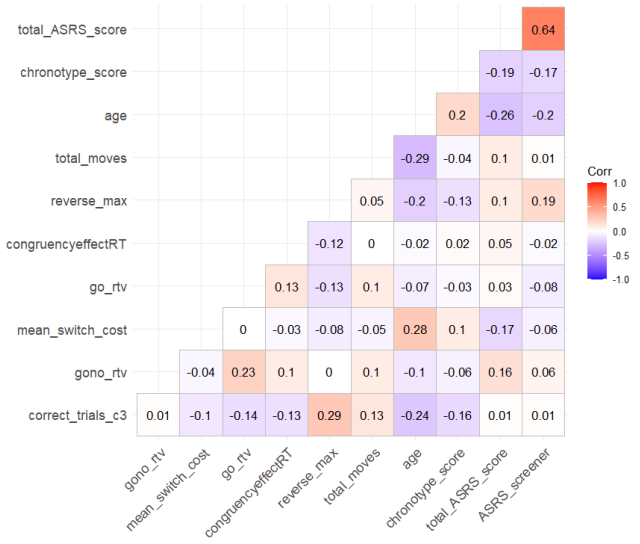
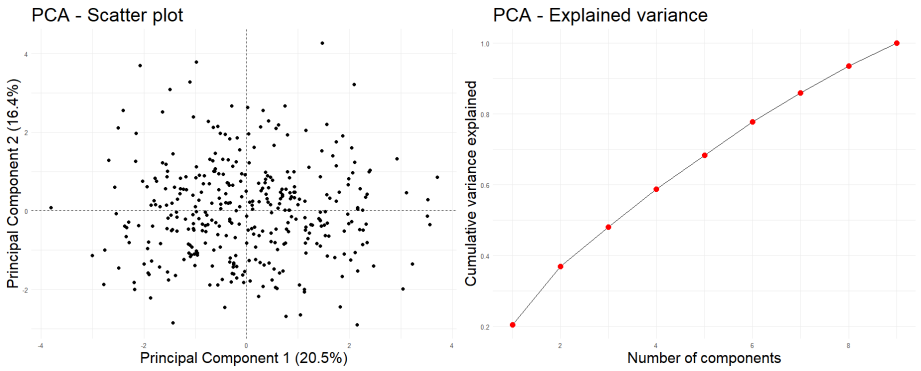


Figure 12: Heatmap of pairwise Pearson correlations between the features. Moderate correlations were observed, with no strong collinearity.

Feature-to-feature correlations were generally low to moderate ($|r| < 0.3$). Chronotype score and age were modestly negatively correlated with total ASRS scores ($r = -0.19$ and $r = -0.26$, respectively). As expected, the total ASRS symptom score showed a strong positive correlation with the binary ASRS screener flag ($r = 0.64$). Based on these values, no features had to be removed due to collinearity.

To get an initial overview of the data structure, PCA was performed on all nine standardized features. The first two principal components are plotted in Figure 13a. No clear clustering structure was observed, with data points forming a relatively uniform cloud around the origin. Other combinations of principal components were examined and revealed similar results, with no distinct groups or separation. Figure 13b shows the cumulative explained variance as a function of the number of principal components. The increase was approximately linear, indicating that no small subset of components captured the majority of variance. Ideally, one might expect to see a steep initial rise followed by a plateau, but this was not observed.



(a) Participant projections (PC1 vs. PC2) (b) Cumulative variance vs. components

Figure 13: PCA results. (a) Participant projection shows no clear separation. (b) The line looks linear and gradual, requiring at least 8 components to explain 95% of the variance.

To explore potential latent structures, clustering was performed using Gaussian mixture models (see Section 2.1). The choice of method was primarily driven by the nature of psychological data, where boundaries between clusters tend to be gradual rather than discrete. Since the initial PCA embeddings indicated considerable overlap in the feature space, hard clustering methods were deemed unsuitable.

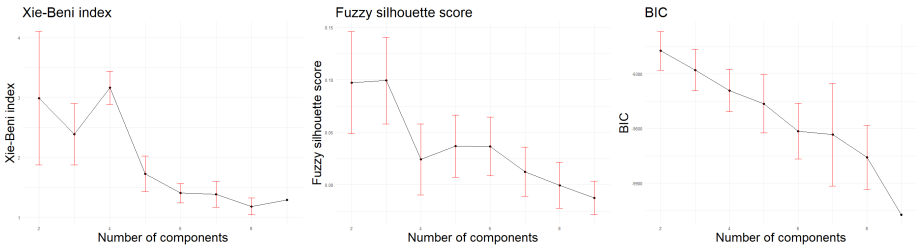
GMMs were fitted using the `VVV` covariance model from the `mclust` -package in R. This model allowed each component to vary in volume, shape and orientation, providing maximal flexibility without imposing strong constraints on cluster geometry. Such flexibility is justified given the unknown structure of the latent feature space. The GMMs were fitted for $K = 2, \ldots, 9$ components and all features were standardized to zero mean and unit variance prior to clustering. For each K, the Xie-Beni index and fuzzy silhouette scores were computed to evaluate cluster quality, as well as BIC for comparison (see Section 2.1.2).

The results from the GMM clustering are presented in Figure 14, showing the validation indices plotted against the number of Gaussian components K. The Xie-Beni index (a), displays a local minimum at $K = 3$, followed by an increase at $K = 4$ before it decreases for larger K. Although lower Xie-Beni values suggest more compact clustering, the behavior for $K > 4$ was likely due to overfitting and not reflective of meaningful structure.

The fuzzy silhouette score in (b) reached a maximum at $K = 3$ but remained low across all K values, ranging between -0.01 and 0.1. As with the standard silhouette score, values close to 1 indicate good separation of clusters, whereas values close to 0 or negative suggest poor clustering. Our results imply great overlap among clusters, consistent with what we saw in the PCA visualization.

The BIC results are shown in Figure 14c, where a two-component model ($K = 2$) was favored.

However, the results presented in Figures 14a-14c should be interpreted cautiously, as the big overlap of the error bars across different K indicates instability.



(a) Xie-Beni index (b) Fuzzy silhouette score (c) BIC

Figure 14: GMM validation results. All metrics indicate weak or unstable clustering structure, especially beyond $K = 3$. Red lines represent error bars generated via bootstrapping.

Despite the lack of strong evidence for the clustering, we proceeded with $K = 3$ to inspect potential structure. The densities of the three components were drawn and overlaid on the feature histograms to compare marginal cluster differences, see Figure 15. In the figure, it is very clear that the clusters are overlapping, supporting the conclusion that no meaningful clusters have been found.

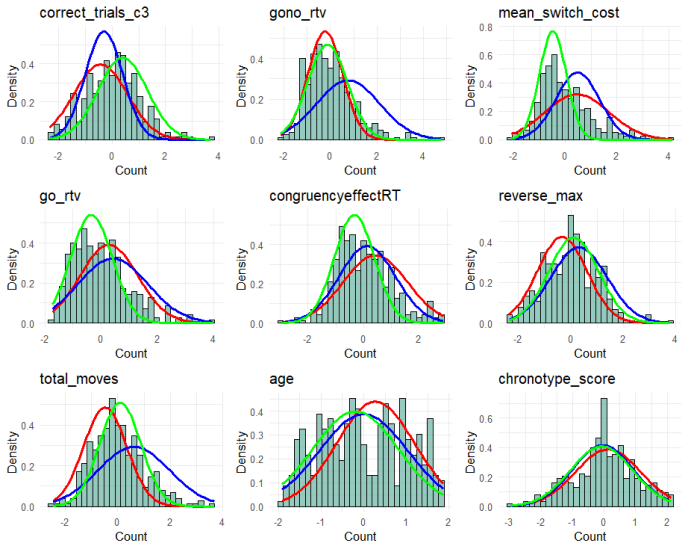


Figure 15: Standardized feature histograms overlaid with densities from the $K = 3$ GMM solution. Colors correspond to different Gaussian components. Strong overlap between components indicate poor separation between clusters.

3.2.2 Predictive Modeling

XGBoost (eXtreme Gradient Boosting) was used to predict total ASRS score in a regression task (see Section 2.2). The method was chosen due to its built-in regularization and its ability to produce accurate predictions on nonlinear data. While alternative methods such as Support Vector Regression or Random Forests were considered, XGBoost offered better integration with explainable ML methods used later in the analysis.

Squared error loss was chosen as the regression objective since outlier removal had already been done during preprocessing, making more robust alternatives unnecessary (see Section 2.2.3). For completeness, we also experimented with the Pseudo-Huber loss; however, it did not improve performance and only added unnecessary computational complexity.

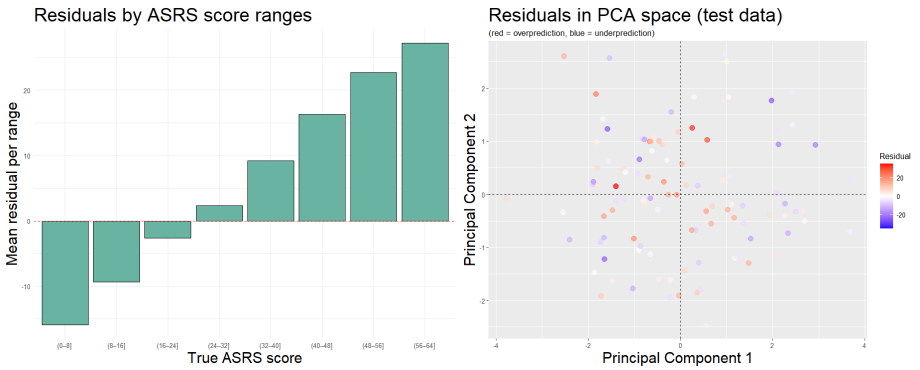
Given the minimal influence of remaining outliers and the dataset size, hyperparameter tuning was limited to three parameters: `nrounds` (number of boosting iterations), `max_depth` (maximum tree depth) and `lambda` (L2 regularization strength). We fixed `eta` $= 0.3$ and `gamma` $= 0$ (both default values) to reduce the search space and risk of overfitting during cross-validation.

Hyperparameters were manually tuned based on learning curve analysis and cross-validation performance. Specifically, 5-fold cross-validation was performed using the full data across `nrounds` $\in \{1, \ldots, 100\}$, for each `max_depth` $\in \{2, 3, 4\}$, with `lambda` initially fixed at its default value 1. Learning curves were inspected to choose a value of `nrounds` that minimized the test error, and a tree depth that achieved comparable performance with minimal model complexity (Figure

28 in Appendix C). A maximum depth of 2 was chosen for simplicity, as higher depths showed similar behavior without significant performance gains.

Finally, models with `lambda` $= 0$ and `lambda` $= 1$ (default) were compared to assess whether L2 regularization was necessary. Given the relatively small dataset, it was possible that the underlying relationship was simple enough that regularization would not improve the results. Cross-validation results showed a negligible difference in test error between the models ($\Delta\text{RMSE} \approx 0.05$), with `lambda` $= 1$ performing slightly better. Based on cross-validation, the chosen hyperparameters were `nrounds` $= 10$, `max_depth` $= 2$ and `lambda` $= 1$. A final model was then trained on a 70% training split and evaluated on the 30% test set.

The final XGBoost regression model achieved a test RMSE of 11.70, representing approximately 16.3% of the theoretical range $(0 - 72)$ and 18.3% of the observed data range $(0 - 64)$ of the ASRS score target variable. To assess whether model errors followed any systematic patterns, residuals were analyzed by binning true ASRS scores into 8-point intervals ($[0, 8], (8, 16], \ldots, (56, 64]$), and computing the mean residual per bin (Figure 16a). The figure reveals a clear trend: the model tended to overpredict ASRS scores for participants with low true values and increasingly underpredicted for higher true values. Predictions in the middle range $(16 - 32)$ showed minimal bias.

