XGBoost in Survival Analysis in Customer Churn Prediction

2. Januar 2026

Zusammenfassung

Der Erfolg von XGBoost beruht auf einer Kombination aus Algorithmen und wirkungsvollen Systemarchitekturen. XGBoost (eXtreme Gradient Boosting) gilt als einer der leistungsfähigsten Algorithmen im maschinellen Lernen. Die Anwendung von XGBoost ist besonders vorteilhaft, wenn er auf Probleme der Überlebenszeitanalyse angewendet wird. Hier treffen zwei Welten aufeinander, die klassische statistische Überlebenszeitanalyse und der moderne Methode des Machine Learning des Gradient Boosting. Durch die Verbindung der zwei Welten wird es erst ermöglicht, dass zeitliche Abhängigkeiten und Zensierung direkt in das Framework von XGBoost integriert werden können. Im Kontext der Kundenabwanderung zeigt sich die besondere Relevanz dieser Kombination, denn Standard-Klassifikationsmodelle wie Nicht-parametrische Modelle (Kaplan-Meier) oder Semiparametermodelle (Cox-Proportional-Hazards) würden die Information ignorieren, wann ein Kunde abgewandert ist oder dass viele Kunden zum Zeitpunkt der Analyse noch aktiv sind. Survival-Modelle mit XGBoost haben den Vorteil, dass genau diese zeitliche Dimension berücksichtigt und dadurch präzisere Vorhersagen liefert.

1 XGBoost Orientierung

Die Kombination aus XGBoost, dass eines der leistungsstärksten ML-Modelle für tabellarische Daten ist und der Überlebensanalyse (Survival Analysis) ist sehr mächtig. Diese Verbindung ermöglicht es nicht-lineare Zusammenhänge zu modellieren, die die klassische Modelle wie das Cox-Proportional-Hazards-Modell (Cox PH) oft ignorieren.
Die Hauptgründe für XGBoost in Survival Analysis

- Unterstützung spezifische beobachtete Daten für Survival-Analyse
- Hohe Vorhersagegenauigkeit und Robustheit
- Effiziente Handhabung großer und hochdimensionaler Datensätze
- Interpretierbarkeit und Feature-Importance (Eingabevariable des Modells und die Maßeinheit der Beeinflussung)

- Flexibilität durch `xgboost`-Bibliothek und Interpretierbarkeit
- Anwendungen jenseits der Standard-Regression wie z.B. Klassifikation mit komplexen Algorithmen.

XGBoost macht die Survival-Analyse zugänglicher und verbessert das Ergebniss, macht genauer die Vorhersage und ist skalierbarer. Das ML-Toll XGBoost findet sowohl in der quantitativen Gesundheitsforschung als auch in der Customer-Churn-Prediction Anwendung, da es durch seine mathematische Robustheit zuverlässige und präzise Ergebnisse liefert.

1.1 Was ist Churn?

Churn (auch Kundenabwanderung oder Kundenfluktuation) beschreibt den Prozess, bei dem Kunden ihre Geschäftsbeziehung zu einem Unternehmen beenden und dessen Produkte oder Dienstleistungen nicht mehr nutzen. Quantitative kann der Churn als Prozentsatz (Churn Rate) innerhalb eines bestimmten Zeitraums gemessen werden. Die Kennzahl des Churn ist wesentlich für die Unternehmensgesundheit, insbesondere bei Abo-Modellen, da die Gewinnung von Neukunden in der Regel teurer ist als die Bindung bestehender Kunden. Unternehmen analysieren den Churn, um Gründe für die Abwanderung zu verstehen und Strategien zur Kundenbindung zu entwickeln.

1.2 Was ist XGBoost?

XGBoost fasst viele kleine, einfache Entscheidungsbäume zu einem großen Modell zusammen. So kann er sehr genaue Vorhersagen treffen. Er wird besonders häufig für Klassifikation und Regression. XGBoost beruht auf dem Prinzip des Gradient Boosting: Eine Vielzahl schwacher Modelle (meist Entscheidungsbäume) wird schrittweise trainiert, wobei jedes neue Modell die Fehler seiner Vorgänger korrigiert.

2 Einführung XGBoost

Der Erfolg von XGBoost gründet auf einer Vielzahl algorithmischer und systemischer Optimierungen, darunter Sparsity Awareness, Weighted Sketches, effiziente Cache-Nutzung, Datenkomprimierung und Sharding. Diese Arbeit mit dem Schwerpunkt Survival Analysis in Kombination mit XGBoost bietet einen prägnanten Überblick über die Methode XGBoost (eXtreme Gradient Boosting) und ist weit verbreiteter Algorithmus für maschinelles Lernen in der Datenanalyse.

