ABSTRACT

ASSESSING THE PERFORMANCE AND MERIT OF THE RANDOM SURVIVAL FOREST AND COX MODELS ON A PANCREATIC CANCER DATA SET

Carl E. Mueller
Division of Statistics
Northern Illinois University, 2019
Dr. Haiming Zhou, Director

Random Survival Forest (RSF) is one of the most powerful and easily applied machine learning models for survival data. RSF sacrifices some of the interpretability of the decision trees used to grow the forest in order to significantly reduce the bias and variance of the basic classification and regression tree (CART) paradigm. The lessened interpretability and higher computational intensity of RSF means that it may not always be the preferred method, even in settings where black-box methods are readily used. By contrast, the Cox Proportional Hazards (PH) model is incredibly flexible, resistant to overfitting, and transparently estimable. The tradeoff for the Cox PH model is the difficulty in construction when rigorous best practices are followed and model assumptions are violated, requiring complex extensions.

This thesis finds the best performing RSF, as measured by the Concordance Index (CI), for a cohort of pancreatic cancer patients from the Surveillance, Epidemiology and End Results (SEER) index and compares the predictive power of that model to a carefully constructed Cox model, while providing commentary on the relative strengths and weaknesses of each approach. Our conclusion marries the pros and cons of the classical and contemporary to cap a complete and cohesive narrative that insists on analyst customization for the machine learning technique and careful construction in the context of the Cox approach.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

AUGUST 2019

ASSESSING THE PERFORMANCE AND MERIT OF THE RANDOM SURVIVAL FOREST AND COX MODELS ON A PANCREATIC CANCER DATA SET

BY

CARL EDWARD MUELLER

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF STATISTICS

Thesis Director:
Haiming Zhou

ACKNOWLEDGEMENTS

Special thanks to my advisor Haiming Zhou for his time, expertise, kindness, and unbelievable patience. I am indebted to Duchwan Ryu and Alan Polansky for their willingness to help in spite of their busy schedules.

DEDICATION

To Katelyn Fink, with love, for her unwavering belief and support.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

LIST OF APPENDICES

Appendix

CHAPTER 1

INTRODUCTION

Survival analysis is an important tool for real-world scenarios in which the variable of interest is time until an event occurs. The events include, but are not limited to: disease remission, recidivism, and death. One important aspect of survival analysis is that it is generally applied to datasets that are incomplete; there may be subjects for which we know that survival up to a certain time has occurred, but have no information about the subject beyond that time point. In fact, there may be subjects about whom we know nothing *before* a certain time point, or even subjects for which we have a period of time during which no data exists. These three scenarios constitute the possible ways in which *censoring* of a subject may occur.

Survival analysis is concerned with three primary goals. Firstly, it seeks to estimate and interpret survival (and the analogous hazard) functions. Another goal is a comparison of these functions, as in the case of a treatment group and a placebo group. The third goal includes assessment of the relationship between the predictor or explanatory variables, and survival time (Kleinbaum 2012). The survival time is the time at which the event of interest, generally referred to as failure (taken to be death for this study) is observed. The main complication in our study is the existence of *right-censored* data – subjects for which we know that failure has not occurred up to

a certain time, but have no further information after that time. This type of censoring occurs for subjects which have withdrawn from a study, were lost to follow-up, or subjects for which death (due to event of interest) did not occur by the end of the observation period. Subjects dead from other causes not related to their cancer are considered to be right-censored as well. Survival data consists of a vector of survival times and a corresponding status or censoring indicator variable, which specifies whether or not a subject is censored. The general form of the survivor function may be expressed as:

$$S(t) = P(T > t)\,, \tag{1.1}$$

Where the probability of survival beyond time *t* is of interest, with *t* taking on some unit of time as specified by the data. An analogous function which expresses the instantaneous rate of failure at time *t*, called the *hazard rate*, is given by:

$$h(t) = \lim_{\Delta t \to 0} \frac{P(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t}\,, \tag{1.2}$$

The hazard rate and survivor function are related by

$$h(t) = -[\frac{dS(t)/dt}{S(t)}]\,, \tag{1.3}$$

which can be used to determine *h(t)* from *S(t)* and vice-versa. Thus estimating either function allows estimation of the other. Consequently, different methods in survival analysis will estimate different functions, but we will able to make direct comparisons between estimates by way of the relation in (1.3).

The presence of censored data points within a set of collected data poses challenges to conventional statistical approaches, and has necessitated the development of application-specific techniques. The Cox PH model headlines a class of semi-parametric relative-risk models that are widely used and highly generalizable (Kalbfleisch 2002). Machine learning algorithms have been adapted to be able to function with censored data as well, allowing new types of analyses to flourish (Wang 2017). We examine the effectiveness of what is widely considered the "best" overall approach to survival from a machine learning perspective, the Random Survival Forest (Gong 2018). Using a set of pancreatic cancer data from the surveillance, epidemiology and end results (SEER) study, we provide a comprehensive overview of the RSF algorithm with special attention paid to considerations of each possible algorithm option and tuning parameter, while attempting to find the "optimal" settings for analysis of this type of data. Previous studies in the literature are used to give perspective for the observed results, and for a more grounded perspective we establish a baseline comparison using the Cox Proportional Hazards model as a benchmark. The goal of this project involves modeling the (possibly nonlinear) effects of the predictor variables and their interactions on the survival time for subjects with this very deadly cancer, and the assessment of the predictive power of those models using the most common contemporary metric for this type of analysis; the concordance index.

The black-box machine learning methods of today sacrifice interpretability for accuracy of predictions and prognostic power. If these models, when optimally tuned, cannot outperform a carefully treated Cox model to a significant degree, their use should be highly scrutinized. Our aim is threefold: explore the theoretical motivations for choices of various tuning parameters in the RSF algorithm, perform an exhaustive search for the best possible ensemble forest model for

our data and interpret the results to provide a starting point for future analyses of categorically dominated predictor spaces of failure data related to cancer, and finally to compare the performance of a rigorously constructed Cox model to that of the ‘best’ RSF model and offer commentary on the advantages and disadvantages of each approach as representatives for their respective regimes; classical survival analysis and machine learning.

CHAPTER 2

BACKGROUND OF THE STUDY

2.1 Classical Survival Analysis

2.1.1 Kaplan-Meier Survival Estimation

The Kaplan-Meier estimator is a non-parametric statistic used to estimate the survival function. It is a step function of ordered failure (death) times. The power of this type of estimation is that it does not make any restrictive assumptions about the shape of the underlying true survival function. The KM curve is the complement of the empirical distribution function in cases where no censoring occurs in the data. (Kleinbaum 2012) The KM estimator at time $t_{(f)}$ is the ratio of subjects surviving beyond $t_{(f)}$ to the total number of subjects,

$$\hat{S}(t_{(f)}) = \frac{\#\ surviving\ beyond\ t_{(f)}}{\#\ in\ cohort}, \tag{2.1}$$

where $t_{(f)}$ denotes failure time *f* in the set of ordered failure times; $t_{(0)} = 0$ and $t_{(1)}$ is the first time at which any subject dies.

2.1.2 The Log-Rank Test Statistic

The log-rank test for two groups compares the KM survival curves for those groups to determine if there is a statistical difference in survival. To calculate the statistic for two groups, let $j = 1, \dots, J$ be the unique observed death times across both groups. Take $n_{1,j}\ and\ n_{2,j}$ to be the risk sets for the two groups at time *j*; the number of subjects not yet dead or censored at the start of that time period. Calculate n_j as $n_{1,j} + n_{2,j}$ and O_j as $O_{1,j} + O_{2,j}$, where $O_{1,j}, O_{2,j}$ are the number of observed deaths in each group at time *j*. Under a null hypothesis of equal hazard functions for the two groups, the LR statistic compares each $O_{i,j}$ to its expected value. Further derivation of the statistic can be found in Peto & Peto (1972), but the following equations for the expectation and variance of the distribution of observed events under the null hypothesis of equal hazard functions will be stated without proof in this work as follows:

$$E_{i,j} = O_j \frac{N_{i,j}}{N_j}. \tag{2.2}$$

$$V_{i,j} = E_{i,j}\left(\frac{N_j - N_{i,j}}{N_j}\right)\left(\frac{N_j - O_j}{N_j - 1}\right) \tag{2.3}$$

Taking the sum of the deviations of $O_{i,j}$ from their expectations and dividing by the square root of the variances, we produce a test statistic, the distribution of which approaches a standard normal as long as the number of observed event times is large. The LR statistic is thus:

$$Z_i = \frac{\sum_{j=1}^{J}(O_{i,j} - E_{i,j})}{\sqrt{\sum_{j=1}^{J} V_{i,j}}}. \tag{2.4}$$

Segal (1988) was one of the first to use the LR test as a method for splitting nodes in a regression forest (see chapter 3). Leblanc and Crowley (1993) developed an algorithm based on the idea that the highest log-rank test statistic for all possible sub-groups may be used to determine the node split that results in the highest *heterogeneity*; the split that will result in the two groups which have the greatest probability of having come from different populations. This is a desirable property because it will theoretically divide the data into the most homogenous sub-groups, allowing new observations to be placed into the group that will best predict their survival. Hothorn and Lausen (2003) proposed a modified rule based on maximizing a standardized log-rank statistic over a number of cutpoints computed at each node. This "log-rank score" (Ishwaran 2008) statistic is not used in this study due to similar performance on a wide variety of data coupled with much greater computation time.

2.1.3 Cox Proportional Hazards Model

The Cox Proportional Hazards (Cox PH) model is the most commonly used failure model belonging to the class of proportional hazards models (Kalbfleisch 2002). The usual form of the Cox model is (Cox 1972):

$$h(t, \boldsymbol{X}) = h_0(t) e^{\sum_{i=1}^{p} \beta_i X_i}. \tag{2.5}$$

In (2.5), $\boldsymbol{X}$ is a vector of p explanatory variables and t is time. The Cox model can be thought of as the product of two functions; the so-called "baseline hazard" function, $h_0(t)$, which depends on time but does not depend on any predictor in $\boldsymbol{X}$, and an exponential function of $\boldsymbol{X}$ that does

not involve time *t*. (Kleinbaum 2012) The Cox model belongs to a class of semi-parametric models, the parameters in this case being the β_i's. β_i represents the unknown coefficient of variable X_i, to be estimated. The model yields an expression for the hazard at time *t* for a subject with a particular vector of covariates. As $h_0(t)$ is not specified, the Cox model is said to be semiparametric. (Kalbfleisch 2002) This unspecified baseline hazard means that we cannot easily obtain an estimate of the hazard for one particular individual, but if given two vectors of predictors, we may find the ratio of the hazards for those two subjects easily:

$$\widehat{HR} = \frac{h(t,\boldsymbol{X}) = h_0(t)e^{\sum_{i=1}^{p}\beta_i X_i}}{h(t,\boldsymbol{X}^*) = h_0(t)e^{\sum_{i=1}^{p}\beta_i X^*{}_i}} . \tag{2.6}$$

The baseline hazards in 2.6 cancel, leaving us with something that depends only on $\boldsymbol{X}$ and $\boldsymbol{X}^*$**.** This predicted hazard ratio should be constant in time, since the only part of the numerator or denominator that depends on time is the baseline hazard function, which is the same for both subjects at each time *t*. An alternative representation of 2.6 is:

$$\widehat{HR} = exp\left[\sum_{i=1}^{p}\beta_i(X_i^* - X_i)\right] . \tag{2.7}$$

From 2.7 the interpretation of β_i is more clear; for two subjects with values X_a, X_a^* for covariate a, which differ by one unit, their hazard ratio is $exp(\beta_a)$ (holding all other variables constant). This means that a one unit increase in covariate a will correspond to an increase of $exp(\beta_a)$ in the hazard ratio when all other covariates are held constant.

The proportionality of hazards over time in (1.6) may be violated if a predictor variable in $\boldsymbol{X}$ depends on or changes with time. For variables that have this time dependence property and

whose effects we do not wish to model, a stratified model may be fit which stratifies on that covariate. For covariates which have a known time dependency, the extended Cox model for time dependent covariates may be employed, as shown in (2.8):

$$h(t, \boldsymbol{X}(t)) = h_0(t)\exp\left[\sum_{i=1}^{p_1} \beta_i X_i + \sum_{j=1}^{p_2} \beta_j X_j(t)\right]. \quad (2.8)$$

For covariates which do not have an inherent time dependence but have an apparent time dependence via violation of the PH assumption, the extended model for time dependent coefficients may be used. The general form for this model is (Therneau 2019):

$$h(t, \boldsymbol{\beta}(t)) = h_0(t)\exp\left[\sum_{i=1}^{p_1} \beta_i X_i + \sum_{j=1}^{p_2} \beta_j(t) X_j\right] . \quad (2.9)$$

In both 2.8 and 2.9 the baseline hazard is a function of time and is multiplied by an exponential; the key difference is the inclusion of covariates that explicitly time-dependent in 2.8 compared to the time dependent coefficients in 2.9. The inclusion of time dependent coefficients to reconcile a PH violation makes sense; by specifying a function of time for each case where the assumption fails we acknowledge that our estimates for the coefficients of these covariates must evolve with time, which is by definition a violation of the proportional hazards assumption. By specifying a proper function of time we may be able to keep the hazards proportional, and move forward with all model assumptions satisfied.

2.1.4 Concordance Index

The concordance index (CI) is the standard performance measure for model assessment in survival analysis (Raykar 2007). At its core the concordance index is a method for assessing

the quality of rankings by a particular model. In the general case, model predictions for survival time are compared to actual known survival time, where the *ordering* of the predicted and observed times is important instead of the *distance,* as in common metrics such as mean squared error (MSE). A pair of observations is said to be concordant if their predicted survival times are ranked in the same order as their observed survival times. Of course, this requires that the shorter of the observed survival times for the pair not be censored; if the shorter survival time was censored there would be no guarantee that this subject failed prior to the paired subject's failure. Similarly, we do not consider pairs with tied survival times, unless at least one is a death. We will call such a pair of observations which follows these two rules a permissible pair. This leads to the following algorithm for determining concordance, as laid out by Ishwaran in his 2008 paper introducing random survival forests (and used in the R package randomForestSRC in chapter 3):

1. Form all permissible pairs of observations over the data.
2. For each permissible pair with non-tied survival times, count 1 if the longer survival time has the longer predicted outcome, count 0.5 if the predicted outcomes are tied.
3. For each permissible pair with tied survival times where both got the event, count 1 if the predicted times are tied, otherwise count 0.5.
4. For each permissible pair where survival times are tied and one observation is censored, count 1 if the censored observation had the longer predicted survival time, otherwise count 0.5.
5. Calculate the sum over all permissible pairs

6. The Concordance Index, C, is defined as this sum divided by the total number of permissible pairs; loosely the proportion of pairs which are concordant, with special rules on how to handle tied survival times.

Raykar (2007) has shown that maximizing Cox's partial likelihood can be thought of as roughly maximizing a lower bound on the concordance index. Use of the CI to recast survival analysis as a ranking problem has become more common as models become more abstract, less interpretable, and intra-model performance assessment difficulties arise due to the very different contexts in which time-to-event analysis is employed. The survival problem as a ranking problem is natural when thinking in terms of high and low risk groups, or when a statement about survival beyond a certain time as compared to previous similar individuals is desirable; such as being able to tell a patient with confidence that he or she has 6 months – 1 year to live, allowing for plans to be made and affairs to be put in order.

2.2 Machine Learning Approaches

This section discusses the various machine learning models that we have leveraged in our analysis, building up from simple decision trees to random forests. All of the machine learning formulations covered in this section are related back to their specific adaptations for the survival setting, with special care paid to the original implementation of random survival forest in Ishwaran (2008).