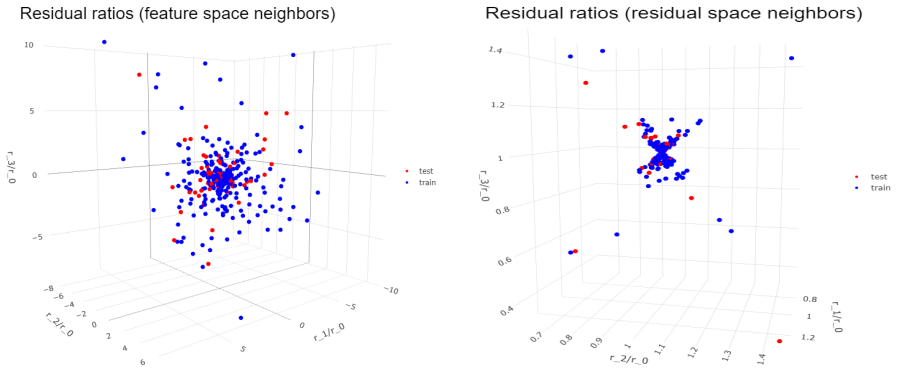


(a) Mean residual value by true ASRS score bins.

(b) PCA projection of testing data.

Figure 16: Residual analysis of the XGBoost regression model. (a) The model tends to overpredict low scores and underpredict high scores, indicating a bias toward the middle. (b) The projections are colored by residual value. No clear regional clustering of errors is observed in reduced feature space.

Additionally, the standardized feature space was projected onto the first two principal components, with each point colored by its residual value (Figure 16b). Test-set residuals in this reduced dimensionality space showed no strong regional clustering, suggesting no systematic bias toward feature structure. Given the



(a) Ratios in feature space.

(b) Ratios in residual space.

Figure 17: Ratios of nearest neighbor residuals in (a) feature space and (b) residual space. In feature space, ratios are scattered around zero, indicating no strong similarity in prediction errors among participants with similar features. In (b), ratios cluster tightly around (1, 1, 1), reflecting coherence among nearby residuals by construction.

small test set size ($N = 107$), residuals for the full dataset (training + testing) were also examined (Figure 27 in Appendix C). The full data projection similarly showed a diffuse residual distribution pattern without clear trends, supporting the absence of feature-driven residual error.

To further investigate the structure of prediction errors, each residual was compared to its three nearest neighbors. For each reference residual r_0, the ratios r_1/r_0, r_2/r_0, and r_3/r_0 were computed, where r_i denotes the residual of the ith nearest neighbor. Two approaches were used to define "nearest": (i) identifying neighbors based on proximity in feature space and (ii) by proximity in residual value. The first assesses whether participants with similar feature profiles have similar prediction errors, while the second examines whether residuals of similar magnitude are structurally related.

When neighbors were selected in feature space (Figure 17a), the ratio coordinates were centered around (0,0,0), indicating that participants with similar cognitive features did not necessarily exhibit similar prediction errors. Train and test sets were intermixed.

On the other hand, when neighbors were defined based on residual closeness (Figure 17b), points clustered tightly around (1,1,1), as expected given the construction. The shape resembled a symmetric elliptic cone, possibly reflecting small random deviations from the theoretical ideal ratio of 1. A small number of outliers with near-zero denominators were excluded from both plots for clarity.

These results support the view that prediction errors are not systematically structured relative to participant features, but that residual values themselves showed local consistency.

To interpret the output of the XGBoost model, SHAP (SHapley Additive exPlanations) values were computed using TreeSHAP (see Section 2.3.2.2). SHAP was chosen over alternatives such as LIME due to its stronger theoretical foundation and built-in support for tree-based models. Because the dataset was relatively small, computational cost was not a barrier, allowing for exact SHAP value computation. The analysis focused on participants with true ASRS scores between 15 and 33, where model performance was most stable. This range was selected based on a detailed inspection of the residuals beyond what is shown in Figure 16a: while the binned bar plot suggests relative stability between $16-32$, closer examination showed that prediction errors remained consistently small and unbiased between approximately 15 and 33, and began increasing sharply outside this interval.

Figure 18 shows the SHAP beeswarm plot for this subset of participants. The most influential feature was age, where lower age values tended to increase the predicted ASRS scores, while higher ages decreased them. Chronotype scores showed a similar directional trend: morningness (higher values, purple) reduced predictions, while moderate to low chronotype values increased them.

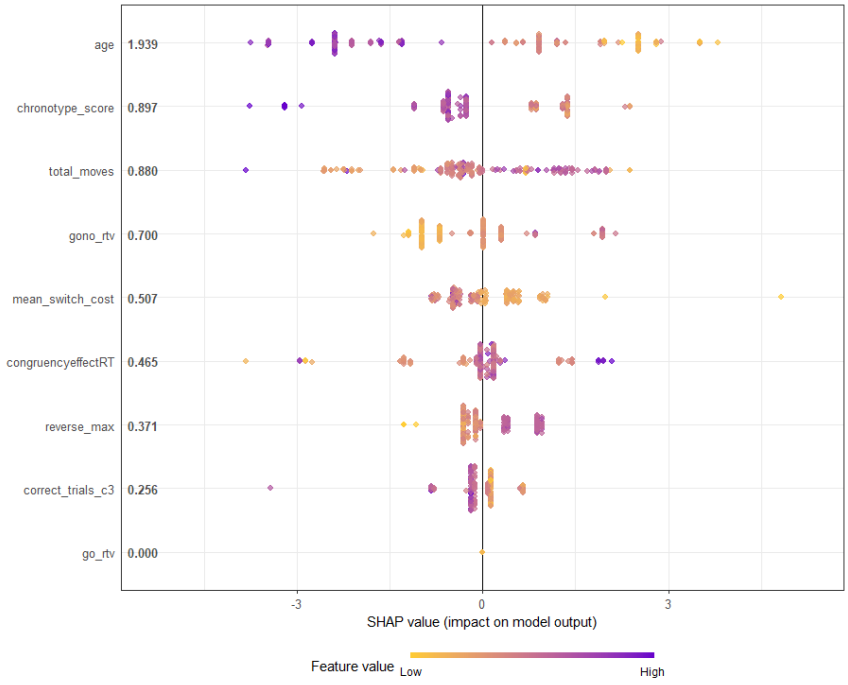


Figure 18: SHAP beeswarm plot for participants with ASRS scores between 15-33. Each point represents a participant's SHAP value for a feature, with color indicating the original feature value (purple = high, yellow = low). The SHAP values indicate the direction and magnitude of each feature's contribution to the model output. The features are sorted by overall importance.

Total moves in the Tower of London task showed a mixed distribution, but higher values generally corresponded to increased prediction scores, aside from a few outliers. For Go/No-go response time variability, moderate values pushed predictions higher, while low values lowered them. Mean switch cost showed the opposite: low values raised the predictions, and moderate values decreased them.

Some features (e.g., `congruencyeffectRT`, `reverse_max`, and `correct_trials_c3`) showed minimal global influence but contributed meaningfully in subsets of participants, as is clear by the apparent clusters.

For `correct_trials_c3`, SHAP values formed two mirrored groupings on either side of the vertical zero line. On the right, two distinct clusters of yellow-orange points appear, with one larger cluster near zero and a smaller cluster further out. On the left, two similarly structured clusters of purple points are seen. This suggests that the model predicts differently for subgroups of participants with similar correct trial scores, assigning slightly lower or higher ASRS predictions depending on interactions with other features or other unknown causes.

Notably, `go_rtv` had zero influence in this SHAP analysis, with all points aligned at zero, indicating that the feature had no effect on model predictions within this subset.

To explore whether participants had similar explanation patterns, we treated each participant's SHAP values as a feature vector in a new explanation space. These vectors were then projected onto a two-dimensional embedding using Multidimensional scaling (MDS), allowing us to visualize if participants are clustered based on how their features influenced the model's predictions. Both raw and normalized SHAP values were analyzed. The normalized values, or *relative SHAP* contributions, were computed as

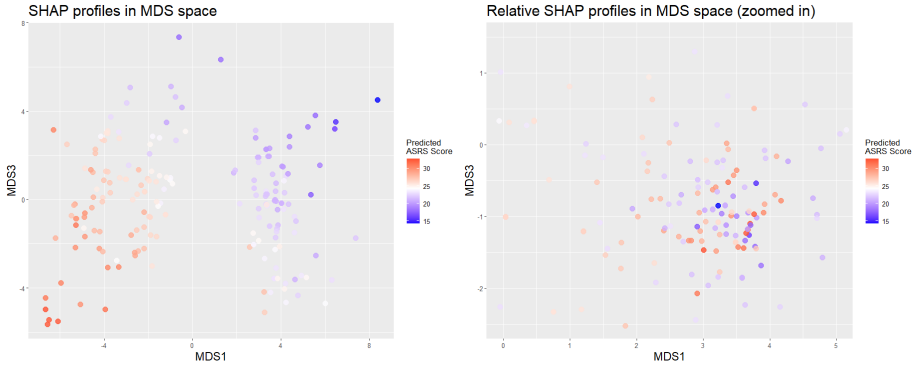
$$\varphi_{\text{rel},i} = \frac{\varphi_i}{\hat{f}(\mathbf{x}) - \varphi_0}, \tag{73}$$

where φ_0 is the baseline prediction. This reflects the proportion of each feature's contribution to the deviation from the baseline prediction φ_0.

The intuition behind this normalization is that it enables comparison between participants based on patterns of influence, rather than magnitude. For example, two participants whose predictions are composed of 40% from one feature and 60% from another will appear close in MDS space, even if their total SHAP values differ in scale.

Pairwise distances between SHAP profiles were computed using the Manhattan (L1) distance, which better preserves additive and sparse structures of SHAP vectors compared to Euclidean distance. The resulting MDS projections are presented in Figure 19.

In the raw SHAP projection (Figure 19a), participants formed two vertically oriented and elongated clusters. Within each, predicted ASRS scores followed a color gradient: higher predictions at the bottom of the clusters and lower scores at the top. This pattern suggests that participants with similar SHAP profiles



(a) MDS projection of raw SHAP profiles. (b) MDS projection of relative SHAP profiles.

Figure 19: Multidimensional scaling (MDS) projection of SHAP profiles, colored by predicted ASRS scores. (a) Raw SHAP values reveal two elongated clusters, each with a vertical gradient in predicted score, indicating that SHAP profile similarity aligns with model prediction similarity. (b) Relative SHAP profiles show no strong color gradient or clustering pattern, suggesting consistent influence patterns across participants. Several extreme outliers were excluded from the plot for visualization purposes and were likely due to near-zero denominators in the normalization formula.

also received similar model predictions. It also suggests that the model relies on at least two distinct sets of explanatory patterns to produce its predictions.

In contrast, the normalized SHAP projection (Figure 19b) showed no clear structure or prediction gradient. The absence of clusters implies that, although participants vary in the strength of feature contributions, the relative pattern of influence is overall consistent across individuals. Some outliers were removed from the plot and likely arose from near-zero denominators in Equation (73). The outliers were all associated with predicted ASRS scores close to the sample average (≈ 25), where this pattern may be a mathematical artifact of the normalization process, rather than a meaningful result.

Having completed the interpretability analysis of the regression model, we next evaluated the performance of a classification approach using the binary ASRS screener flag as the target variable.