2.1 Gradient Boosting: Die Kernidee

XGBoost basiert auf dem Prinzip des Ensemble-Lernens, genauer gesagt auf dem Boosting, das durch iterative Fehlerkorrektur die Modellleistung kontinuierlich verbessert. Als Beispiel für Survival-Analyse (Zeit-zu-Ereignis-Modellierung)

kann ein iterative Korrektur von Residuenfehlern angenommen werdne. Betrachten wir genauer das Regressionsproblem, dabei soll die Funktion $y = x^2$ geschätzt werden, basierend auf nur wenigen Datenpunkten. XGBoost baut ein Ensemble aus schwachen Lernmodellen Entscheidungsbäume). Es werden die folgenden systematische die Berechnunssequenzen beschrieben:

- Initiales Modell : $(F_0(x))$: Starte mit einem einfachen Schätzer, z.B. dem Mittelwert der Zielwerte y.
- Iterative Korrektur: Für jedes nachfolgenden Modell $m = 1, 2, \cdots, M$:
  1. Berechne die Residuen: $r_{i,m} = y_i \breve{} F_{m-1}(x_i)$
  2. Fitte ein neues Modell $h_m(x)$ (flachen Baum) auf diese Residuen.
  3. Aktualisiere das Gesamtmodell: $F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x)$, wobei ν die Lernrate ist (0.1 um Overfitting zu vermeiden)

Um zu verstehen, warum XGBoost „extrem“ gut ist, müssen wir die Zielfunktion(Objective Function) und deren Optimierung des mittleren Taylor-Polynoms näher betrachten. Zunächst wird die Zielfunktion definiert:

- Zielfunktion: das Resultat beim überwachten Lernen ist es, eine Zielfunktion zu minimieren, die aus zwei Teilen besteht, dem Verlust (Loss) und der Regularisierung:
  1. $\mathcal{L}(\phi) = \sum_{i=1}^{n} l\left(\hat{y}_i, y_i\right) + \sum_{k=1}^{K} \Omega\left(f_k\right)$
  2. $l(\hat{y}_i, y_i)$: Die Verlustfunktion (Mean-Squared Error), ein Maß, wie gut das Modell die Daten erklärt.
  3. $\Omega(f)$ wird Regularisierungsterm genannt, dabei kommt XGBoost ins Spiel. Der Buchstabe „X“ in XGBoost bestraft im Gegensatz dazu die Komplexität des Modells, um Overfitting zu verhindern.
- Es wird die additive Strategie angewandt, weil nicht alle Bäume gleichzeitig trainiert werden können, der Zeitpunkt t ist die Vorhersage für eine Instanz i und wird als Formel wie folgt dargestellt: $\hat{y}^(t) = \hat{y}_i^{(t-1)} + f_t(x_i)$

Gesucht wird derjenige Baum $f(t)$, dessen Hinzufügung zu den bestehenden Vorhersagen die stärkste Reduktion des Gesamtverlusts bewirkt. Wird als Formel wie folgt dargestellt:

$$\mathcal{L}^t = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{t-1} + f_t(x_i)\right) + \Omega(f_t)$$

2.1.1 Taylor-Entwicklung und warum die Nutzung im XGBoost ein großartiger Kunstgriff ist

Die Taylor-Entwicklung (eine Funktion die näherungsweise durch ein Polynom beschrieben wird) unterscheidet sich von XGBoost vom „einfachem“ Gradient Boosting. Um die Verlustfunktion allgemein (nicht nur für mittlere quadratische Fehler (mean-square-error)) optimiert werden können, nutzt XGBoost eine Taylor-Entwicklung 2. Ordnung zur Annäherung der Verlustfunktion.

Der Trick dabei: Der unterschied von „normalen“ Gradient Boosting, dabei wird jedes neue Bäumchen einfach der negative Gradient (1. Ableitung der Loss-Funktion) berechnet. Diese Vorgehensweise begrenzt insbesondere die folgenden Punkte:

- Benutzt lineare Information
- Loss-Funktion nicht exakt berücksichtigt
- Optimierung weniger stabil.

Anwendung in XGBoost durch die Verwendung von Gradient und Krümmung der Funktion:
Taylor-Entwicklung 2.Ordnung:

$$L(f+\delta) \approx L(f) + g\delta + \frac{1}{2}h\delta^2$$

Dabei ist g die 1. Ableitungswert (Gradient) und h: die 2. Ableitungswert (Hesse-Matrix (Matrix der zweiten partiellen Ableitung)). Diese Vorgehensweise ist sehr Mächtig, weil beim Training XGBoost jedesmal fragt:

„Wenn der Split hier gemacht wird, wie verbessert das meine Loss-Funktion?“

Approximation zweiter Ordnung: XGBoost berechnet jeden Split analytisch mit den Schwerpunkten:

- starke Loss verbessert
- wie optimale Vorhersage aussieht. Funktioniert fast mit jeder Verlustfunktion durch Taylor Entwicklung, also auch für Log-Loss, Poisson-Loss, $\cdots$

Zweite Ableitung ist entspricht einer Verbesserung der Stabilität:

2. Ableitung kann interpretiert werden „wie stark ist die Verlustfunktion gekrümmt“

XGBoost nutzt die zweite Ableitung und demnach die Krümmungsinformation:

