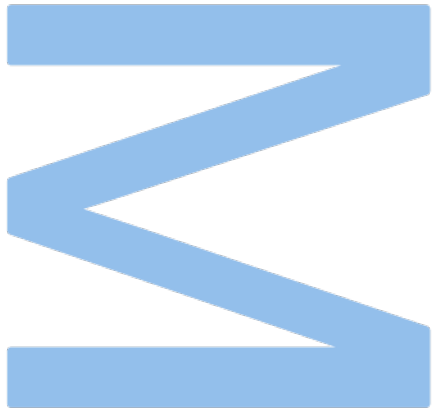
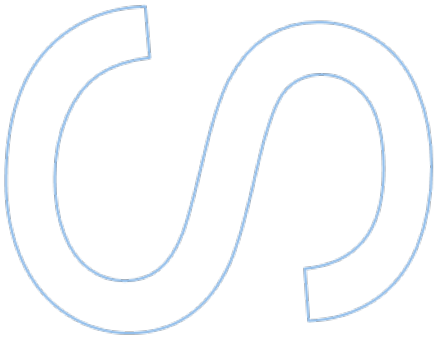
Distributed Cox Proportional-Hazards Model: A Step Towards Federated Survival Analysis

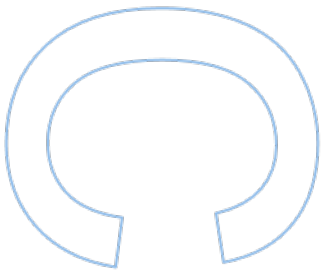


Xavier Branco Ribeiro
Mestrado em Estatística Computacional e Análise de Dados
Departamento de Matemática
2023

Orientador
Maria Goreti Carvalho Marreiros, Professora Coordenadora
Instituto Superior de Engenharia do Porto

Co-orientador
Margarida Maria Araújo Brito, Professora Associada
Faculdade de Ciências da Universidade do Porto






U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

UNIVERSIDADE DO PORTO

INTERNSHIP REPORT

Distributed Cox Proportional-Hazards Model: A Step Towards Federated Survival Analysis

Author:
Xavier Ribeiro

Supervisor:
Tiago Taveira-Gomes

A thesis submitted in fulfilment of the requirements
for the degree of MSc. Computational Statistics and Data Analysis

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

February 3, 2024

Acknowledgements

I would like to express my heartfelt gratitude to the many individuals who have been instrumental in the completion of this thesis. Their unwavering support, guidance, and encouragement have been invaluable throughout this challenging journey.

First and foremost, I am deeply thankful to my supervisor from MTG R&D Lab, and most importantly firend, Tiago Taveira-Gomes. Your wisdom, patience, and commitment to my academic and personal growth have been a constant source of inspiration. Tiago was the one who gave me all the means to perform to develop this work.

I extend my appreciation to the faculty members of the Faculdade de Ciências da Universidade do Porto e do Instituto Superior de Engenharia do Porto for their insightful feedback and for fostering an intellectually stimulating environment. Their contributions have been pivotal in refining the quality of this work.

I am also indebted to my friends and family for their endless support. Your belief in my abilities and your unwavering encouragement kept me going during the toughest moments. Without your understanding and love, this accomplishment would not have been possible.

I would like to acknowledge the financial support provided by GECAD, which allowed me to pursue my studies with greater focus.

Lastly, I would like to express my gratitude to all the participants who were involved in this research. Your cooperation and willingness to share your insights were indispensable to the success of this study.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

MSc. Computational Statistics and Data Analysis

Distributed Cox Proportional-Hazards Model: A step towards federated survival analysis

by Xavier Ribeiro

This Internship Report was written as part of an internship conducted between September 2022 and June 2023 at the company MTG Research & Development Lab. During this period, I had the opportunity to be a scholarship holder at the Institute of Superior Engineering of Porto (ISEP), where I joined the Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD).

I was assigned to the ITEA project *Secur-e-Health* where, in collaboration with the two institutions mentioned above, I was presented with a challenge.

The proposed challenge was to implement a distributed survival analysis method. The ultimate goal is to create a federated learning strategy that incorporates the Cox model.

In order to demonstrate that this is possible, this work focuses on considering subsets of a data set, fitting the distributed Cox model, and comparing the results with the analysis conducted on the entire data set.

We begin by introducing the necessary concepts of Survival Analysis, particularly the characteristics of the Cox model. We then delve into the Gradient Descent method, the strategy used to make distribution possible, and the topic of Federated Learning. Next, we explain the implementation of the method, as well as the results obtained in the simulation conducted with public data sets.

Keywords: Analysis, Survival, Distribution, Federated Learning, Gradient Descent, Bootstrapping, Cox Proportional-Hazards

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

Mestrado em Estatística Computacional e Análise de Dados

Modelo de Cox Distribuido: Um passo em direção da análise de sobrevida federada

por Xavier Ribeiro

Este Relatório de Estágio foi escrito no âmbito de um estágio curricular realizado entre Setembro de 2022 e Junho de 2023, na empresa *MTG Research & Development Lab*. No período referido tive a oportunidade de ser bolseiro no Instituto Superior de Engenharia do Porto (ISEP), no qual integrei a unidade *Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development* (GECAD).

Fui inserido no projeto ITEA *Secur-e-Health*, onde, em sintonia com as duas instituições acima referidas, me foi lançado um desafio.

O desafio proposto foi o de implementar um método de análise de sobrevida distribuida. O objetivo final é o de criar uma estrtégia de aprendizagem federada que incorpora o modelo de Cox.

De maneira a provar que isso é possível, este trabalho foca-se em considerar parcelas de um conjunto de dados, realizar o ajuste do modelo de Cox distribuido e comparar os resultados com a análise feita sobre o conjunto de dados como um todo.

Começamos por introduzir os conceitos necessários da Análise de Sobrevida, nomeadamente as características do modelo de Cox. De seguida mergulhamos sobre o método Gradient Descent, estratégia utilizada para tornar a distribuição possível, e também pelo tema da Aprendizagem Federada. Segue-se a explicação da Implementação do método, bem como os resultados obtidos na simulação efetuda com conjuntos de dados públicos.

Palavras-chave: Análise, Sobrevida, Distribuição, Aprendizagem Federada, Gradient Descent, Bootstrapping, Cox Proportional-Hazards

Contents

List of Figures

Chapter 1

Introduction

Survival analysis has long been a cornerstone in computational statistics, with the Cox Proportional-Hazards Model as a central figure. This research introduces a novel approach by integrating the principles of federated learning with the Cox Proportional-Hazards Model.

The essence of this work revolves around a distributed approach to survival analysis. Traditional methods have been rooted in centralized data systems, but this research advocates for a shift towards decentralization. By allowing survival analysis to be conducted directly at the data source, we eliminate the need for central data aggregation. This approach, inspired by federated learning, offers enhanced data privacy and security.

It's important to note that the primary goal of this distributed methodology isn't to enhance computational speed. In some instances, execution time might even be longer. However, the true value lies in the ability to perform survival analysis in a distributed manner. This perspective emphasizes the importance of decentralized data processing in our interconnected world, showcasing the potential of federated survival analysis that prioritizes data privacy without compromising analytical depth.

This challenge was first purposed by a company in Portugal of which the next section provides details about.

1.1 MTG Research & Development Lab

MTG Research & Development Lab is an R&D company that provides Real World Evidence, Implementation Science and Scientific Consulting services. The team is made up of expert biomedical scientists, doctors, academics and engineers that are passionate

about understanding diseases, injuries and improving the quality of healthcare systems. Their lab covers every aspect of the study lifecycle for Real World Evidence and Implementation Science, fostering truly collaborative and federated research with institutions across the globe.

1.2 GECAD

GECAD is a Research Unit having as mission the development of scientific research and innovation for the incorporation of Intelligence in Engineering and Computing Complex Systems. Intelligent Energy Systems is the International and National reference of Excellence of GECAD, namely in areas like Smart Grids, Electricity Markets, Distributed Resources, Demand Response, Energy Efficiency and Intelligent Buildings.

1.3 Secur-e-Health Project

Sensitive health data is often kept in silos in a way that cannot be efficiently leveraged for legitimate medical, research and data analysis purposes. The goal of the Secur-e-Health project is to integrate new approaches for digital ID technologies and privacy-preserving analysis techniques in a secure system infrastructure. The Secur-e-Health system allows medical institutions of all types to collaborate together and leverage data analyses and insights. This is expected to have a significant impact on the quality of the medical predictive models, the efficiency of data-driven treatments, the acceleration of new clinical research, and the improvement of healthcare in general.

By September 2022, MTG R&D Lab and Gecad Unit offered me the opportunity to become a member of the Secur-e-Health Project. The chalenge was: Take the Cox Proportional-Hazards Model and implement a distributed gradient descent method with the purpose of performing federated survival analysis. Always in sync with MTG's CEO, we developed the method that is being presented in this work.

Chapter 2

Survival Analysis

Considering that this study posits a distribution strategy to be applied to survival models, we shall embark upon the exposition of the intricate domain of survival analysis.Survival analysis, sometimes referred to as event time analysis or time-to-event analysis, revolves around predicting the time until a particular event of interest takes place, as Figure 2.1 suggests. Traditionally, this analytical approach finds its roots in the medical field, predominantly for studying the time to death or failure. However, the breadth of its application is wide, ranging from engineering, equipment failure for example, to social sciences, such as time to employment after graduation, and beyond.

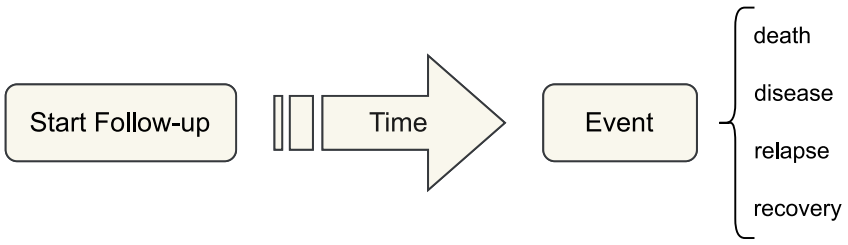


FIGURE 2.1: Survival Analysis

At the core of survival analysis is the concept of time and the complexities that accompany it. Unlike traditional regression models, where the dependent variable is straightforward and continuous, the dependent variable in survival analysis, time, often brings challenges. The most notable of these challenges is censoring.

Censoring occurs when we don't observe the event of interest for an individual during the study period. This can happen for a myriad of reasons: a patient might drop out of a study, a mechanical component might not fail by the end of an observational period,

or an unemployed individual might remain jobless by the study's conclusion. Censoring can take various forms, but the most common is right censoring, where the event hasn't occurred by the study's end or the participant leaves the study early.[1]

2.1 Censored Data

Most survival analyses have to consider censoring. In essence, censoring occurs when we have some information about individual survival time, but we don't know the survival time exactly. As a simple example of censoring, consider some kind of cancer patients followed until they go out of remission. If for a given patient, the study ends while the patient is still in remission (i.e., the event does not occur), then that patient's survival time is considered censored. We know that, for this person, the survival time is at least as long as the period that the person has been followed, but if the person goes out of remission after the study ends (as the **X** represents), we do not know the complete survival time. Figure 2.2 is a simplified representation of the scenario just described.

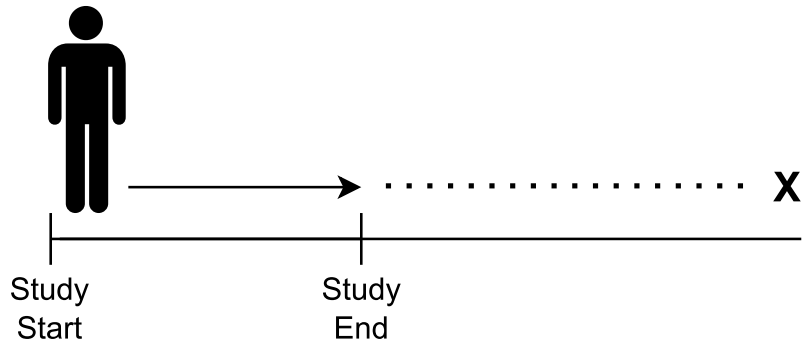


FIGURE 2.2: Censoring

Censoring may occur due to several reasons, some of them being: the person not experiencing the event before the study ends, the individual being lost to follow-up during the study period, a person withdraws from the study because of death (if death is not the event of interest) or some other reason (e.g., adverse drug reaction or other competing risk), to name a few. [2]

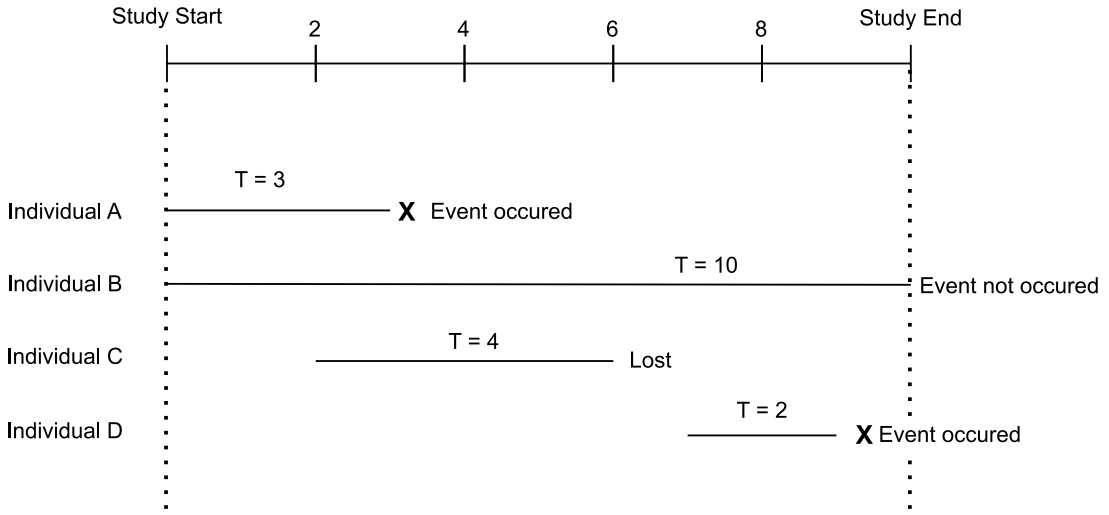


FIGURE 2.3: Censoring example

If we were to follow the information in Figure 2.3, a table of the survival time data regarding the four individuals in the graph would be as follows.

TABLE 2.1: Survival Time Data

Person	Survival time	Event
A	3	1
B	10	0
C	4	0
D	2	1

In our example, Figure 2.3, for all individuals who were censored, their exact survival times were undetermined towards the end of the monitoring period. This uncertainty arises when the study concludes, or if the participant is no longer reachable or decides to leave. This type of data is typically labeled as right-censored. Essentially, the full survival time span, which remains unknown to us, is truncated or cut short on the latter end of the observed survival duration. While left-censoring is also possible in data, the predominance in survival studies is of right-censored data.

Data that is left-censored emerges when an individual's genuine survival duration is equal to or shorter than the observed duration. Take, for instance, a study tracking participants until they test positive for HIV [2]. A failure might be noted at the point when a participant first shows a positive result for the virus. Yet, the precise moment they were first exposed to the virus might be unclear, leaving us uncertain about the exact

point the failure took place, Figure 2.4. As a result, this is considered left-censored data. The real survival period, culminating at the point of exposure, is actually briefer than the monitoring period that ends with a positive test result.