2.2.1 Classification and Regression Trees

Classification and regression trees (CART) are a longstanding staple of machine learning due to their simplicity and interpretability. These trees segment the feature space into regions using certain user-supplied criteria. Decision trees are relatively simple to understand and interpret, and perform implicit feature selection. These trees, however, are prone to overfitting training data and have high variance when used outside of ensemble approaches. The tradeoff of moving to an ensemble method is interpretability of the model, and so care must be taken in settings where this is important, such as pharmaceutical and financial industries. CART divides the predictor space of *p* covariates; $X_1, X_2 \ldots X_p$, into *J* distinct and non-overlapping regions,$R_1, R_2 \ldots R_j$. It is generally not practical to consider all possible region splits, so a greedy algorithm, known as recursive binary splitting, is used (Hastie 2009). The approach to partitioning being greedy means that each node is split in the *best* way at the time the split occurs, without attention or consideration to future potential splits. The tree algorithm begins with all data at the root node, then splits that data by some specified criteria on a particular covariate. This splitting of the data in two continues down the tree until a terminal node is reached—a node at which a certain threshold of minimum occupancy is reached after which no further data splits will occur. Test data dropped down the trained tree will result in an observation falling into a particular terminal node based on its particular covariate values, and an estimate (in the case of regression) for the response is taken to be the mean of that response by the occupants of that terminal node from the training data.

2.2.2 Bootstrap Aggregation

Bootstrapping in the general case refers to random sampling with replacement. In the machine learning setting, bootstrapping usually involves the creation of "bootstrapped" data sets; artificially produced data created by repeated sampling with replacement from a larger data set. For use with CART, a number of bootstrapped samples may be produced, and for each the procedure of growing a tree is repeated. The inherent advantages of this approach in this setting are many. By resampling the training data we generate a number of bootstrapped samples B which can each be used to grow a tree. After growing B trees, rather than pruning trees to reduce variance we average all of the fully grown trees to produce our model. As long as we choose large B, we can even use the OOB (out of bag) error rate as an unbiased estimate of the test error (Hastie 2009). This is useful in many cases, as cross-validation methods with a high number of folds will take a computationally long time on large data sets (Hastie 2009). The OOB error rate, which will be defined by the concordance index in the RSF setting, can thus be used to assess the predictive accuracy of the model.

2.2.3 Random Forests

Breiman (2001) extended previous ensemble CART methods to include a random subset of covariates to split on at each node, rather than considering the entire feature space. This extension allows for a significant reduction of bias within the model by decorrelating trees which

would otherwise be grown quite similarly (Hastie 2009). The random forest attempts to improve the bootstrap aggregation approach by reducing correlation between the trees. After taking B bootstrapped samples, while building the trees, we restrict consideration of the predictors at each split to a random subset of those predictors. Generally, with p predictors, $\sqrt{p}$ predictors will be considered at each split. This allows trees to grow which would otherwise have no chance in the original algorithm due to daughter node heterogeneity being highly dependent on splitting for that particular predictor. Random forest is thus a more generalized version of bootstrap aggregation where all predictors were considered at each node.

2.3 Machine Learning Applied to Survival Analysis

This section discusses how machine learning has been leveraged to model survival data, including the specific adaptations of the random forest algorithm cast in the survival setting. In addition, we provide context for the difficulties in optimizing and tuning the machine learning approaches for large data sets, including high numbers of observations and/or a large feature space.

2.3.1 ANN, SVM, Bagging and Boosting

Work by Gong et al. in 2018 compared artificial neural network (ANN) and random survival forest (RSF) approaches at different levels of censoring. The study used varying sample sizes of simulated data ranging from 200 to 1000 observations, with censoring rates ranging from

0 to 75%. It was found that RSF performed similarly for the different sample sizes, and higher censoring rate resulted in only slightly worse model performance (evaluated by Concordance Index). The overall performance was better for the forest models, reinforcing statements by Ishwaran (2008) regarding the efficacy of the RSF paradigm. Osman (2018) stated that random survival forest achieved the best results as compared to other supervised machine learning models. Chen (2013) showed that a boosting algorithm which optimizes the concordance index directly can outperform boosted Cox models and RSF, but the model has not been widely adopted for use and the performance increase was marginal.

Support vector machines, traditionally used for categorical classification, have been adapted to function in the regression domain. These SVM models have been further extended to work in the survival setting, but have not seen the widespread success of RSF or even ANN models (Wang 2017). Van Belle (2011) found concordance indices of 0.7512 for a Cox model compared to 0.7251 and 0.6796 for SVM (two different kernels) when all variables were used, and essentially equivalent error rates when variable selection was performed. SVM for survival requires more decisions be made by the analyst, is not as well documented or implemented in R or Python as RSF, and does not perform implicit variable selection (evidenced by poor performance relative to Cox model in Van Belle (2011) when all variables were used.) We offer this as evidence to the point that random forests, on average, outperform their ML counterparts in the survival setting and are both easier to implement and troubleshoot.

2.3.2 Random Survival Forests

The random survival forest algorithm is an extension of the random forest algorithm, using survival trees. The default algorithm proceeds as follows. (Ishwaran 2008)

Step 1: Draw B bootstrapped samples from the full data set.

Step 2: Grow a survival tree for each of these B samples. At each node, randomly select $\sqrt{p}$ or $\frac{p}{3}$ predictor variables. Split the node using the designated splitting criterion.

Step 3: Grow the tree to the specified constraint, where no terminal node shall be allowed to have fewer than $d_0 > 0$ deaths.

Step 4: The cumulative hazard function for each tree is calculated, and then these are averaged together across all trees to obtain an ensemble CHF.

The choice of splitting rule for our study is to use the adjusted log-rank statistic without pruning of the trees. This criteria was shown to have the best results for bootstrap samples from a real dataset (Radespiel-Troger 2003). As this split rule means all trees will be grown to full depth with no pruning, we will consider different d_0 in our model selection.

The hazard function gives the instantaneous potential per unit time for the event to occur, given that the individual has survived up to time *t*. As such, integrating the hazard function over time gives a probability of failure at time *t*, given survival up to that time. We call this the cumulative hazard function, and this is the target output generated by a single tree in the RSF algorithm. The estimate of the true CHF is the Nelson-Aalen estimator. This estimator sums the

number which got the event relative to the number in the risk set at each ordered failure time. Formally, this estimate is given by equation 2.10:

$$\widehat{H_h(t)} = \sum_{t_{l,h}<t} \frac{d_{l,h}}{N_{l,h}} \tag{2.10}$$

Equation 2.10 sums the ratio of observed events in node *h* at time *l* to the number in the risk set (defined in section 2.1.2 above) at that time (Ishwaran 2008). The summation is indexed over the observed event times for observations residing in node *h.* This estimate of the cumulative hazard function is shared by each node of the tree (denoted by the shared *h* subscript). In this way, test subjects may have their hazard and thus their survival predicted by the terminal node CHF in which they are placed, in the single decision tree setting. For our ensemble forest model, the predicted hazard is taken to be the unweighted mean of the CHF for each tree in the forest.to the "true" cumulative hazard function. This ensemble method for calculating the CHF reduces the bias of the estimate as the number of trees increases (Hastie 2009).

CHAPTER 3
METHODOLOGY

Chapter 3 details how data for this project was selected, cleaned and imported for use in our various analyses. In the second section, we lay out the methods used to generate, assess, and compare models.

3.1 Data Selection

Data was selected from the Surveillance, Epidemiology and End Results (SEER) program. This program "provides information on cancer statistics in an effort to reduce the cancer burden among the U.S. population." (SEER website) SEER provides their data sets to students, researchers, professionals and individuals who complete a personalized SEER Research DATA Agreement form. The original data set includes 193 variables, many of which are redundant or cancer-specific. In addition, many variables exist that seek to quantify the same attribute of a tumor, but use a different standard or system of measurement. As such, many variables were dropped in the preprocessing step in an effort to reduce both noise and highly correlated variables within the data. Particular variables for analysis were chosen with reference to recent literature analyzing pancreatic cancer survival (Wahutu 2016). As an additional step, only data for subjects with a diagnosis year ranging from 2004 – 2015 were initially considered because many attributes only began to be measured and collected in 2004, and many more went

through revisions or definitional changes in the late 1990s and early-mid 2000s. The original complete data set can be found at https://seer.cancer.gov/data/. SEERStat, proprietary relational database software, available at the same link as the full data, was initially used to select only the relevant and desired observations from the overall cancer database.

After data was imported to R the data was processed and cleaned. All figures produced and analysis conducted was done using the R programming language. Observations with unknown values of literature-reviewed variables of importance were not considered, as the imputation of these values would have high impact on the profile of such observations. Next, contrasts with very infrequent occurrences were dropped from certain categorical variables, including the surgery and marital status variables. For some variables, such as marital status and race, categories were combined; for race we tested only the categories of "white" and "other", and for the marital status covariate the level "separated" was joined with "divorced" and relabeled as divorced. Surgery levels were grouped into categories which signify only whether surgery was performed or was not performed, rather than include reasons why it was or was not performed (some of which were quite rare or unknown). The stage variable was grouped by numbered stages, removing 'A' and 'B' designations.

Only data for the diagnosis years 2011 and 2012 are considered. The primary reasons for this include the desire for relatively recent data which are not so recent as to have an incredibly high censoring rate (>50%). In addition, with the extremely low 5 year survival rate for pancreatic cancer, the year of diagnosis variable becomes dominant when the spread of the years is too wide; it effectively predicts survival when the censoring status is considered conjunctively. On a personal note, the author's aunt was diagnosed with pancreatic cancer in 2012, and for that

reason the year was desirable for inclusion. Blaine and Sahak (2011) found that marital status, not thought to be an impactful variable, was quite important for predicting pancreatic cancer survival. We will examine if the importance of this covariate exists in our data set, according to one or both of the methods under consideration.

Figure 1 and Table 1, below, show the variable summaries grouped by numerical and categorical variables. The data, ready for analysis, contains 10 categorical and 3 numerical variables, with the censoring indicator variable only being used in various model statements as required by R. Figure 1 shows the categories or levels for each variable, except month of diagnosis, which has an (Other) category for the unlisted 6 months; the default for the summary statement in R displays only the first 6 levels, and the counts for all months are relatively close.

Table 1: Numerical variable summary statistics.

n = 4320	Mean	STD	Min	Max
Time	18.825463	17.1599155	0	59
Tumors	1.029167	0.1763546	1	3
Age at Diagnosis	65.409259	11.5507520	18	98

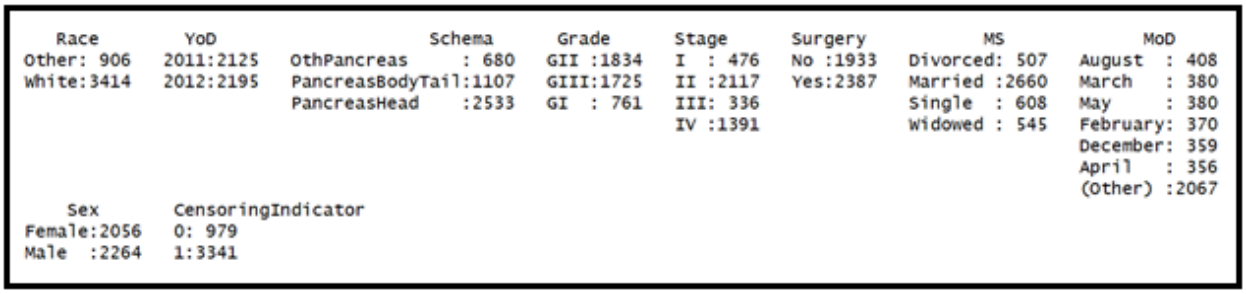

Figure 1: Categorical variable summary, broken down by contrast counts.

3.2 Models and Parameter Estimation

This section discusses model assumptions and analyst decisions related to model fitting.

3.2.1 Cox Methodology

To perform a rigorous Cox PH analysis, we must first check that the proportional hazards assumption is not violated (or not *badly* violated) for each covariate. Figures 2 shows the KM log-log survival function estimates for two covariates.

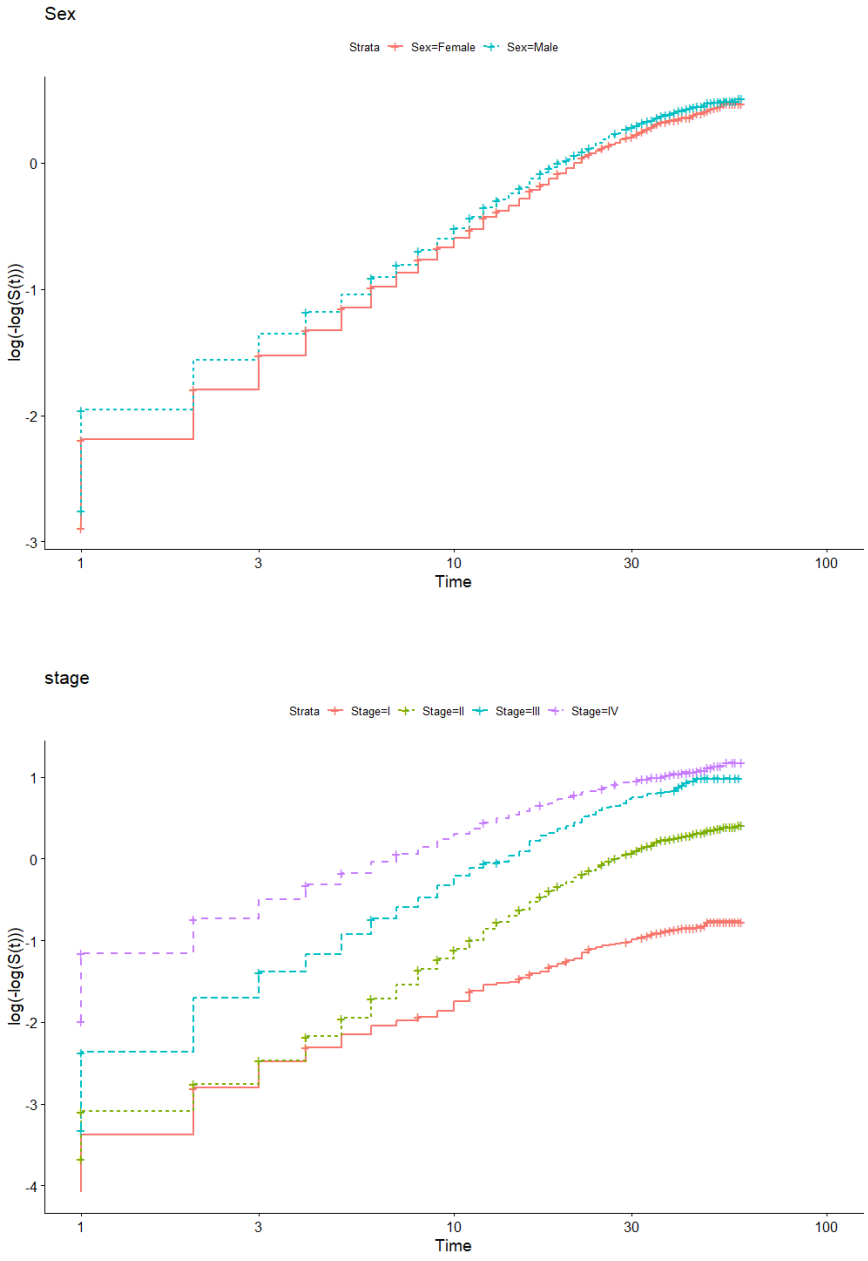

Figure 2: KM log-log survival estimate for Sex and Stage covariates

These plots are produced by taking the negative log of the negative log of the estimated survival probability for each level of each covariate and plotting the result as a function of time. The estimated survival probabilities are the Kaplan-Meier estimates discussed in 2.1.1. The Age at Diagnosis variable was partitioned into categories of "Young", "Middle-Age" and "Old" so that the proportional hazards assumption could be tested. The ages corresponding to each of these categories were 50 and younger, 51 – 70 years old, and 71+. The parallel nature or constant distance of lines in these plots means that the proportional hazards assumption of the Cox PH model is not violated for that predictor; thus the assumption holds for race, surgery, sex, year of diagnosis, month of diagnosis, tumors, and age group. If the lines are not quite parallel, but remain very close over time, it is likely that the variable will not be important enough to include in the model and so any perceived violation will be ignored. There are four covariates for which the proportional hazards assumption violation must be dealt with; the stage, grade, marital status and schema variables, which show a non-constant vertical distance between the lines at later survival times, indicating the PH assumption is not valid. Further testing for violations of the PH assumption may be conducted by performing a hypothesis test of the correlation between ranked survival time and the Schoenfeld residuals. (Kleinbaum 2012) The R code included in Appendix 2 details this process using the cox.zph function. We test a null hypothesis of no correlation vs an alternative hypothesis that the correlation is not equal to zero. Table 2 shows the results of our tests for the variables of interest with factor levels tested against an assigned reference level, with a chosen significance level of $a = 0.01$.