- um zu große Update zu vermeiden
- Lernschritte zu „dämpfen“

- Overfitting zu reduzieren
- Konvergenz zu beschleunigen

Die Taylor Approximation kann der XGBoost Algorithmus optimale Werte direkt ausrechnen, im Unterschied zu dem klassischen Gradienten Boosting, dabei werden die Werte iterativ geschätzt, als Formel ausgedrückt.

$$w^\star = -\frac{\sum g_i}{\sum h_i + \lambda}$$

XGBoost nutzt eine Taylor-Entwicklung zweiter Ordnung zur Annäherung der Verlustfunktion, mit der Approximation und wird mathematisch wie folgt beschrieben:

$$\mathcal{L}^t \approx \sum_{i=1}^{n} \left[(y_i, \hat{y}_{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

Dabei sind Terme g_i und h_i entscheidend für den Algorithmus. g_i (ist der Gradient (1. Ableitung der Verlustfunktion), die Richtung des größten Anstiegs bzw. Abfalls des Fehlers beschreibt), h_i (Term der Hesse-Matrix (2. Ableitung des Verlustfunktion), die man sich grafisch als Krümmung vorstellen kann). Wir können folgendes festhalten. XGBoost „versteht“ also durch die 2. Ableitung (h_i) die „geografische“ Landschaft der Fehlerfunktion besser als Machine Learning Verfahren und es kann aggressiver lernen, wo die Krümmung geringer ist und vorsichtiger sein, wo sie stärker ist.

2.1.2 Warum Standard-Verlustfunktionen in der Survival-Analyse versagen

Survival-Analyse (auch Time-to-Event-Analyse) beschäftigt sich mit der Modellierung von der Zeit bis zum Eintritt eines Ereignis (z.B. Tod, Rezidiv, Maschinenausfall), unter der Berücksichtigung von Zensierung. Die Standard-Verlustfunktionen wie der Mean-Square-Error (MSE) oder Cross-Entropy sind für unzensierte und vollständige Daten konzipiert, jedoch scheitern aufgrund struktureller Eigenschaften der Daten.

Die Grundlegende Datenstruktur in Survival-Analyse
In der Survival-Analyse wird jede Einheit mit den Variablen i (z.B. Patient), T_i als die Zeit und einem Indikator δ_i beschrieben:

- $T_i = min(T_i^\star, C_i)$, wobei $T_i^\star$ wahres Ereigniszeit ist und C_i die Zensierungszeit (Lost-to-Follow-up)
- $\delta_i = 1$, wenn das Ereignis eintritt ($T_i = T_i^\star$) und $\delta_i = 0$, wenn zensiert ($T_i = C_i$), Ereignis tritt nach T_i ein.

Die Verteilung von T_i wird durch die Überlebensfunktion:

$$S(t) = P(T_i^\star > t)$$

oder der Hazard-Funktion:

$$h(t) = \lim_{\Delta t \to 0} \frac{P(t \leq T_i^\star < t + \Delta t (T_i^\star \geq t}{\Delta t}$$

oder der Dichtefunktion:

$$f(t) = -\frac{d}{dt} S(t)$$

beschrieben. Denn jede Funktion zur Modellierung von der Zufallsvariable T beschreibt eine nützliche Perspektive. Die Dichte $f(t)$ zeigt die absolute Eintrittswahrscheinlichkeit pro Zeiteinheit, die Überlebensfunktion $S(t)$ die Wahrscheinlichkeit, dass das Ereignis noch nicht eingetreten ist und die Hazard-Funktion $h(t)$ das bedingte Risiko in einem kleinen Zeitintervall gegebenen Überlebens bis t.

Definition und Ziel der Standard - Verlustfunktionen
Die Standard-Verlustfunktionen hat das Ziel die Minimierung eines Fehler zwischen beobachteten und vorhergesagten Werten abzubilden:

- MSE für Regression:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i(\theta))^2$$

- Cross-Entropy für Klassifikation:

$$L(\theta) = -\sum i = 1^n [y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i]$$

Diese Funktionen sind konvex und differenzierbar, das ist die Voraussetzung, dass

- alle y_i sind vollständig beobachtbar,
- die Fehler sind symmetrisch und unabhängig von der Zensierung.

Die Herleitung des Versagens
Annahme: Wir setzen voraus, dass die beobachtete Ereigniszeit durch ein gängiges Überlebensmodell (z.B. exponentiell, Weibull, Cox) beschrieben werden kann. Weiterhin nehmen wir nicht-informative Zensierung an, d.h. die Ursache und der Zeitpunkt der Zensierung liefern keine zusätzliche Information über die wahre Ereigniszeit $T_i^\star$, außer derjenigen, die bereits durch die beobachteten Kovariaten beschrieben werden Zensierte Fälle ($\delta_i = 0$) ist T_i (untere Schranke). Ein Standard-Verlustfunktion wie MSE würde $L = \sum (T_i - \hat{T}_i)^2$ minimieren, was für $\delta_i = 0$ impliziert, dass $\hat{T}_i \approx T_i$, obwohl die wahre $T_i^\star > T_i$. Führt zu einer systematischen Unterschätzung der Überlebenszeiten (Bias).

