NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

JOSÉ TOMÁS SILVEIRA

Master in Civil Engineering

MODELLING CUSTOMER CHURN WITH TIME-TO-EVENT APPROACHES

MASTER IN ANALYSIS AND ENGINEERING OF BIG DATA

NOVA University Lisbon
December, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

MODELLING CUSTOMER CHURN WITH TIME-TO-EVENT APPROACHES

JOSÉ TOMÁS SILVEIRA

Master in Civil Engineering

Adviser: Cláudia Alexandra Magalhães Soares
Assistant Professor, NOVA University Lisbon

Co-adviser: João Varela
Data Scientist, NOS

MASTER IN ANALYSIS AND ENGINEERING OF BIG DATA

NOVA University Lisbon
December, 2022

Modelling customer churn with time-to-event approaches

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.6.7) [25].

Para a minha família, Sophie e amigos.
Por me acompanharem durante este percurso, e tornarem
realidade a sua conclusão. Gosto muito de vocês.

Acknowledgements

This dissertation was only possible thanks to the support of a number of people, to whom I owe my most sincere thanks. Firstly, I want to express my gratitude to my advisors, Professor Claudia Soares and Joao Varela, for the invaluable knowledge transmitted and for the availability and patience with me. This work would also not have been possible without the help of my defence committee, Professor Ludwig Krippahl and Professor Paula Amaral, who generously provided knowledge and expertise. To PhD. Ana Guedes, I am grateful for the help and support during the first phase of this dissertation. I would also like to thank my colleagues at NOS, Nuno Paiva, Francisco Delgado and Gabriel Dias, for their help and feedback throughout this journey. Additionally, I want to thank NOS SGPS, who generously provided the necessary infrastructure to make this dissertation possible.

I also want to thank my family for all the support, motivation and confidence given to me, even when my path wasn't clearly defined. I want to express my gratitude to Sophie for accompanying me on this journey and for the patience and support given when hard times eventually came. Lastly, I want to thank my friends, Duarte, Rafa, Manuel, Andre, Francisco and Luis, for all the friendship and support over the years and for making this journey much easier.

"Sometimes you lie in bed at night and you don't have a single thing to worry about. That always worries me!" (Charlie Brown)

Abstract

Overtime, companies have shown an increased interest in enhancing their relationship with existing clients, due to the high cost of attracting new customers. In lieu of this, organizations started to adopt more proactive strategies in order to keep existing clients. Modern approaches include the creation of predictive models to determine which clients are more at risk of churning, and also which factors contributed the most for that decision.

Most churn predictive models used in industry comprise of binary classifiers, capable of identifying potential churners in discrete time-windows. One could use classical regression techniques to get continuous predictions, however, they do not work so well when dealing with censored observations. In this dissertation, we use a framework that generates continuous predictions whilst allowing censored observations by using so called time-to-event models, or survival analysis models. Time-to-event models allow for continuous churn predictions, thus obviating the need to train a new model for each time-window, which by itself is already an advantage over classical models. However, the main point is that survival analysis allows for a richer understanding on how churn risk varies over time, which is essential for Customer relationship management (CRM) to create better strategies at retaining clients. Since churn datasets have substantial class imbalance, appropriate techniques were researched and used to deal with this problem. An experimental pipeline was developed to run survival analysis experiments, and each step thoroughly evaluated using the appropriate metrics and discussed. In the end, a case-study is presented as a means to show the potential of survival modeling regarding churn prediction and prevention, and also present several ways to leverage these models in order to generate Key Performance Insights (KPI) about customer churn.

Keywords: Customer churn, Survival analysis, Machine learning, Ensemble methods, Class imbalance

Resumo

Com o passar do tempo, as organizações empresariais têm demonstrado um maior interesse em aprimorar o relacionamento com os seus clientes, particularmente devido ao elevado custo associado à aquisição de novos consumidores. Devido a estes custos, as empresas começaram a adotar estratégias proativas de retenção de clientes. Abordagens modernas incluem a criação de modelos preditivos para determinar quais os clientes mais em risco de fazer churn, e também quais os fatores que mais contribuíram para essa decisão.

No que toca ao churn, a maior parte dos modelos preditivos usados na industria sao compostos por classificadores binários, capazes de identificar potenciais churners em janelas de tempo discretas. É possível usar técnicas clássicas de regressão para obter previsões contínuas, no entanto, estas não são compatíveis com observações censuradas. Nesta dissertação, vamos usar uma abordagem que não só tem a capacidade de gerar previsões contínuas, como também de permitir observações censuradas, através de um grupo de modelos denominados por time-to-event, ou modelos de análise de sobrevivência. A maior vantagem deste tipo de modelos passa por permitir uma compreensão mais rica da variação do risco de churn ao longo do tempo. Esta informação é essencial para o departamento de gestão de relacionamento com o cliente (CRM), que assim consegue criar melhores estratégias de retenção de clientes. Como as bases de dados relativas a churn têm um desequilíbrio entre classes substancial, foram também estudadas quais as técnicas apropriadas para lidar com este problema. Um processo experimental foi desenvolvido de forma a correr experiências relacionadas com análise de sobrevivência, e cada passo detalhadamente discutido e avaliado utilizando métricas apropriadas. No final, um caso-de-estudo é apresentado como meio de mostrar os potenciais contributos que um modelo de sobrevivência traz no contexto de previsão e prevenção de churn.

Palavras-chave: Churn de clients, Análise de sobrevivência, Aprendizagem automática, Métodos de agrupamento, Desequilíbrio de classes

Contents

List of Figures

List of Tables

List of Listings

Glossary

censored An observation whose we do not know the true survival time. 7, 13, 14, 16, 30, 65

Censoring Happens when we do not know the true survival time for an observation. There are three types of censoring. The one we are interested, right censoring, occurs when an observation does not experience an event before the end of the survival study. 7, 8, 11, 12, 14, 16

churn A metric that represents the number of customers that have stopped using a product or service during a given period of time. vi, 1, 2, 3, 4, 5, 7, 9, 11, 12, 18, 19, 20, 21, 22, 23, 26, 29, 30, 31, 36, 51, 55, 56, 57, 58, 62, 63, 64, 65, 66, 72

churn-dur The time value, in months, when the customer churns. xvi, 28, 29, 30, 32, 41, 45

client-dur The time value, in months, since the customer signs a contract with NOS. xi, 28, 29, 30, 40, 46, 57, 58

duration The time difference between the beginning of an observation till an event happens, or until the survival study is over. xvi, xvii, 11, 14, 29, 30, 55, 66

end-dur The end time value, in months, of the survival study duration. 29, 30, 43, 44, 45, 47, 48, 55, 59

event An experience of interest, such as death, disease occurrence or, in our case, a client leaving a service. 9, 11, 12, 13, 14, 18, 26, 29, 30, 31, 32, 41, 44, 47, 48, 52, 55, 56, 60, 63, 65, 66

granularity Aggregates churn-dur values into time bins (e.g. daily or weekly bins). xiii, 30, 33, 40, 44, 45, 46, 48, 55

Hazard function A function that represents the potential risk of having an event given that the observation has survived up to time t. 12, 13

init-dur The start time value, in months, or feature of the survival study duration. 30, 39, 40, 41, 46, 47, 48, 55

lift-dur The time value, in months, to consider a customer churned (for the computation of the lift and gain metrics). x, xi, 31, 32, 33, 36, 37, 39, 40, 41, 43, 44, 45, 46, 47, 48, 50, 51, 55, 58, 59

pf-dur The time value, in months, until the customer ends the binding period. Negative values indicate a customer that has already ended the binding period. xi, 28, 29, 39, 46, 57, 58, 61, 62

Survival function A function that gives the probability of an observation surviving longer than some specific time t. 12, 13

Acronyms

AdaBoost Adaptive Boosting 4
AUC Area under the ROC Curve 6

BP Binding period 1

CHF Cumulative Hazard function 4, 12, 13, 15, 16
CNN Condensed Nearest Neighbors 9
CPH Cox Proportional Hazards 4, 5, 8, 13, 50, 52, 53, 55
CRM Customer relationship management vi, 4, 7

DBSCAN Density-based spatial clustering 6
DBSMOTE Density-Based SMOTE 6
DECO The Portuguese Association for Consumer Protection 1
DOF Degrees of Freedom xiii, 26, 27, 31, 32, 33, 35, 39, 43, 45, 46, 48, 55, 66
DT Decision Trees 4, 6, 15

GAN Generative Adversarial Network 6
GBoost Gradient Boost 16, 17, 31, 50, 53
GBSurv Gradient Boosting for Survival Analysis 17, 50, 51, 53, 55

IBS Integrated Brier-score 8, 9

KM Kaplan-Meier 3, 4, 8, 12, 13, 57
KPI Key Performance Insights vi

NA Nelson-Aalen 3, 4, 12, 16

OOB Out-Of-Bag 16

PH Proportional Hazards 5, 13

ProWSyn Proximity Weighted Synthetic Oversampling 6

RF Random Forests 4, 15

RMSE Root of the Mean Squared Error 7

ROC Receiver operating characteristic period 3

ROS Random Oversampling x, 6, 9, 10, 36, 38, 39, 40

RSF Random Survival Forest xi, 4, 5, 15, 50, 51, 53, 55, 59, 60, 61

RUS Random Undersampling x, 5, 6, 9, 10, 36, 38, 39, 40

SMOTE Synthetic Minority Over-Sampling Technique x, 6, 10, 11, 29

telecom Telecommunications 1, 2, 66

XGBoost eXtreme Gradient Boosting 4

1 Introduction

1.1 Problem

Customer churn is defined as an event in which a customer leaves a service. This is a major problem in service oriented companies, as losing a client is substantially more costly than retaining or getting new ones [39]. In the case of Europe and the US, churn costs Telecommunications (telecom) operators more than US$4 billion per year [18]. Furthermore, clients with high churn rates (more than 30% per year), do not have any financial return for new customers, since it takes, on average, three years to recover the acquisition cost.

Churn can be divided into two categories: voluntary and involuntary (most commonly known in the literature as churn by payment default). In voluntary churn, a client formally leaves the company by terminating his/her contract, whereas in involuntary churn, the company has the final decision in terminating the contract, usually because of continuous payment failure. The main focus of this work was in voluntary churn. The decision to leave a company can be attributed to many factors such as bad or delayed customer support, poor service quality or competition offering better deals. Furthermore, this problem is exacerbated when a customer shares a bad experience with relatives or friends. This is verifiable by looking at the rise in complaints telecom companies receive. In Portugal, The Portuguese Association for Consumer Protection (DECO), has reported that telecom complaints have risen 30% in 2020, and are at the top of complaint rankings for 13 years [40]. Another important factor that can have a negative impact on churn rates is changes in the Binding period (BP) terms by the Portuguese government [38]. By changing BP terms (e.g. new customers not having to comply with a loyalty period) churn rates are expected to increase due to a generalized rise in service price (including entry costs), worse service quality and reduced mobility between customers and the service providers.

1.2 Motivation

The problem of customer churn has a major negative impact on a company's revenue. This impact can be attributed to direct and indirect costs when a client churns. Direct costs include loss of recurring revenue, that is, the company will stop receiving the client's annual/monthly subscription fees. Indirect costs include the cost to acquire new clients, which are usually much more expensive than retaining existing ones [39], and also the loss of expansion opportunity revenue, that is, the opportunity that a company has to upsell an existing customer. In [39], the author gives some important facts about the costs of churning: 1) It costs five times more to acquire a new costumer than it does to keep a current one; 2) It costs sixteen times more to bring a new costumer up to the same level as a current one. Furthermore, as per [39], a small 2% increase in customer retention can lower cost by as much as 10 percent, and lowering customer churn rate by five percent can increase a company's profitability by 25 to 125 percent. For the aforementioned reasons, it is crucial for companies to be proactive by actively identifying potential churners and creating effective retaining campaigns. Knowing the reason(s) that lead a customer to be a potential churner can give the company powerful insights on what is lacking in their service, thus allowing for targeted improvement. This and being able to effectively predict a customer's churn risk over time, is the reason why time-to-event models were chosen over classifiers as the modeling framework in this dissertation.

1.3 Objectives and Strategy

The objective in this work is twofold: First, to identify potential churners and when they are most likely to churn; Second, to gain additional insights on the main reasons that lead a customer to churn. By leveraging this information, the company can adequately allocate resources to retain these customers, and also to improve their own service, which, in return, will translate into more profits.

The work will be done with NOS, a leading Portuguese telecom and media company that provides phone, television and internet services. NOS currently uses a binary classifier to predict whether a client churns or not in a pre-determined time window. Even though this model has proven to be useful in NOS context, it is unable to predict time-to-event, which means that the model needs to be retrained for each time window. Linear regression can be used to predict the time-to-event, however, as the large majority of customers have not churned yet, they cannot be used in the model, resulting in huge losses of data. We will go one step further, and use a group of models that allow customers that have not churned yet whilst being able to predict their churn risk over time, by using survival models. With a survival model, we look to make improvements on NOS current approach, not only by offering richer predictions, such as customer's risk over time, but also by solving the problem of having to use multiple classification models to analyze different time windows.

2

State of the Art

Since customer churn has a big impact on a company's total revenue, there has been a lot of effort made, both in the industry and in academia, in accurately predicting potential churners and understanding what led them to take that decision. The first survival models were based on statistical methods such as the Kaplan-Meier (KM) estimator for predicting survival curves and Nelson-Aalen (NA) estimator for hazard ratios. With advancements in computing power, machine learning and advanced survival models have been increasingly researched and used in the context of churn prediction.

2.1 Metrics

Accuracy is one of the most popular metrics when it comes to evaluating the performance of a model. However, since churn prediction datasets are highly imbalanced for the minority class, the use of accuracy is not advisable. For example, in a dataset where we have 10 churners and 990 non churners, if we classify every observation as non-churned, we will have 99% accuracy, which would be misleading. For this reason, the metrics used should not depend on the number of true negatives. Furthermore, metrics should be appropriate in the context of survival analysis, since that is the class of models being implemented in this dissertation.

As seen in most survival analysis papers, the concordance index (C-score) [32] is almost always used as an evaluation metric, mainly due to its interpetability [27]. C-score has the advantage of being similar to accuracy and Receiver operating characteristic period (ROC), but without suffering from the same data imbalance problem, since it does not depend on False Negatives. Furthermore, C-score only depends on the rank order of the predictions, making it a very useful metric for evaluating proportional hazard models. Another popular metric in the context of survival analysis is the Brier-score, which not only measures the discriminative power of a survival model, but is also used to calibrate[1] it [10]. For the comparison of survival curves, the log-rank test is usually the preferred

[1] Calibration performance measures the similarity between the observed probabilities and the ones predicted by the model, whilst descriminative performance refers to the ability of the model predicting the right order of events [27]

statistical test. In survival trees, nodes are usually split using the log-rank statistic as a dissimilarity measure [19]. And finally, the lift and gain evaluation metric [33] measures how well a model performs when compared to random guessing. Even though these last two metrics are not usually used in the context of survival analysis, we will include them due to its utility in CRM retention strategies.

2.2 Models

Several types of churn prediction models have been researched and developed along the years. Most of the models used in industry fall under the domain of binary classifiers, where the objective is to predict whether a client will churn in a pre-determined time-window. Regression models are seldom used because of censoring, and since most people are censored, a substantial percentage of the data would be excluded from the analysis. Other than regression, time-to-event models allow for both continuous survival prediction and censoring. This section will give an overview of some of the most commonly used models in this domain.

Regarding classification models, the most commonly found in the literature for churn prediction are tree-based algorithms, such as Decision Trees (DT), and its related ensemble methods (Random Forests (RF) and eXtreme Gradient Boosting (XGBoost)). Even though DT have some desirable properties in the context of churn prediction such as easy interpetability, they usually fall short when compared with other methods because DT are very prone to overfit, and thus failing to generalize for data outside of the training set [13]. RF were developed to tackle this problem [2], by using an ensemble of DT instead of just one. By averaging over several DT, and combining random bootstrap sampling and random feature selection in each tree, RF are able to significantly reduce overfitting and thus improve predictive performance [2]. Boosting methods are another class of ensemble methods that are widely used in churn prediction problems, mainly Adaptive Boosting (AdaBoost) and XGBoost. The latter is currently being used in NOS's voluntary churn prediction pipeline.

Even though classification models are able to produce satisfactory results, they are limited to discrete predictions, meaning that they need to be trained for each desired time-window. Time-to-Event models solve this problem whilst allowing customers that have not churned yet, making it an interesting approach to take in the context of churn prediction. This interest is proven by the increasing number of published papers using survival models to predict churn risk, including [26, 14]. Besides univariate methods such as the KM and NA estimators, the most used survival method is the Cox Proportional Hazards (CPH) model [30]. This methods uses a linear combination of features with a baseline hazard function to make estimates of the Cumulative Hazard function (CHF). However, CPH has several limitations, namely its proportional hazards assumption and its inability to deal with non-linear relationships between covariates. Alternative survival models such as RSF are better suited to handle these non-linear relationships and, at the

same time, not depend on the Proportional Hazards (PH) assumption [16]. Due to its flexibility and predictive performance, RSF are usually used as a benchmark model in state of the art survival methods.

In [17], the authors improve on the linear CPH model by replacing the relative risk function with a neural network architecture, denoted by DeepSurv. This solution was shown to have better performance in regards to the C-score, in both linear and non-linear experiments, when compared to CPH and RSF. Another solution, denoted by DeepHit [23], uses a neural network to estimate the probabillity mass function of the survival time and event. DeepHit was shown to perform better than CPH, RSF and DeepSurv with regards to the C-score. The authors in [22] propose a neural network model that uses a novel loss function that has the advantage of being scalable to big datasets, which enables the fitting of both a PH extension of the Cox model, and a non-proportional extension. Measuring the model's performance with regards to the C-score and the Brier-score, the non proportional extension has better performance in almost all the tests performed by the authors when compared to other state of the art models, such as RSF, DeepSurv and DeepHit. A Recurrent Neural Network survival solution denoted by RNN-Surv was proposed in [11], and has the advantage of being able to leverage relationships between variables in different timesteps, which can be interesting in the context of churn prediction. The authors show that RNN-Surv performs better with regards to the C-score when compared to CPH, DeepSurv and RSF.

2.3 Dealing with Class imbalance

In [35], the author suggests six classes of problems that arise when when working with imbalanced data, such as improper evaluation metrics, lack of data, relative lack of data, data fragmentation, inappropriate inductive bias and noise. Approaches to deal with this range from using appropriate metrics (as seen in Section 2.1) to sampling techniques and cost sensitive learning. Another important aspect is how to deal with noisy data, which tends to negatively affect how the model performs. Interestingly, this effect is greater in instances from the minority class [5]. Noisy data can be dealt with more advanced sampling methods that are able to selectively remove/add examples, or combine under-sampling with over-sampling such as CUBE [31].

In Cost-sensitive learning, the cost of misclassifications is taken into consideration, treating different misclassifications differently [5]. Cost-sensitive learning can be divided into two categories, the first one being to design classifiers that are cost-sensitive by nature, that is, the cost is introduced in the error function of the learning algorithms. The second one is converting cost-insensitive classifiers into cost-sensitive ones, without modifying them, for example, by introducing weights in certain instances and using those weights in the total classification error.

Regarding sampling techniques, let us start with a naive approach named random sampling. In RUS, randomly sampled observations are taken out of the dataset; the idea is

that since many observations from the majority class are redundant, then removing them at random will not affect the variable distribution too much. In ROS, the minority class is randomly sampled with replacement until both classes have the same frequency, or a pre-defined ratio. However, since examples from the minority class are being repeated, the risk of overfitting increases. Tomek links [15] improve on RUS by creating a rule to find pairs of instances considered to be noisy or overlapping, thus making them more suitable to be removed.

SMOTE, improves on ROS by making it possible to generate new synthetic examples that are close, in feature space, with observed instances. In [6], the authors show a clear improvement in performance when using SMOTE instead of ROS. This is most likely due to random oversampling's replication process, which yields smaller and more specific decision regions. However, SMOTE does not consider the majority class observations when generating new ones, which results in overgeneralized examples if there is a strong overlap between classes. Safe-Level-SMOTE [4] improves on SMOTE by assigning each positive (minority class) observation a weight, called safe-level, before generating new examples. The objective is to position each new synthetic instance closer to the largest safe-level, resulting in instances that are created mostly in safe-regions. The authors show that Safe-Level-SMOTE outperforms SMOTE in regards to precision and F-value, when the new synthetic examples are applied in DT. Density-Based SMOTE (DBSMOTE) [3], is another technique that improves on SMOTE, but this time by using the concept of Density-based spatial clustering (DBSCAN) to generate minority class clusters of arbitrarily defined size. DBSMOTE then uses the shortest path between each minority class observation to the center of mass of the clusters to generate new synthetic examples. The experimental results in [3] show that, for unbalanced datasets, DBSMOTE performs better than SMOTE and Safe-Level-SMOTE when it comes to accuracy, F-measure and Area under the ROC Curve (AUC). Another technique that improves on SMOTE is Proximity Weighted Synthetic Oversampling (ProWSyn) [1]. This method also adds weights to the minority class examples, but in this case, by a ranked proximity level, where lower proximity instances have an higher weight. In [20], the author shows that, among state-of-the-art oversampling techniques, that ProWSyn is one of the best performant techniques.

An interesting class imbalance technique is the application of conditional Generative Adversarial Network (GAN) to generate new synthetic examples [8]. By using this technique, the authors showed that after the network training is complete, their method has superior performance when compared with other state of the art oversampling techniques, such as SMOTE, ADASYN and Borderline SMOTE. In [36] the author states that, combining an undersampling technique with an oversampling technique, can improve the performance of the predictive models when compared to either model alone. In this case, the oversampler SMOTE was combined with the undersampler Tomek-Links.

3

Methodology

In this chapter we present the models and techniques that are of most interest for the strategy defined in Section 1.3. We will also provide a description of the model used as benchmark. Most of these topics were already presented in the literature review presented in Section 2.

3.1 Metrics

The evaluation of churn prediction models is treated in a different way when compared to other models. The main reason being the disproportional amount of non-churners when compared to churners. This class imbalance makes metrics such as accuracy not suitable for the churn problem. Furthermore, the presence of censored events in the data makes standard evaluation metric, such as the Root of the Mean Squared Error (RMSE) and R^2 not suitable in the context of survival analysis. There exist several specialized evaluation metrics of survival analysis such as the Concordance index, or C-score, and the Brier-score. Other metric suitable in the context of the churn problem, and often used in CRM departments, is the lift and gain charts.

Concordance Index (C-score): Unlike accuracy metrics, the C-score evaluates how well the rank order of the predictions match the true outcomes, and is defined as the ratio between concordant pairs and the total number of comparable pairs [32]. Let (i, j) be a comparable instance pair, t_i and t_j the respective observed times and $s(t_i)$, and $s(t_j)$ the predicted survival times. A pair is either concordant or discordant if,

$$\begin{cases} t_i > t_j \ \ \& \ \ s(t_i) > s(t_j) & \text{,pair (i,j) is concordant} \\ t_i > t_j \ \ \& \ \ s(t_i) < s(t_j) & \text{,pair (i,j) is discordant} \end{cases}$$

It is important to note that not all instances are comparable, that is, a pair (i, j) is comparable only if both instances are uncensored, or if the observed event time of the uncensored instance is smaller than the Censoring time of the censored instance [34]. The C-score

is computed in different ways depending on whether we have or not predicted survival times. In the former case, the C-score is given by:

$$c = \frac{1}{n} \sum_{i:\delta_i=1} \sum_{j:t_i<t_j} I\left(s(\hat{t}_i) < s(\hat{t}_j)\right) \tag{3.1}$$

where I is the indicator function (1 if the argument is true, 0 otherwise), $s(\hat{t}_i)$ is the predicted survival probability of instance i, and n is the total number of comparable pairs. In the case that the output of the model is an hazard ratio, such as in the case of the CPH model, then the C-score is given by:

$$c = \frac{1}{n} \sum_{i:\delta_i=1} \sum_{j:t_i<t_j} I\left(x_i\hat{\beta} > x_j\hat{\beta}\right) \tag{3.2}$$

where $\hat{\beta}$ is the set of estimated parameters from the CPH model. The c-index ranges from 0 to 1, being 1 a perfectly ordered set of predictions and 0.5 random guessing.

Brier-score: Is a method used to measure the overall performance of a model. In the case of survival analysis, the Brier-score is used to measure the accuracy of the predicted survival function at a given time t. Let $y_i(t)$ be 0 if the observation is a non-event at time t and 1 if it is an event, and $\hat{y}_i(t)$ be the predicted survival probability, then the Brier-score is given by:

$$BS(t) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i(t) - y_i(t))^2 \tag{3.3}$$

where n is the total number of observations. By looking at (3.3), we can intuitively say that a model has better performance if its score is closer to 0. We can also deduce that for a random model, that is, a model that always predicts 0.5 for all instances, results in a Brier-score of 0.25. Nevertheless, the above expression only works for datasets that do not contain censored observations. In order to allow for Censoring, it is necessary to weigh each squared distance:

$$wBS(t) = \frac{1}{n} \sum_{i=1}^{n} w_i(\hat{y}_i(t) - y_i(t))^2 \tag{3.4}$$

the weights w_i can be estimated by several methods [34]. In the case of the Brier-score, we compute the weights based on the censored distribution G, which can be estimated with the KM estimator:

$$w_i(t) \begin{cases} \delta_i/G(y_i) & \text{,if } y_i \leq t \\ 1/G(y_i) & \text{,if } y_i > t \end{cases} \tag{3.5}$$

where δ_i is the binary indicator for censored observations ($\delta_i = 1$ when an instance is non-censored, and 0 otherwise). Additionally, we can also compute the Integrated Brier-score

(IBS) [12], which provides an overall calculation of the model's performance within event time points $t_{min} \leq t \leq t_{max}$. The IBS is defined as:

$$\text{IBS} = \int_{t_{min}}^{t_{max}} BS^c(t) dw(t) \tag{3.6}$$

where $w(t) = t/t_{max}$ represents the weighting function. The integral is approximated using the trapezoidal rule.

Lift and Gain: Is an evaluation metric that measures how well a predictive model is able to correctly classify observations against random guessing. Lift compares the true ratio of churn and the prediction churn ratio, meaning that a good model should have a high correlation between both ratios. The basic idea is to sort the population by percentage of churn, create equally spaced subsections, such as deciles, and then compute the true ratio of people that churned in each group and compare it to the respective predicted survival ratio [5]. The Lift score can be obtained by:

$$Lift = \frac{Precision}{P/(P+N)} \tag{3.7}$$

where $P = TP + FN$, $N = TN + FP$ and $Precision = \frac{TP}{TP+FP}$. After this, we can plot the lift scores for each subsection. We can also use the accumulated lift scores to plot the gains chart, and then compare it against a random guessing model, or baseline model.In a gains chart, the greater the distance between both curves, the better the model.

3.2 Dealing with class imbalance

As churn rates are usually very low, the ratio between the churned and non-churned classes is also very low, resulting in an imbalanced dataset. From the methods discussed in Chapter 2, five potential candidates were chosen to handle class imbalance in NOS voluntary churn dataset.

Random sampling: One of the simplest strategies to handle imbalanced data is random sampling. Random sampling can take two forms: ROS, where random observations from the minority class are added to the training data; and RUS, where random observations from the majority class are removed from the training dataset. Both approaches are repeated till a desired class ratio is achieved. It is important to note that this approach is only applicable to the training data, as the objective is to improve the fit of a model. When evaluating the model, the test-data should remain in its original size. Figure 3.1 shows an illustration of the RUS and ROS techniques.

Tomek-Links: Is a popular undersampling method that originated from the Condensed Nearest Neighbors (CNN) method. Unlike CNN, where examples are removed randomly,

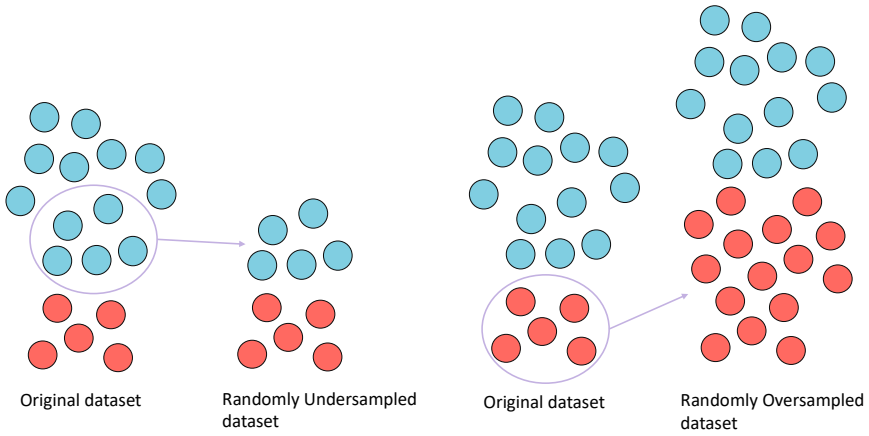


Figure 3.1: Class Imbalance: RUS and ROS illustrations.

Tomek-links uses a rule to find pairs of instances, one from each class, such that they have the smallest Euclidean distance between each other in feature space. Let a and b be two instances from different classes in a binary classification problem. These instances define a Tomek link if:

1. The nearest neighbor of instance a is b
2. The nearest neighbor of instance b is a

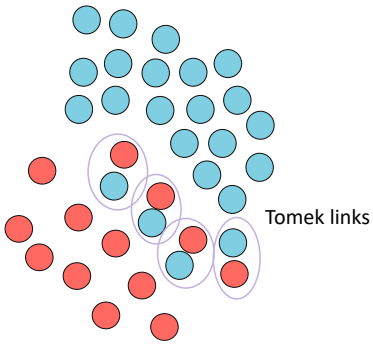


Figure 3.2: Class Imbalance: Tomek-links illustration.

These pairs are valuable as they can either define a class boundary or noise [15]. By using Tomek-links we can clean up any overlap between classes by removing either only the majority class instance from the pair, or both instances. This should lead to a better classification model performance. This model is also often used in combination with an oversampler, such as SMOTE, to improve model performance.

SMOTE: Unlike ROS, SMOTE approach is to generate new synthetic examples that are similar to the minority class examples [6]. Simply put, SMOTE generates new examples along line segments that join k nearest neighbors of the minority class. The algorithm

works as follows: First we choose a random example from the minority class and compute the difference between the sample's feature vector and its nearest neighbor. Second, we generate a random number between 0 and 1, multiply it by the previously computed difference, and add this value to the chosen sample. By doing this, we select a random point along the line between the two minority class examples, effectively generating a new synthetic examples. This process can be repeated many times along the lines between the k nearest neighbors. Figure 3.3 illustrates the process:

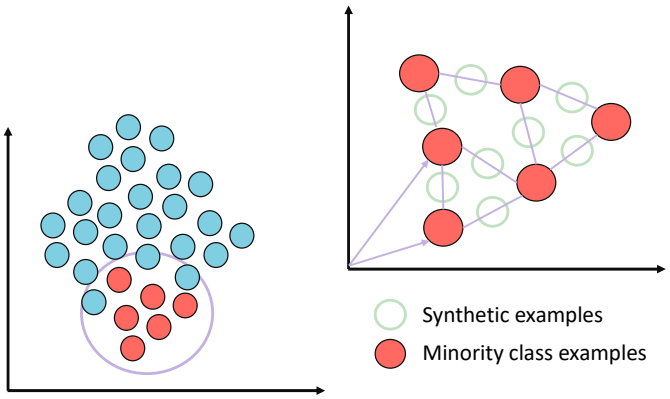


Figure 3.3: Class Imbalance: SMOTE illustration.

3.3 Survival Analysis

Survival analysis is a set of statistical methods where the outcome variable of interest is the time it takes until an event happens. Unlike regression models, in survival analysis we can consider observations with events that have not happened yet to be included in the analysis. This is called Censoring. In this section we will begin by presenting the necessary survival analysis concepts that are needed to understand the techniques presented next.

3.3.1 Fundamental concepts

We will begin by defining what is an event. An event is an experience of interest, such as death, disease occurrence or, in our case, a client leaving a service. duration is the time difference between the beginning of an observation till an event happens, or until the survival study is over [19]. In the case of churn prediction, duration can be defined as the time from when a customer signs a contract (or any other date) until he leaves (churns) or the survival study period is over. It is important to note that some companies, including NOS, have a binding period when a customer first signs a contract that forbids the client from leaving the service. Whether we should include or not this period in our analysis is up to debate, and will be something that will be analyzed in the next sections. Censoring happens when we do not know the true survival time for an observation. There are three

types of Censoring, however, in this work we will only focus on right Censoring since that is the only one applicable in churn prediction. Right Censoring occurs when an observation does not experience an event before the end of the study [19]. In the case of NOS, Censoring happens when the customer has not churned when the survival study is over.

