Clinical and Translational Science

TUTORIAL OPEN ACCESS

A Tutorial and Use Case Example of the eXtreme Gradient Boosting (XGBoost) Artificial Intelligence Algorithm for Drug Development Applications

Matthew Wiens[1] | Alissa Verone-Boyle[2] | Nick Henscheid[3] | Jagdeep T. Podichetty[3] | Jackson Burton[2]

[1]Metrum Research Group, Boston, Massachusetts, USA | [2]Biogen, Cambridge, Massachusetts, USA | [3]Critical Path Institute, Tucson, Arizona, USA

Correspondence: Jackson Burton (jackson.burton@biogen.com)

Received: 15 October 2024 | **Revised:** 28 January 2025 | **Accepted:** 6 February 2025

Funding: The authors received no specific funding for this work.

Keywords: boosting | machine learning | quantitative clinical pharmacology | XGBoost

ABSTRACT

Approaches to artificial intelligence and machine learning (AI/ML) continue to advance in the field of drug development. A sound understanding of the underlying concepts and guiding principles of AI/ML implementation is a prerequisite to identifying which AI/ML approach is most appropriate based on the context. This tutorial focuses on the concepts and implementation of the popular eXtreme gradient boosting (XGBoost) algorithm for classification and regression of simple clinical trial-like datasets. Emphasis is placed on relating the underlying concepts to the code implementation. In doing so, the aim is for the reader to gain knowledge about the underlying algorithm and become better versed with how to implement the algorithm functions for relevant clinical drug development questions. In turn, this will provide practical ML experience which can be applied to algorithms and problems beyond the scope of this tutorial.

JEL Classification: Artificial Intelligence and Machine Learning

1 | Introduction

The goal of this tutorial is to enable the reader to understand and implement the XGBoost machine learning (ML) algorithm for both classification and regression tasks. The implementation is in the R programming language, utilizing open-source packages and open-source clinical trial-like datasets given the wide use of R in the field of clinical pharmacology and pharmacometrics. The ideal audience for this guide is those familiar with clinical trial data and some proficiency in R coding, but with no significant experience in hands-on implementation of machine learning algorithms. Significant emphasis will be given to the conceptual framework that underpins the XGBoost algorithm to help the reader cultivate intuition behind the methods. This tutorial will not only aid the reader in understanding the implementation itself, but also address some of the "black box" vernacular that is common with artificial intelligence (AI) approaches. Note that XGBoost is only one of the many choices available for AI/ML algorithms. It was selected for this tutorial because of its popularity, flexibility, and generally good performance across a range of datasets.

1.1 | Tutorial Structure

For readers new to XGBoost, an introductory description and brief history of the algorithm are provided in addition to some features that differentiate it from existing methods for classification and regression.

Jagdeep Podichetty and Jackson Burton Co-last author.

The "AI/ML Applications in Drug development" section provides an overview of the types of use cases where XGBoost and general artificial intelligence/machine learning (AI/ML) approaches have been applied in drug development.

The "Working Example" section provides an overview of the problem, including the underlying dataset used for the implementation of the classification example. The focus for this hands-on example is to provide comprehensive code implementation details. A second dataset and corresponding regression problem are then described, and details of adapting the code from classification to regression are provided.

The "Background Concepts" section contains the underlying technical aspects of the XGBoost algorithm accompanied by a basic numeric example. The basic example borrows data from the working example but only for illustration purposes. The section concludes with an overview of the common hyperparameters and optimization features of XGBoost.

The "Data Pre-processing" section briefly covers the importance of understanding the clinical and biological context of the data. Within this section, the "Code for Data Preprocessing and Model Preparation" provides step-by-step instructions for preparing the data for building the XGBoost model. These include common transformations of different data types (categorical and ordinal variables) using R data wrangling packages such as dplyr.

The "Code Implementation for Classification" section covers step-by-step instructions for implementing the classification working example. Detailed descriptions of the code including function inputs and outputs are included. Code for model implementation and evaluation are also included. Special attention is given to hyper parameter tuning for optimizing the model performance and as a key general ML principle. Subsequently, in the "XGBoost for Regression" section, code is provided for adapting the code for a regression problem on a separate dataset. The reader is encouraged to generate this model based on learnings from the classification example.

Lastly, the "Discussion/conclusions" section outlines a summary of the tutorial and reemphasizes the types of applications appropriate for XGBoost. Important caveats of using the oversimplified case study are discussed.

A supplemental glossary file accompanies the tutorial, which contains essential terms defined throughout the main body of the tutorial. Words bolded in the main body indicate that they are defined in the glossary. In the glossary, a definition is provided with references. In this setting, more informal references were selected (online tutorials and websites) to lower the barrier to access to relevant teaching material. The reader is encouraged to refer to the glossary and references as needed. The data for the working example and model files are located in a GitHub repository (https://github.com/metrumresearchgroup/ascpt-ml-tutorial/).

2 | What Is XGBoost?

XGBoost is a powerful and efficient learning method based on an implementation of gradient-boosted decision trees. It is typically used for supervised learning tasks, particularly regression and classification problems. At a high level, XGBoost works by combining weak learners, such as decision trees, sequentially, with each new learner effectively correcting errors made by the previous ones. Similar to other supervised learning methods, such as neural networks, XGBoost seeks to minimize a loss function, such as mean squared error for regression problems or crossentropy loss for classification problems. XGBoost does so by sequentially building trees to fit the negative gradient of the loss function with respect to the predictions, in contrast to neural network training via backpropagation.

The history of XGBoost is intertwined with the broader evolution of machine learning, particularly the field of ensemble learning techniques. Ensemble methods, which combine multiple models to improve predictive performance, have been around since the early days of machine learning. The development of boosting, an ensemble method, was significantly advanced by Jerome Friedman's seminal work on gradient boosting in 2001 [1]. Friedman's gradient boosting machine (GBM) provided a powerful framework for combining weak learners (typically decision trees) to form a strong predictive model. Building on this foundation, the introduction of XGBoost by Tianqi Chen and Carlos Guestrin marked a major leap forward in the efficiency and scalability of gradient boosting algorithms [2]. Their 2016 work, "XGBoost: A Scalable Tree Boosting System," presented at the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), demonstrated how novel algorithmic optimizations and system designs could significantly enhance the performance of gradient boosting models for real-world data. Specifically, XGBoost includes an array of techniques to handle missingness, imbalanced data via weighting and large datasets in addition to other powerful software-level optimizations.

3 | Advantages of XGBoost Compared With Alternative Methods

There are many supervised learning algorithms available, with logistic regression for classification and linear regression for regression being perhaps the most widely known. In addition to these classic methods, a wide range of AI architectures exists, from a vast range of neural network methods, nonlinear "kernel" methods such as support vector machines, tree-based methods, and various ensemble methods. Every architecture comes with performance and implementation overhead tradeoffs, with more complex methods offering potential state-of-the-art performance but typically at a high implementation cost. While XGBoost can at first appear complex to understand and tune compared with other methods, its superior performance and versatility often make it worth the investment, especially in scenarios where high accuracy and predictive power are crucial. Implementation time overhead is partially mitigated by the powerful software features of the XGBoost framework, such as automatic hyperparameter tuning and robust missingness handling. Moreover, XGBoost is advantageous due to its ability to efficiently handle large datasets, its robustness against overfitting thanks to regularization, and its flexibility in tuning through various hyperparameters. Additionally, XGBoost has robust open-source implementations in many major programming languages such as Python, C++, Julia, and R [3], with every

implementation providing the same powerful features, allowing the user to use their preferred language. It should be emphasized that XGBoost's strength is best demonstrated on large, complex datasets. Smaller datasets may be more appropriately analyzed with simpler methods. In ML terms, XGBoost has lower bias because the model is more flexible but higher variance because it is more sensitive to the data than parametric methods such as logistic regression or linear regression.

One of the notable features of XGBoost is its ability to handle missing data during training and prediction phases. This capability is achieved through a process called sparsity-aware split finding [2]. Additionally, XGBoost includes features like built-in crossvalidation, which is essential when evaluating algorithm performance. The algorithm also benefits from being able to run on multiple cores during training, significantly speeding up computation times and making it scalable for larger datasets. Table 1 provides a comparison of XGBoost to established methods for classification and regression.