Der Erwartungswert des Schätzers $\hat{\theta} = \arg\min L(\theta)$ ist verzerrt, da die Likelihood der Daten nicht maximiert wird. Dabei ist der wahre Likelihood Schätzer für die Survival - Daten:

$$L(\theta) = \prod_{i=1}^{n} f(T_i;\theta)_i^{\delta} S(T_i;\theta)^{1-\delta_i}$$

Dabei wird der Log-Likelihood wie folgt beschrieben:

$$\ell(\theta) = \sum_{i=1}^{n} \left[\delta_i \log f\left(T_i;\theta\right) + \left(1-\delta_i\right) \log S\left(T_i;\theta\right)\right].$$

Im Standardverlust wird er Term $S(T_i)^{1-\delta_i}$ ausgelassen; dieser Term entspricht der Überlebenswahrscheinlichkeit für zensierte Beobachtungen, also $P(T > T_i)$.

Die Asymmetrie der Fehler:

Die Fehler- bzw. Residualverteilung in Überlebensdaten liegt meist nicht symmetrisch um null, wegen der vorhanden Zensierung und der zeitabhängigen Risiken, weiterhin liegt es an der Natur der Sache, dass Zeit-bis-Ereignis Variablen zu Schiefe führen. Das hat direkte Folgen für die Diagnose, Schätzung und Inferenz in den Überlebensmodellen.

Nicht-Parametrisierbarkeit:

Modelle wie das Cox-Proportional-Hazards-Modell verwenden semi-parametrischen Form:

$$h(t|x) = h_0(t)\exp(\beta^T x)$$

wobei die Baseline-Hazard $h_0(t)$ nicht parametrisch spezifiziert wird. Ein herkömmlicher Verlust (Standard-Loss) setzt dagegen eine vollständige Parametrisierung voraus. Im Gegensatz dazu beim Cox-Modell mit der partiellen Likelihood gearbeitet, die nur die Reihenfolge und die relativen Risiken der Ereignisse nutzt, um β zu schätzen:

$$L(\beta) = \prod_{i:\delta_i=1} \frac{\exp \beta^T x_i}{\sum_{i\in R(t_i)} \exp(\beta^T x_j)}$$

XGBoost

Beim Standard-XGBoost für Regression minimiert man die mittlere quadratische Abweichung (MSE) mittels Gradient Boosting. Für die MSE mit Verlustfunktion $L = \frac{1}{2}(\hat{y}_i - y_i)$ sind Gradient und Hesse-Matrix:

$$g_i = y_i - \hat{y}_i, \qquad\qquad h_i = 1$$

Für Survival-Probleme müssen die Loss-Funktionen jedoch an Zensierung und zeitabhängige Risiken angepasst werden. Üblicherweise werden als Alternativen die negative partielle Log-Likelihood des Cox-Modells oder ein zensurberücksichtigender Verlust für Accelerated Failure Time (AFT)-Modelle herangezogen.

Für Accelerated Failure Time (AFT) Modell ähnlich der Verlustfunktion mit Normalannahme lautet:

$$L = \sum_{\delta_i=1} (\log \hat{\sigma}_i) + \frac{(y_i - \hat{\mu}_i^2)}{2\hat{\sigma}_i} + \sum_{\delta_i=0} \log(1 - \Phi(\frac{y_i - \hat{mu}_i}{\hat{\sigma}_i}))$$