Apart from the concepts presented above, there are two curves that are of most interest in survival analysis. The first one is called the Survival function, $S(t)$, and is a function that gives the probability of an observation surviving longer than some specific time t [19]. Let T be a continuous random variable, then the survival function is given by $S(t) = P(T > t)$. The second one is called Hazard function, $h(t)$, and represents the potential risk of having an event given that the observation has survived up to time t:

$$h(t) = \lim_{\delta t \to 0} \frac{P(t < T \leq t + \delta t | T \geq t)}{\delta t} \tag{3.8}$$

Technically speaking, the hazard does not represent a probability because we are dividing by a time interval, so it is a rate instead. However, if we consider this rate to be instantaneous, we can say that the hazard represents the probability of an observation that has not had an event up until time t, to have an event at that time. Hazard is also known as the Hazard rate. The hazard rate is integrated over time to get the CHF:

$$H(t) = \int_0^t h(u)du \tag{3.9}$$

The CHF ($H(t)$) can be interpreted as the total amount of risk accumulated up until time t [7], and can be thought as the number of expected events over a period of time. The Survival function, CHF and Hazard ratio can all be determined based on one another:

$$S(t) = exp\{-H(t)\} \tag{3.10}$$

$$F(t) = 1 - exp\{-H(t)\} \tag{3.11}$$

$$f(t) = h(t)exp\{-H(t)\} \tag{3.12}$$

where $F(t) = P(T \leq t)$ is the probability of an event happening before time t. The Survival function and Hazard rate are estimated with the KM and NA estimators, respectively.

3.3.2 Statistical Methods

Survival Analysis statistical methods can be divided into three classes: Non-parametric, semi-parametric and parametric [7]. Non-parametric methods are the most commonly used of the three because we usually do not know the underlying distribution of survival times, however, it is also the one that yields the most inaccurate estimates. In Semi-parametric models, the distribution of the outcome remains unknown, however, the regression parameters are known. The non-parametric estimate of the Hazard function makes this class of models much more flexible than parametric approaches. And lastly,

in parametric models, we assume that the outcome follows a known distribution, such as the Weibull, exponential or log-normal distributions. Since we are estimating fewer parameters, parametric models offer several advantages if we do indeed choose the right outcome distribution, such as more accurate survival times and easier interpretation of the model. In the case of our dataset, since we are unable to accurately identify a distribution for the outcome, non-parametric and semi-parametric approaches are preferred, and these are the ones that will be presented next.

Kaplan-Meier Estimator (KN): Is a non-parametric method for estimating survival curves. This estimator uses the durations of each observation to estimate the survival rate at each point in time [7]. Given $T_1, T_2, ..., T_n$ i.i.d. survival times with Survival function $S(t)$. The KM estimate for time t, $\hat{S}(t)$, is given by:

$$\hat{S}(t) = \prod_{j:T_j<t} \left(1 - \frac{d_j}{r_j}\right) \tag{3.13}$$

where d_j represent the number of events at T_j, d_j is the number of censored observations between T_j and T_{j+1} and r_j is the number of individuals at risk before the j^{th} event, and is given by $r_j = r_{j-1} - d_{j-1} - c_{j-1}$ It is important to note that the observations should be ordered from the shortest to the longest.

Nelson-Aalen Estimator (NA): Is another non-parametric estimator, which is used to estimate the CHF for right censored data [34]. The estimate of the Hazard function at time t, $\hat{H}(t)$, is given by:

$$\hat{H}(t) = \sum_{t_j \leq t} \frac{d_j}{r_j} \tag{3.14}$$

where d_j is the number of deaths at time t_j, and r_j is the number of individuals at risk at t_j.

Cox Proportional Hazards model (CPH): Is a semi-parametric regression model used to investigate the correlation between survival time and one or multiple covariates [30]. Furthermore, CPH also provides the effect that each predictor has on the output, which is important for model interpretability. This model is based on the PH assumption, which means that every observation's Hazard function is the same (same shape), but with a different scaling factor. For example, given two independent observations a and b, the Hazard function of a is given by:

$$h_a(t) = Ch_b(t) \tag{3.15}$$

where C is the scaling factor. There are two major takeaways from this assumption, the first being that there is a baseline Hazard function, and so all other hazards can be derived

from the product between the baseline and some scaling factor; and the second one being that the impact variables have over survival times, do not change over time.

Formally, let T be the event time of an observation, which in the case of right-censored data is given by $T = min\{T^*, C^*\}$ where T^* is the true event time and C^* is the Censoring time (if there is any). Let us also define the $D = I\{T = T^*\}$ as an indicator labeling whether an observed time is an event or a censored observation. We can now define the full likelihood of censored survival times as:

$$L = \prod_i f(T_i|x_i)^{D_i} S(T_i|x_i)^{1-D_i} = \prod_i h(T_i|x_i)^{D_i} exp[-H(T_i|x_i)] \tag{3.16}$$

where each observation is denoted by i, with covariates x_i and observation duration T_i. Next, let us define the Cox model as per Cox regression:

$$h(t|x) = h_0(t)exp[g(x)], \quad g(x) = \beta^T x \tag{3.17}$$

where $h_0(t)$ is the baseline hazard, $exp[g(x)]$ is the relative risk function, x the covariate vector and β the parameter vector. We fit (3.17) by, first maximizing the Cox partial likelihood, which does not contain the baseline hazard. Next, the non-parametric baseline hazard is estimated based on the former operation. The Cox partial likelihood is given by:

$$L_{cox} = \prod_i \left(\frac{exp[g(x_i)]}{\sum_{j \in R_i} exp[g(x_j)]} \right)^{D_i} \tag{3.18}$$

where the negative partial likelihood is used as a loss function:

$$loss = \sum_i D_i log \left(\sum_{j \in R_i} exp[g(x_j) - g(x_i)] \right) \tag{3.19}$$

The cumulative baseline hazard function can be estimated using the Breslow estimator.

$$\hat{H}_0(t) = \sum_{T_i \leq t} \left[\Delta \hat{H}_0(T_i) \right] \tag{3.20}$$

$$\Delta \hat{H}_0(T_i) = \frac{D_i}{\sum_{j \in R_i} exp[\hat{g}(x_j)]} \tag{3.21}$$

where $\hat{g}(x) = \hat{\beta}^T x$. After fitting the model, both the estimated baseline function and the parameters β can be inspected. By exponentiating these parameters, we obtain the hazard ratio between two groups. For example, if this ratio is bigger than 1, then the positive class has a lower hazard when compared, which means a longer median survival time when compared to the negative class. And finally, the survival function of the Cox model can be estimated with:

$$\hat{S}(t) = exp[-\hat{H}_0(t)exp[\hat{g}(x)]] = \hat{S}_0(t)^{exp[\hat{g}(x)]} \tag{3.22}$$

where $\hat{S}_0(t)$ is the baseline survival function.

3.3.3 Machine Learning

Random Survival Trees (RSF): Ensemble models are built on top of a collection *base learners*, and have been empirically proven to substantially improve prediction performance. However, base learners need to be sufficiently distinct from each other, otherwise the model would just overfit as a regular decision tree. Two ways were developed to tackle this problem, the first one is based on injecting randomness into the samples that each learner trains on, with bootstrapping and with replacement [2]. The second one is randomly selecting features on each node split. Combining this randomization with averaging over the trees, allows a RF to be highly accurate and have low generalization error. Based on the RF model, RSF is an extension that allows for rightly-censored data to be included in the analysis. One of the core ideas proposed by RSF is the conservation of events principle, which is used to define a new type of predicted outcome for survival data called Ensemble mortality [16]. Ensemble mortality can be interpreted as the expected total number of events, and is derived from the ensemble CHF[1]. The RSF model can be summarized in the following steps:

1. Draw B bootstrap samples from the original data.
2. Grow a survival tree for each bootstrap sample. At each node of the tree, randomly select p candidate variables. The node is then split using the candidate variable that maximizes survival difference between daughter nodes.
3. Grow the tree to full size under the constrain that a terminal node should have no less than $d_0 > 0$ unique deaths.
4. Calculate a CHF for each tree and average them to obtain the ensamble CHF.
5. Using the remaining of the bootstrap data (OOB), calculate the prediction error for the ensemble CHF.

Just as a regular DT, a survival tree is comprised of decision nodes and end nodes (called leafs). The top node contains all the data, and the algorithm works by recursively splitting nodes in a left and right node based on a splitting criterion until a stopping criterion is met. In the case of survival trees, nodes are split by maximizing the survival difference between daughter nodes, pushing dissimilar cases apart. The work [16] evaluated four splitting rules based on the log-rank test statistic. In between these, splitting by maximizing the log-rank test and a standardized log-rank statistic yielded the lowest prediction error[2]. The authors also noted that a random log-rank splitting rule, described

[1] Since we are more interested in predicting survival time, this last metric will not be discussed in this document. For more information see [16].

[2] The authors in [16] used the C-score for evaluating the performance of splitting rule.

as choosing between random split candidate variables, as being significantly faster than the other rules whilst having a good performance as well.

When a survival tree reaches saturation, the most extreme nodes, called terminal nodes, are formed at the end of each tree when nodes break the minimum $d_0 > 0$ unique deaths rule. Let these nodes be denoted by τ, and let $(T_{1,h}, \delta_{1,h}), ..., (T_{n(h),h}, \delta_{n(h),h})$ be the survival times and the Censoring information (0 for censored, 1 for not censored) in a terminal node $h \in \tau$. Also, define $d_{l,h}$ and $Y_{l,h}$ as the number of deaths and individuals at risk at time $t_{l,h}$. The CHF estimate for h is the NA estimator,

$$\hat{H}_h(t) = \sum_{t_{l,h} \leq t} \frac{d_{l,h}}{Y_{l,h}}, \tag{3.23}$$

where the CHF for observation i with m-dimensional covariate x_i is the NA estimator for x_i's terminal node,

$$H(t|x_i) = \hat{H}_h(t), \qquad if\ x_i \in h. \tag{3.24}$$

In order to compute the ensamble CHF, we average over B survival trees. Let us define a Out-Of-Bag (OOB) estimate and a Bootstrap estimate. Let us now define $I_{i,b} = 1$ if i is an OOB case for b, otherwise, set $I_{i,b} = 0$. Let $H^*_b(t|x)$ denote the CHF for the tree grown from the bth bootstrap sample. The OOB ensemble CHF for observation i is then given by:

$$H^{**}_e(t|x_i) = \frac{\sum_{b=1}^{B} I_{i,b} H^*_b(t|x_i)}{\sum_{b=1}^{B} I_{i,b}}. \tag{3.25}$$

Gradient Boosting for Survival Analysis: Boosting algorithms have recently become popular among data science communities not only because of its ability to find nonlinear relationships between covariates and target features, but also its ability to deal with outliers and missing values inside non-cleaned datasets. Gradient Boost (GBoost) [9] is a flexible boosting framework that allows for the optimization of arbitrary loss functions, as long as these are differentiable. The main idea behind this ensemble model is the combination of multiple weak *base learners* in order to obtain a better model. The *base learners*'s predictions are combined in an additive manner in order to give the model's final prediction. Formally, given a covariates's matrix $\mathbf{x}$, a Boosting model is defined as,

$$F(\mathbf{x}) = \sum_{m=1}^{M} \beta_m g(\mathbf{x}, \theta_m), \tag{3.26}$$

where M represents the number of *base learners*, β_m the mth weighting term and g the *base learners* parameterized by vector θ. As we can see in Eq. (3.26), a GBoost model is built sequentially. In these ensemble models, the predictions from the previous *base learner* used in the computation of the next one.

Since this framework allows for the use of any loss function, we can use the GBoost framework for survival analysis by choosing an appropriate loss function. For example, in this work we use Cox's Partial likelihood function, Eq (3.19), as the loss function for the GBoost framework. From now on, we will denominate this model by Gradient Boosting for Survival Analysis (GBSurv), which uses Cox's loss function with regression trees as *base learners*.

4

Dataset and Exploratory Data Analysis

4.1 Database description

The data used in this dissertation consists of a customer database provided by NOS, and is updated weekly, with dates ranging from June of 2019 up to June of 2021. The dataset consists of a single file, which is hosted in the company's data lake. It can be accessed and queried using different technologies, namely, HUE [37] to quickly inspect tables and make queries, and pySpark [42] to query and manipulate data in Python.

The dataset consists of 748 features and several million instances. Features include date of snapshot, client identifier and type, number of months since the client started a contract with NOS, number of months until the client's binding period is over, number of months until the client churns and other explanatory features. A description of every feature group is presented in Table A.1 of Annex A.

4.2 Variable Description

Across all the feature groups present in Table A.1, there are three groups that need to be addressed independently:

Data keys have the date of when the entry was added, and also the client identifier. Entry date is referred as snapshot, and each snapshot consists of a weekly view of all clients that have a subscription service that includes TV, which is then appended to the table. Regarding churn events, the table is updated monthly with a time window, such that entries within the window will have a churn event inputted (date and duration until the event).

Target features has information about the churn event, namely the date of the event, number of months until the event and a binary feature symbolizing whether a client has churned or not in different monthly time windows.

Client features have information related to the client. There are two features from this group that are usually used by NOS to do client segmentation, which, is also relevant to churn behavior and the way NOS deals with customers from each segment. These two features are the number of months since the client joined the company (*client-dur*) and the number until the binding period (*pf-dur*) is over.

4.3 Exploratory Data Analysis

Some work has already been done to understand the population in this dataset. The first thing was identifying which features would be the base of this study, which were then presented in Section 4.2. It is important to remember that, client features such as *client-dur* and *pf-dur* are not only important because of their impact in customer churn (as demonstrated in an internal study done by NOS), but also because they offer powerful business insights by doing a segmentation of these features into different groups of clients. In order to perform this analysis correctly, the data was cleaned using the filters and data imputation values used in NOS churn pipeline, which are mentioned in Section 5.3. The entire dataset was used in this analysis.

Before starting, let us recall what variables will be used in this the exploratory analysis. Table 4.1 shows a mapping between the original names of the variables (present in some plots), and the naming scheme used in this dissertation. A brief description of each variable is also shown.

Table 4.1: Explanatory and target features naming scheme

Name	Original Name	Description
Snapshot	cal_day_dat	Date of snapshot
Client Id	sa_cod	Client Identifier
churn-dur	next_voluntary_churn_months_qty	Number of months until the client churns
Churn date	next_voluntary_churn_dat	Date of the churn event
Churn duration (discrete)	next_mx_voluntary_churn_flg	1 if the client churned in the next x months, 0 otherwise
client-dur	sa_activation_months (client_born)	Number of months since the client signed a contract with NOS
pf-dur	loyalty_end_months_qty (pf_dur)	Number of months remaining until the end of the binding period

It's important to note that the identifier variables present in Table 4.1 were not used in the modelling phase of this disseration, such as the *Snapshot* and the *Client ID* variables. The remainder exploratory features were previously selected by NOS, and include variables related to service usage, package information, outreach from and to NOS and customer basic information, as presented in the feature groups in Table A.1. Even though this set of features resulted in good results, as seen in the next section, It's possible there

exist others that weren't used during training, and which might improve the model's performance.

Checking number of Nulls and invalid instances Firstly it is important to check if any of the key variables have Null values. Albeit not having any Null values, many of the numeric features such as *client-dur*, *pf-dur* and the target feature *churn-dur* had some invalid values, which were then adequately imputed by following Section 5.3.

Checking class imbalance Another important aspect related to this dataset is its class imbalance problem, so it is fundamental that we measure how imbalanced our data is, specially for the worst case scenario: one months churners. In order to achieve this, for each snapshot, we counted the total number of customers that have a *Churn duration discrete (m1)* equal to 1 (clients that churned the month after the snapshot was taken), divided it by the total number of customers in that snapshot, and finally multiplying this value by 100 to obtain the result in percentage. Table 4.2 shows the average, minimum and maximum values of the class imbalance between all snapshots.

Table 4.2: Class imbalance summary

Average	Min	Max
0.60%	0.26%	0.74%

With these values, we can estimate how much we need to balance our dataset using the techniques shown in Section 3.2 in order to obtain better results. These tests will be done in the next phase of this dissertation.

Overall churn distribution using the discrete churn variable In this section we want to study how the churn rate varies along time. Figure 4.1 shows the discrete churn rate (churn in the next 1-5 months) along snapshots, where $m1, m2, ..., mx$ refers to the number of churners in the next $1st, 2nd, ..., xth$ months. There are some interesting takeaways from this plot: First, the rate of churn varies between 0.26% – 0.74% for $m1$, which is lower than the average 1.8% monthly churn rate for telecom companies as per [41]. Another important aspect is that, on average, churn rate does not seem to vary much over time, and that no sazonal variation was observed. However it is interesting to see that there is a clear decrease of the churn rate after the Covid19 pandemic started (March of 2020), which is partially due to some measures imposed by NOS that made it more difficult to churn during this time, and also because people spent more time at home.

Client Segmentation by *client-dur* and *pf-dur* As was said before, both *Client age* and *PF age* present themselves as both important predictors for churn, and are the basis of a client segmentation that defines different treatments for different customers. Because of this, both variables were segmented using pre-defined groups used in NOS's CRM division. With this segmentation we expect to identify groups of customers that are more

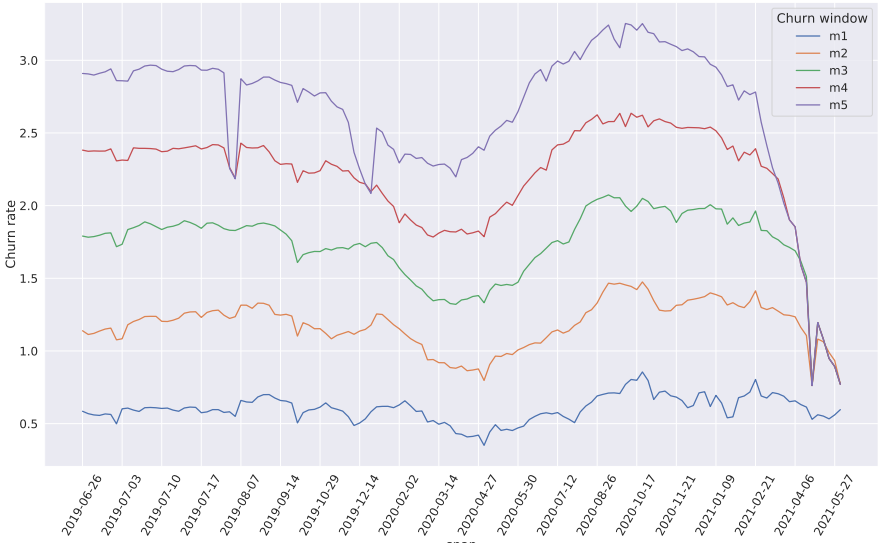


Figure 4.1: It is possible to see a drop in the number of churners around the time Covid19 appeared (March 2020). The drop at the end, specially at higher *mx* is due to the fact that the churn variables have not been updated yet for dates after 2021-06-06.

prone to churning, which then translates into more adequate client retention strategies for each one of these groups. Client segmentation was done in the following manner:

$$\text{pf\_dur} = \begin{cases} x < 0 & \text{,No PF} \\ 0 \leq x < 6 & \text{,Finishing PF} \\ 6 \leq x < 12 & \text{,Last year PF} \\ x > 12 & \text{,PF} \end{cases} \tag{4.1}$$

$$\text{client\_dur} = \begin{cases} x < 4 & \text{,New} \\ 4 \leq x < 25 & \text{,Recent} \\ x > 25 & \text{,Old} \end{cases} \tag{4.2}$$

Let us start by counting the number of instance of each group in Figure 4.2. These counts refer to the average number of clients in each snapshot for all snapshots.

Regarding *client-dur*, Figure 4.2 shows that the large majority of clients belong to the *Old* category, which was to be expected. When it comes to *pf-dur*, the number of customers under a binding period is more than double the customers with no binding period. It is important to note that older clients can re-instate a new binding period when accepting a new offer from NOS (this binding period is usually 24 months). It is also possible to see that there is not much difference in each group counts along the snapshots.

Churn count and distribution per *client-dur* and *pf-dur groups* After getting the counts for each group, it is interesting to check if there is any difference in the churn rate in each

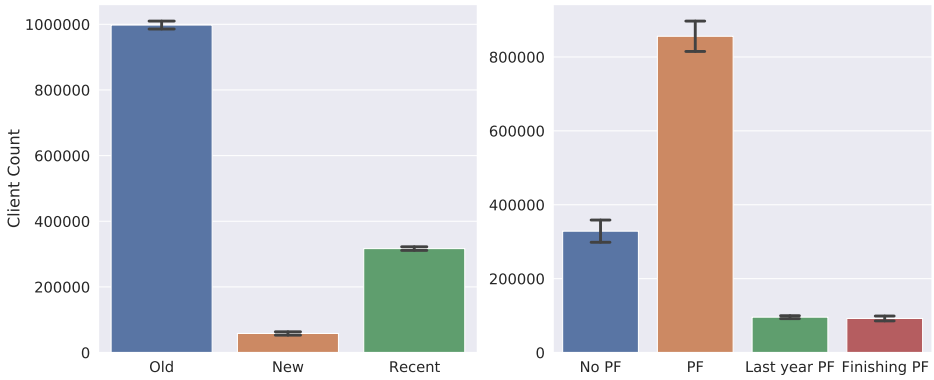


Figure 4.2: There is a clear difference in the total number of customers in the *Old* and *PF* groups, for the *client age* and *PF age* variables respectively.

group. To do this, let us first check the percentage of churners in each group in Figure 4.3. This percentage was obtained by dividing the number of customers with a *Churn duration discrete (m1)* equal to 1 by the total number of customers in each snapshot, and for each group.

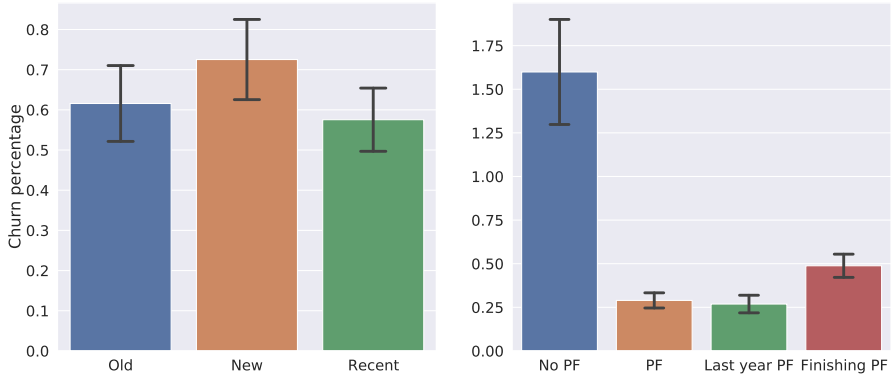


Figure 4.3: For the *client-dur* variable, the *New* group has more churners, as expected than the other groups, but not much difference overall. In regards to *pf-dur*, there are three times more churners in the *No PF* group than the *2nd* group with the most number of churners, *Finishing PF*.

It is possible to see that for the *client-dur* variable, the percentage of churners does not vary much among the different groups. However, it is curious that the *New* group is the one that has the highest amount of churners, when it is known that it is much more difficult for clients to churn when they are in the binding period. In the case of the *pf-dur* variable, it is clear that the *No PF* group is the one with most amount of churners, which is not surprising since not only are older customers more prone to churning, but also because this group has customers that just ended their binding period, which usually it is a precursor to churn. It is important to note that this information will be used to assess if it is worth it or not to consider every customer group in the analysis, and also what is

the gain/loss of predictive power when dropping one of the groups.

Apart from churn counts, it is also interesting to check the churn evolution per *client-dur* and *pf-dur* groups. In order to achieve this, a plot with the churn counts for each group and for each *churn date* is presented in Figure 4.4 and in Figure 4.5

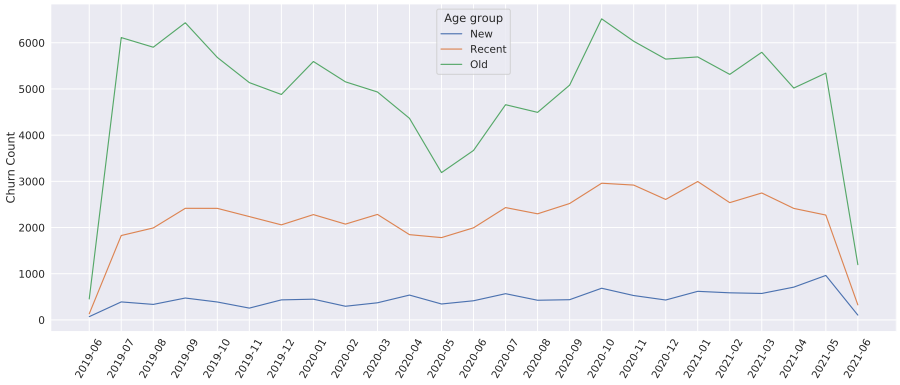


Figure 4.4: Clear drop in the number of churners around the time Covid19 hit (March 2020).

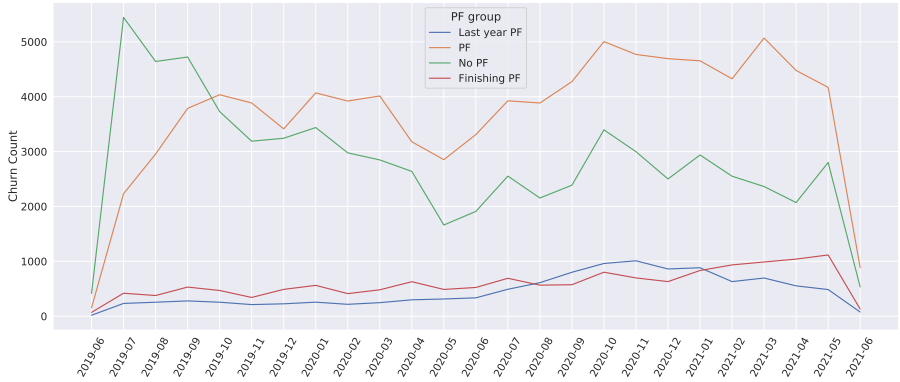


Figure 4.5: There is a clear drop in the number of churners around the time Covid19 hit (March 2020).

Figure 4.4 and Figure 4.5 clearly shows the the presence of a drop in the churn count around the time Covid19 started.

We can also check the churn distribution over the *client-dur* and *pf-dur* values to see if there is any peak values along the features's values. A Kernel density plot was used to achieve this by representing the distribution of both variables in regards to churn count. This plot is presented in Figure 4.6.

Figure 4.6 shows that there is a clear peak in churn density in between a *client-dur* value of 20 and 30, which coincides with the usual maximum binding period of 24 months. It is also possible to observe a clear churn peak when *pf-dur* equals to 0, which means that most clients churn after their binding period is over.

Figure 4.6: Both variables have a peak in some range of values. In the case of the *client_born* variable (*client-dur*) there is a peak at around 24 months. For the *pf_dur* variable (*pf-dur*) the peak is around the 0 month mark. Both these values represent the end of the binding period.

5

Experimental setup

As a scientific field, data science demands reproducible and statistically validated results in order to generate knowledge from data. In this chapter we are going to explore each step of the experimental process used in the context of survival analysis for this dissertation, and discuss a newly developed pipeline to make this validation process possible. Let us begin by discussing the methodology used to validate experiments.

5.1 Time-series validation

Statistical methods are usually used to evaluate and validate the performance of scientific experiments. In the domain of machine learning, k-fold cross-validation is usually one of the preferred methods to achieve this. This technique consists of randomly splitting the dataset into k folds, train a model on all folds except one, and finally test the model on the remaining fold. These steps are repeated until the model is tested on all folds, and a final metric given by its average over the k folds. This method has not only the advantage of robustly validating a model's performance, but also helps prevent overfitting. However, since our data is time-series in nature, it is illogical to train a model using values from the future and test it on past ones, as would naturally happen in randomly selected folds. This means that there is a temporal dependency between instances, which must be preserved during the experimental validation. For time-series data, a rolling basis cross-validation is usually preferred. In this method, we split the data into time-steps, train each one, and test it with its subsequent steps. For each step, the final score is given by its average over all the k subsequent steps. Figure 5.1 illustrates this process more intuitively.

Here we split the data into monthly sets, and for each month, given by *Train index*, we train a model, and then test it in k subsequent months. The final score for each *Train index* is given by the average of its k evaluations. Finally, we can obtain a single score by averaging over all *Train indexes*. With this methodology we are not only able to validate experiments over a single time-step, but also across different steps over time. As we saw in Section 4.3, each snapshot (or time-step) consists, on average, of at least a million instances before sampling, which as we will see in Section 6.3.3, grows quadratically

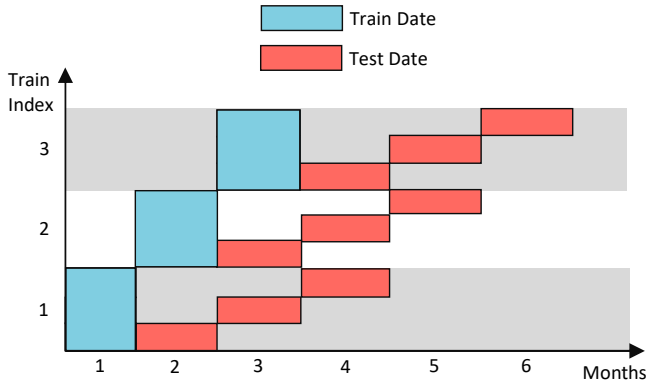


Figure 5.1: Rolling cross-validation in a time-series.

in computational fit-time for datasets above a certain size (>50000 samples). As such, since the dataset size surpasses, on average, this value, the train-set will be split into 'k' stratified folds (in relation to the churn event survival feature), and a model fitted for each fold. The final score for each train-test pair is given by the average over all the scores in each fold. The test set is the same for each fold, and contains all test instances.

5.2 Experimental Pipeline

Based on the cross-temporal validation methodology presented in the previous section, an experimental pipeline was developed. This pipeline is composed of a set of ordered steps, which we will denominate as DOF from now on. A visual representation of the process, is given by Figure 5.2.

Due to time constraints in this dissertation, some assumptions were made about this process. The most important one being the assumption of independence between different DOF, that is, each *DOF* serves as a scaling factor regarding the final evaluation, and as such, we assume no interaction between them. Formally, given a train-test data pair as X, split in n folds, and k parameterized DOF as D, the resulting evaluation E is given by:

$$E_{ij} = \frac{1}{n} \sum_{i=0}^{n} \sum_{j=0}^{k} E\left(D_j, X_i\right) \tag{5.1}$$

where $E(D_j, X_i)$ represents the pseudo-contribution that *DOF* D_j applied to train-test pair X_i has on the final evaluation E_{ij}. The same assumption is made about DOF that have hyper-parameters, such as *Class imbalance* and *Modeling*. This means that hyper-parameter tuning was also done independently for each hyper-parameter, and no interaction between hyper-parameters was tested. As stated before, both assumptions were made in order to have results in a timely manner, because validating each and every

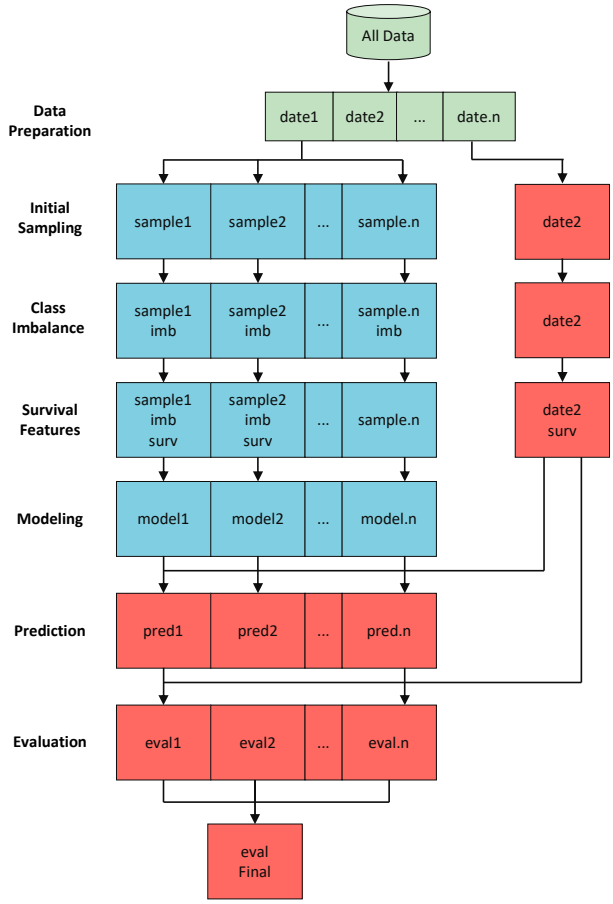


Figure 5.2: Experimental pipeline.

possible DOF combination and/or every *hyperparameter* combination would be too computationally expensive for the scope of this dissertation. Nevertheless, this pipeline is equipped to combine different DOF if desired. In order to run this pipeline, a Python script was developed. Each experiment is defined using a *yaml* configuration file, which after completed, produces a file with the resulting metrics (and other relevant information) for each train-test data pair. These files are then combined and used to produce the results shown in the next chapter.