4 | AI/ML Applications in Drug Development

Exploration of ML algorithms that aid in clinical drug development questions is becoming widespread. The ability to predict the likelihood of achieving a clinically meaningful risk or outcome is a focus across new AI/ML tools as they permeate through the drug development landscape. As these methods evolve, predictions and outputs from AI/ML models should be validated, with assumptions stated, and clinical relevance over existing models/methodologies highlighted. It is also imperative that clinical development teams are familiar with these methods in order to critically assess conclusions supported by AI/ML.

ML algorithms, such as XGBoost, can be used to support predictions of the risk of disease worsening [4, 5], patient response to a therapy type [6], clinical outcomes or response [7, 8], or the risk of an adverse event occurring [9, 10]. XGBoost has been utilized in ML efforts to determine cancer-related mutations in cell-free, circulating tumor DNA, where the feature importance aspect of the algorithm improved data dimensionality and overall model interpretability [11]. ML-based models are being explored for risk assessment in clinical settings. For instance, they are being used to assess the risk of mortality in individuals experiencing first-episode psychosis, which could aid in determining the most appropriate therapy [12]. Additionally, they are being applied to predict the risk of developing Type 1 diabetes earlier (before symptoms arise), using data from electronic health records [13]. Furthermore, ML algorithms are also being investigated to aid in therapeutic drug monitoring, including both prediction of the correct therapeutic dose and prediction of drug concentration or exposure, in contrast to traditional Bayesian approaches [14, 15]. Finally, digital or virtual twin models are beginning to gain traction within the medical community [16].

TABLE 1 | Comparison of XGBoost properties to established classification and regression methods.

Considerations	XGBoost	Logistic/linear regression
Accuracy	High	Moderate
Flexibility	High	Moderate
Regularization	Yes	Yes
Efficiency	High	Moderate
Feature importance	Yes	Yes (Indirectly)
Complexity	Moderate to High	Low
Computation	Moderate to High	Low
Tuning	Required	Minimal
Interpretability	Moderate	High
Ideal data quantity	High	Low to moderate

Note: Bold items are defined in the supplemental glossary.

ML is being used to strengthen and improve existing quantitative approaches in clinical development to build upon dose finding and selection methodologies. ML algorithms can help predict drug concentration, supplement or enhance pharmacokinetic model prediction, and can be applied in the development of critical exposure–response models informing optimal dose selection across different patient populations and age groups [17–21]. ML-based tools are being developed to aid clinicians in decision making—such as predicting the optimal therapeutic dose based on the likelihood of achieving a predefined pharmacodynamic response [22].

In the clinic, AI is highly prevalent in the fields of medical imaging and radiomics [23–27]. The application of AI has been prominent in oncology, advancing cancer imaging and detection [24, 25]. Furthermore, ML and deep learning algorithms have become prevalent in the radiomics processing of large medical dataset evaluations [28]. Ge and colleagues have reviewed numerous examples of using AI/ML to extract data from PET/CT images and successfully classify tumors based upon *EGFR* mutational status [29]. Despite the widespread use, more learning is needed on how AI/ML and DL algorithms can be more relevant in the clinic. During the model building process, developers should consider any potential data or subgroup imbalances, how to minimize such effects, improve the integrity of data annotations, support translation from research to clinical settings, and avoid bias [23, 25, 27, 30]. For example, while datasets of natural images such as those used to train generative AI methods are vast in size, medical datasets are much smaller, and patient characteristics, protocols, and data preprocessing methods can vary substantially across institutions [27].

Outside of the purview of XGBoost, we note that AI is also being used to mine and extract information from electronic health record data with natural language processing. These efforts help in structuring, interpreting, and organizing medical datasets for use in current and future clinical research endeavors [31–34].

5 | Working Example

For this tutorial, the task is to predict which patients will have recurrence of breast cancer using nine predictive features in

the open-source dataset "Breast Cancer" from the UC Irvine Machine Learning Repository (Table 2) [35]. This dataset is a common example and benchmark in the Machine Learning literature and consists of 286 patients with breast cancer. Since the outcome is binary (recurrence = yes or no), this is a binary classification task. Notable features of the dataset include imbalanced classes, with 201 out of 286 patients having no recurrence; mostly ordered categorical type data for the predictors; and missing values for covariates. Note that we selected this dataset as a baseline to illustrate the implementation details of XGBoost, with the goal of understanding the method rather than comparing XGBoost's performance to other approaches or achieving state-of-the-art results for this particular benchmark.

For a regression task, we consider a second dataset from a study on alcoholic drinks and liver function. This dataset can also be found on the UCI Machine Learning Repository [36]. In this example, the task is to predict the number of daily drinks based on several biomarkers of liver function, such as alkaline phosphatase levels. The model building is similar to the classification example, but we will discuss several key differences.

TABLE 2 | Breast cancer dataset features and their properties used in the working example.

Variable name	Type	Description
Class	Binary (Target)	Dependent Variable of the analysis. Coded as "no-recurrence-events," "recurrence-events"
Age	Ordinal	Age of the patient at the time of diagnosis
Menopause	Binary	Whether the patient is pre- or postmenopausal at time of diagnosis
Tumor size	Ordinal	The greatest diameter (in mm) of the excised tumor
Inv nodes	Ordinal	The number (range 0–39) of axillary lymph nodes that contain metastatic breast cancer visible on histological examination
Node caps	Binary	Lymph node capsular invasion (yes or no)
Deg-malig	Ordinal	The histological grade (range 1–3) of the tumor
Breast	Binary	Left or Right
Breast-quad	Categorical	Quadrant of the breast (left-up, left-low, right-up, right-low, central)
Irradiat	Binary	Has radiation therapy been used (yes or no)

Note: For each feature, the variable type is described, along with a brief description.

6 | Background Concepts

6.1 | Conceptual Foundations

The purpose of this section is to provide the reader with an understanding of the fundamental concepts pertaining to the XGBoost algorithm. Gaining an understanding of these concepts enables users to understand which applications are ideally suited for using XGBoost, how the syntax of code implementation relates to the mechanism of the algorithm, and how to evaluate/tune a model. As stated earlier, this section is intended to be bottom-up, beginning with the basic building blocks of the algorithm before advancing to more complex aspects using intuitive examples and illustrations.

6.2 | XGBoost Algorithm Concepts

The approach in XGBoost for assembling multiple weak learners (decision trees) to build a strong learner is based on gradient boosting. Conceptually, gradient boosting builds each new weak learner sequentially by correcting the errors, that is, the residuals, of the previous weak learner. Prediction for a new data point then makes use of the entire sequence of learners to arrive at a final prediction. In this manner, a more accurate overall prediction is generated. Note that more generally, gradient boosting performs sequential correction using the negative gradient of a loss function. For the most commonly used loss functions, these negative gradients are vectors of residuals, that is, actual minus predicted, so we will continue to use this intuitive terminology.

XGBoost uses decision trees as the weak learners and iteratively combines them into a single strong learner. Normal decision trees split the target based on the levels (or continuous value) of a feature into decision nodes. This process continues producing additional decision nodes until arriving at a final node, that is, a leaf node. The longest path from the parent node to the leaf node is called the tree depth (referred to as "tree_depth" in the R code). The general aim of decision trees was to generate leaf nodes with groups of targets that are as homogeneous as possible. The completed decision tree can then take new data as input and classify the input based on the decision node splits derived during training to classify the input.

Decision trees in XGBoost are used differently compared to the typical approach of making predictions directly. Instead, XGBoost trees are built to fit the *residuals* of the previous tree (or initial prediction) to improve the prediction iteratively. To help illustrate this overall process, an example is provided in Figure 1 with sequential steps. The decision tree embedded in this example is specific to the residual tree-building step of XGBoost with the intent to provide a clear depiction of how XGBoost proceeds algorithmically.