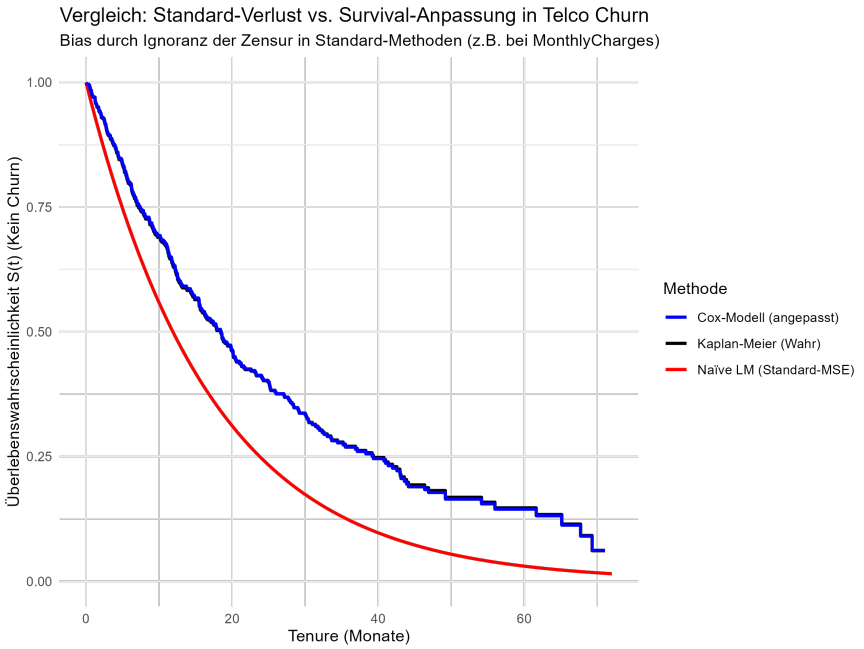

Abbildung 1: *Die Grafik veranschaulicht eine simulierte Churn-Survival-Analyse für eine Stichprobe von $n = 500$ Telco-Kunden mit einer maximalen Beobachtungszeit von 72 Monaten, realistischen Churn-Raten von etwa 40-50%, einer Zensierung von etwa 40% und einer Kovariate monthly-charges, deren höhere Werte den Churn beschleunigen. Die Überlebensfunktion S(t) beschreibt dabei die Wahrscheinlichkeit, dass ein Kunde bis zum Zeitpunkt t nicht abwandert.*

2.1.3 Struktur des Baumes und optimale Gewichte

Der Baum ist durch eine Struktur q gekennzeichnet, die die Daten in den Blättern organisiert, und durch die Gewichte w in den Blättern. Die Aggregation sämtlicher Datenpunkte erfolgt im selben Blatt, wobei die Datenpunkte der Gruppe j in der Menge I_j landen. Die Entfernung der Konstanten führt dazu, dass sich die Zielfunktion zu einer Summe über alle T-Blätter vereinfacht. Die Iterations-Verlustfunktion für einen Baum mit T Blättern lautet:

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$$

wobei I_j die Menge der Beobachtungen im Blatt j, g_i (Gradient) und h_i (Hesse-Matrix) der Verlustfunktion für Beobachtung i und w_j ist das Gewicht (Leaf-Score) des Blattes, λ die Blatt-Regularisierung und γ die Komplexitätsstrafe pro Blatt ist.
Die quadratische Form in w_j ist für jedes Blatt j der Summand einer quadrati-

schen Funktion w_j. Jedes Blatt lässt sich unabhängig ein geschlossener Ausdruck für das optimale Gewicht $w_j^\star$ herleiten. Dabei ist das optimale Blattgewicht wie folgt definiert:

$$w_j^\star = -\frac{\sum_{j \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Das Ergebnis wird durch das Nullsetzen der Ableitung der quadratischen Funktion nach w_j.

2.1.4 Strukturscore (Informationsgewinn)

Es gilt, die Bedingungen zu ermitteln, die der XGBoost-Algorithmus festlegt, um eine Entscheidung darüber treffen zu können, ob ein Baum geteilt oder gesplittet werden soll. Die vorliegende Untersuchung befasst sich mit der quantitativen Erfassung des Informationsgewinns, auch als Effektstärke bezeichnet, die durch einen Schnitt erzielt wird. der Strukturscore für den Informationsgewinn gilt folgende Formel:

$$\text{Gain} = \frac{1}{2}\left[\frac{\left(\sum_{i \in L} g_i\right)^2}{\sum_{i \in L} h_i + \lambda} + \frac{\left(\sum_{i \in R} g_i\right)^2}{\sum_{i \in R} h_i + \lambda} - \frac{\left(\sum_{i \in L \cup R} g_i\right)^2}{\sum_{i \in L \cup R} h_i + \lambda}\right] - \gamma.$$

Interpretation der Gain-Formel:

- $\sum g_L, \sum g_R$: Summe der Gradienten in den linken bzw. rechten Teilmengen und messen die Richtung und Stärke des Fehler in jedem Teilblatt.
- $\sum h_L, \sum h_R$: Summe der Hesse-Matrix (2. Ableitung) in den Teilmengen, quantifizieren Unsicherheit bzw. Krümmung der Verlustfunktion.
- λ: Blatt-Regularisierung, vermindert zu große Blattwerte durch Stabilisierung des Nenners.
- γ: Komplexitätsstrafe pro zusätzlichem Blatt, sie bestraft das Hinzufügen neuer Blätter

Erklärung der Terme:

1. Term: $\frac{1}{2}\frac{(\sum g_L)^2}{\sum h_L + \lambda}$, nutzen des linken neuen Blattes, wie stark sich die Zielfunktion verbessert, wenn die Beobachtungen in L ein eigenes Blatt bekommen.
2. Term: $\frac{1}{2}\frac{(\sum g_R)^2}{\sum h_R + \lambda}$, nutzen des rechten neuen Blattes.
3. Term: $-\frac{1}{2}\frac{(\sum g_{L \cup R})^2}{\sum h_{L \cup R} + \lambda}$, Es erfolgt der Abzug des Nutzens des ursprünglichen, nicht geteilten Blattes, das ist der Nettogewinn durch die Aufteilung

4. Term: $-\gamma$ Kosten für die Erzeugung zusätzlicher Blätter, dies reduziert den Gain um die Komplexitätsstrafe.

Ist der Gewinn (Effektstärke) kleiner als γ, dann wird der Baum nicht weiter wachsen (Pruning).