5.3 Data Preparation

At the top of the process we have *Dataset import*, which has its own pipeline. Figure 5.3 shows each step of this pipeline and whether it is done locally with Pandas, or in a distributed manner with pySpark.

Load Panel: The process begins by loading a panel from the data lake defined in *Panel config*, resulting in a train and a test pySpark dataframes. To obtain these, the dataframe is filtered by a train and a test snapshot dates. Both sets are also filtered so that the

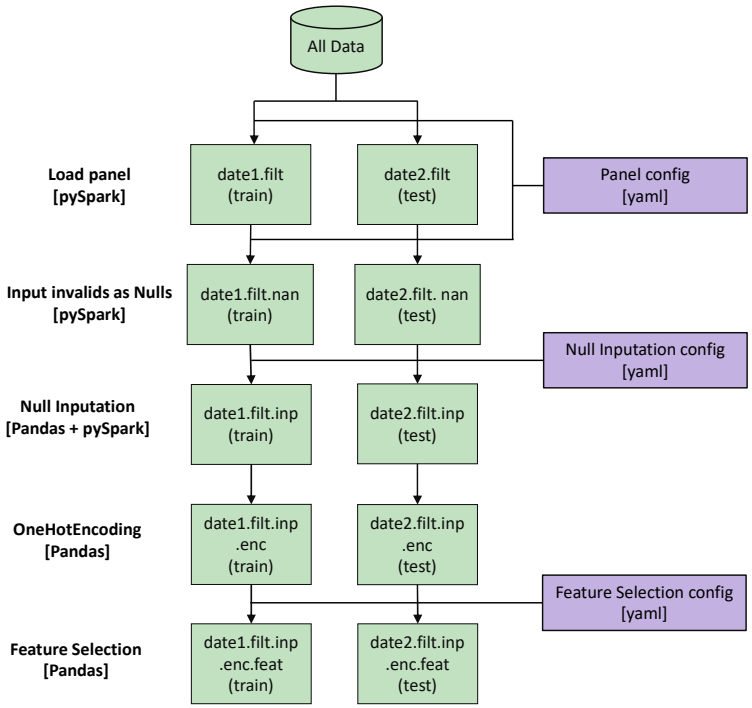


Figure 5.3: Data pipeline: Data loading, Nan Inputation, OneHotEncoding and Feature selection.

dataframe contains only *Consumer* type costumers.

Input invalids as Nulls: Next, invalid entries will be inputted as Null, to be replaced by an appropriate value in the next step. In this dataset, invalid values are given by -9999 or NaN for numerical features, and as '-9999', 'Indefinido' or Null for categorical ones. It is important to note that special care must be taken for some variables, namely client-dur, pf-dur and churn-dur. In the case of client-dur, entries with invalid values were filtered from the dataset. For pf-dur, this value is capped at 24 since it is the maximum number of months of binding period a client can have. And finally, invalid entries for churn-dur were neither inputted as Null nor removed from the dataset, as they simply mean that the client has not churned yet.

Null Imputation: The resulting Null values are then inputted using a set of rules defined in the *Null inputation config* file. These rules were pre-determined by NOS, and consist of using the median for continuous variables and the mode for categorical ones. During this step, the distributed datasets are firstly converted to a Pandas dataframe and some basic statistics taken (average, median, mode, etc) among other basic information about the train dataset. The values are then used to input Null values. This step is partially done locally in Pandas to allow for more flexibility.

OneHotEncoding: The inputed train dataset is then fed into a OneHotEncoder from

sklearn to binary encode categories within categorical variables. It is important to note that these variables can have a large number of categories, which would explode the dimensionality of the dataset after encoding. To avoid this, before encoding, categories with a representativity threshold inferior to a certain value (e.g. categories that appear in less than 1% of the dataset) were converted to a default category. The representativity threshold used during this work was the same used by NOS, and has a value of 0.1%. The encoder is fitted on the train dataset, and used in the test dataset to avoid data leakage. As data is going to be standardized in the next section, resulting in numerical features in the $[-1, 1]$ interval, one-hot encoded categorical variables are mapped to $(-1, 1)$ instead of $(0, 1)$ to keep consistency.

Feature Selection: The final step consists of removing features that are not included in the *Feature Selection config* file. The features present in this file were generated using NOS churn prediction pipeline, and calibrated with domain knowledge. Before leaving this stage, both datasets are standardized using the train-set's mean and standard deviation values for each numerical feature.

5.4 Class Imbalance

Albeit being a controversial topic inside machine learning and statistics communities, the Class balancing techniques shown in Section 3.2 will be experimented and evaluated in this dissertation as there is a big class imbalance in our dataset, which can, theoretically, harm our evaluation metrics. Undersampling methods can be applied freely as these only rely on removing instances. However, oversampling methods that generate new examples, need more careful consideration. Since some numeric features such as churn-dur, client-dur and pf-dur are used in the computation of survival features, without proper care, erroneous values can be generated. In order to avoid such values, any new generated example with erroneous survival targets will be corrected with respect to Eq. (5.2). It should be noted that an hybrid class was created based on the combination of the SMOTE oversampler and the Tomek-links undersampler. The imbalanced methods were implemented using classes from the *imbalanced-learn* python package [24] and can be checked in Table B.1.

5.5 Survival Targets

The performance of a survival model is heavily influenced, in terms of evaluation metrics, by the choice of its survival targets, duration and event. As stated in Section 3.3, event indicates whether a customer is censored or not, and duration represents the total lifetime of a customer. For the event target, firstly we need to specify an end-dur, which represents a quantity of time (in months), after a snapshot, that we run our hypothetical

study. A customer is considered censored ($event = 0$) if end-dur is greater than churn-dur, and non censored otherwise ($event = 1$). For the computation of duration, besides end-dur, it additionally depends on an init-dur value that represents a quantity of time (in months), before the snapshot, where we begin our study, and a granularity which aggregates churn-dur values into time bins (e.g. daily or weekly bins). The latter can influence the performance of the model, in particular the computation time required to make predictions. Figure 5.4 shows a visual example of how the survival targets are computed.

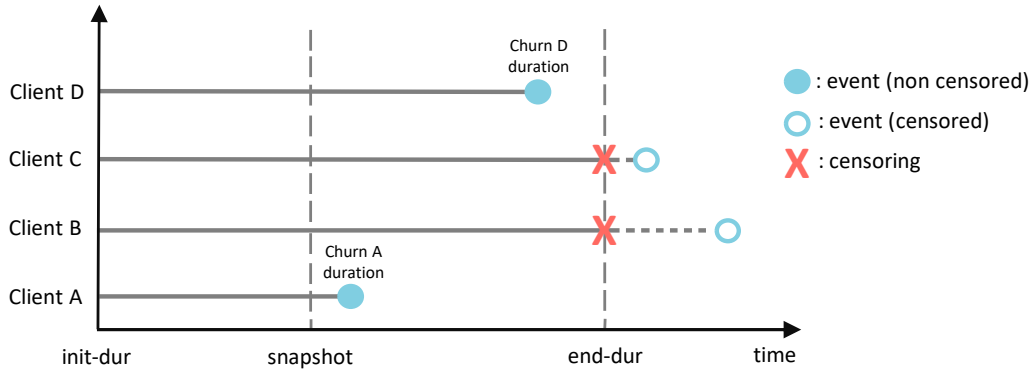


Figure 5.4: Event and duration and how censoring influences its computation.

Formally, the survival targets event and duration are computed as:

$$\text{duration} = \begin{cases} d_i + d_c & \text{,if } d_c \leq d_f \\ d_i + d_f & \text{,otherwise} \end{cases} \qquad \text{event} = \begin{cases} 1 & \text{,if } d_c \leq d_f \\ 0 & \text{,otherwise} \end{cases} \tag{5.2}$$

where d_i is init-dur, d_f is end-dur and d_c is churn-dur. It is important to note that init-dur cannot take values less than the client-dur feature, that is, we cannot consider the survival study of a customer to begin before he actually joins NOS. As such, init-dur will be capped to client-dur in cases where init-dur > client-dur.

5.6 Modeling

In this *DOF*, the survival models presented in Section 3.3.3 will be fitted using the previously transformed dataset and evaluated in both metric and computational performance[1]. In order to fit these models, the Python package *scikit-survival* was used [29]. Besides these, a benchmark model resembling a model generated by NOS's churn prediction pipeline, will also be fitted and evaluated in order to validate the performance of the survival models. The benchmark model is a binary classifier, and was built using the

[1] For reference, every experiment was ran locally on a linux server with an Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz and 64GB of Ram

same set of DOF parameters as the survival models, except for *Survival Targets*, of which we are only interested in the binary event column. The classifier used by NOS is a GBoost tree classifier, *GBTClassifier* from the pyspark.ml library [42]. Since all survival models are ran locally, to keep comparisons between models fair, the benchmark model was also fitted locally using sklearn's *GradientBoostingClassifier* [28], which is conceptually equivalent to pySpark's *GBTClassifier*. To validate the benchmark model against NOS's pipeline model, a train-test data pair was chosen and both models fitted using this data with the same set of hyper-parameters. After fitting, the lift chart and feature importance table for the top 10 features was used to compared both models. If the results mostly agree, then we consider the benchmark model validated. Initial testing with the new pipeline was done and validated both of aforementioned tests. After using the train data to fit survival models, both survival curves and risk scores were predicted and used in the model's evaluation. The risk scores are computed differently depending on the model, as per [**scikit-survival**]:

- **Cox Proportional Hazards**: Using the log hazard ratio, that is, the exponential of the dot product between the feature and parameter vectors;
- **Survival Random Forests**: Total number of events, which is estimated using the sum of the estimated ensemble cumulative hazard function, as per Eq. (3.25);
- **Gradient Boosted Survival**: Using the log hazard ratio, similar to the linear predictor of a Cox Proportional Hazards model.

In the case of the benchmark model, the predicted probabilities are used instead. The Class name used to fit every model, as well as its package can be seen in Table B.1.

5.7 Evaluation

The models presented in Section 3.3 will be evaluated using the metrics shown in Section 3.1. From the four chosen metrics, the lift and gain charts will be discussed in more detail as they are the ones that carry the most impact business-wise. Both lift and gain charts are discriminatory metrics, that is, they measure how well a model ranks patients according to risk. In the case of a classification model, the ranking is as straight-forward as ordering customers based on their predicted churn probability. In survival models, this can be done in several ways, such as using the survival probability from the survival curves or the time-to-event based on a survival probability threshold. However, we will use the risk scores predicted for each model on the test data instead.[2] In order to get these metrics, we define a new duration, lift-dur, in months, that will be used to get the ground-truth labels for the test-set, which will serve to compute the true ratio of people that

[2]Initial testing proved the other two methods to give poor results for the lift and gain metrics, so they were not considered going further. For other models that do not provide a risk score, these are still a valid option if one wants to rank instances based on survival risk.

churned in each quantile. Regarding the lift and gain metrics, a customer is considered churned (event=1) if churn-dur < lift-dur, and not churned (event=0) otherwise.

All other metrics, C-score and Brier-score, are obtained directly from the model using the appropriate method. It should be noted that the survival curves must be estimated prior to computing the time-dependent Brier score. This metric also requires a set of time points t to compute the score, and must fall between the minimum event test time (including) and the maximum event test time (excluding). Furthermore, this method requires survival curves to have a minimum of two points. The former constraint was dealt using a data-driven approach by selecting all time points between the 10% and 90% percentile of observed test event points. As for the latter, this is a problem that can arise if the train-set has few unique event points. In those cases, the brier score is given by the respective score for a random model, that is, one that would predict a probability of 0.5 for every survival curve, in each time point. Apart from the aforementioned evaluation metrics, the fit times will also be reported.

5.8 Experiments

Taking into consideration the aforementioned DOF, experiments were implemented in several *yaml* configuration files to be ran by the newly developed pipeline. An example of an experiment configuration file can be seen in Listing 5.1.

Listing 5.1: Experiment: yaml configuration file example

```
n_dates: 2
n_folds: 1
lift_durs: 1
n_samples_train: 1000
n_samples_test: 1000
samplers:
  RUS:
    sampling_strategy: 0.33
init_durs: '0'
end_durs: 1
granularities: 'weekly'
models:
  RSF:
    n_estimators: 100
    min_samples_leaf: 2
    max_depth: 4
    min_samples_split: 10
```

In this experiment, we run two Train-indexes (*n_dates*), which are evaluated in two subsequent test-sets, each. Initially, 1000 examples are sampled from each train-set and test-set (*n_samples_train* and *n_samples_test*), with stratification and without replacement, and each sample is split into 2 folds (*n_folds*). If one does not want to run an initial sampling, then *n_samples_train* and *n_samples_test* must be equal to -1. Each experiment

will be evaluated on the average, over all train-test pairs and lift-durs, for the lift and gain metrics, and also for each lift-dur in separate. The C-score and Brier-score will be grouped and evaluated by each train-test pair in order to assess the variability of these metrics across date-pairs. Furthermore, all metrics are also evaluated over the subsequent test dates, for each train date, in order to check how stable a model performance is in future dates. The Class Imbalance method (*samplers*) used was a Random Under Sampler with one hyper-parameter, the initial study duration starting on the date of the snapshot (*init_dur*=0), the respective end study duration on the next one month (*end_dur*=1), with a *weekly* granularity. And finally, the model used was a Random Survival Forest with the respective hyper-parameters (*models*). It is important to note that all items in this configuration file can be incremented. For example, it is possible to add more than one item to *end_durs* to be evaluated in the experiment.

Every parameter used in *Data Preparation* was chosen based on empirical studies ran by NOS, which includes steps such as null imputation, feature encoding and feature selection, and thus were not calibrated using this pipeline. For the remaining DOF, the pipeline was used to calibrate its parameters and, when applicable, its hyper-parameters[3] (in the case of the class imbalance and modeling DOF). For these experiments, ten monthly spaced snapshots (dates) were chosen. One immediate problem that arises when using the validation methodology explained in Section 5.1 is that each subsequent test-set must have the same features as the respective train-set, which, as we saw in Section 5.3, might not always be true due to the way categorical variables are encoded. For example, it is possible that a category, in either the train-set or test-set, does not have sufficient representativity to be considered, resulting in mismatched features between datasets. Since the features we will be using were already chosen (including one-hot encoded ones), the ten dates were chosen with the condition that the resulting encoded features were also present in the *Feature Selection config* file.

Table 5.1: Experiments: DOF, Experiment Name and Description

DOF	Experiment Name	Description
Class Imbalance	study.sampling.s1	Class Imbalance Hyper-parameter search
Class Imbalance	study.sampling.s2	Metric performance between Class imbalance methods
Survival Features	study.survfeats.s1	Metric performance between Init-dur and Granularity
Survival Features	study.survfeats.s2	Metric performance between End-dur and Granularity
Modelling	study.model.s1	Model Hyper-parameter search
Modelling	study.model.s2	Metric performance between Models
Modelling	study.model.s3	Computational performance between Models

Each experiment will be further detailed in the next Chapter, such as the list of parameters tried for experiments with a hyper-parameter search, fixed DOF values in each

[3] each evaluated hyper-parameter uses the respective model's default hyper-parameter values for the remaining model hyper-parameters

experiment, the number of dates used, among others.

6

Results and Discussion

6.1 Class Imbalance

6.1.1 Hyper-parameter search

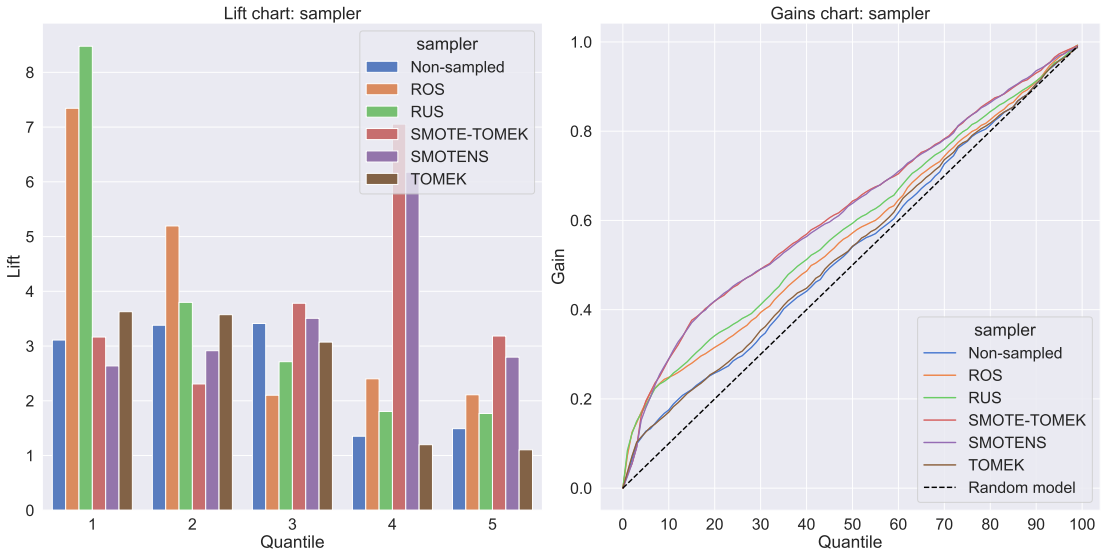
Let us begin with the Class Imbalance DOF experiments. The purpose of the first experiment, *study.sampling.s1*, is to find the best set of hyper-parameters for each class imbalance methodology. As discussed in Section 5.4, a set of different hyper-parameter values was chosen for each sampler. These set of values can be seen in Table C.1. The values for the remaining parameters of this experiment can be viewed in Table C.2. After the experiment finished, the resulting set of metric files was combined and used to produce lift and a gain charts, as well as a c-score and a time-dependent brier score plots for each hyper-parameter. These plots can be seen in Appendix C.1. To compute the best values for each hyper-parameter, we rank each of the four metrics, and then sum the resulting ranking values (first-place = 1, second-place = 2, etc). The hyper-parameter value with the lowest score will be chosen as the best value. The set of best hyper-parameters for each sampler can be seen in Table 6.1.

Table 6.1: Best Class imbalance hyper-parameters (*study.sampler.s1*)

Sampler Name	Hyper-Parameter	Best Value
RUS	sampling_strategy	0.2
ROS	sampling_strategy	0.2
SMOTE-NC	sampling_strategy	0.05
	k_neighbors	1

6.1.2 Sampler Comparison

The sets of best hyper-parameters from *study.sampler.s1* were used to run another experiment, *study.sampler.s2*, to test which of the class imbalance methods performed better. The values for the remaining parameters of this experiment can be viewed in Table C.3. Let us begin by plotting the lift and gain charts, which are represented in Figure 6.1.



(a) Lift chart

(b) Gains chart

Figure 6.1: Lift and Gain Charts: Class Imbalance methods.

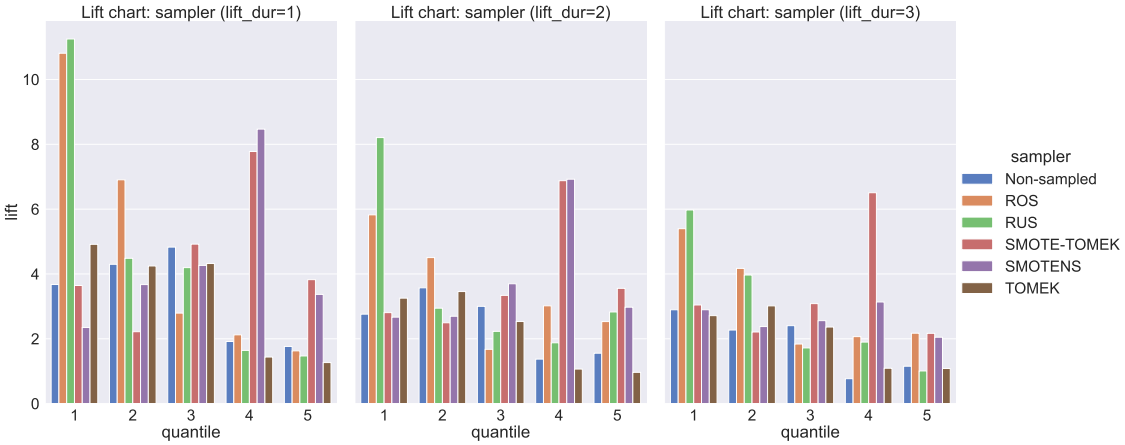


Figure 6.2: Lift chart: Class imbalance methods (lift-dur).

From Figure 6.1(a), we can see that the majority of class imbalance methods is better than no-sampling. The same thing can be seen in Figure 6.1(b). In both Figures we can see that the best performers are the RUS and ROS methods, since they are the ones that have the highest lift in the first quantile and the most accumulated gain in the first twenty quantiles (RUS coming slightly ahead). We can also see that neither *SMOTENS*, *TOMEK* or its hybrid performed well on these metrics. Also, as expected, in Figure 6.2 we can observe an inverse relationship between *lift-durs* and both lift and gain scores. This is because, the higher the *lift-dur*, the more customers are considered to be churned in the ground-truth, thus increasing the true churn average and a lower lift score.

Apart from these, it is also important to evaluate if any of these metrics change over time for the same model. Figure 6.3 shows that, for the most part, the choice of method

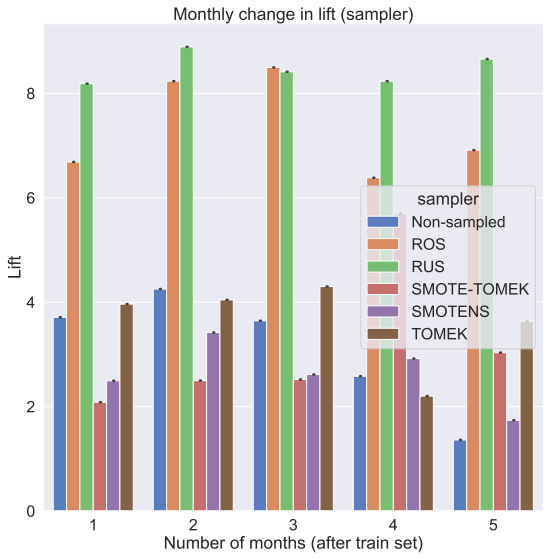


Figure 6.3: Monthly change in Lift (1st quantile): Class imbalance methods.

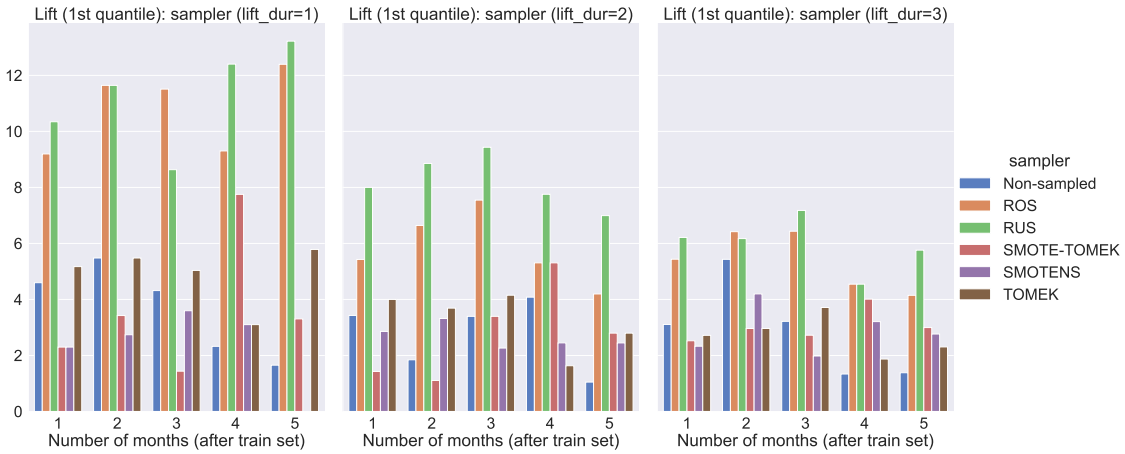
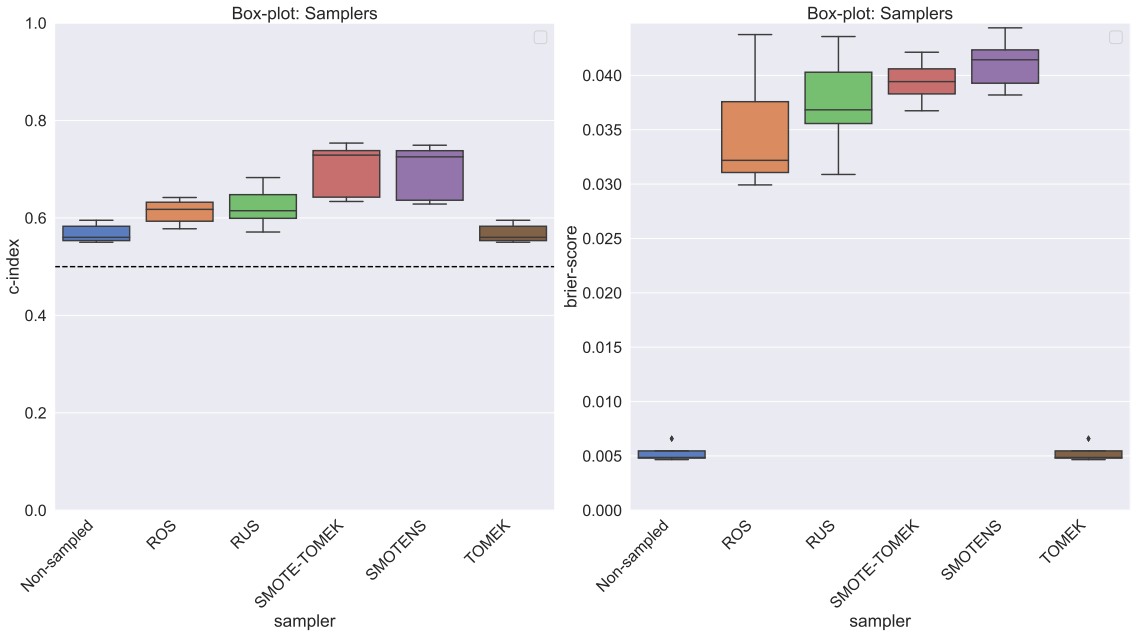


Figure 6.4: Monthly change in lift (1st quantile): Class imbalance (lift-dur).

does not have an influence on the lift score (first quantile) over time, apart from the *Non-sampled*, where the score was halved in month 5 (compared to month 1). The same behavior can be seen across different lift-dur values, as per Figure 6.4.

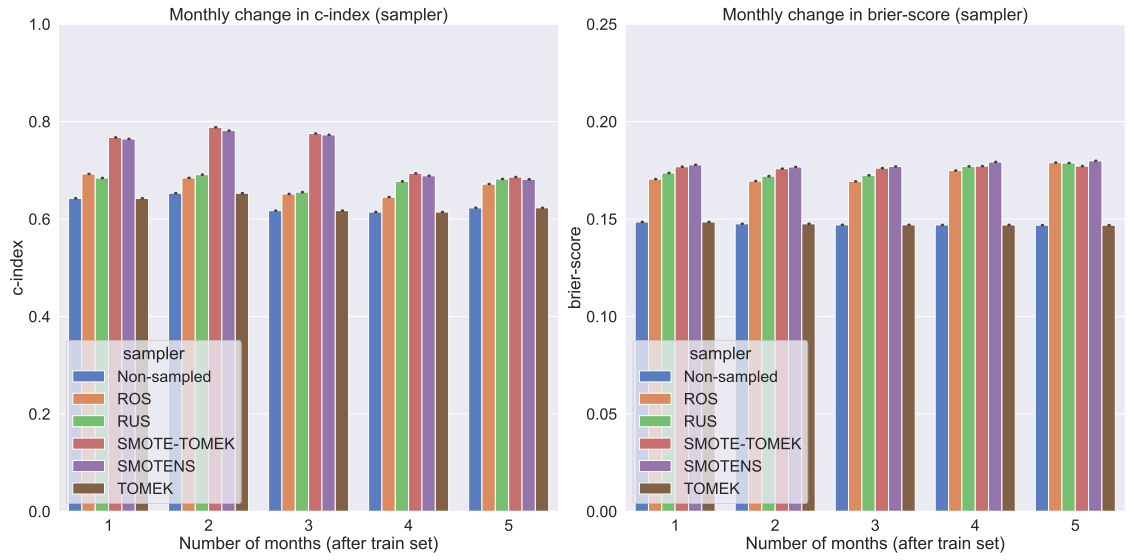
Regarding the C-score and Brier-score, the first thing that stands out from both Figure 6.5(a) and Figure 6.5(b) is that both *Non-sampled* and *Tomek* have, approximately, the same C-score and Brier-score, which means that there is a possibility that no Tomek-links were formed using the *TOMEK* method. Surprisingly, these were the best performers on the Brier-score, which can be explained by the lower percentage of churners obtained with these methods, making predicting the duration to churn easier, since churners duration will default to *end_dur*. Regarding the C-index, the best performers were *SMOTENS* and the *SMOTE-TOMEK* hybrid method. This is not in concordance with the results from the lift and gain charts, and also with the brier-score results, which show a better performance



(a) C-score (b) Brier-score

Figure 6.5: C-score and Brier-score: Class Imbalance methods.

for the RUS and ROS samplers.



(a) C-score (b) Brier-score

Figure 6.6: Monthly change in C-score and Brier-score: Class Imbalance methods.

Lastly, let us see how the C-score and Brier-score vary across subsequent test dates. Figure 6.6(a) and Figure 6.6(b) show that there is even less variation when it comes to these metrics over time. We can thus conclude that the choice of class imbalance method

does not have an influence on either metric over future test dates.

6.1.3 Recommendation

Since *Tomek* and *Non-sampled* were the worst performers on 3/4 of metrics, then we conclude that they should be avoided. *SMOTENS* and *SMOTE-TOMEK* performed the best in the C-score metric, but lacked in comparison to the random samplers (RUS and ROS) in the Lift, Gain and Brier-score metrics. Since for NOS the former two metrics have the most importance, we recommend to either RUS or ROS to be used moving forward. Between these two, we recommend using RUS since it will result in a lower number of instances, which translates into less time fitting models. In Table 6.2, we present the final recommendation for the Class Imbalance DOF.

Table 6.2: Class Imbalance: Final recommendation.

DOF	Parameter	Hyper-parameter
Class Imbalance	RUS	sampling_strategy = 0.2

6.2 Survival Targets

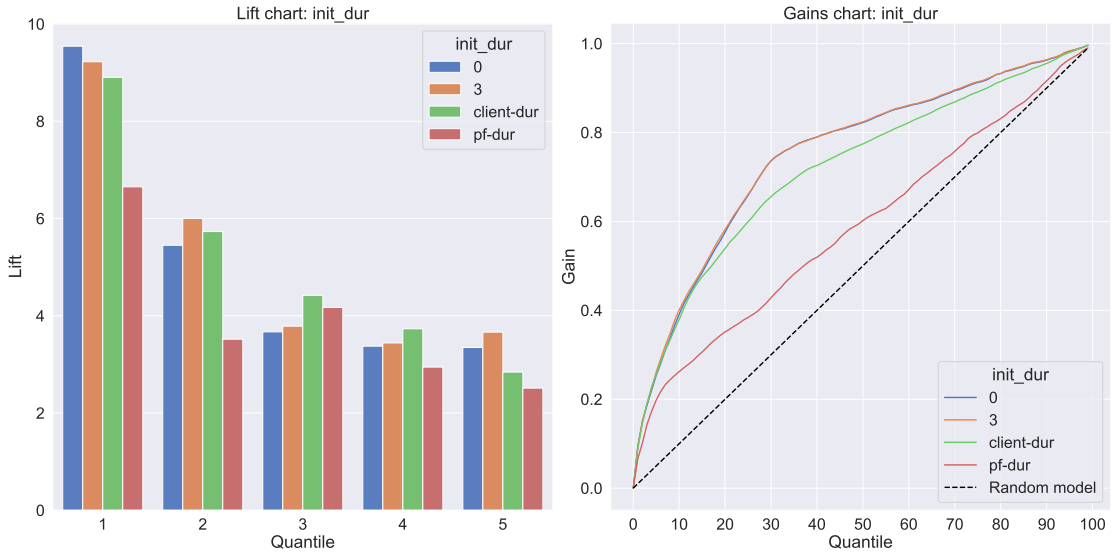
6.2.1 Init-dur comparison

The first Survival feature we are going to discuss is init-dur, which is the variable that defines the beginning of the survival study period, and was evaluated in *study.survivalfeats.s1*. This feature can be given by either a value, indicating the number of months before the snapshot, or by a feature, such as the number of months since the client joined NOS. In Table 6.3 we show the number of init-dur values being tested in this section. The fixed variables for the remaining DOF parameters can be seen in Table C.4.

Table 6.3: Survival Targets: *Init-durs* values.

Survial Feat	Values			
init-dur	0	3	client-dur	pd-dur

From Figure 6.7(a) and Figure 6.7(b), the first observation we can make is that pf-dur is the worst performer among all init-dur values regarding the lift and gain metrics, and that init-dur=0 performed the best. In Figure 6.8 we can observe again the same inverse relationship pattern between lift and gain scores and increasing *lift-durs*. We can also see that the relative difference in the lift scores between init-dur values is reduced over lift-dur values, but this is most likely just normal variation. From this we can conclude that, regarding the lift and gain metrics, we should avoid using pf-dur as the beginning of our survival period.