Step 1

$$F_0 = \log(odds) = log\left(\frac{\#\,Yes}{\#\,No}\right) = log\left(\frac{2}{3}\right) = -0.41$$

Tumor size (mm)	Radiation	Class (target)	F_0
30-34 (3)	Yes	Yes	-0.41
15-19 (1)	No	No	-0.41
25-29 (2)	Yes	No	-0.41
30-34 (3)	No	No	-0.41
40-44 (4)	No	Yes	-0.41

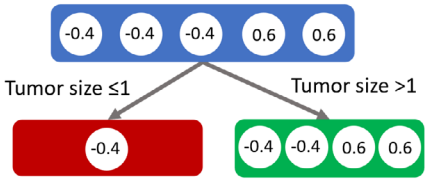
Step 2

$$p_0(x) = \frac{1}{1 + e^{-F_0(x)}} = \frac{1}{1 + e^{-(-0.41)}} = 0.4$$

$$Res_{F0} = (Observed - Predicted) = (Prob(obs) - p_0)$$

Tumor size (mm)	Radiation	Class (target)	F_0	Res_{F0} = observed(x)-$p_0(x)$
30-34 (3)	Yes	Yes	-0.41	1-0.40 = 0.60
15-19 (1)	No	No	-0.41	0-0.40 = -0.40
25-29 (2)	Yes	No	-0.41	0-0.40 = -0.40
30-34 (3)	No	No	-0.41	0-0.40 = -0.40
40-44 (4)	No	Yes	-0.41	1-0.40 = 0.6

Step 3



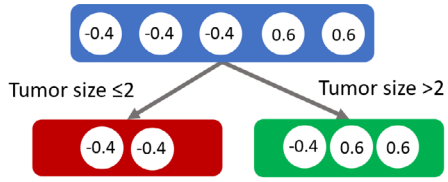
$$SS_{RN} = \frac{((-0.4) + (-0.4) + (-0.4) + (0.6) + (0.6))^2}{\sum_{i=1}^{5}[0.4 \times (1 - 0.4)]} = \frac{0}{1.2} = 0$$

$$SS_{TS\leq 1,LL} = \frac{(-0.4)^2}{0.4 \times (1 - 0.4)} = \frac{0.16}{0.24} = 0.66$$

$$SS_{TS>1,RL} = \frac{((-0.4) + (-0.4) + (0.6) + (0.6))^2}{\sum_{i=1}^{4}[0.4 \times (1 - 0.4)]} = \frac{0.16}{0.96} = 0.17$$

$$Gain_{tumorsize>1} = SS_{T\leq 1,LL} + SS_{T>1,RL} - SS_{RN}$$
$$= 0.66 + 0.17 + 0 = 0.83$$

Step 4



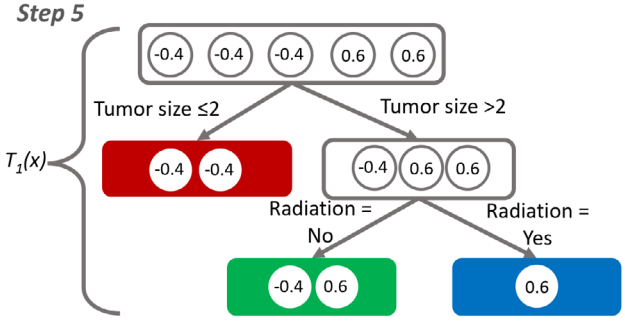
$$SS_{RN} = 0$$

$$SS_{TS\leq 2,LL} = \frac{((-0.4) + (-0.4))^2}{\sum_{i=1}^{2}[0.4 \times (1 - 0.4)]} = \frac{0.64}{0.48} = 1.33$$

$$SS_{TS>2,RL} = \frac{((-0.4) + (0.6) + (0.6))^2}{\sum_{i=1}^{3}[0.4 \times (1 - 0.4)]} = \frac{0.64}{0.72} = 0.88$$

$$Gain_{tumorsize>2} = SS_{T\leq 2,LL} + SS_{T>2,RL} - SS_{RN}$$
$$= 1.33 + 0.88 + 0 = 2.21$$

Step 5



$$LO_{LL} = \frac{((-0.4) + (-0.4))}{\sum_{i=1}^{2}[0.4 \times (1 - 0.4)]} = \frac{-0.8}{0.48} = -1.66$$

$$LO_{CL} = \frac{((-0.4) + (0.6))}{\sum_{i=1}^{2}[0.4 \times (1 - 0.4)]} = \frac{0.2}{0.48} = 0.417$$

$$LO_{RL} = \frac{0.6}{0.4 \times (1 - 0.4)} = \frac{0.6}{0.24} = 2.5$$

Step 6

Tumor size (mm)	Radiation	Class (target)	F_0	Res_{F0}	F_1	Res_{F1}
30-34 (3)	Yes	Yes	-0.41	0.6	0.04	1-(0.28)=0.72
15-19 (1)	No	No	-0.41	-0.4		
25-29 (2)	Yes	No	-0.41	-0.4		
30-34 (3)	No	No	-0.41	-0.4		
40-44 (4)	No	Yes	-0.41	0.6		

$$F_1(x) = F_0(x) + \varepsilon \times T_1(x)$$

$$F_1(x_{row\,1}) = F_0(x_{row\,1}) + \varepsilon \times T_1(x_{row\,1})$$
$$= -0.41 + 0.3 \times (-1.66)$$
$$= -0.91$$

$$p_1(x_{row\,1}) = \frac{1}{1 + e^{-F_1(x_{row\,1})}}$$

$$p_1(x_{row\,1}) = \frac{1}{1 + e^{-(-0.91)}}$$

$$p_1(x_{row\,1}) = 0.28$$

FIGURE 1 | XGBoost numerical example.

In the example presented, the key concepts of gradient boosting are discussed. Practicality, XGBoost has additional algorithmic features that help improve its performance and accuracy (discussed later). Second, the mathematical approach to gradient boosting differs somewhat between regression and classification. Here we focus on gradient boosting for classification in line with the example, but the reader is encouraged to examine the additional resources highlighted for background on regression.

Mathematically, gradient boosting can be summarized as

$$F_k(x) = F_0(x) + \varepsilon T_1(x) + \varepsilon T_2(x) + \ \ldots \ + \varepsilon T_k(x), \tag{1}$$

where k is the number of trees, x is the vector of features, F_k is the current strong learner, F_0 is the initial prediction (e.g., mean), ε is the learning rate, and T_n is the kth decision tree that helps in correcting the residuals from the previous prediction. It is important to note that for binary classification, Equation (1) predicts the log-odds which is defined here as the natural logarithm of the ratio of yes to no for the target. Because these need to be converted to class probabilities, we perform the inverse logit transform given by Equation (2). Which is the logistic function

$$p_k(x) = \frac{1}{1 + e^{-F_k(x)}}, \tag{2}$$

where p is the class probability and x is a record (row of the data). This is interpreted as the probability of a recurrence event occurring. Once a decision threshold is selected (using ROC analysis, for instance), the final binary classifier can be defined. The probabilities in Equation (2) will be used to calculate the residuals at each step.

A brief numerical example is developed here that uses a subset of the breast cancer dataset. We show the steps to perform the details of the calculation of Equation (1).

In Figure 1, Step 1 a subset of the breast cancer dataset is provided with features including tumor size (ordinal) and radiation (binary yes/no) with the outcome class (recurrence [yes]/nonrecurrence [no]). Because of the ordinal nature of tumor size, it is assigned ordered integer values to represent each ordered level of the feature as noted in parenthesis in the Table of Figure 1, Step 1. The initiation of XGBoost starts with an initial prediction defined as F_0 based on Equation (1), which is calculated as the observed log-odds, that is, log (# yes/# no) shown in Figure 1, Step 2. Note that this initial prediction is identical for all observations. In theory, any initial prediction can be used, including the output of another classifier such as logistic regression.

Next, the residuals for this prediction must be calculated as Res_F0 = (Observed—Predicted). Given that we are working with probabilities, observed values for recurrent events are defined as 1 while observed values for nonrecurrent events are defined as 0. The residuals are then calculated for F_0 shown in Figure 1, Step 2. Mathematically speaking, these values arise as the gradient of the crossentropy loss, but conceptually is not critical to understand this fact.

The next step highlights a key aspect of gradient boosting: A decision tree is built to *predict the residuals*. This approach iteratively corrects the errors of the previous prediction, making gradual improvements that result in a strong learner. This approach contrasts with methods like random forests, which use bootstrap aggregating ("bagging") to train multiple trees independently on random subsets of the data and combine their outputs, rather than iteratively refining predictions. The steps to build the residual tree are as follows.

An initial node called the root node is specified, which contains all the current residuals, derived either from the initial prediction or from a previous tree. For simplicity, let's begin with residuals from the initial prediction, as seen in Figure 1, Step 3. The features are now used to split the root node into leaf nodes. For the sake of illustration, we will compare two proposed thresholds of tumor size and demonstrate how the "better" split is chosen based on something called maximum gain.