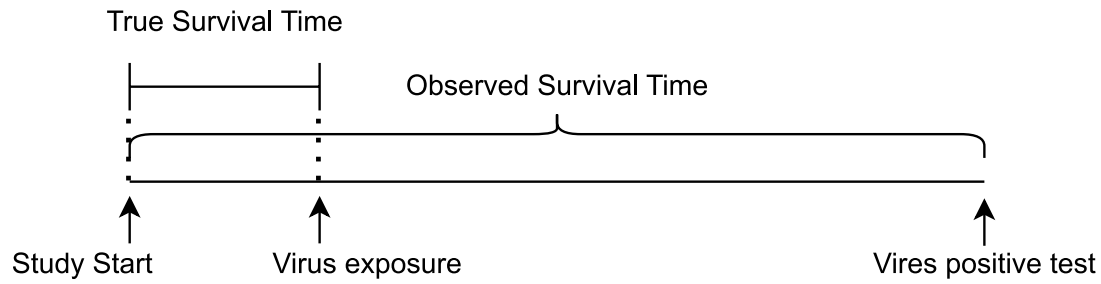


FIGURE 2.4: Left Censoring Example

Now that we have covered one of the most important characteristics of survival time data, Censoring, we can move on to the mathematical side of survival analysis.

2.2 Survivor and Hazard functions

Considering T to be the random variable for an individual's survival time, and t to denote any specific value for T, let's start by introducing the survivor function, $S(t)$. The survivor function gives the probability that a person survives longer than some specified time t, i.e., $S(t)$ gives the probability that the random variable T exceeds the specified time t.

The hazard function, denoted by $h(t)$, equals the limit, as Δt approaches zero, of a probability statement about survival, divided by Δt, where Δt denotes a small interval of time, Equation 2.1. In other words, the hazard function $h(t)$ gives the instantaneous potential per unit time for the event to occur, given that the individual has survived up to time t.[2]

$$h(t) = \lim_{\Delta t \to 0} \frac{P(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t} \tag{2.1}$$

In fact, if one knows the form of $S(t)$, one can derive the corresponding $h(t)$, and vice versa. For example, if the hazard function is constant, i.e., $h(t) = \lambda$, for some specific value λ, then it can be shown that, [2], the corresponding survival function is given by the following formula:

$$S(t) = e^{-\lambda t}$$

More generally, the relationship between $S(t)$ and $h(t)$ can be expressed equivalently in either of two calculus formulae shown here.

The first of these formulae describes how the survivor function $S(t)$ can be written in terms of an integral involving the hazard function. The formula says that:

$$S(t) = \exp\left(-\int_0^t h(u)\,du\right)$$

The second formula describes how the hazard function $h(t)$ can be written in terms of a derivative involving the survivor function. This formula says that:

$$h(t) = -\left(\frac{\frac{dS(t)}{dt}}{S(t)}\right)$$

The point here is simply that if you know either $S(t)$ or $h(t)$, you can get the other directly. We can now dive into the core of this work, the Cox Proportional-Hazards Model.

2.3 Cox Proportional-Hazards Model

As it has been said, the hazard function describes the instantaneous rate at which events occur over time. In other words, it represents the probability of an event happening in a very small time interval, given that the individual has survived up to that time.

The Cox Proportional Hazards regression model enrolls with the hazard function to evaluate the effect of a set of individual features, measured on a patient, in the time until a given event occurs. Let's consider $X = (X_1, ..., X_p)$ to be the set of features and T the variable that measures the event time. Another way to define the hazard of an event is

$$h(x,t) = \frac{p(t|x)}{S(t|x)} = \frac{\frac{d}{dt}P(T < t|X = x)}{P(T > t|X = x)}$$

We can think of the hazard function to be the probability density of having an event at time t, given that a patient, with a certain set of characteristics, has not had an event up

until that time. The Cox PH model assumes the following form of the hazard function

$$h(t, x; \beta) = h_0(t)e^{f_\beta(x)}$$

where $f_\beta(x)$ models the role played by the patient features x in the hazard and $h_0(t)$ is the baseline hazard function, which is the only element of $h(t, x; \beta)$ that depends on time. Although the weights function may take many forms, from now on we will considered it to have the following linear shape

$$f_\beta(x) = x^T\beta = \beta_1 x_1 + ... + \beta_p x_p$$

where β is of course the weights vector and p the number of covariates considered in the model.

An important property of the Cox model is that the baseline hazard, $h_0(t)$, is an unspecified function. It is this property that makes the Cox model a semi-parametric model. In contrast, a parametric model is one whose functional form is completely specified, except for the values of the unknown parameters. One of the reasons why the Cox model is so popular is that it is semi-parametric [2].

Another appealing property of the Cox model is that, even though the baseline hazard part of the model is unspecified, it is still possible to estimate β in the exponential part of the model. All we need are estimates of the β to assess the effect of explanatory variables of interest. The measure of effect, which is called a hazard ratio, is calculated without having to estimate the baseline hazard function. In general, a hazard ratio (HR) is defined as the hazard for one individual divided by the hazard for a different individual. The two individuals being compared can be distinguished by their values for the set of predictors [2].

2.3.1 Maximum Likelihood Estimation and Hazard Ratios

As with Logistic Regression [3], the Maximum Likelihood estimates of the Cox model parameters are derived by maximizing a likelihood function, usually denoted as L. The likelihood function is a mathematical expression which describes the joint probability of obtaining the data actually observed on the indiviudals in the study as a function of the unknown parameters β in the model being considered. L is sometimes written notationally as $L(\beta)$ where β denotes the collection of unknown parameters.

The formula for the Cox model likelihood function is actually called a "partial" likelihood function rather than a "complete" likelihood function. The term "partial" likelihood is used because the likelihood formula considers probabilities only for those subjects who fail, and does not explicitly consider probabilities for those subjects who are censored.

In particular, the partial likelihood can be written as the product of several likelihoods, one for each of, say, k failure times, Equation 2.2. Thus, at the jth failure time, L_j denotes the likelihood of failing at this time, given survival up to this time. Note that the set of individuals at risk at the jth failure time is called the "risk set", R_j, and this set will change, actually get smaller in size, as the failure time increases.

$$L = L_1 \times L_2 \times L_3 \times ... \times L_k = \prod_{j=1}^{k} L_j \tag{2.2}$$

This way, although the partial likelihood focuses on subjects who fail, survival time information prior to censorship is used for those subjects who are censored. That is, a person who is censored after the jth failure time is part of the risk set used to compute L_j, even though this person is censored later.

The construction of the likelihood function is presented in the implementation chapter, Chapter 6.

Once the likelihood function is formed for a given model, the next step for the computer is to maximize this function. This is generally done by maximizing the natural log of L, which is computationally easier.

The maximization process is carried out by taking partial derivatives of $log(L)$ with respect to each of the p parameters in the model, Equation 2.3, and then solving a system of equations.

$$\frac{\partial log(L)}{\partial \beta_i} \tag{2.3}$$

Once the Maximum Likelihood estimates are obtained, we are usually interested in carrying out statistical inferences about hazard ratios defined in terms of these estimates. We can write the hazard ratio (HR) as the estimate of $h(t, X^*)$ divided by the estimate of $h(t, X)$, Equation 2.4, where X^* denotes the set of predictors for one individual, and X denotes the set of predictors for the other individual.

$$HR = \frac{h(t, X^*)}{h(t, X)} \tag{2.4}$$

The definition of hazard ratio takes us to the most important assumption of the Cox model, the Proportional Hazards.

The Proportional-Hazards assumption requires that the hazard ratio is constant over time, or equivalently, that the hazard for one individual is proportional to the hazard for any other individual, where the proportionality constant is independent of time [2].

$$\frac{h(t, X^*)}{h(t, X)} = K \in \mathbb{R} \tag{2.5}$$

For the purpose of this work, we have now described all the Cox Proportional-Hazards Model characteristics necessary to further understand the method suggested in the next chapters. That being said let's move on to describe gradient descent in the next chapter.

Chapter 3

Gradient Descent

Gradient descent is a powerful tool in the field of machine learning and has been used in a wide range of models, from linear regression to neural networks. It is an essential technique for optimizing the performance of machine learning models and has become a cornerstone of modern data science.

In order to fully understand gradient descent we must first look at the mathematical side of the method.

Gradient descent approaches are made when the goal is to minimize a certain loss function. In Machine Learning algorithms this function usually represents an error measure between the predicted value for a variable of interest and it's real value.

Considering our goal is to find a minimum of a function, gradient descent based methods allow us to iteratively get closer to this minimum by following, at each step, the opposite growth direction of the function in question.

The key to visualize gradient descent is to understand what a function's derivative actually represents. Considering a certain function f and x_0 to be a point where f is continuous. The derivative $f^{'}(x_0)$ represents the slope of the tangent line to the graph of f, Figure 3.1.

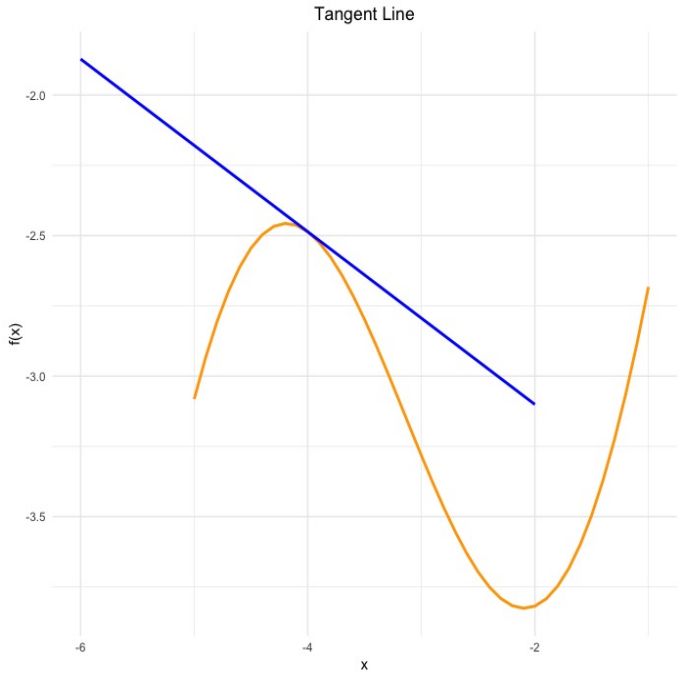


FIGURE 3.1: Derivative

The same logic is taken to define the gradient of a function ∇. In this case, $\nabla f(x_0)$ is the vector with same direction of the tangent line with slope $f^{'}(x_0)$ that also points in the growth direction of the function (Figure 3.2).

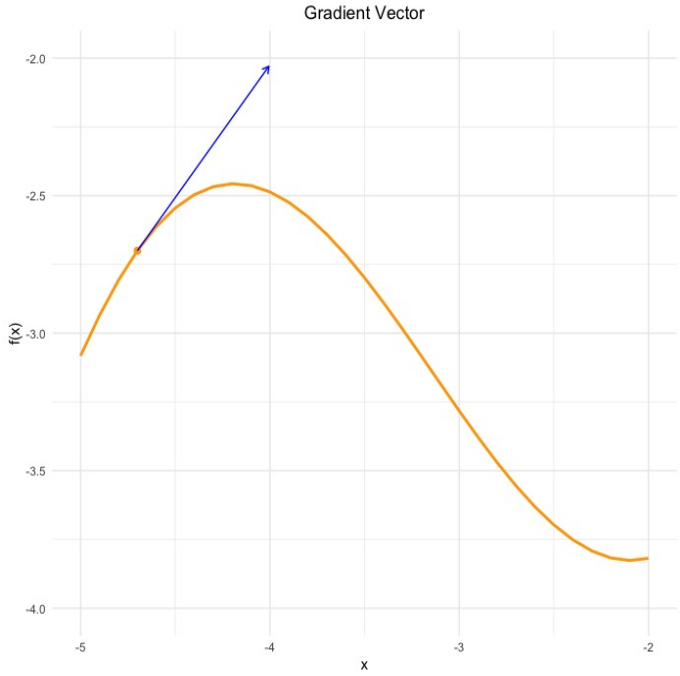


FIGURE 3.2: Gradient Vector

If we want to define the gradient in mathematical language, considering f to be an n-dimensional function and $x = (x_1, ..., x_n)$ to be the dimension's vector, the gradient of f at a point a, assuming $a \in D_f$, is defined as

$$\nabla f(a) = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n})$$

Now that the mathematical introduction of the method was described, we can move on to the application, starting by the requirements that need to be met in order to correctly apply gradient descent.

3.1 Usage requirements

Gradient descent based algorithms can not be used in every scenario. For this method to work and supply significant results it must be applied to a convex differentiable function. Considering D_f to be a function's domain, f is differentiable in all it's domain, if and only if Equation 3.1.

$$\forall x \in D_f, \exists a \in \mathbb{R} : f^{'}(x) = a \tag{3.1}$$

Although the mathematical definition of a convex function is a little bit more complicated, let's define this property in a geometric way. A function is called convex if the line segment between any two points intercepts the graph of the function only on those two points, as Figure 3.3 represents.

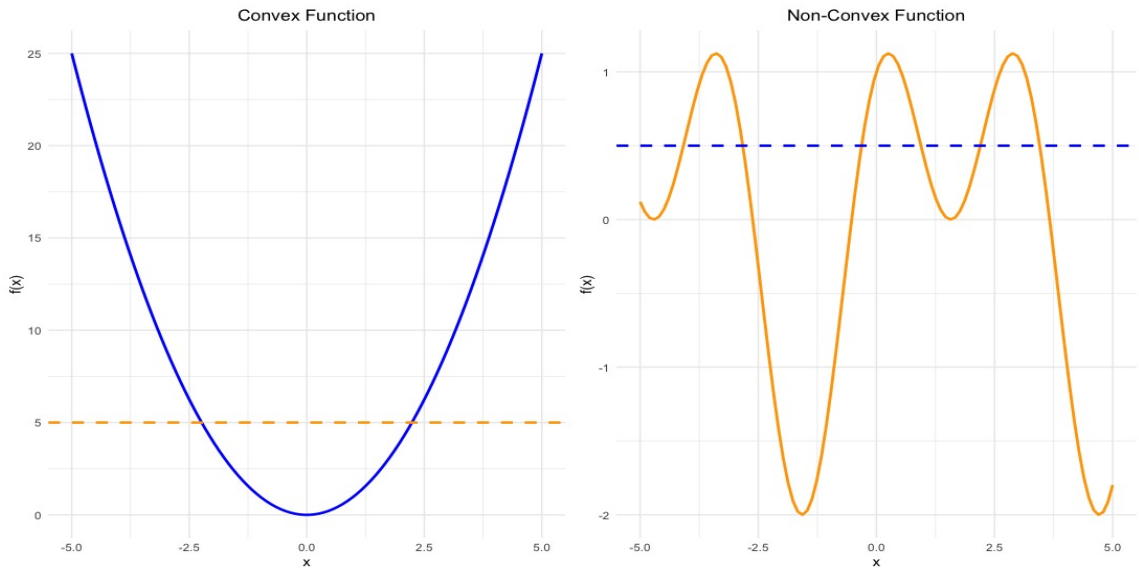


FIGURE 3.3: Convex and Non-Convex functions

More formally, a function $f(x)$ is said to be convex if, for any two points x_1 and x_2 in its domain and for any $\lambda \in]0, 1[$, the following inequality holds:

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

Once these two assumptions are checked, gradient descent should provide good results. However, there are some function characteristics that may help the method to work better. That being said, to apply gradient descent, the function that we want to optimize should satisfy the following requirements:

1. **Continuity**: The function should be continuous over its entire domain, meaning that there are no abrupt jumps or discontinuities in the function's values. A continuous function ensures smooth transitions between points and allows for meaningful derivative calculations.