(a) Lift chart

(b) Gains chart

Figure 6.7: Lift and Gain Charts: Init-durs.

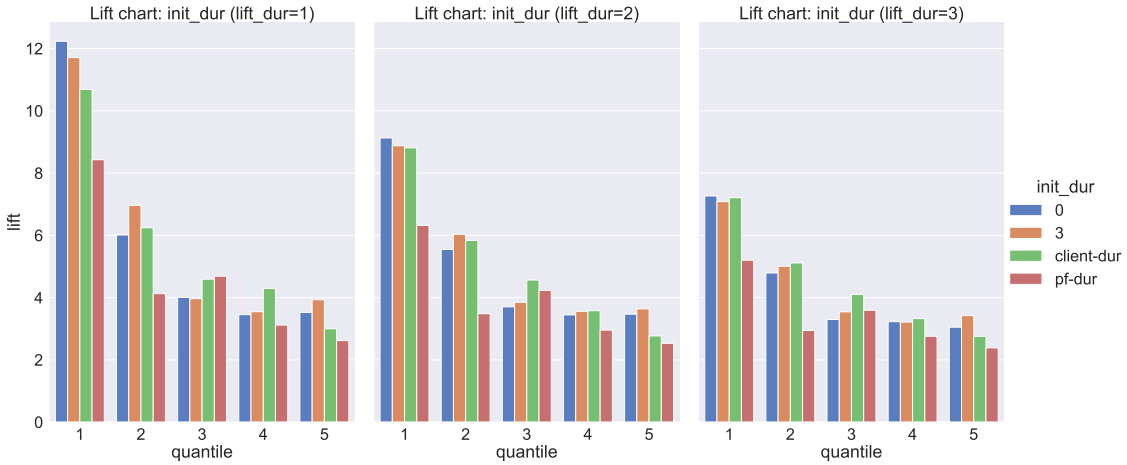


Figure 6.8: Lift chart: Init-durs (lift-dur).

With Figure 6.9 we observe a slight decrease across subsequent test dates for all init-dur values. From Figure 6.3 (RUS and ROS), we can see the same pattern as client-dur in Figure 6.9. This is because client-dur was the chosen init-dur in the *study.sampler.s2* experiment.

Next, in Figure 6.11(a) and Figure 6.11(b) we analyze how different init-dur values perform with respect to the C-score and Brier-score. We discretized each init-dur by granularity to assess how the latter influences these metrics. It is possible to see that init-dur values '0' and '3' were the best performers for both metrics. Regarding granularity, we can see an inverse relationship between the relative performance of its values in both the C-score and Brier-score, where 'daily' was the best regarding the C-score and 'weekly'

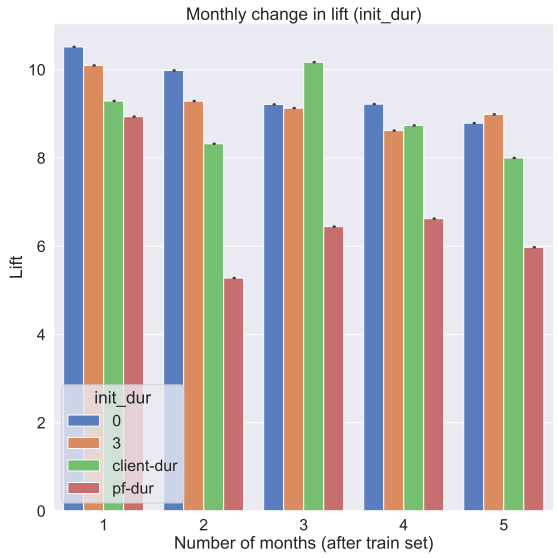


Figure 6.9: Monthly change in Lift (1st quantile): Init-durs.

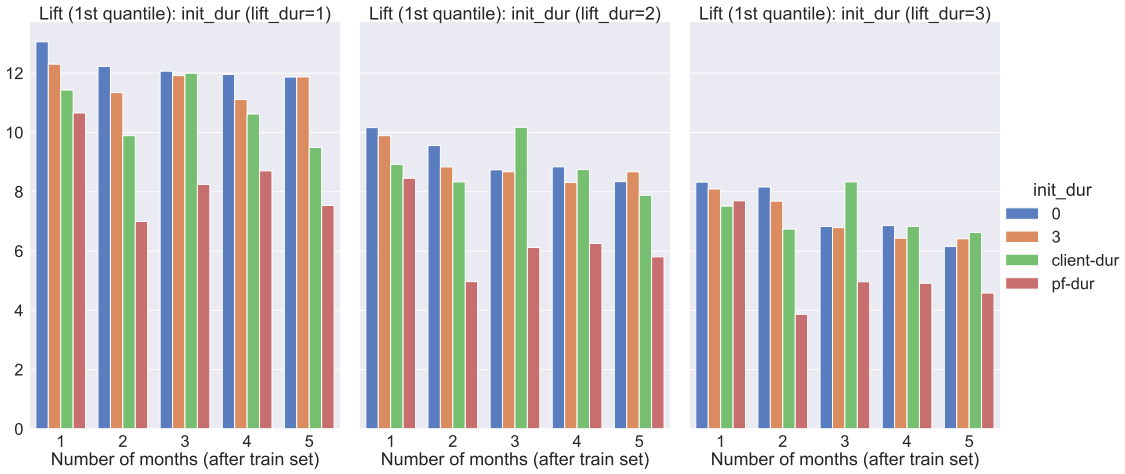
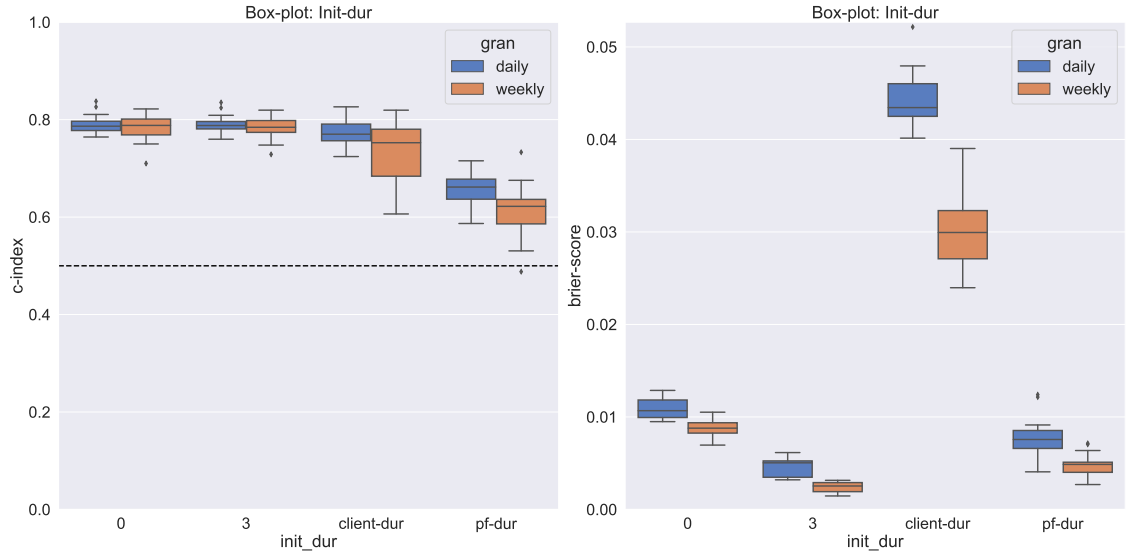


Figure 6.10: Monthly change in Lift (1st quantile): Init-durs (lift-dur).

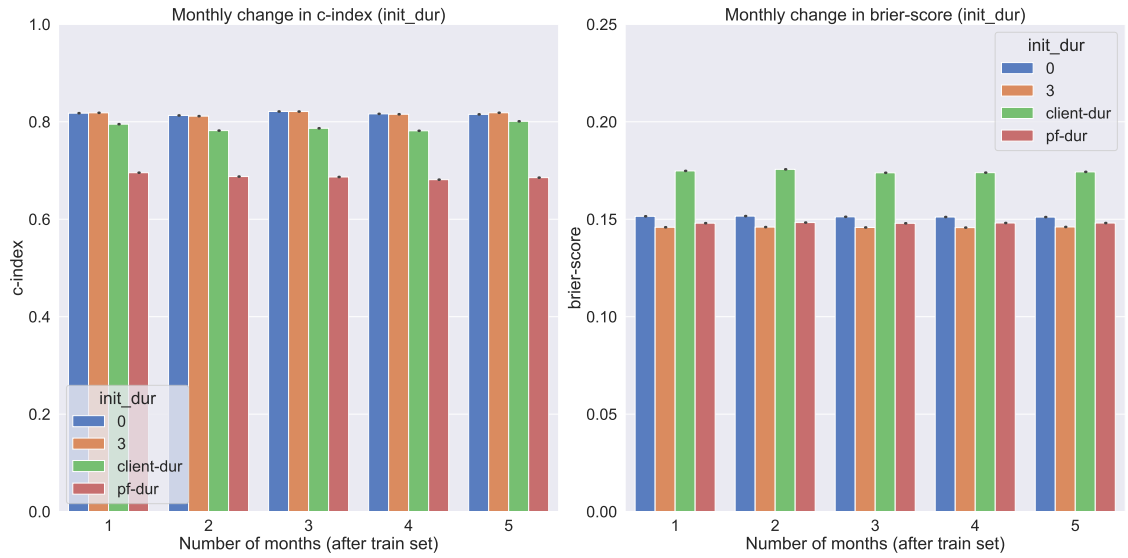
had the best Brier-score. One possible explanation is that, since 'weekly' discretizes churn-dur values to a lesser degree, and seeing that Brier-score is a calibration metric, that is, measures how similar observed event times are compared to the ones predicted by the model, then having less unique event points can be seen as an advantage.

As was the case with the previous experiment, Figure 6.12(a) and Figure 6.12(b) show almost no variation in the subsequent months with regards to the C-score and Brier-score. We can thus conclude that the choice of init-dur has almost no influence on the performance of the model over future test dates for these two metrics.



(a) C-score (b) Brier-score

Figure 6.11: C-score and Brier-score: Init-durs.



(a) C-score (b) Brier-score

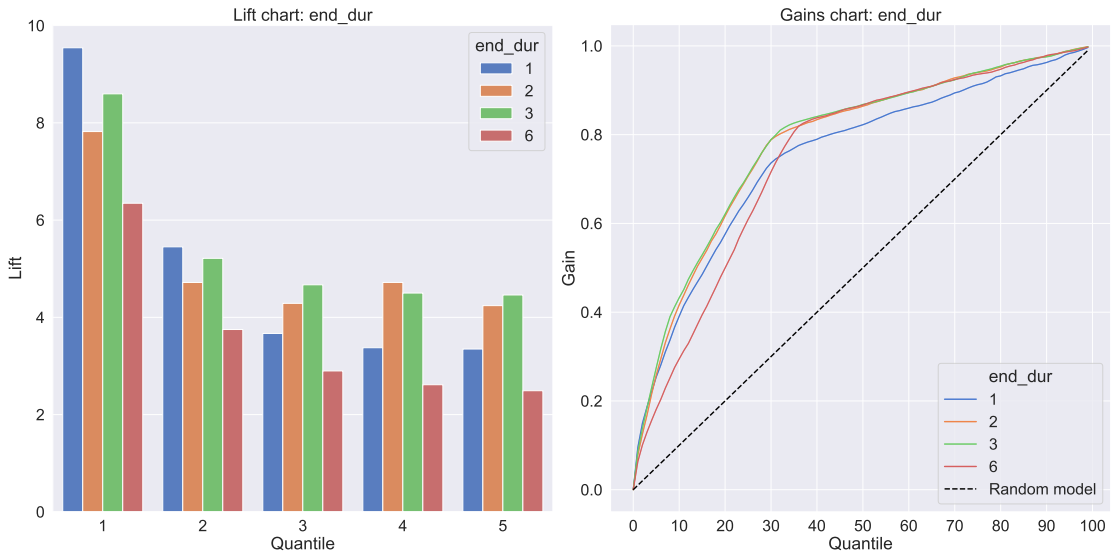
Figure 6.12: Monthly change in C-score and Brier-score: Init-durs.

6.2.2 End-dur comparison

Having evaluated the different values for the beginning of the survival study period with *study.survivalfeats.s1*, let us now assess how different values for the end of this period affect the performance of survival models with *study.survivalfeats.s2*. Unlike the previous DOF, end-dur can only take values, and each value indicate the number of months (after the snapshot), that the survival study period ends. Table 6.4 shows the set of end-dur values being tested in this experiment. The remaining variables have fixed values, and can be seen in Table C.5.

Table 6.4: Survival Targets: *End-durs* values

Survial Feat	**Values**			
end-dur	1	2	3	6



(a) Lift chart

(b) Gains chart

Figure 6.13: Lift and Gain Charts: End-durs.

In both Figure 6.13(a) and Figure 6.13(b) it is possible to see that end-dur with values 1,2 and 3 performed similarly throughout the different quantiles, where end-dur=1 was the best performer in the first quantile. Also, end-dur=6 was the worst performer among all end-dur values. In Figure 6.14 and Figure 6.14 we group the lift and gain charts by different lift-dur values. In these plots we can see, once again, an inverse relationship between lift and gains scores and lift-dur. Furthermore, the more lift-dur increases, higher values of end-dur start to perform better than lower ones.

In Figure 6.15 we grouped lift scores (first quantile) over successive test dates. We can see an interesting pattern: higher end-dur values perform better than lower ones on the first months, but the reverse is observed in later months. Furthermore, we can

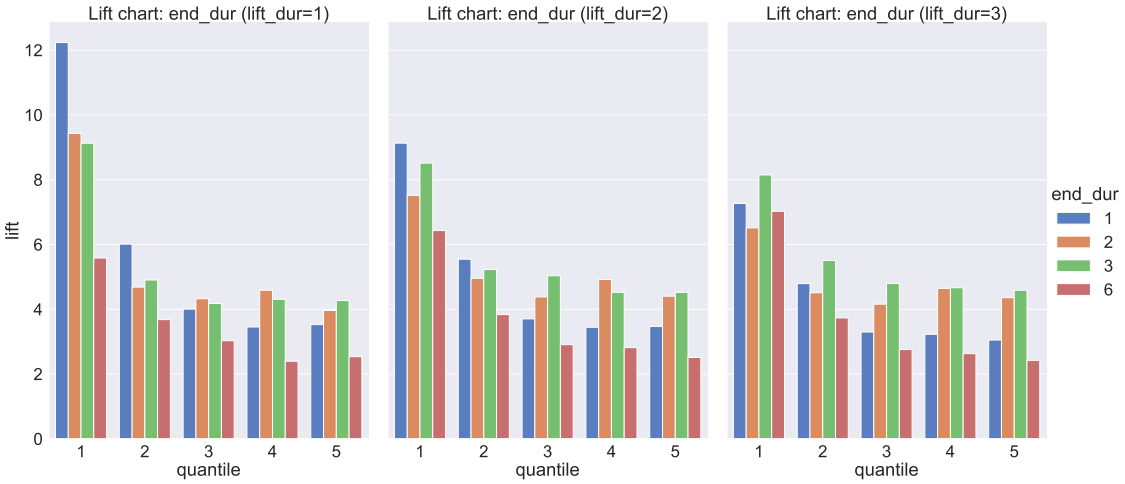


Figure 6.14: Lift chart: End-durs (lift-dur).

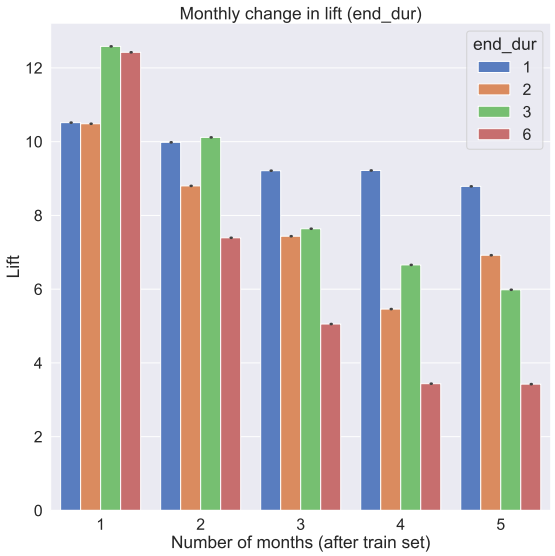
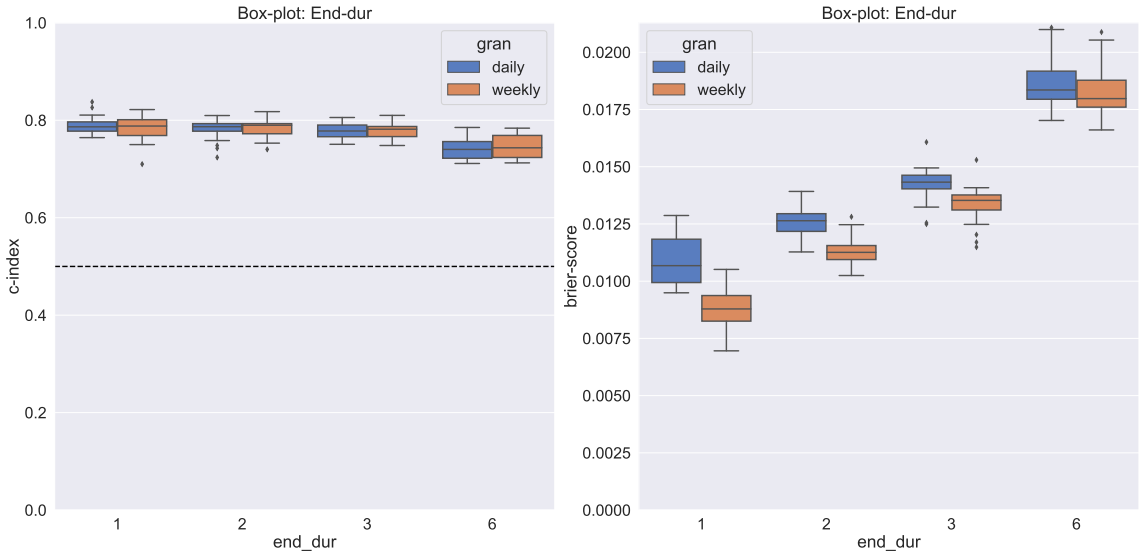


Figure 6.15: Monthly change in Lift (1st quantile): End-durs.

observe that, the greater the end-dur value, the greater the disparity of lift scores over the months. With Figure 6.17, we achieved the same aforementioned conclusions, but now for each lift-dur. In this figure we can also see that, larger end-dur values perform better for greater lift-dur values and vice-versa.

From Figure 6.16(a) and Figure 6.16(b) we observe a decrease in performance for both the C-score and Brier-score in higher end-dur values, specially regarding the Brier-score. The reasoning for this can be the increase in the number of unique event points. It is also important to remember that end-dur is influenced by granularity, that is, there is a direct relationship between the number of unique event points, and end-dur values with increasing granularity. As we saw in the previous experiment, this can have a detrimental influence in some metrics, such as the brier-score. In Figure 6.16(b) we can observe a



(a) C-score

(b) Brier-score

Figure 6.16: C-score and Brier-score: End-durs.

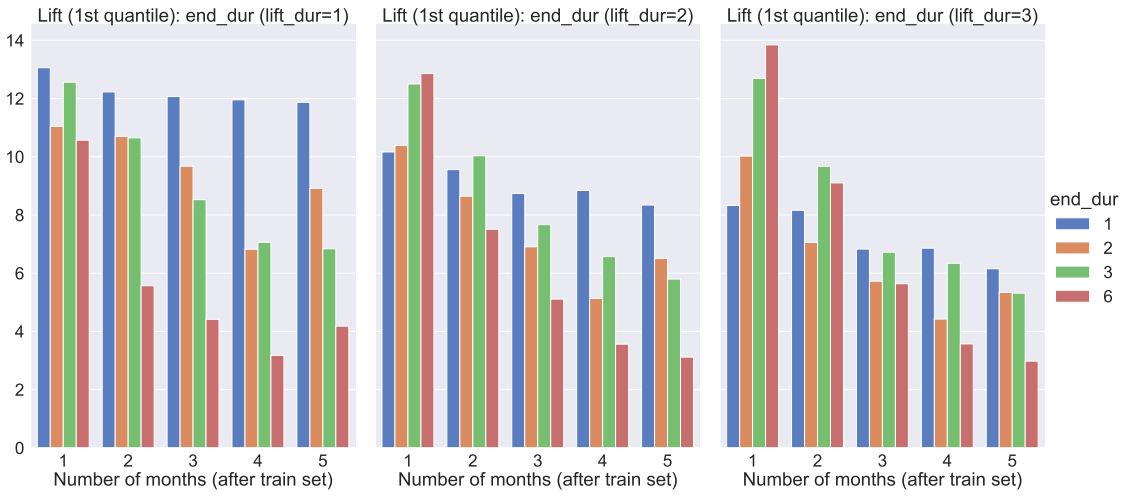


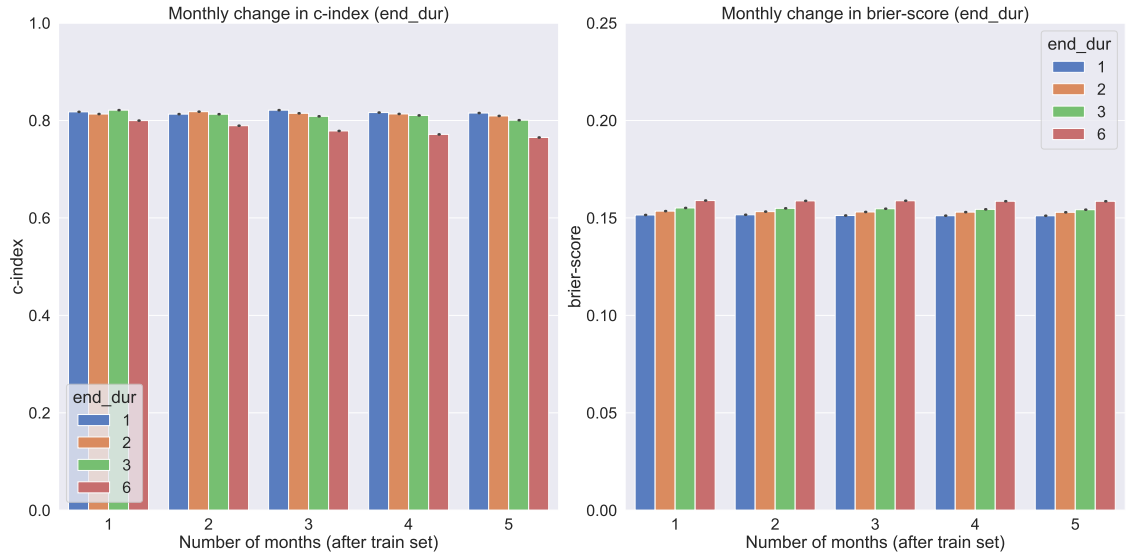
Figure 6.17: Monthly change in Lift (1st quantile): End-durs (lift-dur).

clear decrease in the Brier-score for 'weekly' granularity when compared to 'daily'.

Figure 6.18(a) and Figure 6.18(b) show almost no variation in the subsequent months regarding the C-score and Brier-score. We can thus conclude that the choice of end-dur has no influence, regarding the C-score and Brier-score, in the five following months after the current train-set.

6.2.3 Granularity comparison

Finally, let us discuss the granularity DOF, which is responsible to discretize churn-dur into differently sized time bins. Table 6.5 presents the granularity categories being



(a) C-score

(b) Brier-score

Figure 6.18: Monthly change in C-score and Brier score: End-durs.

tested, and on Table C.6 the fixed variables for the remaining DOF. Since granularity was already evaluated for the C-score and Brier-score in the two previous sections, it will not be compared here.

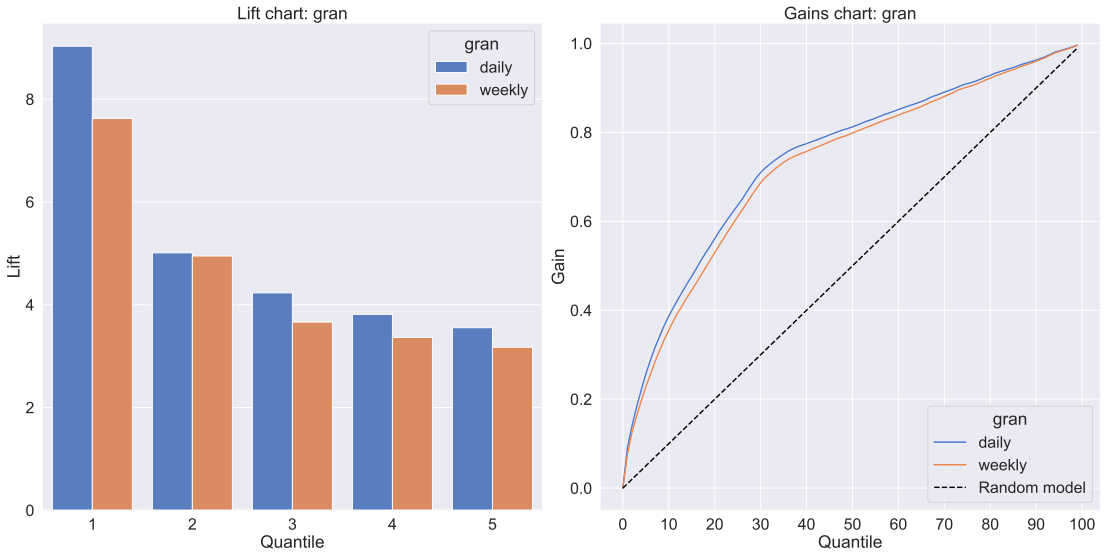
Table 6.5: Survival Targets: granularity values

Survival Feat	**Values**	
granularity	daily	weekly

From Figure 6.19(a) and Figure 6.19(b) we can observe that, overall, 'daily' performs slightly better than 'weekly' on both lift and gain metrics. The same pattern can be see when we compare these metrics across different lift-dur values, as can be seen in Figure 6.20 and Figure 6.20. With this we conclude that using a finer granularity, such as 'daily', results in a slightly better performance. Also, when analyzing the lift score (first quantile) over subsequent months in Figure 6.21, and for different lift-dur values with Figure 6.22, we reach the same conclusion of a small improvement by using a 'daily' granularity over 'weekly'.

6.2.4 Recommendation

Even though we discussed and evaluated each *Survival Target* DOF, in the end, the choice of these values are case dependent for what the data scientist want to analyze. For example, whilst choosing client-dur as the value for init-dur may lead to better results, if one wants to study the survival of customers starting from the end of the binding period (pf-dur), then this should be the init-dur value used instead. Nevertheless, for the best



(a) Lift chart

(b) Gains chart

Figure 6.19: Lift and Gain Charts: Granularity.

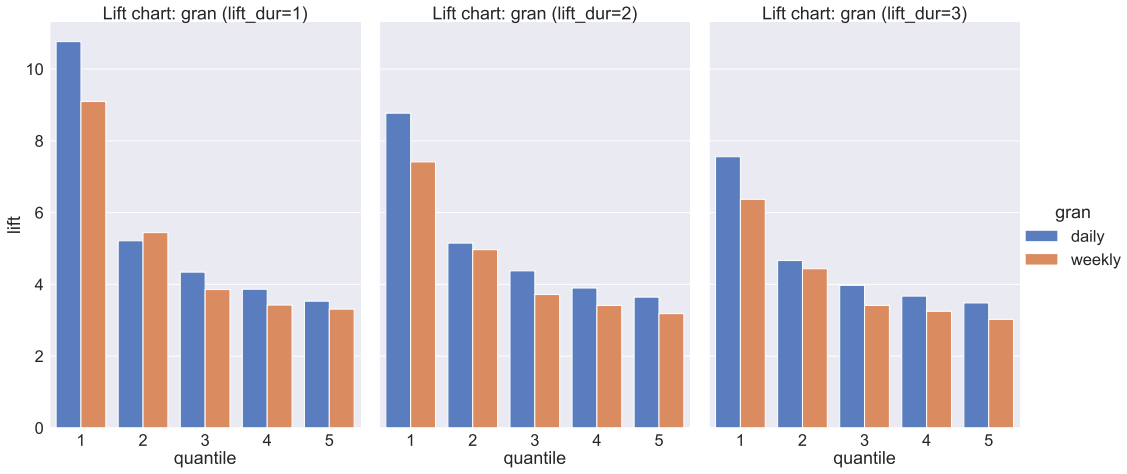


Figure 6.20: Lift chart: Granularity (lift-dur).

performance (both metric and computation wise), it is recommended the use of '3' as the value for init-dur because it not only has a similar performance to '0' regarding the lift, gain and C-score metrics, but it also has the best performance on the Brier-score, meaning that it has better calibrated predicted event times. Next, for end-dur, the recommendation is case dependent, that is, the value should be chosen based on what month we want to compute the lift and gain scores on (lift-dur), and for how many months (test months) we want to use the same survival model. If one wants to prioritize the immediate performance and train a new model each month, then high end-dur values are not recommended, and smaller ones are preferred. In this scenario, the recommendation is to use '1' as the value for end-dur as the performance is the best overall. It also has

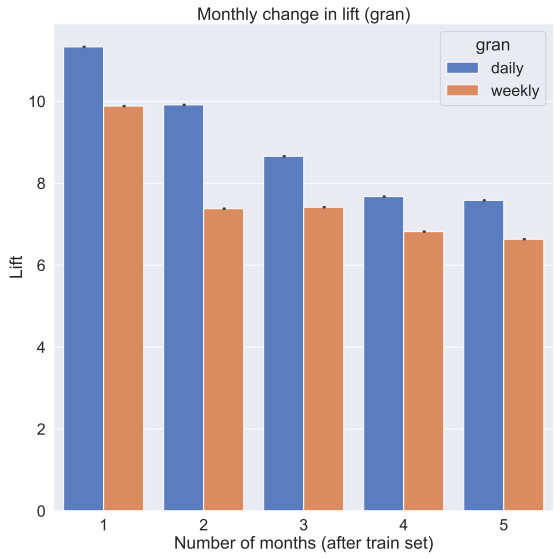


Figure 6.21: Monthly change in Lift (1st quantile): Granularity.

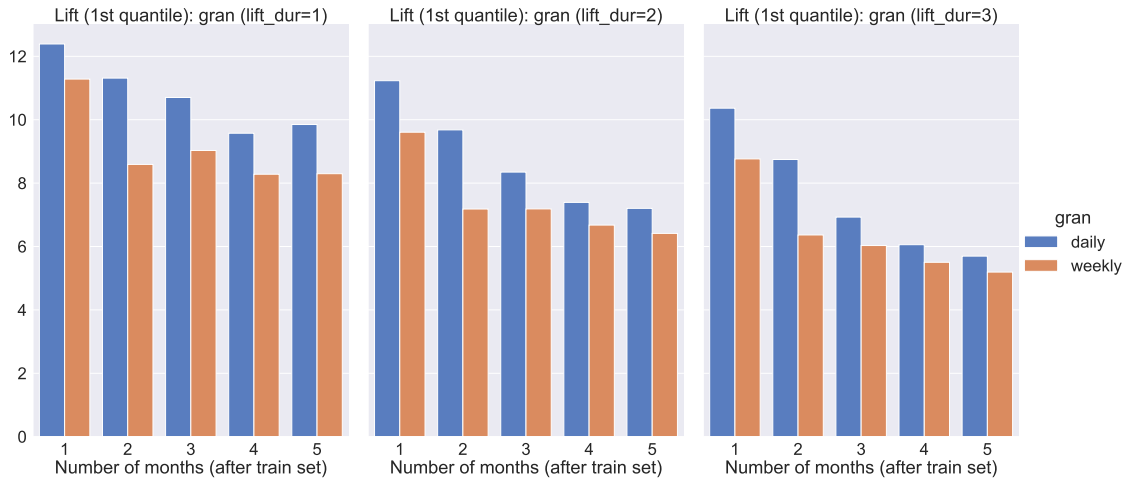


Figure 6.22: Monthly change in Lift (1st quantile): Granularity (lift-dur).

the advantage of being the most stable over successive test sets. This is confirmed by its performance on the C-score and Brier-score metrics, where it was the best performer. And finaly, if we are using '3' and '1' as our init-dur and end-dur values respectively, then we should use a 'daily' granularity. Otherwise we end up with few event points (with 'weekly' we would get a maximum of four events in this configuration), which might not be enough for analyzing the predicted survival curves. Plus, using 'daily' slightly improves the performance of the model. In summary, Table 6.6 shows the recommended values for the different survival target DOF for the general case where we train a new survival month each month.