The metric used for determining optimal splits is based on the similarity score (SS) given by

$$\text{SS} = \frac{\left(\sum_{i=1}^{n} residual_i\right)^2}{\sum_{i=1}^{n} \left[prior\, p_i \times \left(1 - prior\, p_i\right)\right] + \lambda}, \tag{3}$$

where $residual_i$ is the ith residual in the node, $prior\ p_i$ is previous predicted probability, λ is the regularization parameter, and n is the total number of residuals in the leaf. The parameter λ will be discussed later; for now, assume it is 0. An initial split for the leaf nodes is produced by using a threshold of tumor size above 1. The SS score is derived for each node as shown in Figure 1, Step 3. Note that in the SS, the residuals are summed first before squaring. Also note that initially, the $prior\ p_i$ values are simply the initial probabilities, p_0 derived in Figure 1, Step 2.

To determine an overall value for the quality of the split, a metric called gain is calculated, given by

$$\text{Gain} = \text{SS}_{\text{Left leaf}} + \text{SS}_{\text{Right leaf}} - \text{SS}_{\text{Parent}}, \tag{4}$$

where $\text{SS}_{\text{Left leaf}}$, $\text{SS}_{\text{Right leaf}}$, and $\text{SS}_{\text{Parent}}$ are the similarity scores for the left, right, and parent nodes. Recall that the parent node is the node from which the leaf nodes originated, which is currently the root node. Gain is calculated for different features with different thresholds chosen. The split resulting in the highest gain value is then determined to be optimal. In Figure 1, Step 3, the gain is calculated for the tumor size split using the threshold > 1. In Figure 1, Step 4, the same calculations are carried out, instead using a tumor size threshold of > 2. Note that the gain for tumor size > 2 is higher, which is based on the split using this threshold results in leaves with more homogenous groupings of residuals. Hence, the tumor size threshold > 2 results in the better split, and thus is kept going forward.

Building off the initial split using tumor size, the tree can be grown with additional leaf nodes to further separate the residuals. For simplicity, let us assume that the tree can be grown further using radiation (yes/no) as in Figure 1, Step 5. Note that since radiation is binary, no threshold is needed. Hence, gain values can

be compared directly across different categorical features. Notice now that the resulting tree depth is 2 (the longest path from the root node to a leaf node). In XGBoost, the user specifies the maximum tree depth such that the tree can grow no larger, which prevents overfitting and reduces overall model complexity.

We now have a "best" tree for classifying the current residuals (i.e., Res_{F0}). The reader is encouraged to check how a data point given by row in the data table of Figure 1, Step 1 leads to a specific leaf node. For example, the data in row one start at the root node, go to the right node (tumor size >2), then go to the left leaf node (radiation = no) which contains the residual for that observation.

Before computing $F_1(x)$, we need to convert the leaf nodes of $T_1(x)$ into a log-odds form as they were derived based on probabilities. To calculate the output value for each leaf (LO) into a log-odds form, we use a formula very similar to the SS given by

$$\text{LO} = \frac{\sum_{i=1}^{n} residual_i}{\sum_{i=1}^{n} \left[prior\, p_i \times \left(1 - prior\, p_i\right)\right] + \lambda}, \tag{5}$$

where the variables are defined similarly as in Equation (3). The calculation for each leaf node is shown in Figure 1, Step 5. Each observation, therefore, begins in the root nodes and then arrives at a leaf node with a specific log-odds value defined by Equation (5). For example, row 2 in Figure 1, Step 1 arrives at the blue node Figure 1, Step 5. In other words, $T_1(\text{row 2}) = 0.66$.

We are now ready to generate F_1 as we have T_1, F_0, and ϵ, which is the learning rate hyperparameter (referred to as "learn_rate" in the R code). Here we will assign it the value of 0.3. Conceptually the learning rate helps reduce overfitting, but a very low value will require a greater sequence of trees to be generated. From Equation (1), we have $F_1(x) = F_0(x) + \epsilon T_1(x)$. $F_1(x)$ is calculated as shown in Figure 1, Step 6 for the data from the first row, noting that $T_1(\text{row 1}) = 1.5$. To determine the probability of cancer recurrence based on F_1, that is, our target, we convert $F_1(x)$ to a probability via Equation (2). At this point, the residuals of $F_1(x)$ may be calculated, and the process can repeat with a new tree for $F_2(x)$. The reader is encouraged to calculate the remaining values for $F_1(x)$ and the residuals for $F_1(x)$.

A few comments are made regarding the example. First, the origins of Equations (3) and (5) are derived from minimizing the loss function for classification, and details can be found in the reference provided here: https://www.youtube.com/watch?v=ZVFeW798-2I&t=359s [37]. Second, it should be noted here that since XGBoost is ideal for large complex datasets, testing all possible features and splits can be computationally infeasible. XGBoost instead employs various algorithms to more efficiently select meaningful features and splits. Details around various optimizations that XGBoost utilizes are briefly described later in this section. Third, the SS score is different (and simpler) for regression. Lastly, XGBoost utilizes several regularization techniques that help address overfitting. As regularization is a key aspect of XGBoost, these techniques are discussed with simple examples in combination with other XGBoost hyperparameters in the next section.

6.3 | XGBoost Hyperparameters

As with most ML algorithms, XGBoost relies on several hyperparameters that help dictate the training process and overall model structure. These hyperparameters can either be set explicitly by the user or automatically tuned. While heuristics and prior knowledge can guide their selection, best practice is to perform hyperparameter tuning to optimize model performance. We will focus on regularization-related hyperparameters, as they play a critical role in controlling overfitting and shaping the structure of the final XGBoost model. Other commonly used hyperparameters are described in the code implementation section.

The parameter λ, also known as *reg_lambda*, helps prevent an XGBoost model from overfitting the data by adding a penalty to the model's loss function. It effectively achieves this by lowering the split score (SS) function when $\lambda > 0$, which can be seen in Equation (3). Importantly, if a node has relatively few observations in it, then a positive value of λ, say $\lambda = 1$, will lower the SS more significantly in comparison with a node with many observations. For example, suppose $\lambda = 1$ and the SS score was recalculated for the green node in Figure 1, Step 3. The new value of $\text{SS}_{\text{TS>1,RL}}$ would be 0.082, about 50% lower from the original value of 0.16 when $\lambda = 0$. In contrast, if $\lambda = 1$ and the SS was recalculated for the red node in Figure 1, Step 3, the new value of $\text{SS}_{\text{TS}\leq 1,\text{LL}}$ would be 0.13, which is about an 80% lower from the original value of 0.66. Intuitively, this is because the effect of lambda is inversely proportional to the number of residuals. Hence, λ penalizes nodes with very few observations by lowering the SS score more so compared to nodes with more observations.

The hyperparameter γ, also known as the tree complexity parameter (and referred to as "loss_reduction" in R code), controls the criteria for how new trees are built, and operates largely independent of λ. The value of γ is used to "prune" trees by preventing branches that do not result in sufficient gain. If the gain of a split is $<\gamma$, that is, $Gain < \gamma$, where Gain is defined in Equation (5), it implies that the split did not produce a substantial enough increase of information, that is, the loss function did not significantly reduce, and the branch node is removed (or rather, not built in the first place). Hence the tree becomes simpler. The value of γ therefore dictates how simple or complex trees will get. For example, if γ was set to 3 in Figure 1, Step 4, then the branch split by tumor size >2 would be removed since $\text{Gain}_{\text{tumorsize>2}} - \gamma = 2.21 - 3 < 0$ and therefore, only the root node would remain. However, if $= 2$, then this branch would remain.

Another hyperparameter in XGBoost is called the cover, which is also known as min_child_weight (referred to as "min_n" in the R code) specifies the minimum sum of instance weights needed in a child node. For classification, this often correlates with the minimum number of observations in a node but is not identical. If the value is high enough, it may prune a leaf with relatively few observations.

These regularization hyperparameters help prevent the model from essentially "fitting the noise," that is, they prevent the model from overfitting the training set. Regularization reduces the model's variance, making it less sensitive to the training data. Optimizing these hyperparameters is a key aspect of

building a robust XGBoost model, and more generally, understanding hyperparameter optimization is key to understanding model development in machine learning.

6.4 | XGBoost Optimization Features

The concepts described so far are not specific to XGBoost alone, but rather to the more general class of gradient-boosted tree methods of which XGBoost is a member. Where XGBoost is differentiated is in its optimization features that allow for efficient model building for large, complex datasets. These features are briefly described with general descriptions as they are not critical to the underlying algorithmic process.