2. **Differentiability**: The function should be differentiable in its domain, meaning that its derivative exists at every point within its domain. The derivative provides information about the rate of change of the function at each point.

3. **Convexity or Convex-Like Behavior**: While gradient descent can be applied to non-convex functions, it is most effective for convex or convex-like functions. A convex function has a single global minimum, where any two points within the function's domain are connected by a line segment that lies entirely above the graph of the function. Convex-like behavior refers to functions that exhibit locally convex regions, even if they are not globally convex. This allows gradient descent to converge towards promising solutions, although they may not be the global minimum.

4. **Boundedness**: While not strictly required, it is beneficial for the function to be bounded. A bounded function has finite values within a certain range. Boundedness helps prevent the gradient descent algorithm from encountering numerical instability or diverging towards infinity. However, unbounded functions can still be optimized with gradient descent, especially if they possess other desirable properties such as convexity or convergence towards finite limits.

5. **Finite Gradients**: The gradients of the function should be finite. This ensures that the updates made during the gradient descent process remain well-behaved and do not lead to divergence or instability. If the gradients become too large or tend

towards infinity, the optimization process may become unstable or fail to converge. Finite gradients indicate that the function's slope remains within reasonable bounds, facilitating a more reliable and effective gradient descent optimization.

While these requirements provide a general framework for applying gradient descent, it is important to note that there are variations and adaptations of gradient descent that relax some of these assumptions. For example, stochastic gradient descent and mini-batch gradient descent can handle non-convex functions and large datasets more efficiently, even if the function is not strictly differentiable or bounded. However, meeting these requirements generally increases the likelihood of achieving desirable optimization results with gradient descent [4].

3.2 Method

The basic idea behind gradient descent is to iteratively update the parameters of a model or the variables of a function in order to minimize a given loss or cost function. The steps involved in the gradient descent method are as follows:

1. **Initial Parameters**: Start with initial parameter values or variable values.

2. **Compute Gradient**: Compute the gradient of the function with respect to the parameters or variables at the current values. This involves calculating the partial derivatives of the function.

3. **Update Parameters**: Update the parameters or variables by taking a step in the direction opposite to the gradient. The step size is controlled by a parameter called the learning rate.

4. **Repeat**: Repeat steps 2 and 3 until convergence or a stopping criterion is met. Convergence is achieved when the values of the parameters or variables no longer change significantly or when the loss function reaches a sufficiently small value.

The learning rate determines the step size taken in each iteration. A high learning rate may result in overshooting the minimum, causing the algorithm to diverge. On the other hand, a very low learning rate can lead to slow convergence or getting stuck in a local minimum. Selecting an appropriate learning rate is crucial for the success of the gradient descent algorithm [5].

Consider λ to be a fixed value for a learning rate, that controls the step size we take at each iteration. Let's say we want to take a step from a certain point a_n. The result of the step a_{n+1} is defined by the following rule

$$a_{n+1} = a_n - \lambda \nabla f(a_n)$$

Adopting the rule above is nothing more than at each iteration move in the opposite direction in which the function grows (as Figure 3.4 suggests), since we are subtracting the vector that points to the growth. This way we hope to reach the minimum.

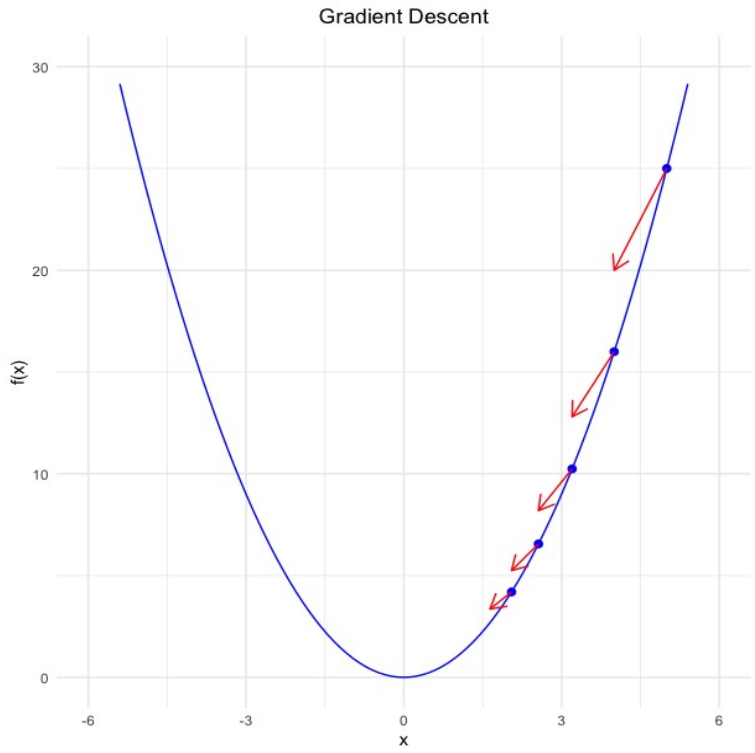


FIGURE 3.4: Gradient Descent

The value taken to be λ pretty much dictates how the algorithm will behave. Choosing small values for the learning rate results in very small step sizes, which can lead the algorithm to converge very slowly and become unreliable. In the other hand, setting λ to a high value results on bigger steps, increasing the odds of skipping the point in which the function actually is minimized, Figure 3.5. Therefore, the tuning of this parameter is the key to a well behaved gradient descent algorithm.

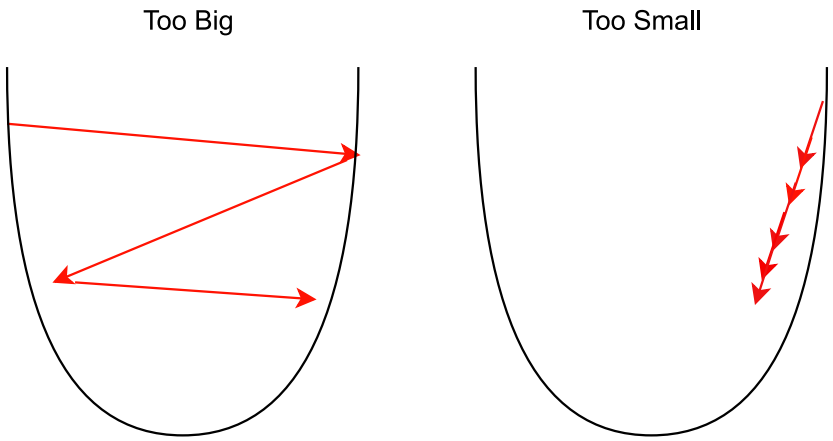


FIGURE 3.5: Effect of a too big or too small learning rate

Gradient Descent joined with a Bootstrapping approach are the methods used to perform the Cox analysis in this work. We will now move on to explain the Bootstrapping method.

Chapter 4

Bootstrapping

Introduced by Bradley Efron in 1979, bootstrapping has revolutionized the field of statistics by providing a simpler way to assess the accuracy of statistical estimates. Unlike traditional methods that often require the derivation of complex mathematical formulas, bootstrapping is conceptually simpler and widely applicable. By repeatedly resampling, with replacement, from the observed data and calculating the statistic of interest for each resample, bootstrapping allows for the empirical estimation of the sampling distribution of almost any statistic [6].

Resampling methods involve drawing repeated samples from an observed dataset and recalculating the statistic of interest for each sample. Resampling techniques can include methods such as permutation tests, cross-validation, jackknife, and of course, bootstrapping. These methods serve as a practical alternative to analytical approaches for statistical inference, especially in cases where analytical solutions are difficult to derive.

A fundamental assumption in classical statistics is that data are independent and identically distributed (i.i.d.). This means that each data point is independent of all others and follows the same probability distribution. The i.i.d. assumption is crucial for the validity of many statistical procedures, including the classical forms of bootstrapping [7].

4.1 How it works

The general idea of how Bootstrapping works involves the following steps:

1. **Draw a Sample**: Take a random sample of size n from an original dataset of size n, with replacement.

2. **Compute the Statistic**: Calculate the statistic of interest (for example a mean, median or standard deviation) from the new sample.

3. **Repeat**: Perform the previous steps many times, sometimes thousands or even tens of thousands of times.

By following these steps, a distribution of the statistic of interest is generated. This distribution, known as the Bootstrapping distribution, serves as an empirical representation of the sampling distribution for that statistic. From this distribution, one can make inferences about the population parameter. The principle behind bootstrapping is the assumption that the observed data provides a reasonable representation of the underlying population. If the original sample genuinely reflects the population, then resampling from it should approximate the variations we might see across different samples from the actual population.

This resampling technique can be used to construct confidence intervals, as we will see next.

4.2 Empirical Confidence Intervals

The first step in constructing confidence intervals using bootstrapping is to generate the Bootstrapping distribution of the statistic of interest. This involves repeatedly drawing samples (with replacement) from the observed data and calculating the statistic for each sample. This process results in an empirical distribution known as the Bootstrapping distribution, as mentioned above. The general steps involving a computation like this are the following:

1. **Draw a Random Sample with Replacement:** From your original data set of size n, draw a random sample of n observations with replacement.

2. **Compute the Statistic of Interest:** Calculate the statistic for the resampled data.

3. **Repeat the Process:** Perform steps 1 and 2 B times, where B is a large number.

4. **Sort the Bootstrapping Estimates:** After B repetitions, you will have B Bootstrapping estimates of the statistic.

5. **Identify Confidence Limits:** For a 95% confidence interval, you would use the 2.5th percentile and the 97.5th percentile of the sorted Bootstrapping estimates as the lower and upper confidence limits, respectively.

Let's move on to understand how we can apply this steps in order to calculate empirical confidence intervals in a Gradient Descent Cox Proportional-Hazards analysis.

4.3 Cox PH Model Confidence Intervals

In the Cox model, Wald confidence intervals are often employed to quantify the uncertainty associated with the hazard ratios. The Wald confidence interval for a given coefficient β is calculated as:

$$\text{Lower limit} = \hat{\beta} - z_{\alpha/2} \times \text{SE}(\hat{\beta})$$

$$\text{Upper limit} = \hat{\beta} + z_{\alpha/2} \times \text{SE}(\hat{\beta})$$

Where $\hat{\beta}$ is the maximum likelihood estimate (MLE) of the coefficient, $z_{\alpha/2}$ is the critical value from the standard normal distribution, and $\text{SE}(\hat{\beta})$ is the standard error of $\hat{\beta}$.

The standard error $\text{SE}(\hat{\beta})$ is derived from the Fisher Information matrix. This matrix is computed based on the partial likelihood function in the Cox model. Specifically, the standard error is the square root of the diagonal elements of the inverse of the observed information matrix I^{-1}:

$$\text{SE}(\hat{\beta}_i) = \sqrt{[I^{-1}]_{ii}}$$

The observed information matrix itself is the negative of the second derivative (Hessian matrix) of the log partial likelihood function, evaluated at the MLEs [8].

4.4 Distributed Cox PH Model Confidence Intervals

Consider $\hat{\beta}$ to be the original estimate of the coefficients vector, obtained from the original dataset. Let b represent one of the resamples from the dataset and $\hat{\beta}^b$ the corresponding coefficients. It has been showed that $\sqrt{n}(\hat{\beta} - \beta)$ and $\sqrt{n}(\hat{\beta}^b - \hat{\beta})$ converge in distribution to the same limit distribution [9].

Given this information, the steps to calculate the confidence intervals (as proposed in [10]) are:

1. **Calculate** $\hat{\beta}$: Obtain the estimate from the original dataset, via gradient descent.

2. **Calculate** $\hat{\beta}^b$: Get the estimate for all of the B resamples, using $\hat{\beta}$ as the initial set of weights.

3. **Obtain the quantiles**: Calculate $q_{\alpha/2}$ and $q_{1-\alpha/2}$ as the $\alpha/2$ and $1-\alpha/2$ quantiles of $\hat{\beta}^b - \hat{\beta}$

4. **Results**: Report the $(1-\alpha)100\%$ CI as $(\hat{\beta} - q_{1-\alpha/2}, \hat{\beta} - q_{\alpha/2})$.

The computation of Bootstrapping confidence intervals is the end of the description of the method being proposed in this work.

As the title suggests, this work aims to be a *step towards federated survival analysis*. That being said, we will now dive into the overview of Federated Learning, in the next chapter.

Chapter 5

Federated Learning

Federated learning is an innovative approach to machine learning that revolutionizes the conventional method of centralized data storage and processing. It provides an effective solution to the long-standing problem of how to gain insights from vast amounts of data while respecting privacy and maintaining security.

In traditional machine learning, data from various sources are centralized in a single location for the training of machine learning models. This process often raises a host of issues, particularly regarding data privacy, security, and the massive computational resources required to process these large datasets.

Federated learning, on the other hand, shifts the paradigm by decentralizing the learning process. It allows for data to be kept on the source (on local devices) and for machine learning models to be trained directly on these devices. The insights gained from each local model are shared with a central server, which aggregates the updates to improve a global model. Crucially, the raw data never leaves the local device, thereby preserving data privacy and security [11]. Figure 5.1 is a simplified representation of a federated learning scenario.

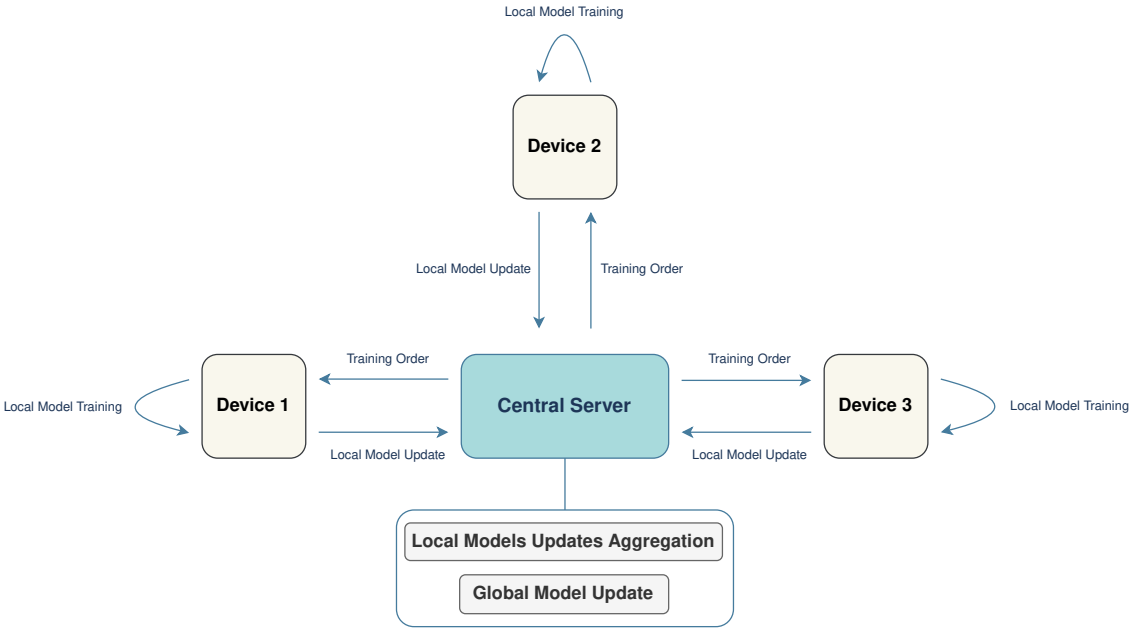


FIGURE 5.1: Federated Learning Template

The origin of federated learning can be traced back to the early 2010s when the rapid proliferation of smart devices resulted in an unprecedented increase in data generation. These devices, often equipped with powerful processors and abundant storage, offered a novel opportunity to perform machine learning tasks locally. However, the idea only started gaining traction when Google introduced the term *Federated Learning* in 2016, proposing it as a solution to train machine learning models on users' devices without compromising their privacy [12].