Table 2: Test statistic and p-values for proportional hazards test using cox.zph.

	Rho	Chi-Squared	p-value
SchemaPancreasBodyTail	0.007397402	0.18780043	6.647536e-01
SchemaPancreasHead	0.023785208	1.89448539	1.686969e-01
GradeGIII	-0.074476252	18.46091316	1.734249e-05
GradeGI	-0.056631936	11.15986077	8.358620e-04
StageII	0.061746421	13.00509026	3.106454e-04
StageIII	0.059636247	12.78065958	3.502214e-04
StageIV	0.012732983	0.57712114	4.474430e-01
SurgeryYes	0.090296227	26.44598067	2.710118e-07
MSMarried	0.038678946	5.04274031	2.472934e-02
MSSingle	0.017784896	1.06456074	3.021773e-01
MSWidowed	0.004993813	0.08577622	7.696171e-01

The marital status levels “single” and “widowed” have large p-values, while the “married” level has a p-value of approximately 0.02 for this test; not enough to reject at the 0.01 level. This is, however, quite close to a statistical violation, and would be a violation at the 0.05 level; future work may wish to move forward with the conclusion that PH assumption is violated for the “married” level vs the “divorced” reference level and handle it accordingly, but this work will conclude that the PH assumption is satisfied overall for the marital status covariate. The schema variable provides a much more clear decision to fail to reject the null hypothesis; with p-values 0.58 and 0.19, we conclude that schema does *not* violate the PH assumption, despite apparent visual evidence to the contrary. The very low p-values for stage and grade reinforce our visual inspection of the log-log KM plots and highlight the need for a modification to our model framework which can handle covariates that change over time. In addition, the statistical test for significant violations of the PH assumption makes us aware of a violation that was not obvious from the KM plot visualizations; the surgery variable shows a significant violation and must be dealt with.

The violations of the PH assumption by surgery, grade and stage necessitate the implementation of a more robust model. One methodology involves stratification of variables which violate the PH assumption, however this approach will result in the fitting of different models for each strata and disallow direct observation of overall covariate effects. The 5-year survival rate difference for Stage I vs Stage IV pancreatic cancer is 31%, so the stage variable is likely important and stratification will not yield the desired results. (American Cancer Society) The violations by surgery, grade and stage will be handled by fitting a Cox model with time dependent coefficients. We should expect that the stage of cancer would change with time for

many individuals; stage can roughly be thought of as describing the locality of the tumor, and over time cancer tends to grow and spread out. Our variable only considers the stage *at diagnosis*, and does not include information about how stage might evolve. In general cancer will have spread for patients that die from their cancer, while some subjects will see an overall reduction in the amount of cancerous tissue present via various forms of treatment. Although re-staging is uncommon after diagnosis, our use of a time-varying stage variable captures the essence of what "stage" measures and resolves the violation of the PH assumption.

Thomas (2014), Zhang (2018), and Therneau (2019) lay out formulations in R to resolve PH assumption violations through the use of time-dependent coefficients. The functions and methods referenced in this section were carried out in R and the code is included in the appendix. Most approaches require that we convert our data to the "counting process" format described in Kleinbaum (2012), Thomas (2014) and many other places. This results in a data structure that has multiple rows for each subject, and "counts" up from subjects which got the event time earliest to those that got it later on. For each subject, there is a row for each event time up until that subject dies, with the covariate values for that subject recorded for each interval. One alternative, employed here, is to use the built-in time-transform option in the coxph function in R; this works around using the counting-process data, which in our case has over 81,000 rows when formed.

$$\beta(t) = a + b\log(t). \tag{3.1}$$

Equation 3.1 is a commonly used form for a time-dependent coefficient (Therneau 2014), (Zhang 2018). When this function of time is multiplied by a covariate $\boldsymbol{X}$, it results in equation 3.2:

$$\beta(t)X = aX + b\log(t)X. \quad (3.2)$$

Figure 3 shows the result of plotting the coefficient estimate for each contrast which violates the proportional hazards assumption as a function of time from the overall model with all covariates. The horizontal lines represent the mean coefficient, which is what is used in the fitted model and reported by the coxph model statement. Violations of the PH assumption may be seen visually as significant deviations from this line by the estimates; non-constant coefficients with a time dependence result in a hazard ratio that will vary over time. Figures for all covariates are in Appendix A.2.

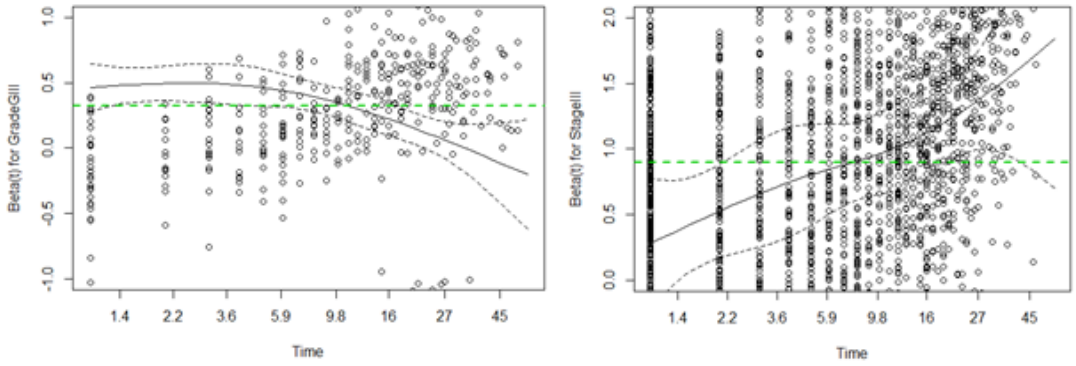

Figure 3: Plot of coefficient estimates for selected contrasts from Grade and Stage variables, with dashed lines showing average estimates.

Including in the model a time dependent term for each contrast in violation may fix the issue. The figures here show that a linear transformation will be a much better fit in each case of violation, though the fit for the surgery variable is much less clear (see figure in Appendix). While we will end up fitting a linear time dependency to surgery, future work may consider fitting something like a logistic function to more closely model the behavior. Note that stage IV had a non-significant p-value but the figure is included for comparison. Figure 4 shows the beta estimates for each contrast in the form described by equation 3.2 above. The fit of these lines is quite good, and our inclusion of a time varying coefficient has resulted in a satisfaction of the original proportional hazards assumption.

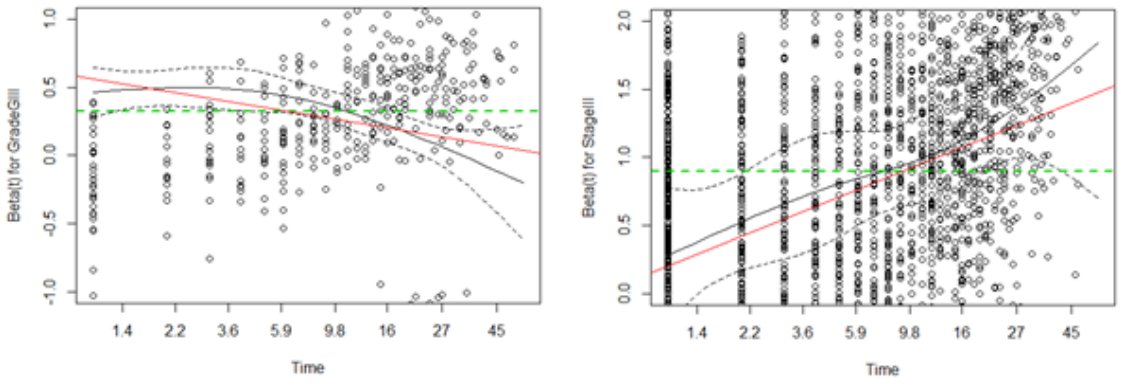

Figure 4: Linear functions of time (red) show better fit than dashed line to coefficient estimates

We begin our Cox regression by checking the univariate model for each covariate to assess significance. We choose a significance level of $\alpha = 0.01$ for entry into the model. For categorical predictors, testing significance in the univariate case requires creating dummy variables, which is done automatically via the *fastDummies* package in R and uses a reference level as determined by the software. For categorical predictors with more than two levels, we get a p-value for each level as compared to the reference; thus we will deem significant any variable which has any category with a significant p-value. The results of this univariate model fitting is Table 5 (in Appendix A; not all months shown, but none significant). We see that the variable 'Race' has a p-value outside of our range, and so we drop this variable from further consideration. In addition, the month of diagnosis variable appears to be noise, with high p-values for all contrasts, and thus is removed before the backward selection procedure.

With the removal of two insignificant variables, we fit the remaining variables and consider the resultant full model. Following a backwards selection procedure, we remove the variable with the highest p-value from the subset which meet our criteria for removal from the model; $p > 0.05$. In this way the year of diagnosis variable is removed. Next, schema is removed, and the remaining variables are fit. The final variable dropped via backwards selection is the sex variable. Upon completion of the backwards procedure, a forward procedure is employed to consider variables removed during the first procedure which may actually be important in the absence of since-removed covariates. The model with current covariates and sex has already been considered, so the first forward selection variable that will be added is schema. It is found that the schema variable is still not important enough to include in the model.

The next variable we add is the year of diagnosis variable; again the result is a p-value that is not small enough to let year of diagnosis back into the model. We have now arrived at a final model, shown in Table 3 below.

Table 3: Final Cox model parameter estimates and p-values

	Beta	exp(Beta)	p-value
Tumors	-0.71647461	0.4884713	3.110219e-08
AaD	0.01908667	1.0192700	6.187909e-29
MSMarried	-0.21265564	0.8084345	1.107868e-04
MSSingle	-0.02921828	0.9712044	6.688272e-01
MSWidowed	-0.05581927	0.9457100	4.317647e-01
Grade_GI	-0.23964405	0.7869079	6.718251e-02
tt(Grade_GI)	-0.17128797	0.8425789	1.481076e-03
Grade_GIII	0.54641272	1.7270465	5.227177e-13
tt(Grade_GIII)	-0.12187087	0.8852627	2.957390e-04
Stage_IV	1.17041981	3.2233455	2.527836e-38
Stage_III	0.23770007	1.2683287	1.375902e-01
tt(Stage_III)	0.32323763	1.3815936	1.392850e-07
Stage_II	0.26931398	1.3090661	4.631469e-02
tt(Stage_II)	0.26540583	1.3039601	5.898010e-09
Surgery_Yes	-1.87958523	0.1526534	7.351247e-56
tt(Surgery_Yes)	0.34972535	1.4186779	7.467198e-12

3.2.2 Random Forest Tuning

The random survival forest algorithm is capable of taking many different user inputs as tuning parameters. In R, the widely used 'randomForestSRC' package's main function 'rfsrc' lets the user set nine parameters for supervised forests which will directly affect model performance. Beyond the 'ntree', 'mtry', 'nodesize' and 'splitrule' parameters that we will discuss in greater detail later in this section, the package offers other ways to tune the algorithm. The 'nodedepth' parameter specifies the maximum depth that trees will be grown to; this is useful for speeding up computation, but setting 'nodesize' will be our preferred method of controlling the size of trees in the forest. 'nsplit' is another parameter that may be used to speed up execution by defining a maximum number of split points to be chosen at random from the possible splits each variable. The default for 'nsplit' is 10, which may seem low, but over a large number of forests this parameter will become less relevant to our results. In addition, our data contains mainly categorical variables, so we will be much closer to deterministic splitting than in the case where there are a large number of continuous covariates.

There are several parameters that allow the user to define various weight vectors; a vector to assign weighted probabilities to the chance of each variable being selected at each split, a vector of weights to adjust the split statistic for each variable, and a vector capable of assigning weights to each observation's probability of being selected for each bootstrapped sample. Our last available parameters involve changing the way our bootstrapped samples are created. We do not create any weight vectors nor change the default bootstrapping process for this project. In fact, the only tuning parameters we will vary are ntree, mtry and nodesize. This follows

empirical practices with random forests (survival and otherwise) for which these three parameters have been found to have the greatest impact on model performance (Schmid 2016, Ishwaran 2008, Ehrlinger 2016)

The 'randomForestSRC' package in R used for this analysis allows the user to choose between two split rules based on the log-rank statistic. The two rules, logrank and logrankscore, were found to have very similar overall performance across a diverse array of data sets (Ishwaran 2008). The 'logrankscore' rule is much more computationally intensive, and so based on the very similar performance of the two rules we will only consider the log-rank criteria for splitting. Having settled on a singular split rule, we turn to the model parameters that will have a great impact on prediction. The parameters of interest to us are as follows: the number of trees in each forest (ntree), the number of variables to try at each split (mtry), and the average number of unique cases for each terminal node (nodesize).

Increasing the number of trees in the RSF algorithm results in a linear increase in computation time. We begin with a forest of size 50 based on empirical data showing the OOB and prediction errors settling down between forest sizes of 50 and 100 trees (Hastie 2009). We increment the forest size by 50 up to 250 trees, and then double that to a forest of 500 trees. We recall that choosing the number of trees in the forest is equivalent to choosing how many bootstrapped samples to draw from our original data set. Probst (2018) argues that one should not attempt to tune the number of trees in a random forest; rather, the largest number of trees which is computationally feasible should be used. This begs the questions; what number of trees is feasible? A starting point for choosing the number of trees might be the default in the rfsrc package, $ntree = 1000$. As a data set grows in both features and number of observations, 1000

trees becomes quite cumbersome, especially if other parameters require tuning. As such, we consider various numbers of trees in the range 50 – 500 for our first grid search, but include tests for improvement based on a large number of trees in later analyses. It seems that only in rare cases, and depending on the convergence properties of the performance metric, should one attempt to tune the number of trees; it is almost always better to use a larger forest (Probst 2018).

The number of variables to try at each split is traditionally taken to be $\sqrt{p}$, where p is the total number of predictor variables under consideration. The tradeoff for considering more variables is higher correlation between trees; with all variables considered at each node split, trees will be grown quite similarly, especially if there are particular variables with much higher relative importance (Shalev-Shwartz 2014). We have 11 predictor variables that we may consider at each split, and the default will round up, in our case as $\sqrt{11} = 3.32 \approx 4$. Of course consideration of only one random variable at each split is not desirable, so we will consider a lower range of mtry = 2, all the way up to mtry = 11 per split. Considering all 11 covariates and an arbitrarily large value for nsplit will result in deterministic splitting. The nodesize parameter is quite interesting, as stated in Scornet (2018), "There is no more theoretical supports for choosing nodesize = 1 or nodesize = 5…" Considering this, and the default nodesize in our algorithm of 15, setting reasonable bounds on this parameter is not easy. Much lower values will tend to overfit, and very high values won't provide as much predictive power. We consider terminal node sizes of 1, 15, 50, 100, 500 and1000 in our first tuning pass. This massive range should allow us to gain actionable insight and refine our search in subsequent attempts.