Table 6.6: Survival Targets: Final recommendation

DOF	Parameter	Value
Survival Features	init-dur	3
Survival Features	end-dur	1
Survival Features	granularity	daily

6.3 Modeling

6.3.1 Hyper-parameter search

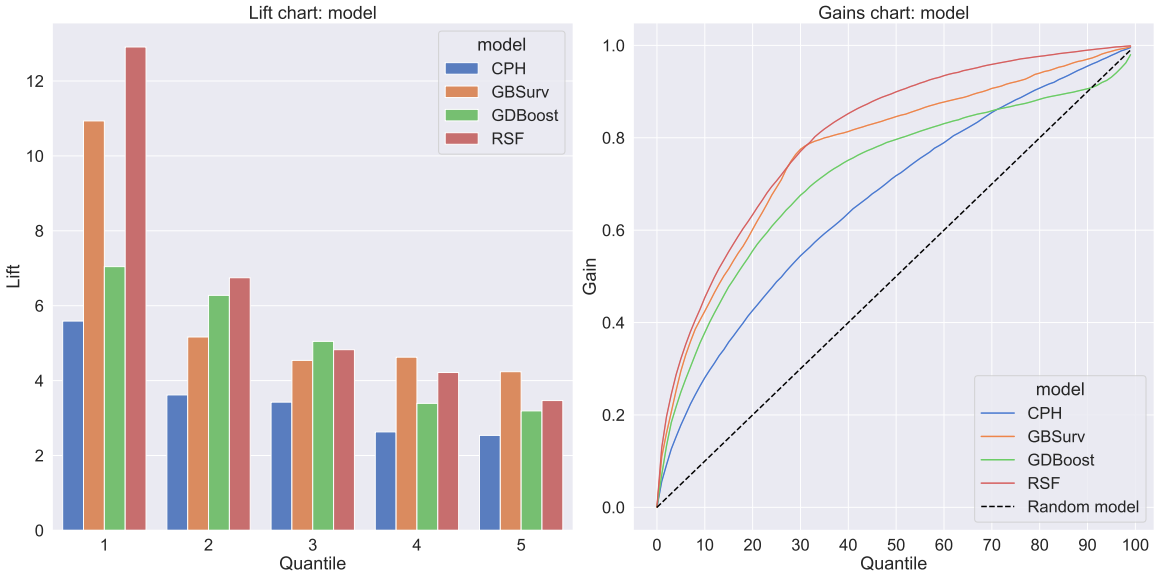
Similarly to *study.sampling.s1*, a hyper-parameter search was done for each model in *study.model.s1*, and the best set of hyper-parameters was chosen to compare the different models. The set of tested hyper-parameters can be consulted in Table C.7. The values for the remaining parameters of this experiment can be viewed in Table C.8. After the experiment finished, the resulting set of metric files was combined and used to produce lift and a gain charts, as well as a C-score and a time-dependent Brier-score plots for each hyper-parameter. These plots can be seen in Appendix C.3. Similarly to *study.sampler.s1*, in order to compute the best values for each hyper-parameter, we rank each of the four metrics, and then sum the resulting ranking values (first-place = 1, second-place = 2, etc). The hyper-parameter value with the lowest score will be chosen as the best value. The set of best hyper-parameters for each model can be seen in Table 6.7.

Table 6.7: Model hyper-parameters (*study.model.s2*)

Model Name	Hyper-Parameter	Best Value
RSF	n_estimators	100
RSF	min_samples_leaf	4
RSF	max_depth	16
RSF	min_samples_split	20
GBSurv	n_estimators	100
GBSurv	learning_rate	0.2
GBSurv	max_depth	2
GBSurv	dropout_rate	0.1
CPH	n_alphas	20
CPH	l1_ratio	0.1
GBoost	n_estimators	100
GBoost	learning_rate	0.2
GBoost	max_depth	16

6.3.2 Model Comparison

After finishing the previous experiment, a new one named *study.model.s2* was used to compare the performance of each model, using the best set of each hyper-parameter from the previous experiment. The values for the remaining parameters of the experiment can be seen in Table C.9.



(a) Lift chart

(b) Gains chart

Figure 6.23: Lift and Gain Charts: Models.

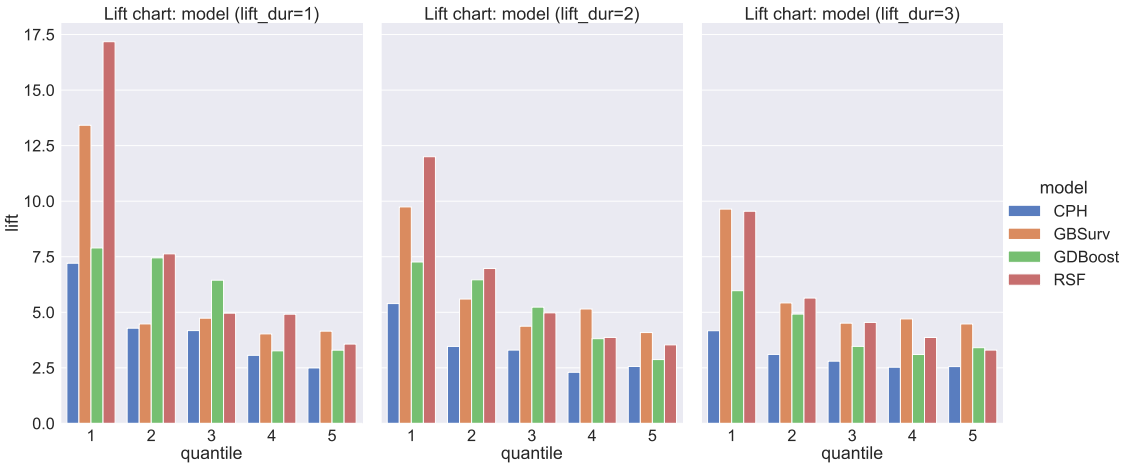


Figure 6.24: Lift chart: Models (lift-dur).

With Figure 6.23(a) and Figure 6.23(a) we can see that both survival models RSF and GBSurv had the best performance regarding the lift and gain metrics when compared to the benchmark model (GBoost) and CPH. The latter is to be expected as CPH is a linear model, and is not able to deal with non-linearities in the data, which the other

three models can. We can observe the same thing if we discretize the lift chart by lift-dur values, as represented in Figure 6.24. We can also conclude that, on average, over every train-test pair, the two ensemble survival models (RSF and GBSurv) performed better than the benchmark model.

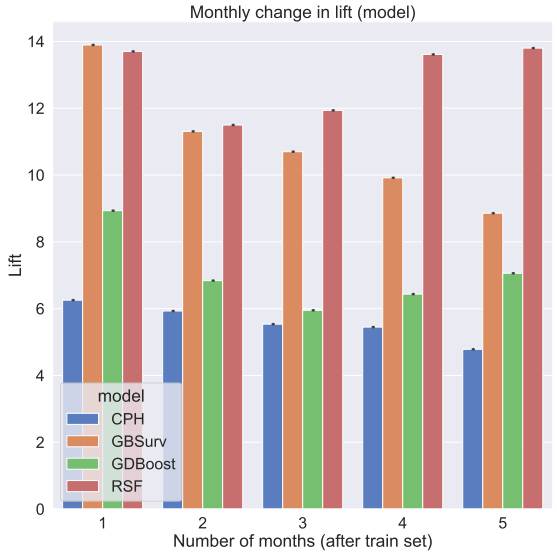


Figure 6.25: Monthly change: Models comparison (Lift chart).

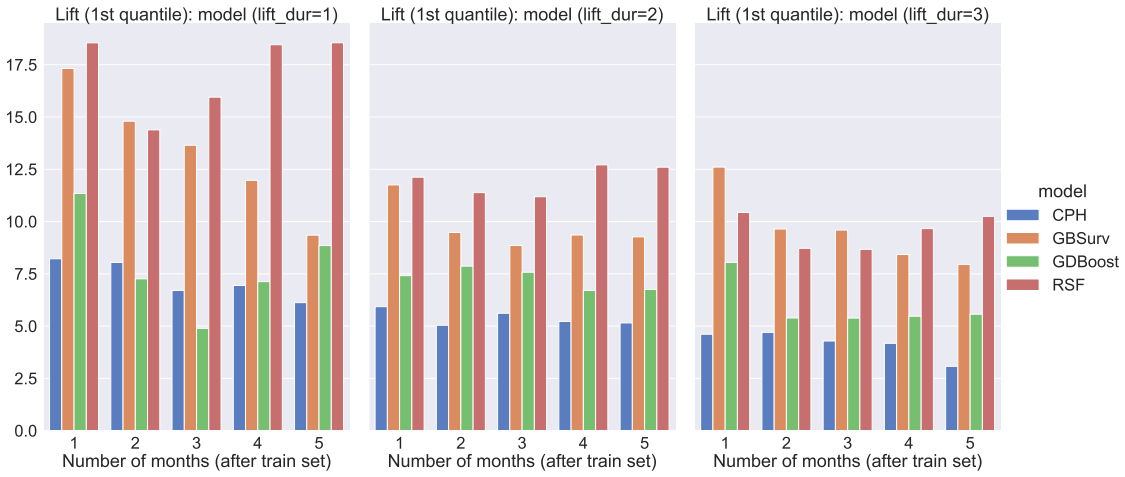
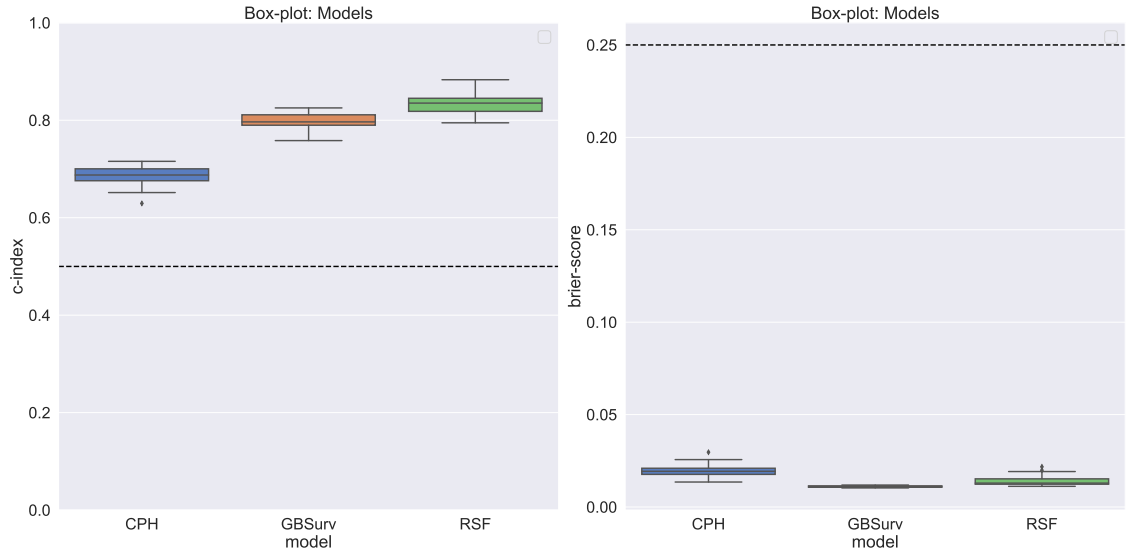


Figure 6.26: Monthly change in Lift (1st quantile): Models (lift-dur).

In Figure 6.25 we evaluate the lift (first quantile) across successive test dates. In this figure and Figure 6.26 we see that the ensemble survival models continue to be the best performers, even across successive test dates. We also observe that there is not much performance decay for this metric over time and across different lift-dur values.

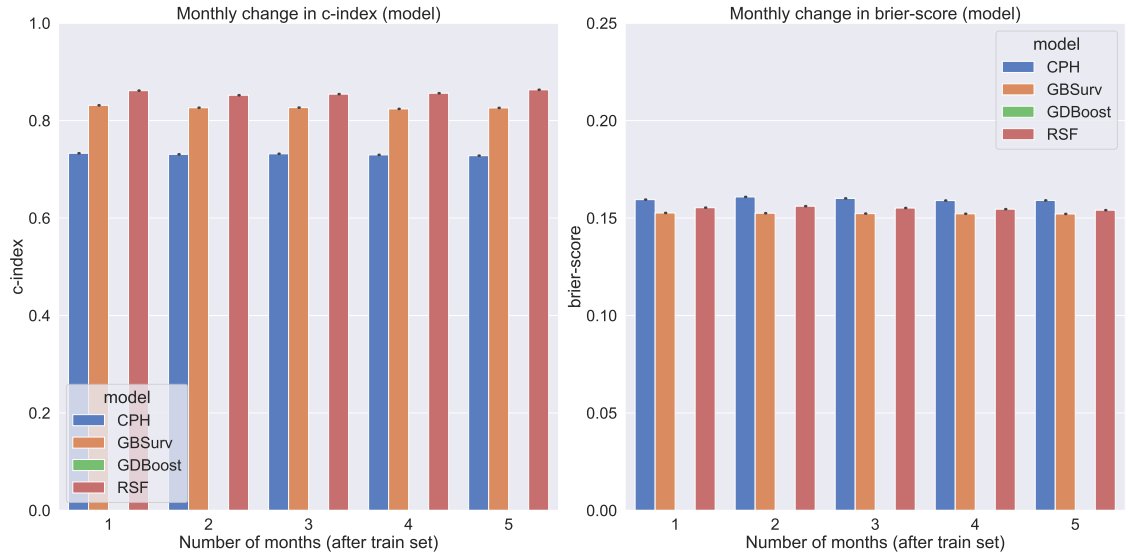
Regarding the C-score and Brier-score, in Figure 6.27(a) and Figure 6.27(b) we see that in the former, the RSF is the best model, which means that it is better able to correctly rank customers by churn risk, however, GBSurv is the best performer on the brier-score metric,



(a) C-score (b) Brier-score

Figure 6.27: C-score and Brier-score: Models.

which means that it is better calibrated regarding predicted event times. As expected, CPH performed the worst in both metrics.



(a) C-score (b) Brier-score

Figure 6.28: Monthly change in C-score and Brier-score: Modeling.

Similarly to the lift score, in Figure 6.28(b) and Figure 6.28(b) we assess how the c-score and brier-score vary across successive test dates (after the train date). In both figures we observe that there is not much variation for both these metrics across the months. As such, we conclude that the choice of model does not have an impact on how

well it performs in later test dates, for these two metrics.

6.3.3 Model computational performance

In the *study.model.s3* experiment we compare the computational cost of each model for a set of different sample sizes, and also it is performance on the different evaluation metrics. Table 6.8 shows the different train sample sizes tested. This experiment was done using the same parameters (and hyper-parameter) values used in the *study.model.s2* experiment.

Table 6.8: Train Sample size values: *study.model.s3*.

Item	Values
Train Sample Size	[1000, 2000, 4000, 6000, 8000, 10000, 20000, 50000, 100000]

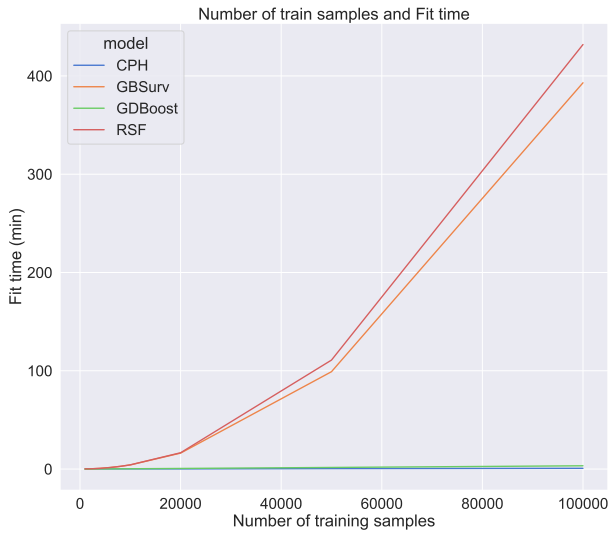
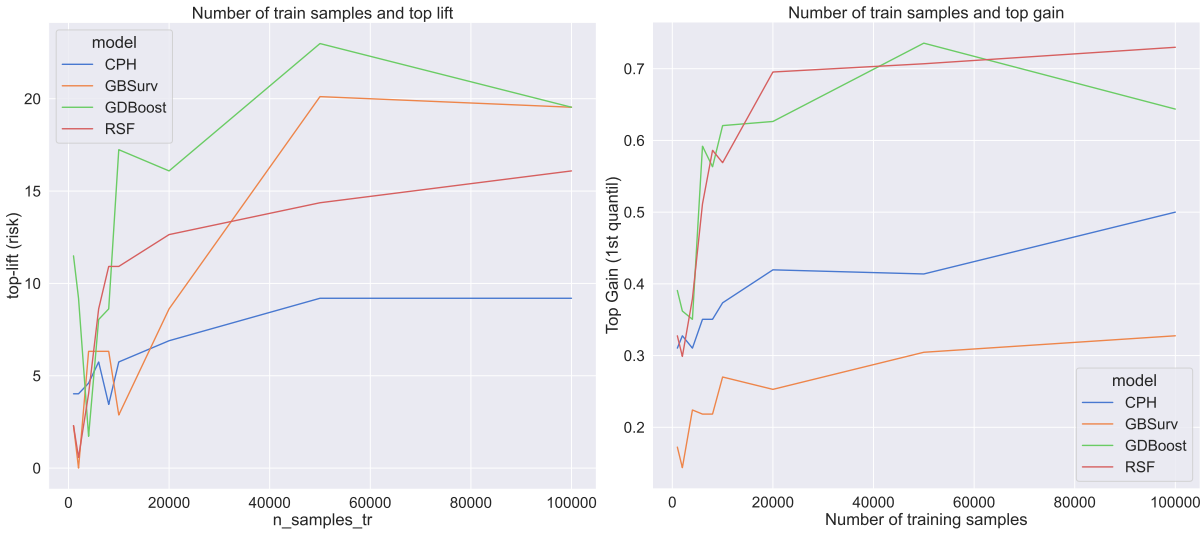


Figure 6.29: Fit-time: Models (number of train samples).

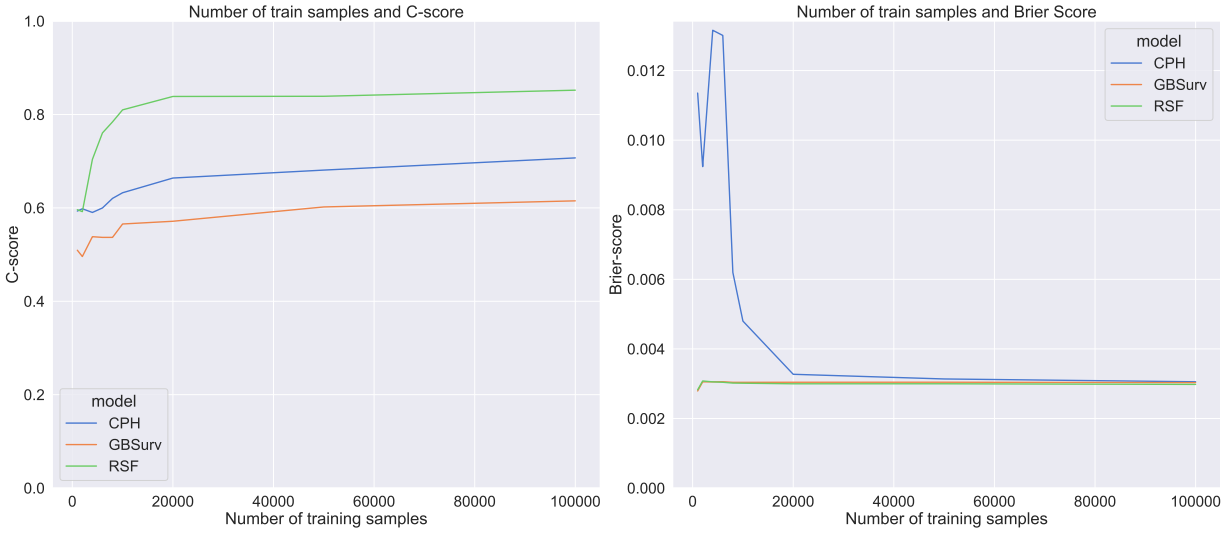
We begin by comparing the computation cost, fit-time, for each model over the different train sample sizes. In Figure 6.29 we can see that both survival ensemble models have the biggest computational cost, by orders of magnitude higher. We can also see that, for the RSF and GBSurv models, the model fit-time scales linearly until a sample size equal to 50,000, but scales quadratically for 100,000 samples.

Regarding the performance metrics across different train sample sizes, in Figure 6.30(a) we can see that both boosting models (GBoost and GBSurv) are the ones that take the longest to achieve the best lift score (first quantile) at around 50,000 train samples. Both CPH and RSF achieve a stable lift score earlier, at around 30,000 samples. Regarding the gain score (1st quantile), this value is achieved much earlier for all the models, at 20,000 train samples. For the C-score, we can see that this metric becomes stable at approximately 20,000 train samples for each model. And for the Brier-score, It is stable for



(a) Lift (b) Gain

Figure 6.30: Lift and Gain (1st quantile): Models (number of train samples).



(a) C-score (b) Brier-score

Figure 6.31: C-score and Brier-score: Models (number of train samples).

both ensemble survival models throughout sample sizes, except for CPH, where it only becomes stable at around 20,000 train samples.

6.3.4 Recommendation

From the previous sections, we can conclude that both survival ensemble models (RSF and GBSurv) performed the best across all metrics, lift-dur values and also across successive test dates. This statement remained truthful even when compared to the benchmark model, thus making these two survival models a good alternative to NOS's pipeline churn model. As expected, the linear survival model CPH did not perform particularly well, as non-linearities are expected. We also saw that the ensemble survival models took orders of magnitude longer to fit the model than the benchmark model, which is something to take into consideration for very large datasets. Nevertheless, since we only use one train-date at a time, plus undersampling, then this scenario does not present as a problem. The recommended model and the respective hyper-parameters can be seen in Table 6.9.

Table 6.9: Modeling: Final recommendation

DOF	Parameter	Hyper-parameter
Models	RSF	n_estimators=100
		min_samples_leaf=4
		max_depth=16
		min_samples_split=20

In Table C.11 of Annex A, a summary of the recommendations for each *DOF* is presented.

6.4 Case study: Churn prevention and Customer retention

In the previous section we evaluated and discussed how the different survival analysis DOF performed compared to each other, mainly in regards to the lift and gain metrics, as these are the most important metrics for NOS, and the only ones where we are able to make a fair comparison with NOS's churn model. Let us now examine how we can leverage survival models to produce business insights regarding customer churn, and use these to improve the company's customer retention strategy. In this study, we used '2019-06-18' as the snapshot date for our train-data and '2019-07-24' for the test-data. We will use the recommended *Class Imbalance* and *Modeling* DOF parameters (with the respective recommended set of hyper-parameters) from the previous sections. Regarding the *Survival Targets* DOF, we will start the survival study in the date of the current snapshot (init-dur=0) and will end it in the next 12 months (end-dur=12). Since we have a wide survival study duration interval, we choose granularity='weekly'. With this survival study duration and granularity, we have a maximum of $12 \times 4 - 1 = 47$ unique event time

points. The list of parameters used can be seen in Table 6.10. It is important to note that, both train and test sets, were cleaned using the same process seen in *Experimental Pipeline*, which was described in Section 5.3.

Table 6.10: Case-study: Survival parameters (Class Imbalance and Modeling)

Parameter	Value
n_dates	1
n_folds	1
lift_durs	[1,2,3]
n_samples_train	[100000]
n_samples_test	[100000]
init_durs	['0']
end_durs	['12']
granularities	['weekly']

Initial Survival analysis

We start our survival analysis by plotting the Kaplan-Meier (KM) estimate for the survival curve of our test-set. This curve gives us an idea of the survival probability, for the average customer, over churn event time points. It is important to remember that this curve only considers churn events and the respective churn durations, and thus does not factor in other covariates in the estimate.

Figure 6.32: Survival Curve: Kaplan-Meier Estimate.

In Figure 6.32 we see that, for the whole study period, the survival probability does not go much below 95%, which confirms a low incidence of customer churn. This also means that NOS retains the large majority of customers in a 12 month period. As stated

before, the KM estimate does not take into consideration other covariates. However, it is possible to segment survival curves in regards to these features. Let us follow the same procedure seen in Section 4.3 and segment customers by client-dur and pf-dur using Eq. 4.1. Before segmenting, we begin by examining the customer and churn distribution for both features.

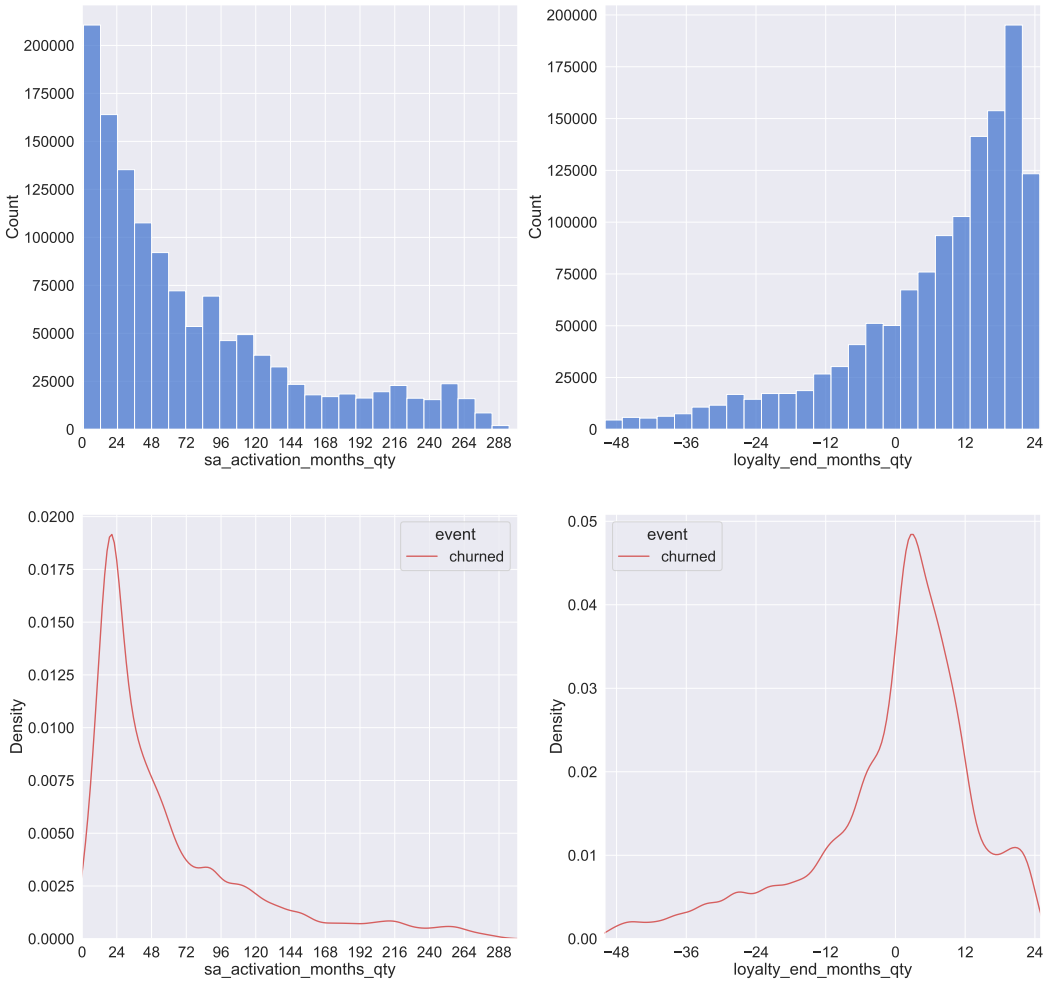
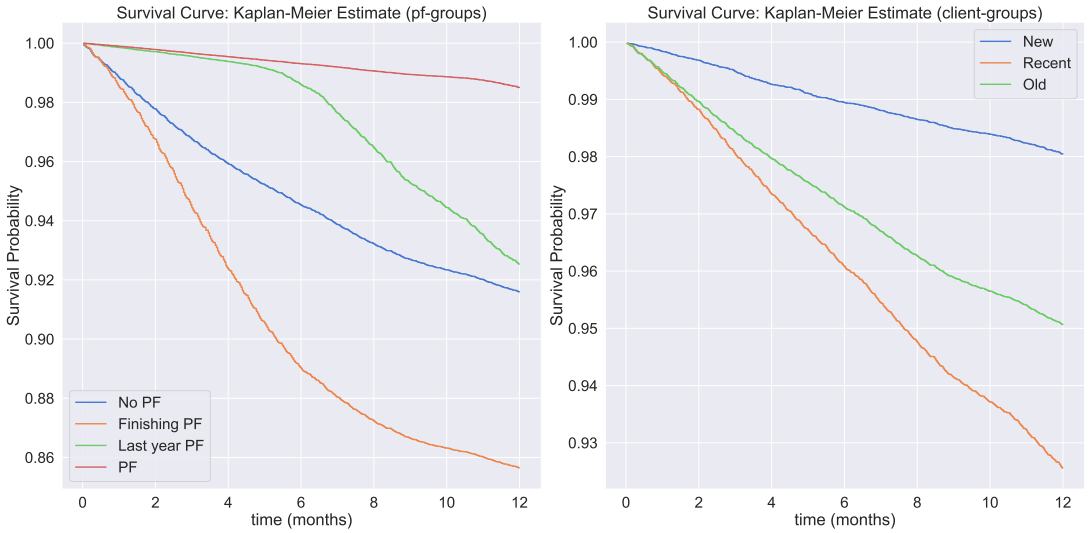


Figure 6.33: Customer distribution (total and churned) for pf-dur and client-dur.

With Figure 6.33, we can see the customer distribution in the two upper plots. In the client-dur (sa_activation_months_qty) plot we observe that the majority of customers belong in the [0, 48] month interval. As for pf-dur (loyalty_end_months_qty), the majority of customers is in the [0, 24] month interval, which coincides with the obligatory binding-period duration by NOS. Regarding the bottom two plots, we can reach the same conclusions as in Figure 4.6, that is, both features have churn peaks in the month where the binding-period ends. For client-dur this equals to month 24, and for pf-dur it is month 0. Now that we know the distributions of each feature and the respective churn distribution, let us segment them and study the resulting survival curves.

In Figure 6.34(a) we observe that customers almost finishing their binding-period



(a) pf-dur groups

(b) client-dur groups

Figure 6.34: Survival Curve: Kaplan-Meier Estimate (pf-dur and client-dur groups).

with NOS (*Finishing PF* group) have the lowest survivability among all the pf-dur groups. This is to be expected as there is a clear peak in the number of churners when the binding-period ends, as we saw in Figure 6.33. Clients in the *PF* group have the highest survivability, which is also to be expected since our survival study ends at month 12, and as such, this group of customers (with pf-dur>12), did not have enough time to finish their binding-period. As for the *Last year PF* group we can see a clear decline in survivability around the 6 months mark. This can be explained by the fact that month 6 coincides with the lower-end pf-dur of this group, that is, there are customers in this group that will no longer have an active binding-period by the end of month 6. In Figure 6.34(b), we can see that the *New* group has the highest survival probability overall. This is due to the fact that this group (client-dur<4) has their binding contract active, thus making it more difficult to churn. The *Recent* group has the lowest survivability, which can be explained by the fact that this group includes customers ending their binding-period.

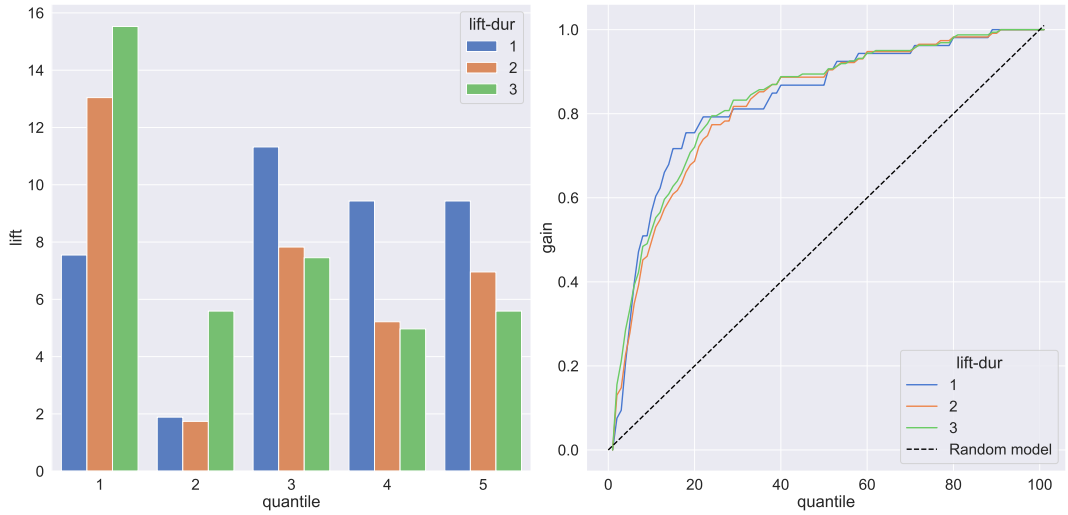
Survival Modeling and Evaluation

Now that we have a basic understanding of the average survivability for our dataset, let us run a survival model to analyze the interaction between the survival targets and the other features in our dataset. For this, we will run an experiment with the recommended *Class Imbalance* and *Modeling* parameters from the previous sections. Table 6.11 presents the chosen parameters. After fitting the survival model, we obtained the survival curves and risk scores for each customer. Let us now evaluate how the model performed.