Federated learning not only represents a significant leap in the realm of distributed machine learning, but it is also a testament to the relentless pursuit of a balance between leveraging data for technological advancement and upholding the fundamental rights of data privacy and security.

5.1 Fundamental Principals of Federated Learning

Although there are many ways to implement a federated analysis, this type of machine learning approach always relies on the same principles to guide the way data is handled, computations are performed, and how models are updated in a decentralized manner.

1. **Data Privacy and Security:** In federated learning, data privacy and security are paramount. The raw data remains on local devices and is not directly shared or transmitted, which allows for a high degree of data privacy.

2. **Local Computation:** Federated learning shifts the computation to the edge, i.e., on local devices. This aspect significantly reduces the need to transmit data over the network, thereby saving bandwidth and further enhancing privacy.

3. **Global Aggregation:** While the training occurs locally on devices, the model updates are aggregated on a central server to create a global model.

4. **Decentralized Model Updating:** Decentralized model updating is a distinctive feature of federated learning, which involves local training of models and sharing of the model updates rather than raw data.

5.2 Process

In contrast to traditional machine learning, where data is collected and centralized in one location, federated learning allows data to reside on local devices where it is generated. This approach not only respects user privacy but also mitigates the potential security risks involved with data transfer and central storage. Each local device, or node, in a federated learning system retains its raw data. This data can be diverse and reflective of the device's unique environment or user.

Model Training on Local Nodes

The training of local models is a crucial component of federated learning. Using the collected data, each node independently trains its local model. The beauty of federated learning lies in this decentralized learning process, where valuable insights are gleaned directly from source data, minimizing the risk of data leakage.

The local training process involves optimizing the model on local data using iterative methods like gradient descent. The specific details of the local training process may vary depending on the task and the model at hand. Local training can be has simple has calculating a gradient given a set of weights, or performing matrix operations. The key factor is that this operations are performed without the central server party having to be in direct contact with the data.

For the local training to happen, the central party must develop the algorithm to implement in the other parties local device.

Aggregation of Model Updates and Global Update

Once the local models are trained, they send model updates - which can be gradients, weights, or other parameters - to a central server. This aggregation process is a key step, and its performance can significantly influence the learning outcome.

The method of aggregation can differ based on the specifics of the federated learning system. Some common strategies include simple averaging, weighted averaging based on the number of samples on each node, and more sophisticated methods designed to handle heterogeneity in the data and models.

Upon receiving the model updates, the central server performs a global model update. This update represents an integration of insights learned across all participating nodes in the federation, thereby creating a model that benefits from the collective knowledge of all nodes.

The next chapter dives into the details of the implementation of Distributed Cox Proportional-Hazards Model. As we will see, the method performs the analysis across multiple subsets of data, aggregating information and updating a central global model. This strategy is key for performing federated learning analysis, as it only requires the computation of gradients in the distinct parties involved.

Chapter 6

Implementation

This chapter provides a comprehensive examination of the process and methodology involved in the implementation of distributed survival analysis. We initiate the discussion by introducing an example of a survival dataset, which will serve as a practical illustration for subsequent computations, specifically, for computing the gradient of the loss function.

Following this, we move to an exploration of the hazard function, examining its features and the manipulation and calculus necessary to calculate the gradient.

Finally, we delve into an in-depth walk-through of the entire methodology, elucidating each stage of the process. The objective is to offer a global understanding of the implementation steps involved in distributed survival analysis, from preliminary data selection and preparation to final computational procedures.

6.1 NCCTG Lung Cancer Data

Consider a survival dataset available in the *survival R* package, focusing on lung cancer survival analysis. Lung cancer is one of the leading causes of cancer-related deaths worldwide. Analyzing survival data in this context can provide valuable insights into prognosis, treatment efficacy, and patient outcomes.

The lung cancer survival dataset, named *lung*, contains information collected from a clinical study conducted at a major medical center. The dataset comprises data from 228 patients diagnosed with lung cancer. The primary endpoint of interest in this study is survival time, measured in days, defined as the time from diagnosis to death or censoring.

The variables that take part in the dataset in question, as-well as their description, are:

TABLE 6.1: Head of the *survival* package *lung* dataset

inst	time	status	age	sex	ph.ecog	ph.karno	pat.karno	meal.cal	wt.loss
3	306	2	74	1	1	90	100	1175	NA
3	455	2	68	1	0	90	90	1225	15
3	1010	1	56	1	0	90	90	NA	15
5	210	2	57	1	1	90	60	1150	11
1	883	2	60	1	0	100	90	NA	0
12	1022	1	74	1	1	50	80	513	0

inst: Institution code

time: Survival time in days

status: Censoring status (1 = censored, 2 = dead)

age: Age in years

sex: Gender (1 = Male, 2 = Female)

ph.ecog: ECOG performance score as rated by the physician. Scale: 0 = asymptomatic, 1 = symptomatic but completely ambulatory, 2 = in bed less than 50% of the day, 3 = in bed more than 50% of the day but not bedbound, 4 = bedbound

ph.karno: Karnofsky performance score rated by physician (0 = bad, 100 = good)

pat.karno: Karnofsky performance score as rated by the patient

meal.cal: Calories consumed at meals

wt.loss: Weight loss in the last six months (pounds)

Variables such as age, sex, some kind of treatment categorical variable, tumor size representative variable and others, are very common in survival datasets.

Now that we understand the type of data we are working with, let's take a deep look into the hazard function.

6.2 Hazard Function Manipulation

Let's remember the equation for the hazard function, where $f_\beta(x)$ models the role played by the patient features x in the hazard and $h_0(t)$ is the baseline hazard function, which is the only element of $h(t, x; \beta)$ that depends on time.

$$h(t, x; \beta) = h_0(t)e^{f_\beta(x)} \tag{6.1}$$

Although the weights function may take many forms, from now on we will consider it to have the following linear shape

$$f_\beta(x) = x^T\beta = \beta_1 x_1 + ... + \beta_p x_p$$

where β is the weights vector and p the number of covariates considered in the model.

Maximization of the likelihood associated with the hazard function will lead us to find the optimal vector of weights $\beta = (\beta_1, ..., \beta_p)$. For the purpose of the following calculations, let us consider $D^{(n)} = \{D_i = (y_i, x^{(i)}) : i = 1, 2, ..., n\}$ to be the n independent observations in a given dataset.

Let's remember section 2.3.1. Assuming proportional hazards, the partial-likelihood function is the joint probability of observing an event for individual i at time t. Therefore, by considering $R_i = \{j : t_j \geq t_i\}$ to be the set of patients at risk at the same time as individual i, we can write the partial-likelihood as in Equation 6.2.

$$L(\beta|D^{(n)}) = \prod_{i=1}^{n} \frac{h(x^{(i)}; \beta)}{\sum_{j \in R_i} h(x^{(j)}; \beta)} \tag{6.2}$$

By simply applying the log function we obtain the log-partial-likelihood function, which properties make it easier to manipulate.

$$l(\beta|D^{(n)}) = log\left(\prod_{i=1}^{n} \frac{h(x^{(i)}; \beta)}{\sum_{j \in R_i} h(x^{(j)}; \beta)}\right) \tag{6.3}$$

Once the hazard function is time dependent, our partial-likelihood function is as well. By using the proprieties of the log function we can then isolate the baseline hazard function and make it so the objective function we want to maximize only depends on the parameters we are interested in. By following the steps written in A we simplify Equation 6.3 obtaining a non time dependent function, presented in Equation 6.4.

$$l(\beta|D^{(n)}) = \sum_{i=1}^{n}\left(f_\beta(x^{(i)}) - log\left(\sum_{j \in R_i} e^{f_\beta(x^{(j)})}\right)\right) \tag{6.4}$$

Let us note that maximizing the log-partial-likelihood function is the same as minimizing it's negative. This way, our aim is to find the weights β such that

$$\beta = \arg\min_{\beta}\{-l(\beta|D^{(n)})\} \tag{6.5}$$

Once the method being proposed relies on gradient descent, a first step is to calculate the gradient of the function we wish to minimize, that being the negative of Equation 6.4, in order to the set of weights β. Appendix B includes all the calculus necessary to obtain the gradient equation 6.6.

$$\nabla_\beta\{-l(\beta|D^{(n)})\} = -\sum_{i=1}^{n}\left(\nabla_\beta\{f_\beta(x^{(i)})\} - \frac{\sum_{j\in R_i}\nabla_\beta\{f_\beta(x^{(j)})\}e^{f_\beta(x^{(j)})}}{\sum_{j\in R_i}e^{f_\beta(x^{(j)})}}\right) \tag{6.6}$$

Now that the mathematical features were covered, we can move on to the gradient descent method itself.

6.3 Gradient Descent Method

The approach being proposed is one of distributed analysis, that is to say, analysis across multiple parties. That being said, the implementation being described in this section is used to show that the distribution works, in other words, gives the same (or very close) results as the conventional Cox Proportional Hazards model estimation.

6.3.1 Data Preparation

Before applying the gradient descent method to the Cox Proportional Hazards model, a series of crucial data preparation steps are essential to ensure the successful convergence and accuracy of the model. In this section, we outline the key data preparation procedures performed prior to applying the gradient descent optimization algorithm.

1. **Numerical covariates transformation:** Numerical covariates often have different ranges, which can lead to issues during gradient descent optimization. To address this, we perform feature scaling on the numerical covariates. Scaling brings all covariates to a similar range, typically the variable values are subtrated by it's mean and divided by the standard deviation, thereby preventing one feature from dominating the optimization process over others.

2. **Categorical covariates transformation:** Cox Proportional Hazards models can handle numerical covariates directly but require encoding categorical variables as dummy variables. We apply one-hot encoding to transform categorical variables into a set

of binary variables, where each category becomes a separate column with binary values (0 or 1). This process ensures that categorical variables are appropriately incorporated into the model.

3. **Data frame split:** To demonstrate the effectiveness of distributed analysis across parties, we divide the received data frame into smaller subsets or batches. Each batch represents a portion of the data, and we perform mini-batch gradient descent on these subsets. This division can be made randomly, if a batch size value is imputed in the function, or can be done accordingly to a set of indices for each batch, also imputed at first.

After performing this three steps we then create a list with all the batches being used in the study. Now that we have this list, gradient descent can be applied. The next section will show how the gradient is calculated given a set of observations.

6.3.2 Gradient Computation

When working with survival data, the order of the observations based on time is essential. Ordering the dataset by descending time ensures that the algorithm properly accounts for the censoring mechanism and captures the time-dependent nature of survival analysis.

Let's go back to the lung cancer data represented in Table 6.1. In order to exemplify how the gradient of the loss function is computed, we will focus only on the time, status, age, sex, ph.karno and pat.karno variables.

As Section 6.3.1 described, the method receives a list of batches. To make it simpler we will consider the first 10 observations of the data to represent one batch. Table 6.2 shows the data after the transformations mentioned in Section 6.3.1.

TABLE 6.2: 10 observations batch of the *lung* data after data preparation

Time	Status	Age	Sex	Ph.karno	Pat.karno
306	1	1.2656434	0	0.7410483	1.50
455	1	0.4907597	0	0.7410483	0.75
1010	0	-1.0590077	0	0.7410483	0.75
210	1	-0.9298605	0	0.7410483	-1.50
883	1	-0.5424186	0	1.3585885	0.75
1022	0	1.2656434	0	-1.7291126	0.00
310	1	0.4907597	1	-0.4940322	-1.50
361	1	0.8782015	1	-1.1115724	0.00
218	1	-1.4464496	0	-0.4940322	0.00
166	1	-0.4132713	0	-0.4940322	-0.75

For the purpose of this exemplification, we will consider a random vector of weights

$$\beta = (0.8485218, 0.6933678, 0.7041070, 0.2222322) \tag{6.7}$$

The first step on the gradient computation is to arrange the data in descending time order. In fact, this step is the only one that requires information from the time variable. From this moment on, all the steps performed rely on this arrange having been done correctly.

TABLE 6.3: Batch arranged by descending time order

Time	Status	Age	Sex	Ph.karno	Pat.karno
1022	0	1.2656434	0	-1.7291126	0.00
1010	0	-1.0590077	0	0.7410483	0.75
883	1	-0.5424186	0	1.3585885	0.75
455	1	0.4907597	0	0.7410483	0.75
361	1	0.8782015	1	-1.1115724	0.00
310	1	0.4907597	1	-0.4940322	-1.50
306	1	1.2656434	0	0.7410483	1.50
218	1	-1.4464496	0	-0.4940322	0.00
210	1	-0.9298605	0	0.7410483	-1.50
166	1	-0.4132713	0	-0.4940322	-0.75

Now we need to considered only the information provided by the covariates.

The most important aspect here is that considering two consecutive observations, the first one has been followed by a longer time.

Let's recall the gradient formula represented in Equation 6.6. The process of the gradient computation consists of constructing a matrix with n entries where each row is the contribution of each individual to the gradient. In the end, we will perform the multiplication by the status vector and them a column sum of this matrix, giving us the final vector.