3.2.2.1 Classification Task

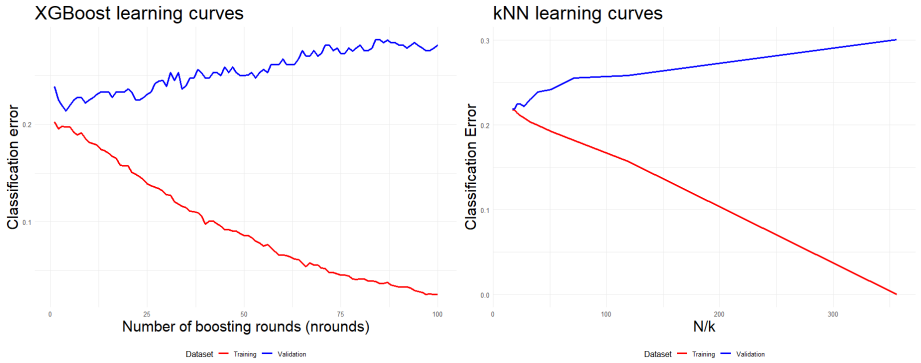
After the regression model was analyzed, a classification approach was implemented using the binary ASRS screener flag as the target variable. A second XGBoost model was trained using the same cross-validation strategy and hyperparameter grid as in the regression task. The objective function was log loss (see Section 2.2.4), and validation RMSE was used to select the final model. The optimal parameters were `nrounds` $= 6$, `max_depth` $= 2$, and `lambda` $= 1$.

Although learning curves (not shown) indicated stable convergence behavior, the final model had poor predictive performance. As shown in Table 4, the classifier correctly labeled 82 of 83 negative ASRS cases but failed to identify most positive ones, classifying only one participant as positive. This yielded a balanced accuracy of 0.51, with sensitivity 0.99 and specificity 0.04. This outcome likely reflects both the strong class imbalance and the limited discriminative power of the features.

Table 4: Confusion matrix for the XGBoost classifier.

		Actual	
		0	**1**
Prediction	**0**	82	23
	1	1	1

Further, to investigate the learning behavior of the XGBoost classifier, 5-fold cross-validation was repeated with classification error as the evaluation metric [5]. The resulting learning curves are shown in Figure 20a. Training error decreased steadily with the number of boosting rounds, while validation error remained relatively high and kept increasing again after a minimum at `nrounds` = 4, indicating overfitting and weak generalization.



(a) XGBoost: Classification error vs. number of boosting rounds. Max tree depth was set to 2. (`nrounds`).

(b) kNN: Classification error vs. model complexity (N/k).

Figure 20: Classification error curves for XGBoost (a) and kNN (b). Both models show relatively high validation error across model configurations, suggesting poor class separation and potentially limited predictive power of the features.

We also experimented by seeing if feature reduction could improve classification. All $\binom{9}{5} \times 2 = 252$ combinations of five features, at tree depths of 2

[5] Classification error $= \frac{\#\text{ of misclassifications}}{\#\text{ classified total}}$.

and 3, were evaluated. However, none of these subsets outperformed the full 9-feature model (confusion matrix and RMSE unchanged), and thus all features were used in the final evaluation.

To determine whether the poor performance was due to modeling problems or data-driven, a simple k-Nearest Neighbors (kNN) classifier was trained as a baseline. The number of neighbors k was optimized using 5-fold cross-validation, with classification error plotted against k. The lowest validation error occurred at $k = 19$. Figure 20b presents the learning curve, plotted against the kNN degrees of freedom N/k, for $k = 1, 3, 5, \ldots, 21$.

As with XGBoost, training error decreased with flexibility (higher k), but validation error remained high across all k values. The confusion matrix for $k = 19$ (Table 5) further confirms that the model failed to classify any positive ASRS cases correctly.

Table 5: Confusion matrix for the kNN classifier ($k = 19$).

		Actual	
		0	**1**
Prediction	**0**	83	24
	1	0	0

In an effort to understand why the classifiers performed poorly, feature distributions were plotted separately by ASRS screener class. As shown in Figure 21, serious overlap between positive and negative cases was observed across all features. No single feature, appeared able to reliably separate the classes.

Taken together, these results suggest that the classification task was particularly challenging due to class imbalance and overlapping feature distributions. As a result, SHAP or counterfactual analyses were not pursued for the classification models.

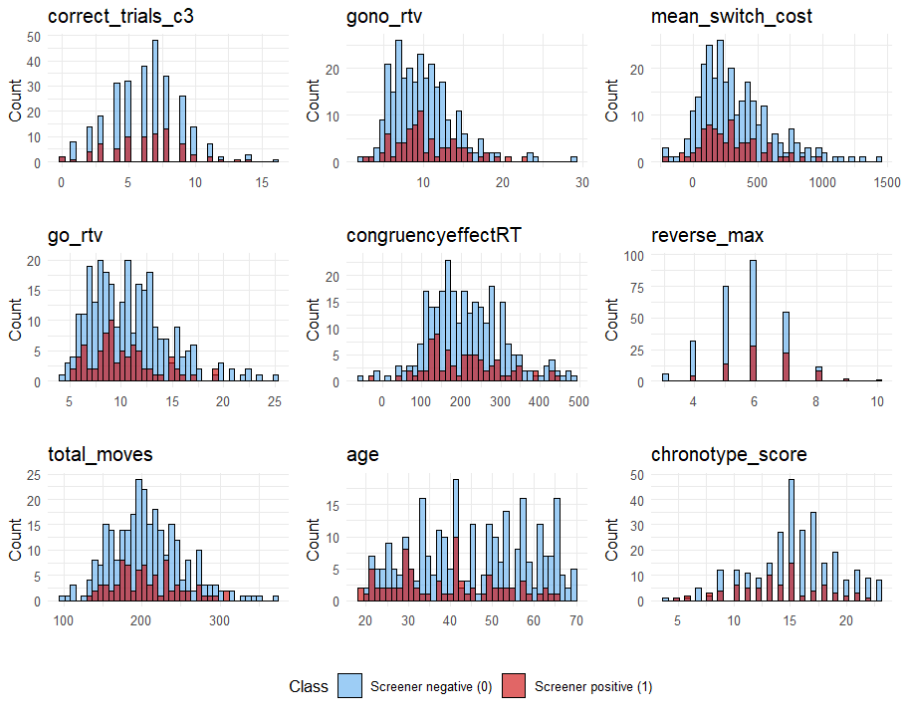


Figure 21: Feature distributions by ASRS screener class (0 = negative, 1 = positive). Severe distributional overlap is visible for all features.

3.2.3 Counterfactual Analysis

Counterfactual explanations were generated to understand how minimal feature changes could reduce predicted ASRS score (see Section 2.3.3). While SHAP values attribute importance to individual features for a given prediction, counterfactuals complement this by identifying the smallest set of changes needed to reach a different outcome.

This analysis was applied only to the regression model, since classification performance was poor and not reliable enough for interpretation. The focus was further restricted to participants with predicted ASRS scores between 15 and 33, based on the same residual pattern that guided the SHAP analysis.

Because the total ASRS score (derived from both Part A and B of the questionnaire) does not have an established clinical cutoff for diagnosis, a heuristic threshold of 30 points was used to distinguish between moderate and low predicted symptom severity. This value was chosen only to facilitate the application of counterfactual analysis, and should not be interpreted as a clinically meaningful boundary.[6] Participants with predicted scores above 30 were selected as

[6]In clinical settings, the binary screener is typically based on Part A of the ASRS questionnaire, which is used to determine whether further ADHD evaluation is recom-

targets, and counterfactuals were generated to shift their prediction to below this threshold.

Seven participants met these criteria. For each, we attempted to find four counterfactuals under the constraint that `age` could not decrease, but was allowed to increase to the maximum observed value (70 years). All other features were allowed to vary within the range observed in the training data, enabling the optimization to be very flexible in finding counterfactual instances.

The model successfully found 28 counterfactuals (four per participant). Figure 22 displays how often each feature was altered across all generated counterfactual instances. Features related to planning and response time (`total_moves`, `mean_switch_cost`, `gono_rtv`, `go_rtv` and `congruencyeffectRT`) were changed in every counterfactual, suggesting they had the greatest impact on reducing predicted ASRS scores. In contrast, `correct_trials_c3`, `chronotype_score` and `reverse_max` were altered less frequently. Notably, despite being the only constrained variable, `age` was modified in 27 cases.

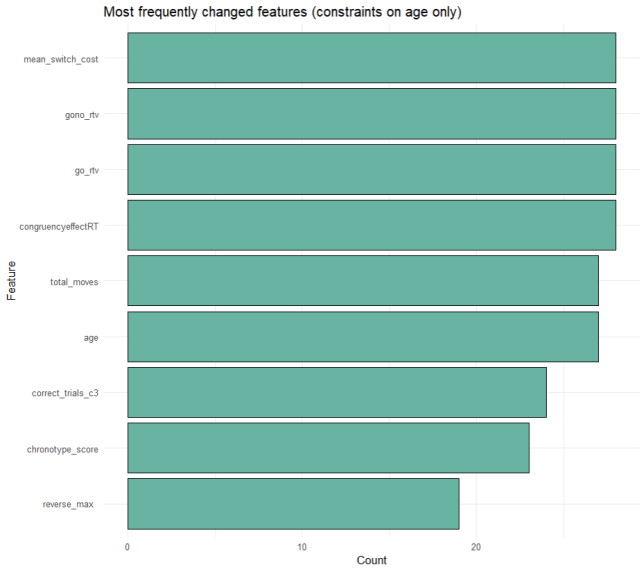


Figure 22: Frequency of feature changes across 28 counterfactual instances with predicted ASRS > 30. Here, `age` could only increase (up to 70 years). All other features were bounded by their training set minimum and maximum values. Features related to response time and planning were most frequently altered.

Summary metrics for these counterfactuals are presented in Table 6. On average, 8.14 out of 9 features were changed, with a mean scaled distance of 0.13 from the original instance. Minimality scores, which measure the number

mended. Part B is intended for additional symptom insight, and the combined total score is not used with formal thresholds. See, e.g., https://novopsych.com/assessments/diagnosis/adult-adhd-self-report-scale-asrs/ for an example of unofficial cutoff-ranges.

of features that could be reverted without affecting the desired prediction, were also high. When both the number of altered features and minimality score are high, it means that the model changed a lot, but most were not essential.

Table 6: Summary metrics across 28 counterfactuals (minimally constrained).

Metric	Mean
Number of features changed	8.14
Distance from original (dist_{x^*})	0.13
Minimality	7.42

In a secondary analysis, cognitive task features were bounded more realistically to try to reflect what could possibly be changed through clinical interventions. Specifically, each feature was allowed to vary by ± 1.5 standard deviations from the original instance's value, where standard deviations were computed from the training dataset. The earlier restriction for age remained, where it could not decrease but was allowed to increase by up to 70 years. These bounds were based on recommendations from the data provider and by SHAP value inspection. However, under these constraints, only a single counterfactual was successfully generated across all seven participants. This implies that most participants could not be shifted below the threshold through small, "realistic" changes.

To illustrate the solutions, Table 7 and Figure 23 show four successful counterfactuals generated for a participant with a predicted ASRS score of 32.38. The participant's original features are shown in the first row, followed by the four counterfactuals that reduced the score below 30.

Table 7: Feature-level comparison between the original test instance $\mathbf{x}$ (top row, bold; predicted ASRS = 32.38) and the four nearest counterfactuals (CF 1–CF 4) that reduce the regression model's predicted ASRS score below the heuristic threshold of 30.