1. Approximate greedy algorithm—A greedy algorithm is used to efficiently generate reasonable quantiles within a feature that can be used as thresholds to build trees. This is motivated by the fact that it is not feasible to test all possible thresholds within a feature for large datasets with high-dimensional feature sets.
2. Block structure for parallelization—For a feature with many observations, subsets of the feature/target can be distributed in parallel across computing cores or machines and used to generate an approximate histogram of the feature, for which quantiles are approximated.
3. Weighted quantile sketch—To simplify the search space for continuous features, histograms are used to set candidate split points. In the case of imbalanced classes or other data-dependent situations, weights can be incorporated into these distribution sketches.
4. Sparsity-aware split finding: This is an approach that prescribes how to build trees when there are missing observations for some features. This is especially useful for real-world data that tends to have missing information, which does not have to be addressed a priori.
5. Cache aware access: This is a hardware-level feature in which the cache memory within the CPU is used to do the first and second derivative calculations, which are computationally more intensive. Cache memory is very efficient in comparison to RAM memory or a hard drive.
6. Blocking for out-of-core computation—This feature compresses a portion of the dataset onto the hard drive when the dataset is too large for the cache and RAM. By compressing the data, it reduces the time for writing the data to the hard drive, which is traditionally very slow. XGBoost will then bring back compressed portions and uncompress them, which has a net efficiency without compression of any kind.

7 | Data Preprocessing

Data preprocessing describes the steps taken to initially evaluate and prepare a dataset(s) prior to modeling. Preprocessing involves any filtering, transforming, or encoding of data to ensure it is in a format compatible with the AI/ML algorithm(s) being used. Common steps in data preprocessing include reviewing the dataset, handling null or missing values, and defining rules for treating missing data, such as removing incomplete entries or applying imputation techniques. Depending on the dataset and algorithm, additional steps of encoding (converting them into a format readable by the algorithm) and scaling the data (normalizing data to a specific range) may also be performed. It should be noted that XGBoost, like other tree-based methods, is effectively scale-invariant since it relies on the relative ordering of features or categorical splits rather than the absolute values of features. Hence, scaling is not typically needed. Finally, the data are split into training, validation, and test subsets [38–40] to facilitate model building and evaluation.

8 | Code for Data Preprocessing and Model Preparation

The R code (provided in alternate font) is designed to preprocess the dataset for a binary classification task, specifically predicting whether a subject had a recurrence of cancer. Below is a description of the key parts of the code, integrated into the explanation.

1. Data Transformation and Feature Engineering

The code begins by transforming the raw dataset. The mutate function from `dplyr` is used to create new variables and transform existing ones:

- A logical variable `class_n` is created to indicate whether the subject had recurrence events (TRUE) or not (FALSE).
- The Class variable is converted into a factor with levels `c("recurrence-events", "no-recurrence-events")`, ensuring that "recurrence-events" is the first level because it is the event of interest, and creating the factors this way will cause other functions to have the desired behavior by default.
- The age, `tumor_size`, and `inv_nodes` variables are converted into ordered factors, preserving the natural order of these categories. For example, `tumor_size` is ordered from "0–4" to "50–54" and `inv_nodes` is ordered from "0–2" to "24–26"
- Logical variables are created for other binary attributes: `irradiat_n` for whether the subject received radiation, `node_caps_n` for whether lymph node capsular invasion was present, and `breast_left` for whether the tumor was in the left breast.

This transformation prepares the data for machine learning models by handling categorical and ordinal data effectively.

2. Defining the Preprocessing Formula

A preprocessing formula, using typical R formula syntax, is defined to specify the target variable and the predictors:

- The formula `class_n ~ age_f + menopause + tumor_size_f + inv_nodes_f + node_caps + deg_malig + breast + breast_quad + irradiat` captures the covariate model, with `class_n` as the target and the remaining variables as predictors.

This formula serves as the blueprint for subsequent data preprocessing.

3. Recipe for Preprocessing with tidymodels

A recipe is created using the recipes package to preprocess the data. The recipe performs the following steps:

- Ordinal factors like `age_f`, `tumor_size_f`, and `inv_nodes_f` are converted into numeric scores using the `step_ordinalscore()` function, ensuring that their ordinal nature is preserved in a numeric format.
- Dummy variables are created for categorical variables such as `breast`, `breast_quad`, `irradiat`, `menopause`, and `node_caps`. The `one_hot = TRUE` option ensures that all levels except the reference level are represented as separate dummy variables.

This recipe ensures the data are transformed appropriately for model fitting.

4. Fitting and Applying Preprocessing

The preprocessing steps are then fitted on the training data to avoid data leakage:

- The `prep(preprocessing_XGBoost, training-(init_split))` function fits the recipe to the training data, learning the transformations without leaking any information from the test set.

5. Preparing Data for Model Training and Testing

Finally, the transformed data is extracted and prepared for model training and testing:

- The `bake()` function is used to apply the learned transformations to both the training and testing datasets, resulting in the `x_train` and `x_test` matrices, which contain the preprocessed features.
- The `y_train` and `y_test` vectors are extracted using `pull(class_n)` to obtain the target variable.

This process ensures that both the training and testing data are ready for fitting the XGBoost model, with consistent preprocessing applied to both sets.

9 | Code Implementation For Classification

The data and R file code for fitting and evaluating the XGBoost model is provided in the supplemental materials (https://github.com/metrumresearchgroup/ascpt-ml-tutorial/), with key sections described below. Two approaches for implementing the model are used in this example. The first uses the XGBoost package interface directly, and the second uses the tidymodels package. More emphasis is placed on the second approach using tidymodels.

9.1 | Approach 1: Direct Use of XGBoost Package Interface

The XGBoost package interface (XGBoost function) does not directly use the R formula modeling syntax but instead uses the design matrix (x_train) and labels (y_train) from the data preprocessing steps directly as input. Some arguments (e.g., nrounds) are arguments directly provided to the function, while other arguments to control the model fit are specified as a list in the params argument. A simple example of fitting an XGBoost model for binary classification is:

```
xgboost::xgboost(data = x_train,
label = y_train,
nrounds=15,
params = list(objective =
"binary:logistic")).
```

- `x_train`: is the design matrix from the previous preprocessing section where the covariate model is specified.
- `label`: vector of the dependent variable (binary for binary classification, continuous for regression).
- `nrounds`: number of boosting iterations (the value of p in Equation (1).
- `params`: additional information for the model, including the loss function and specification of the type of model (e.g., classification or regression).

By default, during training, the loss of the training set is returned every iteration, which helps in monitoring the training process. Additional options allow for more information to be returned, including the loss on a test set. The returned XGBoost fit object contains information about the model and training process but is useful only with further analysis functions (e.g., when compared to the results of lm in R). The generic predict function is a straightforward way to return predictions for the training set or for new observations.

9.2 | Approach 2: Use of the Tidymodels Package

Alternatively, the tidymodels collection of packages uses a more standardized and generic syntax to specify the model. The workflows subpackage combines data preprocessing with model fitting to aid in common ML tasks. The model, using the recipes package, is specified similarly to the preprocessing method, using the R formula syntax:

```
recipes::recipe(
Class ~ age_f + menopause + tumor_size_f +
        inv_nodes_f + node_caps +
        deg_malig + breast + breast_quad +
        irradiat,
data = training(init_split)).
```

9.2.1 | Hyperparameter Tuning

As described previously, hyperparameters are user-chosen to help guide the training process. Unfortunately, there is little intuition on the optimal selection of training parameters. However, within XGBoost, a systematic search and crossvalidation scheme within the training data is typically performed to find approximately optimal hyperparameter tuning. In practice, the space of hyperparameters can be large, and several hyperparameters have similar

effects in the final model. Therefore, in practice, only a subset of hyperparameters will be tuned, with the others set to reasonable default values. Tuning procedures are varied and have a range of theoretical complexity, implementation difficulty, and computational requirements. Here, we will use a grid search to perform the tuning. A grid search can be easily parallelized, and the grid values to test are easily computed using a regular grid or a Latin-hypercube to efficiently cover the space of hyperparameters.

Within the tuning process, for each candidate set of hyperparameters, the training data is again split into training and validation sets via crossvalidation. Within each fold, the model is fit using the candidate hyperparameters and evaluated on the hold-out set using the same objective function as for the final model, for example, log-likelihood, RMSE, or ROC-AUC. The performance metric across the crossvalidation folds is aggregated (e.g., mean) and the best set of hyperparameters is chosen for a final fit of the model using all the training data.