We will divide the process by computing each parcel at a time, starting by the weights function's gradient.

$$\nabla_\beta\{f_\beta(x^{(i)})\} \tag{6.8}$$

As we have seen, the weights function is a linear one,

$$f_\beta(x) = x^T\beta = \beta_1 x_1 + ... + \beta_p x_p$$

and for that reason the gradient computed for each individual, in respect to β, actually is the vector of the covariates values belonging to this same individual. Matrix 6.9 shows the gradient calculated in respect to each individual.

$$\nabla_\beta\{f_\beta(x^{(i)})\} = \begin{pmatrix} \nabla_\beta\{f_\beta(x^{(1)})\} \\ \nabla_\beta\{f_\beta(x^{(2)})\} \\ \nabla_\beta\{f_\beta(x^{(3)})\} \\ \nabla_\beta\{f_\beta(x^{(4)})\} \\ \nabla_\beta\{f_\beta(x^{(5)})\} \\ \nabla_\beta\{f_\beta(x^{(6)})\} \\ \nabla_\beta\{f_\beta(x^{(7)})\} \\ \nabla_\beta\{f_\beta(x^{(8)})\} \\ \nabla_\beta\{f_\beta(x^{(9)})\} \\ \nabla_\beta\{f_\beta(x^{(10)})\} \end{pmatrix} = \begin{pmatrix} 1.2656434 & 0 & -1.7291126 & 0.00 \\ -1.0590077 & 0 & 0.7410483 & 0.75 \\ -0.5424186 & 0 & 1.3585885 & 0.75 \\ 0.4907597 & 0 & 0.7410483 & 0.75 \\ 0.8782015 & 1 & -1.1115724 & 0.00 \\ 0.4907597 & 1 & -0.4940322 & -1.50 \\ 1.2656434 & 0 & 0.7410483 & 1.50 \\ -1.4464496 & 0 & -0.4940322 & 0.00 \\ -0.9298605 & 0 & 0.7410483 & -1.50 \\ -0.4132713 & 0 & -0.4940322 & -0.75 \end{pmatrix} \tag{6.9}$$

Let's now move on to the next parcel, that being the quotient

$$\frac{\sum_{j \in R_i} \nabla_\beta \{f_\beta(x^{(j)})\} e^{f_\beta(x^{(j)})}}{\sum_{j \in R_i} e^{f_\beta(x^{(j)})}} \tag{6.10}$$

Let's start by computing what is usually called the scores vector. The scores vector contains n rows, where row i keeps the value of $e^{f_\beta(x^{(i)})}$.

$$e^{f_\beta(x^{(i)})} = \begin{pmatrix} e^{f_\beta(x^{(1)})} \\ e^{f_\beta(x^{(2)})} \\ e^{f_\beta(x^{(3)})} \\ e^{f_\beta(x^{(4)})} \\ e^{f_\beta(x^{(5)})} \\ e^{f_\beta(x^{(6)})} \\ e^{f_\beta(x^{(7)})} \\ e^{f_\beta(x^{(8)})} \\ e^{f_\beta(x^{(9)})} \\ e^{f_\beta(x^{(10)})} \end{pmatrix} = \begin{pmatrix} 0.8662739 \\ 0.8104710 \\ 1.9406283 \\ 3.0188372 \\ 1.9268280 \\ 1.5350888 \\ 6.8829797 \\ 0.2069671 \\ 0.5484945 \\ 0.4209693 \end{pmatrix} \tag{6.11}$$

Focusing now on the nominator of Equation 6.10. We need to consider, for each individual i, the sum of the multiplication between the weights function gradient and the score value, calculated only in respect to the individuals who belong to the individual's i risk set. Using a cumulative sum strategy, this computation becomes straight forward.

We first take the $\nabla_\beta \{f_\beta(x)\}$ matrix, 6.9, and multiply every element from row i by the ith row element on the scores matrix, 6.11. We compute this for every individual and we get a matrix with the same dimensions as the $\nabla_\beta \{f_\beta(x)\}$ matrix.

$$\nabla_\beta\{f_\beta(x^{(i)})\}e^{f_\beta(x^{(i)})} = \begin{pmatrix} 1.0963938 & 0.000000 & -1.4978851 & 0.0000000 \\ -0.8582950 & 0.000000 & 0.6005981 & 0.6078532 \\ -1.0526329 & 0.000000 & 2.6365153 & 1.4554712 \\ 1.4815236 & 0.000000 & 2.2371041 & 2.2641279 \\ 1.6921433 & 1.926828 & -2.1418088 & 0.0000000 \\ 0.7533597 & 1.535089 & -0.7583833 & -2.3026332 \\ 8.7113978 & 0.000000 & 5.1006202 & 10.3244695 \\ -0.2993674 & 0.000000 & -0.1022484 & 0.0000000 \\ -0.5100234 & 0.000000 & 0.4064609 & -0.8227418 \\ -0.1739745 & 0.000000 & -0.2079724 & -0.3157270 \end{pmatrix} \tag{6.12}$$

At this point, remember we started by arranging the data in descending time order. That being said, if we perform a cumulative sum of the matrix's columns we get the result we wanted. Let's take the fourth individual for example. The risk set R_4 includes only individuals 1, 2 and 3, once they were the only ones followed for a longer time than individual 4. Therefore, the fourth row on the matrix below, corresponds to the sum of the multiplication of the gradient with the score for individuals one to three, that being exactly $\sum_{j\in R_4} \nabla_\beta\{f_\beta(x^{(j)})\}e^{f_\beta(x^{(j)})}$.

$$\sum_{j\in R_i} \nabla_\beta\{f_\beta(x^{(j)})\}e^{f_\beta(x^{(j)})} = \begin{pmatrix} 1.0963938 & 0.000000 & -1.497885 & 0.0000000 \\ 0.2380988 & 0.000000 & -0.897287 & 0.6078532 \\ -0.8145341 & 0.000000 & 1.739228 & 2.0633244 \\ 0.6669895 & 0.000000 & 3.976332 & 4.3274523 \\ 2.3591328 & 1.926828 & 1.834524 & 4.3274523 \\ 3.1124925 & 3.461917 & 1.076140 & 2.0248191 \\ 11.8238903 & 3.461917 & 6.176760 & 12.3492886 \\ 11.5245229 & 3.461917 & 6.074512 & 12.3492886 \\ 11.0144995 & 3.461917 & 6.480973 & 11.5265468 \\ 10.8405250 & 3.461917 & 6.273001 & 11.2108199 \end{pmatrix} \tag{6.13}$$

The last parcel to compute is the Equation's 6.10 denominator. To calculate $\sum_{j \in R_i} e^{f_\beta(x^{(j)})}$ we use the same strategy mentioned to compute the last matrix, but this time we calculate the cumulative sums matrix of the scores matrix only.

$$\sum_{j \in R_i} e^{f_\beta(x^{(j)})} = \begin{pmatrix} 0.8662739 \\ 1.6767448 \\ 3.6173731 \\ 6.6362103 \\ 8.5630383 \\ 10.0981271 \\ 16.9811068 \\ 17.1880739 \\ 17.7365684 \\ 18.1575376 \end{pmatrix} \tag{6.14}$$

Now that we have all the parcels computed, the last step is to join them together to obtain the gradient vector.

Equation 6.10 is computed by diving each element of the $\nabla_\beta\{f_\beta(x)\}$ matrix, 6.9, by the the corresponding row element of the scores matrix, 6.11. We then subtract the quotient to the $\nabla_\beta\{f_\beta(x)\}$ matrix, 6.9.

Before performing the sum of the columns of the resulting matrix, we need to consider the censuring occurred in the data. To do that, we simply multiply the matrix by the event variable. By doing this, the information of the censored individuals is not added to the final gradient, even though they contributed to the calculation of the non censored individuals contribution to the gradient (once they were still part of some risk set).

$$\nabla_\beta\{f_\beta(x^{(i)})\} - \frac{\sum_{j\in R_i}\nabla_\beta\{f_\beta(x^{(j)})\}e^{f_\beta(x^{(j)})}}{\sum_{j\in R_i}e^{f_\beta(x^{(j)})}} = \begin{pmatrix} 0.0000000 & 0.0000000 & 0.0000000 & 0.00000000 \\ 0.0000000 & 0.0000000 & 0.0000000 & 0.00000000 \\ -0.3172458 & 0.0000000 & 0.8777898 & 0.17960696 \\ 0.3902521 & 0.0000000 & 0.1418611 & 0.09790307 \\ 0.6026997 & 0.7749831 & -1.3258099 & -0.50536412 \\ 0.1825350 & 0.6571724 & -0.6006005 & -1.70051432 \\ 0.5693466 & -0.2038687 & 0.3773052 & 0.77276303 \\ -2.1169449 & -0.2014139 & -0.8474465 & -0.71848008 \\ -1.5508656 & -0.1951853 & 0.3756465 & -2.14987469 \\ -1.0102975 & -0.1906600 & -0.8395086 & -1.36741961 \end{pmatrix} \tag{6.15}$$

The final matrix contains the contribution of each individual to the gradient. We now need to perform the sum of the columns and take the negative to get the gradient vector.

age	**sex**	**ph.karno**	**pat.karno**
−3.2505204	0.6410276	−1.8407629	−5.3913798

(6.16)

This section described how to calculate the gradient for a batch of data. Yet, we haven't showed how to perform gradient descent across the batches, or to update parameters iteratively. To do that we will now talk about gradient aggregation and parameter update.

6.3.3 Gradient Aggregation

This implementation performs analysis across batches by, at each step, computing the gradient for each batch, taking the mean of the resulting vector and obtained a global gradient vector, as Figure 6.1 suggests.

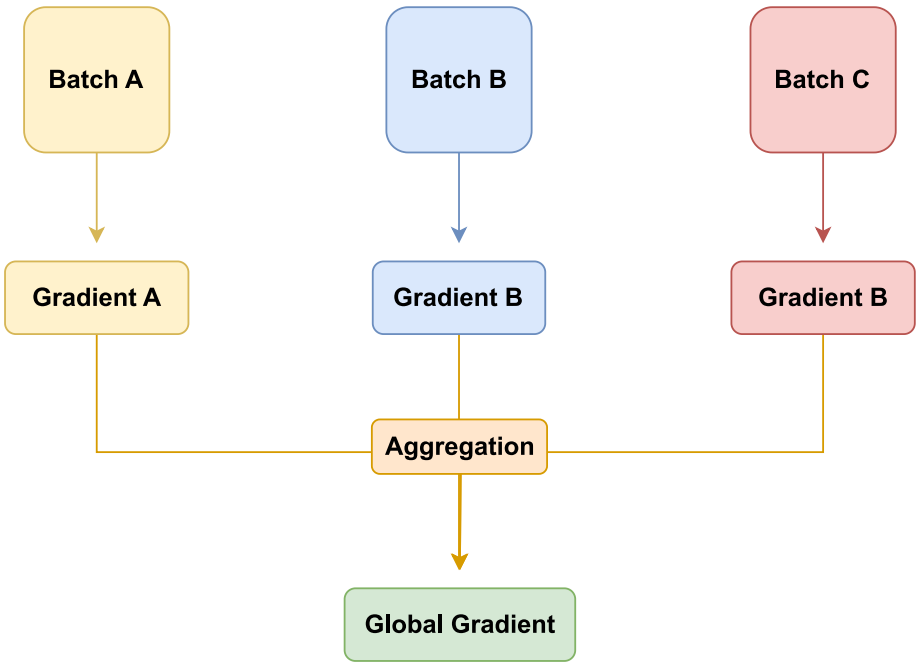


FIGURE 6.1: Gradient Aggregation Across Batches

This strategy allows to gather information from different sources, and therefore different loss functions, and then apply an aggregation function to obtain a result that resembles information from a global loss function.

As we will see in the next section, the global gradient allows us to take a step into the global parameters estimate.

6.3.4 Parameter Update

Section 6.3.2 shows how to compute the gradient of the log-partial likelihood hazard function, using a time rank strategy, and Section 6.3.3 describes how to aggregate the gradients from different sources. This computing and aggregation is the core of the method being proposed in this work.

The next step is to make the method iterable, by adopting a gradient descent approach.

Remembering we start with a initial weights vector, random or not, the process consist in, at each step, compute the global gradient and subtract it to the parameter vector, moving towards the minimum.

To ensure that the closer we get to the minimum, the smaller the step will be, we use an adaptive learning rate function, that depends only on the number of the iteration in question, for example Equation 6.17.

$$learinng.rate(x) = \frac{1}{100\sqrt{x}} \tag{6.17}$$

At each iteration, we multiply the learning rate to the gradient vector before performing the subtraction mentioned above. Consider i to be the iteration number, Equation 6.18 gives the update rule for consecutive parameters.

$$\beta_i = \beta_{i-1} - learning.rate(i) * gradient \tag{6.18}$$

We will consider convergence to be obtained when the norm between consecutive parameter vectors is smaller than a certain error value (ϵ) imputed in the function. The updating will also stop if a max number of iterations, also imputed in the function, is reached.

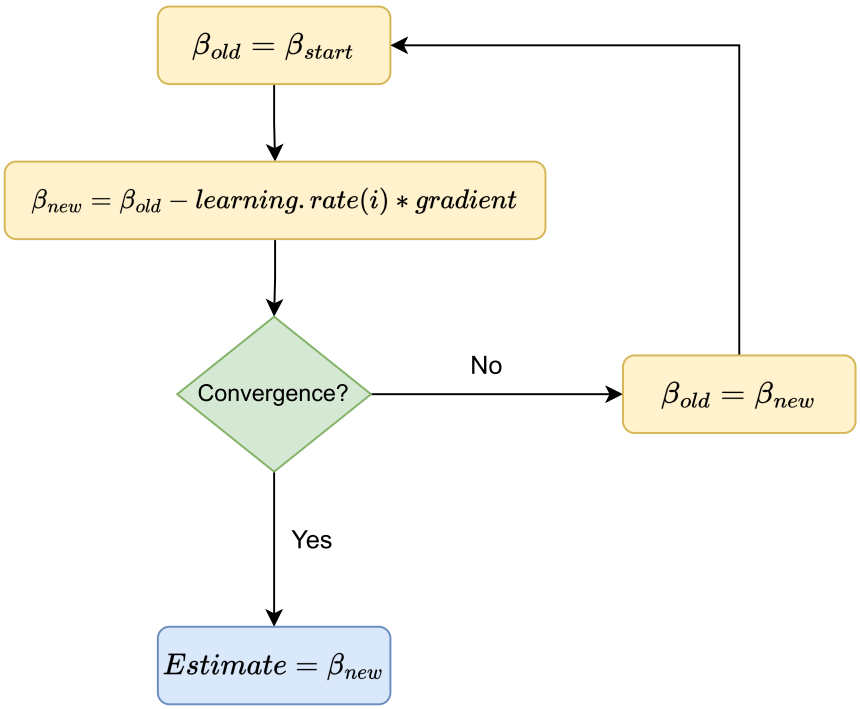


FIGURE 6.2: Parameter Update Rule

Figure 6.2 describes the steps towards the computation of the hazard ratios estimates. Having described the steps necessary to reach the given estimates, the final step is to show how the confidence intervals are computed.

6.3.5 Bootstrapping Confidence Intervals

To construct the confidence intervals for the hazard ratios we adopt a Bootstrapping resampling strategy.

Let's take B to be the number of resamples we will be considering. For each $b \in [1, B]$ we start by resampling (with replacement) the batches imputed in the function. Then we perform the gradient descent method using the original estimate, $\tilde{\beta}$, as our starting weight vector. We then take the difference between the resample's estimate and the original estimate and store it, just like the diagram on Figure 6.3 suggests.

In the end we are left with B difference vectors stored in a matrix. We now need to take the quantiles in reference to the confidence level (α) imputed in the function and subtract them to the orignal estimate, obtaining the confidence intervals.