3 XGBoost bei Churn

XGBoost als Churn-Klassifikator(Ja/Nein) leidet typischerweise an (1) schlecht kalibrierten Wahrscheinlichkeitsvorhersagen und (2) Problemen bei stark unausgeglichenen Klassen.

1. Censoring (Zensierung): Nicht alle Beobachtungen erleben bzw. überstehen das Ereignis während der Studie, ohne Zensierung wären Standard Regression verzerrt, es wird zwischen den 3 unterschiedlichen Zensierungen unterschieden:

   - Rechtszensierung: Ereignis tritt nach dem Ende der Studie ein
   - Linkszensierung: Ereignis ist schon vor der Studie passiert.
   - Intervallzensierung: Ereignis liegt im Intervall.

   Ein Kunde, der heute der noch aktiv ist, wird als „Nicht-Churner“ (Klasse:0) gelabelt. Das ist so falsch, denn es ist festzustellen, dass der Kunde „noch nicht“ abgewandert ist. Es ist nur bekannt, dass seine Lebensdauer $T > t_{\text{heute}}$ ist.

2. Zeitwert: Es ist viel wertvoller zu wissen, wann jemand abgewandert ist, als nur ob er irgendwann geht. Um dies in der Survival Analysis zu nutzen, ändern wir nicht den Algorithmus, sondern die Objective Funktion (Zielfunktion). Anstatt „LogLoss“ (Klassifikation) oder „MSE“ (Regression), wird eine Funktion optimiert, die die Zensierung berücksichtigt.

Cox-Proportionale-Hazards (Partielle Likelihood)
Dieser Ansatz modelliert XGBoost die Hazard-Rate $h(t|x)$, also das unmittelbare Risiko, zum Zeitpunkt t abzuwandern. Üblicherweise wird $h(t|x) = h_0(t) \cdot \exp(f(x))$ angenommen, wobei die Regressionsparameter β über die partielle Likelihood geschätzt werden und die Basisrate $h_0(t)$ nicht parametrisch festgelegt ist.
Im Falle der Abwanderung eines Kunden zum Zeitpunkt t_i erfolgt seitens XGBoost eine Optimierung der Cox-Partial-Likelihood. Es besteht die Möglichkeit, dass genau dieser besondere Kunde abwandert und nicht einer der anderen aktiven Kunden, die sich im „Risk Set“ (R_i) befinden, maximiert werden. Es gilt die Formel:

$$\mathcal{L}(\theta) = -\sum_{i \in E} f\left(x_i\right) - \log \sum_{j \in R(t_i)} \exp((f\left(x_j\right)))$$

Die Verlustfunktion (Loss) im XGBoost minimiert dies und ist gerade der Negativanteil der Log-Likelihood:

$$\mathcal{L}(\theta) = -\sum_{i \in E} f(x_i) - \log \sum_{j \in R(t_i)} \exp f(x_j)$$

Dabei ist in der ersten Summe E die Menge der Ereignisse (die tatsächliche Churns), in der zweiten Summe ist $R(t_i)$ der Risk-Set (alle Kunden, die zum Zeitpunkt t_i noch aktiv waren).
Im vorliegenden Kontext wird der XGBoost-Effekt berechnet, wobei der Gradient und die Hesse-Matrix für die betreffende Funktion ermittelt werden. In der Folge erlernen die Bäume, den „Risk Score“ $\exp(f(x))$ für jeden Kunden so zu ordnen, dass Churner höhere Werte aufweisen als Nicht-Churner.

Accelerated Failure Time (AFT):
Der im XGBoost-Algorithmus bevorzugte Ansatz ist die Verwendung des Ziels `objective='survival:aft'`. Dabei wird der Logarithmus der Zeit bis zum Churn direkt modelliert. Das Modell lässt sich wie folgt darstellen:

$$\ln(T) = f(x) + \sigma \cdot W$$

Dabei bezeichnet $f(x)$ die von XGBoost gelernte Vorhersagefunktion, während W eine Zufallsvariable ist, die einer vorgegebenen Fehlerverteilung folgt (z.B. einer Normalverteilung oder einer Extreme-Value-Verteilung).
Die Optimierung erfolgt über eine Intervallregression, die zwischen beobachteten und zensierten Daten unterscheidet:

- **Churner (Ereignis):** Für Kunden, bei denen ein Churn beobachtet wurde, soll die Vorhersage $f(x)$ möglichst genau dem logarithmierten Churn-Zeitpunkt $\ln(T_{\text{churn}})$ entsprechen.
- **Aktive Kunden (zensiert):** Für Kunden, die zum Beobachtungszeitpunkt noch aktiv sind, soll die Vorhersage $f(x)$ größer als der logarithmierte aktuelle Zeitpunkt $\ln(T_{\text{heute}})$ sein.