Commonly tuned hyperparameters for XGBoost classification and regression applications are included below: [41]

- Step size (eta, learning rate): How much weight is given to each tree. Lower numbers reduce overfitting but increase the number of trees required and therefore increase the run time.
- Maximum Tree Depth: The maximum depth of any single tree that can occur (deep trees are not recommended).
- Minimum Number of Observations at Each Leaf (min_n, min_child_weight): Controls overfitting of each tree by adjusting the minimum number of observations in each leaf node. Larger values cause less complex trees and less overfitting.

The number of trees to create is an important hyperparameter, but is commonly fixed in advance. Other tuning parameters can also be used to control the complexity of the tree and have slightly different results than the tuning parameters described above. However, in our experience, and in most occasions, adjusting more than a couple of hyperparameters does not yield improved model fit. Within the tidymodels package, the `tune()` function is used in place of a specific hyperparameter argument to specify that the parameter will be tuned.

9.2.2 | Crossvalidation and Final Model Fitting

For this working example, two hyperparameters will be tuned: the maximum tree depth and the minimum number of observations in a leaf. The crossvalidation scheme as described previously is specified with the following lines of code:

```
boost_tree(
trees=15,
tree_depth = tune(),
min_n = tune(),
loss_reduction = 0,
sample_size = 1.0,
learn_rate = 0.3
) %>%
set_engine("XGBoost") %>%
set_mode("classification")
```

In this code, all the possible hyperparameters are specified, along with the algorithm used (XGBoost) and the problem type (classification). Note that all hyperparameters in the code above have been defined in the "Background Concepts" section except for "sample_size," which is another hyperparameter to help reduce overfitting. It is fixed at a default value of 1.0. Additionally, the grid of points to search over is described, here using a Latin-hypercube design:

```
grid_latin_hypercube(
tree_depth(range = c(2, 6)),
min_n(range = c(1, 15)),
size = 25
)
```

Twenty-five sets of parameters are used for the crossvalidation (referred to as "size" in the R code), and the possible ranges for the hyperparameters are also specified. While increasing the number of crossvalidation points may give the appearance of better results, often this is simply noise because many similar values of the hyperparameters generate very similar final predictive powers. Therefore, in practice, only reasonable and not perfect hyperparameters are necessary.

Once the final hyperparameters are selected with some criteria, for example, receiver operating characteristic-area under the curve (ROC-AUC), the model is refit on all the training data. Again, the tidymodels package simplifies this workflow, with two steps: first, to apply the selected hyperparameters in the workflow, and second, to fit the model:

```
final_wf <- finalize_workflow(
xgb_wf,
best_auc_params
)

final_fit <- final_wf %>%
last_fit(init_split)
```

This fit can be used to perform model evaluation, both ML-specific checks and standard pharmacometrics modeling checks. For example, the ROC curve for the test set can be plotted (Figure 2):

```
final_fit %>%
collect_predictions() %>%
roc_curve(Class,
     `.pred_recurrence-events`,
     event_level = "first") %>%
autoplot()
```

A confusion matrix created for a default classification threshold:

```
final_fit %>%
     collect_predictions() %>%
     conf_mat(Class, .pred_class)
```

Alternatively, because these classes were imbalanced, the precision-recall curve may be more informative (Figure 3):

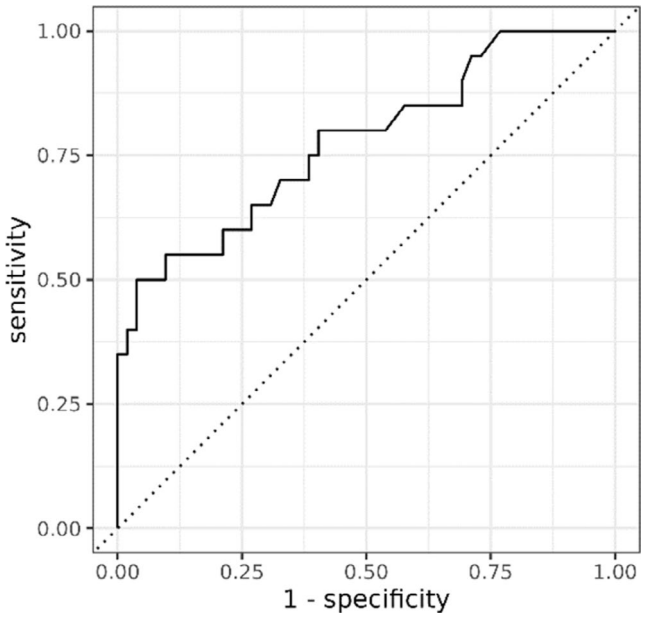


FIGURE 2 | ROC-AUC curve for classification example.

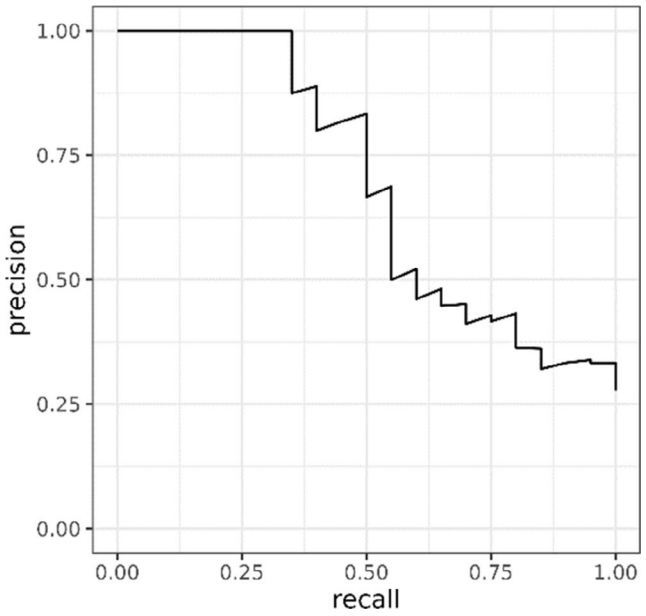


FIGURE 3 | Precision-recall curve for classification example.

```
final_fit %>%
collect_predictions() %>%
pr_curve(Class, `.pred_recurrence-events`,
event_level = "first") %>%
autoplot()
```

A calibration table analogous to a predicted versus observed plot for a regression problem can help evaluate the predictive power of the model on a test set. In Table 3, the test set is divided into quartiles (or any number of groups, e.g., deciles) of the model predicted probabilities, and the observed and predicted event rates are computed within each group. Two aspects of the model are evaluable. First, the predicted and observed should have similar rates for the accuracy of the model, that is, smaller residuals. Second, large differences between the lower quartiles and higher quartiles should be large, which demonstrates the predictive power of the model. This is because if differences across quartiles were small, it would indicate that the predicted probabilities are similar, and hence the model does not have a high degree of discrimination.

TABLE 3 | Calibration table for classification example.

Predicted quartile	*n*	Observed rate	Predicted rate
1	18	0.111	0.106
2	18	0.111	0.225
3	18	0.222	0.361
4	18	0.556	0.560

```
final_fit %>%
collect_predictions() %>%
mutate(predicted_quartile = ntile(`.pred_re-
currence-events`, 4)) %>%
group_by(predicted_quartile) %>%
summarize(n = n(),
       observed = mean(Class ==
       "recur rence-events"),
       predicted = mean
       (`.pred_recurrence-events`))
```

For new data, for example, in simulations, the predict() function is a simple way to make predictions. The functions available in the XGBoost and tidymodels packages perform common model evaluation tasks, but any specific model should be evaluated within the context it will be used. In general, quantitative evaluation of models in clinical pharmacology and pharmacometrics typically reports substantially worse performance than nonclinical pharmacology ML applications, such as image recognition. Hence, performance should be considered within the context of use and compared against evaluation metrics for alternative approaches to reliably compare performance.