Feature	correct_trials_c3	gono_rtv	mean_switch_cost	go_rtv	congruency effectRT	reverse_max	total_moves	age	chronotype_score
Original (x)	**10**	**12.91**	**11.87**	**7.17**	**245.36**	**4**	**245**	**22**	**10**
CF 1	10	13.74	309.54	15.17	204.61	6	234	23	11
CF 2	10	5.32	363.70	8.40	199.08	5	213	27	12
CF 3	8	13.70	333.78	6.10	233.57	6	177	30	12
CF 4	7	9.66	352.60	12.33	165.14	4	225	31	12

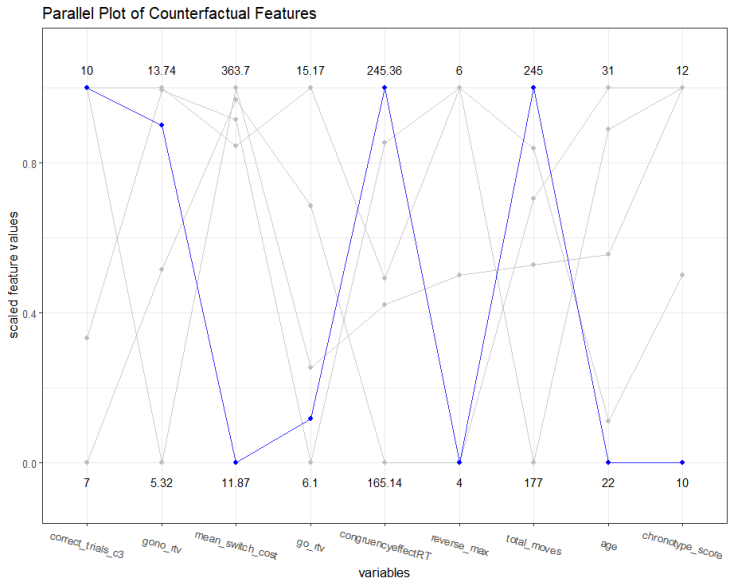


Figure 23: Parallel coordinate plot comparing an original instance (blue) with predicted ASRS score of 32.38 to four counterfactuals (gray) with scores below 30. Most features required large changes to lower the prediction.

As seen in Table 7, both `correct_trials_c3` and `reverse_max` were changed the least times, which aligned with the results in Figure 22. Distances to the original instance ranged from 0.12 to 0.14 in scaled feature space, with each counterfactual altering between 8 and 9 features. Minimality scores ranged from 7 to 8, meaning that in most cases, only one or two of the altered features were actually necessary to achieve the prediction shift; the remaining changes could be reverted without affecting the outcome.

Summary of results

The exploratory and predictive analyses found limited success in predicting total ADHD symptom level from cognitive task performance, age and self-assessed chronotype. Clustering attempts yielded no reliable structure, but regression modeling achieved reasonable accuracy in the mid-range of ASRS scores. SHAP analysis identified age, chronotype and planning-related features as most influential, with some features showing impact in specific participant subgroups. Counterfactual analysis demonstrated that reducing predicted symptom levels generally required changes to most features, and few plausible counterfactuals were found when realistic bounds were imposed.

By contrast, classification models performed poorly, primarily due to class imbalance and overlapping feature distributions, limiting their potential for screening applications.

4 Discussion

The main aim of this thesis was to investigate whether cognitive task performance, along with age and chronotype, could be used to predict adult ADHD symptom level as measured by the ASRS questionnaire. Beyond predicting, we also wanted to interpret the model's decisions using SHAP values and counterfactual explanations.

The clustering analysis using Gaussian mixture models revealed no clear structure in the data. The absence of distinct clusters aligns with the overlapping feature distributions (Figure 21), PCA projections and weak validation metrics. The not-so-contrastive Xie-Beni indices and near-zero fuzzy silhouette scores show the poor separation between components, suggesting that the feature space lacks clear subgroups. Given the "`VVV`" model allowed full covariance flexibility, the number of free parameters reached 164 for three components and nine features. With only 356 observations, this yields a low ratio of $\frac{356}{164} \approx 2.17$ observations per parameter.

In practice, there is no strict theory linking N to number of parameters in unsupervised models, but many rules-of-thumb exist. A common guideline is on the order of 5-10 observations per free parameter. For example, one review states that the sample size should preferably be 10 per parameter (Psutka & Psutka, 2019, p. 31). In latent-variable mixture modeling more broadly, Bentler and Chou (1987, p. 90-91) recommend a 5-10 ratio. Later overviews (e.g. Muthén and and (2002, p. 599-600)) advise even larger ratios, noting that "*the sample size needed for a study depends on many factors, including the size of the model, distribution of the variables, amount of missing data, reliability of the variables, and strength of the relations among the variables.*"

On that basis, our ratio of 2.17 lies well below even the relaxed end of the recommended range, making overfitting and ill-conditioned $\hat{\mathbf{\Sigma}}_k$ almost inevitable unless the covariance structure is regularized. The failure to identify clusters may therefore be a combination of limited sample size, overlap in feature space and heterogeneity in the ADHD symptom variable, where boundaries between severity levels are gradual rather than discrete.

Likewise, visualizing the regression residuals in PCA space resulted in a random scatter with no obvious pattern or cluster structure. Together, these observations suggest that the model did not systematically miss any hidden clusters among participants; the ADHD and non-ADHD individuals largely occupy overlapping regions of feature space.

The XGBoost regression model performed reasonably well in predicting total ASRS scores, with a test RMSE of 11.70, approximately 16% of the scale's 72-point range. Performance was best in the mid-range but struggled at the extremes, systematically underpredicting high ASRS scores and overpredicting low ones (Figure 16a). This bias may stem from the squared error loss, which penalizes large errors quadratically. In regions with sparse data (e.g., the extremes), the model defaulted to predictions closer to the mean, a behavior that could be due to the built-in regularization of the method. By default, XGBoost

"shrinks" its predictions toward a baseline on each tree's node, which reduces variance but increases bias. In our case, with the limited sample, the model favored more central predictions over trying to fit outliers. It seems reasonable to assume that a very flexible model on a small dataset can underfit at the tails (adding parameters lowers bias but raises variance).

The random distribution of residuals in PCA space (Figures 16b and 27) further supports the idea that the errors are not systematically tied to feature combinations but instead a result of global model limitations. The nearest-neighbor plots (Figure 17) reinforce this, where residuals in feature space showed no specific pattern or clustering, suggesting that the model's errors are not local to certain regions of the input space but instead arise from not being able to learn the nonlinear relationships.

For nearest neighbors based on residual value (Figure 17b), the majority of points are centered around (1,1,1). This means that points with similar ASRS scores have similar residuals (all underpredicted or all overpredicted by similar amounts), forming the center of the cone. As we move from middle scores toward extremes, the magnitude of residuals increases consistently, but their sign (positive/negative) is predictable based on the ASRS range. This creates an expanding cone shape as the relationships become more varied but still structured. Symmetry likely reflects that the model's bias is proportional to how extreme the score is, whether extremely low or extremely high.

Finally, there seems to be no difference in how test points (red) and training points (blue) behave, as they both appear in the same regions in both plots in Figure 17. The same global limitations that affect the training set carry over to unseen data.

The SHAP analysis revealed that age and chronotype were the most influential predictors (Figure 10), together explaining roughly half of the model's summed SHAP importance. Younger participants and those with more evening-type chronotypes (lower feature values) pushed the prediction higher, suggesting that younger "night-owl" participants receive the largest positive contributions to their ASRS score. This is consistent with existing research linking circadian preferences to ADHD symptoms; for example, college students with higher ADHD traits are far more likely to be "evening types" than controls (Becker et al., 2024). The features `total_moves` and `mean_switch_cost`, both measures of executive function and planning, showed moderate influence and may reflect difficulties in task switching or goal maintenance, common in ADHD.

Surprisingly, `go_rtv` had zero impact, contradicting expectations that attention variability would strongly predict ADHD (Epstein et al., 2011; Kofler et al., 2013). This could again be due to the regularization in XGBoost, which suppresses weak or noisy features, or from interactions between `go_rtv` and other variables that were not explicitly modeled. The absence of SHAP impact for Go RTV, despite its structural similarity to the Go/No-Go RTV, could also be explained in psychology terms, where it suggests that inhibitory control, rather than simple reaction time variability, is more diagnostic in this context.

Moreover, MDS projections of SHAP profiles (Figure 19a) showed that participants grouped into two clusters (although not so dense), suggesting that

the model relies on at least two different ways to make predictions. One could attribute this to the additive property of Shapley values (Equation 58), where features contribute independently but may be expressed differently across instances. Normalizing the SHAP values removed this structure (Figure 19b), implying that the relative importance of features is consistent across participants, even when absolute contributions vary.

Hence, predictions are dominated by age and chronotype, followed by planning ability (`total_moves`) and other response-time related variables (except `go_rtv`) forming the second tier. Accuracy-oriented scores (`reverse_max` and `correct_trials_c3`) contribute marginally.

Counterfactual analysis provided a complementary view of feature influence. Results show that no single small change suffices, as most participants require changes in ≥ 8 of 9 features. When feature ranges were unconstrained, the optimization algorithm was able to find valid counterfactuals for all seven participants with predicted ASRS scores above 30, changing an average of 8.14 features out of 9 (Table 6). This could be a result of the regression model relying on global interactions, rather than isolated features, which is a consequence of XGBoost's ensemble structure (Equation 22), where predictions emerge from additive contributions across many trees. The most frequently changed features matched those identified as impactful in the SHAP analysis (e.g., `total_moves`, `gono_rtv`).

Minimality scores showed that most of these changes were not essential, indicating that a few large changes in feature values were enough to cross the decision threshold. However, since counterfactual explanations want to find the *minimal* change, the alterations were spread over multiple features. The reason is the sparsity term in (71), which penalizes the count of altered features, not the magnitude of each alteration. The multi-objective optimization therefore prefers to make many small moves, rather than a few large ones that would inflate the similarity or plausibility objectives (objectives c_2 and c_4, respectively). The resulting counterfactuals are sparse in dimension but not necessarily in total Euclidean distance, consistent with what is observed in Table 6.

However, when more "realistic" bounds were imposed (± 1.5 SD for cognitive features), only a single counterfactual was found. This could imply the model not being able to generalize outside the narrow training distribution, or it being sensitive to multivariate interactions. For example, reducing `gono_rtv` alone might not suffice without simultaneously changing age or chronotype. We can also understand this as a reflection of the high-dimensional optimization problem in Equation (67), where satisfying multiple objectives (prediction shift, similarity, sparsity, plausibility) becomes computationally challenging. The results suggest that actionable interventions would require full changes across cognitive and demographic domains, which is a finding with limited clinical applicability but great methodological implications.

It is very important to note that counterfactual methods assume that one can vary features independently and in meaningful compatible units. However, these assumptions may not fully hold here, as for instance; it is not obvious how a person could specifically "increase" their correct trials in a task or shift their

chronotype at will. Thus, our counterfactual results are best seen as illustrating the model's logic (showing that decisions are globally stable), rather than as realistic means of action.

Turning to classification, the poor performance of classification models (XGBoost and kNN) can be attributed to a combination of class imbalance (78 positive vs. 278 negative cases) and severe class overlap. Only a few participants were labeled to need screening, thus the training set was heavily skewed toward the negative class, and under great class imbalance, many learning algorithms will default to predicting the majority class without class-weighting or resampling.

In such setting, the optimal separating surface may be highly nonlinear and irregular, and with limited data (particularly in the minority class) models like XGBboost an kNN may underfit or simply fail to learn the boundary. While XGBoost's tree ensembles can theoretically model such boundaries, the small sample size and imbalance, together with not being able to build deeper trees, prevent the algorithm from learning properly. Via the bias-variance tradeoff philosophy: increasing model complexity (e.g., deeper trees) risks overfitting (higher variance), while simplicity fails to detect necessary patterns.

In addition, because the dataset contains 78% negative cases, the optimizer sees far more opportunities to reduce the loss on that majority class. Since the logistic loss is class-agnostic, each minority-class error costs the same but occurs far less often [7]. Consequently, the optimizer minimizes the total loss fastest by nearly perfecting class 0 at the expense of class 1, which explains the near-zero specificity (4%).

The kNN baseline model, which has no internal optimization, also failed to classify any positive cases (Table 5), indicating that the poor performance was not due to the complexity of XGBoost, but instead due to the dataset's structure.

This is also evident in the validation curves in Figure 20, where we see high validation error without a clear underfitting phase, which is typically characterized by a steep drop in the beginning of learning. This implies that neither model began to meaningfully learn the decision boundary. Additional attempts to improve classification performance through feature subset selection were made. Evaluating all 252 combinations of choosing 5 features at both tree depths 2 and 3, yielded no improvement in test RMSE or confusion matrices, suggesting that the failure was not due to suboptimal feature choice but rather limitations in the feature signal.