Out-of-bag (OOB) error, as discussed previously, provides an estimate of prediction error for algorithms that use bootstrap aggregation. In order to verify that OOB error is a good

estimate for prediction error, we perform cross validation for our first tuning pass, and compare the average prediction error over the validation sets for each combination of parameters

For our random survival forests, *k*-fold cross validation was used, with k=10. This choice of 10 folds is made based on empirical evidence, and an understanding of the bias-variance trade-off; choosing $k=n$ (known as leave-one-out cross validation or LOOCV) results in substantial bias reduction from the single validation set approach but has higher variance than any choice of k such that $k<n$ (Hastie et al 2009). With this 10 fold validation approach, we take the average prediction error for each combination of tested tuning parameters across 10 forests. Training/validation sets were created and used in the following way:

1) Randomly shuffle the full data set
2) Based on our choice of k=10, create 10 "folds" of size $foldsize = floor(0.10 * n)$
3) Construct 10 training data sets by removing the observations in each fold from the full data. These training sets will each contain about 90% of the full data set
4) The corresponding folds, each containing about 10% of the full data, are used as test sets
5) We train a model for each considered combination of tuning parameters on the newly created training data
6) Test each model on the specific left-out test data by dropping each test set observation down the tree object until a terminal node is reached
7) The predicted survival time for the test observation is taken to be the average of all observations that occupy that terminal node

In this way, 10 lists are created, each containing the prediction error corresponding to all possible tuning parameter combinations, indexed in such a way that allows for easy access without loss of model identification. A cross-validation approach requires much more computation than simply using the OOB error rate as a predicted error rate. Hastie (2009) has shown that for random forests not cast in the survival setting, the OOB error estimates are almost identical to those from cross validation. Ishwaran (2008) similarly advocates the use of the out-of-bag data for testing purposes rather than the traditional machine learning approach of using a validation set. Probst (2018) also advocates the use of the OOB error rate, stating that it is "commonly considered an acceptable proxy" for cross-validation. The intent of this paper was, as a matter of consistency, to demonstrate that the OOB error rate and cross-validated error rate are quite similar and lead to negligible differences in the final model. Table shows the top ten candidate models after considering all possible permutations of tuning parameters. The avg.error is the error obtained from averaging all ten validation set error rates for each parameter configuration, while the oob.err entries show the out-of-bag error rate for each model as a result of running randomForestSRC on the full training data set. The figures are sorted by CV error rate and OOB error rate. The variance in the cross-validation error rates over the ten folds is reported as well.

Table 4: RSF models sorted by OOB error (top) and CV error.

model.id	ntree	mtry	nodesize	avg.error	variance.err	oob.err
6	50	2	1000	0.2584344	0.0001443431	0.2495664
22	50	11	100	0.2492617	0.0001817594	0.2500950
12	50	5	1000	0.2601304	0.0001027513	0.2501017
36	150	5	1000	0.2584331	0.0001304053	0.2503761
11	50	5	500	0.2524952	0.0001635120	0.2503918
49	250	2	1	0.2600589	0.0002701416	0.2506387
14	50	8	15	0.2569611	0.0002165708	0.2507564
23	50	11	500	0.2510645	0.0001808132	0.2509462
21	50	11	50	0.2516330	0.0001955604	0.2510348
31	150	5	1	0.2659497	0.0002319553	0.2511316

model.id	ntree	mtry	nodesize	avg.error	variance.err	oob.err
34	150	5	100	0.2485021	0.0001624753	0.2684679
58	250	5	100	0.2485348	0.0001773919	0.2562163
64	250	8	100	0.2486952	0.0001760475	0.2520318
10	50	5	100	0.2487378	0.0001533637	0.2546730
28	150	2	100	0.2488196	0.0001662184	0.2652672
70	250	11	100	0.2488628	0.0001750828	0.2737937
52	250	2	100	0.2489499	0.0001687009	0.2694060
46	150	11	100	0.2491526	0.0001751987	0.2587786
27	150	2	50	0.2492172	0.0001756579	0.2519887
22	50	11	100	0.2492617	0.0001817594	0.2500950

From the previous figures it can be seen that only one model is in both of the top ten; model.id 22, with $ntree = 50, mtry = 11$ *and* $nodesize = 100$. The best model when sorted by OOB error has an average CV error of 0.258. The best model by CV error rate has an OOB error rate of 0.268. These error rate differences are quite large, and lead to some interesting observations. Half of the best by OOB models have nodesize values of 500 or more, while the best by CV models all have nodesize values of 100 (50 for one). It may be the case that large terminal node sizes, which lead to shallow forests, tend to overestimate test set error rate when using out-of-bag error. The relatively uniformity of the variances in the CV error rates is expected, and welcome. High variances might mean overfitting was taking place within some of the folds, but this is not seen. It was the intention to move forward using OOB error rate as a stand-in for cross-validated error, but the evidence suggests we abandon this strategy in favor of the CV approach. The price we pay for this is time and computational resources.

Employing a grid-search approach for the best tuning parameters requires fitting many different models and honing in on the best combination of tuning parameters by refining the grid across repeated trials. We commit to three "passes" of tuning and subsequent refinement, with the option for more if there is continued significant improvements in the CV error rate. For our second pass we use the top models of the first pass to observe trends and characteristics to refine our search. The best models in the first pass all had terminal node sizes of 100 (one had 50), suggesting that the nodesize parameter was the most important based on the considered values of all others. Three of the top four models have mtry values of 5, suggesting we continue our search with $mtry = 5$ as a mid-point for that parameter. The number of trees in the grown forest appears relatively unimportant as compared to the other two parameters, as expected. We

explicitly did not consider sizes of ntree which would allow the parameter to dominate (forests of size less than 50 in particular have high variance according to Hastie (2009)).

We consider forests of size 150 and 250, dropping the 50 tree forests from consideration; this is done because the computation time for 150 and 250 trees is not prohibitive, and Probst (2018) lays out strong arguments for using as a big a forest as computationally feasible. For the number of variables at each split we consider 3 – 8, and 11. Due to terminal node size 100 dominating the top ten models, we use sizes of 100, 125, 150, 200 and 250 in an attempt to capture the effect; if the best models still use $nodesize = 100$, we will refine our search with 125 as the maximum and work down. The results of this second pass of tuning is seen in Table 13 in Appendix A. We see that the top five models all result from $mtry = 6$ or $mtry = 7$, and $nodesize = 150$ or $nodesize = 200$. Additionally, the error for the best model was an improvement over the previous best model. We move forward by considering only $mtry = 6$ and $mtry = 7$, keeping the number of trees under consideration to the same values of 150 and 250, and attempt to find the optimal nodesize parameter by increasing the resolution substantially over previous iterations. The nodesize values considered range from 150 to 190, in increments of 5. The result of the third tuning pass is presented in Table 13 (bottom). The error rate improvement over the best model from the second pass is marginal, and the improvement is very much within one standard deviation of the CV error rates. It is clear from this figure that $mtry = 7$ is optimal for our data, as the top eight models all have this parameter value. The best model with a different value of mtry has a nodesize value of 165, the same as the best overall model, suggesting this may be the optimal setting here. With the small improvements

over the previous parameter permutations, we cease our grid search approach and conclude that a model with $ntree = 250, mtry = 7$ and $nodesize = 165$ is best.

CHAPTER 4
DATA ANALYSIS AND DISCUSSION

The final Cox model includes the covariates: tumors, age at diagnosis, marital status, grade, stage and surgery. The summary output for the final model is shown in Table 12 in Appendix A. The concordance index for the final Cox model is 0.765. The concordance index for the Cox model is calculated by considering predicted survival times from the model versus actual survival times from the full (training and test) data. For the random survival forest, concordance index is used as the model selection statistic; thus the final forest model must be run on the test data to determine the concordance for comparison to the final Cox model. A plot of survival for the final model run on the test data and forest model output are shown in Figures 12 and 13. Figure 12 also shows the final model performance using a forest of the largest feasible size, 10000, as suggested by Probst (2018). Note that the rfsrc function outputs an error rate that is equivalent to 1 – C, where C represents the concordance, allowing for easy calculation and comparison to the Cox model. The forest with 10,000 trees has a very slightly higher error rate than the 250 tree forest; as expected, increasing the number of trees to this extent will not result in a continued drop in error rate. Dropping the test set observations down the final forest model results in a concordance index of $1 - .248 = 0.752$.

The concordance index for the Cox model is 1.3% higher than that of the random survival forest; the error rates are quite close, but the difference is significant. Ishwaran (2008) found a

slightly lower error rate for RSF; the inverse of our result. However, the best concordance index found for any of his cancer data set studies was ~70%. Zhou and Mcardle (2015) had a best-model error rate of 24.65% for their recidivism data (with 25% censoring) RSF model, which falls in line with the error of our final model.

The $exp(Beta)$ column in Table 12 shows the expected hazard ratio for two subjects whose value for a given covariate differs by one unit, holding all other covariates constant. This means that covariates which have a value in that column less than 1, such as Tumors, a decrease in the number of tumors by one results in a lower hazard for the first subject and thus a hazard ratio less than 1, as we would expect intuitively; fewer tumors is better for survival. For time-transformed covariates, the interpretation of the coefficients is generally the same, but includes the understanding that the hazard ratio changes with time, and the value of β which is to be exponentiated consists of an 'intercept' value and a 'slope' (time dependent) part, which taken together along with a value for time *t* fully describe the hazard ratio for two subjects whose value for that covariate differ by one unit, with all others held constant. Also in Figure 4.1, we see that every covariate included in the final model has a p-value less than 0.01, though not all levels of all categorical variables have significant p-values.

Interpretation of the final random forest model is not as simple as the Cox model, nor as simple as describing a single CART of which the model makes use. While the final forest model may work in the same way as a single binary decision tree, with new observations being "dropped" down the tree structure, finding its way to a terminal node by following the split rule defined at each node of the tree, this interpretation ignores the ensemble methodology by which the splits were determined. A simplistic description of the final forest must include a summary

description of CART in general, the specific rules used to split the tree, and the notion of ensemble ‘voting’ to determine the final model structure. Even then, if trying to interpret how the hazard rate compares for two subjects residing in two different terminal nodes, the forest offers no insight beyond a predicted survival time for each subject; a Cox-like interpretation of the hazard ratio based on model parameters eludes us. This weakness of the forest approach may be written off in cases where model performance outweighs interpretability, but for our study the performance of the two models is nearly identical, and in fact the Cox model exhibits slightly better concordance

CHAPTER 5
CONCLUSION AND FUTURE RESEARCH

We have shown that a carefully constructed Cox Proportional Hazards model can (slightly) outperform a finely-tuned Random Survival Forest model for pancreatic cancer survival data. The Cox model retains interpretability, but the construction of the model required a much deeper knowledge of statistics and the interpretation, while able to be stated simply once the model is assembled, is not obvious to the lay-person. In contrast, the RSF model could be constructed by anyone with the ability to read data into R and load the required package; the trade-off for simplicity being a much longer time requirement to tune the model for best performance. The work in this paper was completed on a fairly good personal computer, and each iteration of tuning using a grid-search approach took many hours to complete. Of course, cloud computing could be leveraged to complete the work in less time at a higher monetary cost.

Future research would aim to pin down the various regimes in which the Cox PH model may fail to compete with the predictive power of the random forest model, by use of completely different data which may contain many more covariates than observations, for example. Of course the data used in our study comes from real people who have contracted this most deadly cancer, but many datasets nowadays will not have the nice properties of ours, such as the large number of observations relative to the number of possible predictor variables. Further work in this area should also explore other possible time-transformations than *log(t)*, though for our purposes this generic and commonly used function of time fit very well to the data. As a final

thought, it would be interesting to use the work conducted here to test the final models on the most recently available data; truly, the goal of an analysis like this is to offer an accurate survival prediction for newly diagnosed subjects and perhaps allow those subjects to learn what the likely affect to their hazard rate might be by electing to receive treatment in the form of surgery.

REFERENCES

Cox, D. R. Regression Models and Life-Tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, *34*(2), pp. 187-220.

Kalbfleisch, J. D., & Prentice, R. L. (2002). The Statistical Analysis of Failure Time Data, Second Edition. Wiley-Interscience. 1-22, 95-118.

Wang, P., Li, Y., & Reddy, C. K. (2017). Machine Learning for Survival Analysis: A Survey. arXiv:1708.04649 [cs.LG].

Gong, X., Hu, M., & Zhao, L. (2018) Big Data Toolsets to Pharmacometrics: Application of Machine Learning for Time-to-Event Analysis. *Clinical and Translational Science*, *11*(3), 305-311. doi: 10.1111/cts.12541.

Peto, R., & Peto, J. (1972). Asymptotically Efficient Rank Invariant Test Procedures. *Journal of the Royal Statistical Society: Series A, 135*, pp. 185-207. doi: 10.2307/2344317.

Segal, M.R. (1988) Regression Trees for Censored Data. *Biometrics, 44*(1), pp. 35-47. doi: 10.2307/2531894.

Leblanc, M., & Crowley, J. (1993). Survival Trees by Goodness of Split. *Journal of the American Statistical Association, 88*(422), pp. 457-467.

Hothorn, T. and Lausen, B. (2003) On the Exact Distribution of Maximally Selected Rank Statistics. *Computational Statistics & Data Analysis*, 43, 121-137. doi: 10.1016/S0167-9473(02)00225-6

Ishwaran, H., Kogalur, U.B., Blackstone, E.U., & Lauer M.S. (2008). Random Survival Forests. *The Annals of Applied Statistics, 2*(3), 841-860. doi: 10.1214/08-AOAS169

Raykar, V.C., Steck, H., & Krishnapuram, B. (2007). On Ranking in Survival Analysis: Bounds on the Concordance Index. *Advances in Neural Information Processing Systems 20*, recovered from: https://papers.nips.cc/paper/3375-on-ranking-in-survival-analysis-bounds-on-the-concordance-index.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*, Second Edition. Springer. 295-312.

Breiman, L. (2001). Random Forests. *Machine Learning, 45*, 5-32.

Chen, Y., Jia, Z., Mercola, D., & Xie, X. (2013). A Gradient Boosting Algorithm for Survival Analysis via Direct Optimization of Concordance Index. *Computational and Mathematical Methods in Medicine, 2013*, Article ID 873595. doi: 10.1155/2013/873595

Van Belle, V., Pelckmans, K., Van Huffel, S. & Suykens, J.A.K. (2011). Support Vector Methods for Survival Analysis: A Comparison between Ranking and Regression Approaches. *Artificial Intelligence in Medicine, 53*(2), pp. 107-118.

Radespiel-Tröger M., Rabenstein, T., Schneider, H.T., & Lausen, B. (2003). Comparison of Tree-Based Methods for Prognostic Stratification of Survival Data. *Artificial Intelligence in Medicine, 28*(3), pp. 323-41.

Wahutu, M., Vesely, S., Campbell, J., Pate, A., Salvatore, A., & Janitz, A. (2016) Pancreatic Cancer: A Survival Analysis in Oklahoma. *J Okla State Med Assoc. 109*(7-8), pp. 391-398.

Blaine, M., Sahak, F., Lin, C., Chakraborty, S., Lyden, El, & Batra, S. (2011). Marital Status and Survival in Pancreatic Cancer Patients: A SEER Based Analysis. *PLOS One*. doi: 10.1371/journal.pone.0021052.

Thomas, L., & Reyes, E.M. (2014). Tutorial: Survival Estimation for Cox Regression Models with Time-Varying Coefficients using SAS and R. *Journal of Statistical Software, 61*(1), 1-23.

Zhang, Z., Reinikainen, J., Adeleke, K.A., Pieterse, M.E., & Groothuis-Oudshoorn, C.G.M. (2018). Time-Varying Covariates and Coefficients in Cox Regression Models. *Annals of Translational Medicine 6*(7): 121. doi: 10.21037/atm.2018.02.12.

Therneau, T., Crowson, C., & Atkinson, E. (2019). Using Time Dependent Covariates and Time Dependent Coefficients in the Cox Model. *Available on CRAN.*

Schmid, M., Wright, M.N., & Ziegler, A. (2016). On the Use of Harrell's C for Clinical Risk Prediction via Random Survival Forests. *Expert Systems with Applications, 63*(30), pp. 450-459. doi: 10.1016/j.eswa.2016.07.018

Ehrlinger, J. (2016). ggRandomForests: Exploring Random Forest Survival. arXiv:1612.08974 [stat.CO]

Probst, P., & Boulesteix, A. (2018). To Tune or Not to Tune the Number of Trees in Random Forest. *Journal of Machine Learning Research 18,* 1-18. arXiv:1705.05654 [stat.ML]

Scornet, E. (2018). Tuning Parameters in Random Forests. *ESAIM: Proceedings and Surveys, 60,* pp. 144-162

Zhou, Y., & McArdle, J.J. (2015). Rationale and Applications of Survival Tree and Survival Ensemble Methods. *Psychometrika, 80*(3): 811-833. doi: 10.1007/s11336-014-9413-1

Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press. 250-256.