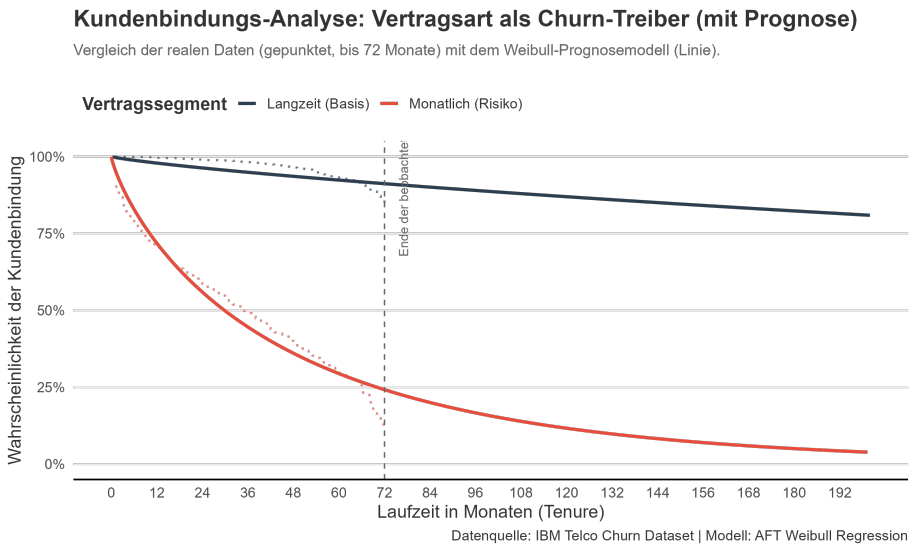

Abbildung 2: *Die Grafik veranschaulicht, dass die Kunden der Langzeit-Basis-Gruppe eine erwartete durchschnittliche Verweildauer von deutlich über 72 Monaten aufweisen (die Kurve fällt sanft weiter ab). Die X-Achse wurde entsprechend bis zu einem Wert von ca. 108 oder 120 Monaten verlängert, um die Überlegenheit des AFT-Modells bei der Prognose von Ereignissen außerhalb des beobachteten Zeitraums zu demonstrieren.*

4 Schlüssel-Features von XGBoost mit Schwerpunkt auf Survival Analysis

Auf Basis der Analyse des Papers „XGBoost: A Scalable Tree Boosting System“ werden in dieser Arbeit folgende Schlüsselfunktionen von XGBoost herausgearbeitet, eine hohe Skalierbarkeit und Effizienz bei großen Datensätzen, die überlegene Leistungsfähigkeit gegenüber Alternativen; sowie die Portabilität als Open-Source-Bibliothek mit Schnittstellen für Python, R und C++.

- Regularisierung: XGBoost integriert L1- (Lasso) und L2- (Ridge) Regularisierungsterme direkt in die Zielfunktion. Diese Terme pönalisieren eine zu hohe Modellkomplexität (z. B. die Anzahl der Blätter und deren Gewichte), um Überanpassung (Overfitting) effektiv zu verhindern und die Generalisierungsfähigkeit zu steigern. Dies stellt einen wesentlichen Vorteil gegenüber klassischen Gradient Boosting Machine (GBM)-Implementierungen dar.

- **Shrinkage** (Lernrate):** Nach jeder Boosting-Iteration wird der Beitrag der neu hinzugefügten Bäume durch eine Lernrate (η) skaliert. Diese „Schrumpfung“ verlangsamt den Lernprozess bewusst, wodurch das Mo-

dell robuster gegenüber Rauschen wird und das Risiko einer Überanpassung sinkt.

- **Spalten-Subsampling:** In Anlehnung an Random Forests ermöglicht XGBoost das Subsampling von Merkmalen (Spalten) beim Aufbau der einzelnen Bäume. Dies reduziert nicht nur die Varianz des Modells, sondern beschleunigt zudem die Berechnung erheblich.

- **Zeilen-Subsampling:** Analog zum stochastischen Gradient Boosting können Trainingsinstanzen (Zeilen) pro Baum subgesampelt werden. Dies erhöht die Robustheit gegenüber Ausreißern und verbessert die Trainingsgeschwindigkeit bei großen Datenmengen.

- **Umgang mit fehlenden Werten (Sparsity-aware Split Finding):** XGBoost verfügt über einen integrierten Mechanismus zur automatischen Behandlung fehlender Daten. Bei jedem Split lernt der Algorithmus die optimale Standardrichtung (linker oder rechter Zweig) für Instanzen mit fehlenden Werten, was eine aufwendige Vorverarbeitung (Imputation) oft überflüssig macht.

- **Parallele Verarbeitung:** Das System ist architektonisch auf die Nutzung paralleler Rechenressourcen ausgelegt. Insbesondere die Suche nach dem optimalen Splitpunkt wird parallelisiert, was XGBoost eine außergewöhnliche Skalierbarkeit verleiht.

- **Effizientes Pruning (Baumschnitt):** Zur Steuerung des Baumwachstums nutzt XGBoost die Parameter `max_depth` und `min_child_weight`. Im Gegensatz zu anderen Algorithmen wendet XGBoost eine „Rückwärts-Beschneidung“ an: Bäume wachsen zunächst bis zur maximalen Tiefe und werden anschließend von unten nach oben bereinigt, sofern ein Split nicht den erforderlichen Mindestgewinn (Gain) erzielt.