A number of limitations should be kept in mind. First, the sample size was small (order of a few hundred observations), especially for the positive class in classification, restricting generalization. Gaussian mixture models, which require sufficient data to estimate covariance matrices (Equation 11b), were probably affected, among the other machine learning methods. This together with great feature overlap hindered both regression and classification.

[7] A loss function is class-agnostic when a misclassification of either class, given the same confidence error, is penalized equally.

Second, the cognitive task features proved to be only weakly informative, as seen in the SHAP analysis. In a large independent study using similar cognitive measures as features (in thousands of youths), machine learning models explained only 15-20% of the variance in ADHD symptom scores (Weigard et al., 2023). This suggests that cognitive task performance captures only a small portion of the underlying ADHD variation. In addition, some of the features used in this analysis were discrete (e.g., `total_moves`), yet were modeled as continuous, which may violate the assumptions of for example GMM.

Third, ASRS scores are self-reported, introducing potential measurement noise and bias, and may not correspond to clinical diagnosis. Lastly, The NSGA-II algorithm (Section 2.3.3) generated theoretically valid but clinically unrealistic explanations, as cognitive features cannot be arbitrarily changed in practice. All of these factors should caution against over-interpreting the results found.

Despite these limitations, this analysis provides valuable groundwork for future projects. To improve predictive capability, collecting more data with a better balance of ADHD-positive cases should be prioritized, which can in turn open up possibilities for more complex machine learning models. Additional feature engineering, or the inclusion of other cognitive tasks, could also be considered. From an interpretability standpoint, extending the SHAP analysis to include interaction effects would help to better understand how combinations of features jointly influence model predictions. Finally, refining counterfactual constraints using clinical or empirical benchmarks could improve realism and utility.

5 Conclusion

This thesis evaluated the potential of using cognitive task performance, age, and chronotype to predict adult ADHD symptoms as measured by the ASRS. The results demonstrated that the XGBoost model could not reliably predict ASRS-based ADHD symptoms from cognitive task performance, age and chronotype. The extreme overlap of feature distributions and the small sample size prevented the model from learning distinguishable behaviors. On the other hand, the SHAP and counterfactual analyses were valuable for understanding the model's logic, as they showed which features had any effect on predictions (e.g., chronotype), and confirmed that most features had only minor influence.

Overall, while our current model did not yield a reliable clinical predictor, the findings highlight both the promise and the current limitations of applying interpretable machine learning to psychological self-report data.

6 Code Availability

Code supporting this project is published online at https://github.com/a-myllymaki/master_thesis

The main `R` packages used in the analysis include `mclust` for GMM-clustering, `stats` for PCA, `xgboost` for XGBoost modeling, `SHAPforxgboost` for Tree-SHAP value computation, `iml` and `counterfactuals` for counterfactual analysis, `ggplot2` and `patchwork` for data visualization, `dplyr` and `data.table` for data manipulation, and `caret` and `Metrics` for performance evaluation. All analyses were performed in `R` version 4.4.1.

References

Aas, K., Jullum, M., & Løland, A. (2021). Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artificial Intelligence*, *298*, 103502. https://doi.org/https://doi.org/10.1016/j.artint.2021.103502

Ayano, G., Tsegay, L., Gizachew, Y., Necho, M., Yohannes, K., Abraha, M., Demelash, S., Anbesaw, T., & Alati, R. (2023). Prevalence of attention deficit hyperactivity disorder in adults: Umbrella review of evidence generated across the globe. *Psychiatry Research*, *328*, 115449. https://doi.org/https://doi.org/10.1016/j.psychres.2023.115449

Becker, S. P., Luebbe, A. M., Kofler, M. J., Burns, G. L., & Jarrett, M. A. (2024). Adhd, chronotype, and circadian preference in a multi-site sample of college students. *Journal of sleep research*, *33*(1), e13994.

Bentler, P. M., & Chou, C.-P. (1987). Practical issues in structural modeling. *Sociological Methods & Research*, *16*(1), 78–117. https://doi.org/10.1177/0049124187016001004

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag.

Boyd et al. (2004). *Convex optimization*. Cambridge University Press. https://web.stanford.edu/~boyd/cvxbook/

Bzdok, D., & Meyer-Lindenberg, A. (2018). Machine learning for precision psychiatry: Opportunities and challenges. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, *3*(3), 223–230. https://doi.org/https://doi.org/10.1016/j.bpsc.2017.11.007

Campello, R., & Hruschka, E. (2006). A fuzzy extension of the silhouette width criterion for cluster analysis. *Fuzzy Sets and Systems*, *157*(21), 2858–2875. https://doi.org/https://doi.org/10.1016/j.fss.2006.07.006

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Dandl, S., et al. (2020). Multi-objective counterfactual explanations. *Parallel Problem Solving from Nature – PPSN XVI*, 448–469.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. https://doi.org/10.1109/4235.996017

Dempster, A. P., et al. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, *39*(1), 1–22. https://doi.org/https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

Developers, X. (2022). Xgboost release notes [Accessed: 2025-04-16].

Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv: Machine Learning*.

Dwyer, D. B., Falkai, P., & Koutsouleris, N. (2018). Machine learning approaches for clinical psychology and psychiatry. *Annual Review of Clinical Psychology*, *14*(Volume 14, 2018), 91–118. https://doi.org/https://doi.org/10.1146/annurev-clinpsy-032816-045037

Epstein, J. N., Langberg, J. M., Rosen, P. J., Graham, A., Narad, M. E., Antonini, T. N., Brinkman, W. B., Froehlich, T., Simon, J. O., & Altaye, M. (2011). Evidence for higher reaction time variability for children with adhd on a range of cognitive tasks including reward and event rate manipulations. *Neuropsychology*, *25*(4), 427.

Fraley, C., & Raftery, A. (2007). Model-based methods of classification: Using the mclust software in chemometrics. *Journal of Statistical Software*, *18*(6), 1–13. https://doi.org/10.18637/jss.v018.i06

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, *27*(4), 857–871.

Guo, J., Ren, L., Zhu, X., Zhuang, J., Jiang, B., Liu, C., & Wang, L. (2024). Pseudo-huber loss function-based affine registration algorithm of point clouds. *2024 39th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 1034–1039. https://doi.org/10.1109/YAC63405.2024.10598436

Hart, S. (1989). Shapley value. In J. Eatwell, M. Milgate, & P. Newman (Eds.), *Game theory* (pp. 210–216). Palgrave Macmillan UK. https://doi.org/10.1007/978-1-349-20181-5_25

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Springer New York Inc.

Kim, Pack, S. P., Lim, Cho, & Lee. (2023). Machine learning–based prediction of attention-deficit/hyperactivity disorder and sleep problems with wearable data in children. *JAMA network open*, *6*(3), e233502–e233502.

Kofler, M. J., Rapport, M. D., Sarver, D. E., Raiker, J. S., Orban, S. A., Friedman, L. M., & Kolomeyer, E. G. (2013). Reaction time variability in adhd: A meta-analytic review of 319 studies. *Clinical Psychology Review*, *33*(6), 795–811. https://doi.org/https://doi.org/10.1016/j.cpr.2013.06.001

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S.-I. (2020). From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, *2*(1), 56–67.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 4768–4777.

McLachlan et al. (2012). The em algorithm. In *Handbook of computational statistics: Concepts and methods* (pp. 139–172). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-21551-3_6

Molnar, C. (2019). *Interpretable machine learning: A guide for making black box models explainable* [https://christophm.github.io/interpretable-ml-book/]. https://christophm.github.io/interpretable-ml-book/.

Molnar, C. (2022a). *Interpretable machine learning* [Accessed: 2025-04-16]. https://christophm.github.io/interpretable-ml-book/shap.html

Molnar, C. (2022b). *Interpretable machine learning* [Accessed: 2025-04-23]. https://christophm.github.io/interpretable-ml-book/counterfactual.html

Muthén, L. K., & and, B. O. M. (2002). How to use a monte carlo study to decide on sample size and determine power. *Structural Equation Modeling: A Multidisciplinary Journal*, *9*(4), 599–620. https://doi.org/10.1207/S15328007SEM0904_8

Polanczyk, G., Horta, B. L., Biederman, J., & Rohde, L. A. (2007). The worldwide prevalence of adhd: A systematic review and metaregression analysis. *American Journal of Psychiatry*, *164*(6), 942–948. https://doi.org/10.1176/ajp.2007.164.6.942

Psutka, J. V., & Psutka, J. (2019). Sample size for maximum-likelihood estimates of gaussian model depending on dimensionality of pattern space. *Pattern Recognition*, *91*, 25–33. https://doi.org/https://doi.org/10.1016/j.patcog.2019.01.046

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65. https://doi.org/https://doi.org/10.1016/0377-0427(87)90125-7

Shapley, L. S. (1953). 17. a value for n-person games. In H. W. Kuhn & A. W. Tucker (Eds.), *Contributions to the theory of games, volume*

ii (pp. 307–318). Princeton University Press. https://doi.org/doi:10.1515/9781400881970-018

Spencer, T. J. (2006). Adhd and comorbidity in childhood. *Journal of Clinical Psychiatry*, *67*, 27.

Tai, A. M., Albuquerque, A., Carmona, N. E., Subramanieapillai, M., Cha, D. S., Sheko, M., Lee, Y., Mansur, R., & McIntyre, R. S. (2019). Machine learning and big data: Implications for disease modeling and therapeutic discovery in psychiatry. *Artificial Intelligence in Medicine*, *99*, 101704. https://doi.org/https://doi.org/10.1016/j.artmed.2019.101704

Wachter, S., Mittelstadt, B., & Russell, C. (2018). Counterfactual explanations without opening the black box: Automated decisions and the gdpr. https://arxiv.org/abs/1711.00399

Weigard, A., McCurry, K. L., Shapiro, Z., Martz, M. E., Angstadt, M., Heitzeg, M. M., Dinov, I. D., & Sripada, C. (2023). Generalizable prediction of childhood adhd symptoms from neurocognitive testing and youth characteristics. *Translational Psychiatry*, *13*(1), 225.

Wright, S. J., & Nocedal, J. (2006). *Numerical optimization* (2nd). Springer New York, NY.

Wright, S. J., & Wright, M. H. (2018). *Optimization: Principles and algorithms* (2nd). Cambridge University Press.

Xie et al. (1991). A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*, 841.

A Mathematical Derivations

A.1 Derivation of GMM Parameters

The goal is to verify Equations (11a),(11b),(11c) by differentiating the log-likelihood

$$\ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} p(k) \mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) \right),$$

where

$$\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right).$$

Verifying Equations (11a) **and** (12)
Solving $\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0$.

Since the log-likelihood involves a sum over all data points and all components, and only the derivative with respect to $\boldsymbol{\mu}_k$ is of interest, the focus is on the term inside the log that involves $\boldsymbol{\mu}_k$. This yields:

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln \left(\sum_{k=1}^{K} p(k) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right) = \frac{1}{\sum_{j=1}^{K} p(j) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \cdot \frac{\partial}{\partial \boldsymbol{\mu}_k} \left(p(k) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right).$$

Note: The index in the denominator is changed to j because the summation there is a sum over *all* components, not just the specific component k that we are differentiating with respect to. To avoid confusion and make it clear that the summation is generic over all components in the mixture, j is used instead.