APPENDIX A

ADDITIONAL TABLES AND FIGURES

A.1 Figures Related to Assessment of PH assumption Violations

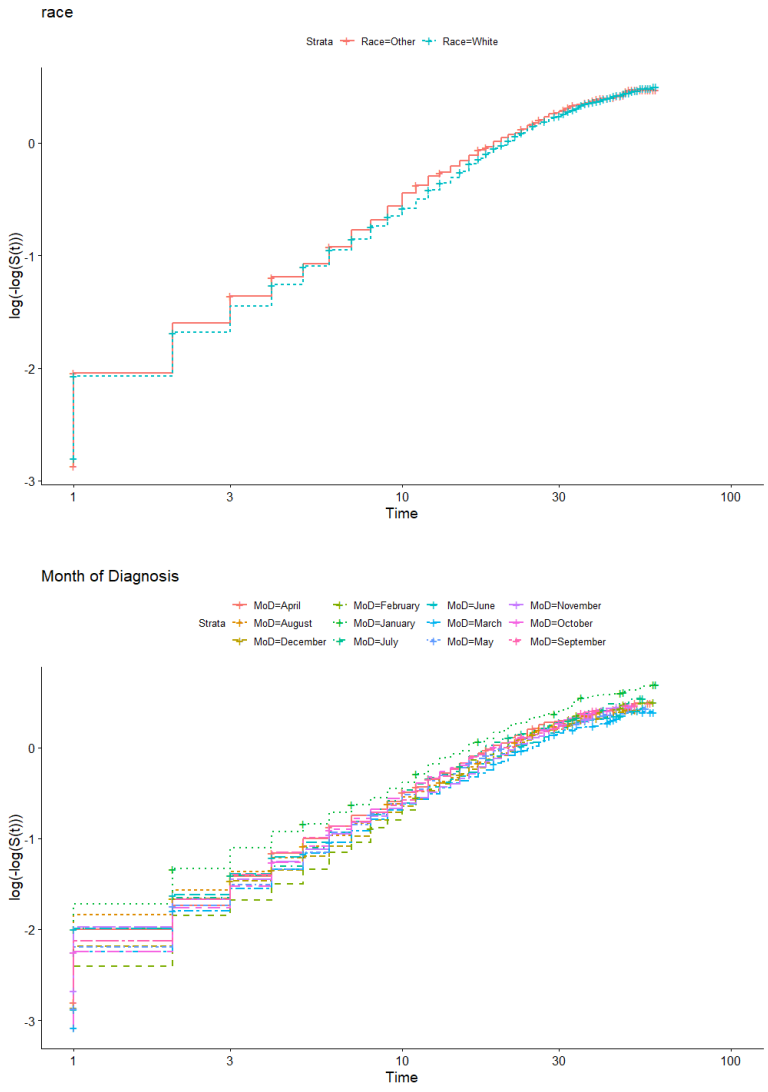

Figure 5: Log-Log KM Survival Estimates for Race and Month of Diagnosis

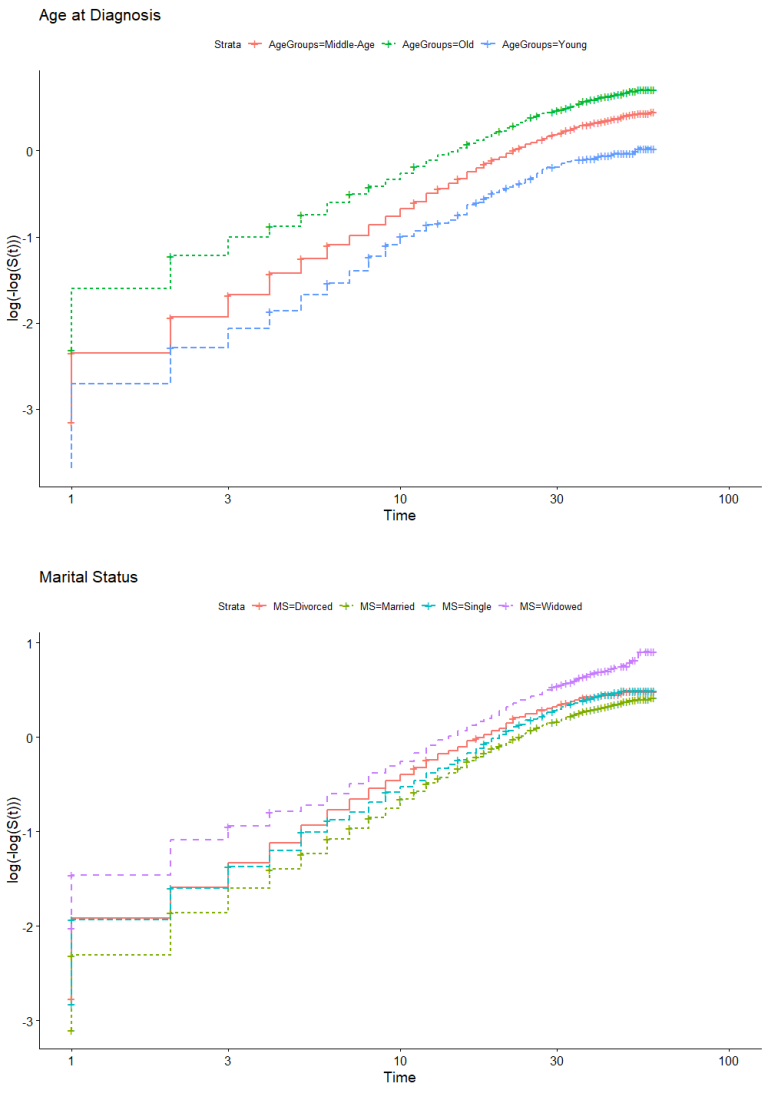

Figure 6: Log-Log KM Survival Estimates for Age and Marital Status

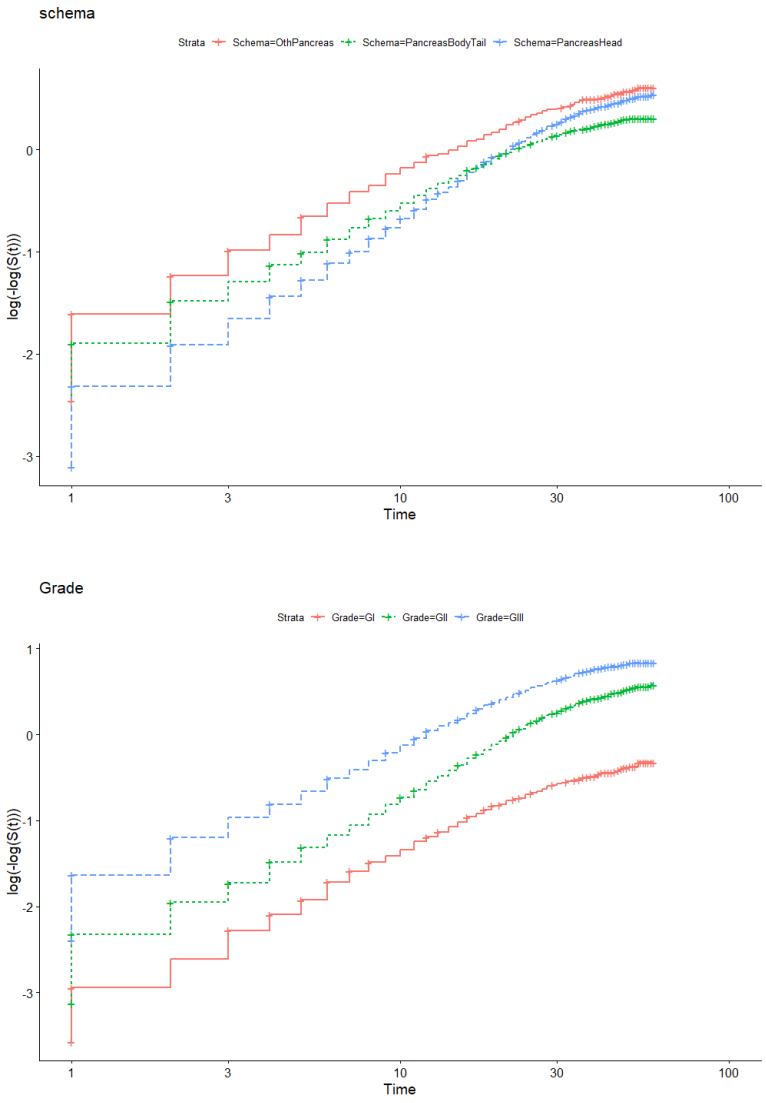

Figure 7: Log-Log KM Survival Estimates for Grade and Schema

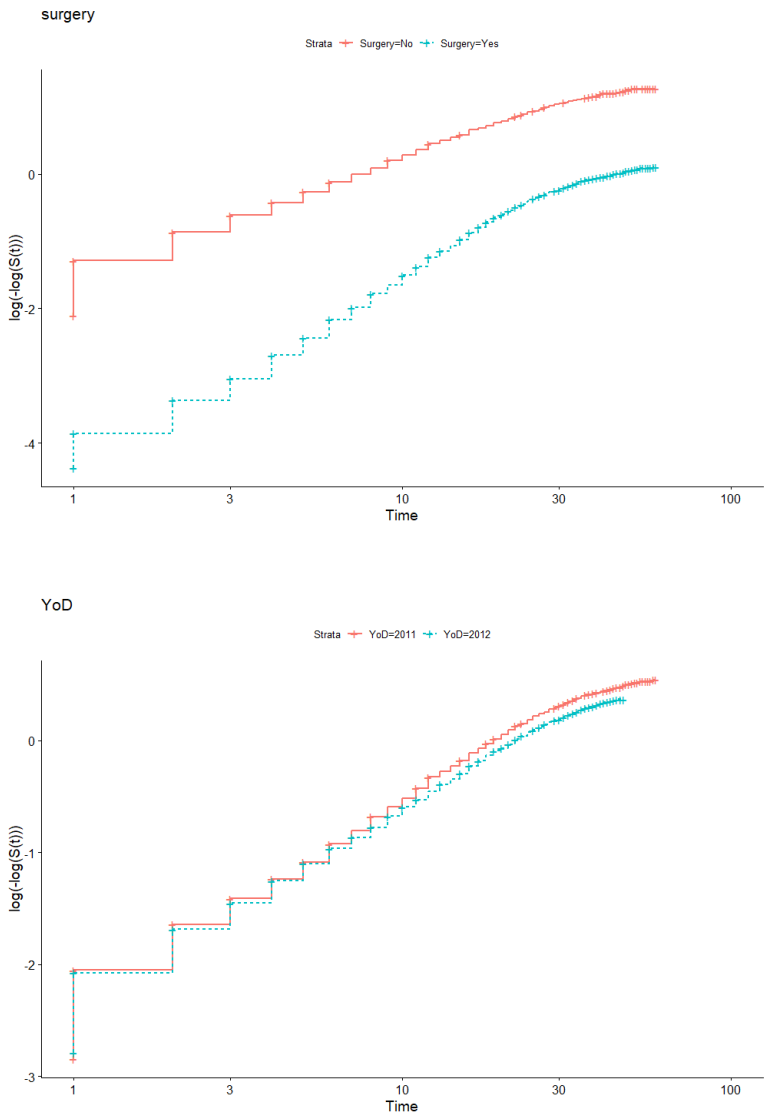

Figure 8: Log-Log KM Survival Estimates for Surgery and Year of Diagnosis

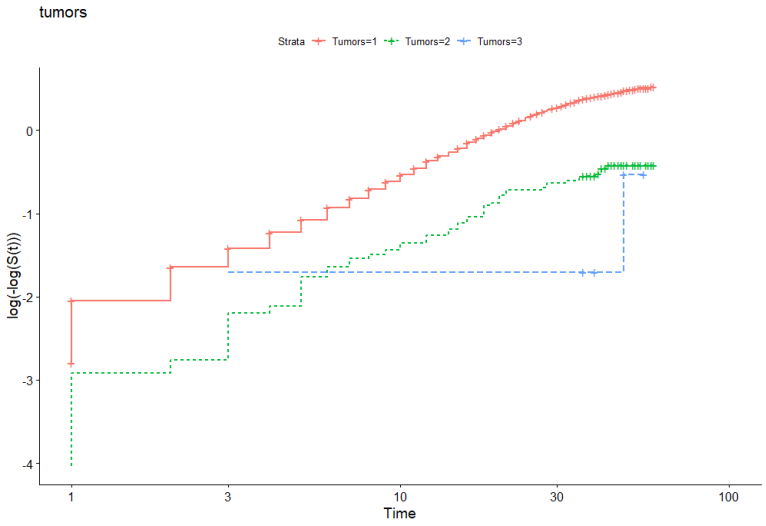

Figure 9: Log-Log KM Survival Estimate for Tumors

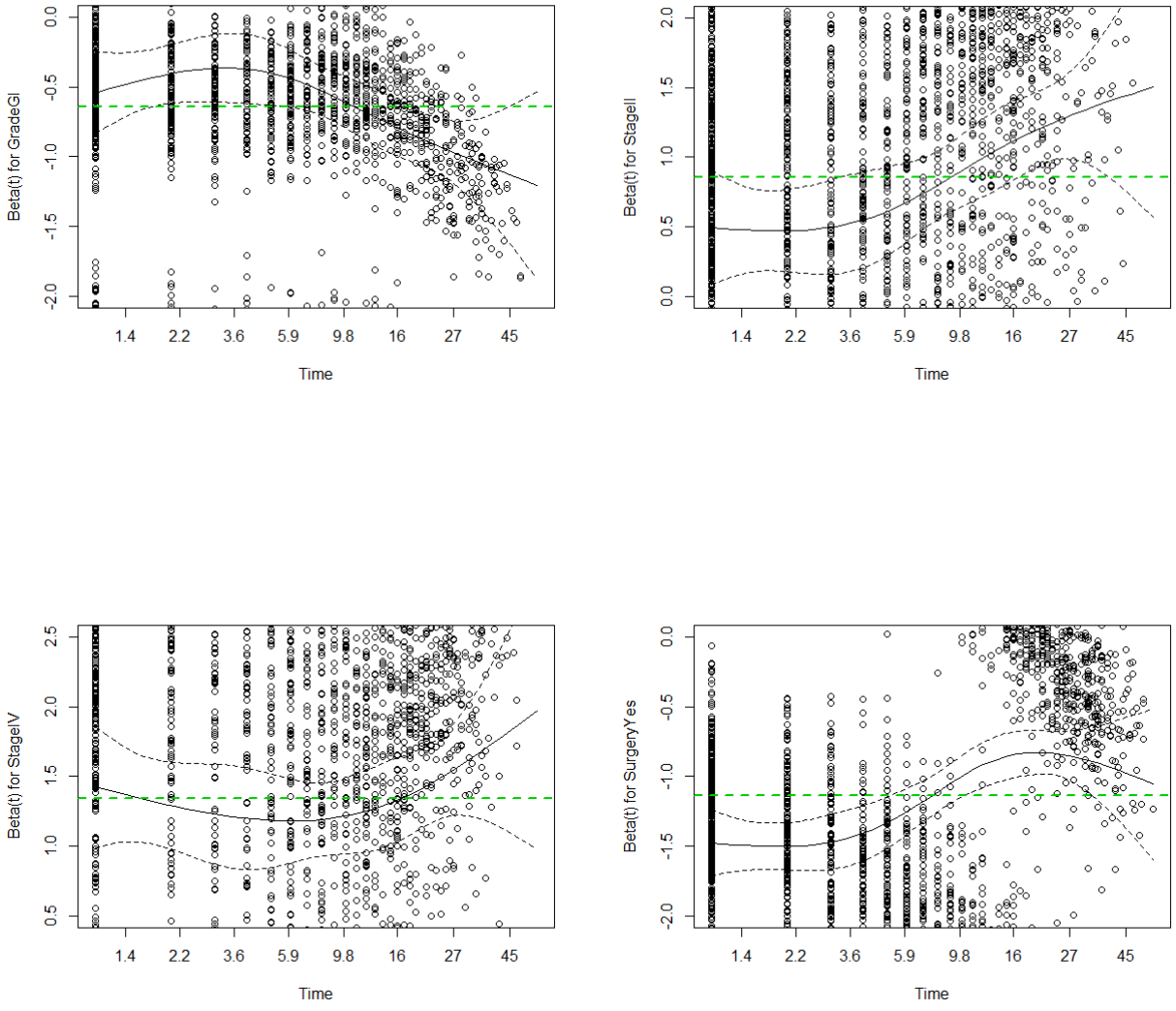

Figure 10: Coefficient Estimates for Grade I, Stage II, Stage IV and Surgery (Yes)