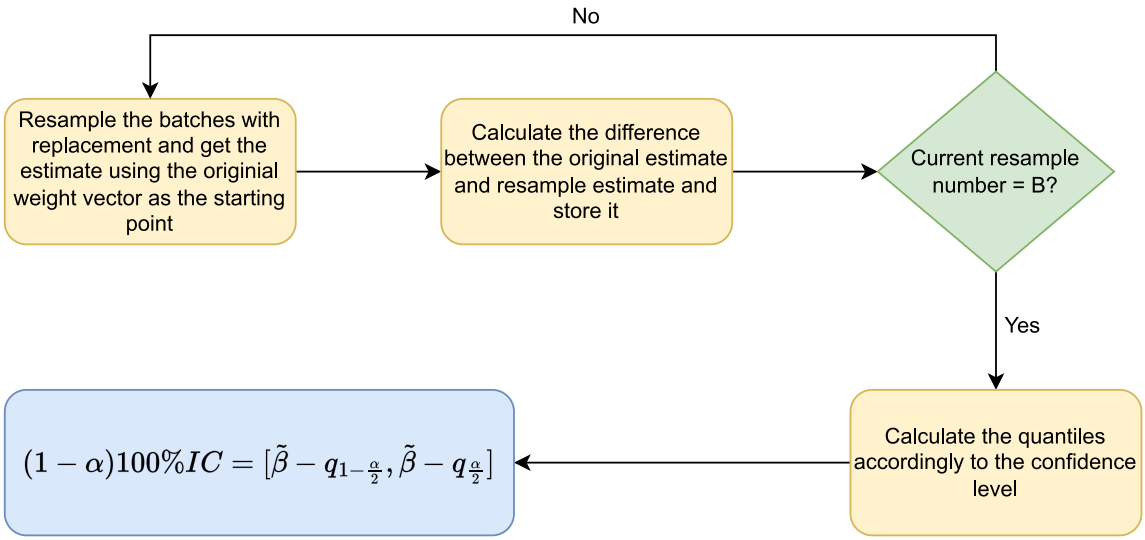


FIGURE 6.3: Bootstrapping Confidence Intervals Computation

6.3.6 Architecture

Now that the implementation's steps were covered, it makes sense to show a diagram of the architecture of the method, bounding all the steps, Figure 6.4. Then, we can move on to the Results chapter, where the method is tested and evaluated.

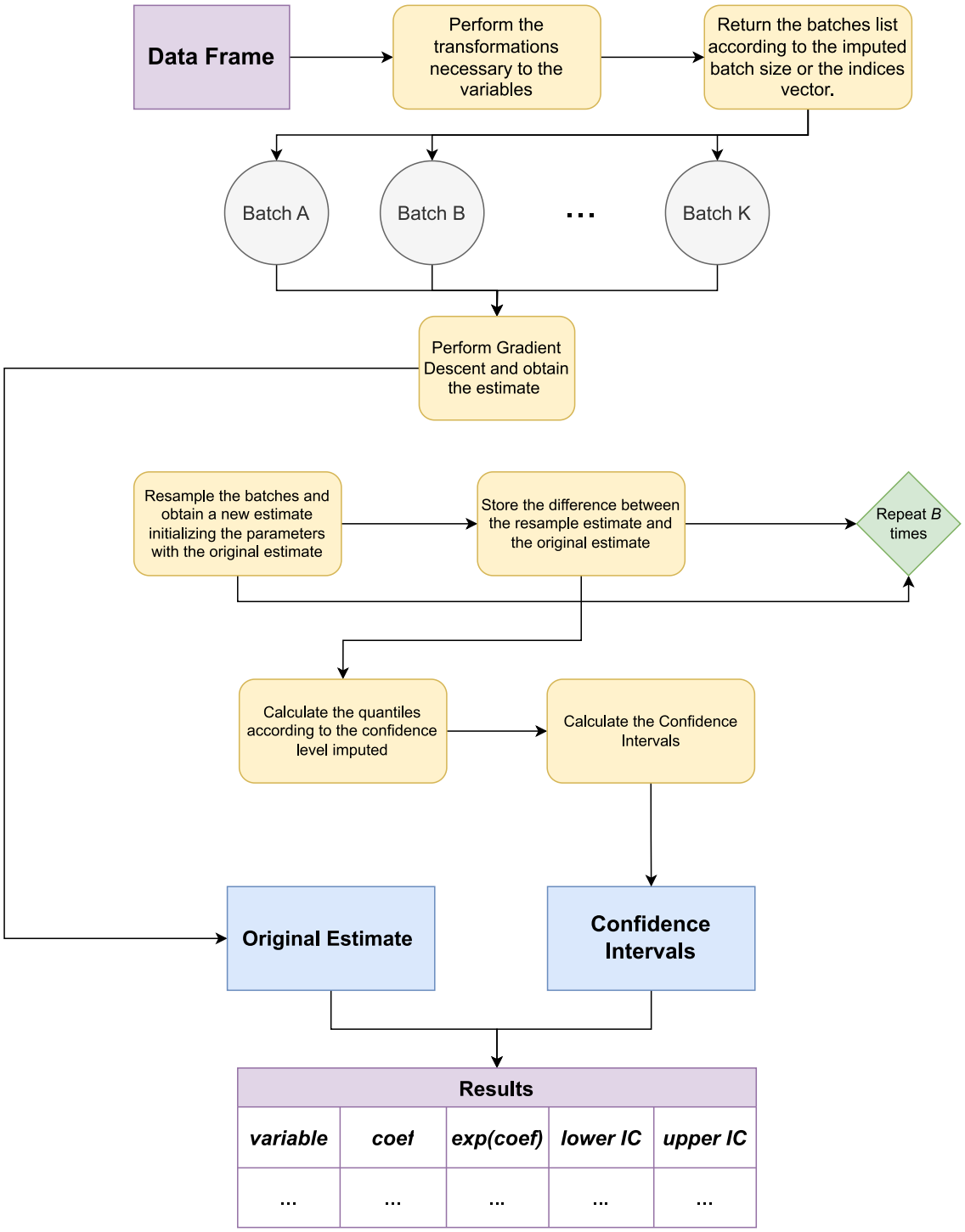


FIGURE 6.4: Architeture

Chapter 7

Results

Chapter 6 described the steps involved in the distribution of the Cox Proportional Hazards model. In this Chapter we will focus on the application of this method in datasets available in the *survival R* package.

The procedure involves computing the coefficients and confidence intervals with the *coxph()* function to get our gold standard results, and then apply the method being proposed to further compare the two. In other words, perform the standard analysis and the distributed analysis and compare the two.

7.1 Colon Chemotherapy Dataset for Stage B/C Colon Cancer

The colon chemotherapy dataset provides data from one of the pioneering trials that investigated the effectiveness of adjuvant chemotherapy for stage B/C colon cancer. The dataset consists of information related to patient characteristics, treatment approaches, and clinical outcomes.

For the purpose of this analysis we focused only on the following variables:

time: Represents the time in days until the event (recurrence or death) or censoring.

status: Indicates the censoring status for each patient. A value of 1 represents an event (recurrence or death), while a value of 0 indicates censoring.

age: Represents the age of the patient at the time of diagnosis.

nodes: Denotes the number of lymph nodes with detectable cancer. Serves as a crucial indicator of disease severity and potential metastasis. Can be used to assess the impact of lymph node involvement on prognosis and treatment outcomes.

sex: Represents the gender of the patient.

obstruct: Indicates the presence or absence of colon obstruction caused by the tumor.

rx: Represents the treatment received by patients. Categorized as ″Observation″ (Obs), ″Levamisole″ (Lev), or ″Levamisole + 5-Fluorouracil″ (Lev+5-FU).

surg: Indicates the time elapsed from surgery to registration. Categorized as ″Short″ (0) or ″Long″ (1).

7.1.1 Gold standard

Once the main goal is to assess the behaviour of our method, we must compare it's results with a gold standard. In this case, the gold standard results are the coefficients obatained by applying the *coxph()* function to the variables listed above, fitting a Cox Proportional Hazard's model to the data.

Surv(time, status) age + nodes + sex + obstruct + rx + surg

TABLE 7.1: Summary of the *coxph()* function

Variable	**coef**	***exp(Coef)***	**se(coef)**	***z***	***p***
age	0.01911	**1.01930**	0.03395	0.563	0.573417
nodes	0.31298	**1.36750**	0.02289	13.673	$<$ 2e-16 ***
sex1	-0.10121	**0.90374**	0.06808	-1.487	0.137141
obstruct1	0.23086	**1.25968**	0.08333	2.770	0.005600 **
rxLev	-0.05748	**0.94414**	0.07916	-0.726	0.467765
rxLev+5FU	-0.42180	**0.65587**	0.08611	-4.898	9.67e-07 ***
surg1	0.24541	**1.27815**	0.07433	3.302	0.000961 ***

TABLE 7.2: Summary of the *coxph()* function: confidence intervals

Variable	**exp(coef)**	**exp(-coef)**	**lower .95**	**upper .95**
age	1.0193	0.9811	**0.9537**	**1.0894**
nodes	1.3675	0.7313	**1.3075**	**1.4302**
sex1	0.9037	1.1065	**0.7908**	**1.0328**
obstruct1	1.2597	0.7939	**1.0699**	**1.4832**
rxLev	0.9441	1.0592	**0.8084**	**1.1026**
rxLev+5FU	0.6559	1.5247	**0.5540**	**0.7765**
surg1	1.2781	0.7824	**1.1049**	**1.4786**

TABLE 7.3: Summary of the *coxph()* function: quality measures

Concordance	0.642 (se = 0.009)
Likelihood ratio test	179.3 on 7 df, $p < 2e-16$
Wald test	234 on 7 df, $p < 2e-16$
Score (logrank) test	237.4 on 7 df, $p < 2e-16$

Although our primary objective does not center around interpreting the results, undertaking such an analysis is not unwarranted. Let us commence the examination of the resulting coefficients and the significance associated with each variable.

Age: The coefficient of 0.01911 suggests a small positive association with the hazard of the event. However, the p-value of 0.573 indicates that this association is not statistically significant.

Nodes: The coefficient of 0.31298 indicates a significant positive association with the hazard of the event ($p < 2 \times 10^{-16}$). For each unit increase in the number of nodes, the hazard ratio (exp(coef)) is 1.3675, implying an increased risk of the event.

Sex: The coefficient of -0.10121 suggests a negative association with the hazard of the event. However, the p-value of 0.137141 indicates that this association is not statistically significant.

Obstruct: The coefficient of 0.23086 indicates a significant positive association with the hazard of the event ($p = 0.005600$). Presence of obstruction in the colon by tumor increases the risk of the event.

Rx (Lev): The coefficient of -0.05748 suggests a small negative association with the hazard of the event. However, the p-value of 0.467765 indicates that this association is not statistically significant.

Rx (Lev+5FU): The coefficient of -0.42180 suggests a significant negative association with the hazard of the event (p ¡ 9.67e-07). Patients receiving the combination treatment of Levamisole and 5-FU have a lower hazard of the event compared to the observation group.

Surg: The coefficient of 0.24541 indicates a significant positive association with the hazard of the event ($p = 0.000961$). A longer time from surgery to registration increases the risk of the event.

The hazard ratios (exp(coef)) indicate the relative change in the hazard of the event associated with a one-unit increase in each variable. For example, the hazard ratio for nodes is 1.3675, indicating that for each additional node with detectable cancer, the hazard of the event increases by approximately 36.75%.

The concordance value of 0.642 indicates the predictive ability of the model, with higher values indicating better predictive performance.

Likelihood ratio test, Wald test, and Score (logrank) test are statistical tests assessing the overall significance of the model. In all three tests, the p-values are less than 2×10^{-16}, suggesting strong evidence against the null hypothesis and indicating that the model is statistically significant.

Overall, the Cox proportional hazards model suggests that the number of nodes, obstruction, rx treatment and time from surgery to registration (surg) are significant predictors of the event outcome. Other variables, such as age and sex, do not show a significant association with the event outcome in this analysis. Let's now move on to applying distributed model.

7.1.2 Distributed Cox Proportion Hazard's

To start, we consider the data to be one batch, and perform the analysis using gradient descent instead of the tradition *coxph()* estimation technique. Following that, we perform the splitting of the data into subsets and apply distributed gradient descent.

Convergence was considered to have been obtained once the norm of the difference vector between to consecutive iterations was less than $\epsilon = 1.00 \times 10^{-7}$.

To better visualise the behaviour of the method, Figure 7.1 shows the error value for each iteration. In this case, in order to be able to plot this measure, the error was taken to be the norm of the vector that results of the subtraction of the gold standard vector and the batch gradient descent resulting vector, and taking the absolute value obviously. For this specific analysis, the method needed only 191 iterations to converge.

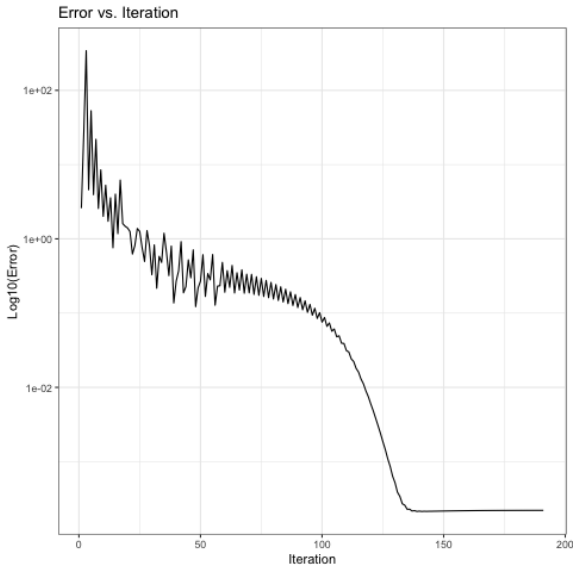


FIGURE 7.1: Batch GD behaviour

In order to simulate a distributed analysis of this kind, we proceeded to split the data frame into 4 disjoint batches representing, hypothetically, 4 different health centers holding batches of data with the same characteristics. As mentioned before in Chapter 6, the

method worked by calculating the gradient for each of the 4 individual loss functions, aggregating them by taking the mean vector, and then taking a step in gradient descent direction. Choosing to split the data into 4 batches specifically was arbitrary.

Figure 7.2 shows the equivalent graph to Figure 7.1 for the distributed procedure. In this case there were needed 771 iterations to achieve convergence, which makes sense once the data are distributed in 4 different places.

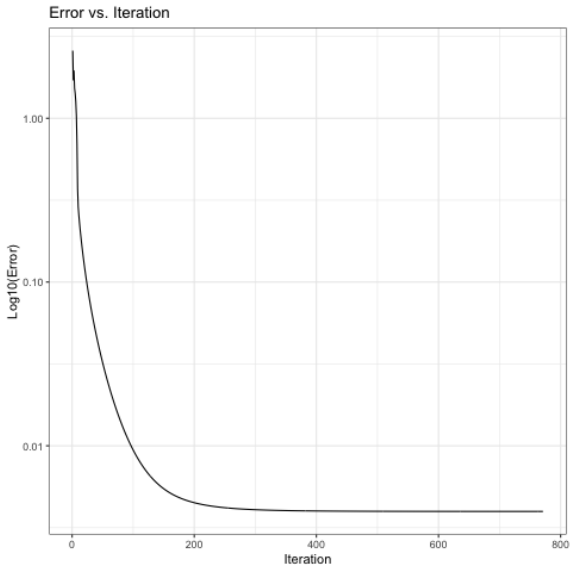


FIGURE 7.2: Mini-batch GD behaviour

Table 7.4 shows the results of the coefficients obatined by the 2 analysis refered above, as-well as the gold standard results. Note that the concordance index with respect to the *coxph()* is different than the one shown in the summary of the model in Table 7.3. This is due to the fact that there are several ways to calculate the c-index. Nevertheless, we obtained almost the same index in the three methods.

TABLE 7.4: Coxph, One-batch and Distributed Coefficients

Variable	***coxph()***	**1 Batch**	**4 Batches**
age	1.0192980	1.0193735	1.0195413
nodes	1.3674958	1.3675203	1.3695729
sex	0.9037442	0.9037792	0.9024311
obstruct	1.2596834	1.2598646	1.2598374
rxLev	0.9441378	0.9441485	0.9467845
rxLev+5FU	0.6558669	0.6558325	0.6568776
surg	1.2781466	1.2782300	1.2768666
Concordance	0.4236017	0.4235992	0.4235395

In order to quantify the similarity between our results and the *coxph()* results, Table 7.5 shows the absolute values difference vectors obtained by subtracting the gold standard coefficients and the batch and mini-batch ones. As we can see, the largest difference in coefficients is of 2.65×10^{-3}, which is a residual value that indicates almost no difference between theese two methods and the *coxph()* fitting model method.