10 | XGBoost For Regression

For the regression example, the task is to predict the number of alcoholic drinks consumed based on liver biomarker data using the open-source dataset [36]. All files, including the data and code, are located in the GitHub repository (https://github.com/metrumresearchgroup/ascpt-ml-tutorial/). The analysis workflow for regression is generally similar to that for the classification example presented thus far. Since the outcome is continuous (number of alcoholic drinks), this is a regression task. This dataset is already appropriately preprocessed for regression with XGBoost. For tidymodels and XGBoost in this context for regression, the three primary changes are: (1) the mode (tidymodels) or objective (XGBoost.fit) is changed from classification to regression; (2) the evaluation metric is changed to root mean square error (RMSE); (3) the classification-specific approaches (confusion matrix, ROC-AUC curve) are removed

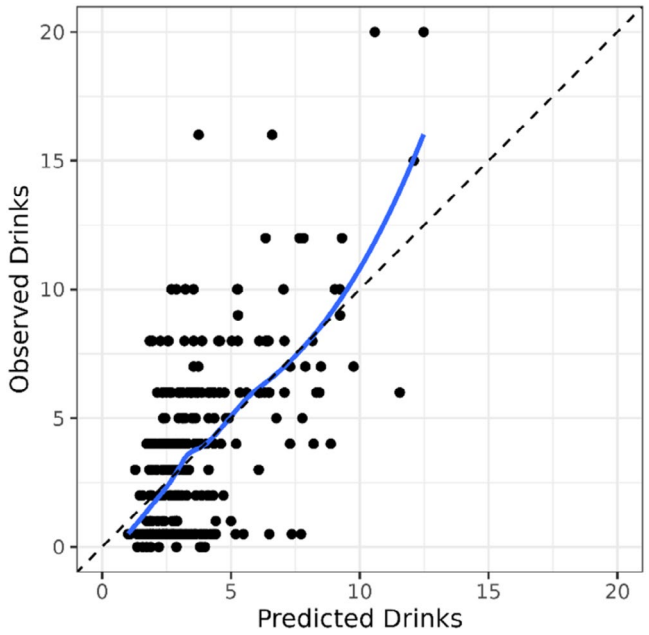


FIGURE 4 | Observed versus predicted results for the regression example.

and replaced with the familiar model evaluation approaches for regression.

Within the tidymodels package, updating XGBoost to a regression problem is set again with the `set_mode()` function:

```
set_mode("regression")
```

The crossvalidation output (metrics) is ones appropriate for regression:

```
metrics <- metric_set(rmse, rsq_trad, rsq)

best_rmse <- select_best(tuning_results, metric = "rmse")
```

Instead of class labels, the output will be a single real number for the model prediction given the input covariate vector. Depending on the hyperparameters, it is not uncommon for the output to visually look like a series of step functions that have a particular visual appearance or do not capture extreme values well. The model evaluation code is a simple predicted versus observed scatterplot, which is more akin to traditional pharmacometrics diagnostic plots compared to the classification example (Figure 4). Note that the GitHub repository contains final results for the regression code adaptation, and the reader is encouraged to refer to it after attempting the implementation.

Other regression variations are available within the XGBoost package. For example, there are loss functions for quantile regression, Poisson regression, or accelerated failure time survival models. Detailed explanations are beyond the scope of this tutorial, but the modeling syntax and ideas described here apply to those types of regression analyses as well.

11 | Conclusion and Discussion

The aim of this tutorial was to provide the reader with a comprehensive introduction to the concepts and implementation of the XGBoost algorithm motivated by clinical drug development applications. By providing a conceptual framework and a practical example, it is expected that the reader will be equipped to better identify practical applications for which XGBoost could be used. Machine learning algorithms such as XGBoost can be applied to support research efforts involving drug discovery or a variety of clinical development questions, spanning from radiomics to predicting clinical outcomes such as risk or response variables. XGBoost can also play a role in clinical pharmacology questions, such as aiding in PK profile prediction and optimizing dose-regimen selection through population PK or exposure–response analyses [42].

In drug development, XGBoost can be particularly useful in applications involving Real-World Data (RWD), which encompasses large-scale, high-dimensional datasets such as electronic health records (EHRs), claims data, or patient-reported outcomes [42]. These datasets are often messy and incomplete, requiring significant time and resources to preprocess them. XGBoost's ability to handle missing data, scale to large datasets, and deliver strong predictive performance makes it well-positioned to handle such data. As a best practice, consider fitting simpler models like linear/logistic regression or decision trees. These provide a baseline for performance and can highlight if navigating the complexity of XGBoost is necessary. Simple models may be more interpretable and sufficient for smaller datasets or applications with limited features.

A key consideration for the use of XGBoost is having a large, complex dataset. Unsurprisingly, there is no formal definition of large and complex, so the choice to use XGBoost rather than more traditional approaches is not always clear. Although, as mentioned, comparing outputs from different methods is highly encouraged, as it provides insight to scenarios that may benefit from more sophisticated approaches.

The textbook and tutorial examples (such as the ones presented here) are oversimplified and missing much of the nuance and typical challenges. Questions regarding sufficient model performance and how this is measured should be given substantially more consideration than what is presented here. Additionally, even as ML algorithms are promoted as having the capability to solve a wide variety of modeling questions, careful thought should be given to model evaluation, much like traditional pharmacometrics analysis. Limitations of the model stemming from the flexible structure and underlying data should also be assessed. Furthermore, more in-depth model validation should be performed for actual case studies. Ideally, such a process helps the modeler learn about what their data does and does not contain. Within this process, feature engineering, data cleaning, data wrangling, and other tasks in an ML pipeline are time-consuming and important even though this example used a generally clean, well-formatted appropriate dataset and question. Such a convenient setup is not the norm in practice.

In summary, there is no doubt that in the proper contexts, approaches like XGBoost can offer tremendous value and insight while being less resource-intensive. As the typical rates of clinical data generation continue to rise, the need for new analytical methodologies, especially those in the AI/ML space, will also continue to grow.

Conflicts of Interest

The authors declare no conflicts of interest.

References

1. J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics* 29, no. 5 (2001): 1189–1232.

2. T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," (2016), Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, https://doi.org/10.1145/2939672.2939785.

3. XGBoost Developers, "XGBoost Documentation," (2022), DMLC XGBoost, https://xgboost.readthedocs.io/en/stable/.

4. K. Takkavatakarn, W. Oh, E. Cheng, G. N. Nadkarni, and L. Chan, "Machine Learning Models to Predict End-Stage Kidney Disease in Chronic Kidney Disease Stage 4," *BMC Nephrology* 24 (2023): 376, https://doi.org/10.1186/s12882-023-03424-7.

5. X. Teng, K. Han, W. Jin, et al., "Development and Validation of an Early Diagnosis Model for Bone Metastasis in Non-Small Cell Lung Cancer Based on Serological Characteristics of the Bone Metastasis Mechanism," *EClinicalMedicine* 72 (2024): 102617, https://doi.org/10.1016/j.eclinm.2024.102617.

6. V. Bouget, J. Duquesne, S. Hassler, et al., "Machine Learning Predicts Response to TNF Inhibitors in Rheumatoid Arthritis: Results on the ESPOIR and ABIRISK Cohorts," *RMD Open* 8, no. 2 (2022): e002442, https://doi.org/10.1136/rmdopen-2022-002442.

7. F. D. Beacher, L. R. Mujica-Parodi, S. Gupta, and L. A. Ancora, "Machine Learning Predicts Outcomes of Phase III Clinical Trials for Prostate Cancer," *Algorithms* 14, no. 5 (2021): 147, https://doi.org/10.3390/a14050147.

8. M. Dal Bo, M. Polano, T. Ius, et al., "Machine Learning to Improve Interpretability of Clinical, Radiological and Panel-Based Genomic Data of Glioma Grade 4 Patients Undergoing Surgical Resection," *Journal of Translational Medicine* 21 (2023): 450, https://doi.org/10.1186/s12967-023-04308-y.

9. Q. Hu, H. Wang, and T. Xu, "Predicting Hepatotoxicity Associated With Low-Dose Methotrexate Using Machine Learning," *Journal of Clinical Medicine* 12, no. 4 (2023): 1599, https://doi.org/10.3390/jcm12041599.

10. F. Rui, Y. H. Yeo, L. Xu, et al., "Development of a Machine Learning-Based Model to Predict Hepatic Inflammation in Chronic Hepatitis B Patients With Concurrent Hepatic Steatosis: A Cohort Study," *EClinicalMedicine* 68 (2024): 102419, https://doi.org/10.1016/j.eclinm.2023.102419.

11. A. Eledkawy, T. Hamza, and S. El-Metwally, "Precision Cancer Classification Using Liquid Biopsy and Advanced Machine Learning Techniques," *Scientific Reports* 14, no. 1 (2024): 5841, https://doi.org/10.1038/s41598-024-56419-1.