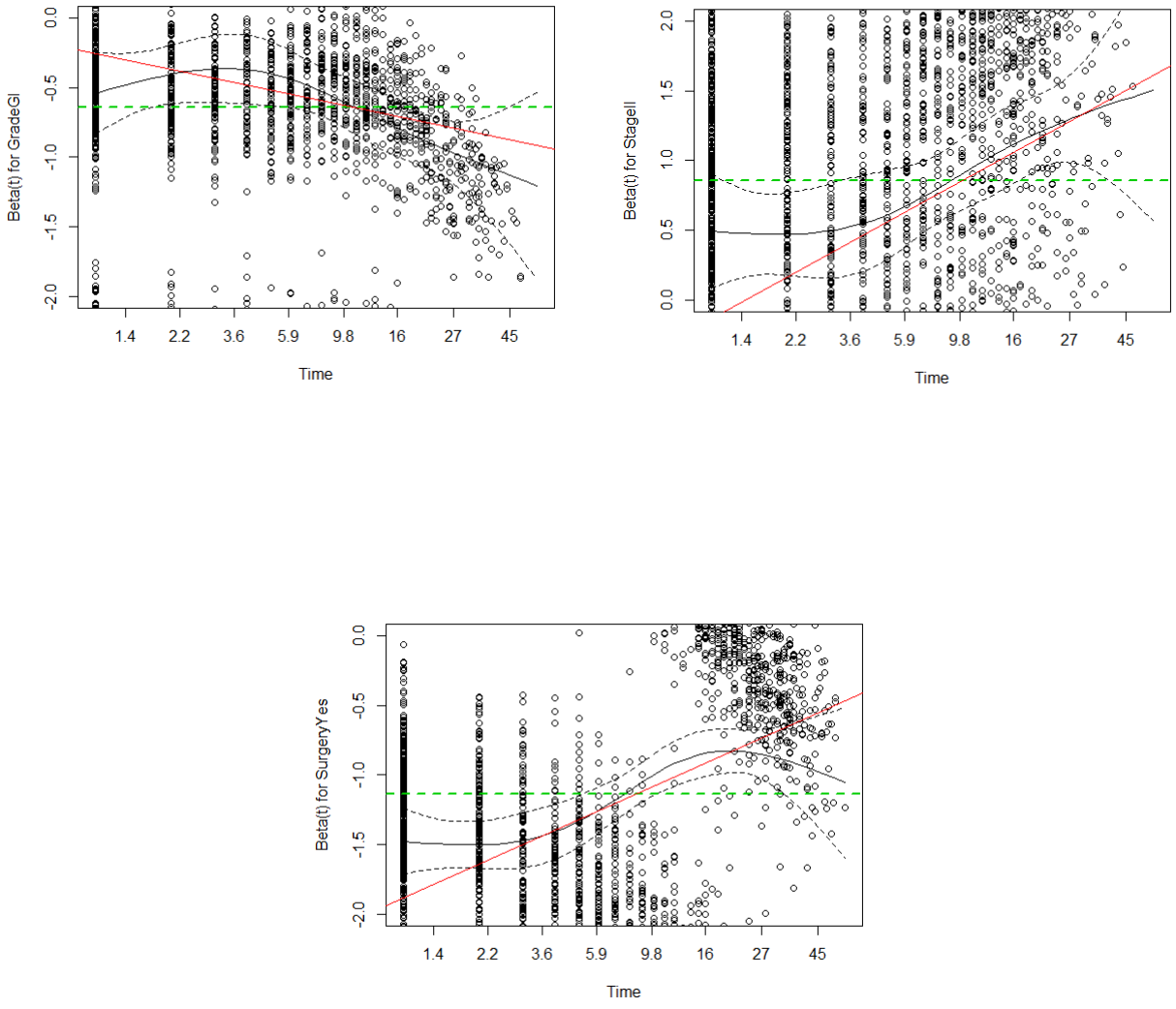

Figure 11: Coefficient Estimates as a function of *log(t)* for Grade I, Stage II, and Surgery (Yes).

A.2 Tables Used in Cox Model Fitting Procedure

Table 5: Univariate Model Fit Results

	coef	Pr(>|z|)
Age at Diagnosis	0.021987101	2.564620e-44
Race	-0.035984350	3.983308e-01
Sex	0.070894583	4.086868e-02
Surgery	-1.449989761	0.000000e+00
Tumors	-0.872698879	1.546231e-11
YoD	-0.102793375	3.136273e-03
Grade_GI	-0.426068773	9.589535e-04
tt(Grade_GI)	-0.183120787	5.279980e-04
Grade_GIII	0.834444676	1.629381e-28
tt(Grade_GIII)	-0.202511062	1.736699e-09
SchemaPancreasBodyTail	-0.307715005	2.726529e-08
SchemaPancreasHead	-0.247070101	2.798777e-07
Stage_II	-0.194105032	1.107152e-01
tt(Stage_II)	0.536210044	3.264748e-42
Stage_III	0.882173824	1.191054e-08
tt(Stage_III)	0.367557605	1.179415e-09
Stage_IV	1.937341350	6.061810e-117
MSMarried	-0.178206280	1.158599e-03
MSSingle	-0.053031749	4.355335e-01
MSWidowed	0.242258778	3.664171e-04
MoDAugust	-0.040812993	6.211971e-01
MoDDecember	-0.081921536	3.387130e-01

Table 6: Model before First Backwards Selection Step

	p-value
YoD2012	1.366006e-01
SchemaPancreasBodyTail	2.511054e-01
SchemaPancreasHead	7.213907e-02
Tumors	4.411121e-08
AaD	4.671180e-29
MSMarried	3.734823e-05
MSSingle	5.011497e-01
MSWidowed	5.170091e-01
SexMale	4.394928e-02
Grade_GI	5.808708e-02
tt(Grade_GI)	2.085131e-03
Grade_GIII	1.027401e-12
tt(Grade_GIII)	2.926377e-04
Stage_IV	2.180605e-38
Stage_III	1.688575e-01
tt(Stage_III)	1.415378e-07
Stage_II	7.725325e-02
tt(Stage_II)	8.732986e-09
Surgery_Yes	3.305676e-56
tt(Surgery_Yes)	7.542086e-12

Table 7: Model before Second Backwards Selection Step

	p-value
SchemaPancreasBodyTail	2.340111e-01
SchemaPancreasHead	7.711155e-02
Tumors	4.123365e-08
AaD	4.099486e-29
MSMarried	4.598821e-05
MSSingle	5.009288e-01
MSWidowed	5.439066e-01
SexMale	4.200174e-02
Grade_GI	6.110292e-02
tt(Grade_GI)	2.026449e-03
Grade_GIII	1.031005e-12
tt(Grade_GIII)	3.025837e-04
Stage_IV	1.190454e-38
Stage_III	1.623811e-01
tt(Stage_III)	1.670585e-07
Stage_II	7.510949e-02
tt(Stage_II)	9.012118e-09
Surgery_Yes	3.075510e-56
tt(Surgery_Yes)	6.453587e-12

Table 8: Model before Final Backwards Selection Step

	p-value
Tumors	4.156097e-08
AaD	3.781276e-29
MSMarried	5.088845e-05
MSSingle	6.118165e-01
MSWidowed	5.986194e-01
SexMale	5.771243e-02
Grade_GI	6.951929e-02
tt(Grade_GI)	1.389523e-03
Grade_GIII	6.025238e-13
tt(Grade_GIII)	2.893476e-04
Stage_IV	7.222723e-38
Stage_III	1.458300e-01
tt(Stage_III)	1.535865e-07
Stage_II	5.003548e-02
tt(Stage_II)	6.003199e-09
Surgery_Yes	1.203964e-55
tt(Surgery_Yes)	9.241899e-12

Table 9: Model before First Forward Selection Step

	p-value
Tumors	3.110219e-08
AaD	6.187909e-29
MSMarried	1.107868e-04
MSSingle	6.688272e-01
MSWidowed	4.317647e-01
Grade_GI	6.718251e-02
tt(Grade_GI)	1.481076e-03
Grade_GIII	5.227177e-13
tt(Grade_GIII)	2.957390e-04
Stage_IV	2.527836e-38
Stage_III	1.375902e-01
tt(Stage_III)	1.392850e-07
Stage_II	4.631469e-02
tt(Stage_II)	5.898010e-09
Surgery_Yes	7.351247e-56
tt(Surgery_Yes)	7.467198e-12

Table 10: Model before Second Forward Selection Step

	p-value
SchemaPancreasBodyTail	2.682316e-01
SchemaPancreasHead	7.435717e-02
Tumors	2.997387e-08
AaD	7.049596e-29
MSMarried	1.078139e-04
MSSingle	5.591168e-01
MSWidowed	3.779396e-01
Grade_GI	5.890844e-02
tt(Grade_GI)	2.154460e-03
Grade_GIII	8.772389e-13
tt(Grade_GIII)	3.066494e-04
Stage_IV	3.936750e-39
Stage_III	1.523182e-01
tt(Stage_III)	1.494131e-07
Stage_II	6.902713e-02
tt(Stage_II)	8.691537e-09
Surgery_Yes	1.825038e-56
tt(Surgery_Yes)	5.165562e-12

Table 11: Model before Final Forward Selection Step

	p-value
YoD2012	1.247643e-01
Tumors	3.327643e-08
AaD	6.716952e-29
MSMarried	8.902534e-05
MSSingle	6.673366e-01
MSWidowed	4.063347e-01
Grade_GI	6.411718e-02
tt(Grade_GI)	1.518442e-03
Grade_GIII	5.262414e-13
tt(Grade_GIII)	2.847249e-04
Stage_IV	4.754176e-38
Stage_III	1.441001e-01
tt(Stage_III)	1.170891e-07
Stage_II	4.760751e-02
tt(Stage_II)	5.829198e-09
Surgery_Yes	8.153514e-56
tt(Surgery_Yes)	8.791322e-12

Table 12: Final Cox Model Output Summary

	exp(coef)	exp(-coef)	lower .95	upper .95
Tumors	0.4885	2.0472	0.3790	0.6295
AaD	1.0193	0.9811	1.0159	1.0227
MSMarried	0.8084	1.2370	0.7258	0.9005
MSSingle	0.9712	1.0296	0.8495	1.1103
MSWidowed	0.9457	1.0574	0.8229	1.0869
Grade_GI	0.7869	1.2708	0.6088	1.0171
tt(Grade_GI)	0.8426	1.1868	0.7581	0.9364
Grade_GIII	1.7270	0.5790	1.4889	2.0032
tt(Grade_GIII)	0.8853	1.1296	0.8287	0.9457
Stage_IV	3.2233	0.3102	2.6999	3.8483
Stage_III	1.2683	0.7884	0.9268	1.7358
tt(Stage_III)	1.3816	0.7238	1.2250	1.5582
Stage_II	1.3091	0.7639	1.0044	1.7061
tt(Stage_II)	1.3040	0.7669	1.1925	1.4259
Surgery_Yes	0.1527	6.5508	0.1208	0.1929
tt(Surgery_Yes)	1.4187	0.7049	1.2836	1.5680

Concordance= 0.765 (se = 0.004)
Rsquare= 0.465 (max possible= 1)
Likelihood ratio test= 2706 on 16 df, p=<2e-16
Wald test = 2142 on 16 df, p=<2e-16
Score (logrank) test = 2744 on 16 df, p=<2e-16

A.3 Tables and Figures for Parameter Estimation in RSF Model

Table 13: First Pass (top) and Second Pass at RSF Tuning

model.id	ntree	mtry	nodesize	avg.error	variance.err
58	250	7	150	0.2473236	0.0001744189
23	150	7	150	0.2473594	0.0001696872
24	150	7	200	0.2474164	0.0001804059
53	250	6	150	0.2474843	0.0001782948
18	150	6	150	0.2475778	0.0001721993
13	150	5	150	0.2476480	0.0001712030
54	250	6	200	0.2476482	0.0001803853
12	150	5	125	0.2477602	0.0001671897
14	150	5	200	0.2477769	0.0001574709
28	150	8	150	0.2477895	0.0001658426

model.id	ntree	mtry	nodesize	avg.error	variance.err
31	250	7	165	0.2472863	0.0001739533
30	250	7	160	0.2473857	0.0001788791
14	150	7	170	0.2474811	0.0001710807
29	250	7	155	0.2475602	0.0001808233
12	150	7	160	0.2475646	0.0001784989
32	250	7	170	0.2475667	0.0001629898
17	150	7	185	0.2475764	0.0001715206
10	150	7	150	0.2475808	0.0001624479
4	150	6	165	0.2476144	0.0001860690
9	150	6	190	0.2476440	0.0001675902

```
Sample size of test (predict) data: 1296
     Number of deaths in test data: 973
               Number of grow trees: 250
Average no. of grow terminal nodes: 18.292
       Total no. of grow variables: 11
     Resampling used to grow trees: swr
  Resample size used to grow trees: 3024
                           Analysis: RSF
                             Family: surv
                Test set error rate: 24.8%
```

```
Sample size of test (predict) data: 1296
     Number of deaths in test data: 973
               Number of grow trees: 10000
Average no. of grow terminal nodes: 18.4159
       Total no. of grow variables: 11
     Resampling used to grow trees: swr
  Resample size used to grow trees: 3024
                           Analysis: RSF
                             Family: surv
                Test set error rate: 24.81%
```

Figure 12: RSF Final Model Run on Test Data (top), and with 10,000 Trees

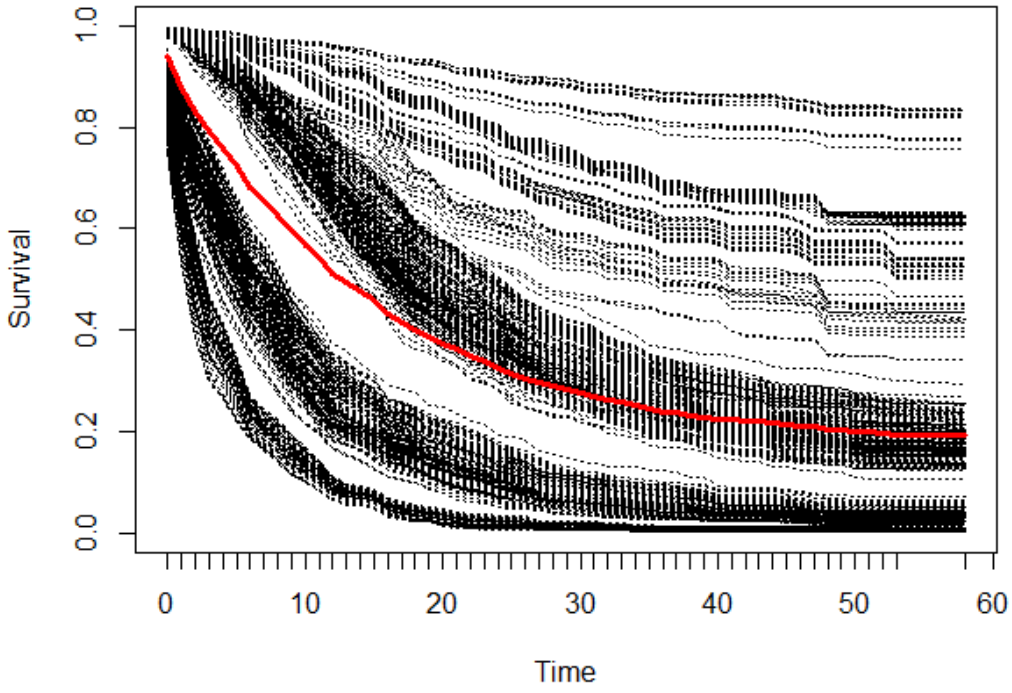

Figure 13: Estimated Survival from RSF Model for each Terminal Node (black) and Mean Survival (red)