TABLE 7.5: Error Vectors

Variable	1 Batch	4 Batches
age	7.55×10^{-5}	2.43×10^{-4}
nodes	2.45×10^{-5}	2.08×10^{-3}
sex	3.50×10^{-5}	1.31×10^{-3}
obstruct	1.81×10^{-4}	1.54×10^{-4}
rxLev	1.07×10^{-5}	2.65×10^{-3}
rxLev+5FU	3.44×10^{-5}	1.01×10^{-3}
surg	8.35×10^{-5}	1.28×10^{-3}

Adopting a Bootstrapping based strategy we are able to construct the confidence intervals as shown in Section 6.3.5. The resamples number was set to 100. Tables 7.6 and 7.7 show the confidence intervals provided by the 1 Batch and 4 Batches analysis, respectively.

TABLE 7.6: 1 Batch Confidence Intervals

Variable	Lower CI: 1 Batch(Coxph)	Upper CI: 1 Batch(Coxph)
age	0.9619 (0.9537)	1.0779 (1.0894)
nodes	1.2842 (1.3075)	1.4292 (1.4302)
sex	0.7985 (0.7908)	1.0347 (1.0328)
obstruct	1.0805 (1.0699)	1.4845 (1.4832)
rxLev	0.8282 (0.8084)	1.1360 (1.1026)
rxLev+5FU	0.5507 (0.5540)	0.7642 (0.7765)
surg	1.1205 (1.1049)	1.4592 (1.4786)

TABLE 7.7: 4 Batches Confidence Intervals

Variable	Lower CI: 4 Batches(Coxph)	Upper CI: 4 Batches(Coxph)
age	0.9527 (0.9537)	1.0825 (1.0894)
nodes	1.2508 (1.3075)	1.4352 (1.4302)
sex	0.7867 (0.7908)	1.0041 (1.0328)
obstruct	1.0532 (1.0699)	1.4515 (1.4832)
rxLev	0.8205 (0.8084)	1.0880 (1.1026)
rxLev+5FU	0.5608 (0.5540)	0.7601 (0.7765)
surg	1.1185 (1.1049)	1.5232 (1.4786)

7.2 Assay of Serum Free Light Chain

The Assay of Serum Free Light Chain dataset is a stratified random sample containing 1/2 of the subjects from a study of the relationship between serum free light chain (FLC) and mortality. The original sample contains samples on approximately 2/3 of the residents of Olmsted County aged 50 or greater. A total of 7874 observations take part in this dataset, for a total of 11 variables.

In Section 7.1 we tested the algorithm only with numerical or binary variables. However, as it happens in most health Big Data analysis, dataframes contain categorical variables with several levels. That being the case, it makes sense to test our approach in a scenario like that. The *flchain* data provides us the opportunity to test that, using, among other, the *flc.grp* variable as a covariate of interest.

The variables choosen to perform the method, as well as a brief description, follow below:

futime: Represents the number of days from enrollment until death or Censoring.

death: A binary variable indicating the status of the subject at the last contact date. A value of 1 means the subject is dead, while 0 indicates the subject was alive at last contact.

age: This variable represents the age of the subject in years at the time of enrollment in the study.

mgus: This binary variable indicates whether the subject had been diagnosed with monoclonal gammopathy (MGUS), with 1 meaning yes and 0 meaning no.

sex: Categorical variable indicating the gender of the subject, where F represents female and M represents male.

flc.grp: This variable indicates the FLC group for the subject, as used in the original analysis. It is important for grouping subjects based on the levels of Free Light Chains (FLC) in the serum.

Similarly to the analysis made for the Colon Cancer data, we will start by showing the gold standard results.

7.2.1 Gold Standard

Surv(futime, death) age + mgus + sex + flc.grp

TABLE 7.8: Summary of the *coxph()* function

Variable	**coef**	***exp(Coef)***	**se(coef)**	***z***	***p***
age	1.06490	**2.90055**	0.02492	42.727	< 2e-16 ***
mgus1	0.05925	**1.06104**	0.25909	0.229	0.81912
sexM	0.31482	**1.37001**	0.04415	7.130	1.00e-12 ***
flc.grp2	-0.04111	**0.95973**	0.13181	-0.312	0.75514
flc.grp3	0.14069	**1.15106**	0.12764	1.102	0.27036
flc.grp4	0.17002	**1.18533**	0.12523	1.358	0.17457
flc.grp5	0.16806	**1.18301**	0.12536	1.341	0.18003
flc.grp6	0.38318	**1.46694**	0.11870	3.228	0.00125 **
flc.grp7	0.29715	**1.34602**	0.11834	2.511	0.01204 *
flc.grp8	0.46306	**1.58893**	0.11628	3.982	6.83e-05 ***
flc.grp9	0.50677	**1.65992**	0.11278	4.493	7.01e-06 ***
flc.grp10	1.06424	**2.89864**	0.10939	9.729	< 2e-16 ***

TABLE 7.9: Summary of the *coxph()* function: confidence intervals

Variable	**exp(coef)**	**exp(-coef)**	**lower .95**	**upper .95**
age	2.9006	0.3448	**2.7623**	**3.046**
mgus1	1.0610	0.9425	**0.6385**	**1.763**
sexM	1.3700	0.7299	**1.2564**	**1.494**
flc.grp2	0.9597	1.0420	**0.7412**	**1.243**
flc.grp3	1.1511	0.8688	**0.8963**	**1.478**
flc.grp4	1.1853	0.8436	**0.9273**	**1.515**
flc.grp5	1.1830	0.8453	**0.9253**	**1.512**
flc.grp6	1.4669	0.6817	**1.1624**	**1.851**
flc.grp7	1.3460	0.7429	**1.0674**	**1.697**
flc.grp8	1.5889	0.6294	**1.2651**	**1.996**
flc.grp9	1.6599	0.6024	**1.3307**	**2.071**
flc.grp10	2.8986	0.3450	**2.3393**	**3.592**

TABLE 7.10: Summary of the *coxph()* function: quality measures

Concordance	0.795 (se = 0.005)
Likelihood ratio test	2847 on 12 df, $p < 2e - 16$
Wald test	2771 on 12 df, $p < 2e - 16$
Score (logrank) test	3408 on 12 df, $p < 2e - 16$

Age: The coefficient of 1.06490 indicates a significant positive association with the hazard of the event ($p < 2 \times 10^{-16}$). For each unit increase in age, the hazard ratio is 2.90055, implying an increased risk of the event.

Mgus: The coefficient of 0.05925 suggests a small positive association with the hazard of the event. However, the p-value of 0.81912 indicates that this association is not statistically significant.

Sex: The coefficient of 0.31482 indicates a significant positive association with the hazard of the event ($p = 1.00 \times 10^{-12}$). This suggests that being male (sexM) increases the risk of the event.

FLC groups: The coefficients for the FLC groups vary. Some groups (flc.grp6, flc.grp7, flc.grp8, flc.grp9, and flc.grp10) show a significant association with the hazard of the event, while others (flc.grp2, flc.grp3, flc.grp4, and flc.grp5) do not. The associated hazard ratios suggest varying levels of risk depending on the FLC group.

The concordance value of 0.795 indicates the predictive ability of the model, with higher values indicating better predictive performance.

Likelihood ratio test, Wald test, and Score (logrank) test are statistical tests assessing the overall significance of the model. In all three tests, the p-values are less than 2×10^{-16}, suggesting strong evidence against the null hypothesis and indicating that the model is statistically significant.

Overall, the Cox proportional hazards model suggests that age, sex, and certain FLC groups are significant predictors of the event outcome. Other variables, such as mgus and some of the FLC groups, do not show a significant association with the event outcome in this analysis.

7.2.2 Distributed Cox Proportional Hazard's

Regarding the Distributed analysis it makes sense to test how the method performs varying the numvber of batches. Table 7.17 shows the results from the gradient method when the number of subsets from the original data frame varies from 1 to 5.

TABLE 7.11: Coefficients from the *coxph()* results and from the Distributed Cox Proportionals Hazards analysis across 1 to 5 bathces

Variable	*coxph()*	1 Batch	2 Batches	3 Batches	4 Batches	5 Batches
age	2.90055	2.90012	2.89921	2.89991	2.90145	2.89939
mgus1	1.06104	1.05926	1.05516	1.04842	1.07359	1.07874
sexM	1.37001	1.37002	1.36929	1.37218	1.36961	1.37358
flc.grp2	0.95973	0.95875	0.95503	0.94311	0.93317	0.92531
flc.grp3	1.15106	1.14988	1.14569	1.13113	1.12307	1.11395
flc.grp4	1.18533	1.18412	1.17842	1.16181	1.15677	1.14062
flc.grp5	1.18301	1.18180	1.17641	1.16197	1.15398	1.13587
flc.grp6	1.46694	1.46549	1.46033	1.44008	1.42622	1.41677
flc.grp7	1.34602	1.34465	1.33768	1.32200	1.31212	1.29846
flc.grp8	1.58893	1.58732	1.57737	1.55884	1.55107	1.52545
flc.grp9	1.65992	1.65825	1.65193	1.63047	1.61904	1.60432
flc.grp10	2.89864	2.89572	2.88675	2.86007	2.82594	2.81134

Similarly to Table 7.5, Table 7.12 shows the absolute values of the difference between the distributed coefficients and the goal standard results. The maximum difference registered was of 0.08730, present in the 5 batches analysis.

TABLE 7.12: Absolute Differences between *coxph()* and Distributed analysis

Variable	1 Batch	2 Batches	3 Batches	4 Batches	5 Batches
age	0.00043	0.00134	0.00064	0.00090	0.00106
mgus1	0.00178	0.00588	0.01262	0.01255	0.01770
sexM	0.00001	0.00072	0.00217	0.00060	0.00357
flc.grp2	0.00098	0.00470	0.01662	0.02656	0.03442
flc.grp3	0.00118	0.00537	0.01993	0.02799	0.03711
flc.grp4	0.00121	0.00691	0.02352	0.02856	0.04471
flc.grp5	0.00121	0.00660	0.02104	0.03004	0.04714
flc.grp6	0.00145	0.00661	0.02686	0.04072	0.05017
flc.grp7	0.00137	0.00834	0.02402	0.02490	0.04756
flc.grp8	0.00161	0.01156	0.03009	0.03786	0.06348
flc.grp9	0.00167	0.00899	0.02945	0.02988	0.05560
flc.grp10	0.00292	0.01189	0.03757	0.03857	0.08730

Having the values from Table 7.12, we will try to understand what are the variables that "suffer" the most.

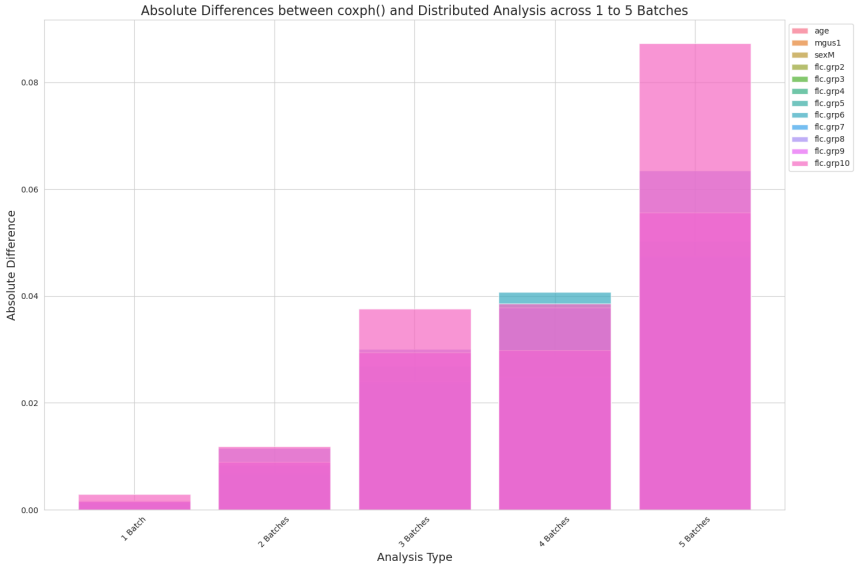


FIGURE 7.3: Absolute differences

From Figure 7.3 can interpret two main topics. First, and as expected, the difference between the gold standard results and the distributed analysis output increases with the number of batches. And second, we can see that the error increases much more in the dummy variables of the *flc.group* covariate. This is probably due to the fact that the different levels of this variable are not as well represented as the ones from the other categorical variables, like *sex* for example.

To finish, the following tables give the results of the coefficients joined with the confidence intervals for 1 to 5 batches.

TABLE 7.13: 1 Batch Analysis Results

Variable	coef	exp(coef)	lower .95	upper .95
age	1.06475294	2.9001224	2.7750150	3.035030
mgus_1	0.05757480	1.0592645	0.7224639	1.698816
sex_M	0.31482534	1.3700200	1.2619615	1.480489
flc.grp_2	-0.04212468	0.9587502	0.7186554	1.206961
flc.grp_3	0.13965878	1.1498814	0.9183617	1.432862
flc.grp_4	0.16900598	1.1841272	0.8954182	1.516233
flc.grp_5	0.16704487	1.1818073	0.9537580	1.521650
flc.grp_6	0.38219118	1.4654922	1.1599442	1.781312
flc.grp_7	0.29613805	1.3446558	1.0907255	1.666457
flc.grp_8	0.46204913	1.5873233	1.2982366	1.943210
flc.grp_9	0.50576503	1.6582537	1.3568354	2.010187
flc.grp_10	1.06323521	2.8957241	2.3614765	3.458557

TABLE 7.14: 2 Batches Analysis Results

Variable	coef	exp(coef)	lower .95	upper .95
age	1.06462741	2.8997583	2.7482468	3.030058
mgus_1	0.05312513	1.0545616	0.7467739	1.697452
sex_M	0.31456358	1.3696614	1.2613033	1.477076
flc.grp_2	-0.04772152	0.9533993	0.7763895	1.217354
flc.grp_3	0.13400193	1.1433950	0.9132039	1.384666
flc.grp_4	0.16225704	1.1761625	0.9233406	1.514825
flc.grp_5	0.15966302	1.1731155	0.9521026	1.440665
flc.grp_6	0.37851269	1.4601113	1.2435356	1.816914
flc.grp_7	0.29282504	1.3402083	1.1040714	1.593306
flc.grp_8	0.45811669	1.5810935	1.2591638	1.993067
flc.grp_9	0.50199642	1.6520161	1.4247986	1.956426
flc.grp_10	1.05812167	2.8809545	2.4291091	3.407524

TABLE 7.15: 3 Batches Analysis Results

Variable	coef	exp(coef)	lower .95	upper .95
age	1.06428551	2.8987671	2.7460207	3.033772
mgus_1	0.05040409	1.0516960	0.7095902	1.722558
sex_M	0.31562058	1.3711099	1.2738953	1.487340
flc.grp_2	-0.05656166	0.9450082	0.7733282	1.104298
flc.grp_3	0.12640262	1.1347389	0.9122153	1.422895
flc.grp_4	0.15434254	1.1668905	0.9476734	1.385793
flc.grp_5	0.15224023	1.1644399	0.9346690	1.448748
flc.grp_6	0.36917698	1.4465436	1.2131987	1.770049
flc.grp_7	0.28336217	1.3275859	1.0737082	1.611084
flc.grp_8	0.44812768	1.5653785	1.2590335	1.875078
flc.grp_9	0.49361000	1.6382195	1.3179462	1.938483
flc.grp_10	1.04961592	2.8565538	2.4267130	3.478408