Now, the derivative of the Gaussian $\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ can be computed using the following matrix multiplication property. Let $\boldsymbol{W} \in \mathbb{R}^{D \times D}$ be a symmetric and invertible matrix, and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^D$ be column vectors independent of $\boldsymbol{W}$. Then:

$$\boxed{\frac{\partial}{\partial \boldsymbol{s}} (\boldsymbol{a} - \boldsymbol{s})^T \boldsymbol{W} (\boldsymbol{a} - \boldsymbol{s}) = -2\boldsymbol{W}(\boldsymbol{a} - \boldsymbol{s})}.$$

This yields

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\mu}_k} \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) &= \frac{\partial}{\partial \boldsymbol{\mu}_k} \left(\frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right) \right) \\
&= \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right) \\
&\cdot \frac{\partial}{\partial \boldsymbol{\mu}_k} \left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right) \\
&= \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right) \\
&\cdot \left(-\frac{1}{2} (-2\Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right) \\
&= \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right) \cdot \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \\
&= \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \cdot \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k).
\end{aligned}$$

The derivative of the Gaussian is substituted back into the log-likelihood derivative:

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \frac{p(k) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} p(j) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \cdot \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k).$$

Recall from (9) that

$$\frac{p(k) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} p(j) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = p(k \mid \mathbf{x}_n).$$

Thus, the derivative becomes:

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n) \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k).$$

Next, the derivative is set equal to zero and is solved for $\boldsymbol{\mu}_k$:

$$\sum_{n=1}^{N} p(k \mid \mathbf{x}_n) \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) = 0.$$

Multiplying both sides by $\boldsymbol{\Sigma}_k$ and rearranging:

$$\begin{aligned}
\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k) &= 0 \\
\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\mathbf{x}_n &= \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\boldsymbol{\mu}_k \\
\boldsymbol{\mu}_k &= \frac{\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\mathbf{x}_n}{\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)}.
\end{aligned}$$

This gives

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\mathbf{x}_n, \qquad (11a)$$

where

$$N_k = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n). \qquad (12)$$

Verifying (11b)**:**
Solving $\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0$.
Following a similar procedure to the derivation of $\boldsymbol{\mu}_k$, yields:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \ln \left(\sum_{k=1}^{K} p(k) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right) = \frac{p(k)}{\sum_{j=1}^{K} p(j) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \cdot \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \left(\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right).$$

Now, the derivative of the Gaussian $\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \left(\frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right) \right).$$

Using the product rule for derivatives yields

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) =$$

$$= \frac{\exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right)}{(2\pi)^{D/2}} \cdot \left[\underbrace{\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \left(\frac{1}{|\boldsymbol{\Sigma}_k|} \right)}_{\text{A}} + \frac{1}{|\boldsymbol{\Sigma}_k|} \cdot \underbrace{\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right)}_{\text{B}} \right].$$

Next, the following matrix multiplication property is used to compute the derivatives of A and B. Let $\boldsymbol{W} \in \mathbb{R}^{D\times D}$ be a symmetric, invertible matrix, and let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^D$ be fixed column vectors that do not depend on $\boldsymbol{W}$. Then

$$\boxed{\frac{\partial |\boldsymbol{W}|}{\partial \boldsymbol{W}} = |\boldsymbol{W}|\boldsymbol{W}^{-T}} \qquad \text{and} \qquad \boxed{\frac{\partial \boldsymbol{a}^T\boldsymbol{W}^{-1}\boldsymbol{b}}{\partial \boldsymbol{W}} = -\boldsymbol{W}^{-T}\boldsymbol{a}\boldsymbol{b}^T\boldsymbol{W}^{-T}}.$$

With this, the derivatives of A and B are

$$A: \frac{\partial}{\partial \boldsymbol{\Sigma}_k}\left(\frac{1}{|\boldsymbol{\Sigma}_k|}\right) = -\frac{1}{2}|\boldsymbol{\Sigma}_k|^{-3/2}|\boldsymbol{\Sigma}_k|\boldsymbol{\Sigma}_k^{-1} = -\frac{1}{2}|\boldsymbol{\Sigma}_k|^{-1/2}\boldsymbol{\Sigma}_k^{-1},$$
$$B: \frac{\partial}{\partial \boldsymbol{\Sigma}_k}\left(-\frac{1}{2}(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)\right) = \frac{1}{2}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}.$$

Inserting this back into $\frac{\partial}{\partial \boldsymbol{\Sigma}_k}\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, yields

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\Sigma}_k}\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \\
&= \frac{\exp\left(-\frac{1}{2}(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)\right)}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_k|}\cdot\left[-\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}+\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}\right] \\
&= \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\cdot\left[-\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}+\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}\right].
\end{aligned}$$

This can now be substituted back into the log-likelihood derivative:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k}\ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\left[-\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}+\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}\right].$$

To solve for Σ_k, the derivative is set equal to zero and the expression is simplified:

$$\begin{aligned}
&\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\left[-\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}+\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}\right] = 0 \\
&\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\left[-\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}\right] + \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)\left[\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}\right] = 0
\end{aligned}$$

The terms $-\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}$ and $\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}$ can be factored out, and both sides of the expression are multiplied with $2\boldsymbol{\Sigma}_k$:

$$\begin{aligned}
&-\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}\sum_{n=1}^{N} p(k \mid \mathbf{x}_n) + \frac{1}{2}\boldsymbol{\Sigma}_k^{-1}\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1} = 0 \\
&-\sum_{n=1}^{N} p(k \mid \mathbf{x}_n) + \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n-\boldsymbol{\mu}_k)(\mathbf{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1} = 0.
\end{aligned}$$

Now, isolate the sum:

$$\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n),$$

and multiply both sides by Σ_k:

$$\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T = \Sigma_k \sum_{n=1}^{N} p(k \mid \mathbf{x}_n).$$

Finally, isolate Σ_k:

$$\Sigma_k = \frac{\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)}.$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} p(k \mid \mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T, \qquad (11b)$$

where

$$N_k = \sum_{n=1}^{N} p(k \mid \mathbf{x}_n). \qquad (12)$$

Equation 11c:
Solving $\frac{\partial}{\partial \mathbf{p}_k} \ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0$.
The log-likelihood function is maximized with respect to $p(k)$ while enforcing the constraint that the mixing coefficients sum to one, i.e., $\sum_{k=1}^{K} p(k) = 1$. As *Pattern Recognition and Machine Learning* page 436 suggests, this constraint can be handled using a Lagrange multiplier (Bishop, 2006, p. 436). The term $\lambda \left(\sum_{k=1}^{K} p(k) - 1 \right)$ is added to the log-likelihood and the modified expression is maximized. Next, define the Lagrangian as:

$$\begin{aligned} \mathcal{L} &= \ln p(\mathbf{X} \mid \mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda \left(\sum_{k=1}^{K} p(k) - 1 \right) \\ &= \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} p(k) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right) + \lambda \left(\sum_{k=1}^{K} p(k) - 1 \right). \end{aligned}$$

The derivative of $\mathcal{L}$ is taken with respect to $p(k)$ and set to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial p(k)} &= \sum_{n=1}^{N} \frac{\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} p(j) \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda \\ &= \sum_{n=1}^{N} p(k \mid \mathbf{x}_n) + \lambda = 0. \end{aligned}$$

To solve for λ, both sides are multiplied by $p(k)$ and summed over all k:

$$\sum_{k=1}^{K} p(k) \sum_{n=1}^{N} p(k \mid \mathbf{x}_n) + \lambda \sum_{k=1}^{K} p(k) = 0.$$

Using the constraint $\sum_{k=1}^{K} p(k) = 1$ yields:

$$\sum_{k=1}^{K} p(k) \sum_{n=1}^{N} p(k \mid \mathbf{x}_n) + \lambda = 0.$$

Since $\sum_{k=1}^{K} \sum_{n=1}^{N} p(k \mid \mathbf{x}_n) = N$ (because the sum of responsibilities over all components for each data point is 1), this yields:

$$\lambda = -N.$$

Substitute $\lambda = -N$ back into the derivative equation:

$$\sum_{n=1}^{N} p(k \mid \mathbf{x}_n) - N p(k) = 0,$$

and solve for $p(k)$:

$$p(k) = \frac{\sum_{n=1}^{N} p(k \mid \mathbf{x}_n)}{N}.$$

Thus,

$$p(k) = \frac{N_k}{N}. \qquad (11c)$$

A.2 Second-Order Objective Optimization

The goal is to optimize the quadratic loss in (28). Because the first-order term $\ell(y_i, \hat{y}_i^{(m-1)})$ is independent of the tree that is fitted, it can be excluded during optimization. The simplified objective thus becomes

$$\tilde{\mathcal{L}}^{(m)} = \sum_{i=1}^{N} \left[g_i t_m(\mathbf{x}_i) + \frac{1}{2} h_i t_m^2(\mathbf{x}_i) \right] + \gamma L_m + \frac{1}{2} \lambda \sum_{j=1}^{L_m} \left(O_j^{(m)} \right)^2. \qquad (74)$$

This is now a weighted least squares problem with L2 regularization, where g_i act as residuals, h_i weight the importance of each instance and λ prevents extreme leaf values.

Having derived the main elements of the XGBoost framework, the full training procedure can be summarized. Algorithm 1 in Appendix A.3 outlines the high-level steps performed during boosting, while the tree-building method is detailed separately in Section 2.2.2.

In the next stage of the derivation, the tree structure s_m is assumed to be fixed, meaning each input $\mathbf{x}_i$ is already assigned to a leaf [8]. The goal is then to determine the optimal output value $O_j^{(m)}$ assigned to each leaf j.

Let $I_j \subseteq \{1, ..., N\}$ denote the set of data points, or *instances*, assigned to leaf j, such that (Chen & Guestrin, 2016, p. 3):

$$I_j = \{i \in \{1, ..., N\} \mid s_m(\mathbf{x}_i) = j\}. \tag{75}$$

Since all instances in a leaf receive the same prediction, the tree's output for any $i \in I_j$ satisfies

$$t_m(\mathbf{x}_i) = O^{(m)}_{s_m(\mathbf{x}_i)}. \tag{76}$$

Substituting this into (74) yields

$$\tilde{\mathcal{L}}^{(m)} = \sum_{j=1}^{L_m} \sum_{i \in I_j} \left[g_i O_j^{(m)} + \frac{1}{2} h_i (O_j^{(m)})^2 \right] + \gamma L_m + \frac{1}{2} \lambda \sum_{j=1}^{L_m} (O_j^{(m)})^2. \tag{77}$$

Grouping terms by leaf j gives

$$\tilde{\mathcal{L}}^{(m)} = \sum_{j=1}^{L_m} \left[O_j^{(m)} \sum_{i \in I_j} g_i + \frac{1}{2} (O_j^{(m)})^2 \left(\sum_{i \in I_j} h_i + \lambda \right) \right] + \gamma L_m. \tag{78}$$

The double sum over instances i and j in (77) collapses into a single sum over leaves because all $i \in I_j$ share the same $O_j^{(m)}$. The objective becomes

$$\tilde{\mathcal{L}}^{(m)} = \sum_{j=1}^{L_m} \left[G_j O_j^{(m)} + \frac{1}{2} (H_j + \lambda) (O_j^{(m)})^2 \right] + \gamma L_m, \tag{79}$$

where $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$. Next, the expression in (79) is minimized with respect to each $O_j^{(m)}$, assuming all G_j, H_j, λ are known. Taking the derivative and setting it equal to zero gives the optimal leaf value $O_j^{*(m)}$:

$$\frac{\partial}{\partial O_j^{(m)}} \left[G_j O_j^{(m)} + \frac{1}{2} (H_j + \lambda) (O_j^{(m)})^2 \right] = 0. \tag{80}$$

$$G_j + (H_j + \lambda) O_j^{(m)} = 0 \Rightarrow \quad O_j^{*(m)} = -\frac{G_j}{H_j + \lambda}. \tag{81}$$

If the gradients G_j are large in magnitude, it implies that the current model is making large errors on instances in leaf j, meaning $O_j^{*(m)}$ has to adjust aggressively. A large value of H_j, suggests steep curvature, prompting smaller updates to avoid overshooting, whereas a low value implies that larger corrective steps

[8] In practice, the tree structure is determined dynamically using greedy split finding; however, for the purposes of deriving the optimal leaf values, it is treated as fixed.

can be taken safely. The parameter λ stabilizes the optimization, especially when Hessians are small.