12. J. Lieslehto, J. Tiihonen, M. Lähteenvuo, et al., "Development and Validation of a Machine Learning–Based Model of Mortality Risk in First-Episode Psychosis," *JAMA Network Open* 7, no. 3 (2024): e240640, https://doi.org/10.1001/jamanetworkopen.2024.0640.

13. R. Daniel, H. Jones, J. W. Gregory, et al., "Predicting Type 1 Diabetes in Children Using Electronic Health Records in Primary Care in the UK: Development and Validation of a Machine-Learning Algorithm," *Lancet Digital Health* 6, no. 6 (2024): e386–e395, https://doi.org/10.1016/S2589-7500(24)00050-5.

14. E. A. Poweleit, A. A. Vinks, and T. Mizuno, "Artificial Intelligence and Machine Learning Approaches to Facilitate Therapeutic Drug Management and Model-Informed Precision Dosing," *Therapeutic Drug Monitoring* 45, no. 2 (2023): 143–150, https://doi.org/10.1097/FTD.0000000000001078.

15. T. Van Gelder and A. A. Vinks, "Machine Learning as a Novel Method to Support Therapeutic Drug Management and Precision Dosing," *Clinical Pharmacology & Therapeutics* 110, no. 2 (2021): 273–276, https://doi.org/10.1002/cpt.2326.

16. A. K. Scott and M. L. Oyen, "Virtual Pregnancies: Predicting and Preventing Pregnancy Complications With Digital Twins," *Lancet Digital Health* 6, no. 7 (2024): e436–e437, https://doi.org/10.1016/S2589-7500(24)00086-4.

17. G. Liu, J. Lu, H. S. Lim, J. Y. Jin, and D. Lu, "Applying Interpretable Machine Learning Workflow to Evaluate Exposure–Response Relationships for Large-Molecule Oncology Drugs," *CPT: Pharmacometrics & Systems Pharmacology* 11, no. 12 (2022): 1614–1627, https://doi.org/10.1002/psp4.12871.

18. Q. Liu, A. Joshi, J. F. Standing, and P. H. van der Graaf, "Artificial Intelligence/Machine Learning: The New Frontier of Clinical Pharmacology and Precision Medicine," *Clinical Pharmacology & Therapeutics* 115, no. 4 (2024): 637–642, https://doi.org/10.1002/cpt.3198.

19. X. Zhu, J. Hu, T. Xiao, S. Huang, Y. Wen, and D. Shang, "An Interpretable Stacking Ensemble Learning Framework Based on Multi-Dimensional Data for Real-Time Prediction of Drug Concentration: The Example of Olanzapine," *Frontiers in Pharmacology* 13 (2022): 975855, https://doi.org/10.3389/fphar.2022.975855.

20. L. Ponthier, J. Autmizguine, B. Franck, et al., "Optimization of Ganciclovir and Valganciclovir Starting Dose in Children by Machine Learning," *Clinical Pharmacokinetics* 63 (2024): 539–550, https://doi.org/10.1007/s40262-024-01362-7.

21. Q. Huang, X. Lin, Y. Wang, et al., "Tacrolimus Pharmacokinetics in Pediatric Nephrotic Syndrome: A Combination of Population Pharmacokinetic Modelling and Machine Learning Approaches to Improve Individual Prediction," *Frontiers in Pharmacology* 13 (2022): 129, https://doi.org/10.3389/fphar.2022.942129.

22. B.-H. Tang, B.-F. Yao, W. Zhang, et al., "Optimal Use of β-Lactams in Neonates: Machine Learning-Based Clinical Decision Support System," *eBioMedicine* 105 (2024): 105221, https://doi.org/10.1016/j.ebiom.2024.105221.

23. R. K. Samala, K. Drukker, A. Shukla-Dave, et al., "AI and Machine Learning in Medical Imaging: Key Points From Development to Translation," *BJR Artificial Intelligence* 1, no. 1 (2024): ubae006, https://doi.org/10.1093/bjrai/ubae006.

24. L. Pinto-Coelho, "How Artificial Intelligence Is Shaping Medical Imaging Technology: A Survey of Innovations and Applications," *Bioengineering* 10, no. 12 (2023): 1435, https://doi.org/10.3390/bioengineering10121435.

25. X. Tang, "The Role of Artificial Intelligence in Medical Imaging Research," *BJR Open* 2, no. 1 (2019): 20190031, https://doi.org/10.1259/bjro.20190031.

26. O. Oren, B. J. Gersh, and D. L. Bhatt, "Artificial Intelligence in Medical Imaging: Switching From Radiographic Pathological Data to Clinically Meaningful Endpoints," *Lancet Digital Health* 2, no. 9 (2020): e486–e488, https://doi.org/10.1016/S2589-7500(20)30160-6.

27. G. Varoquaux and V. Cheplygina, "Machine Learning for Medical Imaging: Methodological Failures and Recommendations for the Future," *NPJ Digital Medicine* 5, no. 1 (2022): 1–8, https://doi.org/10.1038/s41746-022-00592-y.

28. S. Majumder, S. Katz, D. Kontos, and L. Roshkovan, "State of the Art: Radiomics and Radiomics-Related Artificial Intelligence on the Road to Clinical Translation," *BJR Open* 6, no. 1 (2024): tzad004, https://doi.org/10.1093/bjro/tzad004.

29. X. Ge, J. Gao, R. Niu, et al., "New Research Progress on 18F-FDG PET/CT Radiomics for EGFR Mutation Prediction in Lung Adenocarcinoma: A Review," *Frontiers in Oncology* 13 (2023): 392, https://doi.org/10.3389/fonc.2023.1242392.

30. H. Alami, P. Lehoux, J.-L. Denis, et al., "Organizational Readiness for Artificial Intelligence in Health Care: Insights for Decision-Making and Practice," *Journal of Health Organization and Management* 35, no. 1 (2021): 106–114, https://doi.org/10.1108/JHOM-03-2020-0074.

31. A. Samaras, A. Bekiaridou, A. S. Papazoglou, et al., "Artificial Intelligence-Based Mining of Electronic Health Record Data to Accelerate the Digital Transformation of the National Cardiovascular Ecosystem: Design Protocol of the Cardio Mining Study," *BMJ Open* 13, no. 4 (2023): e068698, https://doi.org/10.1136/bmjopen-2022-068698.

32. F. Bocquet, M. Campone, and M. Cuggia, "The Challenges of Implementing Comprehensive Clinical Data Warehouses in Hospitals," *International Journal of Environmental Research and Public Health* 19, no. 12 (2022): 7379, https://doi.org/10.3390/ijerph19127379.

33. A. Bazoge, E. Morin, B. Daille, and P.-A. Gourraud, "Applying Natural Language Processing to Textual Data From Clinical Data Warehouses: Systematic Review," *JMIR Medical Informatics* 11, no. 1 (2023): e42477, https://doi.org/10.2196/42477.

34. Y. Wieland-Jorna, D. van Kooten, R. A. Verheij, Y. de Man, A. L. Francke, and M. G. Oosterveld-Vlug, "Natural Language Processing Systems for Extracting Information From Electronic Health Records About Activities of Daily Living. A Systematic Review," *JAMIA Open* 7, no. 2 (2024): ooae044, https://doi.org/10.1093/jamiaopen/ooae044.

35. M. Zwitter and M. Soklic, "Breast Cancer UCI Machine Learning Repository," (1988), https://doi.org/10.24432/C51P4M.

36. "Liver Disorders," (2016), UCI Machine Learning Repository, https://doi.org/10.24432/C54G67.

37. StatQuest With Josh Starmer, "XGBoost Part 3 (of 4): Mathematical Details," (2020), https://www.youtube.com/watch?v=ZVFeW798-2I&t=359s.

38. S. Ghosh, "A Comprehensive Guide to Data Preprocessing," (2023), Neptune Blog, https://neptune.ai/blog/data-preprocessing-guide.

39. I. Novogroder, "Data Preprocessing in Machine Learning: Steps & Best Practices," (2024), https://lakefs.io/blog/data-preprocessing-in-machine-learning.

40. D. Kumar, "Introduction to Data Preprocessing in Machine Learning," (2018), https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d.

41. "XGBoost Parameters" (2024), https://XGBoost.readthedocs.io/en/stable/parameter.html.

42. U.S. Food & Drug Administration, "Using Artificial Intelligence & Machine Learning in the Development of Drug & Biological Products," Discussion Paper and Request for Feedback, (2024), https://www.fda.gov/media/167973/download?attachment.

Supporting Information

Additional supporting information can be found online in the Supporting Information section.