We begin by evaluating the lift (in the first five quantiles) and the gains metrics. In Figure 6.35(a) we observe an inverse relationship between lift-dur and the lift score in

Table 6.11: Case-study: Survival parameters

Parameter	Value
samplers	RUS: • sampling_strategy = 0.2
models	GBSurv: • learning_rate = 0.2 • max_depth = 2 • dropout_rate = 0.1



(a) Lift (b) Gain

Figure 6.35: Lift and Gain: RSF.

the first quantile. This is to be expected, and a similar pattern was also observed in Section 6.2.2. By comparing these results from the ones from the previous sections, we can again conclude that one should use lower end-dur values to get better performance out of the lift and gains scores for lower lift-dur values. Nevertheless, in this case-study, we are interested in studying the survivability over a longer study period, and thus, lift scores for low lift-dur values are not of interest. Next, we assess the performance of the model with the C-score and Brier-score.

Table 6.12: Case-study: C-score and Integrated Brier-score

Metric	Value
C-score	0.82
Integrated Brier-Score	0.015

In Table 6.12 we see that both the C-score and Brier-score performed better than a random model (0.5 and 0.25 respectively), and also had a similar performance to the RSF

model from Section 6.3.2. Regarding the C-score, having a C-score of 0.82 means that our model correctly predicted 82% of all concordant pairs in our data, that is, it predicted correctly, for 82% of pairs of events, the correct order of events. Even though the C-score is a useful discriminating metric, it cannot assess how well a model is calibrated. For evaluating how well a model is calibrated (and also able to discriminate) we use the Brier-score. In this model we evaluate the Brier score for the event times in the 10% and 90% percentile using the Integrated Brier-score, of which we obtained a score of 0.015, which is a good score when compared to a random model. We can also obtain the Brier-score for each month.

Figure 6.36: Brier-score: Monthly.

In Figure 6.36 we see that there is a direct relationship between the Brier-score and the time (months). This means that the our model predictions become less calibrated the further we move in time.

Survival predictions: Survival Curves

Seeing that our model is well calibrated and validated regarding its discriminatory performance, we can use it to predict survival curves and use these to generate insights about the survivability of customers and the associated client features. Let us begin by plotting the average survival curve, for the average client.

From Figure 6.37 we can see that the survival curve from the Kaplan-Meier estimate is more optimistic than the one from our RSF model. Now that we have a basic idea about the average survivability, we can study the influence of different features on a customer's survival curve. Since we are unable to do this for all the features, we will use only the top four features from the model. Unfortunately, the *scikit-survival* package does not offer a *feature_importance* method for it is classes. Because of this, the author of the package recommends the use of Feature Permuation Importance instead. In this method, for each

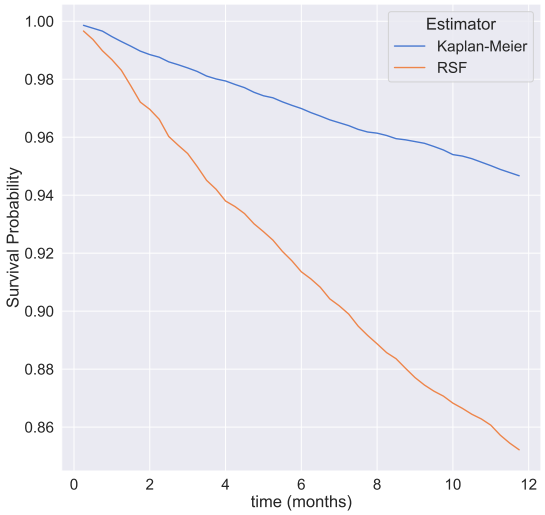


Figure 6.37: Survival Curve: RSF and KM Estimates.

individual feature, we randomly shuffle its values across instances, and then compare the prediction error (or metric) with the one from the original dataset. This means that, if a feature is important, then shuffling will decrease the model's performance. In our case, the metric used will be the C-score.

Table 6.13: Case-study: Top four features (Feature permutation)

Feature	Importance
loyalty_end_months_qty	0.104
last_port_out_req_months_num	0.061
months_to_end_last_pack_promotion_num	0.009
last_contact_insatisfaction_months_num	0.007

From Table 6.13 we can see that pf-dur is the most important feature. This means that if we randomly shuffle it is values, that is, we remove the feature's relationship with survival time, then the score for the test-data drops, on average, by 0.104 points. Feature 'last_port_out_req_months_num' comes slightly behind, and the last two are one order of magnitude less important than the former two features. Table 6.14 shows a description of each feature.

We can now segment each of these four features to check if there are any differences in the survivability of different groups. Since we only have a segmentation for *loyalty_end_months_qty*, the remaining features will be segmented through 1D clustering. These 1D clusters have some nice properties, namely that each will be contained within an interval, and that the intervals for each clusters will be disjoint. We chose the KMeans algorithm for the segmentation task. Since we do not know how many clusters we should have for each feature, we will run the algorithm through 'k' number of clusters (in our case from 1 to 10) and compute the sum of squared distance (SSE) between data points

Table 6.14: Case-study: Top four features description

Feature	Description
loyalty_end_months_qty	Number of months remaining until the end of the binding period
last_port_out_req_months_num	Number of months since the last port-out request (request to change phone numbers)
months_to_end_last_pack_promotion_num	Number of months since the last promotion
last_contact_insatisfaction_months_num	Number of months since the client showed insatisfaction in a commercial outbound

and their respective cluster's centroids. We will then pick the 'k' number of clusters where the SSE starts to flatten. This method is known as the Elbow method.

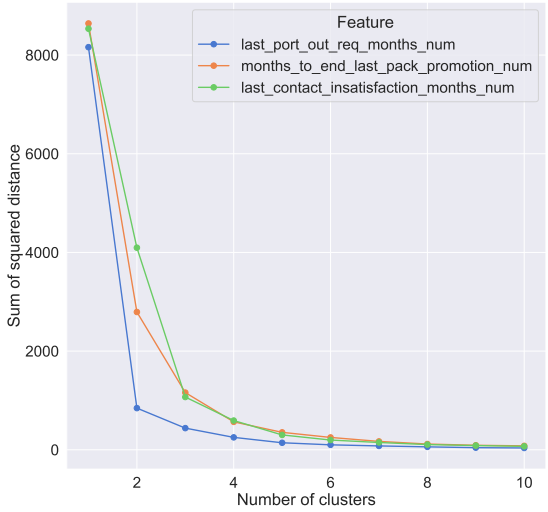


Figure 6.38: Elbow method: Sum of squared distances.

In Figure 6.38 we can see that the SSE curve starts to flatten out at around $k = 4$ clusters for each feature, which means that this will be the number of clusters chosen. In Figure 6.39 we show the clusters generated by KMeans, and the segmentation used for pf-dur. Next, we plot the survival curves for each one of these groups.

From Figure 6.40 we can see that Cluster 3 from *last_port_out_req_months_num* is clearly more at risk for churning than the remaining clusters. This means that the group of customers with a negative value for this feature, that is, customers that will request a new port-out in the feature are much more at risk of churning within the next 12 months. In *months_to_end_last_pack_promotion_num* we can see that clusters 2 and 4 are more at risk, specially cluster 2. This means that having more than 10 month since the last promotion contact drastically increases churn risk. From *last_contact_insatisfaction_months_num*

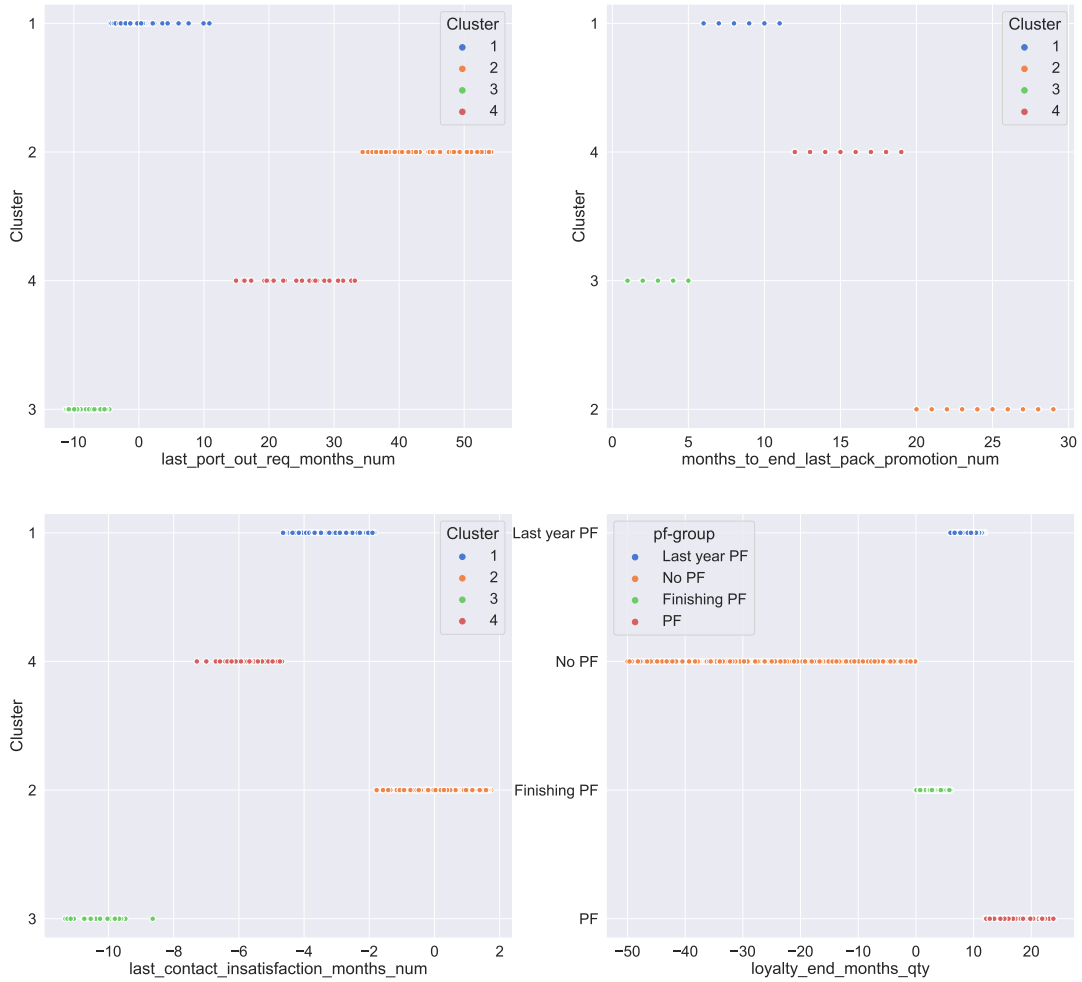


Figure 6.39: Clusters: KMeans (k=4) for top four features.

we clearly see that customers from cluster 2 are more at risk for churning. In this case, this means that customers who made an outbound in-satisfaction contact within the prior 2 months are significantly more at risk of churning. And finally, with *loyalty_end_months_qty* we conclude the same thing as the previous section, in that customers that are much closer to ending their binding period (*Finishing PF* group) are more at risk of churning.

Censored Customers: Clients lost

Now we will focus our attention on customers that are still alive, and assess how many of them are going to churn within the survival study period. In order to achieve this, we filtered customers by those who have not churned yet (event=0), and used the predicted survival curves to get the survival probabilities for different monthly events. With these probabilities, we can estimate the lifetime of a customer by setting a probability threshold for which we consider a customer churned. Since we do not know the ideal value for this threshold, we ran this experiment with several thresholds.

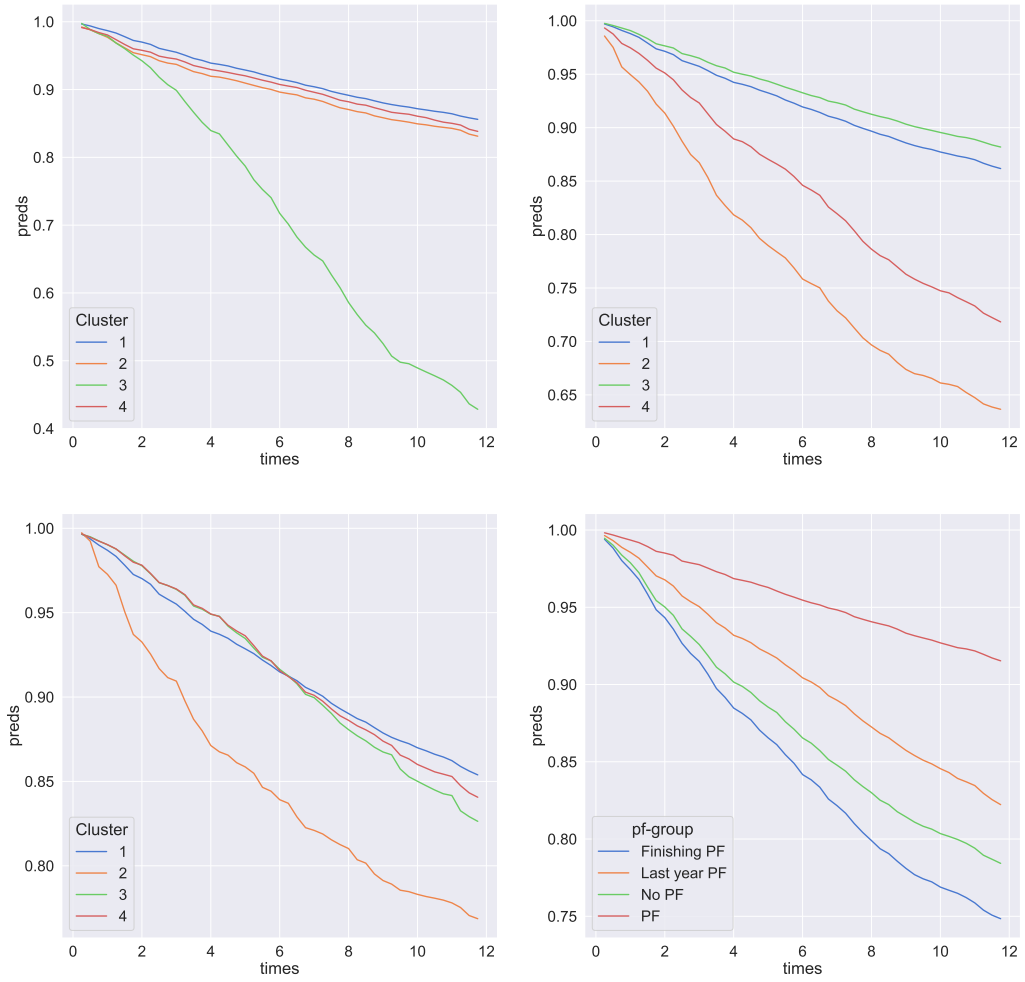


Figure 6.40: Survival Curve: Top four features (with clusters).

Table 6.15: Number of customers lost: Monthly time windows

1m	3m	6m	9m	proba-threshold
0	0	0	12	0.5
0	0	3	99	0.6
0	0	100	486	0.7
0	46	673	2022	0.8

In Figure 6.41 we can see that there is a direct relationship between the churn month window and the number of churned customers. As expected, we can also see an increase in the number of churned customers the more we increase the probability threshold. One can use these figures and then decide a probability threshold to use. Chosen a threshold, we can view the names of the customers, in each month window, ranked by survival probability, and use this information to define better retention strategies for the most at risk customers.

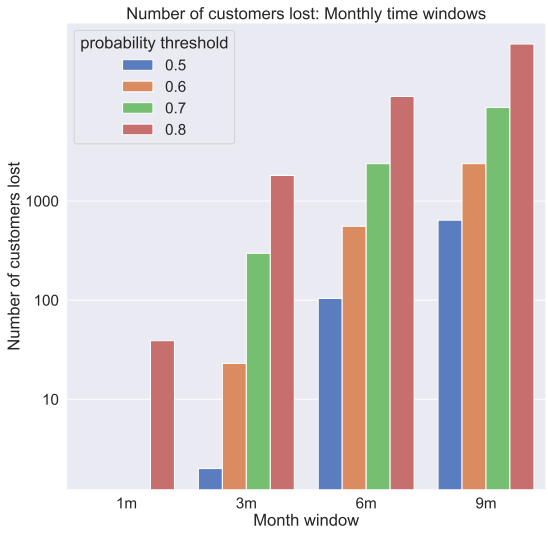


Figure 6.41: Case-study: Number of customers lost (Monthly time windows).

Censored Customers: Expected Loss

Another use for these survival curves is to identify which censored customers (event=0) are the most expensive to lose over the next 12 months, if we were to lose them on the predicted churn event. In order to compute these, we generated random customerIDs and random Monthly charges for each customer. In this experiment, we consider a customer to be churned if it is survival probability drops below 0.5. To compute a customer's *Expected loss* we compute how much they are worth over the next 12 months (12×MonthlyCharges) and subtract the revenue we get until the customer churns (ExpectedChurn×MonthlyCharges).

Table 6.16: Case-study: Expected Loss

CustomerID	MonthlyCharges	Expected Churn dur	Expected loss
5092-YELV	122.76	11.5	1411.74
8759-KCYS	122.71	11.5	1411.16
4449-WHCF	122.09	11.5	1404.04
9342-BZRB	124.36	11.25	1399.05
1536-KQYB	123.32	11.25	1387.35

Table 6.16 shows the top5 customers in regards to their expected loss. With this table we can, for instance, define strategies that factor in how expensive it is to retain a high-value customer (e.g. one with a higher expected loss), and the revenue lost by losing said customer to voluntary churn.

7

Conclusion and Future work

In this dissertation we studied the application of survival methodologies in a Portuguese telecom company, in the topic of churn prediction. As stated before, the use of survival methods allows for the prediction of not only the event, but also the time-to-event, which is not possible in normal regression algorithms due to the presence of censored instances. However, using survival analysis in this context poses several problems, the first being the low incidence of churners, leading to a highly imbalanced dataset. Another complication was the choice of the survival duration, that is, the difference in time between the start of the survival study and its end, which as was shown during this dissertation, can have different impacts on the performance metrics depending on the objective of the survival study.
In order to automate this process, an end-to-end experimental pipeline was developed to run survival experiments. This pipeline is composed of steps from data cleaning to evaluation, and was thoroughly discussed. Several experiments were constructed to assess how the different DOF parameters impacted both the metric performance and computational performance of the survival models. From these, recommendations were made for each DOF in order to better use and understand this type of models.
In the end, a case-study was presented with the objective of showing potential key insights offered by survival models. Key insights included the survival probabilities for the average customer over a year, survival probabilities for different groups in the most important features (by feature permutation importance), number of non-censored customers lost by each month of the survival study, and the expected loss of revenue by customer churn.

Future work

Since an experimental pipeline was developed for survival analysis, one can take advantage of it and experiment with different types of survival modeling techniques, or any other DOF. Among these, the following stand out:

- **Using Time-dependent covariates**. The data used in this dissertation is time-series

in nature, and as such, in order to take full advantage of covariate variation between different time-points, a survival approach able to model these variations would be interesting to experiment with. These models can then be used in the experimental pipeline and evaluated against the already implemented models.

- **Survival Explainability**. Even though survival models demonstrate good performance when compared to statistical models or classical machine learning models, they present themselves as black boxes, specially in the case of ensemble models. In lieu of this, there have been some developments in regards to explainability techniques for survival analysis, namely SurvShap [21]. This model-agnostic explainability technique uses Shapley values to explain the survival functions predicted by the survival models. These explanations can then support domain experts, without the necessary machine learning knowledge, in making decisions based on the predictions from these models.

- **Scaling for big data**. Even though much of the data preparation step was made possible by distributed data processing frameworks, such as Spark, the remainder of this process, including class balancing, modeling and evaluation was done locally. In order to allow for bigger models to be trained, specially if one wants to model time-dependent covariates, distributing the computation for modeling is necessary.

Bibliography

[1] S. Barua, M. Islam, K. Murase, et al. "ProWSyn: Proximity weighted synthetic oversampling technique for imbalanced data set learning". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2013, pp. 317–328 (cit. on p. 6).

[2] L. Breiman. "Random Forests". en. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125, 1573-0565. (Visited on 03/09/2014) (cit. on pp. 4, 15).

[3] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap. "DBSMOTE: density-based synthetic minority over-sampling technique". In: *Applied Intelligence* 36.3 (2012), pp. 664–684 (cit. on p. 6).

[4] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap. "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2009, pp. 475–482 (cit. on p. 6).

[5] J. Burez and D. Van den Poel. "Handling class imbalance in customer churn prediction". In: *Expert Systems with Applications* 36.3 (2009), pp. 4626–4636 (cit. on pp. 5, 9).

[6] N. V. Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357 (cit. on pp. 6, 10).

[7] M. Cleves et al. *An introduction to survival analysis using Stata*. Stata press, 2008 (cit. on pp. 12, 13).

[8] G. Douzas and F. Bacao. "Effective data generation for imbalanced learning using conditional generative adversarial networks". In: *Expert Systems with applications* 91 (2018), pp. 464–471 (cit. on p. 6).

[9] J. H. Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232 (cit. on p. 16).

[10] T. A. Gerds and M. Schumacher. "Consistent estimation of the expected Brier score in general survival models with right-censored event times". In: *Biometrical Journal* 48.6 (2006), pp. 1029–1040 (cit. on p. 3).

[11] E. Giunchiglia, A. Nemchenko, and M. v. d. Schaar. "RNN-SURV: A deep recurrent model for survival analysis". In: *International Conference on Artificial Neural Networks*. Springer. 2018, pp. 23–32 (cit. on p. 5).

[12] E. Graf et al. "Assessment and comparison of prognostic classification schemes for survival data". In: *Statistics in medicine* 18.17-18 (1999), pp. 2529–2545 (cit. on p. 9).

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001 (cit. on p. 4).

[14] M. Havrylovych and N. Kuznietsova. "Survival analysis methods for churn prevention in telecommunications industry". In: *CEUR Workshop Proceeding*. Vol. 2577. 2020, pp. 47–58 (cit. on p. 4).

[15] T. I. *Two Modifications of CNN*. 1976 (cit. on pp. 6, 10).

[16] H. Ishwaran et al. "Random survival forests". In: *The annals of applied statistics* 2.3 (2008), pp. 841–860 (cit. on pp. 5, 15).

[17] J. L. Katzman et al. "DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network". In: *BMC medical research methodology* 18.1 (2018), pp. 1–12 (cit. on p. 5).

[18] S. Kentritas. *Customer Relationship Management: The SAS Perspective* (cit. on p. 1).

[19] D. G. Kleinbaum, M. Klein, et al. *Survival analysis: a self-learning text*. Vol. 3. Springer, 2012 (cit. on pp. 4, 11, 12).

[20] G. Kovács. "An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets". In: *Applied Soft Computing* 83 (2019), p. 105662 (cit. on p. 6).

[21] M. Krzyziński et al. "SurvSHAP (t): Time-dependent explanations of machine learning survival models". In: *arXiv preprint arXiv:2208.11080* (2022) (cit. on p. 67).

[22] H. Kvamme, Ø. Borgan, and I. Scheel. "Time-to-event prediction with neural networks and Cox regression". In: *arXiv preprint arXiv:1907.00825* (2019) (cit. on p. 5).

[23] C. Lee et al. "Deephit: A deep learning approach to survival analysis with competing risks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018 (cit. on p. 5).

[24] G. Lemaître, F. Nogueira, and C. K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning". In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5. URL: http://jmlr.org/papers/v18/16-365.html (cit. on p. 29).

[25] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[26] J. Lu. "Predicting customer churn in the telecommunications industry—-An application of survival analysis modeling using SAS". In: *SAS User Group International (SUGI27) Online Proceedings*. Vol. 114. 2002 (cit. on p. 4).

[27] S. Y. Park et al. "Review of statistical methods for evaluating the performance of survival or other time-to-event prediction models (from conventional to deep learning approaches)". In: *Korean Journal of Radiology* 22.10 (2021), p. 1697 (cit. on p. 3).

[28] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 31).

[29] S. Pölsterl. "scikit-survival: A Library for Time-to-Event Analysis Built on Top of scikit-learn". In: *Journal of Machine Learning Research* 21.212 (2020), pp. 1–6. URL: http://jmlr.org/papers/v21/20-729.html (cit. on p. 30).

[30] C. D. R. "Regression Models and Life Tables". In: *Journal of the Royal Statistic Society* B.34 (1972), pp. 187–202 (cit. on pp. 4, 13).

[31] Y. Tillé. "Ten years of balanced sampling with the cube method: an appraisal". In: *Survey methodology* 37.2 (2011), pp. 215–226 (cit. on p. 5).

[32] H. Uno et al. "On the C-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data". In: *Statistics in medicine* 30.10 (2011), pp. 1105–1117 (cit. on pp. 3, 7).

[33] M. Vuk and T. Curk. "ROC curve, lift chart and calibration plot". In: *Metodoloski zvezki* 3.1 (2006), p. 89 (cit. on p. 4).

[34] P. Wang, Y. Li, and C. K. Reddy. "Machine learning for survival analysis: A survey". In: *ACM Computing Surveys (CSUR)* 51.6 (2019), pp. 1–36 (cit. on pp. 7, 8, 13).

[35] G. M. Weiss. "Mining with rarity: a unifying framework". In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), pp. 7–19 (cit. on p. 5).

[36] M. Zeng et al. "Effective prediction of three common diseases by combining SMOTE with Tomek links technique for imbalanced medical data". In: *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*. IEEE. 2016, pp. 225–228 (cit. on p. 6).

[37] Hue. *Hue documentation*. https://docs.gethue.com/. Accessed: 2022-02-04 (cit. on p. 18).

[38] José Ferreira, Dianne Gomes, BCG. *O valor da Fidelização para o consumidor e o mercado de Telecomunicações em Portugal*. https://www.bcg.com/publications/2021/fidelizacao-mercadotelco. Accessed: 2022-02-04 (cit. on p. 1).

[39] Matt Mansfield, Small Business Trends. *CUSTOMER RETENTION STATISTICS – The Ultimate Collection for Small Business*. https://smallbiztrends.com/2016/10/customer-retention-statistics.html. Accessed: 2022-02-04 (cit. on pp. 1, 2).

[40] Observador. *Queixas à DECO sobem 16% e as das telecomunicações aumentam 30% em 2020 e lideram há 13 anos*. https://observador.pt/2021/02/03/queixas-a-deco-sobem-16-e-as-das-telecomunicacoes-aumentam-30-em-2020-e-lideram-ha-13-anos. Accessed: 2022-02-04 (cit. on p. 1).

[41] OmniSci Team. *Strategies for Reducing Churn Rate in the Telecom Industry*. https://www.omnisci.com/blog/strategies-for-reducing-churn-rate-in-the-telecom-industry. Accessed: 2022-02-04 (cit. on p. 20).

[42] pySpark. *PySpark Documentation*. https://spark.apache.org/docs/latest/api/python/. Accessed: 2022-02-04 (cit. on pp. 18, 31).

A

Appendix 1

Table A.1: Variable Group Description

Data group	Number of features in group	Description
Agente	8	Information related to seller of service
ARPUT	69	Service description (e.g. number of internet cards, sportv subscription)
Campanhas	8	Selling campaigns
Chaves	3	Chaves: Snapshot date and client code
Churn	12	Churn: Churn information (e.g. Next voluntary churn date)
Cliente	29	Information about the client e.g. Age and PF status)
Consumos	260	Client service usage
Contactos	4	Client enquiries about PF status and payment
Dunning	9	Communication with customers regarding their collections
Faturacao	76	Billing information
Participacao	94	Information related with problems reported by the client
Portabilidade	4	Information about port out requests
Promocoes	4	Information related to promotions given to the client
Rede e concorrencia	42	Information about competitors
Retencao	14	Information about client retention programs

B

Appendix 2

Table B.1: Class Imbalance and model class naming

Name	Class Name
Random Undersampling	imblearn.under_sampling.RandomUnderSampler
Random Oversampling	imblearn.over_sampling.RandomOverSampler
Tomek-Links	imblearn.under_sampling.TomekLinks
SMOTE-NC	imblearn.over_sampling.SMOTENC
Cox Proportional Hazards	sksurv.linear_model.CoxnetSurvivalAnalysis
Random Survival Forests	sksurv.ensemble.RandomSurvivalForest
Gradient Boosting Survival Analysis	sksurv.ensemble.GradientBoostingSurvivalAnalysis
Gradient Boosting Trees	sklearn.ensemble.GradientBoostingClassifier

C

Appendix 3

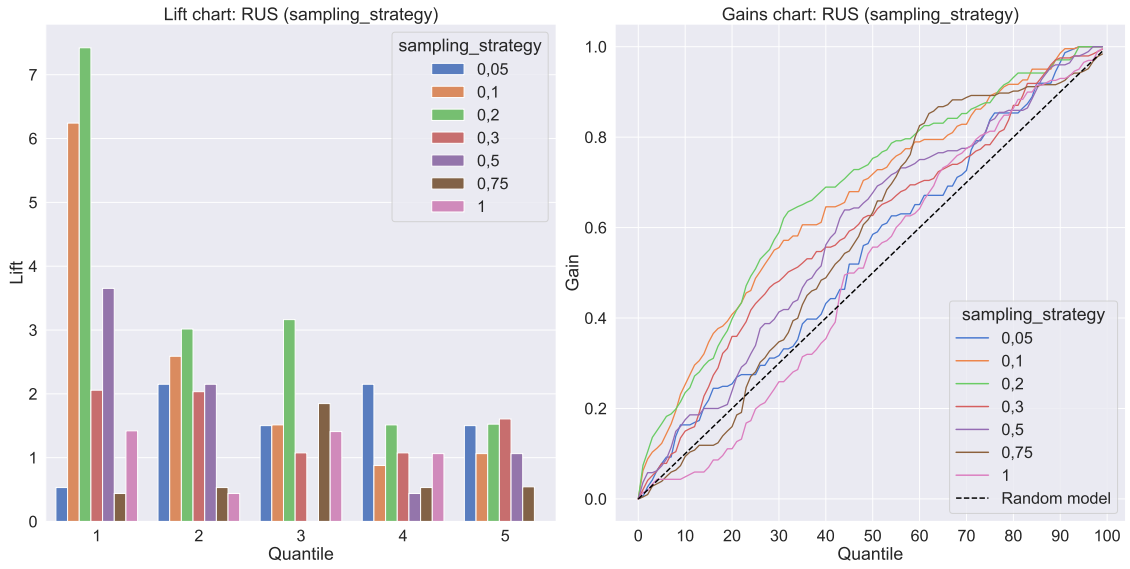
C.1 Class Imbalance

Table C.1: *study.sampler.s1*: Class imbalance hyper-parameter search parameters

Sampler Name	Hyper-Parameter Space
RUS	sampling_strategy: [0.05, 0.1, 0.2, 0.3, 0.5, 0.75, 1]
ROS	sampling_strategy: [0.05, 0.1, 0.2, 0.3, 0.5, 0.75, 1]
SMOTE-NC	sampling_strategy: [0.05, 0.1, 0.2, 0.3, 0.5, 0.75, 1]
SMOTE-NC	k_neighbors: [1, 2, 3]

Table C.2: *study.sampler.s1*: Fixed parameters

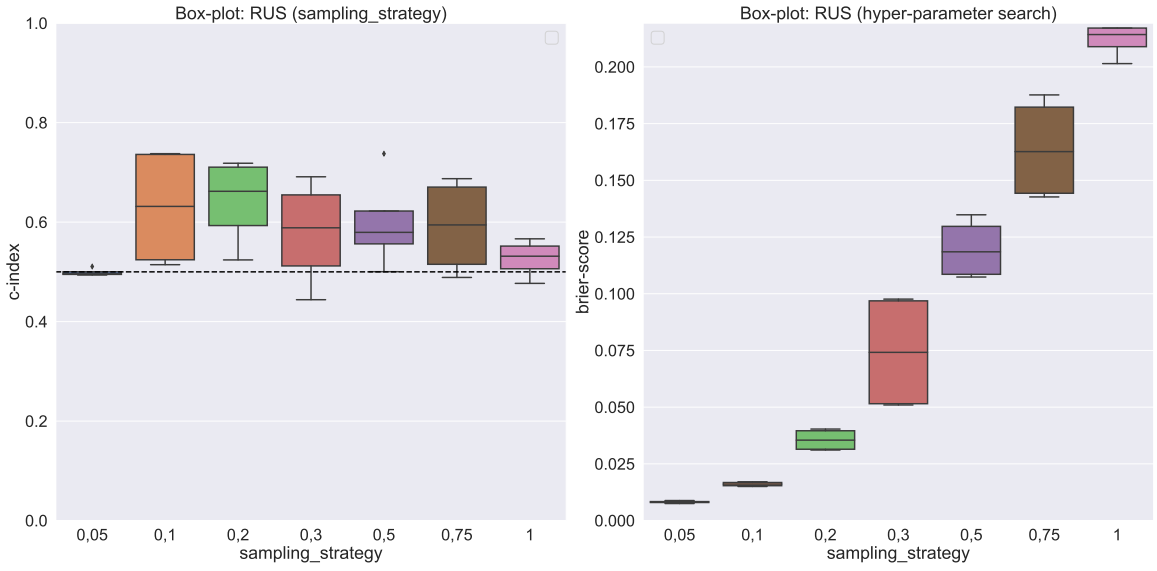
Parameter	Value
n_dates	1
n_folds	100
lift_durs	[1]
n_samples_train	[5000]
n_samples_test	[10000]
init_durs	['sa_activation_months_qty']
end_durs	[1]
granularities	['weekly']
models	GBSurv: • learning_rate = 0.2 • max_depth = 2 • dropout_rate = 0.1