APPENDIX B

R CODE

B.1 Code to Produce Cox Model Analysis

```
library(survival)
library(dplyr)
library(fastDummies)



train.test<-readRDS("train_test_data.rdata")


so.full.sex<-survfit(Surv(time,CI) ~ Sex, data=train.test)
plot(so.full.sex,main="KM Estimate Stratified by Sex",xlab="Months",ylab="Proportion
Surviving",col=1:2)


#Create Y, the vector of survival times and censoring indicator


Y <- Surv(train.test$time,train.test$CI)
Y.1 <- Surv(train.test$time,train.test$CI==1)
#KM Intercept Only Model


km.int <- survfit(Y~1)


#KM log-log plots to test PH assumption


kmfit.race<-survfit(Y~train.test$Race)
kmfit.race$time[kmfit.race$time==0]<-1
ggsurvplot(kmfit.race,data=train.test,fun="cloglog",title="race",linetype = "strata")


kmfit.grade<-survfit(Y~train.test$Grade)
```

```
kmfit.grade$time[kmfit.grade$time==0]<-1
ggsurvplot(kmfit.grade,data=train.test,fun="cloglog",title="Grade",linetype = "strata")

kmfit.stage<-survfit(Y~train.test$Stage)
kmfit.stage$time[kmfit.stage$time==0]<-1
ggsurvplot(kmfit.stage,data=train.test,fun="cloglog",title="stage",linetype = "strata")

kmfit.surg<-survfit(Y~train.test$Surgery)
kmfit.surg$time[kmfit.surg$time==0]<-1
ggsurvplot(kmfit.surg,data=train.test,fun="cloglog",title="surgery",linetype = "strata")

kmfit.ms<-survfit(Y~train.test$MS)
kmfit.ms$time[kmfit.ms$time==0]<-1
ggsurvplot(kmfit.ms,data=train.test,fun="cloglog",title="Marital Status",linetype = "strata")

kmfit.sex<-survfit(Y~train.test$Sex)
kmfit.sex$time[kmfit.sex$time==0]<-1
ggsurvplot(kmfit.sex,data=train.test,fun="cloglog",title="Sex",linetype = "strata")

kmfit.schema<-survfit(Y~train.test$Schema)
kmfit.schema$time[kmfit.schema$time==0]<-1
ggsurvplot(kmfit.schema,data=train.test,fun="cloglog",title="schema",linetype = "strata")

kmfit.YoD<-survfit(Y~train.test$YoD)
kmfit.YoD$time[kmfit.YoD$time==0]<-1
ggsurvplot(kmfit.YoD,data=train.test,fun="cloglog",title="YoD",linetype = "strata")
```

```
kmfit.mod<-survfit(Y~train.test$MoD)

kmfit.mod$time[kmfit.mod$time==0]<-1

ggsurvplot(kmfit.mod,data=train.test,fun="cloglog",title="Month of Diagnosis",linetype =
"strata")


kmfit.tumors<-survfit(Y~train.test$Tumors)

kmfit.tumors$time[kmfit.tumors$time==0]<-1

ggsurvplot(kmfit.tumors,data=train.test,fun="cloglog",title="tumors",linetype = "strata")




#KM log-log for Age at Diagnosis requires partitioning integer variable and turning into factor


min(train.test$AaD)

max(train.test$AaD)


AgeGroups<-cut(train.test$AaD,breaks=c(7.9,50,70,100),labels = c("Young","Middle-
Age","Old"))

train.test$AgeGroups<-AgeGroups

kmfit.age<-survfit(Y~train.test$AgeGroups)

kmfit.age$time[kmfit.age$time==0]<-1

ggsurvplot(kmfit.age,data=train.test,fun="cloglog",title="Age at Diagnosis",linetype = "strata")

train.test$AgeGroups <- NULL
```

```
mod2<-coxph(Surv(time,CI==1)~.,data=train.test)
cox.zph.test<-cox.zph(mod2,transform = rank)
ph.test<-cox.zph.test$table
ph.test.df<-as.data.frame(ph.test)[c(3:9,10,13,14,15),]
colnames(ph.test.df)<-c("Rho","Chi-Squared","p-value")
formattable(ph.test.df)

#Cox

  #Convert data to "counting process" style data,
  #take full data and add id, set all t=0 to t=1 for software

tbl.1<-tbl_df(train.test)

cox.data<-tbl.1 %>% mutate(time=replace(time,time==0,1)) %>% as.data.frame()

cut.points<-unique(cox.data$time[cox.data$CI == 1])

cox.data<-cox.data[order(cox.data$time),]

cox.data$id<-seq(4320)

cp.data<-survSplit(data=cox.data,cut=cut.points,end="time",start="time0",event = "CI")

train.test.cox<-tbl.1 %>% mutate(time=replace(time,time==0,1)) %>% as.data.frame()

nrow(cp.data)
```

```
nrow(train.test.cox)

#The counting process data has 81000+ rows while the original data with modification has 4320


#Visuals for Beta over time

cfit<-coxph(Surv(time,CI)~.,data=train.test.cox)
zp<-cox.zph(cfit,transform = function(time) log(time))
plot(zp[5],ylim=c(-1,1))
abline(h=cfit$coef[5],col=3,lwd=2,lty=2)
plot(zp[6],ylim=c(-2,0))
abline(h=cfit$coef[6],col=3,lwd=2,lty=2)

plot(zp[7],ylim=c(0,2))
abline(h=cfit$coef[7],col=3,lwd=2,lty=2)
plot(zp[8],ylim=c(0,2))
abline(h=cfit$coef[8],col=3,lwd=2,lty=2)
plot(zp[9],ylim=c(.5,2.5))
abline(h=cfit$coef[9],col=3,lwd=2,lty=2)

plot(zp[10],ylim=c(-2,0))
abline(h=cfit$coef[10],col=3,lwd=2,lty=2)

#Create dummy variables

dum2<-dummy_cols(train.test.cox[,4:5])
```

```
dum2<-dum2[,-c(1,2,3,9)]
train.test.cox2<-cbind(train.test.cox,dum2)

cfit3<-coxph(Surv(time,CI) ~
        Race + YoD + Schema + Tumors + AaD + MS + MoD + Sex +
        Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III +
tt(Stage_III) + Stage_II + tt(Stage_II) +
        Surgery,data=train.test.cox2,
      tt = function(x,t,...) x * log(t))

dum3<-dummy_cols(train.test.cox[,4:6])
dum3<-dum3[,-c(1,2,3,4,10,12)]
train.test.cox3<-cbind(train.test.cox,dum3)

cfit4<-coxph(Surv(time,CI) ~
        YoD + Schema + Tumors + AaD + MS + Sex +
        Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III +
tt(Stage_III) + Stage_II + tt(Stage_II) +
        Surgery,data=train.test.cox2,
      tt = function(x,t,...) x * log(t))

cfit5<-coxph(Surv(time,CI) ~
        Race + YoD + Schema + Tumors + AaD + MS + MoD + Sex +
        Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III +
tt(Stage_III) + Stage_II + tt(Stage_II) +
        Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,
      tt = function(x,t,...) x * log(t))
```

```
plot(zp[5],ylim=c(-1,1))
abline(h=cfit$coef[5],col=3,lwd=2,lty=2)
abline(coef(cfit3)[24:25],col=2)

plot(zp[6],ylim=c(-2,0))
abline(h=cfit$coef[6],col=3,lwd=2,lty=2)
abline(coef(cfit3)[22:23],col=2)

plot(zp[7],ylim=c(0,2))
abline(h=cfit$coef[7],col=3,lwd=2,lty=2)
abline(coef(cfit3)[29:30],col=2)

plot(zp[8],ylim=c(0,2))
abline(h=cfit$coef[8],col=3,lwd=2,lty=2)
abline(coef(cfit3)[27:28],col=2)

plot(zp[10],ylim=c(-2,0))
abline(h=cfit$coef[10],col=3,lwd=2,lty=2)
abline(coef(cfit5)[31:32],col=2)

#Fit univariate model for each variable

Y<-Surv(train.test.cox3$time,train.test.cox3$CI)

cox.uv.tumors<-coxph(Y~Tumors,data=train.test.cox3)
summary(cox.uv.tumors)
```

```
cox.uv.schema<-coxph(Y~Schema,data=train.test.cox3)
summary(cox.uv.schema)

cox.uv.mod<-coxph(Y~MoD,data=train.test.cox3)
summary(cox.uv.mod)

cox.uv.yod<-coxph(Y~YoD,data=train.test.cox3)
summary(cox.uv.yod)

cox.uv.sex<-coxph(Y~Sex,data=train.test.cox3)
summary(cox.uv.sex)

cox.uv.grade<-coxph(Y ~ Grade_GI + tt(Grade_GI) + Grade_GIII
        + tt(Grade_GIII),data=train.test.cox3,tt = function(x,t,...) x * log(t))
summary(cox.uv.grade)

cox.uv.ms<-coxph(Y~MS,data=train.test.cox3)
summary(cox.uv.ms)

cox.uv.race<-coxph(Y~Race,data=train.test.cox3)
summary(cox.uv.race)

cox.uv.stage<-coxph(Y ~ Stage_II + tt(Stage_II) + Stage_III
        + tt(Stage_III) + Stage_IV,data=train.test.cox3,tt = function(x,t,...) x * log(t))
summary(cox.uv.stage)

cox.uv.surgery<-coxph(Y ~ Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,tt =
function(x,t,...) x * log(t))
```

```
summary(cox.uv.surgery)

cox.uv.aad<-coxph(Y~AaD,data=train.test.cox3)
summary(cox.uv.aad)

#Extract p-values for each univariate model, put these into dataframe for visual

p.aad<-coef(summary(cox.uv.aad))[,c(1,5)]
p.sex<-coef(summary(cox.uv.sex))[,c(1,5)]
p.race<-coef(summary(cox.uv.race))[,c(1,5)]
p.stage<-coef(summary(cox.uv.stage))[,c(1,5)]
p.grade<-coef(summary(cox.uv.grade))[,c(1,5)]
p.surg<-coef(summary(cox.uv.surg))[,c(1,5)]
p.ms<-coef(summary(cox.uv.ms))[,c(1,5)]
p.mod<-coef(summary(cox.uv.mod))[,c(1,5)]
p.yod<-coef(summary(cox.uv.yod))[,c(1,5)]
p.schema<-coef(summary(cox.uv.schema))[,c(1,5)]
p.tumors<-coef(summary(cox.uv.tumors))[,c(1,5)]

p.values.uv<-
rbind(p.aad,p.race,p.sex,p.surg,p.tumors,p.yod,p.grade,p.schema,p.stage,p.ms,p.mod)
p.values.uv<-as.data.frame(p.values.uv)
rownames(p.values.uv)[1:6]=c("Age at Diagnosis","Race","Sex","Surgery","Tumors","YoD")

uv.p<-as.data.frame(c(p.aad,p.sex,p.race,p.stage,p.grade,p.surg,p.ms))
formattable(p.values.uv)
```

```
#With univariate models fit, remove noise variables MoD and Race and fit full model containing all other covariates

#No automatic stepwise procedure in R works with both categorical predictors and time-transformed variables

cfit6<-coxph(Surv(time,CI) ~

        YoD + Schema + Tumors + AaD + MS + Sex +

        Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III + tt(Stage_III) + Stage_II + tt(Stage_II) +

        Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,

      tt = function(x,t,...) x * log(t))

summary(cfit6)

cfit6.p<-as.data.frame(coef(summary(cfit6))[,5])

colnames(cfit6.p)="p-value"


#Remove YoD for high p-value

cfit7<-coxph(Surv(time,CI) ~

        Schema + Tumors + AaD + MS + Sex +

        Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III + tt(Stage_III) + Stage_II + tt(Stage_II) +

        Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,

      tt = function(x,t,...) x * log(t))

cfit7.p<-as.data.frame(coef(summary(cfit7))[,5])

colnames(cfit7.p)="p-value"

#Remove Schema for high p-value

cfit8<-coxph(Surv(time,CI) ~

        Tumors + AaD + MS + Sex +

        Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III + tt(Stage_III) + Stage_II + tt(Stage_II) +

        Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,
```

```
        tt = function(x,t,...) x * log(t))
cfit8.p<-as.data.frame(coef(summary(cfit8))[,5])
colnames(cfit8.p)="p-value"
#Remove Sex for high p-value
cfit9<-coxph(Surv(time,CI) ~
             Tumors + AaD + MS +
             Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III +
tt(Stage_III) + Stage_II + tt(Stage_II) +
             Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,
           tt = function(x,t,...) x * log(t))
cfit9.p<-as.data.frame(coef(summary(cfit9))[,c(1,2,5)])
colnames(cfit9.p)=c("Beta","exp(Beta)","p-value")
#Forward selection - add Schema (Race and Month had such high pvalues we do not consider
them)
cfit10<-coxph(Surv(time,CI) ~
             Schema + Tumors + AaD + MS +
             Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III +
tt(Stage_III) + Stage_II + tt(Stage_II) +
             Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,
           tt = function(x,t,...) x * log(t))
cfit10.p<-as.data.frame(coef(summary(cfit10))[,5])
colnames(cfit10.p)="p-value"
#Schema still not significant, return to cfit9 as best model and try adding YoD
cfit11<-coxph(Surv(time,CI) ~
             YoD + Tumors + AaD + MS +
             Grade_GI + tt(Grade_GI) + Grade_GIII + tt(Grade_GIII) + Stage_IV + Stage_III +
tt(Stage_III) + Stage_II + tt(Stage_II) +
             Surgery_Yes + tt(Surgery_Yes),data=train.test.cox3,
           tt = function(x,t,...) x * log(t))
```

```
cfit11.p<-as.data.frame(coef(summary(cfit11))[,5])
colnames(cfit11.p)="p-value"

#YoD not significant, return to cfit9 as best model
```