- **Integrierte Kreuzvalidierung:** Der Algorithmus erlaubt die Durchführung einer Kreuzvalidierung innerhalb jeder Boosting-Iteration. Dadurch kann die optimale Anzahl an Iterationen (Boosting Rounds) automatisiert und effizient ermittelt werden.

- Regularisierung: XGBoost enthält L1- (Lasso) und L2- (Ridge) Regularisierungsbegriffe in seiner Zielfunktion. Diese Begriffe bestrafen die Komplexität des Modells (z. B. die Anzahl der Blätter und deren Bewertungen), verhindern Überanpassung und verbessern die Generalisierung. Dies ist ein entscheidender Unterschied zu Standard-GBM-Implementierungen.

- Schrumpfung (Lernrate): Auf den Beitrag jedes Baums wird eine Lernrate (eta) angewendet. Dies verlangsamt den Boosting-Prozess und macht das Modell robuster gegenüber Überanpassung.

- Spalten-Unterabtastung: Ähnlich wie bei Random Forest ermöglicht XGBoost die Unterabtastung von Spalten (Merkmalen) beim Aufbau jedes

Baums. Dies reduziert die Überanpassung weiter und beschleunigt die Berechnungen.

- Zeilen-Subsampling: Das Subsampling von Trainingsinstanzen (Zeilen) kann ebenfalls pro Baum angewendet werden, ähnlich wie beim stochastischen Gradienten-Boosting, was die Robustheit und Trainingsgeschwindigkeit verbessert.

- Umgang mit fehlenden Werten: XGBoost verfügt über einen integrierten Mechanismus zum Umgang mit fehlenden Werten. Bei jeder Teilung lernt der Algorithmus das Beste. Richtung (linker oder rechter Zweig) für Instanzen mit fehlenden Werten.

- Parallele Verarbeitung: Der Algorithmus ist auf die Nutzung paralleler Rechenleistung ausgelegt, wodurch er für große Datensätze hochgradig skalierbar und effizient ist. Insbesondere die Baumkonstruktion kann parallelisiert werden.

- Baumschnitt: XGBoost verwendet einen `max_depth` - Parameter und einen `min_child_weight` - Parameter, um das Baumwachstum zu steuern und übermäßig komplexe Bäume zu vermeiden. Außerdem wird eine „Beschneidungstechnik“ implementiert, bei der Bäume bis zu ihrer maximalen Tiefe wachsen gelassen und dann rückwärts beschnitten werden, wobei Splits entfernt werden, die den Gewinnschwellenwert nicht erreichen.

- Integrierte Kreuzvalidierung: XGBoost ermöglicht die Durchführung einer Kreuzvalidierung bei jeder Boosting - Iteration, sodass Benutzer die optimale Anzahl von Boosting-Runden effizient ermitteln können.

Vorteile und Einschränkungen
Vorteile von XGBoost:

- Hohe Genauigkeit: Erzielt häufig modernste Ergebnisse für eine Vielzahl von tabellarischen Datensätzen.

- Geschwindigkeit und Skalierbarkeit: Dank hochoptimierter $C++$-Implementierung und paralleler Verarbeitungsfunktionen ist das Modell auch bei großen Datensätzen schnell.

- Robustheit: Aufgrund integrierter Regularisierungstechniken weniger anfällig für Überanpassung als herkömmliche GBMs.

- Flexibilität: Kann verschiedene Verlustfunktionen verarbeiten und funktioniert gut mit unterschiedlichen Datentypen.

Einschränkungen von XGBoost:

- Rechenaufwand: Kann trotz seiner Optimierungen bei extrem großen Datensätzen oder Modellen mit vielen Bäumen und großer Tiefe rechenintensiv sein.

- Interpretierbarkeit: Wie andere Ensemble-Baum-Methoden ist das endgültige Modell möglicherweise weniger interpretierbar als einfachere Modelle wie lineare Regression oder einzelne Entscheidungsbäume.
- Parameteroptimierung: Erfordert eine sorgfältige Optimierung zahlreicher Hyperparameter, um eine optimale Leistung zu erzielen.

5 Fazit

XGBoost stellt einen bedeutenden Fortschritt im Bereich Gradient Boosting dar und bietet ein optimiertes, skalierbares und hochwirksames Framework für überwachtes Lernen. Die Kombination aus robuster Regularisierung, effizienter Datenverarbeitung und parallelen Verarbeitungsfunktionen hat seine Position als unverzichtbares Werkzeug für Datenwissenschaftler und Analysten gefestigt. Angesichts der ständig wachsenden Datenmengen und Komplexität bleibt XGBoost ein entscheidender Algorithmus für die Gewinnung von Erkenntnissen und die Erstellung leistungsstarker Vorhersagemodelle in verschiedenen Bereichen. XGBoost-Überlebensmodelle übertreffen Cox bei realen EHR-Daten, lassen sich mit Histogramm auf Millionen von Patienten skalieren und nahtlos in $C++$-Quant-Pipelines integrieren, die im Finanzwesen verwendet werden.