Substituting these optimal leaf values back into the loss yields a scoring function for evaluating candidate trees

$$\tilde{\mathcal{L}}^{(m)}(s) = -\frac{1}{2}\sum_{j=1}^{L_m}\frac{G_j^2}{H_j+\lambda} + \gamma L_m. \tag{82}$$

The first term in (82) measures how well the tree explains the data, and can be interpreted as a form of signal-to-noise ratio for each leaf. When the gradients are large and aligned (G_j large), and curvature is shallow (H_j small), this term grows large, indicating strong predictive contribution. The negative sign reminds that better predictive fit reduces the overall loss.

A.3 Algorithms

Algorithm 1: XGBoost training algorithm

Input: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, loss function ℓ, number of trees M, learning rate η, regularization parameters λ, γ

Output: Ensemble model $F(\mathbf{x}) = \sum_{m=1}^{M} \eta \cdot t_m(\mathbf{x})$

Initialize predictions: $\hat{y}_i^{(0)} \leftarrow \text{argmin}_c \sum_{i=1}^N \ell(y_i, c)$; `// e.g., mean of` y_i

for $m = 1$ **to** M **do**

  `// Compute gradients and Hessians`

  for $i = 1$ **to** N **do**

    $g_i \leftarrow \partial \ell(y_i, \hat{y}_i^{(m-1)}) / \partial \hat{y}_i^{(m-1)}$

    $h_i \leftarrow \partial^2 \ell(y_i, \hat{y}_i^{(m-1)}) / \partial (\hat{y}_i^{(m-1)})^2$

  `// Grow tree using a tree-building algorithm (see Alg. 2)`

  $t_m \leftarrow \text{TreeBuild}(\{g_i, h_i\}_{i=1}^N, \lambda, \gamma)$

  `// Update model predictions`

  for $i = 1$ **to** N **do**

    $\hat{y}_i^{(m)} \leftarrow \hat{y}_i^{(m-1)} + \eta \cdot t_m(\mathbf{x}_i)$; `// Apply shrinkage`

return $\hat{y}_i = \hat{y}_i^{(M)}$

Algorithm 2: Exact greedy split finding algorithm (XGBoost)

Input: Gradients $\{g_i\}$, Hessians $\{h_i\}$, regularization parameters λ, γ
Output: Regression tree t_m with optimal structure s_m and leaf outputs $\{O_j^{*(m)}\}$

Initialize root node with all training instances;
while *there exists a node that can be split* **do**
  foreach *node L in the current tree layer* **do**
    Compute total gradients and Hessians:
    $G \leftarrow \sum_{i \in L} g_i, \quad H \leftarrow \sum_{i \in L} h_i$;
    Initialize best gain $\mathcal{G}_{\max} \leftarrow -\infty$;
    foreach *feature $d \in \{1, \ldots, D\}$* **do**
      Sort instances in L by feature x_i^d **foreach** *threshold s in sorted feature x_i^d* **do**
        Compute left child stats:
        $G_L \leftarrow \sum_{\substack{i \in L \\ x_i^d \le s}} g_i, \quad H_L \leftarrow \sum_{\substack{i \in L \\ x_i^d \le s}} h_i$;
        Compute right child stats: $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$;
        Compute split gain:

$$\mathcal{G} \leftarrow \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}\right] - \gamma$$

        if $\mathcal{G} > \mathcal{G}_{max}$ **then**
          $\mathcal{G}_{\max} \leftarrow$ Gain;
          Store best split: $(d^*, s^*) \leftarrow (d, s)$;
    if $\mathcal{G}_{max} > 0$ **then**
      Split node L into L_L and L_R using (d^*, s^*);
      Assign instances to children based on $x_i^{d^*} \le s^*$;
foreach *leaf node j* **do**
  Assign optimal output:

$$O_j^{*(m)} \leftarrow -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

return *Tree t_m with structure s_m and outputs $\{O_j^{*(m)}\}$*

Algorithm 3: TreeSHAP with path-dependent feature perturbation

Input: Instance $\mathbf{x} \in \mathbb{R}^D$; ensemble $\{t_m\}_{m=1}^M$, each tree $t_m = \{v^{(m)}, a^{(m)}, b^{(m)}, \tau^{(m)}, r^{(m)}, d^{(m)}\}$

Output: SHAP vector $\boldsymbol{\varphi} \in \mathbb{R}^D$

procedure *TREESHAP_PATH(*$\mathbf{x}$, $\{t_m\}$*)*
- $\boldsymbol{\varphi} \leftarrow \mathbf{0}$;
- **for** $m \leftarrow 1$ **to** M **do** // accumulate over trees
  - $\boldsymbol{\varphi} = \boldsymbol{\varphi} +$ TREESHAP_TREE($\mathbf{x}$, t_m);
- **return** $\boldsymbol{\varphi}$

procedure *TREESHAP_TREE(*$\mathbf{x}$, t_m*)*
- $\boldsymbol{\varphi}^{(m)} \leftarrow \mathbf{0}$;

procedure *RECURSE(*j, γ, q_z, q_o, q_i*)*
- $\gamma \leftarrow \text{EXTEND}(\gamma, q_z, q_o, q_i)$; // Extend subset path with a fraction of zeros and ones
- **if** $v_j^{(m)} \neq$ *internal* **then**
  - ; // Check if we are at a leaf node
  - **for** $i \leftarrow 2$ **to** *len*(γ) **do**
    - ; // Calculate the contributions from every feature in our path
    - $w \leftarrow sum(\text{UNWIND}(\gamma, i).w)$; // Undo the weight extension for this feature
    - $\boldsymbol{\varphi}_{\gamma_i}^{(m)} = \boldsymbol{\varphi}_{\gamma_i}^{(m)} + w\,(\gamma_i.o - \gamma_i.z)\,v_j^{(m)}$; // Contribution from subsets matching this leaf
- **else**
  - $h, c \leftarrow (a_j^{(m)}, b_j^{(m)})$ **if** $x_{d_j^{(m)}} \leq \tau_j^{(m)}$ **else** $(b_j^{(m)}, a_j^{(m)})$; // Determine hot and cold children
  - $i_z \leftarrow i_o \leftarrow 1$; $k \leftarrow \text{FINDFIRST}(\gamma.d, d_j^{(m)})$;
  - **if** $k \neq$ *nothing* **then**
    - ; // Undo previous extension if we have already seen this feature
    - $(i_z, i_o) \leftarrow (\gamma_k.z, \gamma_k.o)$; $\gamma \leftarrow \text{UNWIND}(\gamma, k)$;
  - RECURSE$(h, \gamma, i_z\, r_h^{(m)}/r_j^{(m)}, i_o, d_j^{(m)})$;
  - RECURSE$(c, \gamma, i_z\, r_c^{(m)}/r_j^{(m)}, 0, d_j^{(m)})$;

procedure *EXTEND(*γ, q_z, q_o, q_i*)*
- $l, \gamma = len(\gamma), \text{copy}(\gamma)$;
- $\gamma_{l+1}.(d, z, o, w) = (q_i, q_z, q_o, (1 \text{ if } l = 0 \text{ else } 0))$; // Init subsets of size l
- **for** $i \leftarrow l$ **to** 1 **do**
  - ; // Grow subsets using q_z and q_o
  - $\gamma_{i+1}.w = \gamma_{i+1}.w + q_o \cdot \gamma_i.w \cdot (i/l)$; // Subsets that grow by one
  - $\gamma_i.w = q_z \cdot \gamma_i.w \cdot (l - i)/l$; // Subsets that stay the same size
- **return** γ ; // Return the new extended subset path

procedure *UNWIND(*γ, i*)*
- $l \leftarrow \text{len}(\gamma)$; $n \leftarrow \gamma_l.w$; $\gamma \leftarrow \text{copy}(\gamma_{1\ldots(l-1)})$;
- **for** $j \leftarrow l - 1$ **to** *1* **do**
  - ; // Shrink subsets using $\gamma_i.z$ and $\gamma_i.o$ **if** $\gamma_i.o \neq 0$ **then**
    - $\tau \leftarrow \gamma_j.w$; $\gamma_j.w \leftarrow n\,l/(j\gamma_i.o)$; $n \leftarrow \tau - \gamma_j.w\,\gamma_i.z\,(l - j)/l$;
  - **else**
    - $\gamma_j.w \leftarrow (\gamma_j.w\,l)/(\gamma_i.z\,(l - j))$;
  - **for** $j \leftarrow i$ **to** *l-1* **do**
    - $\gamma_j.(d, z, o) \leftarrow \gamma_{j+1}.(d, z, o)$
  - **return** γ
- RECURSE$(1, [\,], 1, 1, 0)$; // Start at first node with all zero and one extensions
- **return** $\boldsymbol{\varphi}^{(m)}$

Legend for Algorithm 3

Inputs / Output	
$\mathbf{x} \in \mathbb{R}^D$	instance to be explained
$\{t_m\}_{m=1}^M$	trained trees (see node arrays below)
$\boldsymbol{\varphi}$	final SHAP vector (D entries)

Per-node arrays in one tree t_m	
v_j	leaf value (or "internal" flag)
a_j, b_j	left / right child indices
τ_j	split threshold
r_j	cover (training-sample count)
d_j	feature index used for the split

Run-time variables	
γ	current path object holding *(d,z,o,w)* tuples
q_z, q_o	fractions of zero / one subsets propagated downward
q_i	feature index being appended to the path

Helper functions

RECURSE: depth-first traversal, accumulates SHAP mass

EXTEND: adds one feature to γ and updates weights

UNWIND: inverse of *EXTEND* (back-tracking). Removes one feature from γ and restores weights

FINDFIRST: returns index of first occurrence of feature d in γ (or `None` if d has not appeared yet)

For a detailed description of Algorithm 3, including its derivation and implementation details, please refer to the original paper by Lundberg et al. (2020).

A.3.1 The NSGA-II Algorithm

The NSGA-II algorithm evolves its solution set over multiple generations, with each generation involving the below steps. The algorithm for a generation can be summarized as follows (Deb et al., 2002):

Step 1: **Initialization.** Each generation begins with a population of candidate counterfactuals $\mathbf{x}^*$ that are generated by modifying the original instance $\mathbf{x}$ slightly. The candidates represent potential explanations, each one an alternative input that might shift the model's prediction.

Step 2: **Evaluation.** Each candidate is evaluated using the four objectives c_1 through c_4 (Molnar, 2022b).

Step 3: **Sorting.** The candidates are then ranked into "fronts" according to Pareto dominance, where a front is a set of candidates with the same level of *non-dominance*. A solution $\mathbf{x}_a^*$ dominates $\mathbf{x}_b^*$ if $\mathbf{x}_a^*$ is better in at least one objective and no worse in all others. Further, a candidate is considered non-dominated if not other candidate in the entire population dominates it.

The first Pareto front then consists of all non-dominated candidates, the second front contains candidates dominated only by members of the first front, and so on. Each front is therefore dominated only by members of earlier fronts, and by none of its own members. Figure 24 illustrates the first few Pareto fronts for a two-objective example.

Step 4: **Selection.** The algorithm selects parent candidates for reproduction, giving preference to those from higher-ranked fronts (less dominated solutions). Within the same front, selection favors candidates with greater diversity.

Step 5: **Recombination and mutation.** Chosen parents are recombined to create new candidates/offspring: numerical features may be averaged and categorical values swapped. Random mutations are also applied to introduce slight variation, allowing exploration of new regions in the feature space (Molnar, 2022b).

Step 6: **Replacement.** To form the next generation, parent and offspring populations are combined, and the NSGA-II then selects from this pool based on two criteria: non-domination rank and "crowding distance". The crowding distance favors candidates that lie in sparse regions of the objective space.