TABLE 7.16: 4 Batches Analysis Results

Variable	coef	exp(coef)	lower .95	upper .95
age	1.06563116	2.9026705	2.7927321	3.010254
mgus_1	0.06714251	1.0694479	0.6956478	1.550533
sex_M	0.31436376	1.3693878	1.2599457	1.511622
flc.grp_2	-0.07006383	0.9323343	0.7597476	1.127320
flc.grp_3	0.11439481	1.1211947	0.8732736	1.336055
flc.grp_4	0.13915876	1.1493065	0.9148906	1.320807
flc.grp_5	0.13787283	1.1478296	0.9530521	1.350505
flc.grp_6	0.35593519	1.4275150	1.1363029	1.747505
flc.grp_7	0.26856977	1.3080922	1.0625509	1.492270
flc.grp_8	0.43027053	1.5376735	1.2773390	1.798540
flc.grp_9	0.48000259	1.6160786	1.3673277	1.902633
flc.grp_10	1.03950163	2.8278074	2.3423640	3.332785

TABLE 7.17: 5 Batches Analysis Results

Variable	coef	exp(coef)	lower .95	upper .95
age	1.06369798	2.8970645	2.7767077	3.045801
mgus_1	0.08405277	1.0876863	0.8105464	1.814962
sex_M	0.31504433	1.3703201	1.2493507	1.478906
flc.grp_2	-0.07936747	0.9237004	0.7228540	1.128279
flc.grp_3	0.10229832	1.1077139	0.8890106	1.286250
flc.grp_4	0.13465980	1.1441475	0.9497162	1.347431
flc.grp_5	0.13149965	1.1405375	0.8984153	1.323851
flc.grp_6	0.34786983	1.4160479	1.1711259	1.598812
flc.grp_7	0.26187698	1.2993667	1.0608931	1.458474
flc.grp_8	0.42909206	1.5358624	1.2487512	1.762339
flc.grp_9	0.47300702	1.6048126	1.3169407	1.799008
flc.grp_10	1.03139876	2.8049866	2.3461164	3.261336

Chapter 8

Discussion and Next Steps

8.1 What if the usage requirements aren't met?

Chapter 7 shows several results from the implementation being proposed in this work. However, if all the usage requirements fail, the results will probably not be what one desires.

The optimization landscape of the Cox Proportional-Hazards (PH) model's loss function is generally convex with standard fixed covariates, making it amenable to optimization techniques like gradient descent. However, various complexities introduced in survival analysis can alter this convex nature, potentially hindering the convergence of gradient-based methods. Let's proceed to enumerate and describe such complexities.

1. **Non-Proportional Hazards**: The Cox PH model operates on the key assumption of proportional hazards. When this is violated, naive model adjustments might unintentionally compromise the loss function's convexity. Also, introducing interaction terms with time or spline functions to accommodate non-proportional hazards can alter the optimization landscape. Such modifications, while aiming to capture the underlying data structure, might lead to multiple local optima or saddle points, challenging gradient-based optimization techniques [8].

2. **High Correlation Among Predictors**: Correlated predictors can cause instability during the optimization process. While the function might remain convex, the behavior of the optimization can appear as if navigating a non-convex space due to

ill-conditioning. The inclusion of highly correlated predictors can cause the Hessian of the loss function to be near singular, causing challenges in its inversion and potentially slowing convergence or causing it to diverge altogether [5].

3. **Complex Hierarchical Structures or Interactions**: Introducing random effects into the Cox model to account for unobserved heterogeneity or clustering can influence the loss function's convexity. These complexities might introduce regions in the parameter space where the function isn't strictly convex. Interaction terms, especially higher-order ones or those derived from categorical variables with many levels, can complicate the loss function, potentially leading to non-convex regions [8].

The three topics enumerated above represent the biggest issues surrounding gradient descent based methods on survival analysis.

8.2 How can Federated Learning be implemented, using the method being proposed, in a secure manner?

The title of this work is *Distributed Cox Proportional-Hazards Model: A Step Towards Federated Survival Analysis*. In this section we will delve into how federation could be implemented using the gradient descent based Cox PH Model.

As the Implementation chapter, 6, describes the method works by aggregating gradients from disjoint subsets of data. In a real world experiment, where the subsets of that represent different locations where data is stored, the communication between the central server and each party concerns only the gathering of the gradient vector. This gradient vector is computed in the local node server, which means that the data never leaves the node, and whoever is managing the central server never actually sees the data itself (obviously one could query the data for model adjustment, which does not imply actually observing the data).

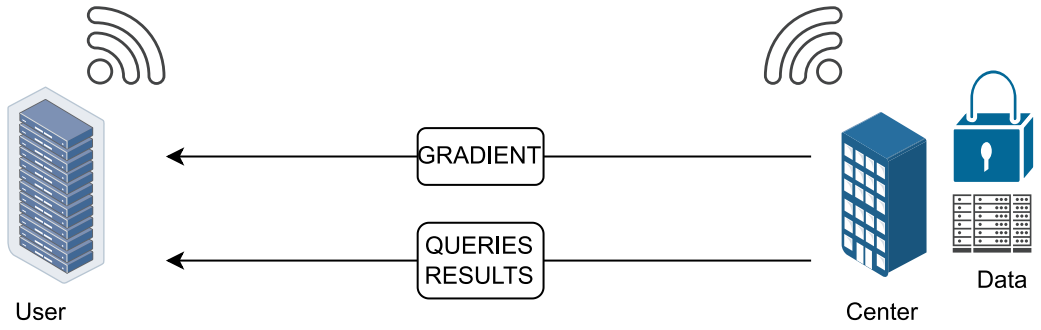


FIGURE 8.1: Secure information flow between central server's user and a data center

8.3 How does the method behave in terms of execution time?

We have now arrived to probably the biggest question about this work. Is the method efficient compared to the standard analysis? The answer is not a straight forward one.

Let's start by addressing the type of gradient descent being used. For the purpose of this work, the basic form of gradient descent was used, which led to, some times, long execution analysis times (mainly on the computation of the bootstrapping confidence intervals). One way to try to reduce execution time is to use variant or optimizer function.

Over the years, researchers have identified certain limitations and challenges inherent to the basic gradient descent algorithm. To overcome these issues and improve the efficiency of optimization tasks, several variants and optimizers of gradient descent have been developed. These adaptations are particularly useful in deep learning applications, where the complexity and size of the models often pose considerable challenges for optimization. Some of the variants are enumerated next [13].

1. **Stochastic Gradient Descent (SGD)**: While traditional gradient descent uses the whole data set to compute the gradient at each iteration (termed "batch" gradient descent), Stochastic Gradient Descent (SGD) introduces randomness into the process by selecting a single random sample from the data set to compute the gradient. SGD is particularly beneficial when dealing with large datasets as it provides faster computation and the ability to escape from local minima due to its inherent noise. However, this noise also leads to fluctuation in the loss function, which can prevent SGD from settling at the true minimum.

2. **Mini-batch Gradient Descent**: This variant strikes a balance between batch gradient descent and SGD. It performs updates for every mini-batch of n training samples, combining the benefits of both SGD and batch gradient descent. With mini-batch gradient descent, updates are less noisy than in SGD, leading to more stable convergence, while still being computationally efficient and able to escape from local minima.

3. **Momentum**: The momentum method helps to accelerate the convergence of gradient descent by adding a fraction of the update vector from the previous step to the current update vector. This approach enables the algorithm to build up velocity in any direction with persistent gradients, reducing the oscillations and leading to faster convergence. Momentum can be particularly helpful when the surface to optimize is shallow and has a high curvature.

4. **Adam**: Adam (Adaptive Moment Estimation) is an algorithm that combines the benefits of two extensions of stochastic gradient descent: Root Mean Square Propagation (RMSProp) and Adaptive Gradient Algorithm (AdaGrad). Adam calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages. The moving averages themselves are estimates of the first moment (the mean) and the second raw moment (the uncentered variance) of the gradient. Hence, Adam performs well in practice and outperforms other adaptive techniques.

5. **AmsGrad**: AmsGrad is an optimizer introduced to improve the original Adam optimizer. It uses the maximum of past squared gradients rather than their exponential average to update parameters. This modification ensures the learning rate does not increase after it has decreased once, addressing the issue of convergence in Adam.

6. **RMSprop**: RMSprop (Root Mean Square Propagation) scales the learning rate by dividing it by an exponentially decaying average of squared gradients. This has the effect of damping the gradient descent update, effectively giving us a learning rate that decreases faster for steep dimensions than for dimensions with gentler slopes. This can be particularly useful in situations where the optimal solution involves a gentle slope in some dimensions and steep slopes in others.

7. **Adagrad**: Adagrad (Adaptive Gradient Algorithm) adaptively scales the learning rate with respect to the accumulated square root of gradients. Each parameter has

its own learning rate, which helps when dealing with sparse data. It is especially beneficial for convex problems but can sometimes prematurely decrease the learning rate when applied to deep learning tasks, leading to poor convergence.

Adding an optimizer function to the algorithm is the next big step, or at least the most important one. The choosing of which one to use would probably be highly influenced by the *Big survival data analysis via stochastic gradient descent* [10] paper, where it is shown that the AmsGRAD optimizer works best in this particular kind of studies.

We will now move on to address some of the weaknesses of Federated Learning that could potentially be a problem when implementing the strategy we purpose.

8.4 Challenges and Limitations of Federated Learning

While federated learning provides numerous advantages in the realm of data privacy and decentralized machine learning, it also comes with its own set of challenges and limitations that must be taken into account.

1. **Data Skewness**: In a federated learning setting, data can be distributed unevenly across different nodes, leading to a phenomenon called non-identically distributed (non-IID) data. This skewness can cause significant challenges in model training and aggregation, potentially leading to models that are biased towards nodes with more data.

2. **Communication Overhead**: Communication between a central server and local nodes is integral to federated learning. However, it can also be resource-intensive, particularly in terms of bandwidth usage. Ensuring efficient communication without sacrificing model accuracy is a major challenge.

3. **System Heterogeneity**: Local nodes in a federated learning system can vary greatly in terms of computational power and availability. Some nodes might be powerful servers, while others might be mobile devices with limited resources. Designing federated learning algorithms that are robust to such heterogeneity is a challenging task.

4. **Privacy and Security Concerns**: While federated learning inherently improves data privacy, it is not without its security concerns. During the model aggregation phase,

sensitive information can potentially be leaked. Moreover, malicious nodes could potentially introduce corrupted updates, leading to compromised global models.

5. **Scalability**: Federated learning systems can potentially involve millions of nodes. Designing systems that can scale to this extent, while still maintaining performance and privacy, is a significant challenge.

Despite these challenges, federated learning holds immense promise for the future of machine learning, particularly in applications where privacy is paramount. With continued research and development, the field is set to overcome these challenges and unlock the full potential of federated learning [14].

Appendix A

Manipulation of the partial-likelihood function

$$l(\beta|D^{(n)}) = \sum_{i=1}^{n} log\left(\frac{h(x^{(i)};\beta)}{\sum_{j\in R_i} h(x^{(j)};\beta)}\right)$$

$$l(\beta|D^{(n)}) = \sum_{i=1}^{n}\left(log(h(x^{(i)};\beta)) - log\left(\sum_{j\in R_i} h(x^{(j)};\beta)\right)\right)$$

$$l(\beta|D^{(n)}) = \sum_{i=1}^{n}\left(log\left(h_0(t)e^{f_\beta(x^{(i)})}\right) - log\left(\sum_{j\in R_i} h_0(t)e^{f_\beta(x^{(j)})}\right)\right)$$

$$l(\beta|D^{(n)}) = \sum_{i=1}^{n}\left(log(h_0(t)) + log(e^{f_\beta(x^{(i)})}) - log(h_0(t)) - log\left(\sum_{j\in R_i} e^{f_\beta(x^{(j)})}\right)\right)$$

$$l(\beta|D^{(n)}) = \sum_{i=1}^{n}\left(f_\beta(x^{(i)}) - log\left(\sum_{j\in R_i} e^{f_\beta(x^{(j)})}\right)\right)$$

Appendix B

Gradient calculation of the log-partial-likelihood function

$$\nabla_\beta\{-l(\beta|D^{(n)})\} = \nabla_\beta\left\{-\sum_{i=1}^{n}\left(f_\beta(x^{(i)}) - log\left(\sum_{j\in R_i} e^{f_\beta(x^{(j)})}\right)\right)\right\}$$

$$\nabla_\beta\{-l(\beta|D^{(n)})\} = -\sum_{i=1}^{n}\left(\nabla_\beta\{f_\beta(x^{(i)})\} - \nabla_\beta\left\{log\left(\sum_{j\in R_i} e^{f_\beta(x^{(j)})}\right)\right\}\right)$$

$$\nabla_\beta\{-l(\beta|D^{(n)})\} = -\sum_{i=1}^{n}\left(\nabla_\beta\{f_\beta(x^{(i)})\} - \frac{\sum_{j\in R_i}\nabla_\beta\{f_\beta(x^{(j)})\}e^{f_\beta(x^{(j)})}}{\sum_{j\in R_i} e^{f_\beta(x^{(j)})}}\right)$$

Bibliography

[1] J. D. Kalbfleisch and R. L. Prentice, *The statistical analysis of failure time data*. John Wiley & Sons, 2011. [Cited on page 4.]

[2] D. G. Kleinbaum and M. Klein, *Survival analysis a self-learning text*. Springer, 1996. [Cited on pages 4, 5, 6, 7, 8, and 10.]

[3] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*. John Wiley & Sons, 2013, vol. 398. [Cited on page 8.]

[4] I. Dabbura, "Gradient descent algorithm and its variants," *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3 [Cited on page 15.]

[5] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999. [Cited on pages 15 and 58.]

[6] B. Efron, *Bootstrap methods: another look at the jackknife*. Institute of Mathematical Statistics, 1979. [Cited on page 19.]

[7] A. C. Davison and D. V. Hinkley, *Bootstrap methods and their application*. Cambridge university press, 1997, no. 1. [Cited on page 19.]

[8] T. M. Therneau, P. M. Grambsch, T. M. Therneau, and P. M. Grambsch, *The cox model*. Springer, 2000. [Cited on pages 21, 57, and 58.]

[9] Y. Fang, J. Xu, and L. Yang, "Online bootstrap confidence intervals for the stochastic gradient descent estimator," *Journal of Machine Learning Research*, 2018. [Cited on page 21.]

[10] A. Tarkhan and N. Simon, "Bigsurvsgd: Big survival data analysis via stochastic gradient descent," *arXiv preprint arXiv:2003.00116*, 2020. [Cited on pages 21 and 61.]

[11] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2017. [Cited on page 23.]

[12] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016. [Cited on page 24.]

[13] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016. [Cited on page 59.]

[14] S. Banabilah, M. Aloqaily, E. Alsayed, N. Malik, and Y. Jararweh, "Federated learning review: Fundamentals, enabling technologies, and future applications," *Information processing & management*, vol. 59, no. 6, p. 103061, 2022. [Cited on page 62.]