(a) Lift chart

(b) Gains chart

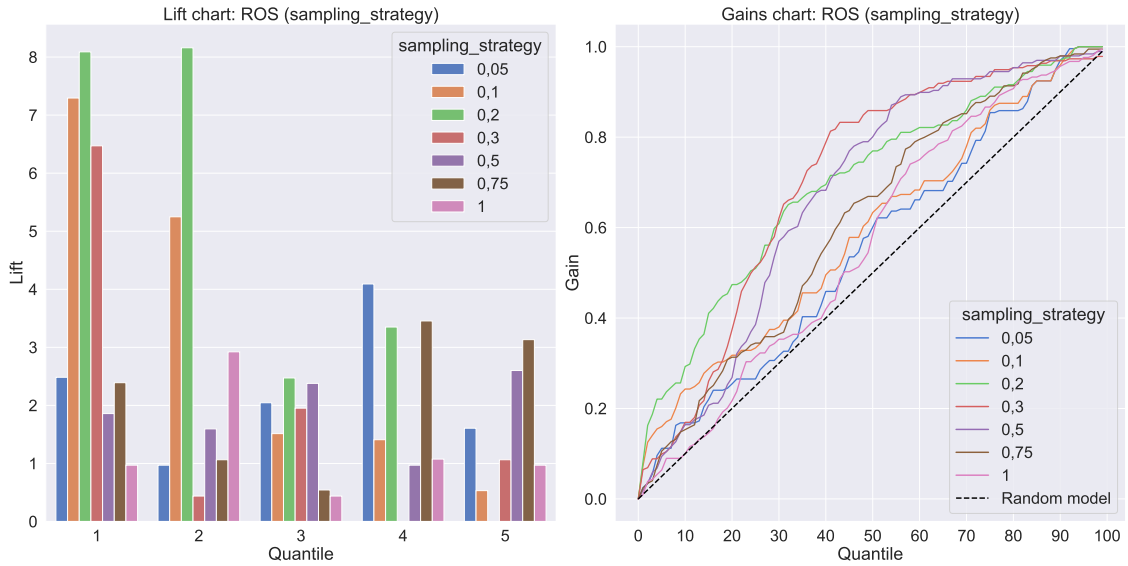
Figure C.1: Lift and Gain Charts: hyper-parameter search for RUS (sampling_strategy)



(a) C-score

(b) Brier-score)

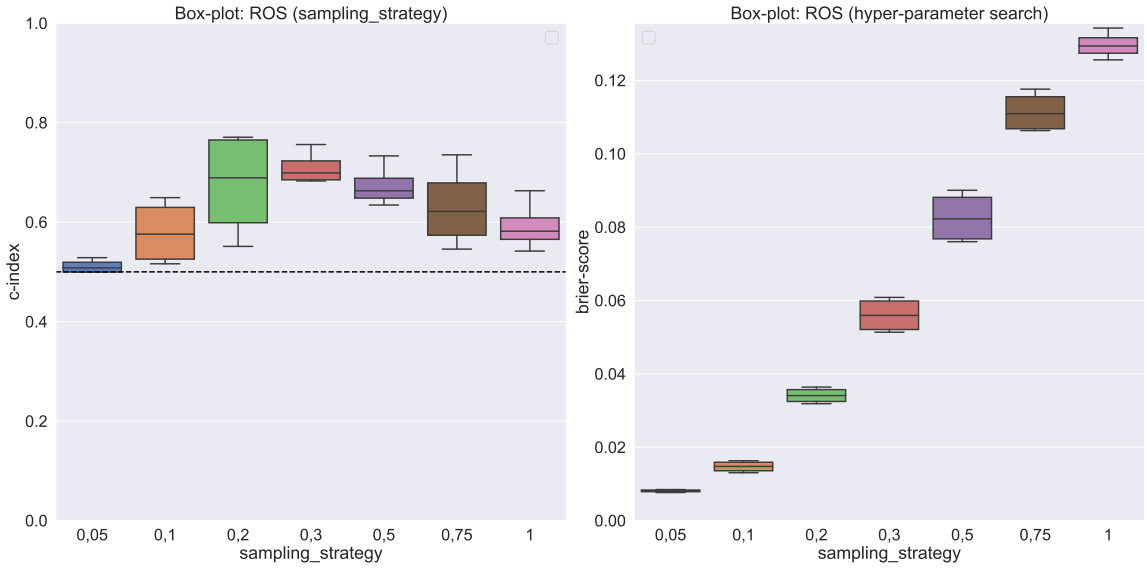
Figure C.2: C-score and Brier-score: hyper-parameter search for RUS (sampling_strategy)



(a) Lift chart

(b) Gains chart

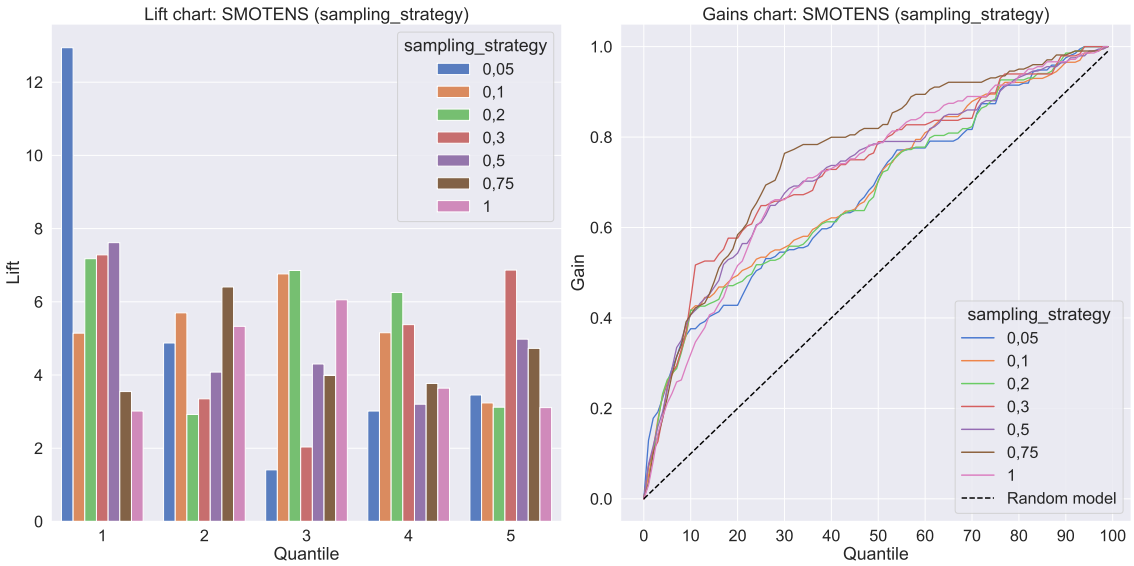
Figure C.3: Lift and Gain Charts: hyper-parameter search for ROS (sampling_strategy)



(a) C-score

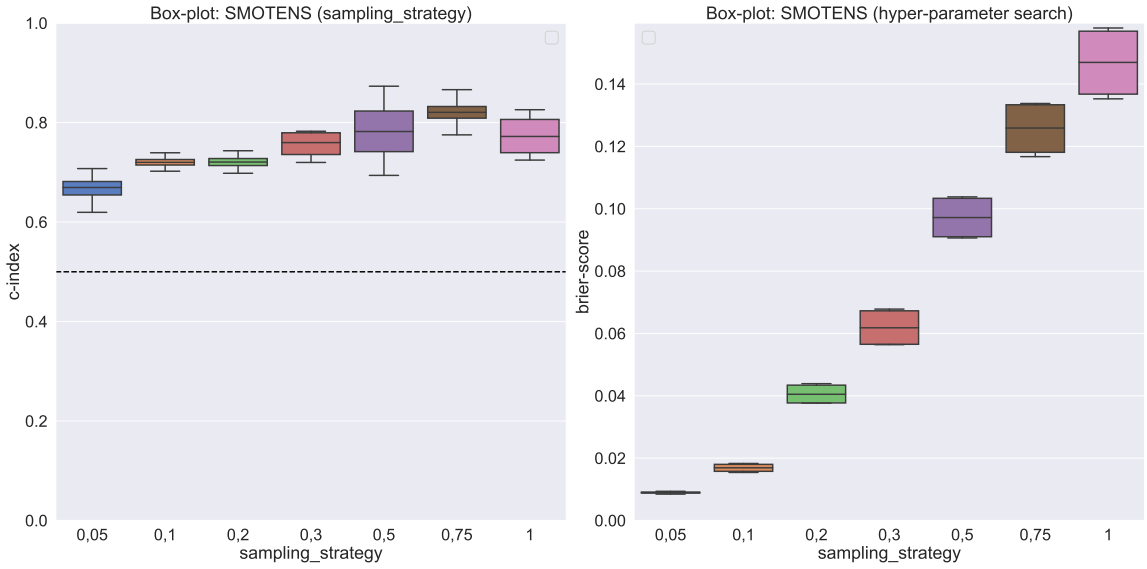
(b) Brier-score

Figure C.4: C-score and Brier-score: hyper-parameter search for ROS (sampling_strategy)



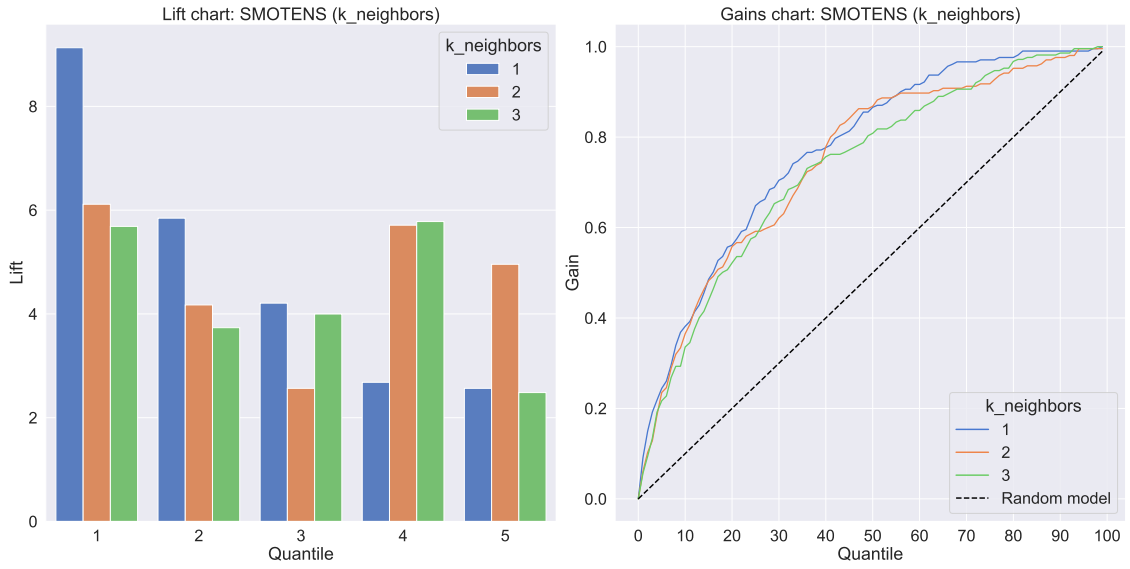
(a) Lift chart (b) Gains chart

Figure C.5: Lift and Gain Charts: hyper-parameter search for SMOTE-NC (sampling_strategy)



(a) C-score (b) Brier-score

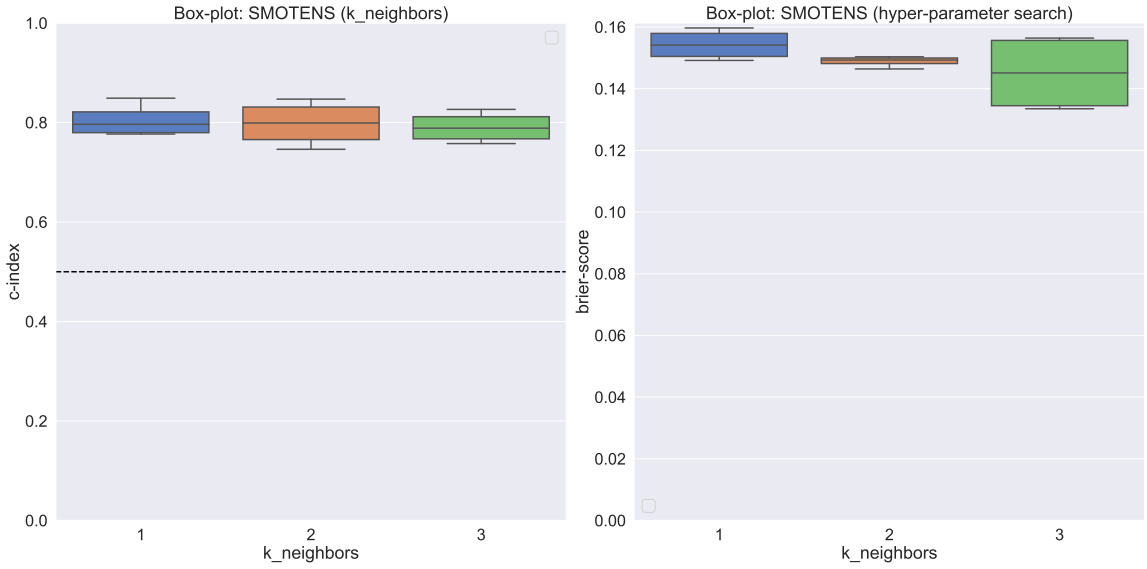
Figure C.6: C-score and Brier-score: hyper-parameter search for SMOTE-NC (sampling_strategy)



(a) Lift chart

(b) Gains chart

Figure C.7: Lift and Gain Charts: hyper-parameter search for SMOTE-NC (k_neighbors)



(a) C-score

(b) Brier-score

Figure C.8: C-score and Brier score: hyper-parameter search for SMOTE-NC (k_neighbors)

Table C.3: *study.sampler.s2*: Fixed parameters

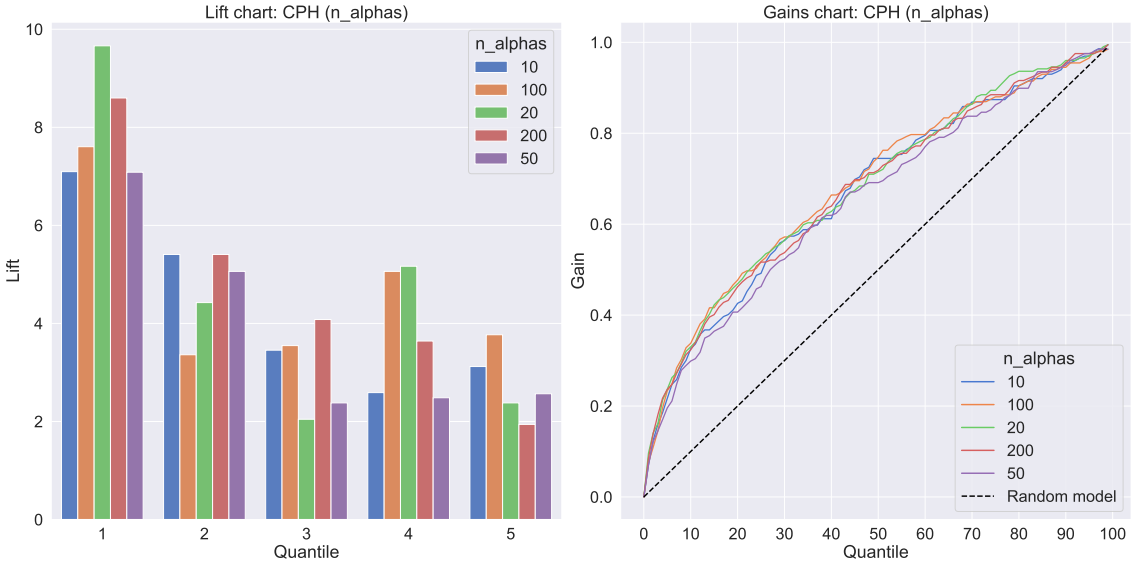
Parameter	Value
n_dates	5
n_folds	100
lift_durs	[1,2,3]
n_samples_train	[10000]
n_samples_test	[30000]
init_durs	['sa_activation_months_qty']
end_durs	[1]
granularities	['weekly']
models	GBSurv: • learning_rate = 0.2 • max_depth = 2 • dropout_rate = 0.1

Table C.4: *study.survivalfeats.s1*: Fixed parameters

Parameter	Value
n_dates	5
n_folds	100
lift_durs	[1,2,3]
n_samples_train	[30000]
n_samples_test	[30000]
end_durs	[1]
granularities	['daily', 'weekly']
samplers	RUS: • sampling_strategy = 0.2
models	GBSurv: • learning_rate = 0.2 • max_depth = 2 • dropout_rate = 0.1

C.2 Survival features

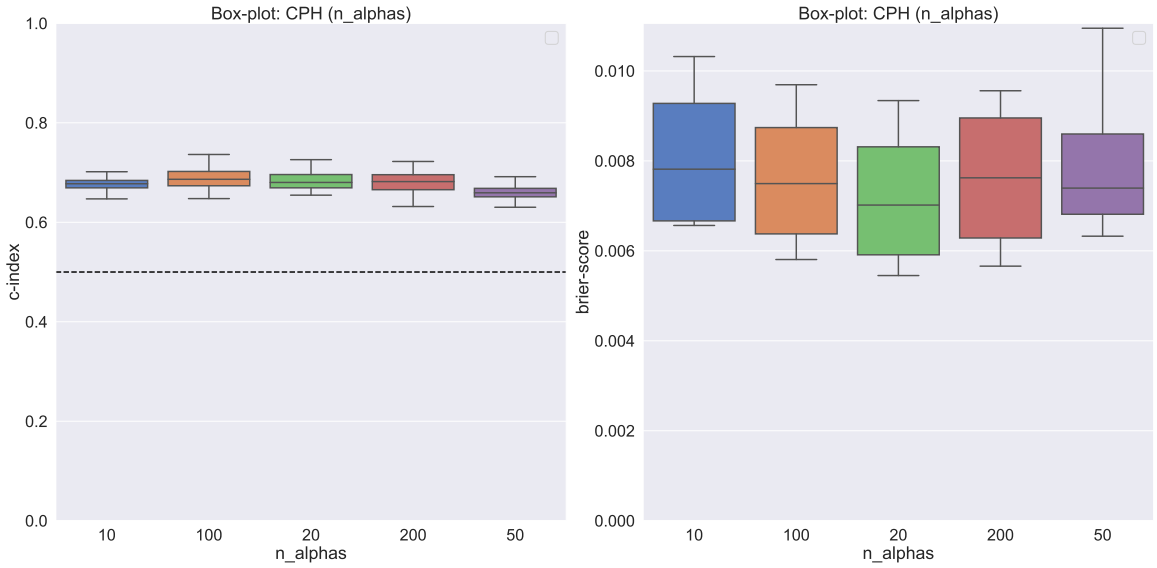
C.3 Survival Models and Benchmark



(a) Lift chart

(b) Gains chart

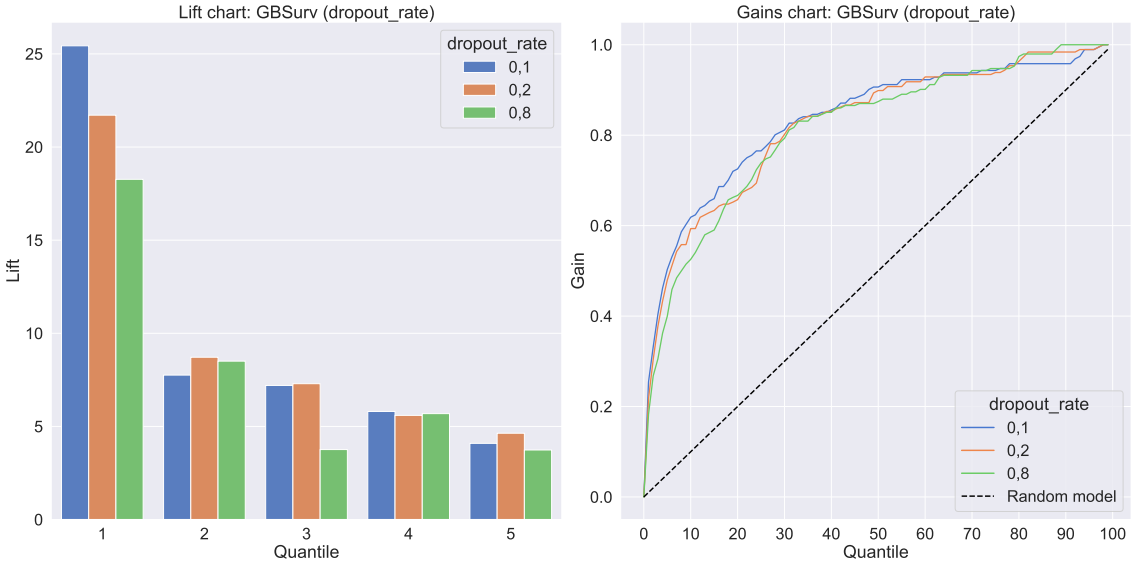
Figure C.9: Lift and Gain Charts: hyper-parameter search for CPH (n_alphas)



(a) C-score

(b) Brier-score)

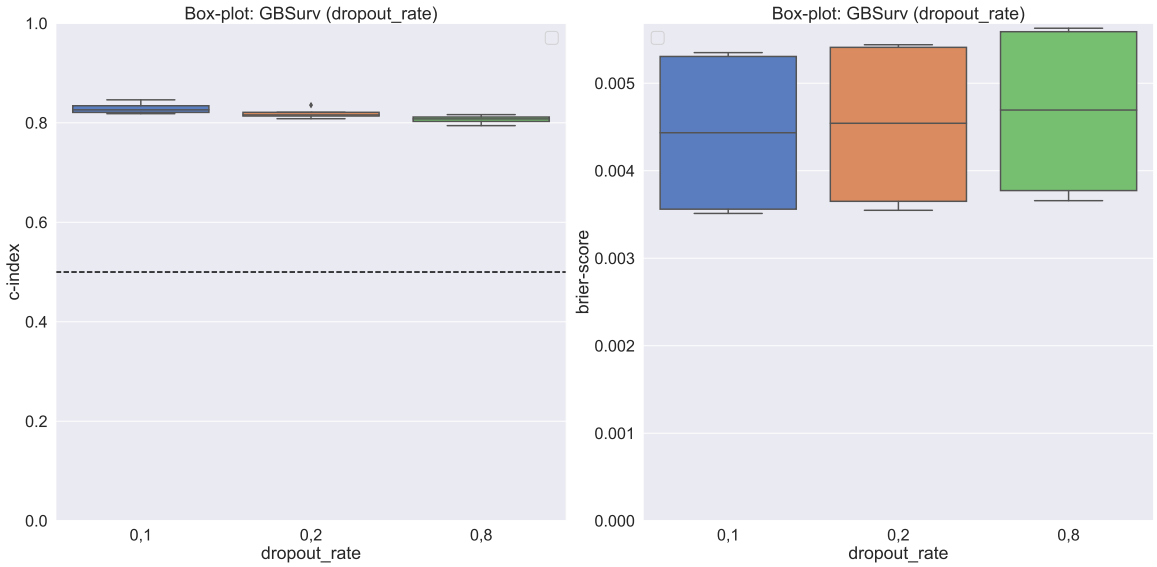
Figure C.10: C-score and Brier-score: hyper-parameter search for CPH (n_alphas)



(a) Lift chart

(b) Gains chart

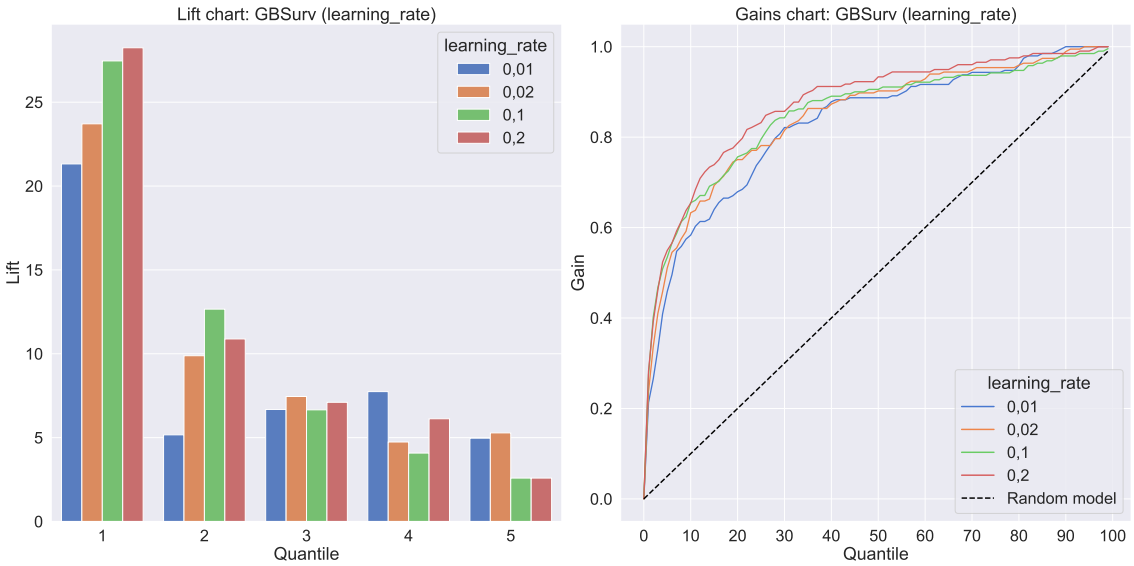
Figure C.11: Lift and Gain Charts: hyper-parameter search for GBSurv (dropout_rate)



(a) C-score

(b) Brier-score

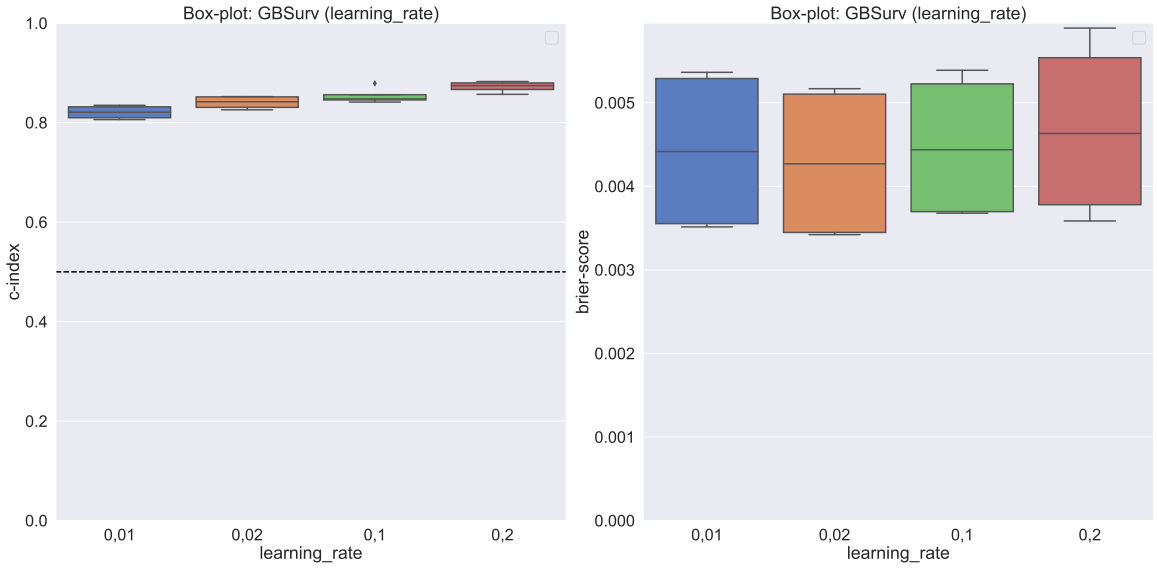
Figure C.12: C-score and Brier-score: hyper-parameter search for GBSurv (dropout_rate



(a) Lift chart

(b) Gains chart

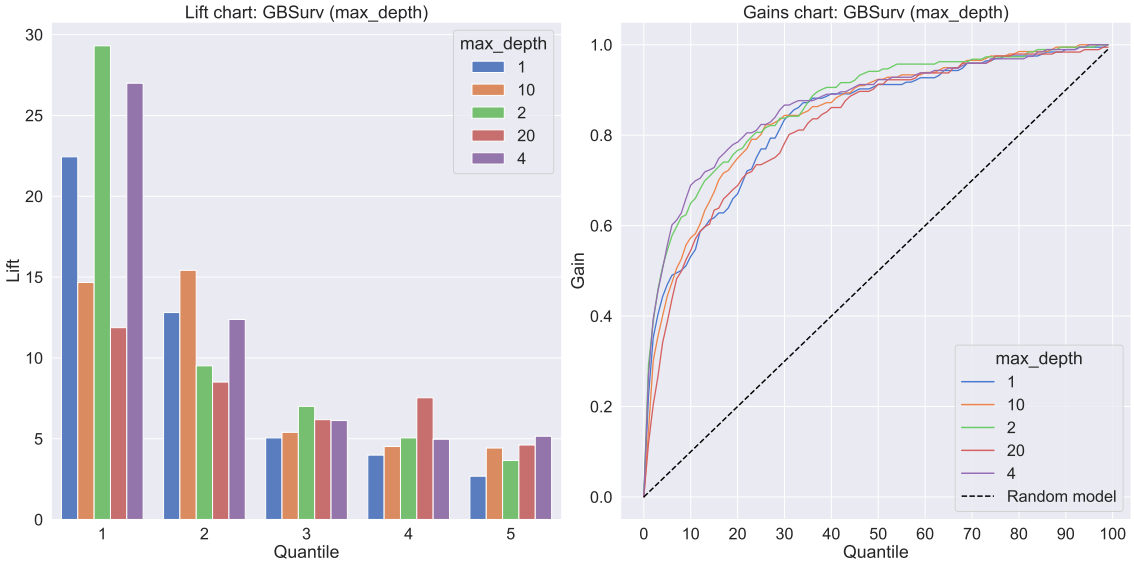
Figure C.13: Lift and Gain Charts: hyper-parameter search for GBSurv (learning_rate)



(a) C-score

(b) Brier-score

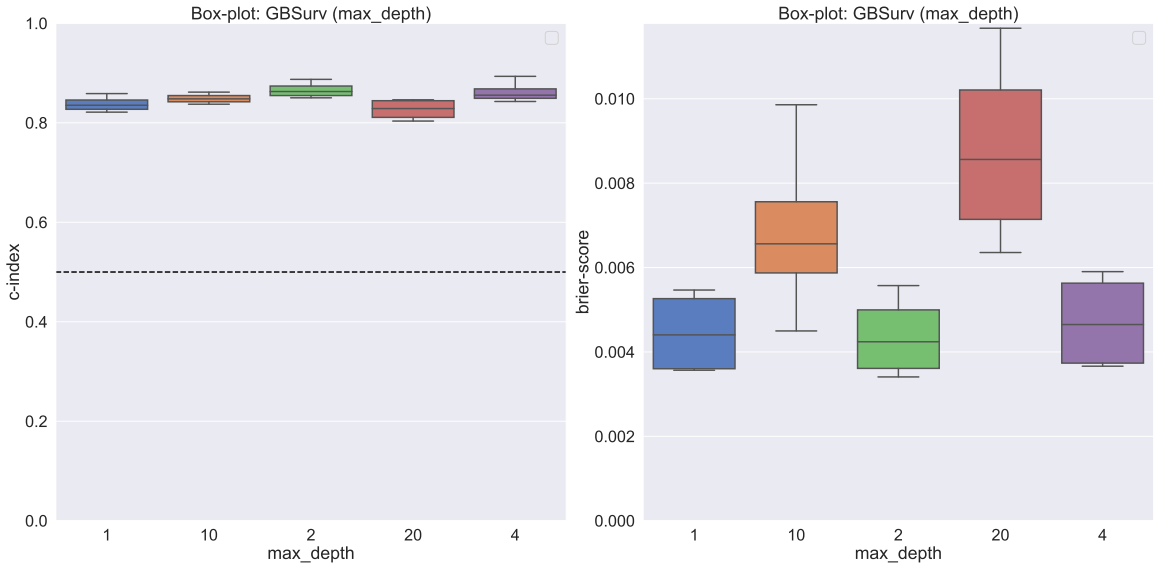
Figure C.14: C-score and Brier-score: hyper-parameter search for GBSurv (learning_rate)