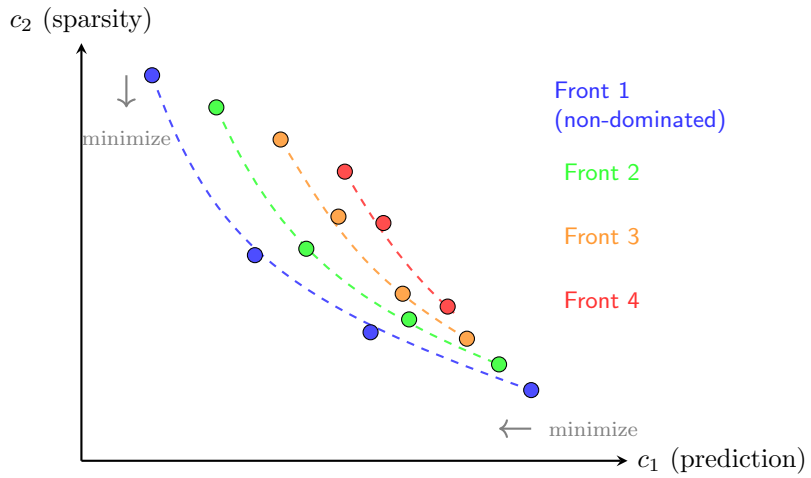


Figure 24: Visualization of Pareto front sorting in the NSGA-II algorithm. Each dot represents a candidate counterfactual solution evaluated on two objectives: prediction difference (c_1) and sparsity (c_2). Lower values are preferred for both objectives (indicated by gray minimization arrows). Candidates on the outermost curve (blue) form the first Pareto front, as they are not strictly dominated by any other solution in the entire population. Solutions in subsequent fronts (green, orange, red) are dominated by at least one candidate in all preceding fronts, meaning they are worse at minimizing both objectives simultaneously. NSGA-II ranks solutions by front and prefers diverse candidates within each front.

By iterating this process over multiple generations, NSGA-II produces a diverse set of counterfactuals that represent different solutions. From this set, one can either select individual counterfactuals of interest or summarize general patterns, such as which features are most commonly altered across the Pareto front. The iteration for one generation is illustrated in Figure 25.

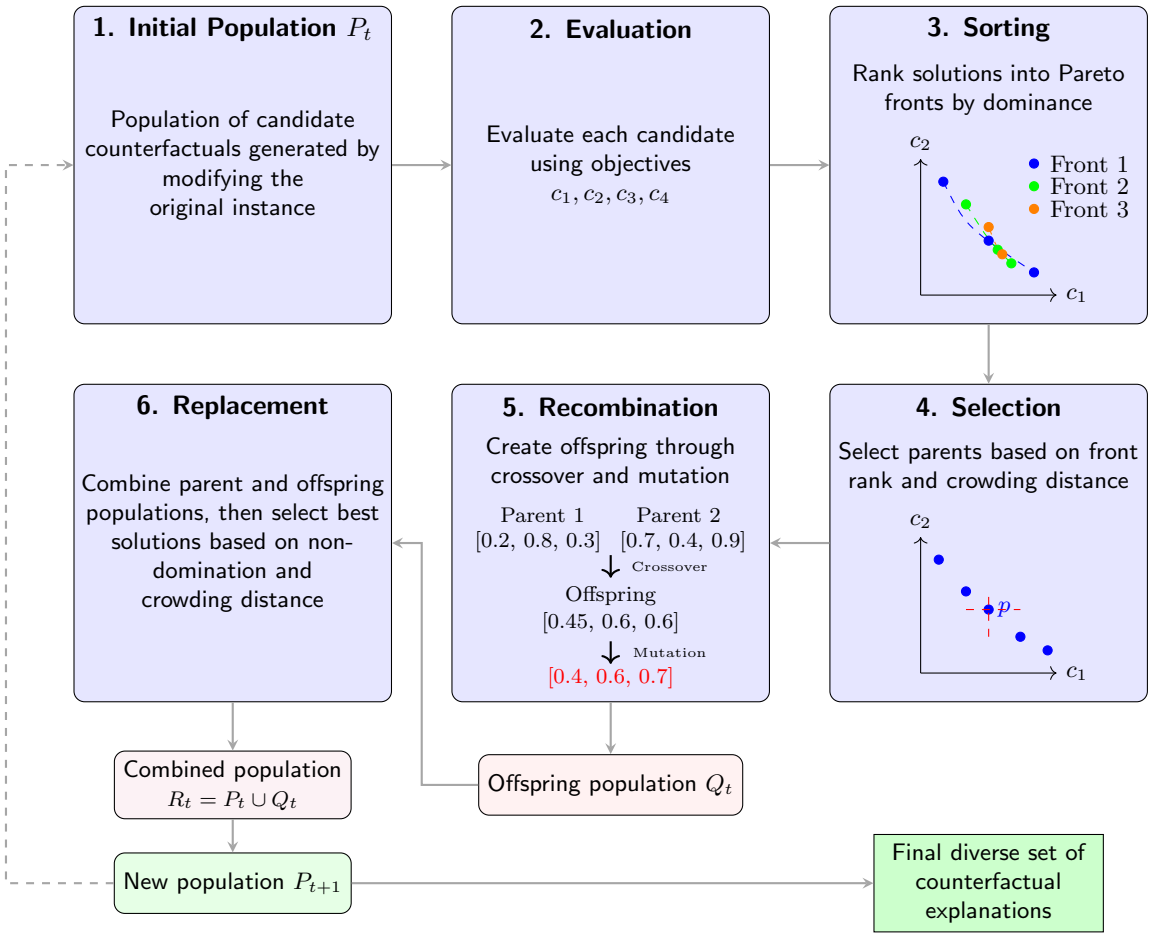


Figure 25: One generation of the NSGA-II algorithm for counterfactual explanation generation. The algorithm sorts solutions into Pareto fronts and selects parents based on front rank (prioritizing solutions in earlier fronts, e.g., Front 1 > Front 2). If solutions share a front rank (are from the same front), the one with larger crowding distance (i.e., isolated candidates) is preferred to maintain diversity. Finally, the algorithm creates new solutions through recombination and mutation of features.

B Data Details

B.1 ASRS Scoring

The Adult ADHD Self-Report Scale (ASRS v1.1) consists of 18 questions divided into two parts: Part A (6 questions) and Part B (12 questions). A binary ADHD screener flag was derived based on responses to the 6 questions in Part

A, following established scoring guidelines. A participant screens positive if they report "often" or "very often" on at least four of the six Part A items. The total ASRS score was calculated by summing the responses of all 18 items (Parts A and B), resulting in a theoretical score range of $0 - 72$. Higher scores indicate greater self-reported symptom severity.

In clinical use, the screener flag is typically used to determine whether a participant should undergo further diagnostic evaluation. The total score lacks a formally validated diagnostic cutoff. In this thesis, the screener flag was used as the classification target, while the total score served as the regression outcome.

B.2 Cognitive Task Description

Before each task, participants completed practice trials to ensure task comprehension. All tasks were administered via participants' own smartphones.

Design Fluency (`correct_trials_c3`)
Participants connected dots under varying rule sets to reach a goal position. In Condition 3, they alternated between black and white dots while avoiding repetitions. The feature represents the number of correct trials in this condition. Duration: 1 minute per condition.

Go (`go_rtv`)
Participants pressed a button as quickly as possible when a black square appeared. The feature represents response time variability (RTV). Duration: 3 minutes

Go/No-Go (`gono_rtv`)
Participants pressed a button as quickly as possible when red cards showed up, but withheld responses to black cards. The feature represents response time variability (RTV) for trials with red cards. Duration: 4 minutes

Stroop (`congruencyeffectRT`)
Participants were shown color words (e.g., "RED," "GREEN") displayed in incongruent ink colors (e.g., the word RED printed in green). They were required to choose the alternative corresponding to the ink color of the word while ignoring the written word itself. Four alternatives were given for each trial.

- Congruent Trials: The word meaning and ink color matched (e.g., **RED**).
- Incongruent Trials: The word meaning and ink color mismatched (e.g., **RED**)

The feature represents the mean response time (RT) difference between incongruent and congruent trials (incongruent RT – congruent RT). Duration: 2 minutes

Stroop switch (`mean_switch_cost`)
A variant of the Stroop task where participants alternated between two rules:

1. Name the ink color (ignoring the word).

2. Read the word (ignoring the ink color).

A cue indicated which rule to apply before each trial, requiring cognitive switching between tasks.

The feature represents the mean difference in response time between trials requiring a rule switch vs. those without a switch (switch RT – no-switch RT). Duration: 2 minutes

Grid (`reverse_max`)
Participants viewed a sequence of flashing grid squares and were asked to recall it in reverse order. The feature represents the longest correctly recalled sequence. Duration: 4.5 minutes per condition.

Tower of London (`total_moves`)
Participants moved colored blocks to match a target configuration using the fewest moves possible. The feature represents total moves across all trials.

B.3 Outlier Removal Criteria

Outliers were removed based on rules specified by the data providers. These included invalid trials, extreme values, or incomplete responses in either cognitive task performance or questionnaire data. Further details on exclusion thresholds are available upon request.

C Supplementary Figures and Tables

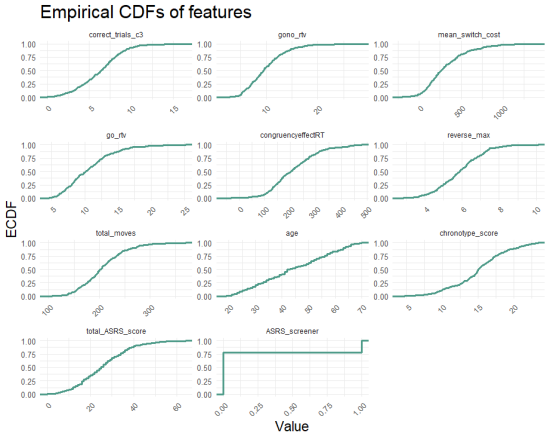


Figure 26: Empirical cumulative distribution functions (ECDFs) for all predictors. The smooth, continuous shapes suggest no clear multimodality or abrupt discontinuities, and therefore no strong signs of subgroup structure.

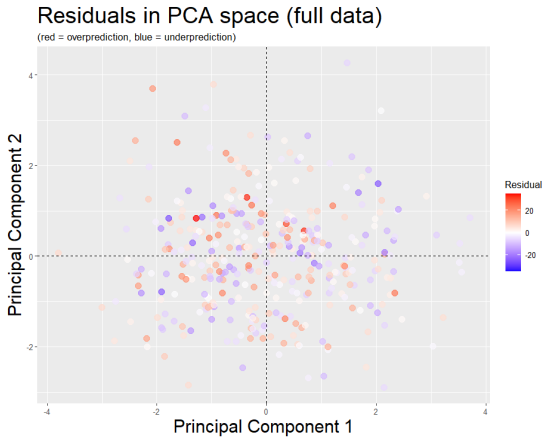


Figure 27: PCA projection of the full data set (training and testing), colored by residual values. No clear regional bias is visible, indicating residuals are randomly distributed in feature space.

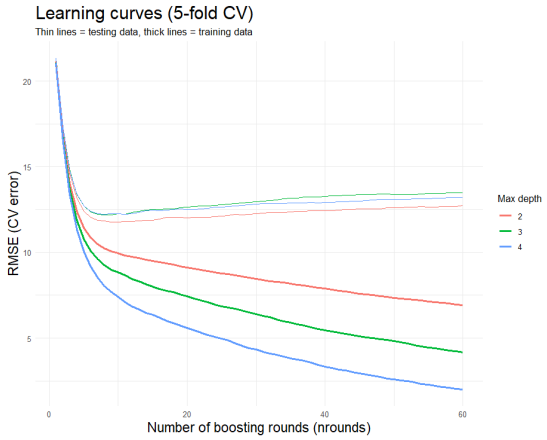


Figure 28: Learning curves for the XGBoost regression model using 5-fold cross-validation to select the optimal number of boosting rounds and maximum tree depth. Thin lines represent validation (test) performance, while thick lines represent training performance. Tree depth 2 achieves a good balance between bias and variance, with minimal gap between training and validation curves and relatively stable error across boosting rounds.