B.2 Code to Produce Random Survival Forest Analysis

```
library(survival)
library(randomForestSRC)
library(dplyr)

fulldata <- read.csv("fdfmf.csv",TRUE,",")
class(fulldata)
colnames(fulldata) <-
c("Race","YoD","Schema","Grade","Stage","AJCCT","AJCCN","AJCCM","Surgery","Survival
Months","Status","Tumors","AaD","MS","MoD","Histology","Sex")

head(fulldata)
str(fulldata)
summary(fulldata)

#Data cleaning and pre-processing

#Drop observations with unknown marital status and "Unmarried or Domestic Partner" due to
ambiguity and
```

```
# very low occurence, then remove unused factor levels; group Divorced and Separated into Divorced


fulldata <- fulldata[fulldata$MS!="Unknown", ]

fulldata$MS <- factor(fulldata$MS)

fulldata <- fulldata[fulldata$MS!="Unmarried or Domestic Partner", ]

fulldata$MS <- factor(fulldata$MS)

fulldata <- fulldata[fulldata$Stage !="IINOS", ]

fulldata$Stage <- factor(fulldata$Stage)

levels(fulldata$MS) <- c("Divorced", "Married", "Divorced", "Single", "Widowed")

levels(fulldata$Race)<-c("Other","Other","White")



#Drop Factor observations with factor level attributes that are very infrequent or unknown


fulldata <- fulldata[fulldata$Surgery!="Recommended, unknown if performed", ]

fulldata$Surgery <- factor(fulldata$Surgery)

fulldata <- fulldata[fulldata$Surgery!="Not performed, patient died prior to recommended surgery", ]

fulldata$Surgery <- factor(fulldata$Surgery)



fulldata <- fulldata[fulldata$Grade!="Undifferentiated; anaplastic; Grade IV", ]

fulldata$Grade <- factor(fulldata$Grade)

levels(fulldata$Grade)<-c("GII","GIII","GI")

#Group all surgery levels into "Yes" and "No" indicating surgery was performed or wasn't performed
```

```
Surgery <- as.numeric(fulldata$Surgery)
Surgery <- replace(Surgery, Surgery==2, 1)
Surgery <- replace(Surgery, Surgery==3, 1)
Surgery <- replace(Surgery, Surgery==4, 1)
Surgery <- replace(Surgery, Surgery==5, 2)
Surgery <- as.factor(Surgery)

fulldata$Surgery=Surgery
Surgery<-NULL
levels(fulldata$Surgery)<-c("No","Yes")

#Group Stage variables into numbered stages, removing A and B designations

levels(fulldata$Stage)<-c("I","I","II","II","III","IV")

#Year of Diagnosis variable to factor

fulldata$YoD <- factor(fulldata$YoD)

#RSF requires censoring vector to have values of 0 for Alive and 1 for Dead
#Create censoring indicator variable

CI <- as.numeric(fulldata$Status)
CI <- replace(CI, CI==1, 0)
CI <- replace(CI, CI==2, 1)
CI <- as.integer(CI)
fulldata$CI=CI
```

```
remove(CI)

fulldata <- subset(fulldata, select = -c(AJCCM, AJCCT, AJCCN, Status, Histology))

colnames(fulldata)[7] <- "time"

d2013 <- fulldata[fulldata$YoD %in% c("2013"),]
train.test <- fulldata[fulldata$YoD %in% c("2011","2012"),]
train.test$YoD<-factor(train.test$YoD)

set.seed(99)
train.obs <- sample(seq(nrow(train.test)),0.70 * nrow(train.test))
train <- train.test[train.obs, ]
test <- train.test[-train.obs, ]
saveRDS(train.test,file="train_test_data.rdata")

saveRDS(train,file = "training_data.rdata")
saveRDS(test,file = "test_data.rdata")

#Create validation sets for 10 fold cross validation

set.seed(999)
fold.size<-floor(0.10 * nrow(train))
train.shffl<-train[sample(nrow(train)), ]
k<-c(1:9)
CV<-list()
```

```
for (i in k){
  CV[[i]]<-train.shffl[-((((i-1)*fold.size)+1):(fold.size*i)), ]
}
CV[[10]]<-train.shffl[-((((10-1)*fold.size)+1):(nrow(train.shffl))), ]

CV.test <- list()

for (i in k){
  CV.test[[i]]<-train.shffl[((((i-1)*fold.size)+1):(fold.size*i)), ]
}
CV.test[[10]]<-train.shffl[((((10-1)*fold.size)+1):(nrow(train.shffl))), ]



#Run first pass of tuning on validation sets
#use ntree = 50,150,250
#use mtry = 2,5,8,11
#use nodesize = 1, 15, 50, 100, 500, 1000
#create objects trained on each validation set, these will be used to measure error on validation test sets

memory.limit(32000)
ntree.1 <- c(50,150,250)
mtry.1 <- c(2,5,8,11)
nodesize.1 <- c(1,15,50,100,500,1000)

rf.tune.1 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
```

```
 j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[1]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.1,file="rft1.rds")

rf.tune.2 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[2]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.2,file="rft2.rds")

rf.tune.3 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
```

```
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[3]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.3,file="rft3.rds")

rf.tune.4 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[4]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.4,file="rft4.rds")

rf.tune.5 <- lapply(seq(length(ntree.1)), function(ii){
```

```
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[5]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.5,file="rft5.rds")

rf.tune.6 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[6]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.6,file="rft6.rds")
```

```
rf.tune.7 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[7]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.7,file="rft7.rds")

rf.tune.8 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[8]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.8,file="rft8.rds")
```

```
rf.tune.9 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[9]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune.9,file="rft9.rds")

rf.tune.10 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[10]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
```

```
saveRDS(rf.tune.10,file="rft10.rds")

#Run each model for full training data so we can compare OOB error to test set error from CV

memory.limit(32000)
ntree.1 <- c(50,150,250)
mtry.1 <- c(2,5,8,11)
nodesize.1 <- c(1,15,50,100,500,1000)

rf.train.1 <- lapply(seq(length(ntree.1)), function(ii){
  i <- ntree.1[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- mtry.1[jj]
    lapply(seq(length(nodesize.1)), function(kk){
      k <- nodesize.1[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=train,
            ntree=i,mtry=j,nodesize=k)
    })
  })
})

oob.err <- lapply(seq(length(ntree.1)), function(ii){
  i <- seq(length(ntree.1))[ii]
  lapply(seq(length(mtry.1)), function(jj){
    j <- seq(length(mtry.1))[jj]
    lapply(seq(length(nodesize.1)), function(kk){
```

```
  k <- seq(length(nodesize.1))[kk]
     as.numeric(na.omit(rf.train.1[[i]][[j]][[k]]$err.rate))
   })
  })
})

oob.err<-unlist(oob.err)



#For each created forest object rft1-10, drop the corresponding validation set observations down the tree and find test set error


rf.tune.first <-
list(rf.tune.1,rf.tune.2,rf.tune.3,rf.tune.4,rf.tune.5,rf.tune.6,rf.tune.7,rf.tune.8,rf.tune.9,rf.tune.10)
n.1 <- length(ntree.1)
n.1.range <- seq(n.1)
m.1 <- length(mtry.1)
m.1.range <- seq(m.1)
no.1 <- length(nodesize.1)
no.1.range <- seq(no.1)
fold.range <- seq(10)


predict.1.err <-lapply(fold.range, function(hh){
  h <- fold.range[hh]
    lapply(n.1.range, function(ii){
    i <- n.1.range[ii]
      lapply(m.1.range, function(jj){
      j <- m.1.range[jj]
```

```
    lapply(no.1.range, function(kk){
      k <- no.1.range[kk]
      predict.temp <- predict.rfsrc(rf.tune.first[[h]][[i]][[j]][[k]],newdata = CV.test[[h]])

      as.numeric(na.omit(predict.temp$err.rate))

      })
    })
  })
})

saveRDS(predict.1.err,file="predict_1.RDS")

fold<-rep(1:10,each=72)

ntree<-rep(c(50,150,250),each=24)
ntree<-rep(ntree,10)

mtry<-rep(c(2,5,8,11),each=6)
mtry<-rep(mtry,30)

nodesize<-rep(c(1,15,50,100,500,1000),120)

prediction.error<-unlist(predict.1.err)

model.id<-rep(1:72,times=10)
predict.error.1<-data.frame(fold,model.id,ntree,mtry,nodesize,prediction.error)
```

```
saveRDS(predict.error.1,file="prediction_error_1.rdata")

#use dplyr

pred.err.1.tidy<-tbl_df(predict.error.1)

mean.pred.err.1<-pred.err.1.tidy %>% group_by(model.id) %>%
  summarise(ntree=mean(ntree),mtry=mean(mtry),nodesize=mean(nodesize),
    avg.error=mean(prediction.error),variance.err=var(prediction.error))



mean.pred.err.1$oob.err<-oob.err

tuned.1.by.CV<-mean.pred.err.1 %>% arrange(avg.error)
tuned.1.by.oob<-mean.pred.err.1 %>% arrange(oob.err)

saveRDS(tuned.1.by.CV,file = "tuned_by_CV.rdata")
saveRDS(tuned.1.by.oob,file = "tuned_by_oob.rdata")



#Run second pass of tuning on validation sets
#use ntree = 150,250
#use mtry = 3,4,5,6,7,8,11
#use nodesize = 100,125,150,200,250
#create objects trained on each validation set, these will be used to measure error on validation
test sets
```

```
memory.limit(32000)
ntree.2 <- c(150,250)
mtry.2 <- c(3,4,5,6,7,8,11)
nodesize.2 <- c(100,125,150,200,250)

rf.tune2.1 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[1]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.1,file="rft21.rds")

rf.tune2.2 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
      rfsrc(Surv(time,CI) ~ .,
```

```
   data=as.data.frame(CV[2]),
          ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.2,file="rft22.rds")

rf.tune2.3 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
      rfsrc(Surv(time,CI) ~ .,
          data=as.data.frame(CV[3]),
          ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.3,file="rft23.rds")

rf.tune2.4 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
```

```
    rfsrc(Surv(time,CI) ~ .,
          data=as.data.frame(CV[4]),
          ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.4,file="rft24.rds")

rf.tune2.5 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
      rfsrc(Surv(time,CI) ~ .,
          data=as.data.frame(CV[5]),
          ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.5,file="rft25.rds")

rf.tune2.6 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
```

```
    k <- nodesize.2[kk]
    rfsrc(Surv(time,CI) ~ .,
          data=as.data.frame(CV[6]),
          ntree=i,mtry=j,nodesize=k)
   })
  })
})
saveRDS(rf.tune2.6,file="rft26.rds")

rf.tune2.7 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[7]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.7,file="rft27.rds")

rf.tune2.8 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
```

```
  lapply(seq(length(nodesize.2)), function(kk){
     k <- nodesize.2[kk]
     rfsrc(Surv(time,CI) ~ .,
          data=as.data.frame(CV[8]),
          ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.8,file="rft28.rds")

rf.tune2.9 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
      rfsrc(Surv(time,CI) ~ .,
           data=as.data.frame(CV[9]),
           ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.9,file="rft29.rds")

rf.tune2.10 <- lapply(seq(length(ntree.2)), function(ii){
  i <- ntree.2[ii]
  lapply(seq(length(mtry.2)), function(jj){
```

```
    j <- mtry.2[jj]
    lapply(seq(length(nodesize.2)), function(kk){
      k <- nodesize.2[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[10]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
saveRDS(rf.tune2.10,file="rft210.rds")


#For each created forest object rft2.1-10, drop the corresponding validation set observations down the tree and find test set error


rf.tune.second <- list(rf.tune2.1,rf.tune2.2,rf.tune2.3,rf.tune2.4,rf.tune2.5,rf.tune2.6,rf.tune2.7,rf.tune2.8,rf.tune2.9, rf.tune2.10)


saveRDS(rf.tune.second,file = "rf_tune_2_before_predict.RDS")
rf.tune.second <- readRDS("rf_tune_2_before_predict.RDS")
n.2 <- length(ntree.2)
n.2.range <- seq(n.2)
m.2 <- length(mtry.2)
m.2.range <- seq(m.2)
no.2 <- length(nodesize.2)
no.2.range <- seq(no.2)
fold.range <- seq(10)
```

```
predict.2.err <-lapply(fold.range, function(hh){
  h <- fold.range[hh]
  lapply(n.2.range, function(ii){
    i <- n.2.range[ii]
    lapply(m.2.range, function(jj){
      j <- m.2.range[jj]
      lapply(no.2.range, function(kk){
        k <- no.2.range[kk]
        predict.temp <- predict.rfsrc(rf.tune.second[[h]][[i]][[j]][[k]],newdata = CV.test[[h]])

        as.numeric(na.omit(predict.temp$err.rate))

      })
    })
  })
})

saveRDS(predict.2.err,file="predict_2.RDS")


fold<-rep(1:10,each=70)

ntree<-rep(c(150,250),each=35)
ntree<-rep(ntree,10)

mtry<-rep(c(3,4,5,6,7,8,11),each=5)
mtry<-rep(mtry,20)
```

```
nodesize<-rep(c(100,125,150,200,250),140)

prediction.error2<-unlist(predict.2.err)

model.id<-rep(1:70,times=10)
predict.error.2<-data.frame(fold,model.id,ntree,mtry,nodesize,prediction.error2)

saveRDS(predict.error.2,file="prediction_error_2.rdata")
predict.error.2<-readRDS("prediction_error_2.rdata")
pred.err.2.tidy<-tbl_df(predict.error.2)

mean.pred.err.2<-pred.err.2.tidy %>% group_by(model.id) %>%
  summarise(ntree=mean(ntree),mtry=mean(mtry),nodesize=mean(nodesize),
            avg.error=mean(prediction.error2),variance.err=var(prediction.error2))
tuned.2<-mean.pred.err.2 %>% arrange(avg.error)
saveRDS(tuned.2,file="tuned_2.rdata")



memory.limit(32000)
ntree.3 <- c(150,250)
mtry.3 <- c(6,7)
nodesize.3 <- c(150,155,160,165,170,175,180,185,190)

rf.tune3.1 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
```

```
 lapply(seq(length(mtry.3)), function(jj){
   j <- mtry.3[jj]
   lapply(seq(length(nodesize.3)), function(kk){
     k <- nodesize.3[kk]
     rfsrc(Surv(time,CI) ~ .,
           data=as.data.frame(CV[1]),
           ntree=i,mtry=j,nodesize=k)
   })
  })
})


rf.tune3.2 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[2]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})


rf.tune3.3 <- lapply(seq(length(ntree.3)), function(ii){
```

```
 i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[3]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})


rf.tune3.4 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[4]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
```

```
rf.tune3.5 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[5]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})


rf.tune3.6 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[6]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
```

```
rf.tune3.7 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[7]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})


rf.tune3.8 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[8]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})
```

```
rf.tune3.9 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[9]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
})


rf.tune3.10 <- lapply(seq(length(ntree.3)), function(ii){
  i <- ntree.3[ii]
  lapply(seq(length(mtry.3)), function(jj){
    j <- mtry.3[jj]
    lapply(seq(length(nodesize.3)), function(kk){
      k <- nodesize.3[kk]
      rfsrc(Surv(time,CI) ~ .,
            data=as.data.frame(CV[10]),
            ntree=i,mtry=j,nodesize=k)
    })
  })
```

```
})

#For each created forest object rft3.1-10, drop the corresponding validation set observations down the tree and find test set error

rf.tune.third <- list(rf.tune3.1,rf.tune3.2,rf.tune3.3,rf.tune3.4,rf.tune3.5,rf.tune3.6,rf.tune3.7,rf.tune3.8,rf.tune3.9, rf.tune3.10)

saveRDS(rf.tune.third,file = "rf_tune_3_before_predict.RDS")

rf.tune.third <- readRDS("rf_tune_3_before_predict.RDS")

n.3 <- length(ntree.3)

n.3.range <- seq(n.3)

m.3 <- length(mtry.3)

m.3.range <- seq(m.3)

no.3 <- length(nodesize.3)

no.3.range <- seq(no.3)

fold.range <- seq(10)

predict.3.err <-lapply(fold.range, function(hh){
  h <- fold.range[hh]
  lapply(n.3.range, function(ii){
    i <- n.3.range[ii]
    lapply(m.3.range, function(jj){
      j <- m.3.range[jj]
      lapply(no.3.range, function(kk){
        k <- no.3.range[kk]
        predict.temp <- predict.rfsrc(rf.tune.third[[h]][[i]][[j]][[k]],newdata = CV.test[[h]])
```

```
  as.numeric(na.omit(predict.temp$err.rate))

    })
   })
 })
})

saveRDS(predict.3.err,file="predict_3.RDS")


fold<-rep(1:10,each=36)

ntree<-rep(c(150,250),each=18)
ntree<-rep(ntree,10)

mtry<-rep(c(6,7),each=9)
mtry<-rep(mtry,20)

nodesize<-rep(c(150,155,160,165,170,175,180,185,190),40)

prediction.error3<-unlist(predict.3.err)

model.id<-rep(1:36,times=10)
predict.error.3<-data.frame(fold,model.id,ntree,mtry,nodesize,prediction.error3)

saveRDS(predict.error.3,file="prediction_error_3.rdata")
```

```
pred.err.3.tidy<-tbl_df(predict.error.3)

mean.pred.err.3<-pred.err.3.tidy %>% group_by(model.id) %>%
  summarise(ntree=mean(ntree),mtry=mean(mtry),nodesize=mean(nodesize),
            avg.error=mean(prediction.error3),variance.err=var(prediction.error3))
tuned.3<-mean.pred.err.3 %>% arrange(avg.error)
saveRDS(tuned.3,file="tuned_3.rdata")

#Pull out best 10 models for each tuning pass

tunedoob<-readRDS("tuned_by_oob.rdata")
tunedcv<-readRDS("tuned_by_cv.rdata")
tuned2<-readRDS("tuned_2.rdata")
tuned3<-readRDS("tuned_3.rdata")
formattable(tunedoob[1:10,])
formattable(tunedcv[1:10,])
formattable(tuned2[1:10,])
formattable(tuned3[1:10,])

#Chapter 4, running final model on test data, using final parameters with ntree=10,000,forest
visual
train <- readRDS("training_data.Rdata")
test <- readRDS("test_data.Rdata")
rf.final1 <- rfsrc(Surv(time,CI)~.,data=train,ntree=250,mtry=7,nodesize=165)
rf.final1.test <- predict.rfsrc(rf.final1,newdata=test)

rf.final2 <- rfsrc(Surv(time,CI)~.,data=train,ntree=10000,mtry=7,nodesize=165)
```

```
rf.final2.test <- predict.rfsrc(rf.final2,newdata=test)


plot.survival(rf.final1.test,plots.one.page = FALSE,show.plots = TRUE)
```