(a) Lift chart

(b) Gains chart

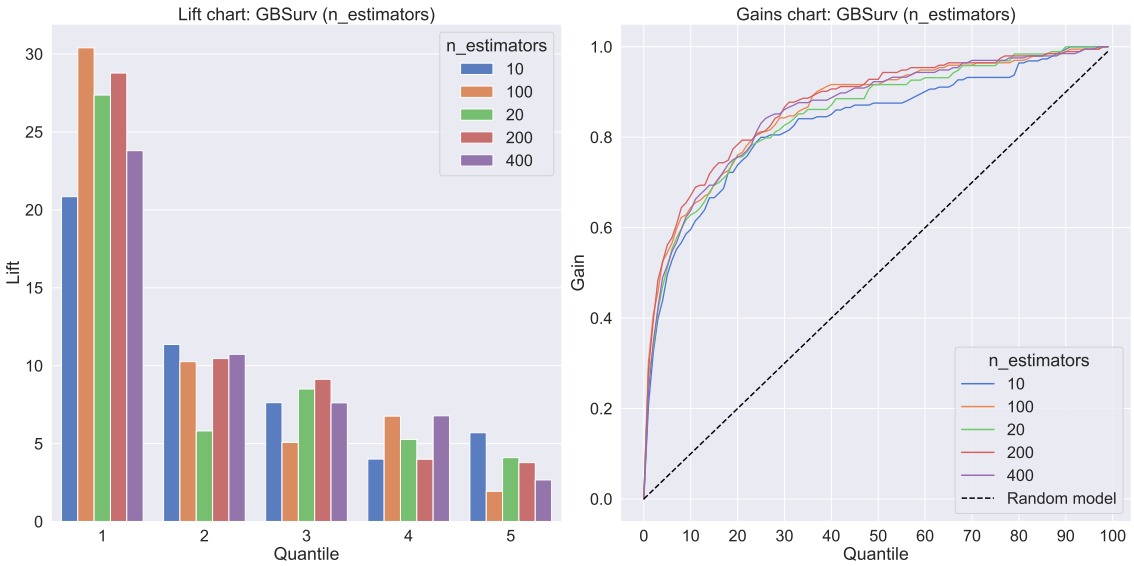
Figure C.15: Lift and Gain Charts: hyper-parameter search for GBSurv (max_depth)



(a) C-score

(b) Brier-score

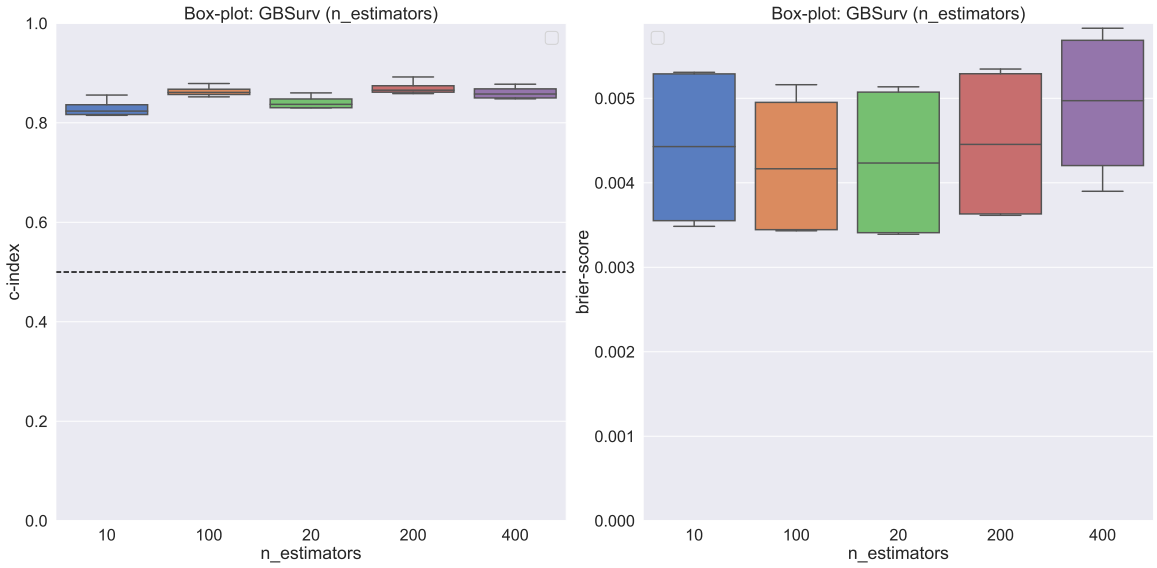
Figure C.16: C-score and Brier-score: hyper-parameter search for GBSurv (max_depth)



(a) Lift chart

(b) Gains chart

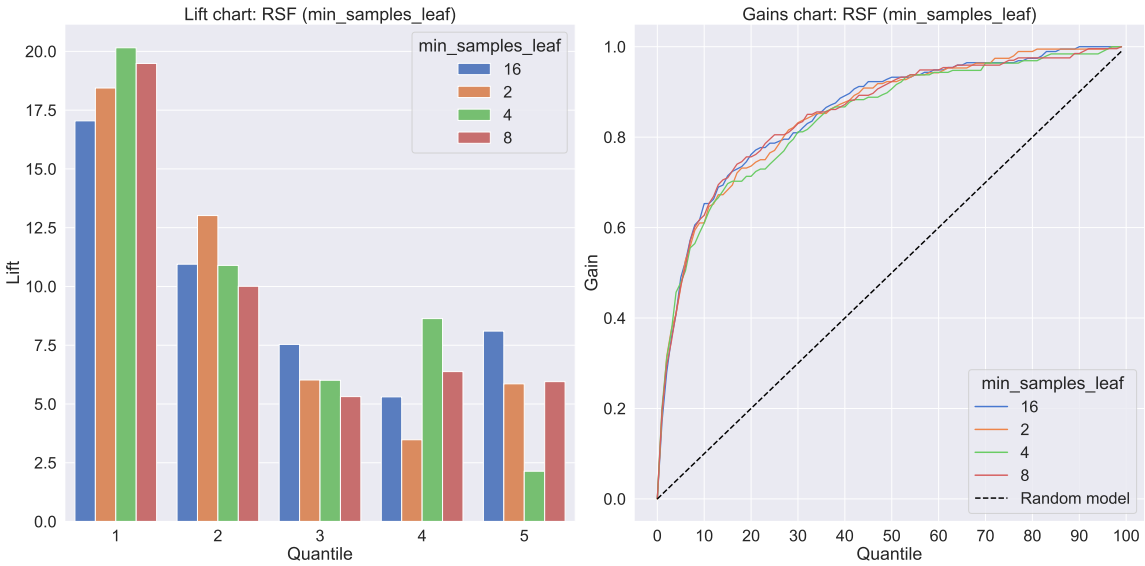
Figure C.17: Lift and Gain Charts: hyper-parameter search for GBSurv (n_estimators)



(a) C-score

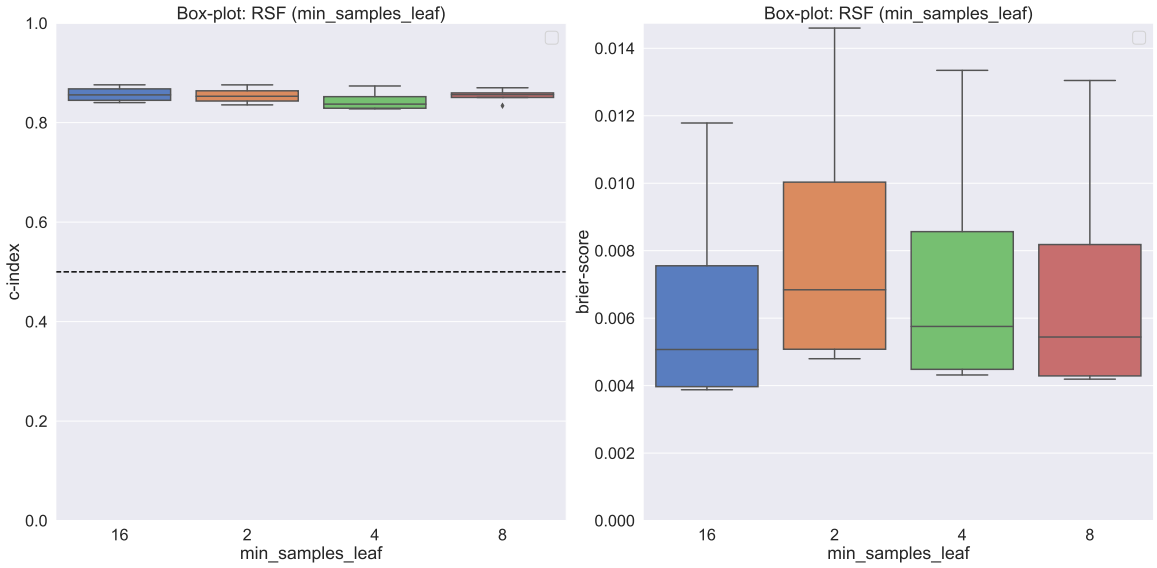
(b) Brier-score

Figure C.18: C-score and Brier-score: hyper-parameter search for GBSurv (n_estimators)



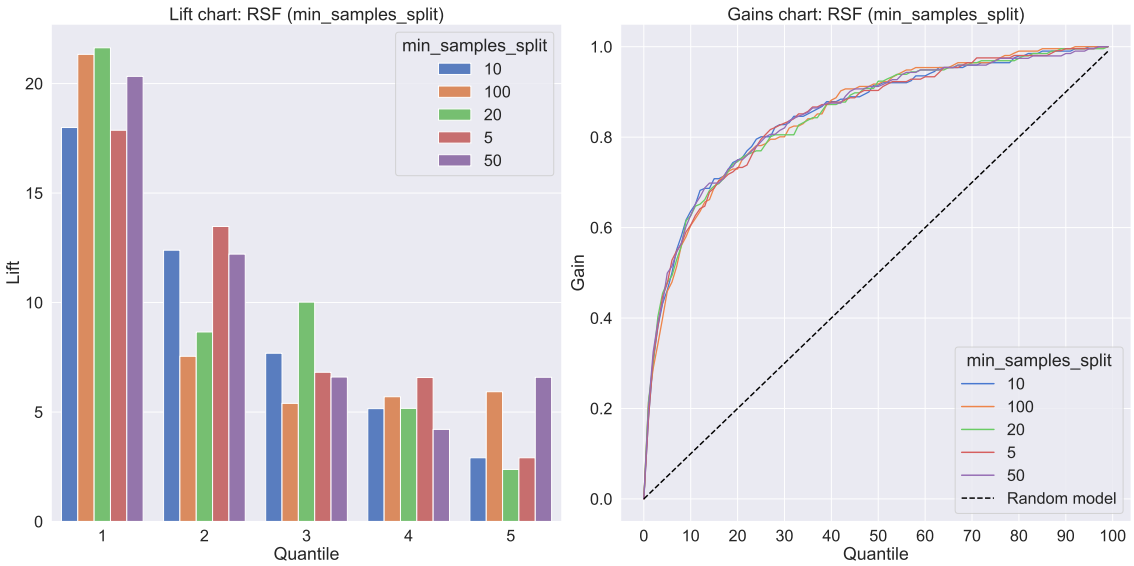
(a) Lift chart (b) Gains chart

Figure C.19: Lift and Gain Charts: hyper-parameter search for RSF (min_samples_leafs)



(a) C-score (b) Brier-score

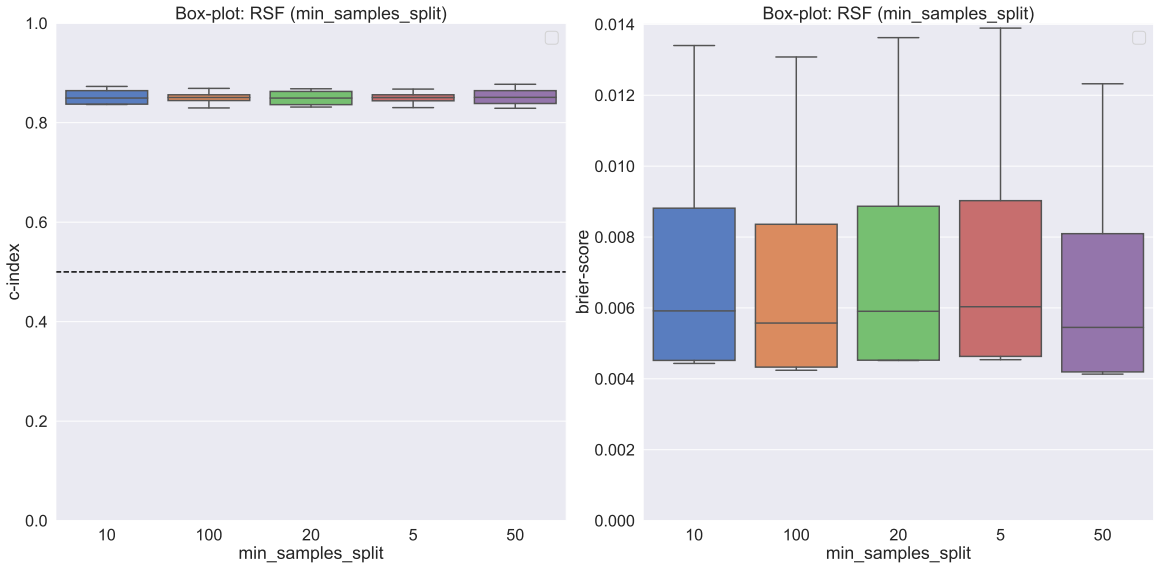
Figure C.20: C-score and Brier-score: hyper-parameter search for RSF (min_samples_leaf)



(a) Lift chart

(b) Gains chart

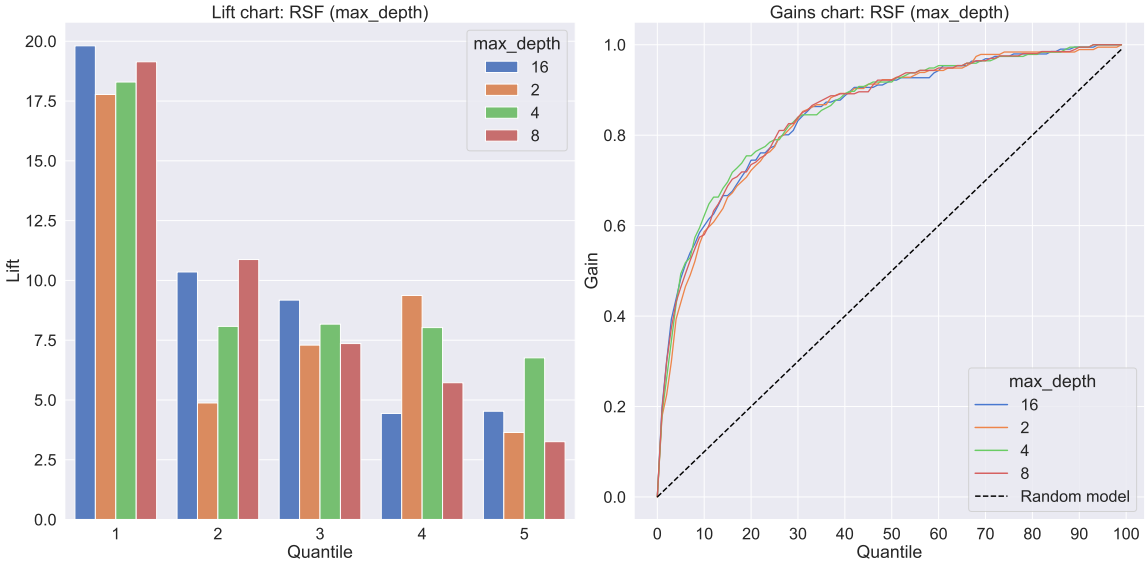
Figure C.21: Lift and Gain Charts: hyper-parameter search for RSF (min_samples_split)



(a) C-score

(b) Brier-score

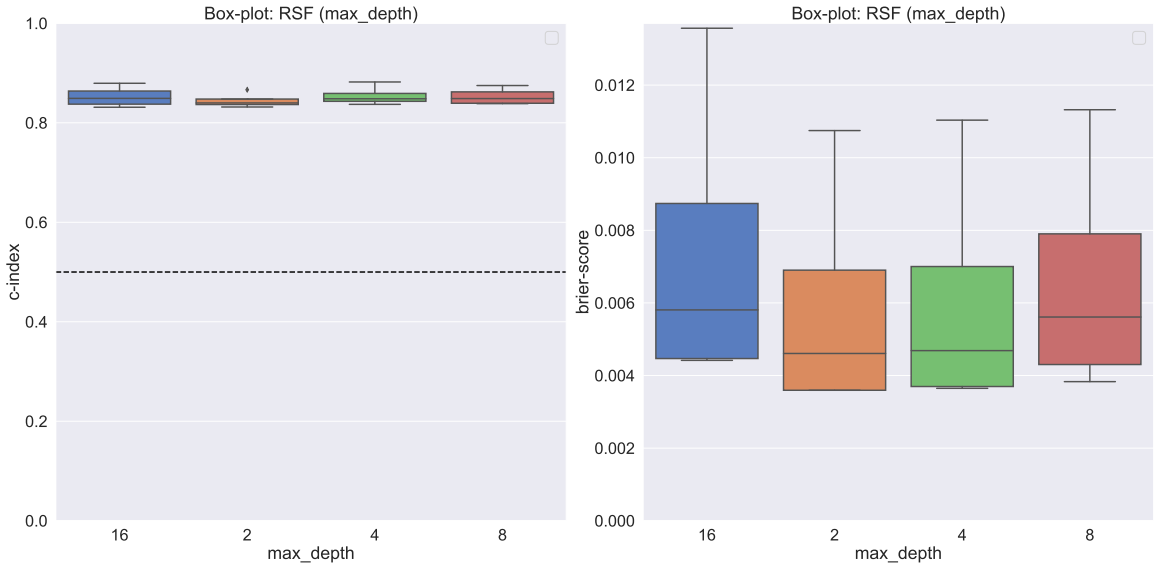
Figure C.22: C-score and Brier-score: hyper-parameter search for RSF (min_samples_split)



(a) Lift chart

(b) Gains chart

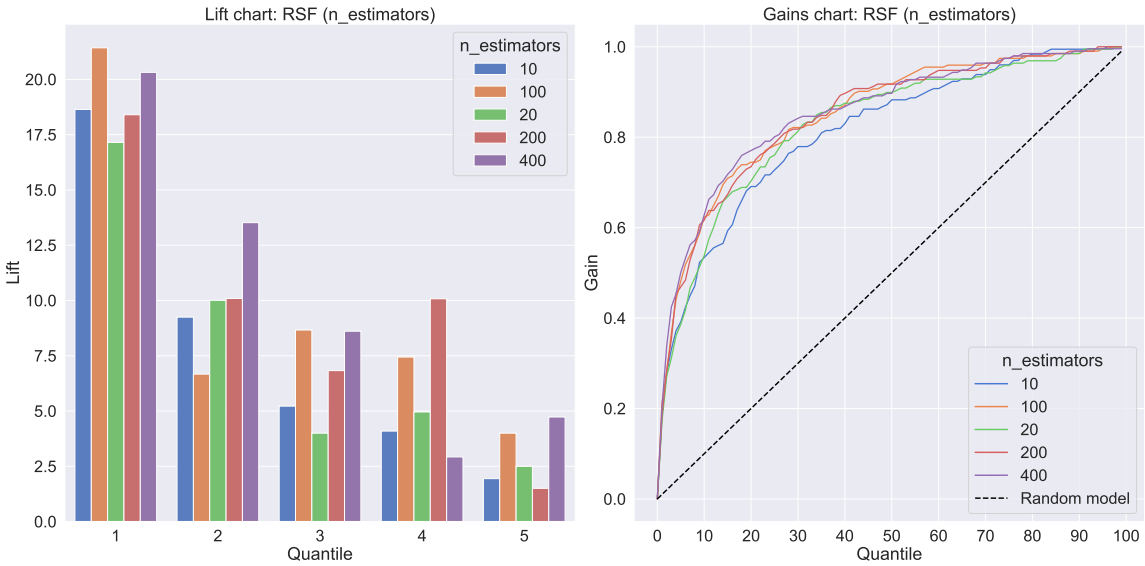
Figure C.23: Lift and Gain Charts: hyper-parameter search for RSF (max_depth)



(a) C-score

(b) Brier-score

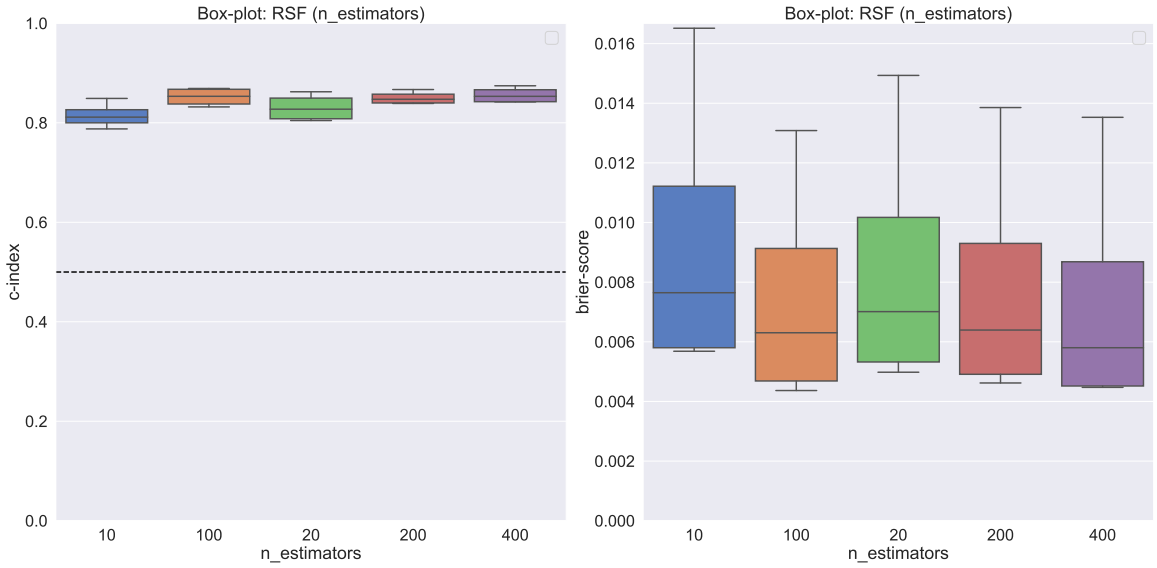
Figure C.24: C-score and Brier-score: hyper-parameter search for RSF (max_depth)



(a) Lift chart

(b) Gains chart

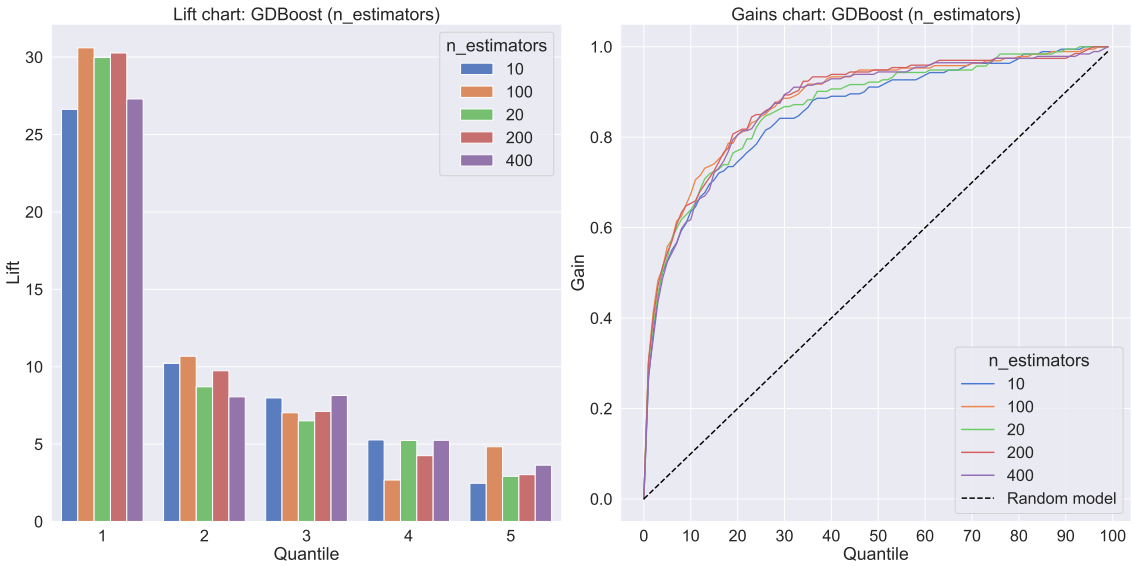
Figure C.25: Lift and Gain Charts: hyper-parameter search for RSF (n_estimators)



(a) C-score

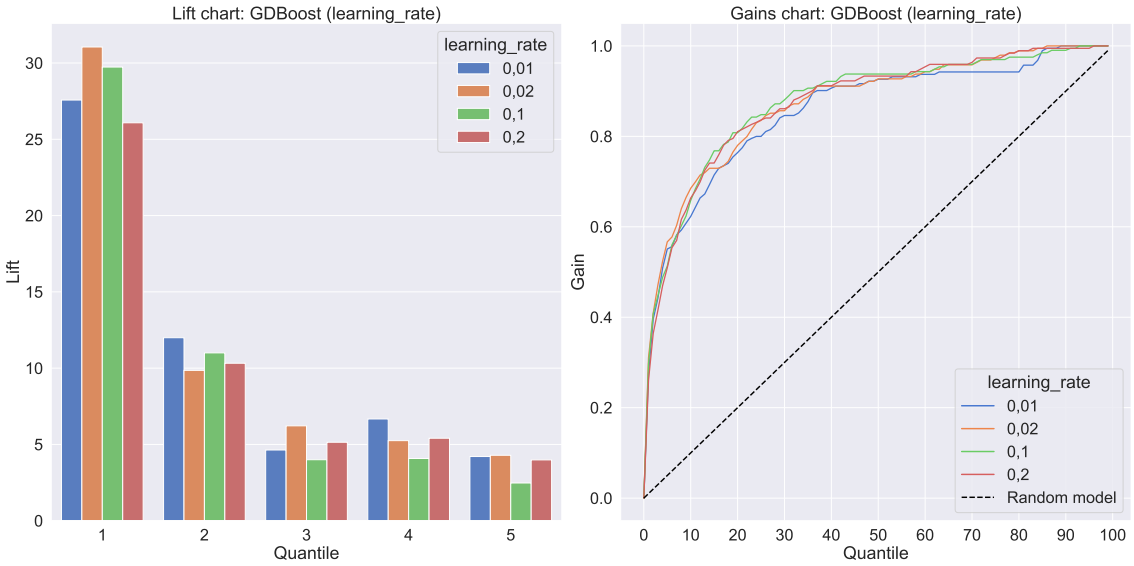
(b) Brier-score

Figure C.26: C-score and Brier-score: hyper-parameter search for RSF (n_estimators)



(a) Lift chart (b) Gains chart

Figure C.27: Lift and Gain Charts: hyper-parameter search for GDBoost (n_estimators)



(a) Lift chart (b) Gains chart

Figure C.28: Lift and Gain Charts: hyper-parameter search for GDBoost (learning_rate)

Table C.5: *study.survivalfeats.s2*: Fixed parameters

Parameter	Value
n_dates	5
n_folds	100
lift_durs	[1,2,3]
n_samples_train	[30000]
n_samples_test	[30000]
init_durs	['0']
granularities	['daily', 'weekly']
samplers	RUS: • sampling_strategy = 0.2
models	GBSurv: • learning_rate = 0.2 • max_depth = 2 • dropout_rate = 0.1

Table C.6: *study.survivalfeats.s1 and s2* (granularity): Fixed parameters

Parameter	Value
n_dates	5
n_folds	100
lift_durs	[1,2,3]
n_samples_train	[30000]
n_samples_test	[30000]
init_durs	['0', '2', 'client-dur', 'pf-dur']
end_durs	[1, 2, 3, 6]
granularities	['daily', 'weekly']
samplers	RUS: • sampling_strategy = 0.2
models	GBSurv: • learning_rate = 0.2 • max_depth = 2 • dropout_rate = 0.1

Table C.7: *study.model.s1*: Models hyper-parameter search space

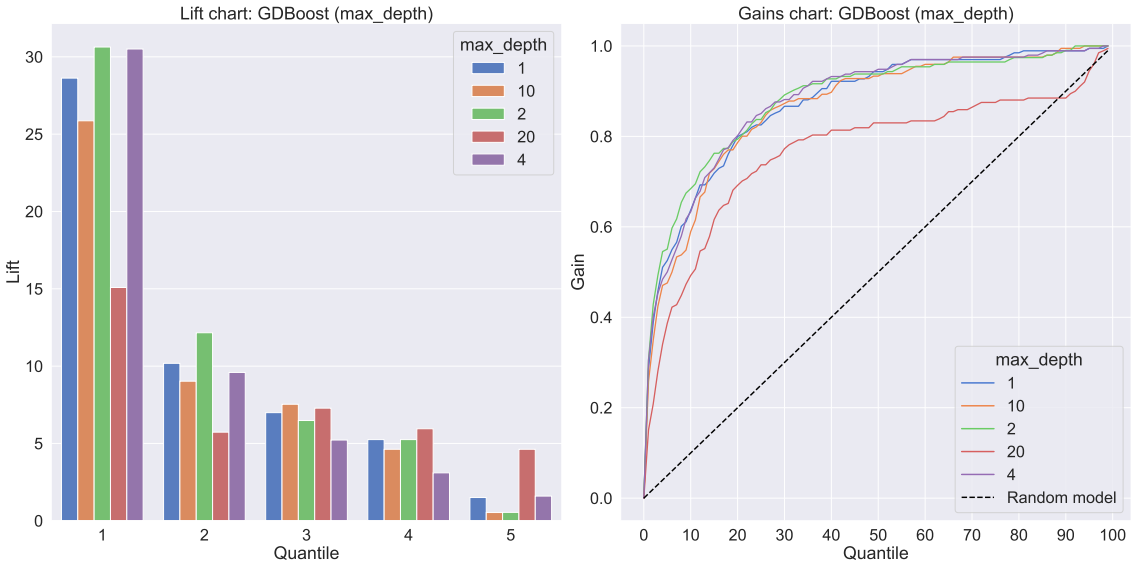
Model	Hyper-parameter search space
RSF	n_estimators: [10, 20, 100, 200, 400]
RSF	min_samples_leaf: [2, 4, 8, 16]
RSF	max_depth: [2, 4, 8, 16]
RSF	min_samples_split: [5, 10, 20, 50, 100]
GBSurv	n_estimators: [10, 20, 100, 200, 400]
GBSurv	learning_rate: [0.01, 0.02, 0.1, 0.2]
GBSurv	max_depth: [1, 2, 4, 10, 20]
GBSurv	dropout_rate: [0.1, 0.2, 0.8]
CPH	n_alphas: [10, 20, 50, 100, 200]
CPH	l1_ratio: [0.1, 0.2, 0.4, 0.8]
GDBoost	n_estimators: [10, 20, 100, 200, 400]
GDBoost	learning_rate: [0.01, 0.02, 0.1, 0.2]
GDBoost	max_depth: [1, 2, 4, 10, 20]

Table C.8: *study.model.s1*: Fixed parameters

Parameter	Value
n_dates	2
n_folds	100
lift_durs	[1]
n_samples_train	[100000]
n_samples_test	[10000]
init_durs	['3']
end_durs	[1]
granularities	['daily']
samplers	RUS: • sampling_strategy = 0.2

Table C.9: *study.model.s2*: Fixed parameters

Parameter	Value
n_dates	5
n_folds	100
lift_durs	[1,2,3]
n_samples_train	[100000]
n_samples_test	[10000]
init_durs	['3']
end_durs	[1]
granularities	['daily']
samplers	RUS: • sampling_strategy = 0.2



(a) Lift chart

(b) Gains chart

Figure C.29: Lift and Gain Charts: hyper-parameter search for GDBoost (max_depth)

Table C.10: *study.model.s3*: Fixed parameters

Parameter	Value
n_dates	5
n_folds	100
lift_durs	[1,2,3]
n_samples_train	[10000]
n_samples_test	[30000]
init_durs	['sa_activation_months_qty']
end_durs	[1]
granularities	['weekly']
models	GBSurv: • learning_rate = 0.2 • max_depth = 2 • dropout_rate = 0.1

Table C.11: Final Recommendation

Parameter	Value
init_durs	[3]
end_durs	[1]
granularities	['daily']
samplers	RUS: • sampling_strategy = 0.2
models	RSF • n_estimators = 4 • min_samples_leaf = 4 • max_depth = 16 • min_samples_split = 20